Introductory Java Programming
Example Code & Notes

Dr. Don Spickler

Last Updated: January 4, 2017

Contents

List of Figures

About this Document

This document is designed to be a supplement for a first course in computer programming aimed at the first year undergraduate mathematics and computer science major. It is not intended to be a textbook but to accompany one. Although these materials were designed for mathematics and computer science majors, the material is accessible for those in a survey course in programming and for high school students.

This document is a combination of the example code I use when teaching introductory Java at Salisbury University, a few runs of the programs, and the handouts I give in that class. I usually do not use a printed textbook for the course but instead, use online and open source materials as the main texts. One very nice online text is by David J. Eck of Hobart and William Smith Colleges, he produces both an online text and a PDF version of *Introduction to Programming Using Java* which can be found at, http://math.hws.edu/javanotes/. All of the example code in this document can be found on the Introductory Java resource page of my web site at, http://facultyfp.salisbury.edu/despickler/ (go to Course Materials then Introductory Java).

Computer programming can be a fun, yet frustrating at times, learning experience. It requires a logical mind, attention to detail, the ability to reduce a complex task into smaller easy tasks, and by far, a lot of patience.

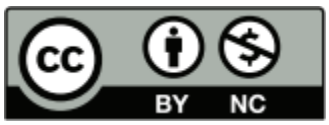
Most importantly, learn, experiment, and enjoy.

Don Spickler

2017

Edition
January 4, 2017

Publisher
Don Spickler
Department of Mathematics and Computer Science
Salisbury University
1101 Camden Ave.
Salisbury, Maryland 21801
USA



Chapter 1

Getting Started

1.1 The Programming Landscape

Programming a computer is like learning another language, say Spanish or German. When you learn another language you need to learn the words in that language and you need to learn how to put those words together to form sentences that communicate an idea to another person. The bottom line is communication. In computer programming you are doing the same thing except that you are not communicating with another person you are communicating with a computer. Instead of communicating an idea you will be communicating a task, something for the computer to do.

You will learn two important skills when programming a computer,

1. Programming Language Functionality: What the programming language has to offer in terms of what it can tell the machine to do.
2. Algorithm Development: The process of taking a complex task and breaking it down into smaller tasks, each of which is in the functionality of the programming language.

Learning the programming language functionality is fairly easy, you learn the words that you are allowed to use in the language and you learn the rules for putting the words together into a statement. Different programming languages have different sets of words you can use and different rules for putting them together. For example, Java has a Scanner object but C++ does not, so Scanner is a word that can be used in a Java program but not in a C++ program. Throughout this document and the text you are using to accompany these examples will concentrate on the functionality of the Java programming language. Java is a very large language with hundreds

of data types and thousands of functions, that is things that it can do. We will only scratching the surface of the language. Although there are many programming languages available the structure of these languages are very similar. So once you are familiar with Java you will find learning another language, like C++, to be fairly easy. It is similar to learning another speaking language. Once you know one of them, say English, learning another language like German has many similarities, and of course some differences. So you learn a translation of single words first and then you learn the similarities and differences in putting a German sentence together.

Learning algorithm development, on the other hand, is the more difficult skill but one that transcends the specifics of the programming language. The process of taking a complex task and breaking it down into very simple tasks is used when programming in any computer language. Fortunately you are not new to this, in fact, you have been doing this for quite a few years in your mathematics classes. As a simple example, say you want to add 123 and 79. Assuming that you don't get out your calculator, or your phone with the calculator app, you would write the two numbers on top of each other, lining up the columns.

$$\begin{array}{cccc} & 1 & 2 & 3 \\ + & & 7 & 9 \\ \hline \end{array}$$

You would then add the far right column of the two one digit numbers, 3 and 9, and get 12. Then you would write the 2 below that column and use the 1 as a carry to the next column.

$$\begin{array}{cccc} & & 1 & \\ & 1 & 2 & 3 \\ + & & 7 & 9 \\ \hline & & & 2 \end{array}$$

Then you would add the next column, $1 + 2 + 7$ and get 10. Then write the 0 below that column and use the 1 as a carry to the next column.

$$\begin{array}{cccc} & 1 & 1 & \\ & 1 & 2 & 3 \\ + & & 7 & 9 \\ \hline & & 0 & 2 \end{array}$$

Finally, you would add the last column $1 + 1 = 2$ and write that below the far left column.

$$\begin{array}{cccc} & 1 & 1 & \\ & 1 & 2 & 3 \\ + & & 7 & 9 \\ \hline & 2 & 0 & 2 \end{array}$$

At this point you have the answer of 202. When you write a computer program

you need to do a similar process. You will not have do addition digit by digit, Java can handle $123 + 79$ quite easily, but you will need break down what you want to ultimately accomplish into smaller tasks that the machine, and language, can handle.

Back to the communication idea. We communicate with each other either by spoken language or by written language, excluding those other forms of communication like facial expressions. With a computer, it is not so simple. A computer understands only one language, machine language. Machine language is a sequence of 1's and 0's, at least that is what we think of them as. In reality, these 1's and 0's are electronic impulses (voltage), magnetic polarity, or optical intensity levels. These 1's and 0's are not human readable, at least not without a extreme amount of patience.

Now you may be saying, "Wait a minute, you said there are lots of computer languages, Java, C++, Python, ...?" These languages are called High-Level Languages. High-Level Languages are human readable and writeable, but a computer cannot understand them as is. Once we create a program in a high-level language we need to translate it to machine language using that is called a compiler. A compiler is a program that translates your high-level program to machine language and then the machine language program is what is run on the computer. More specifically, the compiler loads in your high-level program file (Code File), something Like MyProg.cpp, and produces a machine language file (Executable File), like MyProg.exe. This exe file can then be run on your computer.

$$\text{Code File} \Longrightarrow \text{Compiler} \Longrightarrow \text{Executable File}$$

Not all languages have compilers. Some languages are interpreted, that is, they have an associated program called an interpreter that reads in the high-level program, translates and executes the commands in the program line by line. With an interpreter, the program is not converted to an executable file as it is with a compiler. So if someone else wants to run your program they must have the interpreter for that language. You can think of an interpreter as its own tiny computer that has one calculation function, running program files in that computer language.

$$\text{Code File} \Longrightarrow \text{Interpreter}$$

Languages like C, C++, Pascal, and Fortran are compiled languages. On the other hand, languages like BASIC are interpreted. You will note that Java is not in either of these lists. Java is unique in the sense that it is a hybrid of these two methods. Before we talk about Java's method let's look at some pros and cons of compilers verses interpreters.

Compiler: Code File $\Longrightarrow$ Compiler $\Longrightarrow$ Executable File

1. Faster: In general, an executable file produced by a compiler will run much faster than the same program running through an interpreter. Executable files are run directly on the computer hardware.
2. Platform Dependent: A platform is just another name for an operating system. So if you are running Windows, then you are on a Windows platform, and if you are using a Mac then you are on a Mac platform, more commonly refereed to as OSX. There are many platforms in use today but the three most common are Windows, Mac, and Linux. A program is platform independent if you can run it on any platform without changes. Compiled programs are not platform independent. If you compile a program on a Linux machine and try to run the executable on a Windows machine it will not run. You would need to have the source code and recompile it on a Windows machine to get a Windows executable program.

   In some cases an operating system might have a translator so that you can run executables from other platforms. For example, Linux has an application called Wine (WINdows Emulator) that will run some Windows executable programs.
3. Easily Distributable: If you write and compile a program on a Windows machine you can give the executable to anyone else who has a Windows machine and they can run it on their computer. There is no need for them to recompile the program. So if you have program code you do not want to share, you do not need to make the source code available.

Interpreter: Code File $\Longrightarrow$ Interpreter

1. Slower: In general, an executable file produced by a compiler will run much faster than the same program running through an interpreter. An interpreter is another program running on your computer and is acting as a middleman between the computer and the program. In addition, most interpreters translate and execute the program line by line, or statement by statement. So instead of translating the program statements once, as with a compiler, statements could be translated numerous times in the interpreter, hence slowing down the execution of the program considerably.
2. Platform independent: As long as an interpreter exists for the operating system being used.
3. Must Distributable Source Code: You must distribute source code in this case.

There are, of course, other pros and cons between the two systems but the two big ones are the first two. Compiled languages are faster, which is a good thing, but on the down-side they are platform dependent. Interpreted languages are platform

independent since the source code is run through another program, the interpreter, but on the down-side here you take a significant performance hit. In general, compiled languages are preferred since performance is usually more important to most users than is platform independence.

So how does Java work? Java is a hybrid between the two methods. There is a compiling stage that translates the source code program into a byte code file. The byte code file is not an executable file, it will not run on any native hardware. To run the program you use the Java Virtual Machine (JVM) to run the byte code file, like an interpreter.

Code File $\Longrightarrow$ Java Compiler $\Longrightarrow$ Byte Code File $\Longrightarrow$ Java Virtual Machine (JVM)

Since the program source code has been compiled into byte code, there is no further need to translate the source code, like an interpreter would do. Also since there is a JVM for all major platforms, this byte code file can be distributed to anyone and they will be able to run the program on their computer, regardless of their operating system. Hence Java is platform independent. There are some cons with Java, as with anything. Since Java has an interpreting stage, running the byte code through the JVM will not be as fas as a compiled program running directly on the hardware. Since the JVM is interpreting byte code and not program text, the JVM is much faster than an regular interpreter. Another down-side to Java, one that we will not have any problem with in an introductory programming class, is that since the JVM is a layer between your program and the computer hardware, it is more difficult to take advantage of special hardware, such as graphics card GPUs.

I am frequently asked, "What is the best programming language?" The answer is that there is no best language. If there was, we would all use it and forget about the other ones. A better answer is that a programming language is a tool, and just like any other tool, you want to pick the one that is right for the job. You do not pick up a hammer when you want to cut a board and similarly you do not pick up a saw when you want to hang a picture. If you are doing something that requires heavy number crunching you will want a compiled language like C or C++. Similarly, if you are doing something that requires a lot of direct interaction with the hardware, C is a good choice. On the other hand, if you are writing something that is required to run on many different platforms, including mobile platforms, then Java might be a good choice. If you are writing a program that is along the lines of artificial intelligence then a functional language like LISP would be more suitable.

What you are going to be using the language for is learning a programming language, probably your first programming language. For that purpose, Java is a fine choice, as would be Python. These languages are a little more forgiving than your more system level programming languages like C and C++. The best teaching language I have ever used was Pascal. Pascal is still around but rarely used. It was a

nice language for learning your first programming language. It was relatively small and its structure forced you to learn good programming habits instead of picking up bad ones. It was not object oriented, like Java, and due to the paradigm shift to object oriented programming structure Pascal fell out of favor. There were object oriented versions of Pascal but they did not seem to catch on.

As we mentioned above, a program in a high-level language is simply a text file. Something you could easily create with Notepad or GEdit. You do not need a specialized program to write Java code. If you install the Java JDK (Java Development Kit) then you have a Java compiler and the JVM so you have all of the equipment you need to create Java programs. Although you do not need more than a text editor, there are many editors out there that have many nice features that will make the job of programming a little easier for you and faster for that matter. These programs are called Integrated Development Environments (IDE), they will automatically indent your code, highlight reserved words, show the span of blocks of code, have a toolbar with options to compile and run your program, as well as features that make finding errors easier. The one I use for Java programming is Eclipse.

1.2 A Quick Introduction to the Eclipse IDE

Eclipse is an integrated development environment (IDE) for Java programming. Actually, it is capable of much more than just compiling Java programs but that is primarily what we will be using it for. Eclipse is a professional development environment used by programmers in numerous different fields from mobile app development to medical imaging to rocket guidance systems. There are pros and cons to using a professional development environment. The big pro is that you will be using the same system for learning how to program as you will when you are employed as a programmer, if you go into some area of Java development. Also, Eclipse has a ton of features that make programming easier. The big con is that since this is a professional system there are many options to the system and settings that may not make any sense. We will be using only a small fraction of the options available in Eclipse. This chapter is designed to get you started with the basics.

1.3 Installing Eclipse on Your Personal Computer

To install Eclipse on your personal computer, follow the instructions for your computer platform.

1.3.1 Windows

1. Download the Eclipse IDE from the Eclipse website.
   (a) Go to http://eclipse.org.
   (b) Click on the Download button.
   (c) Click the Eclipse IDE for Java Developers link.
   (d) Click either the Windows 32-bit or the Windows 64-bit download link depending on your system. You can determine the bit size of your system by going into the control panel and selecting System. Beside System type it should say either 64-bit Operating System or 32-bit Operating System.
   (e) Click Download, the file will be stored in the Downloads folder of your computer. The file will be a zip file with a name that begins with eclipse-java.
2. Unzip the file. The process here will depend on the zip utility you have installed. In most cases you can right click the zip file, go to either the zip utility submenu or it may be listed in the main popup menu, and select something like Extract Here. When this is finished you should have a new folder called eclipse.
3. Most programs that are designed for Windows computers have an installation process, Eclipse does not, it is completely stand-alone, and is ready to run. You may move the eclipse folder to any place on your computer, just remember where it is since you will need to use it to start Eclipse.
4. After you have moved the folder to where you want it, go into the eclipse folder and you will see a file eclipse.exe with the eclipse icon. If you double-click eclipse.exe, Eclipse should start. If you get an error, you may have downloaded the wrong bit version of eclipse, if so, go back and download the other version and repeat the above process.
5. Once Eclipse has started you will be asked for the workspace you want to use. If you have not already created a folder for your work, you may wish to cancel Eclipse and create a folder before opening Eclipse. You are now ready to go. You can create a shortcut to Eclipse and place it on your desktop and/or pin it to your start menu or taskbar.
6. At this point you can delete the eclipse-java-XXX.zip file (where XXX is just the rest of the name).

1.3.2 Mac

1. Download the Eclipse IDE from the Eclipse website.

(a) Go to http://eclipse.org.

(b) Click on the Download button.

(c) Click the Eclipse IDE for Java Developers link.

(d) Click Mac OS X (Cocoa) 64-bit.

(e) Click Download, the file will be stored in the Downloads folder of your computer. The file will be a tar.gz file with a name that begins with eclipse-java.

2. The download will go to your Downloads folder. You will normally have an icon for the Downloads folder on the right-hand side of the dock. Find eclipse-java-XXX.tar.gz (where XXX is just the rest of the name) in your Downloads folder, and drag it to the Desktop. Then double-click it. You will see a folder named "eclipse."

3. Drag the "eclipse" folder into your Applications folder. The easiest way to do so is to open a new window in the Finder and click on Applications in the list you get on the left-hand side. Then drag the "eclipse" folder in with the other applications. Make sure that you do not drag it into a folder that's already within Applications. In other words, when you're done, the Applications folder should have directly within it a folder named "eclipse."

4. (This step is not required, but it's strongly recommended.) Double click the "eclipse" folder. You'll see an application named "Eclipse"; it has a purple icon with white horizontal stripes. Drag it into your dock. Now you will be able to launch Eclipse by clicking on the icon in the dock.

5. You may now drag eclipse-java-XXX.tar.gz to the Trash. Empty the Trash whenever you wish.

6. When you launch Eclipse for the first time, you'll be asked "'Eclipse' is an application downloaded from the Internet. Are you sure you want to open it?" Click "Open."

1.3.3 Linux

You can download either a 32 or 64 bit version of Eclipse, it will be a tar.gz file, untar it and run the eclipse program. Most likely, Eclipse will be installable from your distribution's software manager. It might be an older version but that will not matter for us.

1. Download the Eclipse IDE from the Eclipse website.

(a) Go to http://eclipse.org.

(b) Click on the Download button.

(c) Click the Eclipse IDE for Java Developers link.

(d) Click on either the Linux 32-bit or Linux 64-bit links.

(e) Click Download, the file will be stored in the Downloads folder of your computer. The file will be a tar.gz file with a name that begins with eclipse-java.

2. The download will go to your Downloads folder. Find eclipse-java-XXX.tar.gz (where XXX is just the rest of the name) in your Downloads folder. Extract the contents of the tar file. You will see a folder named "eclipse."

3. In the eclipse folder there is an executable file named eclipse. Simply run this program to start up Eclipse.

1.4 Creating a Workspace

Before we open up Eclipse we want to create a place to store our programs. I usually create a subfolder of my documents folder or I create a special library (in Windows) to hold all of my programming work. Since I do more than just program in Java, I usually create a subfolder of the Programming folder that is named for the types of work in it or the name of the class the code is being used for.

Now launch Eclipse by double clicking the eclipse icon in the Academic Software window. When you do a Workspace Launcher will appear. YOUR WORKSPACE IS VERY IMPORTANT! This is where Eclipse will find all of your programs. If you select the wrong workspace you may not find your files or even worse they could be deleted by others, if your workspace is saved to a shared location on the computer.

Eclipse will suggest a default location of "workspace" when you first launch Eclipse, but I do not use the default location. In the screen-shot below, the workspace location is a subfolder "COSC117" of the Programming library I created on my Windows machine. The COSC117 is the class at Salisbury University for which these notes have been created.

To select a different workspace, click on the Browse... button and navigate to the desired folder, and then click OK. Leave the "Use this as the default and do not ask again" check box unchecked. You can have any number of different workspaces as you would like. For example, I have one for each class I teach that uses Java, and one for other software packages that I write. At this point you should see the following.

Click the X on the Welcome tab to close the welcome window. Now you should see the Main IDE Setup.

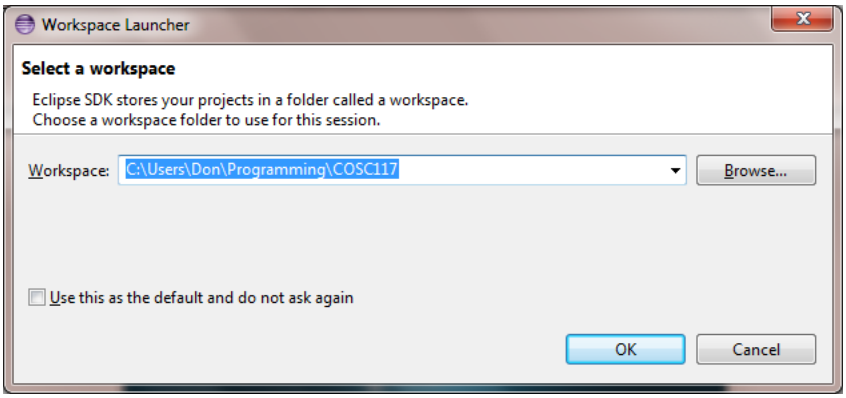


Figure 1.1: Workspace Selector

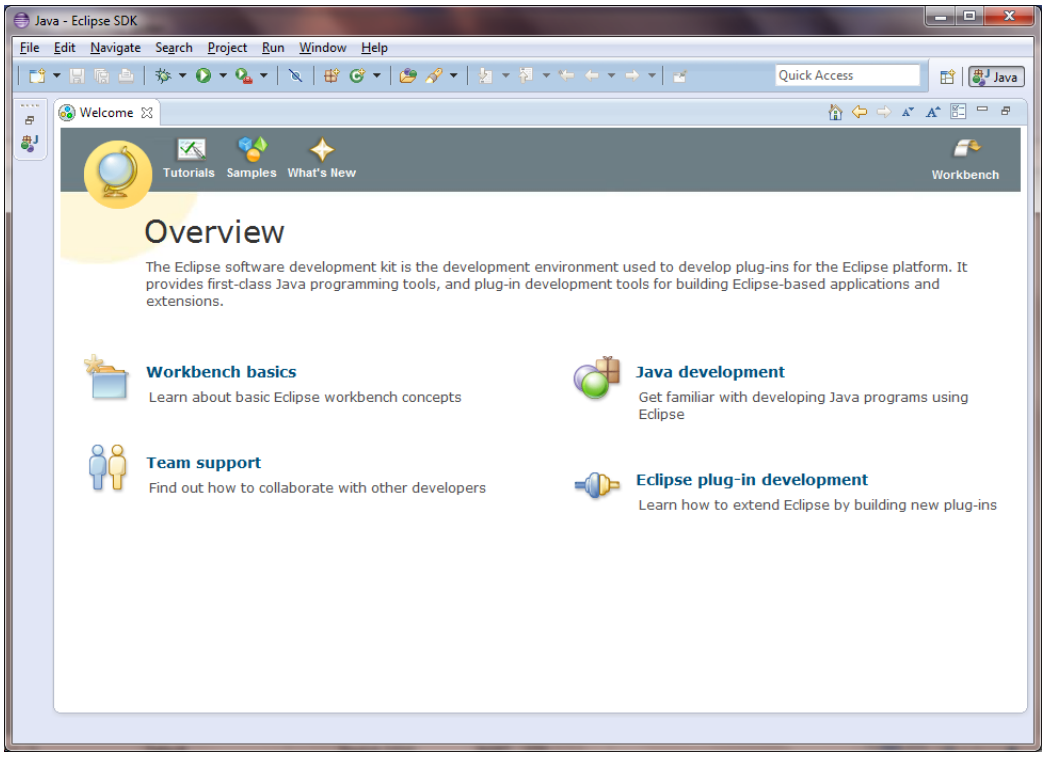


Figure 1.2: Welcome Screen

1.5 The Eclipse IDE Layout

There are four main areas in the interface.

Package Explorer: This area, #1, will hold all of the projects (that is, programs) you write and keep in that workspace. Professional programmers will organize their work by using multiple workspaces but for this class a single workspace

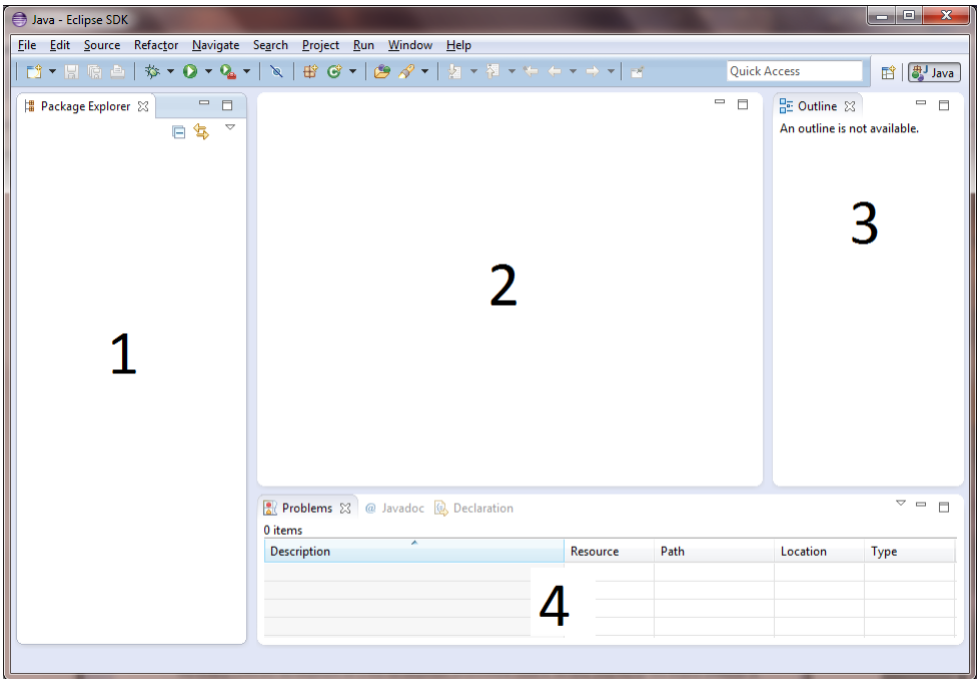


Figure 1.3: Main IDE Setup

holding all of our programs will be sufficient.

Editing Area: This area, #2, is where you will write the code for your programs.

Outline: This area, #3, will contain a sort of outline to your program. That is, it will list all of the "functions" contained in your program and make navigation of your program much easier. This area may not seem too useful at the start when our programs are small but as they get bigger and more complex using the outline area will save you a lot of time.

Output: The output area, #4, has many functions. It is where you will see descriptions of your errors and where you will see "console" output of your programs.

1.6 Your First Java Program: Hello World

To create a Java program in Eclipse we need to first create a Java Project. To do this select File > New > Java Project from the main menu, the far left tool in the toolbar is the New tool so clicking this and selecting Java Project from the drop-down menu will also work. When you do either of these, the New Java Project dialog will appear.

Each project must have a name, type in `HelloWorld` and then click Finish. All of the other default settings will be fine. At this point the Package Explorer window

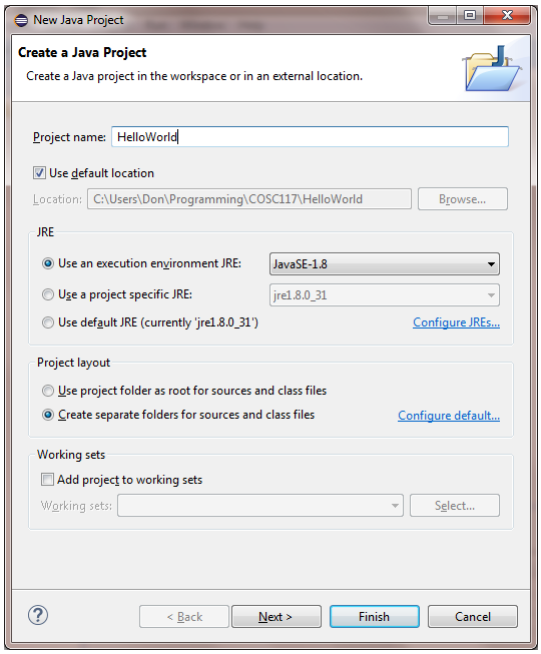


Figure 1.4: Project Creation

will display the new project HelloWorld. If you click the little triangle beside the HelloWorld title you will see several sub folders of information. One of these is src, which stands for source, this is where all of the Java code files will be stored for this project. Each project has its own set of code files. So each programming exercise will have its own project and for most of the semester each project will have a single code file in it. Notice that the src folder is empty at this point. This is because we need to add a code file to the project.

To add a code file select File > New > Class from the main menu, the far left tool in the toolbar is the New tool so clicking this and selecting Class from the drop-down menu will also work. When you do either of these, the New Class dialog will appear.

You must give the class a name, it is not necessary to give the class the same name as the project but for now it is convenient to do so. So type in the name `HelloWorld`. Also, select the `public static void main(String[] args)` check box and uncheck all of the others, leave the other options as they are and click Finish. At this point there will be one file in the src folder and Eclipse will put in a template structure into the editing window, as seen below.

```
1 public class HelloWorld {
2
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5
6     }
7
8 }
```

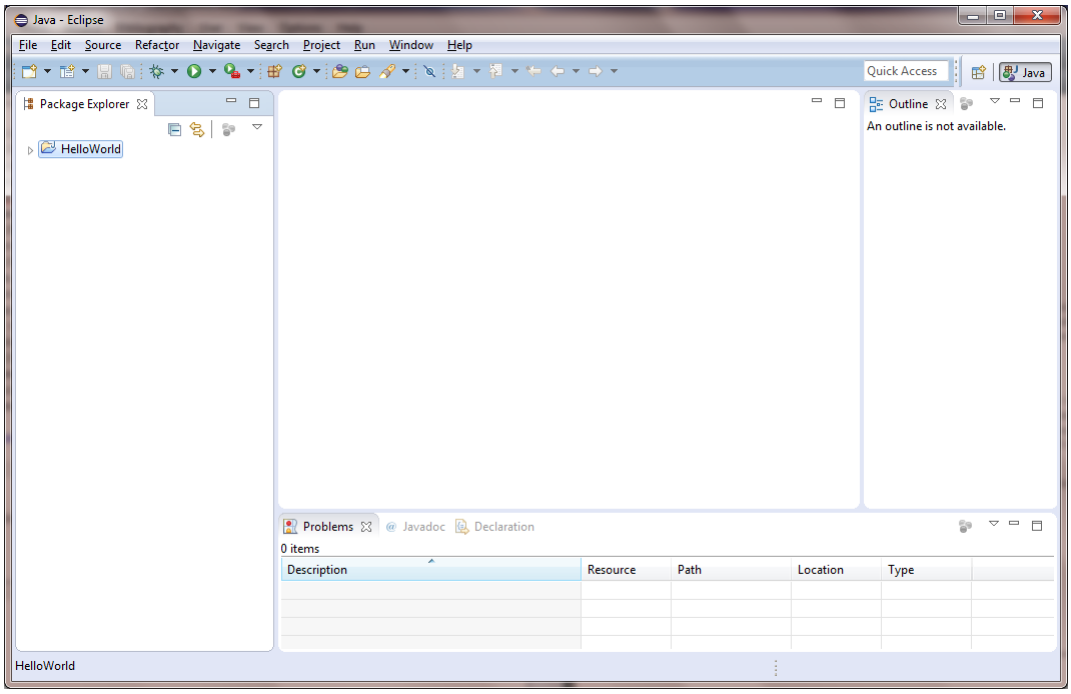


Figure 1.5: Workspace with HelloWorld

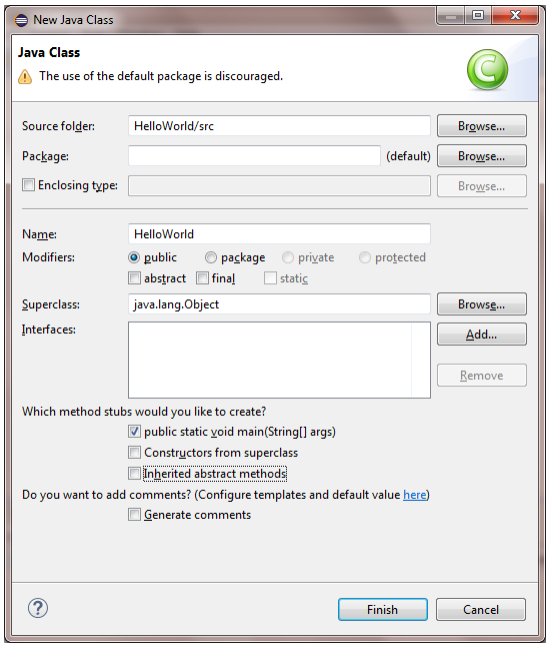


Figure 1.6: Java File Creation

Do not worry about what all the words are, we will discuss their meaning later on in the document. For now, edit the code so that the editing window looks like the

following. Of course, you should put your name in as the author in place of mine and today's date for the date.

```
1  public class HelloWorld {
2
3      /*-
4       * The Hello World program, in Java.
5       * Author: Don Spickler
6       * Date: 8/31/2015
7       */
8
9      public static void main(String[] args) {
10         System.out.println("Hello World");
11     }
12 }
```

Now we are ready to run the program. Select Run > Run from the main menu or click the run tool button, the big green play button. At this point you should see Hello World appear in the console window a the bottom of the screen. If you get an error instead, check to make sure that your code looks like the code of the program above.

Although this is the easiest program you will ever write in Java, and most likely the easiest program you will ever write, the process will be the same for nearly all of the Java programs you create in Eclipse. First create a Java Project then add a main class to the project. Later on, we will discuss adding several classes to a project but there will be only one main class, this is the starting point for the program when it is run.

Chapter 2

Input & Output

2.1 Introduction

Probably the simplest programs that you will write are the input, calculation, output format.

$$\text{Input} \Longrightarrow \text{Do Calculation} \Longrightarrow \text{Output}$$

That is, you get some type of input from the user, do some calculation on the input, and finally output the results. Although this seems rather simplistic, a lot of programs are that simple. If you have a credit card, then you get a bill each month, if you do not do the direct bank account withdraw. When the credit card company gets ready to send out a bill they simply run a billing program that takes as input all of the charges you made along with any previous balance, calculates the new bill, and outputs it in printed form, which then gets sent to you. Also, when you are ready to graduate, the registrar runs a graduation audit program that takes as input all of your courses and grades and determines if you have graduated. So the calculations are a check to see if you have completed all of the courses in your major and if you have satisfied the other requirements for graduation, such as general education, total credit count, number of upper-level credits, and so on. The output is very simple, yes you graduate or no you don't.

Of course, not all programs fall into this category. For example, computer games continually take input from the player and respond accordingly. There are calculations that are being done, in fact an enormous number of calculations. There is also output, the game play on the screen. But since the program is continually taking input and continually producing output we would classify a computer game differently.

Input can come from many different sources, the keyboard, mouse, trackball, joystick, game pad, a network connection, blue-tooth connection, and so on. Most of

the time we will be using just the keyboard as our input device. Similarly, output can go many different places, the computer screen, a printer, the network, etc. For us, the output will be the computer screen, more specifically, the console box at the bottom of the Eclipse IDE window.

2.2 Dollars To Euros Converter Example

Example 1: Create a program that will convert US Dollars to Euros.

Although this is rather simple it is a good exercise to go through the process of constructing an algorithm. There are three questions you should ask yourself,

1. What do I want as the final result of the program?
2. What is the calculation I will need to do?
3. What do I need to complete this calculation?

Note that this is line of questioning is going from the end of the program, (what do we want to produce) to the beginning of the program (what is needed as input from the user).

1. What do I want as the final result of the program? **The number of Euros that corresponds to the number of US Dollars I have.**
2. What is the calculation I will need to do? **Dollars times the number of Euros per Dollar**
3. What do I need to complete this calculation? **The number of dollars the user has to exchange and the current Euros per Dollar exchange rate.**

So we need two numbers to be input, the number of dollars and the current exchange rate. Then we do a simple calculation, a single multiplication. Finally we output the result. This is essentially our algorithm.

Algorithm:

1. Input the number of dollars the user has to exchange.
2. Input the current Euros per Dollar exchange rate.
3. Multiply the number of dollars by the exchange rate to get the number of euros.
4. Output the number of euros.

In Java, any information that is to be entered into the program, or information that is to be used by the program must have a storage location. So we will need to create a storage location for the number of dollars and a storage location for the conversion rate. Also, Java is strongly typed, which means that these storage locations must be designated as a particular data type, which is the type that it is going to store. We will discuss data types in Java later in the notes but for now we will make the simple distinction between integer and real number. An integer is a counting number, 0, or a negative counting number. For example, -5, 0, 2, and 7 are all integers. A real number can have a decimal portion, for example, 3.14159 is a real number. In Java, and many other languages, there is a distinction that is made between storing integers and storing decimal numbers. As you know, all integers are real numbers, so we could always store an integer as a real but there will be times when we want to store integers. In our example, both the dollars and conversion rate are possibly decimal numbers. We could want to exchange $178.34 and the rate might be 2.13 euros per dollar.

The way we create a storage location is we start a statement with the type of data we are going to store, in this example we will use `double`. A `double` is a decimal data type. We then give the storage location a name, that is, a label, we will use `dollars`. Then we put (or assign) a value to be stored, more on that later. The name of a storage location can be any letter followed by letters, numbers, and the underscore. So `dollars` is a valid name, as would be `MyDollars`, `My_Dollars`, `mine1234Save5`, and `My_Saved_Dollars_To_Exchange`. These names that label storage locations are call variables. We will do the same with the conversion rate which we will call `eurosPerDollar`.

To get the values from the user of the program we need to link up the keyboard to the program so that when the program is running and the user types in something, what they type in is transferred to the program. We do this with a Scanner object, which is built into Java.

Once the values are input by the user we need to do the calculation. The calculation is an intermediate value and it must be stored somewhere, so we will also make it a `double` and call it `euros`.

From here all we need to do is display the result. Printing information to the screen, or console window to be more specific, we use either a print statement or a println statement. The difference is that the println statement will move the cursor to the next line, and the print will not.

We will now display the entire program along with a run of the program, afterwards we will dissect the program in detail.

```
1 import java.util.Scanner;
2
3 public class DollarsToEurosConverter {
4
```

```
5       /*-
6        * DollarsToEurosConverter converts dollars to euros.
7        * Author:  Don Spickler
8        * Date: 1/31/2011
9        */
10
11      public static void main(String[] args) {
12          Scanner keyboard = new Scanner(System.in);
13          System.out.print("How many dollars do you want to convert? ");
14          double dollars = keyboard.nextDouble();
15          System.out.print("What is the euros-per-dollar exchange rate? ");
16          double eurosPerDollar = keyboard.nextDouble();
17          keyboard.close();
18          double euros = dollars * eurosPerDollar;
19          System.out.print(dollars);
20          System.out.print(" dollars = ");
21          System.out.print(euros);
22          System.out.print(" euros");
23      }
24  }
```

Program Output **DollarsToEurosConverter.java**

```
How many dollars do you want to convert? 25
What is the euros-per-dollar exchange rate? 2.13
25.0 dollars = 53.25 euros
```

Now let's look at this program line by line. At the top we have an import statement. Java is too large to load all of it into memory, at least it would not be a very efficient way to do it. So whenever you need something beyond the basic Java you need to import it.

```
import java.util.Scanner;
```

This import statement loads in the Scanner object which is what we use to link the program up to the keyboard for input. Later on we will use the scanner to get input from a file.

Next is the class header, this was produced by Eclipse and should not be changed.

```
public class DollarsToEurosConverter {
```

Java expects that the name of the class is exactly the same as the name of the file it is in, so changing this line may cause the program not to compile.

The next block of text is a comment. A comment is text that is ignored by the compiler. This allows the programmer to annotate their code with useful comments to anyone who is reading the code. In Java there are two main types of comments. The single line comments start with two forward slashes `//`.

```
double euros = dollars * eurosPerDollar;  // Calculate Euros
```

In the above statement, the portion before `//` is compiled and the portion after it is a comment that is not compiled. These comments last only to the end of the line, the next line of the program will be compiled.

A block comment is one that is between `/*` and `*/`. This comment is multi-line and the compiler will skip all of the text between the `/*` and `*/`. So our comment block containing the short description, author, and date is skipped by the compiler. There are two more special types of block comments. Comments between `/**` and `*/` are JavaDoc comments. JavaDoc is an automated documentation system for creating external documentation for a program or system of programs. We will not be using JavaDoc in this set of notes but if you go further into computer science you may wish to investigate this and other documentation systems.

Another type of special block comment is between `/*-` and `*/`. This is not a Java type comment but an Eclipse special comment. As far as Java is concerned this is just a regular block comment between `/*` and `*/`. Eclipse has a very nice feature of auto indentation, indentation is very important in reading a program since is makes seeing blocks of code more easily. In Eclipse, Shift+Ctrl+F (or Shift+Command+F on the Mac) will automatically indent your program. Unfortunately, it will also reformat your block comments. Using `/*-` and `*/` for a block comment will tell Eclipse's auto-formatter to skip the comment.

The next line is the header for the main method, as with the class header, this should not be changed.

```
public static void main(String[] args) {
```

The next line is what links the program to the keyboard. The variable keyboard is simply a variable and could be named differently, but since it is going to represent input from the keyboard the name keyboard is fitting. Don't worry about what is happening on this line, later on it will make sense, for now it is just the statement that activates the keyboard.

```
Scanner keyboard = new Scanner(System.in);
```

Up to now the code has just been technical setup, here is where our algorithm starts. We begin with inputting the data we need. The `System.out.print` statements are simply a prompt for the user so they know what they are inputting. Note that in these print statements there is a line of text between " and ". Text between double quotes is a string literal and is printed out verbatim. We can do far more with these print statements as we will see. The line,

```
double dollars = keyboard.nextDouble();
```

is where we get information from the keyboard. The `double dollars` declares the variable (i.e. storage location) dollars as a numeric decimal value. The

```
keyboard.nextDouble()
```

pauses the program and waits for the user to input a decimal number from the

keyboard and press enter. Once enter is pressed, the number is sent into the storage location for dollars. The same is true for the conversion rate line.

```
System.out.print("How many dollars do you want to convert? ");
double dollars = keyboard.nextDouble();
System.out.print("What is the euros-per-dollar exchange rate? ");
double eurosPerDollar = keyboard.nextDouble();
keyboard.close();
```

The last statement in this block is closing the keyboard. This really does not need to be done for a keyboard but in general you should close an input device when you are done with it. This is more of an issue with files. So in this example code you will see times when we close the keyboard and some times when we do not.

The next line is the calculation portion of the algorithm. in this line, the beginning `double euros` creates a storage location named euros and declares it as a numeric decimal type. The program then evaluates the right hand side of the statement and puts the result into the location for euros. This is called an assignment, when we assign a variable a value. The keyboard input statements above are also assignment statements.

```
double euros = dollars * eurosPerDollar;
```

The final block of code is the last segment of our algorithm, displaying the results. As before we use a print statement to display information to the console window. Note that when we use a variable, the contents of that variable are printed to the screen.

```
System.out.print(dollars);
System.out.print(" dollars = ");
System.out.print(euros);
System.out.print(" euros");
```

Although we used four lines of code to display our results, we could have done this in a single line. The + between the segments means to concatenate the results together.

```
System.out.print(dollars+" dollars = "+euros+" euros");
```

Another thing you will notice is that every statement in Java ends with a semi-colon. The only exception to this is when a block of code ends. A block of code is any code between curly brackets, `{` and `}`.

2.3 Cylinder Volume Example

Example 2: Create a program that will calculate the volume of a cylinder.

Again, a rather simple task, the same three questions apply here,

1. What do I want as the final result of the program? **The volume of a cylinder.**
2. What is the calculation I will need to do? $V = \pi r^2 h$.
3. What do I need to complete this calculation? **The radius of the cylinder, r, and the height of the cylinder, h.**

Algorithm:

1. Input the radius of the cylinder.
2. Input the height of the cylinder.
3. Calculate $V = \pi r^2 h$.
4. Output the volume, V.

```
1  import java.util.Scanner;
2
3  public class CylinderVolume {
4
5      /*-
6       * CylinderVolume prints out the volume of
7       * a cylinder given its height and radius.
8       * Author:  Don Spickler
9       * Date: 1/31/2011
10      */
11
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14         System.out.print("Input the radius of the cylinder: r =  ");
15         double radius = keyboard.nextDouble();
16         System.out.print("Input the height of the cylinder: h =  ");
17         double height = keyboard.nextDouble();
18         keyboard.close();
19         double volume = Math.PI * Math.pow(radius, 2) * height;
20         System.out.printf("A cylinder of radius %.4f, and height %.4f has volume %.4f.",
               radius, height, volume);
21     }
22 }
```

Program Output **CylinderVolume.java**

```
Input the radius of the cylinder: r =  3
Input the height of the cylinder: h =  5
A cylinder of radius 3.0000, and height 5.0000 has volume 141.3717.
```

This program is very similar to the previous one, since both the radius and the height could be decimal numbers we declare them as doubles and the same goes for the calculation of the volume.

There are a couple new bits of syntax. Notice in the calculation of the volume we have `Math.PI` which will give us an approximation to π and `Math.pow(radius,2)` which raises the value of the radius to the second power. The `Math` prefix in these statements is Java's Math class, which contains many mathematical functions.

2.4 Sphere Properties Example

Example 3: Create a program that will calculate the volume and surface area of a sphere.

1. What do I want as the final result of the program? **The volume and surface area of a sphere.**
2. What are the calculations I will need to do? $V = \frac{4}{3}\pi r^3$ and $S = 4\pi r^2$.
3. What do I need to complete this calculation? **The radius of the sphere, r.**

Algorithm:

1. Input the radius of the sphere.
2. Calculate $S = 4\pi r^2$ and $V = \frac{4}{3}\pi r^3$.
3. Output the volume, V, and surface area, S.

```
1  import java.util.Scanner;
2
3  public class SphereProperties {
4
5      /*-
6       * SphereProperties prints out the volume and surface area of
7       * a sphere given its radius.
8       * Author:  Don Spickler
9       * Date: 1/31/2011
10      */
11
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14         System.out.print("Input the radius of the sphere: r =  ");
15         double radius = keyboard.nextDouble();
16         keyboard.close();
17         double area = 4 * Math.PI * Math.pow(radius, 2);
18         double volume = 4 / 3 * Math.PI * Math.pow(radius, 3);
19         System.out.printf("In a sphere of radius %.4f, the volume is %.4f and the surface
               area is %.4f.", radius,
20                 volume, area);
21     }
22 }
```

Program Output **SphereProperties.java**

```
Input the radius of the sphere: r =  5
In a sphere of radius 5.0000, the volume is 392.6991 and the surface area is 314.1593.
```

Here is a good time to talk about program testing. Although the program has no syntax errors or run-time errors there is a logical error. If you check the answers to these calculations you will find that the surface area calculation is correct but the volume calculation is incorrect. In the next chapter we will investigate data types a bit further and at that point it will be apparent what is wrong with our calculation,

```
double volume = 4 / 3 * Math.PI * Math.pow(radius, 3);
```

2.5 String Example

Example 4: Now for something a little different. Create a program that will take a person's name in informal form (that is, first last) and print it out in formal form (that is, last, first).

1. What do I want as the final result of the program? **The person's name in formal form.**
2. What are the calculations I will need to do? **This is not what most people would consider a calculation but it is a manipulation of data. We need to separate the first name and the last name, reverse their order, and put a comma between the two.**
3. What do I need to complete this calculation? **The person's name in informal form.**

Algorithm:

1. Input the person's name in informal form.
2. Separate the first name and the last name, reverse their order, and put a comma between the two.
3. Output person's name in formal form.

In this example we are storing textual information that is coming from the user, not numeric values. Textual information can be stored in a String. A string is simply a sequence of characters. There are no length restrictions to a string in Java, they can be as long as you would like. In addition, since we are inputting strings we will not be using the nextDouble() statement for the keyboard. To input strings, the scanner object has two functions, next() and nextLine(). Next extracts the next set of characters and stops at the first white-space it encounters, such as a space or tab. The nextLine() statement reads in all characters, including white-space, until it encounters the enter key. So the way that the Java language deals with input of strings allows us to combine steps one and two of our algorithm. That is, if we use a next() statement, this will extract the first name and then we use a second next() statement to extract the last name. From there we simply use print statements to produce the formal name. This is one of many cases where the way the programming

language is set up changes the way we organize the process or algorithm to accomplish our task. So although an algorithm is a general process to accomplish a task, it may be implemented differently depending on the specific language it is being implemented in.

```
1  import java.util.Scanner;
2
3  /*-
4   * StringExample --- familiar to formal name.
5   * Author:  Don Spickler
6   * Date: 2/2/2011
7   */
8
9  public class StringExample {
10     public static void main(String[] args) {
11         Scanner keyboard = new Scanner(System.in);
12         System.out.print("Input your name as, first last: ");
13         String firstName = keyboard.next();
14         String lastName = keyboard.next();
15         System.out.println(lastName + ", " + firstName);
16         keyboard.close();
17     }
18 }
```

Program Output **StringExample.java**

```
Input your name as, first last: Don Spickler
Spickler, Don
```

Chapter 3

Data Types

3.1 Introduction

As we mentioned before, Java is a strongly typed language. This means that every variable in a Java program must be given a type that it represents. Remember that a variable name is simply a label to a storage location in memory and this type is telling Java what is to be stored in that location. There are a lot of data types that are built into Java, we have seen a few of them in the last chapter, double, Scanner, and String. Then you declare a variable the data type comes first followed by the variable name and then an optional assignment statement to initialize the variable to a value. For example,

```
double t = 4;
```

declares the variable t and sets the value of it to 4.

```
double t;
```

Will declare the variable t but not initialize it to a value. In Java, you must initialize the variable before you use it, so in the code,

```
double t;
double s = 3 * t;
```

Java will give you an error on the second line since `t` has been declared but not initialized before we are using it. The declaration,

```
double area = 4 * Math.PI * Math.pow(radius, 2);
```

creates the variable `area` and sets it to $4\pi r^2$, where r is the radius. The declaration,

```
Scanner keyboard = new Scanner(System.in);
```

creates the variable `keyboard` and sets it to a scanner that is linked up to the computer keyboard.

There are four basic integer types, `byte`, `short`, `int`, and `long`. Of these you will probably use `int` the most and from time to time `long`. Integer types are whole numbers, counting numbers, 0 and the negative of the counting numbers, basically no decimal places. So 3, 17, -5, and 1234567890 are all integers but 15.6 is not an integer.

There are two basic decimal types, `float` and `double`. Although we tend to use `double` more often in these examples, we could use a `float` in most situations.

A computer is a finite device, it can do a lot of things but it cannot do an infinite number of things nor can it store an infinite number of data items. So it is not capable of storing every integer or every decimal number, it has limitations. So there is a limit on the size of the integers the machine can store and there is a limit on the number of decimal places that a float or double can store. This is the difference between the data types that are listed above. The four types of integers store different sizes of numbers and the two types of decimal types store different sizes of real numbers and different sizes of decimal place accuracies. We will look at the limits of these data types in an example a little later on.

In general, if you use a data type that stores more it takes more memory. The computer has a finite amount of memory as well. It is unlikely that we will run out of memory in this class, unless we try really hard, but if you go on into computer science you will be in a situation where you will need to be careful about the amount of memory you are using. So if you are using doubles where floats would be sufficient you would be better off using floats, they take less space and they process faster. On the other hand, if you need the extra accuracy for your application you need a double.

3.2 Data Type Input and Assignment Example

```
1  import java.util.Scanner;
2
3  /*-
4   * InputExamples --- Example of different input types and data types.
5   * Uncomment some of the assignment statements to see that is allowed
6   * and what is not.
7   * Author:  Don Spickler
8   * Date: 2/2/2011
9   */
10
11 public class InputExamples {
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14         System.out.print("Input an integer: ");
15         int num1 = keyboard.nextInt();
16         System.out.println(num1);
17
```

```
18          System.out.print("Input a double: ");
19          double num2 = keyboard.nextDouble();
20          System.out.println(num2);
21
22          System.out.print("Input a byte: ");
23          byte num3 = keyboard.nextByte();
24          System.out.println(num3);
25
26          System.out.print("Input a long: ");
27          long num4 = keyboard.nextLong();
28          System.out.println(num4);
29
30          /*-
31          num1 = num4;
32          num3 = num4;
33          num2 = num4;
34          num4 = num1;
35          num4 = num3;
36          num4 = num2;
37          */
38      }
39  }
```

Program Output **InputExamples.java**

```
Input an integer: 25
25
Input a double: 23.568945
23.568945
Input a byte: 123
123
Input a long: 123456789123
123456789123
```

In the example above, remove the block comment and you will see that Eclipse will show you several errors. In fact, you will get errors on lines 31, 32, and 36.

On line 31 you get the error *Type mismatch: cannot convert from long to int* because the long is "larger" than the int you cannot put something larger into a smaller memory location. On line 32 you get the error *Type mismatch: cannot convert from long to byte* for exactly the same reason.

On line 36 you get the error *Type mismatch: cannot convert from double to long* since a double is a decimal type value and a long is an integer type value, if you would store the double into the long you would lose the decimal portion of the double. You can force Java to put a double into a long location but you still lose the decimal portion of the double, we will discuss how to do this later.

Notice that lines 33, 34, and 35 do not have any errors. So you can store a long in a location created for a double. You can also store an int in a location created for

a long and likewise you can store a byte in a location created for a long.

3.3 Data Type Sizes Example

As we mentioned above, these different data types use a different amount of computer memory and hence have different ranges of values that they can store. This example shows the minimum and maximum values for each data type as well as the number of bits used by each data type. If you read through the code carefully you will see that we doe not use a long but we use a Long. Java is a case-sensitive language, so long and Long represent two different things. The Long is called a wrapper class for the long data type. The same is true for double and Double, and so on. We will seldom use the wrapper classes but they do have a nice feature of being able to extract the maximum and minimum values for a data type, which is why we use them here.

```
1  public class DataTypes001 {
2
3      /*-
4       * DataTypes001 prints out the range and bit sizes of the native data types.
5       * Author:  Don Spickler
6       * Date: 2/3/2011
7       */
8
9      public static void main(String[] args) {
10         System.out.println("byte minimum: " + Byte.MIN_VALUE);
11         System.out.println("byte maximum: " + Byte.MAX_VALUE);
12         System.out.println("short minimum: " + Short.MIN_VALUE);
13         System.out.println("short maximum: " + Short.MAX_VALUE);
14         System.out.println("integer minimum: " + Integer.MIN_VALUE);
15         System.out.println("integer maximum: " + Integer.MAX_VALUE);
16         System.out.println("long minimum: " + Long.MIN_VALUE);
17         System.out.println("long maximum: " + Long.MAX_VALUE);
18         System.out.println("float minimum: " + Float.MIN_VALUE);
19         System.out.println("float maximum: " + Float.MAX_VALUE);
20         System.out.println("double minimum: " + Double.MIN_VALUE);
21         System.out.println("double maximum: " + Double.MAX_VALUE);
22
23         System.out.println("byte bits: " + Byte.SIZE);
24         System.out.println("short bits: " + Short.SIZE);
25         System.out.println("integer bits: " + Integer.SIZE);
26         System.out.println("long bits: " + Long.SIZE);
27         System.out.println("float bits: " + Float.SIZE);
28         System.out.println("double bits: " + Double.SIZE);
29     }
30 }
```

Program Output **DataTypes001.java**

```
byte minimum: -128
byte maximum: 127
short minimum: -32768
short maximum: 32767
integer minimum: -2147483648
integer maximum: 2147483647
long minimum: -9223372036854775808
long maximum: 9223372036854775807
float minimum: 1.4E-45
float maximum: 3.4028235E38
```

```
double minimum: 4.9E-324
double maximum: 1.7976931348623157E308
byte bits: 8
short bits: 16
integer bits: 32
long bits: 64
float bits: 32
double bits: 64
```

From the above program output we see that a byte, which is an integer type, goes from a value of -128 up to 127. It cannot store any value outside of that range. We also see that a byte uses 8 bits of information. A bit is a BInary digiT, we think of them as a 1 or a 0. So 10100101 is a byte of information. So our possible bytes are 00000000, 00000001, 00000010, ..., 11111110, 11111111. So each of the 8 bit locations can be either a 0 or a 1 and hence there are $2^8 = 256$ possible numbers that can be held by a single byte. This is the same number of integer values between -128 and 127, remember to count 0.

A short uses 16 bits (or 2 bytes) so there are $2^{16} = 65536$ possible numbers that can be held by a short, hence the range -32768 and 32767. An int uses 32 bits (or 4 bytes) so there are $2^{32} = 4294967296$ possible numbers that can be held by an int, hence the range -2147483648 and 2147483647. Finally a long uses 64 bits (or 8 bytes) so there are $2^{64} = 18446744073709551616$ possible numbers that can be held by a long, hence the range -9223372036854775808 and 9223372036854775807.

From the output above we see that a float uses 32 bits and a double uses 64 bits. We will not discuss the way that decimal numbers are stored in the computer but we will look at the ranges. For the float, the smallest value it can hold is 1.4E-45, which means $1.4 \cdot 10^{-45}$, up to 3.4028235E38, which is $3.4028235 \cdot 10^{38}$. A double stores values in the range of $4.9 \cdot 10^{-324}$, up to $1.7976931348623157 \cdot 10^{308}$. Note that this is for both positive and negative numbers, so a float can hold values between $-3.4028235 \cdot 10^{38}$ to $-1.4 \cdot 10^{-45}$ and between $1.4 \cdot 10^{-45}$ and $3.4028235 \cdot 10^{38}$. For a double, the range is $-1.7976931348623157 \cdot 10^{308}$ to $-4.9 \cdot 10^{-324}$ and $4.9 \cdot 10^{-324}$ to $1.7976931348623157 \cdot 10^{308}$.

Also keep in mind what we said before about the decimal numbers. The computer cannot store all possible decimal values between these bounds, there are an infinite number of them. So there is only so much accuracy that a float and double can provide. If we run the InputExamples.java program above and put in a large number for the double we see that the output is rounded.

Program Output **InputExamples.java**

```
Input a double: 12345678901234567890123456789012345678901234567890
1.2345678901234568E39
```

Instead of the machine storing 12345678901234567890123456789012345678901234567890 it stores $1.2345678901234568 \cdot 10^{39} = 12345678901234568000000000000000000000000000000$. So

the storage is only good to about 16 significant digits. If we update the InputExamples.java program to do the same for a float we get,

Program Output **InputExamples.java**

```
Input a float: 12345678901234567890
1.2345679395506094E19
```

This shows that with a float we get about 7 significant digits before we notice round-off error.

Since we are discussing the amount of space that is taken up by variables let's discuss computer memory sizes a little. A kilobyte (KB) is 1000 bytes if we take the meaning of kilo for what it usually means. In computer science, a kilobyte is really $2^{10} = 1024$ bytes. This is enough storage to hold approximately the above two paragraphs. A megabyte (MB) is 1024 kilobytes, or 1,048,576 bytes. This is enough storage for a 250 page document. A gigabyte (GB) is 1024 megabytes, or 1,073,741,824 bytes. This is enough space to store a 256,000 page document, about an average of 500 books. So if your computer has 8 GB of memory it has 8,589,934,592 bytes of storage, the capacity to hold about 4,000 books. Although this is a lot of information, it is still finite, and it is fairly amazing the number of times we run out of memory. Your hard drives can store far more information, they are probably close to a terabyte up to a couple terabytes. A terabyte is 1024 gigabytes, or 1,099,511,627,776 bytes.

3.4 Some Mathematical Functions Example

Our next example shows just a few of the many mathematics functions that are available in the Java language. Most of the mathematical functions that you will use are in the Math class in Java, so they will be of the form Math.sin(23.7), that is, Math followed by a period, followed by the function name (sin, cos, abs, max, ...), followed by the values the function will operate on.

```
1  public class MathExamples {
2
3      /*-
4       * MathExamples --- just some calculations.
5       * Author:  Don Spickler
6       * Date: 2/2/2011
7       */
8
9      public static void main(String[] args) {
10         double num1 = Math.random();
11         System.out.println("Random Number: " + num1);
12         System.out.println("e =  " + Math.E);
13         System.out.println("pi = " + Math.PI);
14         System.out.println("floor(27.8) = " + Math.floor(27.8));
15         System.out.println("floor(17.0) = " + Math.floor(17.0));
16         System.out.println("floor(-7.6) = " + Math.floor(-7.6));
```

```
17          System.out.println("sqrt(2) = " + Math.sqrt(2));
18          System.out.println("sqrt(16) = " + Math.sqrt(16));
19          System.out.println("1/3 = " + (1 / 3));
20          System.out.println("4/3 = " + (4 / 3));
21          System.out.println("1.0/3.0 = " + (1.0 / 3.0));
22          System.out.println("1.0/3 = " + (1.0 / 3));
23          System.out.println("1/3.0 = " + (1 / 3.0));
24          System.out.println("1.0/3.0*3.0 = " + (1.0 / 3.0) * 3.0);
25          System.out.println("max(23, 102) = " + Math.max(23, 102));
26          System.out.println("min(23, 102) = " + Math.min(23, 102));
27      }
28  }
```

Program Output **MathExamples.java**

```
Random Number: 0.5697285290062615
e =  2.718281828459045
pi = 3.141592653589793
floor(27.8) = 27.0
floor(17.0) = 17.0
floor(-7.6) = -8.0
sqrt(2) = 1.4142135623730951
sqrt(16) = 4.0
1/3 = 0
4/3 = 1
1.0/3.0 = 0.3333333333333333
1.0/3 = 0.3333333333333333
1/3.0 = 0.3333333333333333
1.0/3.0*3.0 = 1.0
max(23, 102) = 102
min(23, 102) = 23
```

We will not go over all of these but mention a couple. The line,

```
double num1 = Math.random();
```

creates a storage location names num1 that is for a double. It then assigns the result of `Math.random()` to that storage location. The function `Math.random()` produces a pseudo-random decimal number (a double) in the range $[0, 1)$.

From that output, you can see how to get the constant values of e and π, the floor function (i.e. the greatest integer less than or equal to), square roots, and so on. In previous examples we also saw the power function (pow) from the Math class. One portion of the output that we should take a closer look at is the following.

```
1/3 = 0
4/3 = 1
1.0/3.0 = 0.3333333333333333
1.0/3 = 0.3333333333333333
1/3.0 = 0.3333333333333333
```

There are times when you might put in a line of code and be expecting the computer to evaluate it in a certain way and you guess wrong. Note that the first line shows that in Java, `1/3` has the value 0 and not the value of 0.3333333333333333.

Also note that on line 3, Java does evaluate `1.0/3.0` as 0.3333333333333333 like we would expect. So why is there a difference and what is that difference. In both cases we are doing division, and we use the same symbol `/`. What might be surprising is that we are really doing two different divisions and it does not depend on the operator `/` but rather the values we are operating on. In the second case, we are operating on 1.0 and 3.0, both of which are doubles. So the division is a division of decimal numbers and we get the decimal approximation to $\frac{1}{3}$ as we would expect. In the first case, we are operating on 1 and 3, both integers, so we are doing integer division and what gets returned is the integer portion of the division, 0 in this case. This is also why `4/3` evaluates to 1. This is called truncation. Now if you go back to the sphere properties example from the previous chapter you will see why,

```
double volume = 4 / 3 * Math.PI * Math.pow(radius, 3);
```

gives you the wrong answer for the volume. The `4 / 3` at the beginning evaluated to 1 and not 1.3333333333333333 as we would need. We could fix this in a number of ways but simply putting a .0 after the 4 and 3 is the easiest. So the line should look like,

```
double volume = 4.0 / 3.0 * Math.PI * Math.pow(radius, 3);
```

Now let's look at the final two lines.

```
1.0/3 = 0.3333333333333333
1/3.0 = 0.3333333333333333
```

Note that we are dividing again and in this case one of the numbers is an integer and one is a decimal number. So there seems to be a choice on doing the division with integers or with decimals. Since a decimal holds "more information" Java promotes the integers to doubles before it does the division. So when Java evaluates `1.0/3` it first promotes the 3 to 3.0 and then evaluates `1.0/3.0`. Similarly, the `1/3.0` is changed to `1.0/3.0` before evaluation.

3.5 Special Integer Functions Example

Although we have titles this special integer functions, these operations still work on floats and doubles. We can add one to the value of a variable by using `++` either before or after the variable name. Similarly, we can subtract one from a variable by using `--` either before or after the variable name. So if `i` is an integer type and say it is storing the value 5, then `i++` or `++i` updates the value stored by `i` to 6 and `i--` or `--i` would update the value stored by `i` to 4.

There are also short-cut operation/assignment statements. For example, `i += 5` adds 5 to the value of `i`, `i -= 20` subtracts 20 from the value of `i`. In a similar

fashion, `i *= 2` multiplies the value of `i` by two and stores the new value in `i` and `i /= 3` divides the value of `i` by three and stores the new value in `i`. Since `i` is an integer, `i /= 3` is integer division and if `f` is a float or double then `f /= 3` is decimal division.

There is a technical difference between `i++` and `++i`. The statement `i++` increments `i` after the value of `i` is used in whatever the expression was it is in. On the other hand, `++i` increments `i` before the value of `i` is used in whatever the expression was it is in. So for example, say `i` has the value of 5, then the expression `t = 3 * i++` will set the value of `t` to 15 and `i` to 6. The expression `t = 3 * ++i` set the value of `t` to 18 and `i` to 6. So in `t = 3 * i++` the current value of `i` was used to evaluate `t` and then after that the value of `i` was incremented. In `t = 3 * ++i` the value of `i` was incremented first and then `t` was evaluated.

```
1  public class SpecialIntegerFunctions {
2
3      /*-
4       * SpecialIntegerFunctions: Incrementing, decrementing, and
5       * operation-assignment operators.  Note that these operaitons
6       * are also valid on other data types, such as long, float, and
7       * double.
8       * Author:  Don Spickler
9       * Date: 2/2/2011
10      */
11
12     public static void main(String[] args) {
13         int int1 = 7;
14         long long1 = 700;
15         double db1 = 3.24;
16
17         System.out.println("int1 =  " + int1);
18         System.out.println("long1 = " + long1);
19
20         int1++;
21         long1--;
22
23         System.out.println("int1 =  " + int1);
24         System.out.println("long1 = " + long1);
25
26         int1 -= 5;
27         long1 += 10;
28
29         System.out.println("int1 =  " + int1);
30         System.out.println("long1 = " + long1);
31
32         System.out.println("int1 =  " + int1++);
33         System.out.println("int1 =  " + int1);
34
35         System.out.println("int1 =  " + ++int1);
36         System.out.println("int1 =  " + int1);
37
38         System.out.println("int1 =  " + --int1);
39         System.out.println("int1 =  " + int1);
40
41         System.out.println("4*int1++ =  " + 4 * int1++);
42         System.out.println("4*int1 =  " + 4 * int1);
43
44         long1 *= 20;
45         System.out.println("long1 = " + long1);
```

```
46
47          long1 /= 5;
48          System.out.println("long1 = " + long1);
49
50          System.out.println("db1 = " + db1);
51
52          db1++;
53          System.out.println("db1 = " + db1);
54
55          --db1;
56          System.out.println("db1 = " + db1);
57
58          db1 *= 5;
59          System.out.println("db1 = " + db1);
60      }
61  }
```

Program Output **SpecialIntegerFunctions.java**

```
int1 =  7
long1 = 700
int1 =  8
long1 = 699
int1 =  3
long1 = 709
int1 =  3
int1 =  4
int1 =  5
int1 =  5
int1 =  4
int1 =  4
4*int1++ =  16
4*int1 =  20
long1 = 14180
long1 = 2836
db1 = 3.24
db1 = 4.24
db1 = 3.24
db1 = 16.200000000000003
```

3.6 Overloading and Underloading Example

Overloading is when you store a number in a variable that is larger than the maximum value the variable type can hold. Underloading is when you store a number in a variable that is smaller than the minimum value the variable type can hold. If we look back at the results of the DataTypes001.java program we can see what these bounds are.

Program Output **DataTypes001.java**

```
byte minimum: -128
byte maximum: 127
short minimum: -32768
short maximum: 32767
integer minimum: -2147483648
integer maximum: 2147483647
```

```
long minimum: -9223372036854775808
long maximum: 9223372036854775807
float minimum: 1.4E-45
float maximum: 3.4028235E38
double minimum: 4.9E-324
double maximum: 1.7976931348623157E308
```

So if we tried to store the number 150 in a byte variable we would be overloading the byte and if we stored -146 in a byte we would be underloading it. For decimal variables of type float or double, overloading is when we are outside the maximum range and underloading is when we have a value that is close to 0. So if we tried to store $5 \cdot 10^{500}$ or $-5 \cdot 10^{500}$ in a double we would be overloading it and if we tried to store $5 \cdot 10^{-500}$ we would be underloading it.

```
1  public class DataTypes002 {
2
3      /*-
4       * DataTypes002 prints out the result of overloading and
5       * underloading some native data types.
6       * Author:  Don Spickler
7       * Date: 2/3/2011
8       */
9
10     public static void main(String[] args) {
11         int c = 299792458;
12         System.out.println("c = " + c);
13         System.out.println("c*c = " + c * c);
14
15         long lc = 299792458;
16         System.out.println("lc = " + lc);
17         System.out.println("lc*lc = " + lc * lc);
18         System.out.println();
19
20         // For loop simply runs through the values 121, 122, ..., 130.
21         // b++ is the same a b = b + 1;
22         byte b = 120;
23         for (int i = 0; i < 10; i++) {
24             b++;
25             System.out.println(b);
26         }
27
28         System.out.println();
29
30         // For loop simply runs through the values -121, -122, ..., -130.
31         // b++ is the same a b = b + 1;
32         b = -120;
33         for (int i = 0; i < 10; i++) {
34             b--;
35             System.out.println(b);
36         }
37
38         System.out.println();
39
40         // Note the f at the end of the number, forces it to be a float.
41         float fl = 100000000000000000000000000000f;
42
43         // For loop runs through the values 10^31, 10^32, ..., 10^40.
44         // fl *= 10 is the same as fl = fl * 10
45         for (int i = 0; i < 10; i++) {
46             fl *= 10;
47             System.out.println(fl);
```

```
48          }
49
50          System.out.println();
51
52          // Power function, this sets db to 10^300.
53          double db = Math.pow(10, 300);
54          for (int i = 0; i < 10; i++) {
55              db *= 10;
56              System.out.println(db);
57          }
58
59          System.out.println();
60
61          // Power function, this sets db to 10^300.
62          db = -Math.pow(10, 300);
63          for (int i = 0; i < 10; i++) {
64              db *= 10;
65              System.out.println(db);
66          }
67
68          System.out.println();
69
70          db = Math.pow(10, -320);
71          for (int i = 0; i < 5; i++) {
72              db /= 10;
73              System.out.println(db);
74          }
75      }
76  }
```

Program Output — DataTypes002.java

```
c = 299792458
c*c = -1394772636
lc = 299792458
lc*lc = 89875517873681764

121
122
123
124
125
126
127
-128
-127
-126

-121
-122
-123
-124
-125
-126
-127
-128
127
126

1.0E31
1.0E32
1.0000001E33
1.00000004E34
```

```
1.0E35
1.00000004E36
1.00000006E37
1.0000001E38
Infinity
Infinity

1.0E301
1.0E302
1.0E303
1.0E304
1.0E305
9.999999999999999E305
9.999999999999999E306
9.999999999999998E307
Infinity
Infinity

-1.0E301
-1.0E302
-1.0E303
-1.0E304
-1.0E305
-9.999999999999999E305
-9.999999999999999E306
-9.999999999999998E307
-Infinity
-Infinity

1.0E-321
1.0E-322
1.0E-323
0.0
0.0
```

What we can see from the above example is that when you overload a byte the value cycles around to the smallest byte value and when you underload a byte the value cycles back to the largest value. The same is true for the other integer data types, short, int, and long. Overloading or underloading cycles the value, this is why `c*c = -1394772636`. The decimal data types are different. We see from the above example that overloading produces `Infinity` and underloading is rounded to 0.

3.7 Characters Example

This next example shows a few things that can be done with a character. A character is a single text character, such as a keyboard character. The character acts like an integer, you can use ++ and – with them. The correspondence between the number that a character is holding and the character that is displayed on the screen is vis the ASCII table. ASCII stands for American Standard Code for Information Interchange, it is simply a correspondence between integer values and characters. Note in the line,

```
char c = 't';
```

the character variable `c` is created and it is given the value of `t`. Note that the `t` is in single quotes. What `c` is really storing is the ASCII number associated with `t`. The first 32 ASCII values (0 – 31) are unprintable characters, hence the blank lines. ASCII value 32 is the space, 33 is `!`, and so on. Note that both the lowercase and uppercase cahracters are listed separately. So the lowercase letters have different ASCII values than do the uppercase characters. Keep this in mind, it will adversely affect the way that strings are sorted if we are not careful.

Another thing to notice in this example is the use of "escape sequences". This is when we use the backslash in front of special characters. Here are a few of them, `\'` is a single quote, `\"` is a double quote, `t` is a tab, and `\n` is a new line.

```
1  public class DataTypes003 {
2
3      /*-
4       * DataTypes003 shows the correspondence between characters
5       * and integers via the ASCII table.
6       * Author:  Don Spickler
7       * Date: 2/3/2011
8       */
9
10     public static void main(String[] args) {
11         char c = 't';
12         System.out.println(c);
13
14         for (int i = 0; i < 10; i++) {
15             c++;
16             System.out.println(c);
17         }
18
19         System.out.println();
20
21         System.out.println("ASCII Table");
22         c = 0;
23         for (int i = 1; i < 256; i++) {
24             c++;
25             System.out.print(c + "  ");
26             if (i % 10 == 0)
27                 System.out.println();
28         }
29
30         System.out.println();
31         System.out.println();
32
33         c = '"';
34         System.out.println(c);
35
36         c = '\"';
37         System.out.println(c);
38
39         c = '\'';
40         System.out.println(c);
41
42         System.out.println("ASCII Table Again");
43         c = 0;
44         for (int i = 1; i < 256; i++) {
45             c++;
46             System.out.print(c + "\t");
47             if (i % 10 == 0)
48                 System.out.print("\n");
```

```
49              }
50          }
51  }
```

Program Output **DataTypes003.java**

```
t
u
v
w
x
y
z
{
|
}
~

ASCII Table



     !  "  #  $  %  &  '  (
)  *  +  ,  -  .  /  0  1  2
3  4  5  6  7  8  9  :  ;  <
=  >  ?  @  A  B  C  D  E  F
G  H  I  J  K  L  M  N  O  P
Q  R  S  T  U  V  W  X  Y  Z
[  \  ]  ^  _  `  a  b  c  d
e  f  g  h  i  j  k  l  m  n
o  p  q  r  s  t  u  v  w  x
y  z  {  |  }  ~  ¨  ¯  .  ¸
□  _  □  □  □  □  □  □  □  □
□  □  □  □  □  □  □  □  □  □
□  □  □  □  □  □  □  □  □
i  ¢  £  ¤  ¥  ¦  §  ¨  ©  ª
«  ¬     ®  ¯  °  ±  ²  ³  ´
µ  ¶  ·  ¸  ¹  º  »  ¼  ½  ¾
¿  À  Á  Â  Ã  Ä  Å  Æ  Ç  È
É  Ê  Ë  Ì  Í  Î  Ï  Ð  Ñ  Ò
Ó  Ô  Õ  Ö  ×  Ø  Ù  Ú  Û  Ü
Ý  Þ  ß  à  á  â  ã  ä  å  æ
ç  è  é  ê  ë  ì  í  î  ï  ð
ñ  ò  ó  ô  õ  ö  ÷  ø  ù  ú
û  ü  ý  þ  ÿ

"
"
'
ASCII Table Again



                !     "     #     $     %     &     '     (
)     *     +     ,     -     .     /     0     1     2
3     4     5     6     7     8     9     :     ;     <
=     >     ?     @     A     B     C     D     E     F
G     H     I     J     K     L     M     N     O     P
Q     R     S     T     U     V     W     X     Y     Z
[     \     ]     ^     _     `     a     b     c     d
e     f     g     h     i     j     k     l     m     n
o     p     q     r     s     t     u     v     w     x
y     z     {     |     }     ~     ¨     ¯     .     ¸
□     _     □     □     □     □     □     □     □     □
□     ×     □     □     □     □     □     □     □     □
□     □     ÷     □     □     □     □     □     □     □
i     ¢     £     ¤     ¥     ¦     §     ¨     ©     ª
«     ¬           ®     ¯     °     ±     ²     ³     ´
µ     ¶     ·     ¸     ¹     º     »     ¼     ½     ¾
¿     À     Á     Â     Ã     Ä     Å     Æ     Ç     È
É     Ê     Ë     Ì     Í     Î     Ï     Ð     Ñ     Ò
Ó     Ô     Õ     Ö     ×     Ø     Ù     Ú     Û     Ü
Ý     Þ     ß     à     á     â     ã     ä     å     æ
ç     è     é     ê     ë     ì     í     î     ï     ð
ñ     ò     ó     ô     õ     ö     ÷     ø     ù     ú
û     ü     ý     þ     ÿ
```

3.8 Reference Types Example

A reference data type is not a native type like int and double. These are special classes that are built into Java. There are hundreds of these classes, and we will only see a few of them. Later on we will see how we can create our own classes. A reference data type is one that uses the new command to bring the type into existence. With a native data type we can simply use

```
int i = 7;
```

to create the integer and assign it to 7. For a reference data type we create a new instance of the type by using the new command along with the name of the data type, actually with one of the constructors of the data type. We will discuss constructors later in the notes.

```
Scanner keyboard = new Scanner(System.in);
```

So the above statement creates a new `Scanner` object, calls it `keyboard` and assigns it to the input data stream, that is, the computer keyboard. Also note in this example, we create another `Scanner` object called `keyboard2` but we do not create a new one, instead we assign it to null. This means that `keyboard2` is ready to hold a `Scanner` object but the object does not exist at this point. So if we try to use it we get a run-time error, i.e. an exception.

```
1  import java.util.Scanner;
2
3  public class DataTypes004 {
4
5      /*-
6       * DataTypes004 shows the use of reference types.
7       * Author:  Don Spickler
8       * Date: 2/3/2011
9       */
10
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("Input an integer: ");
14         int t = keyboard.nextInt();
15         keyboard.close();
16
17         Scanner keyboard2 = null;
18         System.out.print("Input an second integer: ");
19         int t2 = keyboard2.nextInt();
20         keyboard2.close();
21     }
22 }
```

Program Output **DataTypes004.java**

```
Input an integer: 5
Input an second integer: Exception in thread "main" java.lang.NullPointerException
        at DataTypes004.main(DataTypes004.java:14)
```

3.9 String Example

A string is simply a sequence of characters. Unlike the int and double types, the size of a string is limited only by the the amount of memory that the computer has available. Strings have many different functions and operations for manipulation, this example will show some of the basic operations that are commonly used. This is a long example so we will first discuss some of the lines of code and what they do. Remember that a string literal is any set of characters between double quotes. So

```
String str1 = "This is a test.";
```

creates the variable `str1` and sets it to the text *This is a test.* The `charAt` statement returns a single character at the position designated. So `str1.charAt(0)` returns the first character of the string. Each character in a string has an index, that is, a position in the string. The first character has index 0, the second character has index 1, and so on. So we could look at `str1` as,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T	h	i	s		i	s		a		t	e	s	t	.

The character at position 0 is `T`, the character at position 11 is `e`, and so on. The command,

```
str1.length()
```

returns the length of the string, 15 in this case. Two boolean string functions that are used frequently are

```
str1.equals(str2)
str1.equalsIgnoreCase(str2)
```

Both of these return a boolean result, either true or false, so they are good to use in conditional expressions that we will look at in a following chapter. The first returns true if the two strings contain the same characters, and false otherwise. This function is case-sensitive, so the string *Hi There* is different than the string *hi there* so the equals function would return false. The second is the same except that it is case-insensitive, so using equalsIgnoreCase with *Hi There* and *hi there* would result in a true.

The `valueOf` command turns a number into a string. So 3.14159 is a double but String.valueOf(3.14159) is a string with characters 3.14159, which is the same as "3.14159".

The `indexOf` command returns the position of the substring that is the first parameter. So `str1.indexOf("i")` returns the position of the first *i* in the string `str1`. If there is no *i* in the string `str1` the `indexOf` command returns −1. This

command has an optional second parameter, an integer, that specifies the starting position of the substring search. So `str1.indexOf("i", 3)` returns the position of the first i in the string `str1` on or after position 3. Again, if there is no i in the string `str1` on or after position 3 the `indexOf` command returns -1. The `lastIndexOf` command is the same except that the searching starts at the end of the string `str1`. Note that the substring that is being searched for need not be a single character.

The next several commands are useful when sorting a list of words, which we will look at later on when we talk about arrays and array lists. The

```
str1.compareTo(str2)
```

commands returns either a negative number, 0, or a positive number. It will return a negative number if str1 is less than str2, 0 if the two strings are equal, and a positive number if str1 is greater than str2. So the question is, what does it mean for one string to be less than or greater than another string? You might think that str1 is less than str2 if str1 would appear before str2 in the dictionary. This would be a good guess but unfortunately it is not entirely correct. What this command will do is compare the first two characters, if the first character of str1 is different than the first character of str2, then the `compareTo` function will return the difference between the ASCII numbers of the two letters, that is, the ASCII value of the first character of str1 minus the ASCII value of first character of str2. If the first characters are the same, and same case, the function moves onto the second character of each of the two words.

So if we run

```
str1 = "cat";
str2 = "Dog";
str1.compareTo(str2)
```

the last line will return the value of 31, since the ASCII value of "c" is 99 and the ASCII value of "D" is 68. We have reproduced a portion of the ASCII table and the corresponding values below.

Portion of the ASCII Table

#	Character	#	Character	#	Character
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f

#	Character	#	Character	#	Character
39	’	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		
64	@	96	‘		

The function `compareToIgnoreCase` does the same thing but it does not distinguish between the cases of the characters. It is equivalent to converting both strings to all uppercase and then applying the `compareTo` function.

The `concat` command concatenates the two given strings. So the statement,

```
str3 = str1.concat(str2);
```

sets str3 to the single string which is the characters of str1 followed by the characters of str2. Note that this is equivalent to the statement,

```
str3 = str1 + str2;
```

The commands `startsWith` and `endsWith` are two more boolean functions (returns true or false). They do what you would expect, `str1.startsWith("ta")` will return true if the first two letters of string str1 are *ta* and false otherwise.

The `isEmpty` function is also a boolean function that returns true if the string is the empty string and false if the string contains at least one character.

The functions `toLowerCase` and `toUpperCase` return the string that is all lowercase or all uppercase respectively of the string that called the function. One thing to be careful about is that these functions do not alter the string that is used in the call. So `str1.toLowerCase()` does not change str1. To convert str1 to all lowercase you would need to assign str1 to the result of the call. That is

```
str1 = str1.toLowerCase();
```

Java also has a built-in function for replace substrings with other strings. The command `replaceAll` will replace all occurrences of the first substring with the second substring. So

```
str1 = str1.replaceAll("test", "short");
```

will take the string str1, replace each occurrence of the substring *test* with the string *short*, and finally save the result back into str1. Just like the `toLowerCase` and `toUpperCase` functions, `str1.replaceAll("test", "short")` does not alter str1.

The `substring` command will return a substring of the calling string. The command

```
str1.substring(a, b);
```

returns the characters from position a up to but not including position b, that is, from position a up to position $b-1$.

The `trim` command removes all leading and trailing spaces from a string. So the code,

```
str1 = "  This is a trim test    ";
str1 = str1.trim();
```

will convert str1 to the string *This is a trim test*.

```
1  public class DataTypes005 {
2
3      /*-
4       * DataTypes005 shows some string manipulation functions.
5       * Author:  Don Spickler
6       * Date: 2/3/2011
7       */
8
9      public static void main(String[] args) {
10         String str1 = "This is a test.";
11
12         char ch1 = str1.charAt(0);
13         System.out.println(ch1);
14         System.out.println(str1.charAt(0));
15         System.out.println(str1.length());
```

```
16
17          System.out.println();
18
19          for (int i = 0; i < str1.length(); i++) {
20              System.out.println(str1.charAt(i));
21          }
22
23          System.out.println();
24
25          String str2 = "this is a test.";
26          System.out.println(str1.equals(str2));
27          System.out.println(str1.equalsIgnoreCase(str2));
28
29          System.out.println();
30
31          String str3 = String.valueOf(3.14159);
32          System.out.println(str3);
33
34          System.out.println();
35
36          System.out.println(str1.indexOf("i"));
37          System.out.println(str1.indexOf("i", 3));
38          System.out.println(str1.indexOf("i", 7));
39
40          System.out.println();
41
42          System.out.println(str1.lastIndexOf("i"));
43          System.out.println(str1.lastIndexOf("i", 3));
44          System.out.println(str1.lastIndexOf("i", 7));
45          System.out.println(str1.lastIndexOf("i", 1));
46
47          System.out.println();
48
49          System.out.println(str1.indexOf("is"));
50          System.out.println(str1.indexOf("is", 3));
51          System.out.println(str1.indexOf("that"));
52
53          System.out.println();
54          str1 = "cat";
55          str2 = "dog";
56
57          System.out.println(str1.compareTo(str2));
58          System.out.println(str1.compareTo(str1));
59          System.out.println(str2.compareTo(str1));
60
61          System.out.println();
62          str1 = "cat";
63          str2 = "Dog";
64
65          System.out.println(str1.compareTo(str2));
66          System.out.println(str1.compareTo(str1));
67          System.out.println(str2.compareTo(str1));
68
69          System.out.println();
70
71          System.out.println((int)('c') + "  " + (int)('D'));
72
73          System.out.println();
74
75          System.out.println(str1.compareToIgnoreCase(str2));
76          System.out.println(str1.compareToIgnoreCase(str1));
77          System.out.println(str2.compareToIgnoreCase(str1));
78
79          System.out.println();
```

```
80
81          str1 = str1.concat(str2);
82          System.out.println(str1);
83
84          str1 = "cat";
85          str2 = "Dog";
86
87          str1 = str2 + str1;
88          System.out.println(str1);
89
90          System.out.println();
91
92          System.out.println(str1.endsWith("at"));
93          System.out.println(str1.endsWith("ta"));
94
95          System.out.println();
96
97          System.out.println(str1.startsWith("Dog"));
98          System.out.println(str1.startsWith("dog"));
99
100         System.out.println();
101         System.out.println(str1.isEmpty());
102
103         System.out.println();
104         System.out.println(str1);
105         System.out.println(str1.toLowerCase());
106         System.out.println(str1.toUpperCase());
107         System.out.println(str1);
108
109         str1 = str1.toUpperCase();
110         System.out.println(str1);
111
112         System.out.println();
113
114         str1 = "This is a test string for testing the replace command.";
115         System.out.println(str1);
116         str1.replaceAll("test", "short");
117         System.out.println(str1.replaceAll("test", "short"));
118         System.out.println(str1);
119         System.out.println(str1.replaceFirst("test", "short"));
120         System.out.println(str1);
121
122         System.out.println();
123         System.out.println(str1.substring(5, 7));
124         System.out.println(str1.substring(15, 20));
125         System.out.println(str1.substring(15, 21));
126         System.out.println(str1.substring(15));
127         System.out.println(str1);
128
129         System.out.println();
130         str1 = "  This is a trim test    ";
131         System.out.println(str1.trim() + "*****");
132     }
133 }
```

Program Output — DataTypes005.java

```
T
T
15

T
h
```

```
i
s

i
s

a

t
e
s
t
.

false
true

3.14159

2
5
-1

5
2
5
-1

2
5
-1

-1
0
1

31
0
-31

99  68

-1
0
1

catDog
Dogcat

true
false

true
false

false

Dogcat
dogcat
DOGCAT
Dogcat
DOGCAT

This is a test string for testing the replace command.
```

```
This is a short string for shorting the replace command.
This is a test string for testing the replace command.
This is a short string for testing the replace command.
This is a test string for testing the replace command.

is
strin
string
string for testing the replace command.
This is a test string for testing the replace command.

This is a trim test*****
```

3.10 String Type Conversion Example

From time to time you may want to convert data from one type to another. One instance is if you have a string that represents a number and you want to convert the string contents to a number. For example, the line of code,

```
String str1 = "10.3";
```

sets str1 to the string of the characters 10.3, not the number 10.3. So as it is here, you could not do any arithmetic with this number since it is a string and not a number. Java has a way to easily convert this string to a number,

```
double d = Double.parseDouble(str1);
```

will convert the string str1 to its numeric value and store it in the variable d. If str1 contains a string that is a valid double then the function works fine, but if str1 does not contain a valid number this command will result in a run-time error. Similarly, if str2 contains a valid integer, then the command,

```
int i = Integer.parseInt(str2);
```

will set i to that value.

```
1  public class DataTypes006 {
2
3      /*-
4       * DataTypes006 shows functions to convert strings to numeric variables.
5       * Author:  Don Spickler
6       * Date: 2/3/2011
7       */
8
9      public static void main(String[] args) {
10         String str1 = "10.3";
11         String str2 = "123";
12
13         // Concatenation
14         System.out.println(str1+str2);
15
16         // Double and Integer are wrapper classes for the native data types
17         // double and int.
```

```
18          double d = Double.parseDouble(str1);
19          int i = Integer.parseInt(str2);
20
21          // Addition
22          System.out.println(d+i);
23          System.out.println();
24      }
25  }
```

Program Output **DataTypes006.java**

```
10.3123
133.3
```

3.11 Formatted Output Example

From time to time you will want to have a little more control over the output of a program than the simple `print` and `println` commands will give. For example, say you are writing a program that displays amounts of money. You would not want the output to have 10 decimal places since you only need two. In this case, having an output of $123.4567890 would be silly. You would also not want an output of $123.4, having $123.40 would look much better.

Java has a nice function for formatted output, the `printf` function. The syntax for the `printf` function is a string with "inserts" followed by values that will be inserted. An insert is a special character sequence that starts with % and then a type and format designation.

Insert	Type
%b	boolean
%d	integer or long
%f	float or double
%e	float or double using scientific notation
%c	character
%s	string

Each of the inserts has optional formatting styles,

Insert	Boolean Styles
%b	Inserts boolean value.
%#b	Inserts boolean value using # spaces.

Insert	Integer Styles
%d	Inserts integer value.
%#d	Inserts integer value using # spaces.

Insert	Character Styles
%c	Inserts character.
%#c	Inserts character using # spaces.

Insert	Floating Point Styles
%f	Inserts float or double.
%#.#f	Uses first # total spaces and second # decimal places.
%#f	Uses # total spaces.
%.#f	Uses # decimal places and as many total spaces as needed.

Insert	Scientific Notation Styles
%e	Inserts float or double.
%#.#e	Uses first # total spaces and second # decimal places.
%#e	Uses # total spaces.
%.#e	Uses # decimal places and as many total spaces as needed.

```
1  public class DataTypes007 {
2
3      /*-
4       * DataTypes007 shows functions to do formatted printing.
5       * Author:  Don Spickler
6       * Date: 2/3/2011
7       */
8
9      public static void main(String[] args) {
10         System.out.printf("This statement is %b \n", false);
11         System.out.printf("The numbers are %d, %d, and %d \n", 17, 21, 100);
12
13         long c = 299792458;
14         System.out.printf("The speed of light is %d m/sec \n", c);
15         System.out.printf("The speed of light is %15d m/sec \n", c);
16         System.out.printf("The speed of light is %f m/sec \n", 1.0*c);
17         System.out.printf("The speed of light is %e m/sec \n", 1.0*c);
18
19         System.out.println();
20         System.out.printf("Pi = %10.2f \n", Math.PI);
21         System.out.printf("Pi = %10.4f \n", Math.PI);
22         System.out.printf("Pi = %.2f \n", Math.PI);
23         System.out.printf("Pi = %.7f \n", Math.PI);
24         System.out.printf("Pi = %.20f \n", Math.PI);
25         System.out.printf("Pi = %20f \n", Math.PI);
26
27         System.out.println();
28         System.out.printf("Pi = %20e \n", Math.PI);
29         System.out.printf("Pi = %20.10e \n", Math.PI);
30         System.out.printf("Pi = %.20e \n", Math.PI);
31
32         char ch1 = 'W';
33         System.out.println();
34         System.out.printf("This is a character: %c \n", 'A');
35         System.out.printf("This is a character: %c \n", ch1);
36         System.out.printf("This is a character: %c \n", 125);
```

```
37
38          String str1 = "a string";
39          System.out.println();
40          System.out.printf("This is a string: %s \n", "Here we go.");
41          System.out.printf("This is a string: %s \n", str1);
42      }
43  }
```

Program Output **DataTypes007.java**

```
This statement is false
The numbers are 17, 21, and 100
The speed of light is 299792458 m/sec
The speed of light is       299792458 m/sec
The speed of light is 299792458.000000 m/sec
The speed of light is 2.997925e+08 m/sec

Pi =       3.14
Pi =     3.1416
Pi = 3.14
Pi = 3.1415927
Pi = 3.141592653589793000000
Pi =             3.141593

Pi =         3.141593e+00
Pi =     3.1415926536e+00
Pi = 3.14159265358979300000e+00

This is a character: A
This is a character: W
This is a character: }

This is a string: Here we go.
This is a string: a string
```

3.12 Random Number Example

In many types of programs, specifically simulations and games, one needs a way to produce random numbers. For example, if you are writing a game that involves rolling dice or shuffling a deck of cards you need to "randomize" the output of these operations.

Now a computer cannot produce a truly random sequence of numbers. In reality, the computer produces what we call pseudo-random numbers, more specifically, a pseudo-random sequence of numbers. A pseudo-random sequence of numbers is a sequence of numbers that "act" like a random sequence of number. So if we produce a sequence of numbers between 1 and 10 we would expect the number 4 to show up approximately one tenth of the time, as we would the other numbers. We would also expect that the pair 27 show up about one one-hundredth of the time, the triple 841 about one one-thousandth of the time, and so on. There are other criteria for a good pseudo-random sequence of numbers, but this gives you the idea.

The way a computer generates a pseudo-random sequence of numbers is it starts with a given number, called the seed. The seed value can sometimes be set by the programmer, usually we use the time as a seed. Different programming systems have different ways to get the time from the computer. Some can return the number of seconds that have elapsed since January 1, 1970. Java has a nice function `System.nanoTime()` which returns the number of nano-seconds that have elapsed since the computer was booted. When I ran this function while I was writing this paragraph I got 1746917379654, then after waiting a couple minutes I got 1970816803565. Then I rebooted the computer, went immediately into Eclipse and ran the program again and got 36736535102.

Once the seed is set the rest of the pseudo-random sequence is generated from a simple equation. Hence the pseudo-random sequence is completely determined by the seed. So if you use the same seed twice you will get the same pseudo-random sequence, which is not very random. This is why using a nano-second timer to set the seed is a good idea. It is extremely unlikely that any two runs of a program will start at the exact same time, down to the nano-second.

The standard way to produce a pseudo-random sequence is using a linear congruential algorithm. In a linear congruential algorithm we choose two numbers that stay fixed, one is a multiplier m and the other is an adder a, in addition we usually have a modulus n. The value of the modulus n is usually the largest possible value of an int or a long. The seed is set to the first number in the sequence, which we will call x_0, and the rest of the sequence, x_1, x_2, x_3, x_4, ... is generated by

$$x_{i+1} = x_i \cdot m + a \mod n$$

So it is a linear feedback function with a modulus, hence the name linear congruential. It is surprising that such a simple, and completely deterministic, function can produce a sequence of values that seem random. We should point out that how good this sequence is depends on the choices of m and a. For example, if we let $m = 1$ and $a = 1$ then our sequence is not very random at all. With a seed of 1, our sequence would be 1, 2, 3, 4, ... A detailed discussion of good choices for m and a is a bit beyond the scope of these notes. The interested reader should consult Don Knuth's second volume of *The Art of Computer Programming*, *Seminumerical Algorithms*.

The linear congruential algorithm is not the only random number generator algorithm, there are many others, some better and some worse than the linear congruential algorithm. It is, however, the most common one used. When you invoke a random number generator that is built into a programming language, a calculator, or an app on your phone, it is almost certainly using a linear congruential algorithm. The linear congruential algorithm is fine for most uses, such as games, small simulations, or generating fractal images. For applications that require a high level of security, such as many cryptographic application, the linear congruential algorithm is not sufficiently secure and another method is required.

In Java, there are several ways to create a pseudo-random sequence of numbers, the easiest is with the `Math.random()` function. The `Math.random()` function returns a double in the range $[0, 1)$. That is, it returns a decimal number greater than or equal to 0 and strictly less than 1. In many cases we will want random numbers in a different range and we might not want decimal numbers, we may want integers. Using the fact that `Math.random()` returns a number in the range $[0, 1)$ allows us to manipulate the result into what we need for our application.

For example, say we needed random numbers in the range $[0, 10)$ we could simply use,

```
10 * Math.random();
```

If we needed random numbers in the range $[10, 15)$ we could use,

```
5 * Math.random() + 10;
```

If we wanted random integers, such as in rolling a die, we can cast the double to an integer. We will look at casting in more detail in the next example. If we put `(int)` in front of a double or float it will extract the portion to the left of the decimal point. So `(int)(3.14159)` is the integer 3. In our example below, we use the command,

```
(int)(Math.random()*7) + 5
```

This will produce a pseudo-random sequence of numbers between 5 and 11. Let's look at this in more detail. Say that `Math.random()` returns the value 0.4892883772. The above statement will first multiply by 7 giving, 3.4250186404. The integer cast will extract the 3 and then the addition of 5 will produce the number 8. Similarly, if `Math.random()` returns 0.3812994827, 7 times it is, 2.6690963789, casting to 2 and adding 5 to get the final answer of 7. More specifically, `Math.random()` returns something in the range $[0, 1)$, multiplying by 7 gives something in the range $[0, 7)$, casting gives an integer in the range $[0, 6]$, that is, 0, 1, 2, 3, 4, 5, 6. Finally, the addition of 5 produces something in, 5, 6, 7, 8, 9, 10, 11. So in general, the multiplier, 7 in this example, is the number of possible integers you want and the adder, 5 in this example, is the starting point. Hence to simulate the roll of a single die we could use,

```
(int)(Math.random()*6) + 1
```

```
1  public class DataTypes008 {
2
3      /*-
4       * DataTypes008 shows one way to produce pseudo-random numbers.
5       * Author:  Don Spickler
6       * Date: 2/3/2011
7       */
8
9      public static void main(String[] args) {
10         for (int i = 0; i < 10; i++)
11             System.out.println(Math.random());
```

```
12
13          for (int i = 0; i < 10; i++)
14              System.out.println((int)(Math.random()*7) + 5);
15      }
16  }
```

Program Output — DataTypes008.java Run #1

```
0.49813330422270496
0.3122113161822261
0.23771186454864535
0.5832273653745754
0.7280989383011853
0.2647490361800907
0.29949669826336545
0.274280648747089
0.9324458505657957
0.11971792263381476
6
11
5
8
7
8
7
11
9
7
```

Program Output — DataTypes008.java Run #2

```
0.2345985535349796
0.9230516748679632
0.22552729717924114
0.1736370260526826
0.5618738063630802
0.37155940913307206
0.44976503133025736
0.6030770480140647
0.7240120865909763
0.050443003750694215
8
10
7
6
6
9
5
10
10
8
```

Program Output — DataTypes008.java Run #3

```
0.9069466670574435
0.7462552862521772
0.38727092815220565
0.2829940970187773
0.2495895672982984
0.1688885262023272
0.20229260111376013
```

```
0.0015198166483957332
0.8918788115445913
0.35170090409024357
11
5
8
6
6
7
9
6
9
6
```

3.13 Casting Example

Whenever a data type changes we call that casting. As we saw a few examples ago, Java will do some casting automatically. For example you can assign an integer data type to a double data type and Java will automatically convert the integer to a double. On the other hand Java will not do it the other way around. That is, if you try to assign a double to an integer it will give you an error. As we saw in the last example, we can force Java to do that conversion by putting `(int)` in front of the double you would like to convert. There are many other automatic and forced casts you can do in Java and we will see a few of these as the examples progress.

```
1  /*-
2   * Casting shows one way to produce pseudo-random numbers.
3   * Author:  Don Spickler
4   * Date: 2/3/2011
5   */
6
7  public class Casting {
8      public static void main(String[] args) {
9          double dif1 = 14.36;
10         double dif2 = 14.96;
11
12         int x = (int) dif1;
13         System.out.println(x);
14
15         x = (int) dif2;
16         System.out.println(x);
17
18         System.out.println(dif1);
19         System.out.println(dif2);
20
21         x = (int) (dif1 + 0.5);
22         System.out.println(x);
23
24         x = (int) (dif2 + 0.5);
25         System.out.println(x);
26     }
27 }
```

Program Output **Casting.java**

```
14
14
14.36
14.96
14
15
```

Chapter 4

Decisions

4.1 Introduction

Computers can do several things really well. They can do calculations very quickly, they can do something over and over again without getting bored, they can store a large amount of information, and they can make decisions as long as you phrase the question carefully. For a computer to be able to make a decision you need to phrase your question so that there is only a yes or no answer, that is, either the question has a true answer or a false answer.

For example, a computer can easily decide if the variable x in your program currently is storing the value 4. This is a yes or no question, does x contain the value 4? Yes it does or no it does not. Similarly, the question does x contain a larger number than y? Yes it does or no it does not. On the other hand, asking the computer to tell you how many double letters are in "committee" does not have a yes or no answer. Now you can program the computer to answer the last question but it takes more than a single question and the questions must be formulated into yes and no form.

In Java, most decisions are made using the `if` statement. The general syntax for the `if` statement is,

```
if ( <Boolean Expression> ) {
    // Code Block to do if the Boolean Expression is true.
}
```

So it is the reserved word `if` followed by a boolean expression that is enclosed in parentheses. Following this is a block of code. Recall that blocks of code are enclosed in curly brackets.

A boolean expression is any expression that evaluates to either true or false. In

Java there are numerous types of boolean expressions and functions, we saw a few in the data types chapter and we will see others as these notes progress. The ones you will use the most are those that simply compare two numeric values. Here is a list of these and their syntax. In the following chart we assume that x and y are numeric variables.

Syntax	Meaning
`x == y`	True if $x = y$, false otherwise.
`x < y`	True if $x < y$, false otherwise.
`x <= y`	True if $x \leq y$, false otherwise.
`x > y`	True if $x > y$, false otherwise.
`x >= y`	True if $x \geq y$, false otherwise.
`x != y`	True if $x \neq y$, false otherwise.

If the boolean expressions that follows the `if` statement is true then the block of code is executed and if the statement is false the program will skip over the block of code and continue with the rest of the program. Decision structures, like the `if` statement, are sometimes called flow control since they control the flow of the program, that is, which lines of code are executed and which ones are not.

We can include an `else` part after the block of code that will execute if the boolean expression is false. So with the following syntax, if the boolean expression is true the first block of code is executed and the block after the `else` statement is skipped. If the boolean expression is false then the first block is skipped and the block after the `else` statement is executed. This allows the programmer to do one thing if the answer to the question is yes and another thing if the answer is no.

```
if ( <Boolean Expression> ) {
    // Code block to do if the Boolean Expression is true.
} else {
    // Code block to do if the Boolean Expression is false.
}
```

Frequently, if the first boolean expression is false the programmer may need to ask another question before deciding which block of code to run. In this case there is an `else if` statement. So in place of the `else` we use `else if` with a second boolean expression in parentheses. So if the first boolean expression is true the first block of code is executed and the rest is skipped. If the first boolean expression is false and the second boolean expression is true the second block is run and the rest skipped. Finally, if both of the boolean expressions are false the third block of code is run and the rest are skipped. One thing to note is that the final `else` is not required.

```
if ( <Boolean Expression #1> ) {
    // Code block to do if expression  #1 is true.
```

```
} else if ( <Boolean Expression #2> ) {
    // Code block to do if expression  #2 is true.
} else {
    // Code block to do if both were false.
}
```

One thing to note is that the final `else` is not required. If we do not have this block and the boolean expressions are false then there are no lines of code that are executed.

```
if ( <Boolean Expression #1> ) {
    // Code block to do if expression  #1 is true.
} else if ( <Boolean Expression #2> ) {
    // Code block to do if expression  #2 is true.
}
```

These `else if` statements can be strung along as far as you would like or need.

```
if ( <Boolean Expression #1> ) {
    // Code block to do if expression  #1 is true.
} else if ( <Boolean Expression #2> ) {
    // Code block to do if expression  #2 is true.
} else if ( <Boolean Expression #3> ) {
    // Code block to do if expression  #3 is true.
} else if ( <Boolean Expression #4> ) {
    // Code block to do if expression  #4 is true.
} else {
    // Code block to do if all were false.
}
```

Inside these blocks of code that are executed you can have as many statements as you would like. You can also have more conditional statements in these blocks. When you have conditional statements inside conditional statements we say that the conditional statements are nested.

Boolean expressions can be more complex than simply checking if two values are equal. In many cases you will want to ask more complicated true or false questions. In Java, and most other languages, you can create compound boolean expressions by using logical connectors between simpler statements.

Syntax	Meaning
`&&`	AND — True if both boolean expressions are true, false otherwise.
`||`	OR — True if either of boolean expressions are true, false otherwise.
`!`	NOT — True if the boolean expression was false, false otherwise.

For example, the expression,

```
(x <= y) && (y != z)
```

is true if both $x \leq y$ and $y \neq z$. The expression,

```
(x <= y) || (y != z)
```

is true if either $x \leq y$ or $y \neq z$, or both. The expression,

```
!(x <= y)
```

is true if $x > y$, so it is equivalent to the boolean expression,

```
x > y
```

4.2 Basic if Statement Example

```
1  import java.util.Scanner;
2
3  /*-
4   * Conditional Example #1
5   * Basic if statement.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ConditionalExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("How many eggs do you have? ");
14         int numEggs = keyboard.nextInt();
15         keyboard.close();
16
17         if (numEggs == 12) {
18             System.out.println("You have a dozen eggs.");
19         }
20     }
21 }
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 12
You have a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 5
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? -12
```

4.3 Basic if-else Statement Example

```
1  import java.util.Scanner;
2
3  /*-
4   * Conditional Example #2
5   * Basic if-else statement
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ConditionalExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("How many eggs do you have? ");
14         int numEggs = keyboard.nextInt();
15         keyboard.close();
16
17         if (numEggs == 12) {
18             System.out.println("You have a dozen eggs.");
19         } else {
20             System.out.println("You do not have a dozen eggs.");
21         }
22     }
23 }
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 12
You have a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 5
You do not have a dozen eggs.
```

4.4 Basic if-else if Statement Example

```
1  import java.util.Scanner;
2
3  /*-
4   * Conditional Example #3
5   * Basic if-else if statement.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ConditionalExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("How many eggs do you have? ");
14         int numEggs = keyboard.nextInt();
15         keyboard.close();
16
17         if (numEggs == 12) {
```

```
18              System.out.println("You have a dozen eggs.");
19          } else if (numEggs < 12) {
20              System.out.println("You have fewer than a dozen eggs.");
21          } else {
22              System.out.println("You have more than a dozen eggs.");
23          }
24      }
25  }
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 12
You have a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 5
You have fewer than a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 15
You have more than a dozen eggs.
```

4.5 Multiple else if Blocks Example

```
1  import java.util.Scanner;
2
3  /*-
4   * Conditional Example #4
5   * Multiple else if blocks.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ConditionalExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("How many eggs do you have? ");
14         int numEggs = keyboard.nextInt();
15         keyboard.close();
16
17         if (numEggs == 12) {
18             System.out.println("You have a dozen eggs.");
19         } else if (numEggs == 2) {
20             System.out.println("You have a couple eggs.");
21         } else if (numEggs <= 7) {
22             System.out.println("You have a few eggs.");
23         } else if (numEggs < 12) {
24             System.out.println("You have several eggs.");
25         } else {
26             System.out.println("You have more than a dozen eggs.");
27         }
28     }
29 }
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 12
You have a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 2
You have a couple eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 3
You have a few eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 8
You have several eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 11
You have several eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 15
You have more than a dozen eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 1
You have a few eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? -5
You have a few eggs.
```

Program Output **ConditionalExample.java**

```
How many eggs do you have? 0
You have a few eggs.
```

4.6 Menu Example

```
1 import java.util.Scanner;
2
3 /*-
4  * Conditional Example #5
5  * Menu Example
```

```
 * Author:  Don Spickler
 * Date: 2/6/2011
 */

public class ConditionalExample {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Please select from the following menu:");
        System.out.println("1. Rectangle Properties");
        System.out.println("2. Circle Properties");
        System.out.println();
        System.out.print("Selection: ");
        int menuOption = keyboard.nextInt();

        if (menuOption == 1) {
            System.out.print("Input the width of the rectangle: ");
            double width = keyboard.nextDouble();
            System.out.print("Input the height of the rectangle: ");
            double height = keyboard.nextDouble();
            double area = height * width;
            double perimeter = 2 * height + 2 * width;
            System.out.println("The area of the rectangle is " + area);
            System.out.println("The perimeter of the rectangle is " + perimeter);
        } else if (menuOption == 2) {
            System.out.print("Input the radius of the circle: ");
            double radius = keyboard.nextDouble();
            double area = Math.PI * Math.pow(radius, 2);
            double circumference = 2 * Math.PI * radius;
            System.out.println("The area of the circle is " + area);
            System.out.println("The circumference of the circle is " + circumference);
        } else {
            System.out.println("Invalid Menu Selection!");
        }

        keyboard.close();
    }
}
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 1
Input the width of the rectangle: 4
Input the height of the rectangle: 7
The area of the rectangle is 28.0
The perimeter of the rectangle is 22.0
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 2
Input the radius of the circle: 3
The area of the circle is 28.274333882308138
The circumference of the circle is 18.84955592153876
```

Program Output **ConditionalExample.java**

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 3
Invalid Menu Selection!
```

4.7 Nested if Statement Example

```
1  import java.util.Scanner;
2
3  /*-
4   * Conditional Example #6
5   * Nested if Statement Example
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ConditionalExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         System.out.println("Please select from the following menu:");
15         System.out.println("1. Rectangle Properties");
16         System.out.println("2. Circle Properties");
17         System.out.println("3. Eggs");
18         System.out.println();
19         System.out.print("Selection: ");
20         int menuOption = keyboard.nextInt();
21
22         if (menuOption == 1) {
23             System.out.print("Input the width of the rectangle: ");
24             double width = keyboard.nextDouble();
25             System.out.print("Input the height of the rectangle: ");
26             double height = keyboard.nextDouble();
27             double area = height * width;
28             double perimeter = 2 * height + 2 * width;
29             System.out.println("The area of the rectangle is " + area);
30             System.out.println("The perimeter of the rectangle is " + perimeter);
31         } else if (menuOption == 2) {
32             System.out.print("Input the radius of the circle: ");
33             double radius = keyboard.nextDouble();
34             double area = Math.PI * Math.pow(radius, 2);
35             double circumference = 2 * Math.PI * radius;
36             System.out.println("The area of the circle is " + area);
37             System.out.println("The circumference of the circle is " + circumference);
38         } else if (menuOption == 3) {
39             System.out.print("How many eggs do you have? ");
40             int numEggs = keyboard.nextInt();
41
42             if (numEggs == 12) {
43                 System.out.println("You have a dozen eggs.");
44             } else if (numEggs == 2) {
45                 System.out.println("You have a couple eggs.");
46             } else if (numEggs <= 7) {
47                 System.out.println("You have a few eggs.");
48             } else if (numEggs < 12) {
```

```
49                  System.out.println("You have several eggs.");
50              } else {
51                  System.out.println("You have more than a dozen eggs.");
52              }
53          } else {
54              System.out.println("Invalid Menu Selection!");
55          }
56
57          keyboard.close();
58      }
59  }
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Eggs

Selection: 1
Input the width of the rectangle: 4
Input the height of the rectangle: 5
The area of the rectangle is 20.0
The perimeter of the rectangle is 18.0
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Eggs

Selection: 2
Input the radius of the circle: 3
The area of the circle is 28.274333882308138
The circumference of the circle is 18.84955592153876
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Eggs

Selection: 3
How many eggs do you have? 5
You have a few eggs.
```

Program Output — ConditionalExample.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Eggs

Selection: 7
Invalid Menu Selection!
```

4.8 Boolean Expressions

This example shows values of boolean expressions and compound expressions. Although we are simply printing out the values of these boolean expressions here, we could use these expressions in any of the decision structures we discussed.

```
1  /*-
2   * Boolean Expressions Example
3   * Author:  Don Spickler
4   * Date: 2/6/2011
5   */
6
7  public class BooleanExpressions {
8      public static void main(String[] args) {
9          boolean b1 = true;
10         boolean b2 = false;
11         boolean b3 = true;
12         int x = 2;
13         int y = 3;
14         int z = 4;
15
16         System.out.println("b1 = " + b1);
17         System.out.println("b2 = " + b2);
18         System.out.println();
19         System.out.println("2 == 4 = " + (2 == 4));
20         System.out.println("x == 4 = " + (x == 4));
21         System.out.println("z == 4 = " + (z == 4));
22         System.out.println("x == z = " + (x == z));
23         System.out.println("x != z = " + (x != z));
24         System.out.println();
25         System.out.println("z < y = " + (z < y));
26         System.out.println("z > z = " + (z > z));
27         System.out.println("z >= z = " + (z >= z));
28         System.out.println();
29         System.out.println("!b1 = " + !b1);
30         System.out.println("!b2 = " + !b2);
31         System.out.println();
32         System.out.println("b1 && b2 = " + (b1 && b2));
33         System.out.println("b1 && b3 = " + (b1 && b3));
34         System.out.println("b1 && !b2 = " + (b1 && !b2));
35         System.out.println();
36         System.out.println("b1 || b2 = " + (b1 || b2));
37         System.out.println("b1 || b3 = " + (b1 || b3));
38         System.out.println("!b1 || b2 = " + (!b1 || b2));
39         System.out.println();
40         System.out.println("(x > 1) && (z < 7) = " + ((x > 1) && (z < 7)));
41         System.out.println("(x > 1) && (z < 4) = " + ((x > 1) && (z < 4)));
42         System.out.println("(x > 1) || (z < 4) = " + ((x > 1) || (z < 4)));
43         System.out.println();
44         System.out.println("(x % 2 == 0) && (y % 2 == 0) = " + ((x % 2 == 0) && (y % 2 ==
               0)));
45         System.out.println("(x % 2 == 0) && (z % 2 == 0) = " + ((x % 2 == 0) && (z % 2 ==
               0)));
46         System.out.println();
47         System.out.println("((x % 2 == 0) && (z % 2 == 0)) || b2 = " + (((x % 2 == 0) && (
               z % 2 == 0)) || b2));
48         System.out.println("((x % 2 == 0) && (z % 2 == 0)) && b2 = " + (((x % 2 == 0) && (
               z % 2 == 0)) && b2));
```

```
49          System.out.println("((x % 2 == 0) && (z % 2 == 0)) && b3 = " + (((x % 2 == 0) && (
                z % 2 == 0)) && b3));
50          System.out.println();
51          System.out.println("b1 ^ b3 = " + (b1 ^ b3));
52          System.out.println("b1 ^ b2 = " + (b1 ^ b2));
53          System.out.println();
54          System.out.println("true ^ true = " + (true ^ true));
55          System.out.println("true ^ false = " + (true ^ false));
56          System.out.println("false ^ true = " + (false ^ true));
57          System.out.println("false ^ false = " + (false ^ false));
58          System.out.println();
59          System.out.println("2 ^ 5 = " + (2 ^ 5));
60          System.out.println("23 ^ 15 = " + (23 ^ 15));
61          System.out.println("42 ^ 49 = " + (42 ^ 49));
62      }
63  }
```

Program Output — BooleanExpressions.java

```
b1 = true
b2 = false

2 == 4 = false
x == 4 = false
z == 4 = true
x == z = false
x != z = true

z < y = false
z > z = false
z >= z = true

!b1 = false
!b2 = true

b1 && b2 = false
b1 && b3 = true
b1 && !b2 = true

b1 || b2 = true
b1 || b3 = true
!b1 || b2 = false

(x > 1) && (z < 7) = true
(x > 1) && (z < 4) = false
(x > 1) || (z < 4) = true

(x % 2 == 0) && (y % 2 == 0) = false
(x % 2 == 0) && (z % 2 == 0) = true

((x % 2 == 0) && (z % 2 == 0)) || b2 = true
((x % 2 == 0) && (z % 2 == 0)) && b2 = false
((x % 2 == 0) && (z % 2 == 0)) && b3 = true

b1 ^ b3 = false
b1 ^ b2 = true

true ^ true = false
true ^ false = true
false ^ true = true
false ^ false = false

2 ^ 5 = 7
```

```
23 ^ 15 = 24
42 ^ 49 = 27
```

You will notice at the end we added a new boolean operation that we did not discuss in the introduction. The caret operator is the XOR, which stands for exclusive or. The XOR is true only when the two boolean expressions are different. So

Program Output **BooleanExpressions.java**

```
true ^ true = false
true ^ false = true
false ^ true = true
false ^ false = false
```

At the very end we show why you should not use the caret operator with numbers. Although most calculators and computer algebra systems use the caret symbol to represent exponentiation, in Java as well as most other programming languages, the caret symbol is not exponentiation. This is easily seen in the last several lines of output.

Program Output **BooleanExpressions.java**

```
2 ^ 5 = 7
23 ^ 15 = 24
42 ^ 49 = 27
```

So what is happening in these examples? Well, the caret operator is the XOR. If we write these numbers in binary, do an XOR with the corresponding bits and then convert back to base 10, we get these results. Let's take the last line as an example, `42 ^ 49 = 27`,

Decimal	Binary
42	101010
49	110001
42 ^ 49	011011
27	011011

4.9 Switch Statement Examples

Another type of decision structure is the switch statement, also called a case statement since it uses the keyword `case` in its syntax and essentially it asks what the value of a variable or expression is by asking what case it it in. The general syntax for the switch statement is the reserved word `switch` followed by a variable or expression in parentheses, followed by a block (curly bracket). In the block, there are segments that begin with the reserved word `case` followed by a value and then a colon. Below

that are code lines to be executed and at the end of each of these segments is the reserved word `break`. At the bottom, there is an optional segment beginning with the reserved word `default` and a colon. Below the default is a block of code and the reserved word `break`.

```
switch ( <Variable or Expression> ) {
case <value 1>:
    // Code if it equals value 1.
    break;
case <value 2>:
    // Code if it equals value 2.
    break;
case <value 3>:
    // Code if it equals value 3.
    break;
case <value 4>:
    // Code if it equals value 4.
    break;
case <value 5>:
    // Code if it equals value 5.
    break;
default:
    // Code if it is any other value.
    break;
}
```

The way this works is that when the switch statement is first encountered the value of the variable or expression is found, that is, evaluated. The value is then compared to the values in each case, in order of appearance in the statement. The first one that is equal to the value goes into the block of code in that segment. If none of values in the cases equal the evaluated variable or expression then the code in the default is executed.

A few things to note about the switch statement. The break statements are for the program to break out of the switch block. If you do not put in the break at the bottom of the segment the program will execute the code in the next block, even if the value in the case is not the value of the expression. This is called fall-through and we have an example of it a little later on. As we mentioned above, the default section is optional. Finally, the variable or expression must be able to be evaluated to an integer. So integer expressions can be used, as well as, characters and strings, since they evaluate to integers by the ASCII table. On the other hand, floats and doubles or their expressions cannot be used.

```
1 import java.util.Scanner;
2
```

```
3  /*-
4   * Switch Statement Example #1
5   * Author:  Don Spickler
6   * Date: 2/7/2011
7   */
8
9  public class SwitchExample001 {
10     public static void main(String[] args) {
11         Scanner keyboard = new Scanner(System.in);
12         System.out.print("Input a number (1-5): ");
13         int num = keyboard.nextInt();
14         keyboard.close();
15
16         switch (num) {
17         case 1:
18             System.out.println("One");
19             break;
20         case 2:
21             System.out.println("Two");
22             break;
23         case 3:
24             System.out.println("One more than two");
25             break;
26         case 4:
27             System.out.println("One less than five");
28             break;
29         case 5:
30             System.out.println("half of ten");
31             break;
32         }
33     }
34 }
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 1
One
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 2
Two
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 3
One more than two
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 4
One less than five
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 5
half of ten
```

Program Output **SwitchExample001.java**

```
Input a number (1-5): 7
```

This example shows fall-through. Note that if the user inputs a `c` the program will execute line 26.

```
1  import java.util.Scanner;
2
3  /*-
4   * Switch Statement Example #2
5   * Author:  Don Spickler
6   * Date: 2/7/2011
7   */
8
9  public class SwitchExample002 {
10     public static void main(String[] args) {
11         Scanner keyboard = new Scanner(System.in);
12         System.out.print("Input a single character: ");
13         String str = keyboard.next();
14         char c = str.charAt(0);
15         keyboard.close();
16
17         switch (c) {
18         case 'a':
19             System.out.println("a was typed");
20             break;
21         case 'b':
22         case 'c':
23         case 'd':
24         case 'e':
25         case 'f':
26             System.out.println("b-f was typed");
27             break;
28         case 'g':
29             System.out.println("gee");
30             break;
31         default:
32             System.out.println("something else was typed");
33             break;
34         }
35     }
36 }
```

Program Output **SwitchExample002.java**

```
Input a single character: a
a was typed
```

Program Output **SwitchExample002.java**

```
Input a single character: d
b-f was typed
```

Program Output **SwitchExample002.java**

```
Input a single character: g
gee
```

Program Output SwitchExample002.java

```
Input a single character: z
something else was typed
```

Program Output SwitchExample002.java

```
Input a single character: don
b-f was typed
```

```java
import java.util.Scanner;

/*-
 * Switch Statement Example #3
 * Author:  Don Spickler
 * Date: 2/7/2011
 */

public class SwitchExample003 {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Input a string: ");
        String str = keyboard.nextLine();
        keyboard.close();

        switch (str) {
        case "a":
            System.out.println("a was typed");
            break;
        case "Help":
            System.out.println("Help was requested.");
            break;
        case "Don":
            System.out.println("Me");
            break;
        case "Java":
            System.out.println("Java is Cool!");
            break;
        case "":
            System.out.println("Blank???");
            break;
        default:
            System.out.println("This was typed: " + str);
            break;
        }
    }
}
```

Program Output SwitchExample003.java

```
Input a string: a
a was typed
```

Program Output SwitchExample003.java

```
Input a string: Help
Help was requested.
```

Program Output **SwitchExample003.java**

```
Input a string: Don
Me
```

Program Output **SwitchExample003.java**

```
Input a string: Java
Java is Cool!
```

Program Output **SwitchExample003.java**

```
Input a string:
Blank???
```

Program Output **SwitchExample003.java**

```
Input a string: help
This was typed: help
```

Chapter 5

Repetition

5.1 Introduction

Computers can do several things really well. They can do calculations very quickly, they can store a large amount of information, they can make decisions, and they can do something over and over again without getting bored. Many algorithms need to do a process over and over again to complete. For example, consider a spell checker program. It must take each word on your document and check it against the list of words in its dictionary. So this type of program would find the first word in your document, search the dictionary for a match, if it finds one it moves to the second word and repeats the process, if it does not find a match it underlines the word as being misspelled and then moves to the second word and repeats the process. So this process is done once for each word in your document. Once every word in the document has been checked the spell checking program stops.

In Java there are three types of repetition structures, commonly called loops, the While loop, the Do-While loop, and the For loop. Each have their conveniences but any one of them would be sufficient. That is, given any one of these types of loops you could rewrite the loop in either of the other two loop forms. Loops have two main parts, the loop decision and the loop body. The loop body is a block of code that is executed if the loop decision is to repeat the loop. The loop decision is a boolean expression that determines if the loop body is to be executed again. The loop decision is checked every time the loop body finishes.

This loop condition can be done in one of two places, either before the loop body is executed or directly after the loop body is executed. In the first case, when the program first hits the loop, it checks the loop condition and if the condition is true it will do the loop body and if it is false it will skip the loop body and proceed with the rest of the program. If it dies do the loop body, when it finishes the program control

goes back to the top of the loop and checks the loop condition again. This type of loop is called a precondition loop. Both the While and For loops are precondition loops.

If the check is done after the loop body is executed then when the program hits the loop it will do the loop body and when it finishes it checks the loop condition to determine if it is to do the loop body again. This type of loop is called a postcondition loop. The Do-While loop is a postcondition loop. One thing to note is that in a postcondition loop the loop body is always done at least once. With a precondition loop it i possible that the loop body is never executed. This occurs when the loop condition is false when the program first hits the loop.

Another type of distinction is in the type of loop decision. A loop is said to be Conditionally Controlled if the loop decision is a boolean expression, as in an `if` statement. A loop is said to be Count Controlled if there is some type of counter that stops the loop. For example, say we know that we need to do a loop 10 times. We could set an integer variable to 1 and each time the loop body executes we increment the variable by one. Once the variable exceeds 10 we stop iterating the loop. Thus the variable acts as a counter and controls if the body is executed again or not. In fact, the variable is called the loop counter. The While and Do-While loops are Conditionally Controlled and the For loop is Count Controlled.

Type	Check	Control
While	Precondition	Conditional
Do-While	Postcondition	Conditional
For	Precondition	Count

So the obvious question is, in what circumstances do I use each type of loop? In general, you use the type that best fits the algorithm you are implementing. Here are some things to keep in mind.

While You do not know how many times you need to do the loop before you get to the loop and you might want to skip the loop body entirely.

Do-While You do not know how many times you need to do the loop before you get to the loop and you know that you need to do the loop body at least once.

For You know how many times you need to do the loop before you get to the loop.

One very important thing you should never forget, a loop must stop! It is easy to write a loop that never ends, I have written many of them and you will probably do the same. Remember that at some point something must change so that the loop condition is false and ends the loop. Loops that do not end are called infinite loops. If you do find yourself in an infinite loop the only option you have is to shut down the

program and fix the error. Many IDEs have a way to terminate a running program. If you are running the program directly on the operating system then you might be able to shut it down with a key sequence or through the task manager.

5.2 While Loops

The While loop has a similar structure to the if statement. We start with the reserved word `while`, followed by a boolean expression in parentheses, followed by the loop body.

```
while ( <Condition> ) {
    //  Loop Body
}
```

When the program first hits the loop the condition is checked. If the boolean expression is true the program will do the loop body and if the expression is false the program will skip the loop body and proceed with the rest of the program. If the loop body is done then at the end of the loop body the program will loop back up to the top of the statement and check the condition again. The program will continue executing the loop body as long as the condition is true, that is, *while* the condition is true.

5.2.1 Basic While Loop Example

```
1  import java.util.Scanner;
2
3  /*-
4   * While Loop Example #1
5   * Author:  Don Spickler
6   * Date: 2/6/2011
7   */
8
9  public class WhileLoopExample {
10     public static void main(String[] args) {
11         Scanner keyboard = new Scanner(System.in);
12
13         System.out.print("Input the maximum number to square: ");
14         int maxNum = keyboard.nextInt();
15         keyboard.close();
16
17         int currentNum = 1;
18         while (currentNum <= maxNum) {
19             int square = currentNum * currentNum;
20             System.out.print(square + "  ");
21             currentNum = currentNum + 1;
22         }
23         System.out.println();
24     }
25 }
```

Program Output **WhileLoopExample.java**

```
Input the maximum number to square: 10
1  4  9  16  25  36  49  64  81  100
```

5.2.2 While Loop Accumulator Example

One use of looping structures is as an accumulator. An accumulator is when you have a variable that you keep adding on to. For example, when you balance your check book you add or subtract the checks and deposits from the balance and at the end you have the current amount in your checking account. In this example the variable balance is an accumulator. You start with last months balance, and one by one, add to it the amount if you are on a deposit and subtract the amount if you are on a check.

```
1  /*-
2   * While Loop Example #2
3   * Adds up the numbers between 1 and 100.
4   * Author:  Don Spickler
5   * Date: 2/6/2011
6   */
7
8  public class WhileLoopExample {
9      public static void main(String[] args) {
10         int currentNum = 1;
11         int sum = 0;
12         while (currentNum <= 100) {
13             sum = sum + currentNum;
14             currentNum = currentNum + 1;
15         }
16         System.out.println("The sum of the numbers from 1 to 100 is " + sum + ".");
17     }
18 }
```

Program Output **WhileLoopExample.java**

```
The sum of the numbers from 1 to 100 is 5050.
```

5.2.3 While Loop Accumulator Example #2

```
1  import java.util.Scanner;
2
3  /*-
4   * While Loop Example #3
5   * Computes the sum, sum of squares, and sum of cubes of the first n numbers.
6   * Author: Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class WhileLoopExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13         System.out.print("Input n: ");
14         int n = keyboard.nextInt();
```

```
15          keyboard.close();
16
17          int currentNum = 1;
18          int sum = 0;
19          int sumSquare = 0;
20          int sumCube = 0;
21          while (currentNum <= n) {
22              sum = sum + currentNum;
23              sumSquare = sumSquare + currentNum * currentNum;
24              sumCube = sumCube + currentNum * currentNum * currentNum;
25              currentNum = currentNum + 1;
26          }
27          System.out.println("The sum of the numbers from 1 to " + n + " is " + sum + ".");
28          System.out.println("The sum of the squares of the numbers from 1 to " + n + " is "
                + sumSquare + ".");
29          System.out.println("The sum of the cubes of the numbers from 1 to " + n + " is " +
                sumCube + ".");
30      }
31  }
```

Program Output **WhileLoopExample.java**

```
Input n: 100
The sum of the numbers from 1 to 100 is 5050.
The sum of the squares of the numbers from 1 to 100 is 338350.
The sum of the cubes of the numbers from 1 to 100 is 25502500.
```

5.2.4 While Loop with Sentinel Value Example

A sentinel value is a value that is input by the user, or read in from a file or over a network, that signifies if a loop should stop. In cases like this the programmer does not know how many times the loop must iterate so he or she programs the application to continue with the loop until some type of special input is entered, this value is the sentinel value. A sentinel value should be a value that could not be legitimate input. So in the previous example of writing a loop to balance a check book, positive numbers would represent deposits and negative numbers would represent checks or withdraws. Zero, would be a fine sentinel value since you should never be depositing $0 nor would you write a check for $0.

```
1   import java.util.Scanner;
2
3   /*-
4    * While Loop Example #4
5    * Example of user input with a sentinel value.
6    * Author:  Don Spickler
7    * Date: 2/6/2011
8    */
9
10  public class WhileLoopExample {
11      public static void main(String[] args) {
12          Scanner keyboard = new Scanner(System.in);
13
14          int currentNum = 0;
15          int sum = 0;
16          while (currentNum != -1) {
17              sum = sum + currentNum;
18
```

```
19              System.out.print("Input the next positive number to add (-1 to quit): ");
20              currentNum = keyboard.nextInt();
21          }
22          System.out.println("The sum of the input numbers is " + sum + ".");
23          keyboard.close();
24      }
25  }
```

Program Output **WhileLoopExample.java**

```
Input the next positive number to add (-1 to quit): 12
Input the next positive number to add (-1 to quit): 32
Input the next positive number to add (-1 to quit): 43
Input the next positive number to add (-1 to quit): 22
Input the next positive number to add (-1 to quit): 12
Input the next positive number to add (-1 to quit): -1
The sum of the input numbers is 121.
```

5.2.5 While Loop Menu Example

Before we had nice graphical user interfaces we used text based menus to make using a computer easier.

```
1  import java.util.Scanner;
2
3  /*-
4   * While Loop Example #5
5   * Example of a text-based menu system and user input checking.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class WhileLoopExample {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         int noMore = 0;
15         while (noMore == 0) {
16             System.out.println("Please select from the following menu:");
17             System.out.println("1. Rectangle Properties");
18             System.out.println("2. Circle Properties");
19             System.out.println();
20             System.out.print("Selection: ");
21             int menuOption = keyboard.nextInt();
22             System.out.println();
23
24             if (menuOption == 1) {
25                 System.out.print("Input the width of the rectangle: ");
26                 double width = keyboard.nextDouble();
27                 System.out.print("Input the height of the rectangle: ");
28                 double height = keyboard.nextDouble();
29                 double area = height * width;
30                 double perimeter = 2 * height + 2 * width;
31                 System.out.println("The area of the rectangle is " + area);
32                 System.out.println("The perimeter of the rectangle is " + perimeter);
33                 noMore = 1;
34             } else if (menuOption == 2) {
35                 System.out.print("Input the radius of the circle: ");
36                 double radius = keyboard.nextDouble();
```

```
37                  double area = Math.PI * Math.pow(radius, 2);
38                  double circumference = 2 * Math.PI * radius;
39                  System.out.println("The area of the circle is " + area);
40                  System.out.println("The circumference of the circle is " + circumference);
41                  noMore = 1;
42              } else {
43                  System.out.println("Invalid Menu Selection!");
44                  System.out.println("Please make another selection.");
45                  System.out.println();
46              }
47          }
48          keyboard.close();
49      }
50  }
```

Program Output — WhileLoopExample.java Run #1

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 1

Input the width of the rectangle: 12
Input the height of the rectangle: 43
The area of the rectangle is 516.0
The perimeter of the rectangle is 110.0
```

Program Output — WhileLoopExample.java Run #2

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 2

Input the radius of the circle: 5
The area of the circle is 78.53981633974483
The circumference of the circle is 31.41592653589793
```

Program Output — WhileLoopExample.java Run #3

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 5

Invalid Menu Selection!
Please make another selection.

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties

Selection: 8

Invalid Menu Selection!
Please make another selection.

Please select from the following menu:
```

```
1. Rectangle Properties
2. Circle Properties

Selection: 1

Input the width of the rectangle: 2
Input the height of the rectangle: 3
The area of the rectangle is 6.0
The perimeter of the rectangle is 10.0
```

5.2.6 $3n + 1$ Sequence Example

The following is just another example of using a while loop to iterate until a particular condition is met. There is a nifty sequence in mathematics called the $3n+1$ Sequence. You start with any positive number you would like, we will call it x_0. Then you create the sequence x_0, x_1, x_2, ... by the following rule,

$$x_{i+1} = \begin{cases} 3x_i + 1 & \text{if } x_i \text{ is odd} \\ \frac{x_i}{2} & \text{if } x_i \text{ is even} \end{cases}$$

If any tern in the sequence is 1 then we stop. What is almost miraculous is that every starting number that has ever been tried results in the sequence going to 1 and hence stopping. What is also difficult to believe is that no one has been able to prove this mathematically.

```
1  import java.util.Scanner;
2
3  /*-
4   * NiftySequence: The 3n+1 Sequence.
5   * Author: Don Spickler
6   * Date: 2/6/2011
7   */
8
9  public class NiftySequence {
10     public static void main(String[] args) {
11         Scanner keyboard = new Scanner(System.in);
12         System.out.print("Input a number: ");
13         int n = keyboard.nextInt();
14         System.out.print("Sequence: " + n + "  ");
15         int count = 1;
16         while (n != 1) {
17             if (n % 2 == 0) {
18                 n = n / 2;
19             } else {
20                 n = 3 * n + 1;
21             }
22             System.out.print(n + "  ");
23             count++;
24         }
25         System.out.println();
26         System.out.println("The number of numbers in the sequence is " + count);
27     }
28 }
```

Program Output **NiftySequence.java Run #1**

```
Input a number: 17
Sequence: 17  52  26  13  40  20  10  5  16  8  4  2  1
The number of numbers in the sequence is 13
```

Program Output **NiftySequence.java Run #2**

```
Input a number: 128
Sequence: 128  64  32  16  8  4  2  1
The number of numbers in the sequence is 8
```

Program Output **NiftySequence.java Run #3**

```
Input a number: 25
Sequence: 25  76  38  19  58  29  88  44  22  11  34  17  52  26  13  40  20  10  5  16  8
      4  2  1
The number of numbers in the sequence is 24
```

5.3 Do-While Loops

The Do-While loop has a similar structure to the while loop except that the while and the condition are at the bottom of the loop instead of at the top. In addition, we begin with reserved word `do` before the loop body.

```
do {
    //  Loop Body
} while ( <Condition> );
```

When the program first hits the loop the loop body is done and after that the condition is checked. If the boolean expression is true the program will do the loop body again and if the expression is false the program will proceed with the rest of the program. The program will continue executing the loop body as long as the condition is true, that is, *while* the condition is true.

5.3.1 Basic Do-While Loop Example

```
1  /*-
2   * Do-While Loop Example #1
3   * Example that shows the basic structure of a Do-While loop.
4   * Author:  Don Spickler
5   * Date: 2/6/2011
6   */
7
8  public class DoWhile001 {
9      public static void main(String[] args) {
10         int i = 5;
11         do {
```

```
12              System.out.println(i);
13              i--;
14          } while (i > 0);
15
16          System.out.println();
17
18          i = 5;
19          while (i > 0) {
20              System.out.println(i);
21              i--;
22          }
23
24          System.out.println();
25
26          i = 0;
27          do {
28              System.out.println(i);
29              i--;
30          } while (i > 0);
31
32          System.out.println();
33
34          i = 0;
35          while (i > 0) {
36              System.out.println(i);
37              i--;
38          }
39      }
40  }
```

Program Output **DoWhile001.java**

```
5
4
3
2
1

5
4
3
2
1

0
```

5.3.2 Nested Do-While Loop Example

As with conditional statements, loops can be nested. That is, you can have loop structures inside of loop structures.

```
1  import java.util.Scanner;
2
3  /*-
4   * Do-While Loop Example #2
5   * Example that shows a nested Do-While loop.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class DoWhile002 {
```

```
11      public static void main(String[] args) {
12          Scanner keyboard = new Scanner(System.in);
13          String YesNo;
14
15          do {
16              System.out.print("Input the maximum number to square: ");
17              int maxNum = keyboard.nextInt();
18
19              int currentNum = 1;
20              do {
21                  int square = currentNum * currentNum;
22                  System.out.print(square + "  ");
23                  currentNum = currentNum + 1;
24              } while (currentNum <= maxNum);
25
26              System.out.println();
27
28              System.out.print("Do Another Sequence (Y/N): ");
29              YesNo = keyboard.next();
30
31          } while (!YesNo.equalsIgnoreCase("N"));
32
33          keyboard.close();
34      }
35  }
```

Program Output **DoWhile002.java**

```
Input the maximum number to square: 5
1  4  9  16  25
Do Another Sequence (Y/N): y
Input the maximum number to square: 10
1  4  9  16  25  36  49  64  81  100
Do Another Sequence (Y/N): n
```

5.3.3 Find Maximum And Minimum Values Example #1

This next sequence of examples shows different methods for preforming the same task, finding the largest and the smallest numbers input by the user. These programs will take input from the user and as long as the input is a positive number. If the user inputs 0 or a negative number then the program will end. So in this program any number that is 0 or negative is a sentinel value.

Before we start looking at the code and differences in the three implementations of the algorithm, we will discuss the algorithm. If we were to do this with pencil and paper we would have the person write down the list of numbers then we would examine the list to find the largest and the smallest numbers. Later in the class we will be able to take that approach but right now we have a problem. When we say that the person will write down the list of numbers, this means that we would need to store that list of numbers in the program somewhere. Since we do not know how many numbers there will be we cannot create enough variables to store all of the numbers in the list. Later in the notes we will look at arrays and array lists which

would allow us to take this route but for now we will not use them.

So let's say that we have three storage locations at our disposal. One for the number that is input, one for the maximum and one for the minimum. With this limitation, we could do the following. We will assume that max is the variable for the maximum and that it is storing the maximum value that has thus far been input. We will also assume that min is the variable for the minimum and that it is storing the minimum value that has thus far been input. Then we have the user input the next number, call it num. If num is larger than max we will store the value of num in max. If num is less than min we will store the value of num in min. If num is less than or equal to 0 then we will not check it against the min and max and instead end the program with printing out the current values of min and max. Using the discussion above we can create an outline of an algorithm.

1. Take a number, num, from the user.
2. If num is less than or equal to 0 end the program and display the values of min and max.
3. If num is greater than 0,
   (a) If num is greater than max, set max to num.
   (b) If num is less than min, set min to num.
4. Go back to step 1.

We will refine this algorithm a little before we code it. A couple questions or design decisions need to be answered and made. First, what happens on the very first input? The min and max really have no values at this stage of the program since that are to store the current min and max. If the user inputs a positive number on the first input then this number is both the min and the max, so we should set min and max to num on the first input. Second, we have the program ending at the top of the algorithm, for this to better translate to code we should put the ending of the program at the bottom.

1. Take a number, num, from the user.
2. Since this is the first input, set both min and max to num.
3. While num is positive.
   (a) If num is greater than max, set max to num.
   (b) If num is less than min, set min to num.
   (c) Take a number, num, from the user.

4. Display the values of min and max.

You will note that this implementation of the algorithm adds in conditional statement that takes care of the case where the first input is not positive, printing out that the list is empty.

```
1  import java.util.Scanner;
2
3  /*
4   * This program finds the minimum and maximum of a list of positive numbers
5   * input by the user.
6   * Author: Don Spickler
7   * Date: 9/17/2012
8   */
9
10 public class FindMaximumAndMinimum {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         System.out.print("Input a positive number (<= 0 to quit): ");
15         double num = keyboard.nextDouble();
16
17         if (num > 0) {
18             double min = num;
19             double max = num;
20             while (num > 0) {
21                 if (num < min)
22                     min = num;
23                 if (num > max)
24                     max = num;
25
26                 System.out.print("Input a positive number (<= 0 to quit): ");
27                 num = keyboard.nextDouble();
28             }
29
30             System.out.println("The minimum was " + min);
31             System.out.println("The maximum was " + max);
32         } else
33             System.out.println("No list of positive numbers was input.");
34
35     }
36 }
```

Program Output **FindMaximumAndMinimum.java**

```
Input a positive number (<= 0 to quit): 5
Input a positive number (<= 0 to quit): 3
Input a positive number (<= 0 to quit): 7
Input a positive number (<= 0 to quit): 12
Input a positive number (<= 0 to quit): 0
The minimum was 3.0
The maximum was 12.0
```

5.3.4 Find Maximum And Minimum Values Example #2

This example alters the last implementation slightly. Notice that in the last example there were two places in the code where we took input from the user. At the very

beginning to get the process started and then at the end of the loop to continue. In this example we combined these into one and put it at the beginning of the loop. To do this we needed to add in two things. First was a boolean variable to track if the input was the first input or not. Second we needed to create the variable num and set it to a value that would get us into the loop. This is called seeding the loop, that is , setting values to get into a loop and then have the loop reset them as needed.

```
1  import java.util.Scanner;
2
3  /*
4   * This program finds the minimum and maximum of a list of positive numbers
5   * input by the user.
6   * Author: Don Spickler
7   * Date: 9/17/2012
8   */
9
10 public class FindMaximumAndMinimum2 {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         double min = 0;
15         double max = 0;
16         double num = 1;
17         boolean firstnumber = true;
18
19         while (num > 0) {
20             System.out.print("Input a positive number (<= 0 to quit): ");
21             num = keyboard.nextDouble();
22
23             if (num > 0)
24                 if (firstnumber) {
25                     min = num;
26                     max = num;
27                     firstnumber = false;
28                 } else {
29                     if (num < min)
30                         min = num;
31                     if (num > max)
32                         max = num;
33                 }
34         }
35
36         if (max > 0) // Would be true if a positive number was entered.
37         {
38             System.out.println("The minimum was " + min);
39             System.out.println("The maximum was " + max);
40         } else
41             System.out.println("No list of positive numbers was input.");
42     }
43 }
```

Program Output **FindMaximumAndMinimum2.java**

```
Input a positive number (<= 0 to quit): 89
Input a positive number (<= 0 to quit): 54
Input a positive number (<= 0 to quit): 23
Input a positive number (<= 0 to quit): 85
Input a positive number (<= 0 to quit): 150
Input a positive number (<= 0 to quit): -569
The minimum was 23.0
The maximum was 150.0
```

5.3.5 Find Maximum And Minimum Values Example #3

In this example we make a very slight alteration. Since we know that we need to get input from the user, we know that we need to get into the loop, hence the reason we seeded the loop. If we know that we need to enter the loop at least once we can simply use a Do-While loop in place of the While loop.

```
1  import java.util.Scanner;
2
3  /*
4   * This program finds the minimum and maximum of a list of positive numbers
5   * input by the user.
6   * Author: Don Spickler
7   * Date: 9/17/2012
8   */
9
10 public class FindMaximumAndMinimum2 {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         double min = 0;
15         double max = 0;
16         double num;
17         boolean firstnumber = true;
18
19         do {
20             System.out.print("Input a positive number (<= 0 to quit): ");
21             num = keyboard.nextDouble();
22
23             if (num > 0)
24                 if (firstnumber) {
25                     min = num;
26                     max = num;
27                     firstnumber = false;
28                 } else {
29                     if (num < min)
30                         min = num;
31                     if (num > max)
32                         max = num;
33                 }
34         } while (num > 0);
35
36         if (max > 0) // Would be true if a positive number was entered.
37         {
38             System.out.println("The minimum was " + min);
39             System.out.println("The maximum was " + max);
40         } else
41             System.out.println("No list of positive numbers was input.");
42     }
43 }
```

Program Output **FindMaximumAndMinimum2.java**

```
Input a positive number (<= 0 to quit): 15
Input a positive number (<= 0 to quit): 6
Input a positive number (<= 0 to quit): 123
Input a positive number (<= 0 to quit): 59
```

```
Input a positive number (<= 0 to quit): 78
Input a positive number (<= 0 to quit): -25
The minimum was 6.0
The maximum was 123.0
```

5.3.6 Guessing Game Example

The Guessing Game is a simple and frankly not very exciting game. Once mastered it has about the same appeal as tic-tac-toe. In this game your opponent picks a number between 1 and 100 and your task is to guess the number. After each guess your opponent must tell you if the number ot be guessed is high or lower than the one you guessed, and he or she must tell the truth. You have 7 tries to guess the number, if you fail to guess the number in 7 tries your opponent wins and if you succeed in guessing the number you win.

The strategy of the game is simple, you keep guessing the midpoint of the guessing interval that remains. For example, start out with a guess of 50, if your opponent says higher, then you go with 75 and if he or she says lower you go with 25. Let's say that they said lower and you guessed 25. If they then say higher, you would guess 37, and so on. Since $\log_2(100) < 7$, as long as you do not mess up you will guess the number within your 7 tries.

Let's construct an algorithm for this game. We will add the feature that the program will play as many games as the uses wants to play and the program will keep score of how many games the user wins and how many the player wins. The computer will choose the number randomly and keep track of the number of guesses.

1. While the user wants to play another game.
   (a) While the user has another guess and they have not guessed the number yet.
      i. Get the next guess from the user.
      ii. If the guess was too low, tell the user to guess higher.
      iii. If the guess was too high, tell the user to guess lower.
      iv. If the guess was correct, tell the user that they won.
   (b) Ask the user if they would like to play another game.

One thing this algorithm does not take into account is printing out a message that the player has lost if they do not guess the number within the 7 tries. Furthermore we could get a little more detailed with the algorithm.

1. Initialize two variables for tracking the number of player wins and the number of computer wins to 0.

2. While the user wants to play another game.
   (a) Create a new random number between 1 and 100.
   (b) Set the number of guesses the user has made to 1.
   (c) While the user has another guess and they have not guessed the number yet.
      i. Get the next guess from the user.
      ii. If the number of guesses is less than 7,
         A. If the guess was too low, tell the user to guess higher.
         B. If the guess was too high, tell the user to guess lower.
         C. If the guess was correct, tell the user that they won and increment the player win counter.
      iii. If the number of guesses is 7,
         A. If the guess was correct, tell the user that they won and increment the player win counter.
         B. If the guess was incorrect, tell the user that they lost and increment the computer win counter.
      iv. Increment the guess counter
   (d) Ask the user if they would like to play another game.
3. Output the final score.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * GuessingGame
6   * Plays a numeric guessing game with the user.
7   * Author:  Don Spickler
8   * Date: 2/6/2011
9   */
10
11 public class GuessingGame {
12     public static void main(String[] args) {
13         Random generator = new Random();
14         Scanner keyboard = new Scanner(System.in);
15
16         String playAgain = "Y";
17         int userWins = 0;
18         int computerWins = 0;
19
20         while (playAgain.compareToIgnoreCase("Y") == 0) {
21             int answer = generator.nextInt(100) + 1;
22             int numGuesses = 1;
23             int guess = 0;
24
25             while ((numGuesses <= 7) && (guess != answer)) {
26                 System.out.print("Guess a number: ");
27                 guess = keyboard.nextInt();
28
```

```
29                  if (numGuesses < 7) {
30                      if (guess < answer) {
31                          System.out.println("Higher...");
32                      } else if (guess > answer) {
33                          System.out.println("Lower...");
34                      } else {
35                          System.out.println("You Win");
36                          userWins++;
37                      }
38                  } else {
39                      if (guess == answer) {
40                          System.out.println("You Win");
41                          userWins++;
42                      } else {
43                          System.out.println("I Win, the number was " + answer);
44                          computerWins++;
45                      }
46                  }
47                  numGuesses++;
48              }
49
50              System.out.print("Would you like to play another game? (Y/N): ");
51              playAgain = keyboard.next();
52          }
53
54          System.out.println("Final Score: You " + userWins + "   Computer " + computerWins)
                ;
55      }
56  }
```

Program Output — GuessingGame.java

```
Guess a number: 50
Higher...
Guess a number: 75
Lower...
Guess a number: 62
Higher...
Guess a number: 68
Lower...
Guess a number: 65
Lower...
Guess a number: 64
Lower...
Guess a number: 63
You Win
Would you like to play another game? (Y/N): y
Guess a number: 51
Lower...
Guess a number: 50
Lower...
Guess a number: 49
Lower...
Guess a number: 48
Lower...
Guess a number: 47
Lower...
Guess a number: 46
Lower...
Guess a number: 45
I Win, the number was 12
Would you like to play another game? (Y/N): n
Final Score: You 1   Computer 1
```

When you read the above code you will notice that we used a different method for finding the random number, we used the `Random` class in place of `Math.random`.

```
Random generator = new Random();
int answer = generator.nextInt(100) + 1;
```

The Random class has many nice features for pseudo-random number generation, one of which is its `nextInt` method that returns the next random integer between 0 and the number used for the parameter. So `nextInt(100)` returns a number between 0 and 99, so adding one returns a number between 1 and 100.

5.4 For Loops

The third type of loop is the For loop. Recall that this was a count controlled loop, so usually there is a numeric counter that is linked to this loop, although it is not necessary. The syntax for a For loop starts out with the reserved word `for` and then in parentheses is a group of three items separated by semicolons. The first is the initializer, the second is the loop condition, and the third is the update.

```
for ( <Initializer> ; <Loop Condition> ; <Update> ) {
    // Loop Body
}
```

When the program first hits the for loop it runs the code in the initializer section. It then checks the loop condition. If the loop condition is true the program will execute the body of the loop. If the loop condition is false, the program will skip the loop body and proceed with the rest of the program. If the loop body is executed, the program wil go back up to the top of the loop, run the code in the update section and then check the loop condition again.

5.4.1 Basic For Loop Example

```
1  import java.util.Scanner;
2
3  /*-
4   * For Loop Example #1
5   * Basic example of a For loop.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ForLoopExamples001 {
11     public static void main(String[] args) {
12         for (int i = 0; i < 10; i++) {
13             System.out.print(i + "  ");
14         }
15     }
16 }
```

Program Output **ForLoopExamples001.java**

```
0  1  2  3  4  5  6  7  8  9
```

In this example, when the loop is first encountered, the code `int i = 0` is run, which creates the integer `i` and sets it to 0. It then checks the loop condition `i < 10`, which is true, so it then executes the instructions in the loop. The loop body simply prints out the current value of `i`, 0. The loop then runs the update code `i++` which increments `i` to 1, checks the loop condition `i < 10`, which is true, so it then executes the instructions in the loop again. The loop body simply prints out the current value of `i`, 0. The loop then runs the update code `i++` which increments `i` to 2, checks the loop condition `i < 10`, which is true, so it then executes the instructions in the loop again. And so on, until `i` gets incremented to 10. Then when the loop condition is checked `i < 10`, which is false, so it skips the loop body and continues with the rest of the program.

5.4.2 For Loop Update Example

In this example we have changed the update to `i += 3` so the increments will be by 3 and not by 1.

```
1  /*-
2   * For Loop Example #2
3   * Basic example of a For loop.
4   * Author:  Don Spickler
5   * Date: 2/6/2011
6   */
7
8  public class ForLoopExamples002 {
9      public static void main(String[] args) {
10         for (int i = 0; i < 10; i += 3) {
11             System.out.print(i + "  ");
12         }
13     }
14 }
```

Program Output **ForLoopExamples002.java**

```
0  3  6  9
```

5.4.3 For Loop Update in Body Example

Notice that in this example, we increment the loop counter both in the update and in the body of the loop. Notice the result in the output. In general, you do not want to update the loop counter in the loop body, although there may be situations where is warranted.

```
1  /*-
2   * For Loop Example #3
3   * Basic example of a For loop.
```

```
 * Author:  Don Spickler
 * Date: 2/6/2011
 */

public class ForLoopExamples003 {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + "  ");
            i++;
        }
    }
}
```

Program Output **ForLoopExamples003.java**

```
0  2  4  6  8
```

5.4.4 For Loop Multiple Update Example

In some situations you may want to do several things when the update is executed. To do this you simply need to put a comma between the segments that you want to execute.

```
/*-
 * For Loop Example #4
 * For loop with multiple updates.
 * Author:  Don Spickler
 * Date: 2/6/2011
 */

public class ForLoopExamples004 {

    public static void main(String[] args) {
        int j = 4;
        for (int i = 0; i < 10; i++, j--) {
            System.out.println(i + "  " + j);
        }

        System.out.println();

        j = 2;
        for (int i = 0; i < 10; i = -j, j = j - 3) {
            System.out.println(i + "  " + j);
        }

        System.out.println();

        String str = "";
        for (int i = 0; i < 10; i++, str = str + "A") {
            System.out.println(i + "  " + str);
        }
    }
}
```

Program Output **ForLoopExamples004.java**

```
0  4
1  3
```

```
2  2
3  1
4  0
5  -1
6  -2
7  -3
8  -4
9  -5

0  2
-2  -1
1  -4
4  -7
7  -10

0
1  A
2  AA
3  AAA
4  AAAA
5  AAAAA
6  AAAAAA
7  AAAAAAA
8  AAAAAAAA
9  AAAAAAAAA
```

5.4.5 For Loop with Empty Sections Example

You can have empty sections in the for loop statement. For example, if there is no initializer then when the program first hits the loop it skips to the loop condition check. If the update is empty then the loop does not do anything after the loop body is executed, again it goes back to the loop condition. If the loop condition is empty then the loop condition is always true. So if the loop condition segment is empty the programmer must use a break statement to exit the loop or the program will have an infinite loop.

```
1  import java.util.Scanner;
2
3  /*-
4   * For Loop Example #5
5   * For loop with multiple updates.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class ForLoopExamples005 {
11
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14
15         int j = 7;
16         for (int i = 0; i < j;) {
17             System.out.print(i + " ");
18             i++;
19         }
20
21         System.out.println();
```

```
22
23          j = 7;
24          for (int i = 0;;) {
25              System.out.print(i + " ");
26              i++;
27              if (i >= j)
28                  break;
29          }
30
31          System.out.println();
32
33          for (int i = 0, k = 7;;) {
34              System.out.print(i + " ");
35              i++;
36              if (i >= k)
37                  break;
38          }
39
40          System.out.println();
41
42          int i = 0, k = 7;
43          for (; i < k;) {
44              System.out.print(i + " ");
45              i++;
46          }
47
48          System.out.println();
49          System.out.println();
50
51          System.out.print("Input a number: ");
52          int n = keyboard.nextInt();
53          System.out.print("Sequence: " + n + "  ");
54          int count = 1;
55          for (;;) { // The same as while(true)
56              if (n == 1)
57                  break;
58
59              if (n % 2 == 0) {
60                  n = n / 2;
61              } else {
62                  n = 3 * n + 1;
63              }
64              System.out.print(n + "  ");
65              count++;
66          }
67          System.out.println();
68          System.out.println("The number of numbers in the sequence is " + count);
69
70          keyboard.close();
71      }
72  }
```

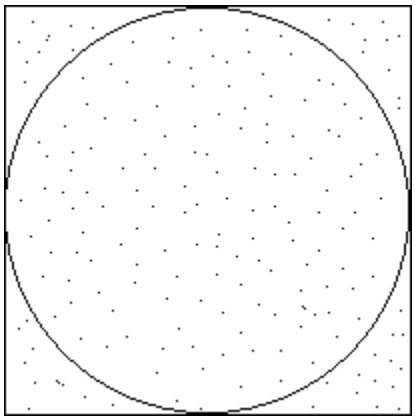
Program Output — ForLoopExamples005.java

```
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6

Input a number: 7
Sequence: 7  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
The number of numbers in the sequence is 17
```

5.4.6 Monte Carlo Approximation of π Example

A Monte Carlo method is a method that incorporates some type of randomness in the algorithm. The Monte Carlo method for approximating π goes like this. Imagine a circular dartboard of radius 1, centered on a background square that is length 2 on each side, as in the picture below. Now close your eyes, tell everyone to stand back, and start throwing darts at the dartboard. We will assume that all of your darts hit the square but are relatively random in their placement.



Since the darts are evenly and randomly distributed over the square you would expect that the ratio of the number of darts that land inside the circle over the number of total darts is approximately the same as the ratio of the area of the circle over the area of the square, that is, $\frac{\pi}{4}$. So to approximate π we simply take this approximation to $\frac{\pi}{4}$ and multiply it by 4. So the algorithm is simple, we first specify the number of darts we want to use, then for each dart we generate a random position in the square, if the position is inside the circle we increment a counter, finally when all of the darts are thrown we calculate the ratio of the darts inside the circle over the total number of darts and multiply by 4.

1. Get the total number of darts to use from the user, numDarts.
2. Create a variable to count the number of darts inside the circle, count, and set it to 0.
3. For each dart,
   (a) Generate a random point (x, y) in the square. This can be done by generating a random x value in the interval $[-1, 1]$ and the same for the y value.

(b) Test if the dart is inside the circle, that is, if the distance between (x, y) and the origin is less than or equal to 1. Specifically, $\sqrt{x^2 + y^2} \leq 1$. If it is, add one to the counter count.

4. Take the ratio of count over numDarts, multiply by 4, and display the result.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * PiApprox: Monte Carlo approximation of pi.
6   * Author: Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class PiApprox {
11     public static void main(String[] args) {
12         Random generator = new Random();
13         Scanner keyboard = new Scanner(System.in);
14
15         System.out.print("Input the number of darts: ");
16         int numDarts = keyboard.nextInt();
17
18         int count = 0;
19         for (int i = 0; i < numDarts; i++) {
20             double x = 2 * generator.nextDouble() - 1;
21             double y = 2 * generator.nextDouble() - 1;
22             if (Math.sqrt(x * x + y * y) <= 1)
23                 count++;
24         }
25
26         double prob = 1.0 * count / numDarts;
27         System.out.println("Pi = " + prob * 4);
28     }
29 }
```

Program Output **PiApprox.java Run #1**

```
Input the number of darts: 100000
Pi = 3.13452
```

Program Output **PiApprox.java Run #2**

```
Input the number of darts: 1000000
Pi = 3.14116
```

Program Output **PiApprox.java Run #3**

```
Input the number of darts: 10000000
Pi = 3.1421416
```

A couple things to note about this example. First, $\pi = 3.14159265358979\ldots$, so the approximations above are good to 2 or 3 decimal places. On the one hand that seems fairly good but on the other hand this required throwing one million to ten million darts. Approximations to π are not done in this manner if you want a lot

of accuracy. Normally what is done is a little bit of Calculus. Usually in Calculus II you learn about sequences and series. In the process you learn about the power series for standard functions like $\sin(x)$ and $\cos(x)$. In addition you probably see the series expansion for $\tan^{-1}(x)$. Using the fact that $\tan^{-1}(1) = \frac{\pi}{4}$, we take the series expansion for the arctangent, evaluate it at 1, and multiply the result by 4.

Second, we could have simplified the program a little. The inequality $\sqrt{x^2 + y^2} \leq 1$ is true if and only if the inequality $x^2 + y^2 \leq 1$ is true. So the condition,

```
if (Math.sqrt(x * x + y * y) <= 1)
    count++;
```

could be rewritten as,

```
if (x * x + y * y <= 1)
    count++;
```

Third, notice that the approximation using 1,000,000 darts is actually better than the one using 10,000,000 darts. Although this is a bit counter-intuitive, with Monty Carlo methods it is fairly common. Since the Monty Carlo method uses randomness (really pseudo-randomness) it is only by chance if some approximations are closer to the actual value or not. In general, the larger number of darts you use the closer you will get to the value of π, but you are not guaranteed to get a better approximation by simply using a larger number of darts.

5.4.7 Break Example #1

As we saw in the switch statement in the previous chapter, the reserved word `break` jumps you out of the switch statement. The reserved word `break` can be used with other structures as well and it will jump you out of the current structure you are in. With that said, you really should not use a break statement in anything else other then a switch control. It tends to lead to hard to read code and lazy programming style. On the other hand, it can be useful when you are debugging a program and want to jump out of a structure just before it creates an error. Outside of a switch statement, debugging is the only thing I use a break for.

```
1  /*-
2   * BreakExample001
3   * Example of using break to break out of a block.
4   * Author:  Don Spickler
5   * Date: 2/6/2011
6   */
7
8  public class BreakExample001 {
9      public static void main(String[] args) {
10         System.out.println("Before the loop");
11
12         for (int i = 0; i < 10; i++) {
13             System.out.println(i);
```

```
14              if (i == 6)
15                  break;
16          }
17
18          System.out.println("After the loop");
19      }
20  }
```

Program Output **BreakExample001.java**

```
Before the loop
0
1
2
3
4
5
6
After the loop
```

5.4.8 Break Example #2

Notice in this example, the break jumps you out of the inner loop but does not jump you out of the outer loop. So a break will jump you out of a structure but only the one at the current level of nesting.

```
1  /*-
2   * BreakExample002
3   * Example of using break to break out of a block.
4   * Author:  Don Spickler
5   * Date: 2/6/2011
6   */
7
8  public class BreakExample002 {
9
10     public static void main(String[] args) {
11         System.out.println("Before the j loop");
12         for (int j = 0; j < 2; j++) {
13             System.out.println("Before the i loop");
14             for (int i = 0; i < 10; i++) {
15                 System.out.println(i);
16                 if (i == 6)
17                     break;
18             }
19             System.out.println("After the i loop");
20         }
21         System.out.println("After the j loop");
22     }
23 }
```

Program Output **BreakExample002.java**

```
Before the j loop
Before the i loop
0
1
2
3
```

```
4
5
6
After the i loop
Before the i loop
0
1
2
3
4
5
6
After the i loop
After the j loop
```

Chapter 6

Methods

6.1 Introduction

Methods are also known as functions and subroutines. These are a way to let the programmer break the program into smaller portions that are more easily manageable and can be called from the main and reused when needed. Up until now our entire program has been written in the main. This is fine for short programs but as your programs get longer and more complex you will want to break them up to make the programming task more manageable.

Although our programs will only be a couple hundred lines in length, at most, professional systems like computer games, operating systems, Java, and Eclipse can be thousands to millions of lines in length. In addition, these programs are not written by a single programmer but teams of programmers. Having a way to easily segment a program is a feature that is needed when the program is being written by several people to several hundred people.

Also, these methods allow you to reuse segments of code and hence reduce the length of your programs. For example, lets say that you are writing a program that prints out student data and these are two places where the data prints out. If you put all of the code in the main then you would would need to write two segments that looked identical to do the printing. With a method you could write the code once and let the main call the method at the two different spots. This will save you lines of code, make the program easier to read and update. If you needed to change something in the printing code you would only need to change it once in the method instead of twice.

There are a lot of ways to create these methods and we will look at several of them as we go through this chapter. The basic idea is to create a segment of code for the method and then have the main, or other method, call the method and run the

code as if it were substituted in place of the call.

6.2 Methods Without Parameters

The most basic type of method is one that has no parameters. In the first example below,

```
public static void PrintLine() {
    System.out.println("This is a line of text.");
}
```

is the method and

```
public static void main(String[] args) {
    System.out.println("Start Here");
    PrintLine();
    System.out.println("Back to the Main");
    PrintLine();
    System.out.println("End Here");
}
```

is the main that calls it. Notice the syntax of the method. it starts out with the reserved words `public static void`, what follows is the name of the method, `PrintLine` in this case, and then empty parentheses and finally the block of code that is the method body.

The `void` reserved word is the return type. Some methods can return information to the main program and if they do, then this `void` is replaced by the data type of the information being sent back. For example, if our method returned an integer then the `void` would be replaced by `int`. We will see examples of this later. Also, at the end of the method name we have a set of empty parentheses. When we look at examples that use parameters, we will be putting variables in these parentheses. Again we will look at examples of this later in the chapter.

In the main, notice the lines `PrintLine();` These are the method calls. In general a method call is simply the name of the method. When the program starts it always starts in the main, no matter what order the methods are in. So the first thing the program does is print out *Start Here*. Then the program hits the second line of the main which is the method call. The program then jumps to the method and runs the code in the method, so it prints out *This is a line of text.*. Once the program hits the end of the method it moves back to the line right after the method call. So the next this that is done is printing out *Back to the Main*. The next line is another method call, so we go back to the method, print out *This is a line of text.*, and return to the main. Finally, the program prints out, *End Here* and then the program halts.

6.2.1 Simple Method Call Example

```
1  public class MethodExample001 {
2
3      /*-
4       * MethodExample001
5       * Simple example of the use of methods in Java.
6       * Author:  Don Spickler
7       * Date: 3/7/2011
8       */
9
10     public static void PrintLine() {
11         System.out.println("This is a line of text.");
12     }
13
14     public static void main(String[] args) {
15         System.out.println("Start Here");
16         PrintLine();
17         System.out.println("Back to the Main");
18         PrintLine();
19         System.out.println("End Here");
20     }
21 }
```

Program Output **MethodExample001.java**

```
Start Here
This is a line of text.
Back to the Main
This is a line of text.
End Here
```

6.2.2 Multiple Method Calls Example

Your program can have as many methods as you would like. In the next example, there are two methods, both are simple print statements. The main calls the first method on line 19 and calls the second method on line 26.

```
1  public class MethodExample002 {
2
3      /*-
4       * MethodExample002
5       * Simple example of the use of methods in Java.
6       * Author:  Don Spickler
7       * Date: 3/7/2011
8       */
9
10     public static void PrintIntro() {
11         System.out.println("This is the intro to the program.");
12     }
13
14     public static void PrintGoodbye() {
15         System.out.println("Have a nice day :-)");
16     }
17
18     public static void main(String[] args) {
19         PrintIntro();
20
21         for (int i = 0; i < 5; i++) {
22             System.out.print(i + " ");
```

```
23          }
24          System.out.println();
25
26          PrintGoodbye();
27      }
28  }
```

Program Output **MethodExample002.java**

```
This is the intro to the program.
0 1 2 3 4
Have a nice day :-)s
```

Most of the time your methods will have more lines of code in them and do something more than simply printing out a line of text. We are keeping it simple here to illustrate the program flow.

6.3 Methods With Parameters

In the examples above, the main program called the methods but there was no information that the methods needed from the main. The methods were simple print statements so there was no need for any information. But what if we constructed a method that calculated the area of a circle. The area of a circle is πr^2 where r is the radius of the circle. If we input the value of r from the keyboard in the main then the main program must pass it to the method so that the method can finish the calculation. You may say, well just input the value of r in the method. This can be done but in some cases you may not want to use this style, and in some cases this style may cause the program to be harder to use.

One way I like to think about methods are like the kitchen in a restaurant, the ones that have a door labeled IN and one labeled OUT. Waiters can only go into the kitchen through the IN door and they can only come out of the kitchen using the OUT door. The IN door of a method is the parameters for the method. This is the only way that information can go from the main to the method. The return type (that has been `void` so far) is the OUT door, that is, the only way that information can go back to the main from the method.

You can pass as many different bits of information to the method as you would like. We will start with the case of passing one bit of information. When passing parameters you place the parameter in the parentheses after the method name. A parameter is a variable so it must be given a type. In the example below, the method `Welcome` has a single parameter named `name` which is a string.

```
public static void Welcome(String name) {
    System.out.println("Welcome to Java Methods " + name);
}
```

```
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Input your name: ");
    String myName = keyboard.nextLine();
    keyboard.close();
    Welcome(myName);
}
```

In the main, we take in a string input from the user that is stored in the variable `myName`. On the last line you see the method call, which is the method name followed by `myName` in parentheses. What happens in this call is that the program goes up to the `Welcome` method and passes the contents of `myName` to `name` in the method. So the contents of `myName` is the waiter who just went through the IN door into the `Welcome` kitchen. In the kitchen he is printed out along with the string *Welcome to Java Methods*.

Several things to notice here.

1. The data type of the variable that is sent to the method must be the same type as parameter or you will get an error. So we could not have put an integer in for `myName`.

2. We do not need to send a variable holding a string from the main, we could use a string literal. So for example, we could have called the `Welcome` method by `Welcome("John");` and the output of the program would have been *Welcome to Java Methods John*.

3. What is really happening in memory is that there are two different variable locations, one for `myName` and one for `name`. When the program starts and the user types in their name the input is stored in the location labeled `myName`. When the method `Welcome` is called the contents of the `myName` location are copied to the location for `name`, essentially it is the assignment statement `name = myName;`. Then the method uses the contents of `name` and the main used the contents of `myName`.

4. Along with the note above, if the method alters the contents of `name` there is no affect to the contents of `myName` since these are different memory locations.

5. The main cannot see the variable `name` and the method cannot see the variable `myName`. Essentially, `name` is in the kitchen and `myName` is in the restaurant.

6. The above notes should seem fairly believable since `name` and `myName` seem like they are different variables, they are in fact different names. The same will be true even if they were both named the same thing. For example,

```
    public static void Welcome(String name) {
        System.out.println("Welcome to Java Methods " + name);
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Input your name: ");
        String name = keyboard.nextLine();
        keyboard.close();
        Welcome(nsame);
    }
```

will produce the same output and in fact act exactly the same way as the previous example. There are two separate memory locations one for `name` in the main and one for `name` in the method. So, as above, altering one of them will not affect the other. When you use `name` in the main it is the one associated with the main and when you use `name` in the method it is the one associated with the method.

6.3.1 Single Parameter Passing Example

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample003
5   * Simple example of the use of methods in Java.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample003 {
11
12     public static void Welcome(String name) {
13         System.out.println("Welcome to Java Methods " + name);
14     }
15
16     public static void main(String[] args) {
17         Scanner keyboard = new Scanner(System.in);
18         System.out.print("Input your name: ");
19         String myName = keyboard.nextLine();
20         keyboard.close();
21         Welcome(myName);
22     }
23 }
```

Program Output **MethodExample003.java**

```
Input your name: Don Spickler
Welcome to Java Methods Don Spickler
```

6.3.2 Multiple Parameter Passing Example

You are not restricted to passing a single parameter to a method, you can pass as many as you need. When passing multiple parameters they are passed by position.

So the first parameter in the call gets sent to the first parameter in the method, the second parameter on the call gets sent to the second parameter in the method, and so on. As with a single parameter, the data types must match between the call and the method. So if a method is expecting a string for the first parameter and there is an integer in the first parameter in the call then we have an error.

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample004
5   * Simple example of the use of methods in Java.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample004 {
11
12     public static void PrintFormalName(String firstName, String lastName) {
13         System.out.println("Hello " + firstName + " " + lastName);
14         System.out.println("Your formal name is " + lastName + ", " + firstName);
15     }
16
17     public static void main(String[] args) {
18         Scanner keyboard = new Scanner(System.in);
19         System.out.print("Input your name as, first last: ");
20         String firstName = keyboard.next();
21         String lastName = keyboard.next();
22         keyboard.close();
23
24         PrintFormalName(firstName, lastName);
25     }
26 }
```

Program Output **MethodExample004.java**

```
Input your name as, first last: Don Spickler
Hello Don Spickler
Your formal name is Spickler, Don
```

6.3.3 Multiple Parameter Passing Example #2

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample005
5   * Simple example of the use of methods in Java.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample005 {
11
12     public static void PrintCircleArea(double radius) {
13         System.out.println("Circle Area = " + Math.PI * radius * radius);
14     }
15
16     public static void PrintRectangleArea(double length, double width) {
17         System.out.println("Rectangle Area = " + length * width);
```

```
18      }
19
20      public static void main(String[] args) {
21          Scanner keyboard = new Scanner(System.in);
22          System.out.print("Input the radius of the circle: ");
23          double rad = keyboard.nextDouble();
24          System.out.print("Input the length of the rectangle: ");
25          double len = keyboard.nextDouble();
26          System.out.print("Input the width of the rectangle: ");
27          double wid = keyboard.nextDouble();
28          keyboard.close();
29
30          PrintCircleArea(rad);
31          PrintRectangleArea(len, wid);
32      }
33  }
```

Program Output **MethodExample005.java**

```
Input the radius of the circle: 3
Input the length of the rectangle: 5
Input the width of the rectangle: 8
Circle Area = 28.274333882308138
Rectangle Area = 40.0
```

6.3.4 Multiple Parameter Passing Example #3

In our last example we had two methods that printed out the area of a circle and the area of a rectangle. What if we wanted to simply calculate these values and not print them out but instead use them in a subsequent calculation. For example, using the areas to then calculate the volume of a box or the volume of a cylinder. Instead of having the method print out the result we could have the program simply send the value of the computation back to the main, or whichever method called the area calculation method.

To do this, we need to make two changes. First we need to change the `void` to the type of data that is to be returned, `double` in this case. Second we need to replace the print statement with a `return` statement. The reserved word `return` is used at the end of a method that is returning a (non-void) type of data. The syntax is simple, it is the word `return` followed by the data value, variable, or calculation that is to be returned. Any calculation that follows a `return` statement is evaluated and its result is sent back as the return value.

In the call, the calling statement is replaced with the value of the return. So in the example below, the user input the radius, rad, of the circle to be 3. This was sent to the the `CircleArea` method and in the method the value of `radius` was assigned to 3. The method then calculated πr^2, to get a value of 28.274333882308138. This value was then sent back to the call and the call was replaced by the value. So the line,

```
System.out.println("Circle Area = " + CircleArea(rad));
```

was essentially replaced by the line,

```
System.out.println("Circle Area = " + 28.274333882308138);
```

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample006
5   * Simple example of the use of methods in Java.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample006 {
11
12     public static double CircleArea(double radius) {
13         return Math.PI * radius * radius;
14     }
15
16     public static double RectangleArea(double length, double width) {
17         return length * width;
18     }
19
20     public static void main(String[] args) {
21         Scanner keyboard = new Scanner(System.in);
22         System.out.print("Input the radius of the circle: ");
23         double rad = keyboard.nextDouble();
24         System.out.print("Input the length of the rectangle: ");
25         double len = keyboard.nextDouble();
26         System.out.print("Input the width of the rectangle: ");
27         double wid = keyboard.nextDouble();
28         keyboard.close();
29
30         System.out.println("Circle Area = " + CircleArea(rad));
31         System.out.println("Rectangle Area = " + RectangleArea(len, wid));
32     }
33 }
```

Program Output **MethodExample006.java**

```
Input the radius of the circle: 3
Input the length of the rectangle: 5
Input the width of the rectangle: 7
Circle Area = 28.274333882308138
Rectangle Area = 35.0
```

6.3.5 Multiple Parameter Passing Example #4

In this example we simply add a method for creating a menu for the user to select between a couple options. In this menu method, at the end, there is some user input checking for a valid input. If the user types in an invalid selection the method will detect it and display the menu again for a valid input. User input checking is a large part of many professional software packages. Validating input keeps the program returning reasonable answers as well as keeping the program from crashing. Although this program will make sure that selected value is in the correct range it

is far from bullet proof. If you run the program and type in a character instead of an integer the program will crash. We will look at ways to keep the program from crashing in a situation like this later on in the notes.

Although this is a text-based menu system it is set up in a similar manner to the graphical user interface (GUI) menu systems in your professional applications, like Eclipse or a word processor. Most GUI systems have their own method (or methods) for displaying the menu and when you make a menu selection this usually triggers another specific method to preform the task the user just asked for.

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample007
5   * Simple example of the use of methods in Java.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample007 {
11
12     public static double CircleArea(double radius) {
13         return Math.PI * radius * radius;
14     }
15
16     public static double RectangleArea(double length, double width) {
17         return length * width;
18     }
19
20     public static int menu() {
21         Scanner keyboard = new Scanner(System.in);
22         int menuOption = 0;
23         while (menuOption < 1 || menuOption > 3) {
24             System.out.println("Please select from the following menu:");
25             System.out.println("1. Rectangle Properties");
26             System.out.println("2. Circle Properties");
27             System.out.println("3. Exit");
28             System.out.println();
29             System.out.print("Selection: ");
30             menuOption = keyboard.nextInt();
31             System.out.println();
32
33             if (menuOption < 1 || menuOption > 3) {
34                 System.out.println("Invalid Menu Selection!");
35                 System.out.println("Please make another selection.");
36                 System.out.println();
37             }
38         }
39         return menuOption;
40     }
41
42     public static void main(String[] args) {
43         Scanner keyboard = new Scanner(System.in);
44
45         int menuOption = 0;
46         while (menuOption != 3) {
47             menuOption = menu();
48
49             if (menuOption == 1) {
50                 System.out.print("Input the width of the rectangle: ");
51                 double width = keyboard.nextDouble();
```

```
52                  System.out.print("Input the height of the rectangle: ");
53                  double height = keyboard.nextDouble();
54                  System.out.println("The area of the rectangle is " + RectangleArea(width,
                        height));
55              } else if (menuOption == 2) {
56                  System.out.print("Input the radius of the circle: ");
57                  double rad = keyboard.nextDouble();
58                  System.out.println("The area of the circle is " + CircleArea(rad));
59              }
60              System.out.println();
61          }
62          keyboard.close();
63      }
64  }
```

Program Output **MethodExample007.java**

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 5

Invalid Menu Selection!
Please make another selection.

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 1

Input the width of the rectangle: 4
Input the height of the rectangle: 7
The area of the rectangle is 28.0

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 2

Input the radius of the circle: 3
The area of the circle is 28.274333882308138

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 3
```

6.3.6 Nifty Sequence Example

The next example is simply the $3n + 1$ sequence we saw before. The only difference here is that the next number in the sequence is calculated in a method. When you are

writing longer programs and you find that the main, or another method, is getting too complicated to easily follow it might be time to rethink some of the program structure, and extract some code for a new method.

```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample008
5   * Nifty Sequence Example
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample008 {
11
12     public static int NiftySequence(int n) {
13         if (n % 2 == 0) {
14             n = n / 2;
15         } else {
16             n = 3 * n + 1;
17         }
18         return n;
19     }
20
21     public static void main(String[] args) {
22         Scanner keyboard = new Scanner(System.in);
23         System.out.print("Input a number: ");
24         int n = keyboard.nextInt();
25         keyboard.close();
26
27         System.out.print("Sequence: " + n + "  ");
28         int count = 1;
29         while (n != 1) {
30             n = NiftySequence(n);
31             System.out.print(n + "  ");
32             count++;
33         }
34         System.out.println();
35         System.out.println("The number of numbers in the sequence is " + count);
36     }
37 }
```

Program Output **MethodExample008.java**

```
Input a number: 7
Sequence: 7  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
The number of numbers in the sequence is 17
```

6.3.7 Another Mathematical Example

This next example is nothing special, just another calculation. Given three points in the Cartesian plane, determine the area of the triangle formed by those three points. To do this calculation we need two mathematical formula. First we need to take the points and determine the distance between these points, this will give us the length of each of the sides. Once we have the lengths of the sides of the triangle we can find the area of the triangle by using Heron's formula. The lengths of the sides can be

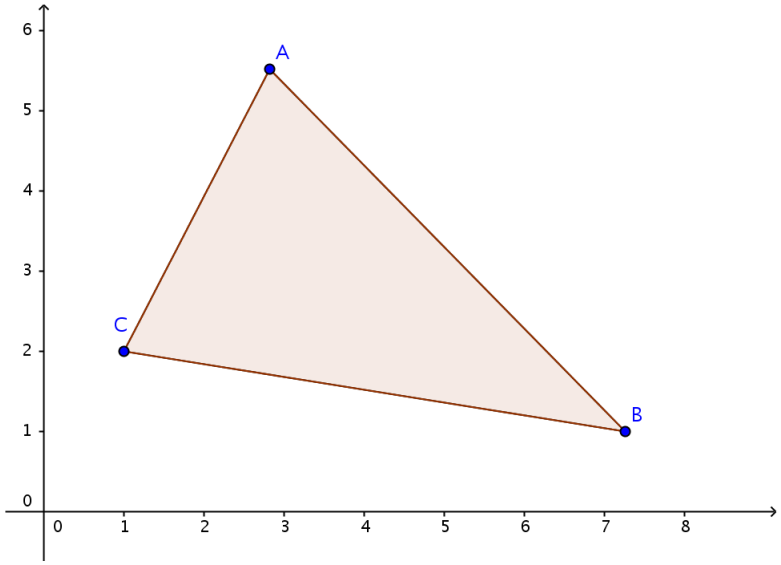
found by,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where (x_1, y_1) and (x_2, y_2) are the coordinates of two of the vertices of the triangle. This will give us the lengths of the sides, lets call them a, b, and c. Heron's formula states that given the lengths of the sides of the triangle we can calculate its area by

$$A = \sqrt{p(p-a)(p-b)(p-c)}$$

where p is the semi-perimeter of the triangle $p = \frac{1}{2}(a + b + c)$.




```
1  import java.util.Scanner;
2
3  /*-
4   * MethodExample009
5   * Heron's Formula for the Area of a Triangle
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class MethodExample009 {
11
12     public static double distance(double x1, double y1, double x2, double y2) {
13         return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
14     }
15
16     public static double heron(double a, double b, double c) {
17         double p = (a + b + c) / 2;
18         return Math.sqrt(p * (p - a) * (p - b) * (p - c));
19     }
20
21     public static void main(String[] args) {
22         Scanner keyboard = new Scanner(System.in);
23         System.out.print("Input point 1 as x y: ");
24         double x1 = keyboard.nextDouble();
25         double y1 = keyboard.nextDouble();
```

```
26          System.out.print("Input point 2 as x y: ");
27          double x2 = keyboard.nextDouble();
28          double y2 = keyboard.nextDouble();
29          System.out.print("Input point 3 as x y: ");
30          double x3 = keyboard.nextDouble();
31          double y3 = keyboard.nextDouble();
32          keyboard.close();
33
34          double a = distance(x1, y1, x2, y2);
35          double b = distance(x1, y1, x3, y3);
36          double c = distance(x3, y3, x2, y2);
37
38          System.out.println("The area of the triangle is " + heron(a, b, c));
39      }
40  }
```

Program Output **MethodExample009.java Run #1**

```
Input point 1 as x y: 1 2
Input point 2 as x y: 3 7
Input point 3 as x y: -3 5
The area of the triangle is 13.000000000000007
```

Program Output **MethodExample009.java Run #2**

```
Input point 1 as x y: 1 1
Input point 2 as x y: 2 2
Input point 3 as x y: 3 3
The area of the triangle is 0.0
```

6.3.8 Guessing Game Example

This is the guessing game example from the previous chapter. If you recall the guessing game code was fairly long with a lot of nested structures, and hence was a bit hard to follow. In this example we extract the game portion and make it a method. This way the method handles a single play of the game and the main handles the score-keeping and repeating the game play. In the method, instead of printing out winners and losers, it returns true if the player wins and false if the computer wins.

```
1   import java.util.Random;
2   import java.util.Scanner;
3
4   /*-
5    * MethodExample010
6    * Guessing Game Example 1
7    * Author:  Don Spickler
8    * Date: 3/7/2011
9    */
10
11  public class MethodExample010 {
12
13      // returns true if the user wins.
14      public static boolean GuessingGame() {
15          Random generator = new Random();
16          Scanner keyboard = new Scanner(System.in);
```

```
17
18          int answer = generator.nextInt(100) + 1;
19          int numGuesses = 1;
20          int guess = 0;
21
22          while ((numGuesses <= 7) && (guess != answer)) {
23              System.out.print("Guess a number: ");
24              guess = keyboard.nextInt();
25
26              if (numGuesses < 7) {
27                  if (guess < answer) {
28                      System.out.println("Higher...");
29                  } else if (guess > answer) {
30                      System.out.println("Lower...");
31                  } else {
32                      System.out.println("You Win");
33                      return true;
34                  }
35              } else {
36                  if (guess == answer) {
37                      System.out.println("You Win");
38                      return true;
39                  } else {
40                      System.out.println("I Win, the number was " + answer);
41                      return false;
42                  }
43              }
44              numGuesses++;
45          }
46          return false; // Never happens but the compiler needs it.
47      }
48
49      public static void main(String[] args) {
50          Scanner keyboard = new Scanner(System.in);
51
52          String playAgain = "Y";
53          int userWins = 0;
54          int computerWins = 0;
55
56          while (playAgain.compareToIgnoreCase("Y") == 0) {
57              boolean youWin = GuessingGame();
58              if (youWin) {
59                  userWins++;
60              } else {
61                  computerWins++;
62              }
63
64              System.out.print("Would you like to play another game? (Y/N): ");
65              playAgain = keyboard.next();
66          }
67
68          System.out.println("Final Score: You " + userWins + "   Computer " + computerWins)
                ;
69          keyboard.close();
70      }
71  }
```

Program Output — MethodExample010.java

```
Guess a number: 50
Higher...
Guess a number: 75
Higher...
```

```
Guess a number: 87
Lower...
Guess a number: 81
Higher...
Guess a number: 84
You Win
Would you like to play another game? (Y/N): y
Guess a number: 1
Higher...
Guess a number: 2
Higher...
Guess a number: 3
Higher...
Guess a number: 4
Higher...
Guess a number: 5
Higher...
Guess a number: 6
Higher...
Guess a number: 7
I Win, the number was 21
Would you like to play another game? (Y/N): n
Final Score: You 1   Computer 1
```

6.3.9 Guessing Game Example #2

There is only one difference between this example and the last. In the last example we called the `GuessingGame` method inside an assignment statement, we assigned the boolean variable `youWin` to the returned value of the method.

```
boolean youWin = GuessingGame();
if (youWin) {
    userWins++;
} else {
    computerWins++;
}
```

In this example we do not assign the return value to a variable, we simply put the call as the condition in the if statement.

```
if (GuessingGame())
    userWins++;
else
    computerWins++;
```

So when the if statement is hit the program evaluates the value of `GuessingGame`, that is, it runs the method and the user plays the game. When the game is over, `GuessingGame` has the value of true or false which is then used by the if statement to keep score. It is interesting that the user is playing the game inside an if condition.

```
1 import java.util.Random;
```

```
2  import java.util.Scanner;
3
4  /*-
5   * MethodExample011
6   * Guessing Game Example 2
7   * Author:  Don Spickler
8   * Date: 3/7/2011
9   */
10
11 public class MethodExample011 {
12
13     // returns true if the user wins.
14     public static boolean GuessingGame() {
15         Random generator = new Random();
16         Scanner keyboard = new Scanner(System.in);
17
18         int answer = generator.nextInt(100) + 1;
19         int numGuesses = 1;
20         int guess = 0;
21
22         while ((numGuesses <= 7) && (guess != answer)) {
23             System.out.print("Guess a number: ");
24             guess = keyboard.nextInt();
25
26             if (numGuesses < 7) {
27                 if (guess < answer)
28                     System.out.println("Higher...");
29                 else if (guess > answer)
30                     System.out.println("Lower...");
31                 else {
32                     System.out.println("You Win");
33                     return true;
34                 }
35             } else {
36                 if (guess == answer) {
37                     System.out.println("You Win");
38                     return true;
39                 } else {
40                     System.out.println("I Win, the number was " + answer);
41                     return false;
42                 }
43             }
44             numGuesses++;
45         }
46         return false; // Never happens but the compiler needs it.
47     }
48
49     public static void main(String[] args) {
50         Scanner keyboard = new Scanner(System.in);
51
52         String playAgain = "Y";
53         int userWins = 0;
54         int computerWins = 0;
55
56         while (playAgain.compareToIgnoreCase("Y") == 0) {
57             if (GuessingGame())
58                 userWins++;
59             else
60                 computerWins++;
61
62             System.out.print("Would you like to play another game? (Y/N): ");
63             playAgain = keyboard.next();
64         }
65
```

```
66          keyboard.close();
67          System.out.println("Final Score: You " + userWins + "   Computer " + computerWins)
                ;
68      }
69  }
```

Program Output **MethodExample011.java**

```
Guess a number: 50
Higher...
Guess a number: 75
Higher...
Guess a number: 87
Higher...
Guess a number: 93
Lower...
Guess a number: 90
Lower...
Guess a number: 88
Higher...
Guess a number: 89
You Win
Would you like to play another game? (Y/N): y
Guess a number: 1
Higher...
Guess a number: 2
Higher...
Guess a number: 3
Higher...
Guess a number: 4
Higher...
Guess a number: 5
Higher...
Guess a number: 6
Higher...
Guess a number: 7
I Win, the number was 61
Would you like to play another game? (Y/N): n
Final Score: You 1   Computer 1
```

6.4 External Class Methods

In the next chapter we will be looking at *objects* in Java. An object is a structure that stores data and has methods that operate on that data. It is a very convenient way to structure your program and it can make many tasks much easier to do. In this set of examples we will be going half way toward the object idea. Here we will be putting methods into external class structures.

In our previous examples all of the methods were in the project file we always create and contains the main. Alternatively, we could create another class structure, which will be in a separate file, that will hold our methods.

To create a new class in Eclipse select File > New > Class from the menu or the toolbar. At this point a dialog box will appear, the same one we use when creating

the main. Type in the name of the class you wish to use. This name cannot be any name that was previously used in the Java project. Make sure that all of the check boxes at the bottom are unchecked, you do not want to create another main, you already have one. Also make sure that the Modifiers are set set to Public and click Finish. This will create a new file with the name you gave it and Eclipse will create a shell for the class structure. For example, if we had used the name Triangle, then Eclipse would create the file `Triangle.java` and its contents would be,

```
public class Triangle {

}
```

If you get something other than this, especially if you see another main method, then most likely your settings in the dialog box are incorrect. In this case, delete the class you just created and create a new one with the correct dialog settings.

Inside this new class you can put in as many methods as you would like. For example, in the first example below we created two new classes for the project, one called Circle and the other called Rectangle. Inside the Circle class we created three methods, Area, Circumference, and Perimeter.

```
public class Circle {

    /*-
     * Circle
     * External Methods for Circle Properties
     * Author:  Don Spickler
     * Date: 3/7/2011
     */

    public static double Area(double radius) {
        return Math.PI * radius * radius;
    }

    public static double Circumference(double radius) {
        return 2 * Math.PI * radius;
    }

    public static double Perimeter(double radius) {
        return Circumference(radius);
    }
}
```

Inside the Rectangle class we created two methods, Area, and Perimeter.

```
public class Rectangle {

    /*-
     * Rectangle
     * External Methods for Rectangle Properties
     * Author:  Don Spickler
     * Date: 3/7/2011
     */

    public static double Area(double length, double width) {
        return length * width;
    }

    public static double Perimeter(double length, double
       width) {
        return 2 * length + 2 * width;
    }
}
```

Notice that the names of the methods are the same, this is valid since they are in different class structures. You can reuse method names when they are in different class structures. In fact, you can use the same name for different methods inside the same class as long as the parameter lists are different, this is called method overloading or function overloading. We will look at that later on in the notes.

When you segment a program in this way the method calls are a little different. Notice the Perimeter method in the Circle class. It simply calls the Circumference method and returns its value. The syntax of the call is exactly the same as before. When you call a method in the class, you simply need the method name and any parameters the method needs. On the other hand, if you call a method that is in a different class then you also need to have the class name on the front followed by a period. So for example, to call the Area method in the Circle class from the main we would use the syntax,

```
Circle.Area(rad)
```

In the examples in this section, each class is in its own file and the name of the file is exactly the name of the class, this is a requirement in Java and frankly it is not a bad system. You may find putting each class structure in its own file with the same name when you program in other languages, like C++.

6.4.1 External Class Method Example

```java
public class Circle {

    /*-
     * Circle
     * External Methods for Circle Properties
     * Author:  Don Spickler
     * Date: 3/7/2011
     */

    public static double Area(double radius) {
        return Math.PI * radius * radius;
    }

    public static double Circumference(double radius) {
        return 2 * Math.PI * radius;
    }

    public static double Perimeter(double radius) {
        return Circumference(radius);
    }
}
```

```java
public class Rectangle {

    /*-
     * Rectangle
     * External Methods for Rectangle Properties
     * Author:  Don Spickler
     * Date: 3/7/2011
     */

    public static double Area(double length, double width) {
        return length * width;
    }

    public static double Perimeter(double length, double width) {
        return 2 * length + 2 * width;
    }
}
```

```java
import java.util.Scanner;

/*-
 * MethodExample012
 * External Class Methods Example
 * Author:  Don Spickler
 * Date: 3/7/2011
 */

public class MethodExample012 {

    public static int menu() {
        Scanner keyboard = new Scanner(System.in);
        int menuOption = 0;
        while (menuOption < 1 || menuOption > 3) {
            System.out.println("Please select from the following menu:");
            System.out.println("1. Rectangle Properties");
            System.out.println("2. Circle Properties");
            System.out.println("3. Exit");
            System.out.println();
            System.out.print("Selection: ");
            menuOption = keyboard.nextInt();
            System.out.println();

```

```
25              if (menuOption < 1 || menuOption > 3) {
26                  System.out.println("Invalid Menu Selection!");
27                  System.out.println("Please make another selection.");
28                  System.out.println();
29              }
30          }
31          return menuOption;
32      }
33
34      public static void main(String[] args) {
35          Scanner keyboard = new Scanner(System.in);
36
37          int menuOption = 0;
38          while (menuOption != 3) {
39              menuOption = menu();
40
41              if (menuOption == 1) {
42                  System.out.print("Input the width of the rectangle: ");
43                  double width = keyboard.nextDouble();
44                  System.out.print("Input the height of the rectangle: ");
45                  double height = keyboard.nextDouble();
46                  System.out.println("The area of the rectangle is " + Rectangle.Area(width,
                        height));
47                  System.out.println("The perimeter of the rectangle is " + Rectangle.
                        Perimeter(width, height));
48              } else if (menuOption == 2) {
49                  System.out.print("Input the radius of the circle: ");
50                  double rad = keyboard.nextDouble();
51                  System.out.println("The area of the circle is " + Circle.Area(rad));
52                  System.out.println("The circumference of the circle is " + Circle.
                        Circumference(rad));
53              }
54              System.out.println();
55          }
56
57          keyboard.close();
58      }
59  }
```

Program Output — MethodExample012.java

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 5

Invalid Menu Selection!
Please make another selection.

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 1

Input the width of the rectangle: 3
Input the height of the rectangle: 5
The area of the rectangle is 15.0
The perimeter of the rectangle is 16.0
```

```
Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 2

Input the radius of the circle: 7
The area of the circle is 153.93804002589985
The circumference of the circle is 43.982297150257104

Please select from the following menu:
1. Rectangle Properties
2. Circle Properties
3. Exit

Selection: 3
```

6.4.2 External Class Method Example #2

This is simply another example of external methods. As we mentioned in the introduction, these examples are half way to the concept of the object. In the next chapter we will be revising this example into an object so you may wish to read through the syntax carefully to notice the differences between using external methods and using an object.

```
1  public class Triangle {
2
3      /*-
4       * Triangle
5       * External Methods for Triangle Properties
6       * Author:  Don Spickler
7       * Date: 3/7/2011
8       */
9
10     public static double Area(double a, double b, double c) {
11         double p = (a + b + c) / 2;
12         return Math.sqrt(p * (p - a) * (p - b) * (p - c));
13     }
14
15     public static double Perimeter(double a, double b, double c) {
16         return a + b + c;
17     }
18
19     public static boolean isRight(double a, double b, double c) {
20         boolean righttri = false;
21         if (a * a + b * b == c * c)
22             righttri = true;
23
24         if (a * a + c * c == b * b)
25             righttri = true;
26
27         if (c * c + b * b == a * a)
28             righttri = true;
29
30         return righttri;
31     }
32
33     public static boolean isTriangle(double a, double b, double c) {
34         boolean tri = true;
```

```
35
36          // Find longest leg
37          double longleg = a;
38          if (b > longleg)
39              longleg = b;
40          if (c > longleg)
41              longleg = c;
42
43          // Check if the two shorter legs do add up to the length of the
44          // longest leg.
45          if (a + b + c - longleg <= longleg)
46              tri = false;
47
48          return tri;
49      }
50  }
```

```
1   import java.util.Scanner;
2
3   /*-
4    * MethodExample013
5    * External Class Methods Example
6    * Author:  Don Spickler
7    * Date: 3/7/2011
8    */
9
10  public class MethodExample013 {
11
12      public static void main(String[] args) {
13          Scanner keyboard = new Scanner(System.in);
14          System.out.print("Input the lengths of the sides of the triangle, a b c: ");
15          double a = keyboard.nextDouble();
16          double b = keyboard.nextDouble();
17          double c = keyboard.nextDouble();
18
19          if (Triangle.isTriangle(a, b, c)) {
20              System.out.println("Area = " + Triangle.Area(a, b, c));
21              System.out.println("Perimeter = " + Triangle.Perimeter(a, b, c));
22              System.out.println("Right Triangle = " + Triangle.isRight(a, b, c));
23          } else
24              System.out.println("This is not a triangle.");
25
26          keyboard.close();
27      }
28  }
```

Program Output — MethodExample013.java Run #1

```
Input the lengths of the sides of the triangle, a b c: 3 4 5
Area = 6.0
Perimeter = 12.0
Right Triangle = true
```

Program Output — MethodExample013.java Run #2

```
Input the lengths of the sides of the triangle, a b c: 3 4 2.7
Area = 4.049351028251316
Perimeter = 9.7
Right Triangle = false
```

Program Output — MethodExample013.java Run #3

```
Input the lengths of the sides of the triangle, a b c: 3 4 8
This is not a triangle.
```

Chapter 7

Objects

7.1 Introduction

Object Oriented Programming (OOP) is a relatively new style of programming where you think about segments of code as physical objects. With these objects you think about their attributes, this will be data that the object stores, and you think about what you can do with them, this is the methods the object will have. For example, say we consider a physical object such as a car. Think about the attributes of the car,

1. Make
2. Model
3. Color
4. Number of doors
5. Size of the engine
6. Number of seats
7. Type of transmission
8. Does it have a sunroof?
9. Is it a convertible?
10. Licence plate number

I am sure that you could add many more items to this list. These attributes would be stored in the object as data members. For example, the make, model, and color would all be strings and the number of doors would be an integer. If it has a sunroof or if it is a convertible would be boolean variables and so on. Now think about what you do with a car. Obviously, you drive it, so drive would be a method that is stored with the object. In addition, you wash a car, take it in for service, put gas in it, and so on. All of these activities that are done to the object are methods inside the object.

Now a car is a fairly complex object so lets start with something a bit less complicated. We will start with a triangle. A triangle has three sides all with positive length. So our attributes, that is the data, is fairly easy. We need to store three numbers, one for each side of the triangle. Since these can be decimal type numbers we should use a float or a double to store them. Now lets think about what you can do with a triangle. You can find its area, find its perimeter, determine if it is a right triangle, and frankly determine if it is in fact a triangle. For example, a triangle with sides length 3, 4, and 5 is a right triangle, one with sides length 3, 4, and 6 is a triangle but not a right triangle, and one with sides length 3, 4, and 8 is not a triangle. Of course, there are many other things you can do but we will stick with these few things for now.

Creating an object in Java is essentially like creating the external methods as we did in the previous chapter. We create a new class structure with the name of the object we want. For example, Triangle would be a good one here. As with the external methods we uncheck all of the check boxes and keep the modifier public. As before you should see a new file Triangle.java with the following code written for you.

```
public class Triangle {

}
```

The data members will be three doubles, so we simply add the following three lines to the class structure.

```
private double a;
private double b;
private double c;
```

These can really go anywhere but I tend to put the data items at the top of the class and put the methods below them. You will notice that we have a new reserved word `private` in front of the declarations. This is called an access modifier. We will not get deeply into what access modifiers are but we will give a shore explanation of the syntax. There are three types of access you can give to a data member or a method in an object (that is, a class structure), private, protected, and public. In these note we will only use private and public. The protected access is used for object

inheritance and is beyond the scope of these notes. Private access means that only the methods inside the class can gain access to them and hence use them. If we tried to get the value of a from inside the main we would get an access error. Public access means that anything can have access to the data or method. So if these were set to public instead of private the main could gain access to them. In this case the main could extract the value of the variables as well as assign them to other values.

You may wonder why we would want to place such restricted assess on the data in an object. The main reason is for data validity, making sure that the data is not bad data. For example, if we made these variables public than in the main, we could set one of them to a negative number, hence invalidating the length of a side. With private data members we can construct accessor methods that can be called from outside the class and update the data values, these methods cam also check data validity before assigning the value to the data member and make adjustments if necessary.

In general, it is best to have your data members have private access and for your methods to have public access. There will be times when you want to make a data member public and a method private, and in some cases it is perfectly reasonable to do that. For now we will stick with data members having private access and methods having public access. If we skip ahead to the next example we see the syntax for the Area and Perimeter methods.

```
public double Area() {
    double p = (a + b + c) / 2;
    return Math.sqrt(p * (p - a) * (p - b) * (p - c));
}

public double Perimeter() {
    return a + b + c;
}
```

If we compare these to the Area and Perimeter methods from the external method example from the previous chapter we notice one significant difference, there is no parameter list. With objects, the data is stored in the object itself, so there is no need to bring the values in as parameters, they are already there. Also notice that we have removed the reserved word `static`. The `static` reserved word allowed us, in the last chapter, to do the call `Circle.Area(rad)`. That is, call a method using the class name. We will not be needing to do that here since we will be working with objects a bit differently.

Two more additions that we need to make to make the object complete. First are the accessor functions we discussed above. In Java, these are called getters and setters since they start with the word get or the word set. The getters will extract data values and the setters will set data values. It is the job of the setters to validate

the data before doing an assignment statement. You do not need to have getters or setters in your class structures but you will probably want some of them. Also, you do not need to start the method name with get or set but this is the convention in Java and it is a good practice to make your code more readable and self-documenting. In our example, we have three getters, one for each side of the triangle, and we have no setters.

```
public double getSide1() {
    return a;
}

public double getSide2() {
    return b;
}

public double getSide3() {
    return c;
}
```

The final thing we need to add is the constructor. Every object needs to have a constructor and the constructor is the method that is called when we first create an instance of the object. If we look ahead to the next example, we see the following line that declares a triangle object, that is, an instance of a triangle.

```
Triangle tri = new Triangle(a, b, c);
```

Note that it looks a lot like the creation of our scanners that we have been linking to the keyboard. We start with the name of the class `Triangle` followed by a variable name, `tri` followed by an assignment to a new triangle. For this to work, the triangle class must have a method named the same as the class, `Triangle`, that takes in three parameters. This is called a constructor and the one for the triangle class is below.

```
public Triangle(double s1, double s2, double s3) {
    a = s1;
    b = s2;
    c = s3;
}
```

Note that it has the same name as the class and that it does not have a return type, or void. When we declare an object the constructor method is run. So in our example, the triangle `tri` is created and the values of a, b, and c from the main are stored in the data of the triangle. To invoke a method on the triangle `tri`, we start with the variable name `tri`, then a period, then the name of the method we wish to run. In each case, the method is run on the data that we stored in `tri`.

```
if (tri.isTriangle()) {
     System.out.println("Area = " + tri.Area());
     System.out.println("Perimeter = " + tri.Perimeter());
     System.out.println("Right Triangle = " + tri.isRight());
} else
     System.out.println("This is not a triangle.");
```

7.2 Triangle Class Example

```
1  /*-
2   * Triangle
3   * Triangle class for storing the lengths of the sides of a triangle
4   * and methods for accessing data and calculating properties.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Triangle {
10     // Data Members
11
12     private double a;
13     private double b;
14     private double c;
15
16     // Methods
17
18     // Constructor
19     public Triangle(double s1, double s2, double s3) {
20         a = s1;
21         b = s2;
22         c = s3;
23     }
24
25     // Accessor Methods
26     public double getSide1() {
27         return a;
28     }
29
30     public double getSide2() {
31         return b;
32     }
33
34     public double getSide3() {
35         return c;
36     }
37
38     // Other Methods
39     public double Area() {
40         double p = (a + b + c) / 2;
41         return Math.sqrt(p * (p - a) * (p - b) * (p - c));
42     }
43
44     public double Perimeter() {
45         return a + b + c;
46     }
47
48     public boolean isRight() {
```

```
49          boolean righttri = false;
50          if (a * a + b * b == c * c)
51              righttri = true;
52
53          if (a * a + c * c == b * b)
54              righttri = true;
55
56          if (c * c + b * b == a * a)
57              righttri = true;
58
59          return righttri;
60      }
61
62      public boolean isTriangle() {
63          boolean tri = true;
64
65          // Find longest leg
66          double longleg = a;
67          if (b > longleg)
68              longleg = b;
69          if (c > longleg)
70              longleg = c;
71
72          // Check if the two shorter legs do add up to the length of the
73          // longest leg.
74          if (a + b + c - longleg <= longleg)
75              tri = false;
76
77          return tri;
78      }
79  }
```

```
1   import java.util.Scanner;
2
3   /*-
4    * ObjectExample001
5    * Basic example of object use.
6    * Author:  Don Spickler
7    * Date: 3/7/2011
8    */
9
10  public class ObjectExample001 {
11      public static void main(String[] args) {
12          Scanner keyboard = new Scanner(System.in);
13          System.out.print("Input the lengths of the sides of the triangle, a b c: ");
14          double a = keyboard.nextDouble();
15          double b = keyboard.nextDouble();
16          double c = keyboard.nextDouble();
17          Triangle tri = new Triangle(a, b, c);
18          keyboard.close();
19
20          if (tri.isTriangle()) {
21              System.out.println("Area = " + tri.Area());
22              System.out.println("Perimeter = " + tri.Perimeter());
23              System.out.println("Right Triangle = " + tri.isRight());
24          } else
25              System.out.println("This is not a triangle.");
26      }
27  }
```

Program Output — ObjectExample001.java Run #1

```
Input the lengths of the sides of the triangle, a b c: 3 4 5
```

```
Area = 6.0
Perimeter = 12.0
Right Triangle = true
```

Program Output **ObjectExample001.java Run #2**

```
Input the lengths of the sides of the triangle, a b c: 3 4 6
Area = 5.332682251925386
Perimeter = 13.0
Right Triangle = false
```

Program Output **ObjectExample001.java Run #3**

```
Input the lengths of the sides of the triangle, a b c: 3 4 8
This is not a triangle.
```

7.3 Nifty Sequence Class Example

In this example we converted the $3n+1$ sequence into an object. The data members for the object are the starting number, the length of the sequence, and the sequence itself. The starting number and the length are both integers. The sequence itself could be stored as a list of integers but since we have not looked at arrays or arraylists we will use a string to hold the sequence. An array or arraylist would be a better structure and we will look at these later on. The constructor sets the starting number and then calls the `createSequence` method to create the sequence and determine the length of the sequence. The other three methods are simply accessor methods.

Note that the `createSequence` method is private. This means that only methods inside the `NiftySequence` class can call it. There is really no reason that we could not make this public, it is just that there is no reason for an outside method to call it since the sequence was created during construction.

```
1  /*-
2   * NiftySequence
3   * NiftySequence class for creating and storing the 3n+1 sequence and some
4   * of its properties.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class NiftySequence {
10     // Data Members
11     private int startNum;
12     private int len;
13     private String seqstr;
14
15     // Constructor
16     public NiftySequence(int num) {
17         startNum = num;
18         createSequence();
19     }
20
21     // Accessor Methods
```

```
22      public int start() {
23          return startNum;
24      }
25
26      public int length() {
27          return len;
28      }
29
30      public String toString() {
31          return seqstr;
32      }
33
34      // private methods
35      private void createSequence() {
36          int n = startNum;
37          len = 1;
38          seqstr = "" + n + "   ";
39          while (n > 1) {
40              if (n % 2 == 0) {
41                  n = n / 2;
42              } else {
43                  n = 3 * n + 1;
44              }
45              len++;
46              seqstr = seqstr + n + "   ";
47          }
48      }
49  }
```

```
1   import java.util.Scanner;
2
3   /*-
4    * ObjectExample002
5    * Making the Nifty Sequence into an object.
6    * Author:  Don Spickler
7    * Date: 3/7/2011
8    */
9
10  public class ObjectExample002 {
11
12      public static void main(String[] args) {
13          Scanner keyboard = new Scanner(System.in);
14          System.out.print("Input a number: ");
15          int n = keyboard.nextInt();
16          keyboard.close();
17
18          if (n > 0) {
19              NiftySequence seq = new NiftySequence(n);
20              System.out.println("Start: " + seq.start());
21              System.out.println("Sequence: " + seq.toString());
22              System.out.println("Length: " + seq.length());
23          }
24      }
25  }
```

Program Output — ObjectExample002.java

```
Input a number: 25
Start: 25
Sequence: 25  76  38  19  58  29  88  44  22  11  34  17  52  26  13  40  20  10  5  16  8
      4  2  1
Length: 24
```

7.4 Employee Class Example

In our next example our object is an employee. An employee will store the employee's name, hourly wage and the number of hours worked for the week. The class also has a full set of getters and setters as well as the constructor and a method for calculating pay. The pay calculation is simply the hours worked times the wage and we give time and a half for overtime, any hour in excess of 40.

```
1  /*-
2   * Employee
3   * Employee class stores name, wage, and hours worked data and calculates
4   * a weekly pay.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Employee {
10
11     // Data Members
12     private String name;
13     private double wage;
14     private double hours_worked;
15
16     // Constructor
17     public Employee(String n, double w, double hw) {
18         name = n;
19         if (w > 0)
20             wage = w;
21         else
22             wage = 0;
23
24         if (hw > 0)
25             hours_worked = hw;
26         else
27             hours_worked = 0;
28     }
29
30     // Accessor Methods
31     public String getName() {
32         return name;
33     }
34
35     public double getWage() {
36         return wage;
37     }
38
39     public double getHoursWorked() {
40         return hours_worked;
41     }
42
43     public void setName(String n) {
44         name = n;
45     }
46
47     public void setWage(double w) {
48         if (w > 0)
49             wage = w;
50         else
51             wage = 0;
52     }
53
```

```
54      public void setHoursWorked(double hw) {
55          if (hw > 0)
56              hours_worked = hw;
57          else
58              hours_worked = 0;
59      }
60
61      // Calculation Methods
62      public double pay() {
63          double payment = 0;
64          if (hours_worked > 40)
65              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
66          else
67              payment = wage * hours_worked;
68
69          return payment;
70      }
71  }
```

```
1   import java.util.Scanner;
2
3   /*-
4    * ObjectExample003
5    * Basic example of employee records.
6    * Author:  Don Spickler
7    * Date: 3/7/2011
8    */
9
10  public class ObjectExample003 {
11      public static void main(String[] args) {
12          Scanner keyboard = new Scanner(System.in);
13
14          System.out.print("Input employee name: ");
15          String name = keyboard.nextLine();
16          System.out.print("Input employee wage: ");
17          double wage = keyboard.nextDouble();
18          System.out.print("Input hours worked: ");
19          double hours = keyboard.nextDouble();
20          keyboard.close();
21
22          Employee emp = new Employee(name, wage, hours);
23
24          System.out.println();
25          System.out.println("Employee Record");
26          System.out.println("Name: " + emp.getName());
27          System.out.println("Wage: " + emp.getWage());
28          System.out.println("Hours Worked: " + emp.getHoursWorked());
29          System.out.println("Pay: " + emp.pay());
30      }
31  }
```

Program Output — ObjectExample003.java Run #1

```
Input employee name: Don Spickler
Input employee wage: 10.25
Input hours worked: 45

Employee Record
Name: Don Spickler
Wage: 10.25
Hours Worked: 45.0
Pay: 486.875
```

Program Output **ObjectExample003.java Run #2**

```
Input employee name: Jane Doe
Input employee wage: 12.43
Input hours worked: 37

Employee Record
Name: Jane Doe
Wage: 12.43
Hours Worked: 37.0
Pay: 459.90999999999997
```

Program Output **ObjectExample003.java Run #3**

```
Input employee name: John Doe
Input employee wage: 15.00
Input hours worked: -34

Employee Record
Name: John Doe
Wage: 15.0
Hours Worked: 0.0
Pay: 0.0
```

7.5 Employee Class Example #2

Just like any other data type, you can have more than one instance of the data type. So we can have as many employees as as we would like. Each instance of the employee has its own name, wage, and hours worked. So when we call `emp1.pay()` it calculates the pay for `emp1` using the wage and hours stored for `emp1` and when we call `emp2.pay()` it calculates the pay for `emp2` using the wage and hours stored for `emp2`.

```
1  /*-
2   * Employee
3   * Employee class stores name, wage, and hours worked data and calculates
4   * a weekly pay.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Employee {
10
11     // Data Members
12     private String name;
13     private double wage;
14     private double hours_worked;
15
16     // Constructor
17     public Employee(String n, double w, double hw) {
18         name = n;
19         if (w > 0)
20             wage = w;
```

```
21          else
22              wage = 0;
23
24          if (hw > 0)
25              hours_worked = hw;
26          else
27              hours_worked = 0;
28      }
29
30      // Accessor Methods
31      public String getName() {
32          return name;
33      }
34
35      public double getWage() {
36          return wage;
37      }
38
39      public double getHoursWorked() {
40          return hours_worked;
41      }
42
43      public void setName(String n) {
44          name = n;
45      }
46
47      public void setWage(double w) {
48          if (w > 0)
49              wage = w;
50          else
51              wage = 0;
52      }
53
54      public void setHoursWorked(double hw) {
55          if (hw > 0)
56              hours_worked = hw;
57          else
58              hours_worked = 0;
59      }
60
61      // Calculation Methods
62      public double pay() {
63          double payment = 0;
64          if (hours_worked > 40)
65              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
66          else
67              payment = wage * hours_worked;
68
69          return payment;
70      }
71  }
```

```
1   public class ObjectExample004 {
2
3       /*-
4        * ObjectExample004
5        * Basic example of employee records.
6        * Author:  Don Spickler
7        * Date: 3/7/2011
8        */
9
10      public static void PrintRecord(Employee employee) {
11          System.out.println("Name: " + employee.getName());
12          System.out.println("Wage: " + employee.getWage());
```

```
13          System.out.println("Hours Worked: " + employee.getHoursWorked());
14          System.out.printf("Pay: %.2f \n", employee.pay());
15          System.out.println();
16      }
17
18      public static void main(String[] args) {
19          Employee emp1 = new Employee("Don Spickler", 12.5, 52);
20          Employee emp2 = new Employee("John Doe", 23.54, 37);
21          Employee emp3 = new Employee("Jane Q. Public", 15.47, 48);
22          Employee emp4 = new Employee("Rip Torn", 30, 25);
23
24          System.out.println("Payment List");
25          PrintRecord(emp1);
26          PrintRecord(emp2);
27          PrintRecord(emp3);
28          PrintRecord(emp4);
29
30          double payout = emp1.pay() + emp2.pay() + emp3.pay() + emp4.pay();
31          System.out.println("Company Payout: " + payout);
32      }
33  }
```

Program Output — ObjectExample004.java

```
Payment List
Name: Don Spickler
Wage: 12.5
Hours Worked: 52.0
Pay: 725.00

Name: John Doe
Wage: 23.54
Hours Worked: 37.0
Pay: 870.98

Name: Jane Q. Public
Wage: 15.47
Hours Worked: 48.0
Pay: 804.44

Name: Rip Torn
Wage: 30.0
Hours Worked: 25.0
Pay: 750.00

Company Payout: 3150.42
```

Chapter 8

Exceptions

8.1 Introduction

An exception is a run-time error. It occurs when you ask the computer to do something that it cannot do. For example, if you try to divide by 0 or if you are entering an integer from the keyboard and type in a string instead. Java has a nice way for the programmer to catch an exception, that is, stop a run-time error from crashing the program.

8.2 Catching Exceptions

Catching an exception is fairly easy. Java has a structure called a try-catch block. Start with the reserved word `try` then in a block of code put the portion of the program that might cause an exception. After the block use the reserved word `catch` followed by the exception type and a variable name for the exception information in parentheses. Then put in a block of code to be done if an exception did occur.

```
try {
    // Code that might cause an exception.
} catch ( <Exception Type> <variable>) {
    // Code to recover from the error.
}
```

So if an exception happens in the try block then the code in the catch block will be run. In Java there are many different types of exceptions, version 1.7 of Java has 74 different types of exceptions. The try-catch block will only catch the type of exception that is listed in the parentheses. Fortunately, there is a way to catch multiple exceptions and we can even use `Exception` for the type to catch all of

them.

8.2.1 Division By 0 Example

This first example does not catch any exceptions, so if a runtime error occurs the program will crash.

```
1  import java.util.Scanner;
2
3  /*-
4   * TryCatch001
5   * Simple example to allow the user to see the result of division by 0.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class TryCatch001 {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         System.out.print("Numerator = ");
15         int num = keyboard.nextInt();
16         System.out.print("Denominator = ");
17         int den = keyboard.nextInt();
18
19         System.out.println(num + "/" + den + " = " + num / den);
20         keyboard.close();
21     }
22 }
```

Program Output **TryCatch001.java Run #1**

```
Numerator = 5
Denominator = 2
5/2 = 2
```

Program Output **TryCatch001.java Run #2**

```
Numerator = 1
Denominator = 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at TryCatch001.main(TryCatch001.java:19)
```

Notice in the exception output you have the type of exception, a message on what the error was `/ by zero`, what file and method it occurred in, and the line number of the error (19). This output is called a stack trace, if the error is deeper in the program this trace will be longer. Not all programming systems give you this much information on where the run-time error happens. For example, in C and C++ you need to run a separate debugger program to find some runtime errors. Even then, it might not tell you exactly what line of code caused the error.

8.2.2 Division By 0 Exception Catch Example

Notice in the previous example that the type of exception that occurred was an ArithmeticException. An ArithmeticException occurs when some type of formula cannot be evaluated. Dividing by 0 is one of them. Not all invalid formulas will cause a run-time error. For example, taking the logarithm of 0 will result in negative infinity. Notice in this example we are catching an exception of ArithmeticException type, any other type of exception will result in a program crash.

```
1  import java.util.Scanner;
2
3  /*-
4   * TryCatch002
5   * Simple example using a try-catch block to catch division by 0.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class TryCatch002 {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         System.out.print("Numerator = ");
15         int num = keyboard.nextInt();
16         System.out.print("Denominator = ");
17         int den = keyboard.nextInt();
18
19         try {
20             System.out.println(num + "/" + den + " = " + num / den);
21         } catch (ArithmeticException e) {
22             System.out.println("Division by 0!");
23         }
24
25         keyboard.close();
26     }
27 }
```

Program Output **TryCatch002.java Run #1**

```
Numerator = 7
Denominator = 3
7/3 = 2
```

Program Output **TryCatch002.java Run #2**

```
Numerator = 1
Denominator = 0
Division by 0!
```

8.2.3 Division By 0 Exception Catch Message Example

The variable that is after the exception type will hold information about the exception. In this example we simply use it to print out the exception message. Some exception

messages are fairly good and give you an idea what to look for in finding the error but others can be a bit cryptic and not helpful at all.

```
1  import java.util.Scanner;
2
3  /*-
4   * TryCatch003
5   * Simple example using a try-catch block to catch division by 0, and print error message.
6   * Author:  Don Spickler
7   * Date: 2/6/2011
8   */
9
10 public class TryCatch003 {
11     public static void main(String[] args) {
12         Scanner keyboard = new Scanner(System.in);
13
14         System.out.print("Numerator = ");
15         int num = keyboard.nextInt();
16         System.out.print("Denominator = ");
17         int den = keyboard.nextInt();
18
19         try {
20             System.out.println(num + "/" + den + " = " + num / den);
21         } catch (ArithmeticException e) {
22             System.out.println(e.getMessage());
23         }
24
25         keyboard.close();
26     }
27 }
```

Program Output **TryCatch003.java Run #1**

```
Numerator = 7
Denominator = 3
7/3 = 2
```

Program Output **TryCatch003.java Run #2**

```
Numerator = 1
Denominator = 0
/ by zero
```

8.2.4 Catching All Exceptions Example

As we pointed out above, we can catch all exceptions with the `Exception` type. With that said, one should be a little careful using this blanket exception handling. You could catch exceptions that you do not take care of in the catch block, which could lead to unexpected results later on in the program. In professional software, the programmers know what types of exceptions could occur in various segments of the code and they will catch only these types. If testing or crash reports come back to them they will catch the new exceptions that come up.

```
1  import java.util.Scanner;
2
```

```
3  /*-
4   * TryCatch004
5   * Simple example using a try-catch block to catch division by 0, and print error message.
6   * The difference here is that we catch all exceptions, not just the arithmetic ones.
7   * Author:  Don Spickler
8   * Date: 2/6/2011
9   */
10
11 public class TryCatch004 {
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14
15         System.out.print("Numerator = ");
16         int num = keyboard.nextInt();
17         System.out.print("Denominator = ");
18         int den = keyboard.nextInt();
19
20         try {
21             System.out.println(num + "/" + den + " = " + num / den);
22         } catch (Exception e) {
23             System.out.println(e.getMessage());
24         }
25
26         keyboard.close();
27     }
28 }
```

Program Output **TryCatch004.java Run #1**

```
Numerator = 7
Denominator = 3
7/3 = 2
```

Program Output **TryCatch004.java Run #2**

```
Numerator = 1
Denominator = 0
/ by zero
```

8.2.5 Catching All Exceptions Example #2

One thing to note about the above example is that the keyboard input is outside of the try-catch block. So if you run the above program and type in a string for the numerator or denominator the program will crash before it gets to the try-catch block. In this example we put the input inside the try-catch block. This is essentially putting the entire program inside the try block. This will keep the program from crashing but it is not very good programming style. Notice that if the input is invalid, that is, a string or decimal number, the message is simply null. This is not the most descriptive error message.

```
1  import java.util.Scanner;
2
3  /*-
4   * TryCatch005
```

```
 * Simple example using a try-catch block to catch division by 0, and print error message.
 * The difference here is that we catch all exceptions, not just the arithmetic ones.
 * Note the inclusion of the input lines in the try block.  This prevents a crash due
 * to improper inputs from the user.
 * Author:  Don Spickler
 * Date: 2/6/2011
 */

public class TryCatch005 {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        try {
            System.out.print("Numerator = ");
            int num = keyboard.nextInt();
            System.out.print("Denominator = ");
            int den = keyboard.nextInt();

            System.out.println(num + "/" + den + " = " + num / den);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        keyboard.close();
    }
}
```

Program Output **TryCatch005.java Run #1**

```
Numerator = 10
Denominator = 3
10/3 = 3
```

Program Output **TryCatch005.java Run #2**

```
Numerator = 7
Denominator = 0
/ by zero
```

Program Output **TryCatch005.java Run #3**

```
Numerator = 3.7
null
```

Program Output **TryCatch005.java Run #4**

```
Numerator = 8
Denominator = C
null
```

8.3 Throwing and Catching Exceptions

In this section we look at ways that the programmer can create their own exceptions and crash their own programs. On the surface you might think, "Cool" but in retro-

spect you may say, "Why would a programmer want to crash their own program?" That would be a good question since in most cases a programmer would like their program not to crash. Exceptions form a communication tool in Java. We could have written a method that, depending on the inputs, could result in unexpected behaviour of the program. For example, say we took the method for calculating the area of a rectangle. This method took in the height and the width and simply multiplied them and returned the result. If we let one of the inputs be negative the method would run fine but it would return a negative area. If this value was then used later on in the program it could result in strange answers. So one way to communicate to the method call that something strange happened would be to cause an exception. The call could be placed into a try-catch block to catch the exception and recover from the error. Another way to handle the negative area error would be to do validation checking on the results of each method call, or validate the data before sending it to the method. This approach tends to lead to a lot of validation code that usually misses a case or two. Using exceptions is actually a cleaner approach.

In Java you can create your own exception types but it is easier to simply use one of those that already exist. If you go further into programming in Java you may want to investigate this, you will create a subclass of the `Exception` class. In these examples we are throwing exceptions that deal with the parameters of a method, so we will be using the `IllegalArgumentException` class. Creating an exception is also called throwing an exception. To throw an exception you start with the reserved word `throw` then `new` (since this is an object), the name of the exception type and then any parameters that are needed. For the `IllegalArgumentException` the only parameters that are needed is a single string that carries the message for the error. For example,

```
throw new IllegalArgumentException("Argument must be
   positive.");
```

8.3.1 Throwing Exceptions Example

In this first example, if the range values are not both positive an exception will be thrown with the message "Starting value and ending value must be positive." and if the starting value is larger than the ending value an exception will be thrown with the message "Starting value must be less than or equal to ending value." Note that the main program does not catch any exceptions so if the method throws the exception the program will crash.

```
1 import java.util.Scanner;
2
3 /*-
4  * ThrowingExceptions
5  * Simple example showing how to use Java's exception structure to cause your own crashes.
6  * Author:  Don Spickler
```

```
 * Date: 2/6/2011
 */

public class ThrowingExceptions {

    public static void printSquares(int startingValue, int endingValue) {
        if ((startingValue <= 0) || (endingValue <= 0))
            throw new IllegalArgumentException("Starting value and ending value must be
                positive.");

        if (endingValue < startingValue)
            throw new IllegalArgumentException("Starting value must be less than or equal
                to ending value.");

        for (int i = startingValue; i <= endingValue; i++) {
            System.out.print(i * i + "  ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.print("Input start: ");
        int start = keyboard.nextInt();
        System.out.print("Input end: ");
        int end = keyboard.nextInt();

        printSquares(start, end);
        keyboard.close();
    }
}
```

Program Output — ThrowingExceptions.java Run #1

```
Input start: 4
Input end: 9
16  25  36  49  64  81
```

Program Output — ThrowingExceptions.java Run #2

```
Input start: -3
Input end: 7
Exception in thread "main" java.lang.IllegalArgumentException: Starting value and ending
    value must be positive.
    at ThrowingExceptions.printSquares(ThrowingExceptions.java:14)
    at ThrowingExceptions.main(ThrowingExceptions.java:33)
```

Program Output — ThrowingExceptions.java Run #3

```
Input start: 8
Input end: 4
Exception in thread "main" java.lang.IllegalArgumentException: Starting value must be less
     than or equal to ending value.
    at ThrowingExceptions.printSquares(ThrowingExceptions.java:17)
    at ThrowingExceptions.main(ThrowingExceptions.java:33)
```

Program Output — ThrowingExceptions.java Run #4

```
Input start: 7
Input end: -4
Exception in thread "main" java.lang.IllegalArgumentException: Starting value and ending
    value must be positive.
    at ThrowingExceptions.printSquares(ThrowingExceptions.java:14)
    at ThrowingExceptions.main(ThrowingExceptions.java:33)
```

8.3.2 Throwing and Catching Exceptions Example

This example is the same as the one above except that the main catches the exception and prints out the error message.

```
1  import java.util.Scanner;
2
3  /*-
4   * ThrowingExceptions2
5   * Simple example showing how to use Java's exception structure to cause your own crashes
6   * and catch them.
7   * Author:  Don Spickler
8   * Date: 2/6/2011
9   */
10
11 public class ThrowingExceptions2 {
12
13     public static void printSquares(int startingValue, int endingValue) {
14         if ((startingValue <= 0) || (endingValue <= 0))
15             throw new IllegalArgumentException("Starting value and ending value must be
                   positive.");
16
17         if (endingValue < startingValue)
18             throw new IllegalArgumentException("Starting value must be less than or equal
                   to ending value.");
19
20         for (int i = startingValue; i <= endingValue; i++) {
21             System.out.print(i * i + "  ");
22         }
23         System.out.println();
24     }
25
26     public static void main(String[] args) {
27         Scanner keyboard = new Scanner(System.in);
28         boolean expThrown = true;
29
30         while (expThrown) {
31             expThrown = false;
32             System.out.print("Input start: ");
33             int start = keyboard.nextInt();
34             System.out.print("Input end: ");
35             int end = keyboard.nextInt();
36
37             try {
38                 printSquares(start, end);
39             } catch (IllegalArgumentException e) {
40                 expThrown = true;
41                 System.out.println(e.getMessage());
42             }
43         }
44
45         keyboard.close();
46     }
```

```
47 }
```

Program Output **ThrowingExceptions2.java**

```
Input start: -3
Input end: 7
Starting value and ending value must be positive.
Input start: 7
Input end: 3
Starting value must be less than or equal to ending value.
Input start: 3
Input end: 7
9  16  25  36  49
```

8.3.3 Throwing and Catching All Exceptions Example

This example is the same as the one above except that we use the blanket `Exception` class to catch the exception.

```
1  import java.util.Scanner;
2
3  /*-
4   * ThrowingExceptions3
5   * Simple example showing how to use Java's exception structure to cause your own crashes
6   * and catch them. The only difference here is that we are catching all of the exceptions.
7   * Author:  Don Spickler
8   * Date: 2/6/2011
9   */
10
11 public class ThrowingExceptions3 {
12
13     public static void printSquares(int startingValue, int endingValue) {
14         if ((startingValue <= 0) || (endingValue <= 0))
15             throw new IllegalArgumentException("Starting value and ending value must be
                   positive.");
16
17         if (endingValue < startingValue)
18             throw new IllegalArgumentException("Starting value must be less than or equal
                   to ending value.");
19
20         for (int i = startingValue; i <= endingValue; i++) {
21             System.out.print(i * i + "  ");
22         }
23         System.out.println();
24     }
25
26     public static void main(String[] args) {
27         Scanner keyboard = new Scanner(System.in);
28         boolean expThrown = true;
29
30         while (expThrown) {
31             expThrown = false;
32             System.out.print("Input start: ");
33             int start = keyboard.nextInt();
34             System.out.print("Input end: ");
35             int end = keyboard.nextInt();
36
37             try {
38                 printSquares(start, end);
```

```
39              } catch (Exception e) {
40                  expThrown = true;
41                  System.out.println(e.getMessage());
42              }
43          }
44          keyboard.close();
45      }
46  }
```

Program Output **ThrowingExceptions3.java**

```
Input start: -4
Input end: 7
Starting value and ending value must be positive.
Input start: 7
Input end: 4
Starting value must be less than or equal to ending value.
Input start: 2
Input end: 10
4  9  16  25  36  49  64  81  100
```

8.3.4 Throwing and Catching All Exceptions Example #2

This example shows how the programmer can string along several catch types in one try-catch block. It works in a similar manner to the the `else if` statements. If an exception is thrown and the exception type is not the first one it will skip down to the next catch and see if that exception type matches, if not it moves to the next catch and so on. If an exception is thrown and none of the catch types match the program will crash.

```
1  import java.util.InputMismatchException;
2  import java.util.Scanner;
3
4  /*-
5   * ThrowingExceptions4
6   * Simple example showing how to use Java's exception structure to cause your own crashes
7   * and catch them. Here we use the method of catching multiple specific exceptions.
8   * Author:  Don Spickler
9   * Date: 2/6/2011
10  */
11
12 public class ThrowingExceptions4 {
13
14      public static void printSquares(int startingValue, int endingValue) {
15          if ((startingValue <= 0) || (endingValue <= 0))
16              throw new IllegalArgumentException("Starting value and ending value must be
                    positive.");
17
18          if (endingValue < startingValue)
19              throw new IllegalArgumentException("Starting value must be less than or equal
                    to ending value.");
20
21          for (int i = startingValue; i <= endingValue; i++) {
22              System.out.print(i * i + "  ");
23          }
24          System.out.println();
25      }
```

```
26
27      public static void main(String[] args) {
28          Scanner keyboard = new Scanner(System.in);
29          boolean expThrown = true;
30
31          while (expThrown) {
32              expThrown = false;
33              int start;
34              int end;
35
36              try {
37                  System.out.print("Input start: ");
38                  start = keyboard.nextInt();
39                  System.out.print("Input end: ");
40                  end = keyboard.nextInt();
41
42                  printSquares(start, end);
43              } catch (IllegalArgumentException e) {
44                  expThrown = true;
45                  System.out.println(e.getMessage());
46              } catch (InputMismatchException e) {
47                  expThrown = true;
48                  System.out.println(e.getMessage());
49                  String str = keyboard.nextLine();
50              }
51          }
52          keyboard.close();
53      }
54  }
```

Program Output — ThrowingExceptions4.java

```
Input start: -4
Input end: 5
Starting value and ending value must be positive.
Input start: 7
Input end: 3
Starting value must be less than or equal to ending value.
Input start: 3.2
null
Input start: 5
Input end: 7.7
null
Input start: 3
Input end: 9
9  16  25  36  49  64  81
```

Chapter 9

User Input Testing

9.1 Introduction

In this chapter we use what we learned about exception handling to create more sophisticated user input checking. When a program can detect incorrect inputs and either ask for the inputs again or alter them in a reasonable manner so that the program does not crash, the program is said to be robust. You want your programs to be as robust as possible, it is frustrating for the user to have to restart a program if they simply mistype an input or did not understand the instructions for input into the program.

9.2 No Input Testing Example

This example uses no user input testing. If the user types in an integer, as they are instructed, the program runs fine but if they type in a string or a decimal number the program will generate an InputMismatchException.

```
1  import java.util.Scanner;
2
3  /*-
4   * UserInputTesting000
5   * Simple example that uses no testing of input.
6   * Author:  Don Spickler
7   * Date: 3/15/2011
8   */
9
10 public class UserInputTesting000 {
11     public static void main(String[] args) {
12         Scanner kb = new Scanner(System.in);
13
14         // Input from keyboard with no data type checking.
15         System.out.print("Input an integer (no data checking): ");
16         int num = kb.nextInt();
17         kb.close();
```

```
18
19          System.out.println("num = " + num);
20      }
21  }
```

Program Output **UserInputTesting000.java Run #1**

```
Input an integer (no data checking): 5
num = 5
```

Program Output **UserInputTesting000.java Run #2**

```
Input an integer (no data checking): 3.6
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at UserInputTesting000.main(UserInputTesting000.java:9)
```

Program Output **UserInputTesting000.java Run #3**

```
Input an integer (no data checking): A
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at UserInputTesting000.main(UserInputTesting000.java:9)
```

9.3 Basic Input Testing Example

In this example we do a little input testing to make sure that the user has typed in the correct data type. The scanner object, the object we link the keyboard to, has a set of `has` methods. The one we use here is `hasNextInt` which determines if the next thing to be read in is an integer. This method returns a boolean value, true if the next thing it an integer and false otherwise. There are other `has` methods for longs, floats, doubles, and other data types.

When the user types in something from the keyboard the input is stored in an input buffer. A buffer is simply another name for a memory location. So the user types in something and hits enter. When the enter key is pressed the `hasNextInt` method looks inside this buffer and determines the data type of the input. If the data type of the next thing in the input is an integer then the method returns true and if not it returns false. If it returns true then we can call `nextInt` without the program throwing an exception. If it returns false, we need to get a different input from the user. One problem here is that the buffer still has the old input in it. The input buffer keeps what is in it until it is read out of the buffer. So we put in the

statement, `String clearbuf = kb.nextLine();` to read the entire buffer and assign it to a dummy variable. At this point the input buffer is empty and ready for the user to try again.

```
1  import java.util.Scanner;
2
3  /*-
4   * UserInputTesting001
5   * Simple example that uses no testing of input.
6   * Author:  Don Spickler
7   * Date: 3/15/2011
8   */
9
10 public class UserInputTesting001 {
11
12     public static void main(String[] args) {
13         Scanner kb = new Scanner(System.in);
14
15         // Input from keyboard with data type checking.
16         // Ask the scanner if the next thing to be read in is an integer.
17         // If so, read it in and if not, print an error message and ask
18         // for another input. Finally clear out the input buffer.
19
20         boolean inputNeeded = true;
21         int value = 0;
22         while (inputNeeded) {
23             System.out.print("Input an integer (data checking): ");
24             if (kb.hasNextInt()) {
25                 value = kb.nextInt();
26                 inputNeeded = false;
27             } else {
28                 System.out.println("Input is not an integer, try again.");
29             }
30             String clearbuf = kb.nextLine();
31         }
32         kb.close();
33
34         System.out.println("value = " + value);
35     }
36 }
```

Program Output **UserInputTesting001.java**

```
Input an integer (data checking): A
Input is not an integer, try again.
Input an integer (data checking): 3.6
Input is not an integer, try again.
Input an integer (data checking): 5
value = 5
```

9.4 Input Testing Example Using a Method

In this example we extract the data testing code and placed it into a method. This allows the programmer to call the input test multiple times and reduce the number of lines of code.

```
1  import java.util.Scanner;
```

```
2
3  /*-
4   * UserInputTesting002
5   * Made the input test into a method, so that it can be called
6   * several times in the main program and reduce the code redundancy.
7   * Author:  Don Spickler
8   * Date: 3/15/2011
9   */
10
11 public class UserInputTesting002 {
12
13     public static int getInteger() {
14         Scanner kb = new Scanner(System.in);
15
16         boolean inputNeeded = true;
17         int value = 0;
18         while (inputNeeded) {
19             System.out.print("Input an integer: ");
20             if (kb.hasNextInt()) {
21                 value = kb.nextInt();
22                 inputNeeded = false;
23             } else {
24                 System.out.println("Input is not an integer, try again.");
25             }
26             String clearbuf = kb.nextLine();
27         }
28         return value;
29     }
30
31     public static void main(String[] args) {
32         int num = getInteger();
33         System.out.println("num = " + num);
34
35         System.out.println();
36
37         int num2 = getInteger();
38         System.out.println("num2 = " + num2);
39     }
40 }
```

Program Output **UserInputTesting002.java**

```
Input an integer: A
Input is not an integer, try again.
Input an integer: 3.256
Input is not an integer, try again.
Input an integer: 15
num = 15

Input an integer: 6
num2 = 6
```

9.5 Input Testing Example Using Method Overloading

In this example we created several more `getInteger` methods. These new methods allow the call to specify messages and some give a range restriction on the input. In

Java, you can have multiple methods with the same name as long as the parameter lists are different. When there is a call to one of these methods, the computer figures out which on to call by the parameters in the call.

```
1  import java.util.Scanner;
2
3  /*-
4   * UserInputTesting003
5   * Overloaded the getInteger methods so that they can be used
6   * in different situations.
7   * Author:  Don Spickler
8   * Date: 3/15/2011
9   */
10
11 public class UserInputTesting003 {
12
13     // Uses default messages.
14     public static int getInteger() {
15         Scanner kb = new Scanner(System.in);
16
17         boolean inputNeeded = true;
18         int value = 0;
19         while (inputNeeded) {
20             System.out.print("Input an integer: ");
21             if (kb.hasNextInt()) {
22                 value = kb.nextInt();
23                 inputNeeded = false;
24             } else {
25                 System.out.println("Input is not an integer, try again.");
26             }
27             String clearbuf = kb.nextLine();
28         }
29         return value;
30     }
31
32     // Uses programmer input message.
33     public static int getInteger(String message) {
34         Scanner kb = new Scanner(System.in);
35
36         boolean inputNeeded = true;
37         int value = 0;
38         while (inputNeeded) {
39             System.out.print(message);
40             if (kb.hasNextInt()) {
41                 value = kb.nextInt();
42                 inputNeeded = false;
43             } else {
44                 System.out.println("Input is not an integer, try again.");
45             }
46             String clearbuf = kb.nextLine();
47         }
48         return value;
49     }
50
51     // Uses programmer input message and error message.
52     public static int getInteger(String message, String errorMessage) {
53         Scanner kb = new Scanner(System.in);
54
55         boolean inputNeeded = true;
56         int value = 0;
57         while (inputNeeded) {
58             System.out.print(message);
59             if (kb.hasNextInt()) {
```

```
60                  value = kb.nextInt();
61                  inputNeeded = false;
62              } else {
63                  System.out.println(errorMessage);
64              }
65              String clearbuf = kb.nextLine();
66          }
67          return value;
68      }
69
70      // Uses programmer input range and default messages.
71      public static int getInteger(int low, int high) {
72          Scanner kb = new Scanner(System.in);
73
74          boolean inputNeeded = true;
75          int value = 0;
76          while (inputNeeded) {
77              System.out.print("Input an integer between " + low + " to " + high + ": ");
78              if (kb.hasNextInt()) {
79                  value = kb.nextInt();
80                  if (value < low || value > high)
81                      System.out.println("Input is not in the correct range, try again.");
82                  else
83                      inputNeeded = false;
84              } else {
85                  System.out.println("Input is not an integer, try again.");
86              }
87              String clearbuf = kb.nextLine();
88          }
89          return value;
90      }
91
92      // Uses programmer input range and message.
93      public static int getInteger(int low, int high, String message) {
94          Scanner kb = new Scanner(System.in);
95
96          boolean inputNeeded = true;
97          int value = 0;
98          while (inputNeeded) {
99              System.out.print(message);
100             if (kb.hasNextInt()) {
101                 value = kb.nextInt();
102                 if (value < low || value > high)
103                     System.out.println("Input is not in the correct range, try again.");
104                 else
105                     inputNeeded = false;
106             } else {
107                 System.out.println("Input is not an integer, try again.");
108             }
109             String clearbuf = kb.nextLine();
110         }
111         return value;
112     }
113
114     // Uses programmer input range and messages for prompt, error, and range
115     // error.
116     public static int getInteger(int low, int high, String message, String errorMessage,
        String rangeErrorMessage) {
117         Scanner kb = new Scanner(System.in);
118
119         boolean inputNeeded = true;
120         int value = 0;
121         while (inputNeeded) {
122             System.out.print(message);
```

```
123             if (kb.hasNextInt()) {
124                 value = kb.nextInt();
125                 if (value < low || value > high)
126                     System.out.println(rangeErrorMessage);
127                 else
128                     inputNeeded = false;
129             } else {
130                 System.out.println(errorMessage);
131             }
132             String clearbuf = kb.nextLine();
133         }
134         return value;
135     }
136 
137     public static void main(String[] args) {
138         int num = getInteger();
139         System.out.println("num1 = " + num);
140         System.out.println();
141 
142         num = getInteger("Input num: ");
143         System.out.println("num2 = " + num);
144         System.out.println();
145 
146         num = getInteger("Input an integer called num: ", "Not an integer, please try
                harder this time.");
147         System.out.println("num3 = " + num);
148         System.out.println();
149 
150         num = getInteger(2, 10);
151         System.out.println("num4 = " + num);
152         System.out.println();
153 
154         num = getInteger(2, 10, "I would like an integer between 2 and 10: ");
155         System.out.println("num5 = " + num);
156         System.out.println();
157 
158         num = getInteger(5, 15, "Please input an integer between 5 and 15: ",
159                 "This was not an integer, please try harder this time.",
160                 "I said between 5 and 15, what word don't you understand?");
161         System.out.println("num6 = " + num);
162         System.out.println();
163     }
164 }
```

Program Output — UserInputTesting003.java

```
Input an integer: A
Input is not an integer, try again.
Input an integer: 2.5
Input is not an integer, try again.
Input an integer: 56
num1 = 56

Input num: 2.41
Input is not an integer, try again.
Input num: 5
num2 = 5

Input an integer called num: 15.93C
Not an integer, please try harder this time.
Input an integer called num: Help
Not an integer, please try harder this time.
Input an integer called num: 23
```

```
num3 = 23

Input an integer between 2 to 10: 6.84
Input is not an integer, try again.
Input an integer between 2 to 10: 15
Input is not in the correct range, try again.
Input an integer between 2 to 10: 8
num4 = 8

I would like an integer between 2 and 10: 1
Input is not in the correct range, try again.
I would like an integer between 2 and 10: 56.23
Input is not an integer, try again.
I would like an integer between 2 and 10: 10
num5 = 10

Please input an integer between 5 and 15: 25
I said between 5 and 15, what word don't you understand?
Please input an integer between 5 and 15: 23.65
This was not an integer, please try harder this time.
Please input an integer between 5 and 15: Blast
This was not an integer, please try harder this time.
Please input an integer between 5 and 15: 12
num6 = 12
```

9.6 Input Testing Example Using Condensed Overloading

One thing you probably noticed in the last example was that there was a lot of repetition of code between the methods. While this is not a terrible thing we can reduce the amount of code by simply taking the most general methods and have the rest call these general ones. This will not reduce the number of `getInteger` methods and hence the programmer has the same input checking functionality.

```
1  import java.util.Scanner;
2
3  /*-
4   * UserInputTesting004
5   * Combined the six getInteger methods into the two most general ones and
6   * the other four simply call these two.  This reduced code redundancy and
7   * makes updating easier.
8   * Author:  Don Spickler
9   * Date: 3/15/2011
10  */
11
12 public class UserInputTesting004 {
13
14     // Made the getInteger method more general by letting the user input
15     // the prompt message and the error message.
16     public static int getInteger(String message, String errorMessage) {
17         Scanner kb = new Scanner(System.in);
18
19         boolean inputNeeded = true;
20         int value = 0;
21         while (inputNeeded) {
22             System.out.print(message);
```

```
23              if (kb.hasNextInt()) {
24                  value = kb.nextInt();
25                  inputNeeded = false;
26              } else {
27                  System.out.println(errorMessage);
28              }
29              String clearbuf = kb.nextLine();
30          }
31          return value;
32      }
33
34      // Added the functionality to check that an input is within a range
35      // of numbers [low, high].
36      public static int getInteger(int low, int high, String message, String errorMessage,
            String rangeErrorMessage) {
37          Scanner kb = new Scanner(System.in);
38
39          boolean inputNeeded = true;
40          int value = 0;
41          while (inputNeeded) {
42              System.out.print(message);
43              if (kb.hasNextInt()) {
44                  value = kb.nextInt();
45                  if (value < low || value > high)
46                      System.out.println(rangeErrorMessage);
47                  else
48                      inputNeeded = false;
49              } else {
50                  System.out.println(errorMessage);
51              }
52              String clearbuf = kb.nextLine();
53          }
54          return value;
55      }
56
57      // The following are overloads of the getInteger method that allow
58      // the user to use some default messages.
59      public static int getInteger() {
60          return getInteger("Input an integer: ", "Input is not an integer, try again.");
61      }
62
63      public static int getInteger(String message) {
64          return getInteger(message, "Input is not an integer, try again.");
65      }
66
67      public static int getInteger(int low, int high) {
68          return getInteger(low, high, "Input an integer between " + low + " to " + high + "
                : ",
69                  "Input is not an integer, try again.", "Input is not in the correct range,
                        try again.");
70      }
71
72      public static int getInteger(int low, int high, String message) {
73          return getInteger(low, high, message, "Input is not an integer, try again.",
74                  "Input is not in the correct range, try again.");
75      }
76
77      public static void main(String[] args) {
78          int num = getInteger();
79          System.out.println("num1 = " + num);
80          System.out.println();
81
82          num = getInteger("Input num: ");
83          System.out.println("num2 = " + num);
```

```
84          System.out.println();
85
86          num = getInteger("Input an integer called num: ", "Not an integer, please try
                harder this time.");
87          System.out.println("num3 = " + num);
88          System.out.println();
89
90          num = getInteger(2, 10);
91          System.out.println("num4 = " + num);
92          System.out.println();
93
94          num = getInteger(2, 10, "I would like an integer between 2 and 10: ");
95          System.out.println("num5 = " + num);
96          System.out.println();
97
98          num = getInteger(5, 15, "Please input an integer between 5 and 15: ",
99                  "This was not an integer, please try harder this time.",
100                 "I said between 5 and 15, what word don't you understand?");
101         System.out.println("num6 = " + num);
102         System.out.println();
103     }
104 }
```

Program Output — UserInputTesting004.java

```
Input an integer: Kind
Input is not an integer, try again.
Input an integer: 23156
num1 = 23156

Input num: 3.14159
Input is not an integer, try again.
Input num: Pi
Input is not an integer, try again.
Input num: 314159
num2 = 314159

Input an integer called num: 32.98
Not an integer, please try harder this time.
Input an integer called num: E
Not an integer, please try harder this time.
Input an integer called num: 2718
num3 = 2718

Input an integer between 2 to 10: 0
Input is not in the correct range, try again.
Input an integer between 2 to 10: 32.94
Input is not an integer, try again.
Input an integer between 2 to 10: 7
num4 = 7

I would like an integer between 2 and 10: -59
Input is not in the correct range, try again.
I would like an integer between 2 and 10: 5.6
Input is not an integer, try again.
I would like an integer between 2 and 10: 9
num5 = 9

Please input an integer between 5 and 15: 0
I said between 5 and 15, what word don't you understand?
Please input an integer between 5 and 15: 2Pi
This was not an integer, please try harder this time.
Please input an integer between 5 and 15: 7
```

```
num6 = 7
```

9.7 Input Testing Example Using Exception Handling

In the previous examples we used the `has` methods in the scanner to do the checking before we read in the data. An alternative method is to go ahead and read in the data and if it is the wrong type the program will throw an exception, we simply catch and recover from the exception.

```
1  import java.util.Scanner;
2
3  /*-
4   * UserInputTesting005
5   * Changed the hasNextInt structure to a try-catch structure.  So we
6   * let the program try to read the input and if it is invalid we let
7   * Java through an exception, in this case we catch the exception and
8   * display the appropriate message.
9   * Author:  Don Spickler
10  * Date: 3/15/2011
11  */
12
13 public class UserInputTesting005 {
14
15     // Made the getInteger method more general by letting the user input
16     // the prompt message and the error message.
17     public static int getInteger(String message, String errorMessage) {
18         Scanner kb = new Scanner(System.in);
19
20         boolean inputNeeded = true;
21         int value = 0;
22         while (inputNeeded) {
23             System.out.print(message);
24             try {
25                 value = kb.nextInt();
26                 inputNeeded = false;
27             } catch (Exception e) {
28                 System.out.println(errorMessage);
29             }
30             String clearbuf = kb.nextLine();
31         }
32         return value;
33     }
34
35     // Added the functionality to check that an input is within a range
36     // of numbers [low, high].
37     public static int getInteger(int low, int high, String message, String errorMessage,
           String rangeErrorMessage) {
38         Scanner kb = new Scanner(System.in);
39
40         boolean inputNeeded = true;
41         int value = 0;
42         while (inputNeeded) {
43             System.out.print(message);
44             try {
45                 value = kb.nextInt();
46                 if (value < low || value > high)
```

```
47                      System.out.println(rangeErrorMessage);
48                  else
49                      inputNeeded = false;
50              } catch (Exception e) {
51                  System.out.println(errorMessage);
52              }
53              String clearbuf = kb.nextLine();
54          }
55          return value;
56      }
57
58      // The following are overloads of the getInteger method that allow
59      // the user to use some default messages.
60      public static int getInteger() {
61          return getInteger("Input an integer: ", "Input is not an integer, try again.");
62      }
63
64      public static int getInteger(String message) {
65          return getInteger(message, "Input is not an integer, try again.");
66      }
67
68      public static int getInteger(int low, int high) {
69          return getInteger(low, high, "Input an integer between " + low + " to " + high + "
                : ",
70                  "Input is not an integer, try again.", "Input is not in the correct range,
                        try again.");
71      }
72
73      public static int getInteger(int low, int high, String message) {
74          return getInteger(low, high, message, "Input is not an integer, try again.",
75                  "Input is not in the correct range, try again.");
76      }
77
78      public static void main(String[] args) {
79          int num = getInteger();
80          System.out.println("num1 = " + num);
81          System.out.println();
82
83          num = getInteger("Input num: ");
84          System.out.println("num2 = " + num);
85          System.out.println();
86
87          num = getInteger("Input an integer called num: ", "Not an integer, please try
                harder this time.");
88          System.out.println("num3 = " + num);
89          System.out.println();
90
91          num = getInteger(2, 10);
92          System.out.println("num4 = " + num);
93          System.out.println();
94
95          num = getInteger(2, 10, "I would like an integer between 2 and 10: ");
96          System.out.println("num5 = " + num);
97          System.out.println();
98
99          num = getInteger(5, 15, "Please input an integer between 5 and 15: ",
100                 "This was not an integer, please try harder this time.",
101                 "I said between 5 and 15, what word don't you understand?");
102         System.out.println("num6 = " + num);
103         System.out.println();
104     }
105 }
```

Program Output **UserInputTesting005.java**

```
Input an integer: Kind
Input is not an integer, try again.
Input an integer: 23156
num1 = 23156

Input num: 3.14159
Input is not an integer, try again.
Input num: Pi
Input is not an integer, try again.
Input num: 314159
num2 = 314159

Input an integer called num: 32.98
Not an integer, please try harder this time.
Input an integer called num: E
Not an integer, please try harder this time.
Input an integer called num: 2718
num3 = 2718

Input an integer between 2 to 10: 0
Input is not in the correct range, try again.
Input an integer between 2 to 10: 32.94
Input is not an integer, try again.
Input an integer between 2 to 10: 7
num4 = 7

I would like an integer between 2 and 10: -59
Input is not in the correct range, try again.
I would like an integer between 2 and 10: 5.6
Input is not an integer, try again.
I would like an integer between 2 and 10: 9
num5 = 9

Please input an integer between 5 and 15: 0
I said between 5 and 15, what word don't you understand?
Please input an integer between 5 and 15: 2Pi
This was not an integer, please try harder this time.
Please input an integer between 5 and 15: 7
num6 = 7
```

Chapter 10

Arrays

10.1 Introduction

So far we have been storing a small amount of data, a few integers or a sting or two. What if we needed to store 100 or 1000 integers? It would be extremely cumbersome to define 100 or 1000 different variables to store all of data. In Java, as well as most other languages, there are structures for storing large amounts of data using a single variable name. We will look at a couple in this set of notes, Java has far more to choose from.

One such structure is the array. An array allows the programmer to specify a single name for all of the values and they can access the values by referencing where the value is in the list. Arrays are usually grouped by their dimension. A One-Dimensional array can be thought of as a single line of values. A Two-Dimensional array is like a spreadsheet of cells each with a value, we will look at examples of these later on in the notes. Java arrays can have higher dimensions. A three-dimensional array is like a stack of spreadsheets, and so on. Most of the time you will be using either one or two dimensional arrays.

10.2 One-Dimensional Arrays

The arrays in this section are called One-Dimensional arrays since they are a single line of values. We can think of a one-dimensional array as a list of cells, each of which contain a single value of the data type of the array. For example, a one-dimensional array with 5 cells containing the integers, 5, -2, 7, 23, and 15, in that order could be represented as,

5	-2	7	23	15

10.2.1 Basic One-Dimensional Array Example

This example shows some of the basic syntax and manipulation of a one-dimensional array. Declaring the array can be done in several ways and examples are on lines 2 and 3. Start with the type of data you intend to store in the array, then the array name followed by empty square brackets or the other way around, empty square brackets followed by the array name. After that, do an assignment to a new same data type and in square brackets the number of data items you wish to store. So on line 2 we declared an array named `intArray` that is storing 5 integers. On line 3 we declared an array named `intArray2` that is storing 10 integers.

```
1  public static void main(String[] args) {
2      int intArray[] = new int[5];
3      int[] intArray2 = new int[10];
4
5      for (int i = 0; i < 5; i++) {
6          intArray[i] = i * i;
7      }
8
9      for (int i = 0; i < 5; i++) {
10         System.out.print(intArray[i] + "   ");
11     }
12
13     System.out.println();
14
15     for (int i = 0; i < 10; i++) {
16         intArray2[i] = i * i * i;
17     }
18
19     for (int i = 0; i < 10; i++) {
20         System.out.print(intArray2[i] + "   ");
21     }
22 }
```

Line 6 shows how we reference one of these positions. We start with the array name and in square brackets we put the index of the integer we want. So `intArray[i]` is the integer in the i^{th} location in the array. The indexing starts at 0 and increases by one until we hit the end of the array. For example, suppose that the `intArray` looked like the following.

5	−2	7	23	15

Then the location references and values are, `intArray[0]` has the value 5, `intArray[1]` has the value −2, `intArray[2]` has the value 7, `intArray[3]` has

the value 23, and `intArray[4]` has the value 15. So the first index is always 0 and the last index is $n-1$ where n is the size of the array. The variables `intArray[0]`, `intArray[1]`, ...can be used like any other variable of that type. They can be assigned values, used in formulas, printed out, and so on. In our example above, the first loop assigns each location in the array using a standard assignment statement.

```
for (int i = 0; i < 5; i++) {
    intArray[i] = i * i;
}
```

The result is,

0	1	4	9	16

The next for loop prints the entire array to the console screen.

```
for (int i = 0; i < 5; i++) {
    System.out.print(intArray[i] + "  ");
}
```

```
1  /*-
2   * ArrayExample001
3   * Example showing the basic declaration and storage access for an
4   * array, specifically a one-dimensional array.
5   * Author:  Don Spickler
6   * Date: 3/20/2011
7   */
8
9  public class ArrayExample001 {
10     public static void main(String[] args) {
11         int intArray[] = new int[5];
12         int[] intArray2 = new int[10];
13
14         for (int i = 0; i < 5; i++) {
15             intArray[i] = i * i;
16         }
17
18         for (int i = 0; i < 5; i++) {
19             System.out.print(intArray[i] + "  ");
20         }
21
22         System.out.println();
23
24         for (int i = 0; i < 10; i++) {
25             intArray2[i] = i * i * i;
26         }
27
28         for (int i = 0; i < 10; i++) {
29             System.out.print(intArray2[i] + "  ");
30         }
31     }
32 }
```

Program Output **ArrayExample001.java**

```
0  1  4  9  16
0  1  8  27  64  125  216  343  512  729
```

10.2.2 One-Dimensional Variable Size Array Example

In the previous example, we defined our array sizes of 5 and 10. One nice feature for Java arrays is that we do not have to assign the size of the array when we write the program. We can use a variable to designate the size and this can change between runs of the program. In this example, we declare the array with the statement,

```
int intArray[] = new int[arraySize];
```

where `arraySize` came in from the user. So in one run of the program we might have an array size of 15, in another we might have an array size of 1000. One should be a little careful with this, array sizes must be greater than or equal to 0, if the user types in a negative number here the program will generate an exception and crash.

```
1  import java.util.Scanner;
2
3  /*-
4   * ArrayExample002
5   * Example showing the basic declaration and storage access for an
6   * array, specifically a one-dimensional array.
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
11 public class ArrayExample002 {
12     public static void main(String[] args) {
13         Scanner keyboard = new Scanner(System.in);
14         System.out.print("Input the array size: ");
15         int arraySize = keyboard.nextInt();
16         keyboard.close();
17
18         int intArray[] = new int[arraySize];
19
20         for (int i = 0; i < intArray.length; i++) {
21             intArray[i] = i * i;
22         }
23
24         for (int i = 0; i < intArray.length; i++) {
25             System.out.print(intArray[i] + "  ");
26         }
27     }
28 }
```

Program Output **ArrayExample002.java**

```
Input the array size: 20
0  1  4  9  16  25  36  49  64  81  100  121  144  169  196  225  256  289  324  361
```

10.2.3 One-Dimensional Array Parameter Example

Arrays can be sent into methods as parameters just like single integers, floats or strings. Let's look the `PrintArray` method from this example.

```
public static void PrintArray(int[] Arr) {
    for (int i = 0; i < Arr.length; i++) {
        System.out.println(Arr[i] + "  ");
    }
}
```

The syntax of the parameter list has the type of the array followed by empty square brackets and then the name of the array for the method. Notice that the parameter list does not specify the size of the array. In Java the size of the array is stored as part of the array. Note in the for loop the use of `Arr.length` in the loop condition. This returns the length of the array. Not all languages will store the array length with the array. In particular, C and C++ will not. You can send an array to a method (or function) in C and C++ in a similar manner as we did here but since the array size is not part of the array, you will need to send another parameter, an integer, which holds the array size.

The method call is similar to a method call for a single integer value, simply send the name of the array, no brackets needed.

```
PrintArray(intArray);
```

```
1  import java.util.Scanner;
2
3  /*-
4   * ArrayExample003
5   * Example showing how to pass a one-dimensional array to a method as a parameter.
6   * Author:  Don Spickler
7   * Date: 3/20/2011
8   */
9
10 public class ArrayExample003 {
11
12     public static void PopulateArray(int[] A) {
13         Scanner keyboard = new Scanner(System.in);
14         for (int i = 0; i < A.length; i++) {
15             System.out.print("Input entry " + (i + 1) + ": ");
16             A[i] = keyboard.nextInt();
17         }
18     }
19
20     public static void PrintArray(int[] Arr) {
21         for (int i = 0; i < Arr.length; i++) {
22             System.out.println(Arr[i] + "  ");
23         }
24     }
25
26     public static int SumArray(int[] Arr) {
27         int sum = 0;
28         for (int i = 0; i < Arr.length; i++) {
```

```
29              sum += Arr[i];
30          }
31          return sum;
32      }
33
34      public static void main(String[] args) {
35          Scanner keyboard = new Scanner(System.in);
36          System.out.print("Input the array size: ");
37          int arraySize = keyboard.nextInt();
38
39          int intArray[] = new int[arraySize];
40          PopulateArray(intArray);
41          PrintArray(intArray);
42          System.out.println("The sum of the array is = " + SumArray(intArray));
43      }
44  }
```

Program Output **ArrayExample003.java**

```
Input the array size: 5
Input entry 1: 2
Input entry 2: 3
Input entry 3: 8
Input entry 4: 5
Input entry 5: 4
2
3
8
5
4
The sum of the array is = 22
```

10.2.4 One-Dimensional Array Parameter Example #2

Arrays can be constructed out of any data type, it does not have to be simple integers or doubles. You can even make an array out of a data type (object) you created. In this example we use the `Employee` class from several chapters ago to create an array to store the employee data of our company. Note that the syntax is the same, we simply replace `int` with `Employee`.

```
Employee company[] = new Employee[arraySize];
```

```
1   /*-
2    * Employee
3    * Employee class stores information about a company employee, name,
4    * wage, and hours worked for the week.
5    * Author:  Don Spickler
6    * Date: 3/20/2011
7    */
8
9   public class Employee {
10      private String name;
11      private double wage;
12      private double hours_worked;
13
14      public Employee(String n, double w, double hw) {
```

```
15          name = n;
16          if (w > 0)
17              wage = w;
18          else
19              wage = 0;
20
21          if (hw > 0)
22              hours_worked = hw;
23          else
24              hours_worked = 0;
25      }
26
27      public String getName() {
28          return name;
29      }
30
31      public double getWage() {
32          return wage;
33      }
34
35      public double getHoursWorked() {
36          return hours_worked;
37      }
38
39      public void setName(String n) {
40          name = n;
41      }
42
43      public void setWage(double w) {
44          if (w > 0)
45              wage = w;
46          else
47              wage = 0;
48      }
49
50      public void getHoursWorked(double hw) {
51          if (hw > 0)
52              hours_worked = hw;
53          else
54              hours_worked = 0;
55      }
56
57      public double pay() {
58          double payment = 0;
59          if (hours_worked > 40)
60              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
61          else
62              payment = wage * hours_worked;
63
64          return payment;
65      }
66  }
```

```
1   import java.util.Scanner;
2
3   /*-
4    * ArrayExample004
5    * Another example showing how to pass a one-dimensional array to a
6    * method as a parameter.
7    * Author:  Don Spickler
8    * Date: 3/20/2011
9    */
10
11  public class ArrayExample004 {
```

```
12
13      public static void InputEmployees(Employee[] A) {
14          Scanner keyboard = new Scanner(System.in);
15
16          for (int i = 0; i < A.length; i++) {
17              System.out.print("Employee " + (i + 1) + " name: ");
18              String name = keyboard.nextLine();
19              System.out.print("Employee " + (i + 1) + " wage: ");
20              double wage = keyboard.nextDouble();
21              System.out.print("Employee " + (i + 1) + " hours worked: ");
22              double hours = keyboard.nextDouble();
23
24              Employee emp = new Employee(name, wage, hours);
25              A[i] = emp;
26
27              System.out.println(); // Put space between inputs.
28              String clear = keyboard.nextLine(); // clear the end of line.
29          }
30      }
31
32      public static void PrintPayReport(Employee[] A) {
33          for (int i = 0; i < A.length; i++) {
34              PrintRecord(A[i]);
35          }
36      }
37
38      public static double CalculateTotalPay(Employee[] A) {
39          double totalpay = 0;
40          for (int i = 0; i < A.length; i++) {
41              totalpay += A[i].pay();
42          }
43          return totalpay;
44      }
45
46      public static void PrintRecord(Employee employee) {
47          System.out.println("Name: " + employee.getName());
48          System.out.println("Wage: " + employee.getWage());
49          System.out.println("Hours Worked: " + employee.getHoursWorked());
50          System.out.printf("Pay: %.2f \n", employee.pay());
51          System.out.println();
52      }
53
54      public static void main(String[] args) {
55          Scanner keyboard = new Scanner(System.in);
56          System.out.print("Input the number of employees: ");
57          int arraySize = keyboard.nextInt();
58
59          Employee company[] = new Employee[arraySize];
60          InputEmployees(company);
61          PrintPayReport(company);
62          double totpay = CalculateTotalPay(company);
63          System.out.printf("Company payout = %.2f \n", totpay);
64      }
65  }
```

Program Output — ArrayExample004.java

```
Input the number of employees: 3
Employee 1 name: Don Spickler
Employee 1 wage: 12.50
Employee 1 hours worked: 45

Employee 2 name: Jane Doe
```

```
Employee 2 wage: 15.25
Employee 2 hours worked: 32

Employee 3 name: Jack Frost
Employee 3 wage: 17.33
Employee 3 hours worked: 29

Name: Don Spickler
Wage: 12.5
Hours Worked: 45.0
Pay: 593.75

Name: Jane Doe
Wage: 15.25
Hours Worked: 32.0
Pay: 488.00

Name: Jack Frost
Wage: 17.33
Hours Worked: 29.0
Pay: 502.57

Company payout = 1584.32
```

10.3 Two-Dimensional Arrays

A Two-Dimensional array is like a spreadsheet of cells, of rows and columns. Each cell contains a location for a value.

10.3.1 Basic Two-Dimensional Array Example

This example shows some basic syntax for two-dimensional arrays. To declare a two-dimensional array is the same as a one-dimensional array except that there are two sets of brackets, one for the rows and one for the columns. The statement,

```
int TwoDimArray[][] = new int[5][7];
```

declares the array `TwoDimArray` as a two-dimensional array of integers with 5 rows and 7 columns. So the rows are the first brackets and the columns are the second brackets. We reference positions in the array the same way, with indices that start at 0. When the array is first declared it is populated with all 0's.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

The statement,

```
TwoDimArray[3][4] = 2;
```

places a 2 in the $(3, 4)$ position, since the indexing starts at 0, this is in row 4 and column 5.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	2	0	0
0	0	0	0	0	0	0

The next two lines,

```
TwoDimArray[1][6] = 12;
TwoDimArray[2][1] = -4;
```

places a 12 in the $(1, 6)$ position, and a -4 in the $(2, 1)$ position.

0	0	0	0	0	0	0
0	0	0	0	0	0	12
0	-4	0	0	0	0	0
0	0	0	0	2	0	0
0	0	0	0	0	0	0

As with one-dimensional arrays, we can extract the number of rows and the number of columns from the array itself. The length will return the number of rows and the length of the 0^{th} position returns the number of columns.

```
int dim1 = TwoDimArray.length; // Number of rows
int dim2 = TwoDimArray[0].length; // Number of columns
```

As with one-dimensional arrays these come in handy when we pass an array as a parameter into a method.

```
1  /*-
2   * ArrayExample005
3   * Example showing the basic manipulation of a two-dimensional array.
4   * Author:  Don Spickler
5   * Date: 3/20/2011
6   */
7
8  public class ArrayExample005 {
9      public static void main(String[] args) {
10         int TwoDimArray[][] = new int[5][7];
11
12         TwoDimArray[3][4] = 2;
13         TwoDimArray[1][6] = 12;
```

```
14          TwoDimArray[2][1] = -4;
15
16          for (int i = 0; i < 5; i++) {
17              for (int j = 0; j < 7; j++) {
18                  System.out.printf("%4d", TwoDimArray[i][j]);
19              }
20              System.out.println();
21          }
22
23          System.out.println();
24          System.out.println();
25
26          int dim1 = TwoDimArray.length; // Gets the number of rows
27          int dim2 = TwoDimArray[0].length; // Gets the number of columns
28          System.out.println(dim1 + "    " + dim2);
29          System.out.println();
30          System.out.println();
31
32          for (int i = 0; i < dim1; i++) {
33              for (int j = 0; j < dim2; j++) {
34                  System.out.printf("%4d", TwoDimArray[i][j]);
35              }
36              System.out.println();
37          }
38
39          for (int i = 0; i < dim1; i++) {
40              for (int j = 0; j < dim2; j++) {
41                  TwoDimArray[i][j] = 2 * i + j;
42              }
43          }
44
45          System.out.println();
46          System.out.println();
47
48          for (int i = 0; i < dim1; i++) {
49              for (int j = 0; j < dim2; j++) {
50                  System.out.printf("%4d", TwoDimArray[i][j]);
51              }
52              System.out.println();
53          }
54      }
55  }
```

Program Output — ArrayExample005.java

```
   0   0   0   0   0   0   0
   0   0   0   0   0   0  12
   0  -4   0   0   0   0   0
   0   0   0   0   2   0   0
   0   0   0   0   0   0   0


5   7


   0   0   0   0   0   0   0
   0   0   0   0   0   0  12
   0  -4   0   0   0   0   0
   0   0   0   0   2   0   0
   0   0   0   0   0   0   0


   0   1   2   3   4   5   6
```

```
   2   3   4   5   6   7   8
   4   5   6   7   8   9  10
   6   7   8   9  10  11  12
   8   9  10  11  12  13  14
```

10.3.2 Two-Dimensional Array Parameter Example

Sending a two-dimensional array to a method is similar to sending a one-dimensional array to a method, the only difference is that we have two sets of brackets and we need to extract the number of rows and columns from the array. If we look at the `Print2DintArray` method from this example, we see that the array parameter just has two sets of empty brackets in the parameter list.

```
public static void Print2DintArray(int A[][]) {
     int dim1 = A.length; // Gets the number of rows
     int dim2 = A[0].length; // Gets the number of columns

     for (int i = 0; i < dim1; i++) {
          for (int j = 0; j < dim2; j++) {
               System.out.printf("%4d", A[i][j]);
          }
          System.out.println();
     }
}
```

The call is identical to the one-dimensional array, just use the array name and no brackets.

```
Print2DintArray(arr1);
```

```
1  /*-
2   * ArrayExample006
3   * Example showing how to pass a two-dimensional array to a method as a parameter.
4   * Author:  Don Spickler
5   * Date: 3/20/2011
6   */
7
8  public class ArrayExample006 {
9
10     public static void Print2DintArray(int A[][]) {
11         int dim1 = A.length; // Gets the number of rows
12         int dim2 = A[0].length; // Gets the number of columns
13
14         for (int i = 0; i < dim1; i++) {
15             for (int j = 0; j < dim2; j++) {
16                 System.out.printf("%4d", A[i][j]);
17             }
18             System.out.println();
19         }
20     }
```

```
21
22      public static void doubleArray(int[][] A) {
23          int dim1 = A.length; // Gets the number of rows
24          int dim2 = A[0].length; // Gets the number of columns
25
26          for (int i = 0; i < dim1; i++) {
27              for (int j = 0; j < dim2; j++) {
28                  A[i][j] = A[i][j] * 2;
29              }
30          }
31      }
32
33      public static void main(String[] args) {
34          int arr1[][] = new int[5][7];
35          int[][] arr2 = new int[12][5];
36
37          for (int i = 0; i < 5; i++) {
38              for (int j = 0; j < 7; j++) {
39                  arr1[i][j] = 2 * i + j;
40              }
41          }
42
43          for (int i = 0; i < 12; i++) {
44              for (int j = 0; j < 5; j++) {
45                  arr2[i][j] = i - j;
46              }
47          }
48
49          Print2DintArray(arr1);
50          System.out.println();
51          Print2DintArray(arr2);
52
53          doubleArray(arr1);
54          System.out.println();
55          Print2DintArray(arr1);
56      }
57  }
```

Program Output **ArrayExample006.java**

```
  0   1   2   3   4   5   6
  2   3   4   5   6   7   8
  4   5   6   7   8   9  10
  6   7   8   9  10  11  12
  8   9  10  11  12  13  14

  0  -1  -2  -3  -4
  1   0  -1  -2  -3
  2   1   0  -1  -2
  3   2   1   0  -1
  4   3   2   1   0
  5   4   3   2   1
  6   5   4   3   2
  7   6   5   4   3
  8   7   6   5   4
  9   8   7   6   5
 10   9   8   7   6
 11  10   9   8   7

  0   2   4   6   8  10  12
  4   6   8  10  12  14  16
  8  10  12  14  16  18  20
 12  14  16  18  20  22  24
```

```
  16  18  20  22  24  26  28
```

10.3.3 Two-Dimensional Array Parameter Example #2

As with the one-dimensional array, we can use a variable for the number of rows or the number of columns. This facility makes our code a little more versatile as we can specify the size of the array at run-time.

```
1  import java.util.Scanner;
2
3  /*-
4   * ArrayExample007
5   * Another example showing how to pass a two-dimensional array to a
6   * method as a parameter.
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
11 public class ArrayExample007 {
12
13     public static void Print2DintArray(int A[][]) {
14         int dim1 = A.length; // Gets the number of rows
15         int dim2 = A[0].length; // Gets the number of columns
16
17         for (int i = 0; i < dim1; i++) {
18             for (int j = 0; j < dim2; j++) {
19                 System.out.printf("%4d", A[i][j]);
20             }
21             System.out.println();
22         }
23     }
24
25     public static void main(String[] args) {
26         Scanner keyboard = new Scanner(System.in);
27         System.out.print("Input the number of rows: ");
28         int rows = keyboard.nextInt();
29         System.out.print("Input the number of columns: ");
30         int cols = keyboard.nextInt();
31
32         int arr1[][] = new int[rows][cols];
33
34         for (int i = 0; i < rows; i++) {
35             for (int j = 0; j < cols; j++) {
36                 arr1[i][j] = i + j;
37             }
38         }
39
40         Print2DintArray(arr1);
41     }
42 }
```

Program Output **ArrayExample007.java**

```
Input the number of rows: 5
Input the number of columns: 7
   0   1   2   3   4   5   6
   1   2   3   4   5   6   7
   2   3   4   5   6   7   8
```

```
   3   4   5   6   7   8   9
   4   5   6   7   8   9  10
```

10.3.4 Two-Dimensional Array Parameter Example #3

This is just another example of a two-dimensional array. Note that we created the array with one extra row and one extra column. These were for row and column sums. Trace through the two loops that calculate the row sums and column sums.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * ArrayExample008
6   * Another example showing how to pass a two-dimensional array to a
7   * method as a parameter.  Also shows how to make the array larger
8   * and use the extra row and column as sum (accumulator) locations.
9   * Author:  Don Spickler
10  * Date: 3/20/2011
11  */
12
13 public class ArrayExample008 {
14
15     public static void PrintSales(int A[][]) {
16         int dim1 = A.length; // Gets the number of rows
17         int dim2 = A[0].length; // Gets the number of columns
18
19         System.out.printf("%8s", "");
20         for (int j = 0; j < dim2 - 1; j++) {
21             System.out.printf("%4s", "P" + (j + 1));
22         }
23         System.out.printf("%4s", "Tot");
24         System.out.println();
25
26         for (int i = 0; i < dim1; i++) {
27             if (i < dim1 - 1)
28                 System.out.printf("%8s", "Week " + (i + 1));
29             else
30                 System.out.printf("%8s", "Total");
31             for (int j = 0; j < dim2; j++) {
32                 System.out.printf("%4d", A[i][j]);
33             }
34             System.out.println();
35         }
36     }
37
38     public static void main(String[] args) {
39         Random generator = new Random();
40         Scanner keyboard = new Scanner(System.in);
41         System.out.print("Input the number of weeks: ");
42         int weeks = keyboard.nextInt();
43         System.out.print("Input the number of sales people: ");
44         int sales = keyboard.nextInt();
45
46         int salesChart[][] = new int[weeks + 1][sales + 1];
47
48         for (int i = 0; i < weeks; i++) {
49             for (int j = 0; j < sales; j++) {
50                 salesChart[i][j] = generator.nextInt(21);
```

```
51              }
52          }
53
54          for (int i = 0; i < weeks; i++) {
55              int weekTotal = 0;
56              for (int j = 0; j < sales; j++) {
57                  weekTotal += salesChart[i][j];
58              }
59              salesChart[i][sales] = weekTotal;
60          }
61
62          for (int i = 0; i < sales; i++) {
63              int personTotal = 0;
64              for (int j = 0; j < weeks; j++) {
65                  personTotal += salesChart[j][i];
66              }
67              salesChart[weeks][i] = personTotal;
68          }
69
70          int total = 0;
71          for (int j = 0; j < weeks; j++) {
72              total += salesChart[j][sales];
73          }
74          salesChart[weeks][sales] = total;
75
76          PrintSales(salesChart);
77      }
78  }
```

Program Output **ArrayExample008.java**

```
Input the number of weeks: 10
Input the number of sales people: 5
           P1  P2  P3  P4  P5 Tot
  Week 1   18  19  20  10  19  86
  Week 2    9   3   8   8   5  33
  Week 3    7   2   7   1  13  30
  Week 4    7  18  19  15  12  71
  Week 5   18   1  14  10  11  54
  Week 6   13  10   0  12  12  47
  Week 7   15   8  20   5  18  66
  Week 8   19  10  10   0   4  43
  Week 9    3   1   6  15   7  32
 Week 10    8   5   8  14  17  52
   Total 117  77 112  90 118 514
```

10.4 More Array Examples

This section simply shows more examples of both one and two-dimensional arrays, including arrays of user-defined objects.

10.4.1 Tic-Tac-Toe

The biggest use of a two-dimensional array is to store and manipulate tabular data, like spreadsheets or matrices. In gaming one of the biggest uses is in board games, such as checkers, chess, or tic-tac-toe. Even if you are playing a nice graphical user interface version of chess with realistic looking chess pieces, under the hood is a simple two-dimensional array that is storing the locations of each of the pieces that are in play. There is really not much difference between storing the contents of a chess board and those of a tic-tac-toe board, although the game of chess is far more complex.

In this example we will construct a program that allows the user to play a game of tic-tac-toe against the computer. We will let the user move first and then have the computer decide the best move it can make with the current state of the game. Although the game of tic-tac-toe is fairly easy to play, telling the computer what to do at each possibility can be rather lengthy. We will develop an algorithm for the process before we begin to do any coding.

We should first think about how we will store the current state of the game board. There are really quite a few ways we can do this but it would seem natural to use a two-dimensional array of characters with three rows and three columns. The characters will be X, O, and a space to signify an empty spot.

If we think about two human players of the game, game play goes something like this.

1. Player #1 moves.
2. Check to see if the game is over.
3. Player #2 moves.
4. Check to see if the game is over.
5. Go back to player #1.

Since player #2 is really the computer we can revise this to the following.

1. Player moves.
2. Check to see if the game is over.
3. Computer moves.
4. Check to see if the game is over.
5. Go back to the player.

The line where we check to see if the game is over is lengthy but straightforward. We check each row to see if the row contains all X's or all O's. If any row has all X's or all O's then announce the winner. If no row contains all X's or all O's check each column for all X's or all O's. If any column has all X's or all O's then announce the winner. If not check the diagonal and the counter diagonal. If no winner is found, check to see if the game board is full, if so declare a tie. If not, continue the game. We will make this more specific,

1. For each row,
   (a) Count the number of X's
   (b) If the number of X's is 3, declare X the winner.
2. For each column,
   (a) Count the number of X's
   (b) If the number of X's is 3, declare X the winner.
3. For the diagonal,
   (a) Count the number of X's
   (b) If the number of X's is 3, declare X the winner.
4. For the counter-diagonal,
   (a) Count the number of X's
   (b) If the number of X's is 3, declare X the winner.
5. For each row,
   (a) Count the number of O's
   (b) If the number of O's is 3, declare O the winner.
6. For each column,
   (a) Count the number of O's
   (b) If the number of O's is 3, declare O the winner.
7. For the diagonal,
   (a) Count the number of O's
   (b) If the number of O's is 3, declare O the winner.
8. For the counter-diagonal,

   (a) Count the number of O's
   (b) If the number of O's is 3, declare O the winner.
9. Count the number of open spaces in the game board.
10. If the number of open spaces is 0, declare no winner, otherwise continue the game.

Before we implement this we can see some places where methods will come in handy. Notice the redundancy in the process, for each row, for each column, and a duplication between searching for O's and X's. When we implement this algorithm we will probably want to create methods that will take in the game board, a character, and a row or column number and return the number of occurrences of that character on that row or column.

The other main portion to this program is for the computer to decide on a move. Since we check for a win before the computer makes a move it can be assumed that the game is not already won and there are still moves to be made. Again, let's think about how a human would play the game. If it is your turn the first thing you would do would see if there is a way to win, if so you would take it. If not then you would want to block your opponent from winning if they would win on the their next move. After that you would want to progress to a win. The first two steps on this are fairly straightforward. You would search each row, column, and diagonal for two occurrences of your letter and an empty space. If found, you put your character in the empty space, and win. If you have to proceed to a block you do the same except that you search for two occurrences of your opponents letter and an empty space. If found, you put your character in the empty space, and block. Thinking ahead to the implementation, we already have methods for determining the number of a specific character in a row, column, or diagonal. So we can search for our character, or opponent's character, and then a space. If the first search comes up with 2 and the second with 1 we know we can either win or block, whichever we need to do. Again we will be a little more specific, remember that the computer is the only player to do this so we know its character is an O and its opponent's character is an X.

1. For each row,
   (a) Count the number of O's.
   (b) Count the number of spaces.
   (c) If the number of O's is 2, and the number of spaces is 1, find the location of the space and insert an O.
2. For each column,

   (a) Count the number of O's.
   (b) Count the number of spaces.
   (c) If the number of O's is 2, and the number of spaces is 1, find the location of the space and insert an O.

3. For the diagonal,
   (a) Count the number of O's.
   (b) Count the number of spaces.
   (c) If the number of O's is 2, and the number of spaces is 1, find the location of the space and insert an O.

4. For the counter-diagonal,
   (a) Count the number of O's.
   (b) Count the number of spaces.
   (c) If the number of O's is 2, and the number of spaces is 1, find the location of the space and insert an O.

5. For each row,
   (a) Count the number of X's.
   (b) Count the number of spaces.
   (c) If the number of X's is 2, and the number of spaces is 1, find the location of the space and insert an O.

6. For each column,
   (a) Count the number of X's.
   (b) Count the number of spaces.
   (c) If the number of X's is 2, and the number of spaces is 1, find the location of the space and insert an O.

7. For the diagonal,
   (a) Count the number of X's.
   (b) Count the number of spaces.
   (c) If the number of X's is 2, and the number of spaces is 1, find the location of the space and insert an O.

8. For the counter-diagonal,

   (a) Count the number of X's.
   (b) Count the number of spaces.
   (c) If the number of X's is 2, and the number of spaces is 1, find the location of the space and insert an O.

9. Select a move that will proceed to a win.

If you were reading this carefully you see that we not only need the number of spaces but if there is one we need the position of the space so that we can insert our character. Looks like a good candidate for another method, determining the position of the space in a given row, column, or diagonal.

The last line of the above portion of our algorithm is to progress to a win. This will take a little thought, but again we will emulate what a human would do. You could progress to a win on a row if that row already had one of your characters and there were no opponent characters. Same goes for a column or for a diagonal. If none of these occur then the standard strategy is to go for the center first, if it is taken, go for one of the corners, then finally go for the middle position of the outside rows or columns. Again we will make this more specific, and this will replace the last line of the above algorithm.

1. For each row,
   (a) Count the number of O's.
   (b) Count the number of X's.
   (c) Count the number of spaces.
   (d) If the number of O's is greater than 0, the number of X's is 0, and the number of spaces is greater then 0, find a location of a space and insert an O.
2. For each column,
   (a) Count the number of O's.
   (b) Count the number of X's.
   (c) Count the number of spaces.
   (d) If the number of O's is greater than 0, the number of X's is 0, and the number of spaces is greater then 0, find a location of a space and insert an O.
3. For the diagonal,
   (a) Count the number of O's.

(b) Count the number of X's.

(c) Count the number of spaces.

(d) If the number of O's is greater than 0, the number of X's is 0, and the number of spaces is greater then 0, find a location of a space and insert an O.

4. For the counter-diagonal,

(a) Count the number of O's.

(b) Count the number of X's.

(c) Count the number of spaces.

(d) If the number of O's is greater than 0, the number of X's is 0, and the number of spaces is greater then 0, find a location of a space and insert an O.

5. If the center is free, insert an O.

6. If a corner is free, insert an O in an open corner position.

7. If a side middle free, insert an O in an open side middle position.

If we think about the implementation and consider the previous methods we have already thought about, it looks like we do not need any new methods, we simply use them in different conditional statements. At this point we have everything we need and we move to the implementation. Read through the code carefully, this is a longer program than the ones we have been writing. As you read through the code go back to the algorithms we developed to see the correspondence.

```
1  import java.util.Scanner;
2
3  /*-
4   * ArrayExample009
5   * Example of using a two-dimensional array as a game board.
6   * This program is a single person player game of Tic-Tac-Toe
7   * where the user plays the computer.
8   * Author:  Don Spickler
9   * Date: 3/20/2011
10  */
11
12 public class ArrayExample009 {
13
14     public static void PrintBoard(char A[][]) {
15         for (int i = 0; i < 3; i++) {
16             for (int j = 0; j < 3; j++) {
17                 if (j == 0)
18                     System.out.print(" ");
19
20                 if (j < 2)
21                     System.out.print(A[i][j] + " | ");
22                 else
```

```
23                      System.out.print(A[i][j]);
24              }
25              System.out.println();
26              if (i < 2)
27                  System.out.println("-----------");
28          }
29      }
30
31      public static int RowCount(char A[][], int row, char c) {
32          int count = 0;
33          for (int i = 0; i < 3; i++)
34              if (A[row][i] == c)
35                  count++;
36
37          return count;
38      }
39
40      public static int RowSpacePos(char A[][], int row) {
41          int pos = -1;
42          for (int i = 0; i < 3; i++)
43              if (A[row][i] == ' ')
44                  pos = i;
45
46          return pos;
47      }
48
49      public static int ColCount(char A[][], int col, char c) {
50          int count = 0;
51          for (int i = 0; i < 3; i++)
52              if (A[i][col] == c)
53                  count++;
54
55          return count;
56      }
57
58      public static int ColSpacePos(char A[][], int col) {
59          int pos = -1;
60          for (int i = 0; i < 3; i++)
61              if (A[i][col] == ' ')
62                  pos = i;
63
64          return pos;
65      }
66
67      public static int DiagCount(char A[][], char c) {
68          int count = 0;
69          for (int i = 0; i < 3; i++)
70              if (A[i][i] == c)
71                  count++;
72
73          return count;
74      }
75
76      public static int DiagSpacePos(char A[][]) {
77          int pos = -1;
78          for (int i = 0; i < 3; i++)
79              if (A[i][i] == ' ')
80                  pos = i;
81
82          return pos;
83      }
84
85      public static int CounterDiagCount(char A[][], char c) {
86          int count = 0;
```

```
87          for (int i = 0; i < 3; i++)
88              if (A[2 - i][i] == c)
89                  count++;
90
91          return count;
92      }
93
94      public static int CounterDiagSpacePos(char A[][]) {
95          int pos = -1;
96          for (int i = 0; i < 3; i++)
97              if (A[2 - i][i] == ' ')
98                  pos = i;
99
100         return pos;
101     }
102
103     public static void PlaceO(char A[][]) {
104         // Finish game if possible.
105         for (int i = 0; i < 3; i++) {
106             if ((RowCount(A, i, 'O') == 2) && (RowSpacePos(A, i) >= 0)) {
107                 A[i][RowSpacePos(A, i)] = 'O';
108                 return;
109             }
110         }
111
112         for (int i = 0; i < 3; i++) {
113             if ((ColCount(A, i, 'O') == 2) && (ColSpacePos(A, i) >= 0)) {
114                 A[ColSpacePos(A, i)][i] = 'O';
115                 return;
116             }
117         }
118
119         if ((DiagCount(A, 'O') == 2) && (DiagSpacePos(A) >= 0)) {
120             A[DiagSpacePos(A)][DiagSpacePos(A)] = 'O';
121             return;
122         }
123
124         if ((CounterDiagCount(A, 'O') == 2) && (CounterDiagSpacePos(A) >= 0)) {
125             A[2 - CounterDiagSpacePos(A)][CounterDiagSpacePos(A)] = 'O';
126             return;
127         }
128
129         // Block a win.
130         for (int i = 0; i < 3; i++) {
131             if ((RowCount(A, i, 'X') == 2) && (RowSpacePos(A, i) >= 0)) {
132                 A[i][RowSpacePos(A, i)] = 'O';
133                 return;
134             }
135         }
136
137         for (int i = 0; i < 3; i++) {
138             if ((ColCount(A, i, 'X') == 2) && (ColSpacePos(A, i) >= 0)) {
139                 A[ColSpacePos(A, i)][i] = 'O';
140                 return;
141             }
142         }
143
144         if ((DiagCount(A, 'X') == 2) && (DiagSpacePos(A) >= 0)) {
145             A[DiagSpacePos(A)][DiagSpacePos(A)] = 'O';
146             return;
147         }
148
149         if ((CounterDiagCount(A, 'X') == 2) && (CounterDiagSpacePos(A) >= 0)) {
150             A[2 - CounterDiagSpacePos(A)][CounterDiagSpacePos(A)] = 'O';
```

```
151             return;
152         }
153
154         // Progress to a win.
155         for (int i = 0; i < 3; i++) {
156             if ((RowCount(A, i, 'X') == 0) && (RowCount(A, i, 'O') > 0) && (RowSpacePos(A,
                    i) >= 0)) {
157                 A[i][RowSpacePos(A, i)] = 'O';
158                 return;
159             }
160         }
161
162         for (int i = 0; i < 3; i++) {
163             if ((ColCount(A, i, 'X') == 0) && (ColCount(A, i, 'O') > 0) && (ColSpacePos(A,
                    i) >= 0)) {
164                 A[ColSpacePos(A, i)][i] = 'O';
165                 return;
166             }
167         }
168
169         if ((DiagCount(A, 'X') == 0) && (DiagCount(A, 'O') > 0) && (DiagSpacePos(A) >= 0))
                {
170             A[DiagSpacePos(A)][DiagSpacePos(A)] = 'O';
171             return;
172         }
173
174         if ((CounterDiagCount(A, 'X') == 0) && (CounterDiagCount(A, 'O') > 0) && (
                CounterDiagSpacePos(A) >= 0)) {
175             A[2 - CounterDiagSpacePos(A)][CounterDiagSpacePos(A)] = 'O';
176             return;
177         }
178
179         // Go for the center
180
181         if (A[1][1] == ' ') {
182             A[1][1] = 'O';
183             return;
184         }
185
186         // Go for a corner.
187         if (A[0][0] == ' ') {
188             A[0][0] = 'O';
189             return;
190         }
191
192         if (A[0][2] == ' ') {
193             A[0][2] = 'O';
194             return;
195         }
196
197         if (A[2][0] == ' ') {
198             A[2][0] = 'O';
199             return;
200         }
201
202         if (A[2][2] == ' ') {
203             A[2][2] = 'O';
204             return;
205         }
206
207         // Go to one of the middle row positions.
208
209         if (A[0][1] == ' ') {
210             A[0][1] = 'O';
```

```
211             return;
212         }
213
214         if (A[1][0] == ' ') {
215             A[1][0] = 'O';
216             return;
217         }
218
219         if (A[1][2] == ' ') {
220             A[1][2] = 'O';
221             return;
222         }
223
224         if (A[2][1] == ' ') {
225             A[2][1] = 'O';
226             return;
227         }
228     }
229
230     public static boolean CheckWin(char A[][]) {
231         for (int i = 0; i < 3; i++) {
232             if (RowCount(A, i, 'X') == 3) {
233                 System.out.println("X wins.");
234                 return true;
235             }
236         }
237
238         for (int i = 0; i < 3; i++) {
239             if (ColCount(A, i, 'X') == 3) {
240                 System.out.println("X wins.");
241                 return true;
242             }
243         }
244
245         if (DiagCount(A, 'X') == 3) {
246             System.out.println("X wins.");
247             return true;
248         }
249
250         if (CounterDiagCount(A, 'X') == 3) {
251             System.out.println("X wins.");
252             return true;
253         }
254
255         for (int i = 0; i < 3; i++) {
256             if (RowCount(A, i, 'O') == 3) {
257                 System.out.println("O wins.");
258                 return true;
259             }
260         }
261
262         for (int i = 0; i < 3; i++) {
263             if (ColCount(A, i, 'O') == 3) {
264                 System.out.println("O wins.");
265                 return true;
266             }
267         }
268
269         if (DiagCount(A, 'O') == 3) {
270             System.out.println("O wins.");
271             return true;
272         }
273
274         if (CounterDiagCount(A, 'O') == 3) {
```

```
275             System.out.println("O wins.");
276             return true;
277         }
278
279         int spacecount = 0;
280         for (int i = 0; i < 3; i++)
281             for (int j = 0; j < 3; j++)
282                 if (A[i][j] == ' ')
283                     spacecount++;
284
285         if (spacecount == 0) {
286             System.out.println("No Winner");
287             return true;
288         }
289
290         return false;
291     }
292
293     public static void main(String[] args) {
294         char TTT[][] = new char[3][3];
295         boolean gameover = false;
296         Scanner keyboard = new Scanner(System.in);
297
298         // Clear the board.
299         for (int i = 0; i < 3; i++)
300             for (int j = 0; j < 3; j++)
301                 TTT[i][j] = ' ';
302
303         while (!gameover) {
304             boolean badSelection = true;
305             int row = 1;
306             int col = 1;
307
308             while (badSelection) {
309                 System.out.print("Input the row of next move (1-3): ");
310                 row = keyboard.nextInt();
311                 System.out.print("Input the column of next move (1-3): ");
312                 col = keyboard.nextInt();
313                 badSelection = false;
314                 if ((row < 1) || (row > 3) || (col < 1) || (col > 3))
315                     badSelection = true;
316
317                 if (badSelection)
318                     System.out.println("Invalid selection, please try again.");
319             }
320
321             // array indexes start at 0 so adjust values.
322             row--;
323             col--;
324
325             if (TTT[row][col] == ' ') {
326                 TTT[row][col] = 'X';
327                 gameover = CheckWin(TTT);
328                 if (!gameover) {
329                     PlaceO(TTT);
330                     gameover = CheckWin(TTT);
331                 }
332             } else
333                 System.out.println("Invalid selection, please try again.");
334
335             PrintBoard(TTT);
336         }
337     }
338 }
```

Program Output **ArrayExample009.java**

```
Input the row of next move (1-3): 1
Input the column of next move (1-3): 1
 X |   |
-----------
   | O |
-----------
   |   |
Input the row of next move (1-3): 3
Input the column of next move (1-3): 3
 X |   |
-----------
   | O | O
-----------
   |   | X
Input the row of next move (1-3): 2
Input the column of next move (1-3): 1
 X |   |
-----------
 X | O | O
-----------
 O |   | X
Input the row of next move (1-3): 1
Input the column of next move (1-3): 3
 X | O | X
-----------
 X | O | O
-----------
 O |   | X
Input the row of next move (1-3): 3
Input the column of next move (1-3): 2
No Winner
 X | O | X
-----------
 X | O | O
-----------
 O | X | X
```

10.4.2 Bubble Sort Example

The Bubble Sort is one of many sorting algorithms for taking the contents of an array and sorting them into order. We will discuss this algorithm and a couple other sorting algorithms later on in the notes. For now, it is a goo exercise to trace through the code with a couple examples. This program generates random numbers for the arrays to do these traces use the arrays

7	8	1	5	3

5	2	15	4	17	10	9	2

and

0	1	4	9	16

You really only need to trace through the `BubbleSort` method.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * ArrayExample010
6   * Example of the Bubble Sort for a one-dimensional array.
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
11 public class ArrayExample010 {
12
13     public static void PrintIntArray(int A[]) {
14         for (int i = 0; i < A.length; i++) {
15             System.out.print(A[i] + "  ");
16         }
17         System.out.println();
18     }
19
20     public static void BubbleSort(int A[]) {
21         for (int i = A.length - 1; i > 0; i--) {
22             for (int j = 0; j < i; j++) {
23                 if (A[j] > A[j + 1]) {
24                     int temp = A[j];
25                     A[j] = A[j + 1];
26                     A[j + 1] = temp;
27                 }
28             }
29         }
30     }
31
32     public static void main(String[] args) {
33         Scanner keyboard = new Scanner(System.in);
34         Random generator = new Random();
35         System.out.print("Input the array size: ");
36         int arraySize = keyboard.nextInt();
37
38         int arr[] = new int[arraySize];
39
40         for (int i = 0; i < arr.length; i++)
41             arr[i] = generator.nextInt(1000);
42
43         PrintIntArray(arr);
44         BubbleSort(arr);
45         PrintIntArray(arr);
46     }
47 }
```

Program Output **ArrayExample010.java**

```
Input the array size: 20
728  562  401  966  378  502  641  580  142  537  883  207  874  651  772  329  304  424
    116  615
116  142  207  304  329  378  401  424  502  537  562  580  615  641  651  728  772  874
    883  966
```

10.4.3 Deck of Cards Example

In many computerized card games, like poker and blackjack, you need to have a deck of cards, be able to shuffle the deck, and be able to deal cards from the deck to hands of the players. So if we are to implement a card game on the computer we need to implement a deck of cards. We will also need to implement a single card and probably some type of hand, like a poker hand or a blackjack hand. In this example we create a start on a card object and use an array of cards to simulate a deck of cards. Later we will extract the array portion of this example and create a deck object.

The card class is fairly simple, the data of a playing card in a standard poker deck is just the face value, A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, and a suit H, D, S, C. Now these can be coded in any number of different ways and stored in various ways as well. We will simply store them as strings. If you look at our coding above you will see that we could have stored then as single characters if we represent the 10 differently, such as T. We could have also stored the value and suit as integers, for example, 1 – 13 for the value, 1 = A, 11 = J, 12 = Q, and 13 = K. The suits could be represented as the values 1 – 4 as well. Here we will use strings. The class will, of course, have a constructor, accessor methods and methods for determining the value of a card and if the card is an ace.

In the main the code should be easy to follow, we generate a deck in an order similar to the one the deck is in when it is purchased. We print the deck out, shuffle the deck and then print it out again. The only part that might look a bit strange is shuffling the deck. There are easier ways to randomize the order of a deck of cards but this one is an implementation of a riffle shuffle. A riffle shuffle is where the shuffler divides the deck in half, or nearly so, puts one of the half decks n each hand, face down. Then forms the complete deck by letting one side drop a card or two then the other side a card or two and so on. Our method randomly selects a side, then a number of cards to drop, between 1 and 3, and then drops them into the new deck. Notice that this process is done 7 times in a for loop. It has been proven that using a riffle shuffle, the process must be done 7 times to make the resulting deck "randomized".

```
1  /*-
2   * Card
3   * Card class a card value and card suit.  Also contains accessor methods and
4   * methods for determining the face value of the card.
5   * Author:  Don Spickler
6   * Date: 3/6/2011
7   */
8
9  public class Card {
10     private String value;
11     private String suit;
12
13     public Card(String v, String s) {
14         value = v;
15         suit = s;
16     }
```

```
17
18      public String getValue() {
19          return value;
20      }
21
22      public String getSuit() {
23          return suit;
24      }
25
26      public int getWorth() {
27          if (value.equals("A"))
28              return 1;
29          else if (value.equals("2"))
30              return 2;
31          else if (value.equals("3"))
32              return 3;
33          else if (value.equals("4"))
34              return 4;
35          else if (value.equals("5"))
36              return 5;
37          else if (value.equals("6"))
38              return 6;
39          else if (value.equals("7"))
40              return 7;
41          else if (value.equals("8"))
42              return 8;
43          else if (value.equals("9"))
44              return 9;
45          else
46              return 10;
47      }
48
49      public boolean isAce() {
50          return value.equals("A");
51      }
52  }
```

```
1   import java.util.Random;
2
3   /*-
4    * ArrayExample011
5    * Example of using a one-dimensional array to simulate a deck
6    * of playing cards and shuffling the deck.
7    * Author:  Don Spickler
8    * Date: 3/20/2011
9    */
10
11  public class ArrayExample011 {
12
13      public static void PrintDeck(Card[] deck) {
14          for (int i = 0; i < deck.length; i++) {
15              System.out.print(deck[i].getValue() + deck[i].getSuit() + " ");
16          }
17          System.out.println();
18      }
19
20      public static void ShuffleDeck(Card[] deck) {
21          Random generator = new Random();
22          Card tempdeck[] = new Card[deck.length];
23
24          for (int i = 0; i < 7; i++) {
25              int tempdeckpos = 0;
26              int mid = deck.length / 2;
27              int start = 0;
```

```
28
29              while (tempdeckpos < deck.length) {
30                  int side = generator.nextInt(2);
31
32                  if (side == 0) {
33                      int skip1 = generator.nextInt(3) + 1;
34                      if (start < deck.length / 2) {
35                          if (start + skip1 > deck.length / 2)
36                              skip1 = deck.length / 2 - start;
37
38                          for (int j = start; j < start + skip1; j++) {
39                              tempdeck[tempdeckpos] = deck[j];
40                              tempdeckpos++;
41                          }
42                      }
43                      start += skip1;
44                  } else {
45                      int skip2 = generator.nextInt(3) + 1;
46                      if (mid < deck.length) {
47                          if (mid + skip2 > deck.length)
48                              skip2 = deck.length - mid;
49
50                          for (int j = mid; j < mid + skip2; j++) {
51                              tempdeck[tempdeckpos] = deck[j];
52                              tempdeckpos++;
53                          }
54                      }
55                      mid += skip2;
56                  }
57              }
58
59              for (int j = 0; j < deck.length; j++)
60                  deck[j] = tempdeck[j];
61          }
62      }
63
64      public static void main(String[] args) {
65          Card deck[] = new Card[52];
66
67          int pos = 0;
68          for (int i = 0; i < 4; i++)
69              for (int j = 0; j < 13; j++) {
70                  String suit = "";
71                  String value = "";
72
73                  if (i == 0)
74                      suit = "H";
75                  else if (i == 1)
76                      suit = "D";
77                  else if (i == 2)
78                      suit = "C";
79                  else if (i == 3)
80                      suit = "S";
81
82                  if (j == 0)
83                      value = "A";
84                  else if (j == 10)
85                      value = "J";
86                  else if (j == 11)
87                      value = "Q";
88                  else if (j == 12)
89                      value = "K";
90                  else
91                      value = "" + (j + 1);
```

```
92
93                  deck[pos] = new Card(value, suit);
94                  pos++;
95              }
96          PrintDeck(deck);
97          ShuffleDeck(deck);
98          PrintDeck(deck);
99      }
100 }
```

Program Output **ArrayExample011.java**

```
AH 2H 3H 4H 5H 6H 7H 8H 9H 10H JH QH KH AD 2D 3D 4D 5D 6D 7D 8D 9D 10D JD QD KD AC 2C 3C 4
    C 5C 6C 7C 8C 9C 10C JC QC KC AS 2S 3S 4S 5S 6S 7S 8S 9S 10S JS QS KS
KH 2H AD 6D KS 6S 4S 5S 3D KC 2C 3C 7S 9D 9C KD 10C JH 5H 6H AS 8S 4D 9S QH 2S 5C 6C 10S 7
    C 3S 10D 8C JS JD 2D QS 5D JC 9H 10H QC 4C 7D 8D 3H QD 4H 7H 8H AH AC
```

10.4.4 Array Storage Example

Here is a subtle point about arrays. Did you notice that when we sent an array to a method as a parameter and changed it in the method it changed the original array from the call. If it hadn't, then the deck of cards in the last example would not have been shuffled. This is contrary to how passing a single integer works with a method. For example, if we had the following method,

```
public static void ChangeInt(int a) {
    a = 25;
}
```

and the following lines of code in the main,

```
int b = 3;
System.out.println(b);
ChangeInt(b);
System.out.println(b);
```

The printouts of b in the main would both print out 3. The method took that value of 3 from the call, changed it to 25 in the method but this new value of 25 was not sent back to the variable b in the main. So altering a in the method did not affect the value of b. This is sometimes called passing by *value*, since only the value gets transferred.

An array is different, if we send an array to a method and alter the array, the alteration is also done to the calling array, hence the shuffled deck. This is because an array is passed by *reference*, that is, its memory location is passed into the method and not a copy of its values. So when you change an array inside a method you are changing the same memory locations as you would be if you made the changes in the main. That is, there is only one copy of the array in memory, but with a single

integer there are two copies of the variable in memory, one in the main and one in the method.

This example is a bit on the detailed side so we should go through it line by line. The first two lines in the main declare two integer arrays both that hold 15 data items. The next two lines display what we were talking about with the memory locations. On line 35 note that we print out the array name, no brackets with locations. This will print out the memory location of the beginning of the array. Our output on this run was `[I@55ca6954`. If we run the program again it is very possible that this value will change since array storage can change from run to run. We should point out that this is really not an actual memory location on your computer. You can get that type of information in a language like C and C++, but in Java this is the memory location in the Java Virtual Machine, not on your actual computer. Nonetheless, it is where the array is. Next we send the array to a method and print out the memory location again, notice that it is the same location, so if we were to change the array in the method it would change the data in the same memory locations as the calling array, and hence change that.

The loop on line 39 just populates the array `Arr1` with random values. Line 42 prints out the contents of `Arr1`. Line 45 copies the contents of the array `Arr1` to array `Arr2`. We will see in a moment why we did not simply use an assignment statement for this. The `CopyintArray` method deserves a few comments. This is the first method we have seen that returns an array. Note that the return type is `int[]`, a one dimensional array of integers. The method takes in an array as a parameter, this is exactly the same array as in the call. It creates a new array of the same size, this creation is done at a different memory location than the one that is pointed to be `A`, so now we have two different arrays. The method then copies the contents of `A` to `B`, one by one, and finally returns `B`.

```
public static int[] CopyintArray(int A[]) {
    int[] B = new int[A.length];

    for (int i = 0; i < A.length; i++)
        B[i] = A[i];

    return B;
}
```

Lines 46 through 49 print out the contents of the two arrays and their memory locations, note that they are different. Next we change two locations of `Arr2` and print everything out again, note the changes to `Arr2` and not `Arr1`. Line 61 does something you should never do, or at least not without good reason. We assign `Arr2` to `Arr1`, note the next printouts. The arrays print out identically, as we would expect, but their memory locations are also identical. So we have two variables pointing to

the same spot in memory, so we only have one array at this point. The data that was in `Arr2` is now lost. What is even worse is what happens next. We make two changed to `Arr2`, note in the printouts that this also alters `Arr1`, of course it does, they are the same array in memory. We then call the copy and make a few more changes, the arrays are now at different locations and altering one does not alter the other.

In the last two blocks of code we copy one to the other again, so same contents but different locations. Note the result of the boolean expression `Arr1 == Arr2`, it is false since the memory locations are different. In the last block we do the assignment statement again and this time `Arr1 == Arr2` returns true, since we have the same memory location. This should tell you that you do not want to test if two arrays have identical contents with the expression `Arr1 == Arr2`, instead you want to check each entry of the arrays one by one.

```
1  /*-
2   * ArrayExample012
3   * Example showing how array storage is different from native variable storage.
4   * Author:  Don Spickler
5   * Date: 3/20/2011
6   */
7
8  public class ArrayExample012 {
9
10     public static void PrintintArrayLocation(int A[]) {
11         System.out.println("Array Position In Method: " + A);
12     }
13
14     public static void PrintintArray(int A[], int width) {
15         String format = "%" + width + "d";
16         for (int i = 0; i < A.length; i++)
17             System.out.printf(format, A[i]);
18
19         System.out.println();
20     }
21
22     public static int[] CopyintArray(int A[]) {
23         int[] B = new int[A.length];
24
25         for (int i = 0; i < A.length; i++)
26             B[i] = A[i];
27
28         return B;
29     }
30
31     public static void main(String[] args) {
32         int[] Arr1 = new int[15];
33         int[] Arr2 = new int[15];
34
35         System.out.println("Array Position in Main:   " + Arr1);
36         PrintintArrayLocation(Arr1);
37         System.out.println();
38
39         for (int i = 0; i < Arr1.length; i++)
40             Arr1[i] = (int) (Math.random() * 100) + 1;
41
42         PrintintArray(Arr1, 4);
43         System.out.println();
44
```

```
45          Arr2 = CopyintArray(Arr1);
46          PrintintArray(Arr1, 4);
47          PrintintArray(Arr2, 4);
48          System.out.println("Array 1 Position: " + Arr1);
49          System.out.println("Array 2 Position: " + Arr2);
50          System.out.println();
51
52          Arr2[2] = -3;
53          Arr2[7] = -9;
54
55          PrintintArray(Arr1, 4);
56          PrintintArray(Arr2, 4);
57          System.out.println("Array 1 Position: " + Arr1);
58          System.out.println("Array 2 Position: " + Arr2);
59          System.out.println();
60
61          Arr2 = Arr1;
62
63          PrintintArray(Arr1, 4);
64          PrintintArray(Arr2, 4);
65          System.out.println("Array 1 Position: " + Arr1);
66          System.out.println("Array 2 Position: " + Arr2);
67          System.out.println();
68
69          Arr2[2] = -3;
70          Arr2[7] = -9;
71
72          PrintintArray(Arr1, 4);
73          PrintintArray(Arr2, 4);
74          System.out.println("Array 1 Position: " + Arr1);
75          System.out.println("Array 2 Position: " + Arr2);
76          System.out.println();
77
78          Arr2 = CopyintArray(Arr1);
79          Arr2[3] = -12;
80          Arr2[10] = -3;
81
82          PrintintArray(Arr1, 4);
83          PrintintArray(Arr2, 4);
84          System.out.println("Array 1 Position: " + Arr1);
85          System.out.println("Array 2 Position: " + Arr2);
86          System.out.println();
87
88          Arr2 = CopyintArray(Arr1);
89          PrintintArray(Arr1, 4);
90          PrintintArray(Arr2, 4);
91          System.out.println("Array 1 Position: " + Arr1);
92          System.out.println("Array 2 Position: " + Arr2);
93          System.out.println(Arr1 == Arr2);
94          System.out.println();
95
96          Arr2 = Arr1;
97          System.out.println("Array 1 Position: " + Arr1);
98          System.out.println("Array 2 Position: " + Arr2);
99          System.out.println(Arr1 == Arr2);
100     }
101 }
```

Program Output — ArrayExample012.java

```
Array Position in Main:   [I@55ca6954
Array Position In Method: [I@55ca6954
```

```
  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19

  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19
  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@3146a9a

  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19
  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@3146a9a

  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19
  74  56  36  90  14  91  45  78  49   4  77  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@55ca6954

  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@55ca6954

  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
  74  56  -3 -12  14  91  45  -9  49   4  -3  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@3162a60a

  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
  74  56  -3  90  14  91  45  -9  49   4  77  12  24  60  19
Array 1 Position: [I@55ca6954
Array 2 Position: [I@4382f3da
false

Array 1 Position: [I@55ca6954
Array 2 Position: [I@55ca6954
true
```

10.4.5 Multiple Array Types Example

This example simply shows different arrays of different types. It does show another syntax for defining an array, instead of using the new statement we can out the values of the array in curly brackets. The following statement,

```
int arr2[] = { 2, 4, 3, 6, 5, 8 };
```

will create the array `arr2`, make it size 6, and load in the data, 2, 4, 3, 6, 5, 8, in that order.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * ArrayExample013
6   * Example showing the use of different array types.
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
```

```
11 public class ArrayExample013 {
12
13     public static void PrintIntArray(int A[]) {
14         for (int i = 0; i < A.length; i++) {
15             System.out.print(A[i] + "  ");
16         }
17         System.out.println();
18     }
19
20     public static void PrintDoubleArray(double A[]) {
21         for (int i = 0; i < A.length; i++) {
22             System.out.print(A[i] + "  ");
23         }
24         System.out.println();
25     }
26
27     public static void PrintStringArray(String A[]) {
28         for (int i = 0; i < A.length; i++) {
29             System.out.print(A[i] + "  ");
30         }
31         System.out.println();
32     }
33
34     public static void main(String[] args) {
35         Scanner keyboard = new Scanner(System.in);
36         Random generator = new Random();
37         System.out.print("Input the array size: ");
38         int arraySize = keyboard.nextInt();
39
40         int arr1[] = new int[arraySize];
41
42         PrintIntArray(arr1);
43
44         for (int i = 0; i < arr1.length; i++)
45             arr1[i] = generator.nextInt(1000);
46
47         PrintIntArray(arr1);
48         System.out.println();
49
50         int arr2[] = { 2, 4, 3, 6, 5, 8 };
51         System.out.println(arr2.length);
52         PrintIntArray(arr2);
53         System.out.println();
54
55         int arr3[] = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
56         System.out.println(arr3.length);
57         PrintIntArray(arr3);
58         System.out.println();
59
60         double arr4[] = new double[15];
61         for (int i = 0; i < arr4.length; i++)
62             arr4[i] = generator.nextDouble();
63
64         PrintDoubleArray(arr4);
65         System.out.println();
66
67         double arr5[] = { 2.3, 5.6, 4, -10.3, Math.PI, 1 / 2, 1.0 / 3.0 };
68         System.out.println(arr5.length);
69         PrintDoubleArray(arr5);
70         System.out.println();
71
72         String strArr1[] = { "cat", "dog", "mouse", "rabbit" };
73         System.out.println(strArr1.length);
74         PrintStringArray(strArr1);
```

```
75          System.out.println();
76      }
77  }
```

Program Output **ArrayExample013.java**

```
Input the array size: 5
0  0  0  0  0
556  145  448  832  508

6
2  4  3  6  5  8

12
1  2  3  4  5  6  7  8  9  10  11  12

0.30090584480059535  0.22606268774205351  0.8646500064338853  0.7898626609869165
    0.9202120598076606  0.06562864946379732  0.4167446470685777  0.6987525201854811
    0.5523966415122554  0.8683374415926548  0.23415223451209055  0.6315928478994556
    0.5600543635292937  0.39098210253944543  0.6565505130814907

7
2.3  5.6  4.0  -10.3  3.141592653589793  0.0  0.3333333333333333

4
cat  dog  mouse  rabbit
```

10.4.6 Multiple Array Types and Method Overloading Example

This example is the previous one except that we overloaded the `PrintArray` method.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * ArrayExample014
6   * Example showing the use of different array types, and overloading the PrintArray method
        .
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
11 public class ArrayExample014 {
12
13     public static void PrintArray(int A[]) {
14         for (int i = 0; i < A.length; i++) {
15             System.out.print(A[i] + "  ");
16         }
17         System.out.println();
18     }
19
20     public static void PrintArray(double A[]) {
21         for (int i = 0; i < A.length; i++) {
22             System.out.print(A[i] + "  ");
23         }
24         System.out.println();
25     }
26
```

```
27      public static void PrintArray(String A[]) {
28          for (int i = 0; i < A.length; i++) {
29              System.out.print(A[i] + "  ");
30          }
31          System.out.println();
32      }
33
34      public static void main(String[] args) {
35          Scanner keyboard = new Scanner(System.in);
36          Random generator = new Random();
37          System.out.print("Input the array size: ");
38          int arraySize = keyboard.nextInt();
39
40          int arr1[] = new int[arraySize];
41
42          PrintArray(arr1);
43
44          for (int i = 0; i < arr1.length; i++)
45              arr1[i] = generator.nextInt(1000);
46
47          PrintArray(arr1);
48          System.out.println();
49
50          int arr2[] = { 2, 4, 3, 6, 5, 8 };
51          System.out.println(arr2.length);
52          PrintArray(arr2);
53          System.out.println();
54
55          int arr3[] = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
56          System.out.println(arr3.length);
57          PrintArray(arr3);
58          System.out.println();
59
60          double arr4[] = new double[15];
61          for (int i = 0; i < arr4.length; i++)
62              arr4[i] = generator.nextDouble();
63
64          PrintArray(arr4);
65          System.out.println();
66
67          double arr5[] = { 2.3, 5.6, 4, -10.3, Math.PI, 1 / 2, 1.0 / 3.0 };
68          System.out.println(arr5.length);
69          PrintArray(arr5);
70          System.out.println();
71
72          String strArr1[] = { "cat", "dog", "mouse", "rabbit" };
73          System.out.println(strArr1.length);
74          PrintArray(strArr1);
75          System.out.println();
76      }
77
78  }
```

Program Output — ArrayExample014.java

```
Input the array size: 5
0  0  0  0  0
699  166  442  706  863

6
2  4  3  6  5  8

12
```

```
1  2  3  4  5  6  7  8  9  10  11  12

0.2633738607685602  0.6193460932128056  0.6081499072536326  0.7210534236011001
    0.18985176577681084  0.845737645019744  0.7469594850277619  0.9436781350936808
    0.0026964447424192572  0.6230321369240002  0.1232354583310975  0.2435501511050222
    0.8035231649922354  0.7068849427576559  0.6183152152568981

7
2.3  5.6  4.0  -10.3  3.141592653589793  0.0  0.3333333333333333

4
cat  dog  mouse  rabbit
```

10.4.7 Launch Parameters Example

Once we started this chapter, you may have realized that the main has an array parameter of strings. This is so that id one were to launch the program from the command window in Windows or the terminal in Linux or Mac, they can send information into the program through this string array. We will have no need to do this in this class but we thought we would mention it if you happen to see this again in a later course.

```
1  /*-
2   * ArrayExample015
3   * Example showing the use of input parameters on a program launch.
4   * Author:  Don Spickler
5   * Date: 3/20/2011
6   */
7
8  public class ArrayExample015 {
9      public static void PrintStringArray(String A[]) {
10         for (int i = 0; i < A.length; i++) {
11             System.out.print(A[i] + "  ");
12         }
13         System.out.println();
14     }
15
16     public static void main(String[] args) {
17         System.out.println(args.length);
18         PrintStringArray(args);
19     }
20 }
```

Program Output **ArrayExample015.java**

```
C:\java ArrayExample015 This is a test of the string parameters of main
10
This is a test of the string parameters of main
```

10.4.8 Deck of Cards Example — Revised

We return to the deck of cards example we started several examples ago. Here we have an updated card class, a deck class and a poker hand class. From here it would

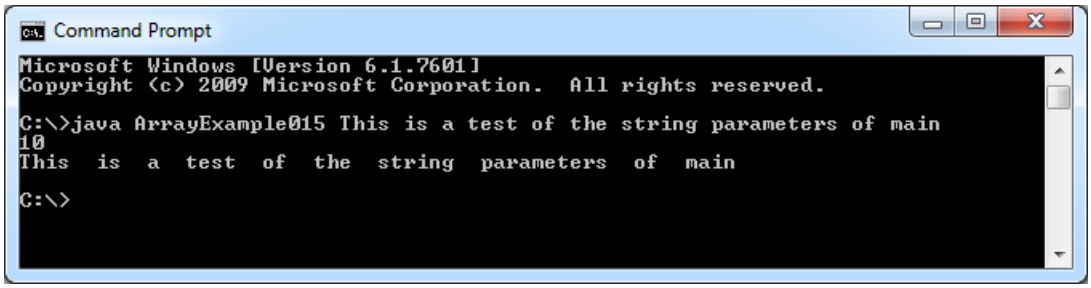


Figure 10.1: Launch Parameters Example Run from the Command Prompt

not be too difficult to create a multi-player game of poker. In the card class we added another `getWorth` method that allows the programmer to set the value of an ace in getting the cards value. This is in anticipation of creating a blackjack program, where the value of an ace can change. There are a few other methods as well that should be self-explanatory.

```
1  /*-
2   * Card
3   * Card class a card value and card suit.  Also contains accessor methods and
4   * methods for determining the face value of the card.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Card {
10     // Data Members
11     private String value;
12     private String suit;
13
14     // Constructor
15     public Card(String v, String s) {
16         value = v;
17         suit = s;
18     }
19
20     public String getValue() {
21         return value;
22     }
23
24     public String getSuit() {
25         return suit;
26     }
27
28     // Determine face value.
29     public int getWorth() {
30         if (value.equals("A"))
31             return 1;
32         else if (value.equals("2"))
33             return 2;
34         else if (value.equals("3"))
35             return 3;
36         else if (value.equals("4"))
37             return 4;
38         else if (value.equals("5"))
39             return 5;
40         else if (value.equals("6"))
```

```
41              return 6;
42          else if (value.equals("7"))
43              return 7;
44          else if (value.equals("8"))
45              return 8;
46          else if (value.equals("9"))
47              return 9;
48          else
49              return 10;
50      }
51
52      // Determine face value, with variable ace.
53      public int getWorth(int ace) {
54          if (value.equals("A"))
55              return ace;
56          else if (value.equals("2"))
57              return 2;
58          else if (value.equals("3"))
59              return 3;
60          else if (value.equals("4"))
61              return 4;
62          else if (value.equals("5"))
63              return 5;
64          else if (value.equals("6"))
65              return 6;
66          else if (value.equals("7"))
67              return 7;
68          else if (value.equals("8"))
69              return 8;
70          else if (value.equals("9"))
71              return 9;
72          else
73              return 10;
74      }
75
76      // Determine if an ace.
77      public boolean isAce() {
78          return value.equals("A");
79      }
80
81      // Check equality of two cards.
82      public boolean equals(Card card2) {
83          return value.equals(card2.getValue()) && suit.equals(card2.getSuit());
84      }
85
86      // String output of card.
87      public String toString() {
88          return value + " " + suit;
89      }
90
91      // String output of card, with or without a space.
92      public String toString(boolean space) {
93          String retstr = value;
94          if (space)
95              retstr += " ";
96          retstr += suit;
97          return retstr;
98      }
99  }
```

The deck class extracts the deck array and methods we saw in the previous example, plus a few more methods. We will discuss the top variable in a moment. The

constructor creates the array of cards in "purchased" order. The `copyDeck` method copies the deck to a new deck, just like the integer array example above. We added a new shuffle method that randomly interchanges two cards in the deck 100 times. The other added methods are for dealing cards off the deck.

When a person deals cards off the top of a deck they physically remove the card from the deck and put it into someone's hand. We could simulate this on the computer but that would require that we change the size of the array, which is a hassle. It is easier to keep the deck as it is and simply change the "pointer" to the top card in the deck. For example, say our deck started out as 4D 5C 7D 10H 2S 10C QS 8S ..., with the 4 of diamonds in position 0 of the deck. If we start the top at 0 then to deal a card we copy the card in the 0 position to a hand, that is the 4 of diamonds is placed in a hand. Then we set the top to 1 (i.e. increment the top). Now the top is pointing to the 5 of clubs, dealing it will increment the top to point at the 7 of diamonds, and so on.

```
1  import java.util.Random;
2
3  /*-
4   * Deck
5   * Deck class uses an array of 52 cards to simulate a deck of cards.
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class Deck {
11     // Data Members
12     private Card deck[] = new Card[52];
13     private int top = 0;
14
15     // Constructor
16     public Deck() {
17         int pos = 0;
18         for (int i = 0; i < 4; i++)
19             for (int j = 0; j < 13; j++) {
20                 String suit = "";
21                 String value = "";
22
23                 if (i == 0)
24                     suit = "H";
25                 else if (i == 1)
26                     suit = "D";
27                 else if (i == 2)
28                     suit = "C";
29                 else if (i == 3)
30                     suit = "S";
31
32                 if (j == 0)
33                     value = "A";
34                 else if (j == 10)
35                     value = "J";
36                 else if (j == 11)
37                     value = "Q";
38                 else if (j == 12)
39                     value = "K";
40                 else
41                     value = "" + (j + 1);
42
```

```
43                  deck[pos] = new Card(value, suit);
44                  pos++;
45              }
46      }
47
48      // Prints the contents of the deck.
49      public void PrintDeck() {
50          for (int i = 0; i < deck.length; i++) {
51              System.out.print(deck[i].toString(false) + " ");
52          }
53          System.out.println();
54      }
55
56      // Makes a copy of the deck.
57      public Deck copyDeck() {
58          Deck tempdeck = new Deck();
59
60          for (int i = 0; i < 52; i++) {
61              tempdeck.deck[i] = new Card(deck[i].getValue(), deck[i].getSuit());
62          }
63
64          return tempdeck;
65      }
66
67      // Simulates a non-perfect Riffle Shuffle of the Deck.
68      public void RiffleShuffleDeck() {
69          Random generator = new Random();
70          Card tempdeck[] = new Card[deck.length];
71
72          for (int i = 0; i < 7; i++) {
73              int tempdeckpos = 0;
74              int mid = deck.length / 2;
75              int start = 0;
76
77              while (tempdeckpos < deck.length) {
78                  int side = generator.nextInt(2);
79
80                  if (side == 0) {
81                      int skip1 = generator.nextInt(3) + 1;
82                      if (start < deck.length / 2) {
83                          if (start + skip1 > deck.length / 2)
84                              skip1 = deck.length / 2 - start;
85
86                          for (int j = start; j < start + skip1; j++) {
87                              tempdeck[tempdeckpos] = deck[j];
88                              tempdeckpos++;
89                          }
90                      }
91                      start += skip1;
92                  } else {
93                      int skip2 = generator.nextInt(3) + 1;
94                      if (mid < deck.length) {
95                          if (mid + skip2 > deck.length)
96                              skip2 = deck.length - mid;
97
98                          for (int j = mid; j < mid + skip2; j++) {
99                              tempdeck[tempdeckpos] = deck[j];
100                             tempdeckpos++;
101                         }
102                     }
103                     mid += skip2;
104                 }
105             }
106
```

```
107             for (int j = 0; j < deck.length; j++)
108                 deck[j] = tempdeck[j];
109         }
110         top = 0;
111     }
112 
113     // Simulates an interchange shuffle of the deck.
114     public void InterchangeShuffleDeck() {
115         Random generator = new Random();
116 
117         for (int i = 0; i < 100; i++) {
118             int card1 = generator.nextInt(52);
119             int card2 = generator.nextInt(52);
120             Card tempcard = deck[card1];
121             deck[card1] = deck[card2];
122             deck[card2] = tempcard;
123         }
124         top = 0;
125     }
126 
127     // Calls one of the two shuffle algorithms.
128     public void ShuffleDeck() {
129         InterchangeShuffleDeck();
130         // RiffleShuffleDeck();
131     }
132 
133     // Simulates dealing a card off the deck.
134     public Card dealCard() {
135         return deck[top++];
136     }
137 
138     // Resets the top of the deck.
139     public void resetTop() {
140         top = 0;
141     }
142 }
```

The poker hand class is similar to the deck class, essentially a hand is a small deck of cards. There are methods for adding to the hand, clearing and printing.

```
1  /*-
2   * PokerHand
3   * PokerHand class uses an array of 5 cards to simulate a single poker-type hand.
4   * Author:  Don Spickler
5   * Date: 3/7/2011
6   */
7 
8  public class PokerHand {
9      // Data Members
10     private Card hand[] = new Card[5];
11     int cardsInHand = 0;
12 
13     // Constructor
14     public PokerHand() {
15         cardsInHand = 0;
16     }
17 
18     // Empty Hand
19     public void clearHand() {
20         cardsInHand = 0;
21     }
22 
23     // Adds a card to the poker hand.
```

```
24      public void addToHand(Card card) {
25          if (cardsInHand < 5)
26              hand[cardsInHand++] = new Card(card.getValue(), card.getSuit());
27      }
28
29      // Prints out the poker hand.
30      public void PrintHand() {
31          for (int i = 0; i < cardsInHand; i++) {
32              System.out.print(hand[i].toString(false) + " ");
33          }
34          System.out.println();
35      }
36  }
```

```
1   /*-
2    * ObjectExample005
3    * Example of using an array inside an object.
4    * Author:  Don Spickler
5    * Date: 3/7/2011
6    */
7
8   public class ObjectExample005 {
9
10      public static void main(String[] args) {
11          Deck cards = new Deck();
12
13          System.out.print("Cards:  ");
14          cards.PrintDeck();
15          cards.ShuffleDeck();
16          System.out.print("Cards:  ");
17          cards.PrintDeck();
18
19          System.out.println();
20
21          Deck cards3 = cards.copyDeck();
22          System.out.print("Cards:  ");
23          cards.PrintDeck();
24          System.out.print("Cards3:  ");
25          cards3.PrintDeck();
26
27          System.out.println();
28
29          cards.ShuffleDeck();
30          System.out.print("Cards:  ");
31          cards.PrintDeck();
32          System.out.print("Cards3:  ");
33          cards3.PrintDeck();
34
35          System.out.println();
36
37          PokerHand hand1 = new PokerHand();
38          PokerHand hand2 = new PokerHand();
39          cards.RiffleShuffleDeck();
40          System.out.print("Cards:  ");
41          cards.PrintDeck();
42
43          for (int i = 0; i < 5; i++) {
44              hand1.addToHand(cards.dealCard());
45              hand2.addToHand(cards.dealCard());
46          }
47
48          System.out.println();
49          hand1.PrintHand();
50          hand2.PrintHand();
```

```
51
52          hand1.addToHand(cards.dealCard());
53          hand2.addToHand(cards.dealCard());
54          System.out.println();
55
56          hand1.PrintHand();
57          hand2.PrintHand();
58
59          System.out.println();
60
61          hand1.clearHand();
62          hand2.clearHand();
63          cards.RiffleShuffleDeck();
64          System.out.print("Cards:  ");
65          cards.PrintDeck();
66
67          System.out.println();
68          hand1.addToHand(cards.dealCard());
69          hand2.addToHand(cards.dealCard());
70          hand1.PrintHand();
71          hand2.PrintHand();
72
73          hand1.addToHand(cards.dealCard());
74          hand2.addToHand(cards.dealCard());
75          hand1.PrintHand();
76          hand2.PrintHand();
77
78          hand1.addToHand(cards.dealCard());
79          hand2.addToHand(cards.dealCard());
80          hand1.PrintHand();
81          hand2.PrintHand();
82      }
83  }
```

Program Output — ObjectExample005.java

```
Cards:  AH 2H 3H 4H 5H 6H 7H 8H 9H 10H JH QH KH AD 2D 3D 4D 5D 6D 7D 8D 9D 10D JD QD KD AC
     2C 3C 4C 5C 6C 7C 8C 9C 10C JC QC KC AS 2S 3S 4S 5S 6S 7S 8S 9S 10S JS QS KS
Cards:  4D 5C 7D 10H 2S 10C QS 8S 9H 4C JH 7H 9D JC JS 8C 6D 6H KD 2C 5H KH 4H AS 2H 7C 6C
     3H AC 3D 10S JD QC 7S 10D KC 5D 3S AH 9S 3C 2D 4S 9C QH 8H 5S KS AD QD 8D 6S

Cards:  4D 5C 7D 10H 2S 10C QS 8S 9H 4C JH 7H 9D JC JS 8C 6D 6H KD 2C 5H KH 4H AS 2H 7C 6C
     3H AC 3D 10S JD QC 7S 10D KC 5D 3S AH 9S 3C 2D 4S 9C QH 8H 5S KS AD QD 8D 6S
Cards3:  4D 5C 7D 10H 2S 10C QS 8S 9H 4C JH 7H 9D JC JS 8C 6D 6H KD 2C 5H KH 4H AS 2H 7C 6
    C 3H AC 3D 10S JD QC 7S 10D KC 5D 3S AH 9S 3C 2D 4S 9C QH 8H 5S KS AD QD 8D 6S

Cards:  5H AS 7D JH 9S 2C AD QS 4S 8C KC 5D 2H 7H 3H KS 2S 9D 8S 3D 4C 7S 5S JD 7C 8H QH
    JS 9C 10C 4H AC QC 10S JC 5C KD 10D AH KH 6C 9H QD 3S 8D 4D 2D 6H 10H 6D 3C 6S
Cards3:  4D 5C 7D 10H 2S 10C QS 8S 9H 4C JH 7H 9D JC JS 8C 6D 6H KD 2C 5H KH 4H AS 2H 7C 6
    C 3H AC 3D 10S JD QC 7S 10D KC 5D 3S AH 9S 3C 2D 4S 9C QH 8H 5S KS AD QD 8D 6S

Cards:  5H 2C 7C 4C AD 6S 8C 2D 8H 2H 7H 10S 3S 6H AH KH 6C AS 10H 6D 9C 10C QS 4S QD 8S
    QC 5D 3D 4H 8D AC 9D 3C 7D JH 9S JC 5C 5S KD KC 9H 7S QH 10D JD 3H KS 2S JS 4D

5H 7C AD 8C 8H
2C 4C 6S 2D 2H

5H 7C AD 8C 8H
2C 4C 6S 2D 2H

Cards:  5D 3D AD JS QC JC 9H 4D QH KH 5H QS 6S 8C 2D 7S AH 6D 9S 8H 4S 7H 10S 3S 10D 9C 4H
     5C 2H JH 2C 5S JD QD 8S 6C AS 10C KD KC 2S 7C 3H KS 6H 4C 10H 8D AC 9D 3C 7D
```

```
5D
3D
5D AD
3D JS
5D AD QC
3D JS JC
```

10.4.9 Blackjack Example

This program allows the user to play the game of blackjack against the computer. The computer is the dealer and the user is the player. The dealer is restricted in its play by some, but not all, of the dealer rules in most casinos. There is no betting in this implementation.

The dealer will stand on,

1. A hand of 18 or higher, unless the player will win.
2. A hard 17, unless the player will win.
3. If the player has gone bust.
4. A soft 17, will randomly stand or hit.

Winner determination is done by,

1. If one player has gone bust but the other has not. The one who has not gone bust wins,.
2. A natural 21 (A+10, J, Q, K)
3. 5 cards under 21
4. Hand value

The card and deck classes are the same as in the previous example.

```
1  /*-
2   * Card
3   * Card class a card value and card suit.  Also contains accessor methods and
4   * methods for determining the face value of the card.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Card {
10     // Data Members
11     private String value;
12     private String suit;
13
```

```
14      // Constructor
15      public Card(String v, String s) {
16          value = v;
17          suit = s;
18      }
19
20      public String getValue() {
21          return value;
22      }
23
24      public String getSuit() {
25          return suit;
26      }
27
28      // Determine face value.
29      public int getWorth() {
30          if (value.equals("A"))
31              return 1;
32          else if (value.equals("2"))
33              return 2;
34          else if (value.equals("3"))
35              return 3;
36          else if (value.equals("4"))
37              return 4;
38          else if (value.equals("5"))
39              return 5;
40          else if (value.equals("6"))
41              return 6;
42          else if (value.equals("7"))
43              return 7;
44          else if (value.equals("8"))
45              return 8;
46          else if (value.equals("9"))
47              return 9;
48          else
49              return 10;
50      }
51
52      // Determine face value, with variable ace.
53      public int getWorth(int ace) {
54          if (value.equals("A"))
55              return ace;
56          else if (value.equals("2"))
57              return 2;
58          else if (value.equals("3"))
59              return 3;
60          else if (value.equals("4"))
61              return 4;
62          else if (value.equals("5"))
63              return 5;
64          else if (value.equals("6"))
65              return 6;
66          else if (value.equals("7"))
67              return 7;
68          else if (value.equals("8"))
69              return 8;
70          else if (value.equals("9"))
71              return 9;
72          else
73              return 10;
74      }
75
76      // Determine if an ace.
77      public boolean isAce() {
```

```
78          return value.equals("A");
79      }
80
81      // Check equality of two cards.
82      public boolean equals(Card card2) {
83          return value.equals(card2.getValue()) && suit.equals(card2.getSuit());
84      }
85
86      // String output of card.
87      public String toString() {
88          return value + " " + suit;
89      }
90
91      // String output of card, with or without a space.
92      public String toString(boolean space) {
93          String retstr = value;
94          if (space)
95              retstr += " ";
96          retstr += suit;
97          return retstr;
98      }
99  }
```

```
1   import java.util.Random;
2
3   /*-
4    * Deck
5    * Deck class uses an array of 52 cards to simulate a deck of cards.
6    * Author:  Don Spickler
7    * Date: 3/7/2011
8    */
9
10  public class Deck {
11      // Data Members
12      private Card deck[] = new Card[52];
13      private int top = 0;
14
15      // Constructor
16      public Deck() {
17          int pos = 0;
18          for (int i = 0; i < 4; i++)
19              for (int j = 0; j < 13; j++) {
20                  String suit = "";
21                  String value = "";
22
23                  if (i == 0)
24                      suit = "H";
25                  else if (i == 1)
26                      suit = "D";
27                  else if (i == 2)
28                      suit = "C";
29                  else if (i == 3)
30                      suit = "S";
31
32                  if (j == 0)
33                      value = "A";
34                  else if (j == 10)
35                      value = "J";
36                  else if (j == 11)
37                      value = "Q";
38                  else if (j == 12)
39                      value = "K";
40                  else
41                      value = "" + (j + 1);
```

```
42
43                  deck[pos] = new Card(value, suit);
44                  pos++;
45              }
46      }
47
48      // Prints the contents of the deck.
49      public void PrintDeck() {
50          for (int i = 0; i < deck.length; i++) {
51              System.out.print(deck[i].toString(false) + " ");
52          }
53          System.out.println();
54      }
55
56      // Makes a copy of the deck.
57      public Deck copyDeck() {
58          Deck tempdeck = new Deck();
59
60          for (int i = 0; i < 52; i++) {
61              tempdeck.deck[i] = new Card(deck[i].getValue(), deck[i].getSuit());
62          }
63
64          return tempdeck;
65      }
66
67      // Simulates a non-perfect Riffle Shuffle of the Deck.
68      public void RiffleShuffleDeck() {
69          Random generator = new Random();
70          Card tempdeck[] = new Card[deck.length];
71
72          for (int i = 0; i < 7; i++) {
73              int tempdeckpos = 0;
74              int mid = deck.length / 2;
75              int start = 0;
76
77              while (tempdeckpos < deck.length) {
78                  int side = generator.nextInt(2);
79
80                  if (side == 0) {
81                      int skip1 = generator.nextInt(3) + 1;
82                      if (start < deck.length / 2) {
83                          if (start + skip1 > deck.length / 2)
84                              skip1 = deck.length / 2 - start;
85
86                          for (int j = start; j < start + skip1; j++) {
87                              tempdeck[tempdeckpos] = deck[j];
88                              tempdeckpos++;
89                          }
90                      }
91                      start += skip1;
92                  } else {
93                      int skip2 = generator.nextInt(3) + 1;
94                      if (mid < deck.length) {
95                          if (mid + skip2 > deck.length)
96                              skip2 = deck.length - mid;
97
98                          for (int j = mid; j < mid + skip2; j++) {
99                              tempdeck[tempdeckpos] = deck[j];
100                             tempdeckpos++;
101                         }
102                     }
103                     mid += skip2;
104                 }
105             }
```

```
106
107             for (int j = 0; j < deck.length; j++)
108                 deck[j] = tempdeck[j];
109         }
110         top = 0;
111     }
112
113     // Simulates an interchange shuffle of the deck.
114     public void InterchangeShuffleDeck() {
115         Random generator = new Random();
116
117         for (int i = 0; i < 100; i++) {
118             int card1 = generator.nextInt(52);
119             int card2 = generator.nextInt(52);
120             Card tempcard = deck[card1];
121             deck[card1] = deck[card2];
122             deck[card2] = tempcard;
123         }
124         top = 0;
125     }
126
127     // Calls one of the two shuffle algorithms.
128     public void ShuffleDeck() {
129         InterchangeShuffleDeck();
130         // RiffleShuffleDeck();
131     }
132
133     // Simulates dealing a card off the deck.
134     public Card dealCard() {
135         return deck[top++];
136     }
137
138     // Resets the top of the deck.
139     public void resetTop() {
140         top = 0;
141     }
142 }
```

The poker hand class from the previous example has been changed to the hand class. We have added several methods that are specific to the game of blackjack, specifically a method that prints out a hand where the first card is face down.

```
1  /*-
2   * Hand
3   * Hand class uses an array of 5 cards to simulate a single blackjack hand.
4   * Author:  Don Spickler
5   * Date: 3/7/2011
6   */
7
8  public class Hand {
9      // Data Members
10     private Card hand[] = new Card[5];
11     private int cardsInHand = 0;
12
13     // Constructor
14     public Hand() {
15         cardsInHand = 0;
16     }
17
18     // Empty Hand
19     public void clearHand() {
20         cardsInHand = 0;
21     }
```

```
22
23      // Returns the number of cards in the hand.
24      public int getNumCards() {
25          return cardsInHand;
26      }
27
28      //  Gets a specific card from the hand.
29      public Card getCard(int i) {
30          if (i < cardsInHand)
31              return hand[i];
32
33          return null;
34      }
35
36      // Adds a card into the hand.
37      public void addToHand(Card card) {
38          if (cardsInHand < 5)
39              hand[cardsInHand++] = new Card(card.getValue(), card.getSuit());
40      }
41
42      // Prints out the hand.
43      public void PrintHand() {
44          for (int i = 0; i < cardsInHand; i++) {
45              System.out.printf("%5s", hand[i].toString(false));
46          }
47          System.out.println();
48      }
49
50      // Prints out the hand using a * for the first card.
51      public void PrintBlackJackHand() {
52          for (int i = 0; i < cardsInHand; i++) {
53              if (i < 1)
54                  System.out.printf("%5s", "*");
55              else
56                  System.out.printf("%5s", hand[i].toString(false));
57          }
58          System.out.println();
59      }
60
61      // Returns the worth of a hand using the input value for the ace.
62      public int getWorth(int ace) {
63          int worth = 0;
64          for (int i = 0; i < cardsInHand; i++) {
65              worth += hand[i].getWorth(ace);
66          }
67
68          return worth;
69      }
70
71      // Determines if the hand contains an ace.
72      public boolean hasAce() {
73          for (int i = 0; i < cardsInHand; i++) {
74              if (hand[i].isAce())
75                  return true;
76          }
77
78          return false;
79      }
80  }
```

The `BlackJack` class is the main driver to the program. The algorithm here is fairly straightforward. There are methods to implement the decision for the dealer

to hit or stand, as described above, and to determine the winner of the hand, also described above. In addition, there are methods to determine if a player has gone bust, and menu selections.

1. While the player continues to hit, and the game is not over because of the 5 card rule or one player visibly going bust.
   (a) Display the two hands, dealer's first card is face down.
   (b) Ask the player if they want to hit or stand. If they wish to hit, deal another card to the hand.
   (c) Determine if the dealer should stand or hit. If the dealer wishes to hit, deal another card to the hand.
2. If dealer knows they will lose, hit until a win or bust.
3. Determine and display winner.
4. Ask to play another game.

```
1  import java.util.Scanner;
2
3  /*-
4   * BlackJack
5   * This program allows the user to play the game of blackjack against the computer.
6   * The computer is the dealer and the user is the player.  The dealer is restricted
7   * in its play by some, but not all, of the dealer rules in most casinos.  There is
8   * no betting in this implementation.
9   * The dealer will stand on
10  * 1. A hand of 18 or higher, unless the player will win.
11  * 2. A hard 17, unless the player will win.
12  * 3. If the player has gone bust.
13  * 4. A soft 17, will randomly stand or hit.
14  *
15  * Winner determination is done by,
16  * 1. If one player has gone bust but the other has not.  The one who has not gone
17  * bust wins.
18  * 2. A natural 21 (A+{10, J, Q, K}).
19  * 3. 5 cards under 21.
20  * 4. Hand value.
21  *
22  * Author:  Don Spickler
23  * Date: 3/11/2011
24  */
25
26 public class BlackJack {
27
28     public static void PrintHands(Hand Dealer, Hand Player) {
29         System.out.print("Dealer: ");
30         Dealer.PrintBlackJackHand();
31
32         System.out.print("Player: ");
33         Player.PrintHand();
34     }
35
36     public static void PrintHandsUp(Hand Dealer, Hand Player) {
```

```
37          System.out.print("Dealer: ");
38          Dealer.PrintHand();
39
40          System.out.print("Player: ");
41          Player.PrintHand();
42      }
43
44      public static boolean DetermineDealerStand(Hand Dealer, Hand Player) {
45          boolean dealerStand = false;
46
47          int worth1 = Dealer.getWorth(1);
48          int worth11 = Dealer.getWorth(11);
49          int handvalue = worth11;
50          if (handvalue > 21)
51              handvalue = worth1;
52
53          if (Dealer.getNumCards() == 5)
54              return true;
55
56          if (worth1 == 7 && Dealer.hasAce()) {
57              if ((int) (Math.random() * 2) == 0)
58                  dealerStand = true;
59          } else if (worth11 >= 17 && worth11 <= 21) {
60              dealerStand = true;
61          } else if (worth1 >= 17) {
62              dealerStand = true;
63          }
64
65          int win = Winner(Dealer, Player);
66          boolean playerbust = Bust(Player, false);
67          boolean dealerbust = Bust(Dealer, false);
68
69          if (win != 2 && handvalue < 21)
70              dealerStand = false;
71
72          if (playerbust || dealerbust)
73              dealerStand = true;
74
75          return dealerStand;
76      }
77
78      public static boolean Bust(Hand TheHand, boolean dealer) {
79          boolean bust = false;
80
81          if (dealer) {
82              int dealerworth1 = 0;
83              int dealerworth11 = 0;
84
85              for (int i = 1; i < TheHand.getNumCards(); i++) {
86                  dealerworth1 += TheHand.getCard(i).getWorth(1);
87                  dealerworth11 += TheHand.getCard(i).getWorth(11);
88
89                  if (dealerworth1 > 21 && dealerworth11 > 21)
90                      bust = true;
91              }
92          } else {
93              if (TheHand.getWorth(1) > 21 && TheHand.getWorth(11) > 21)
94                  bust = true;
95          }
96
97          return bust;
98      }
99
100     public static int Menu() {
```

```
101         int Selection = 0;
102 
103         Scanner kb = new Scanner(System.in);
104         do {
105             System.out.println();
106             System.out.println("1. Hit");
107             System.out.println("2. Stand");
108             System.out.println();
109             System.out.print("Selection: ");
110             Selection = kb.nextInt();
111 
112             if (Selection != 1 && Selection != 2) {
113                 System.out.println("Invalid selection, please try again.");
114             }
115 
116         } while (Selection != 1 && Selection != 2);
117         System.out.println();
118 
119         return Selection;
120     }
121 
122     public static boolean PlayAgainMenu() {
123         String Selection = "";
124 
125         Scanner kb = new Scanner(System.in);
126         do {
127             System.out.println();
128             System.out.print("Would you like to play another game?  (Y/N):  ");
129             Selection = kb.nextLine();
130 
131             if (Selection.length() > 0)
132                 Selection = Selection.substring(0, 1).toUpperCase();
133 
134             if (!Selection.equals("Y") && !Selection.equals("N")) {
135                 System.out.println("Invalid selection, please try again.");
136             }
137 
138         } while (!Selection.equals("Y") && !Selection.equals("N"));
139 
140         if (Selection.equals("Y"))
141             return true;
142         else
143             return false;
144     }
145 
146     public static int Winner(Hand Dealer, Hand Player) {
147         // 0 - draw, 1 - player wins, 2 - dealer wins
148 
149         boolean playerNatural21 = false;
150         boolean dealerNatural21 = false;
151         boolean player5Under21 = false;
152         boolean dealer5Under21 = false;
153 
154         int playerWorth = Player.getWorth(11);
155         if (playerWorth > 21)
156             playerWorth = Player.getWorth(1);
157 
158         int dealerWorth = Dealer.getWorth(11);
159         if (dealerWorth > 21)
160             dealerWorth = Dealer.getWorth(1);
161 
162         // Take care of > 21 loss for player, dealer or both.
163         if (playerWorth > 21 && dealerWorth > 21)
164             return 0;
```

```
165         else if (dealerWorth > 21)
166             return 1;
167         else if (playerWorth > 21)
168             return 2;
169 
170         if (playerWorth == 21 && Player.getNumCards() == 2)
171             playerNatural21 = true;
172 
173         if (dealerWorth == 21 && Dealer.getNumCards() == 2)
174             dealerNatural21 = true;
175 
176         if (playerWorth < 21 && Player.getNumCards() == 5)
177             player5Under21 = true;
178 
179         if (dealerWorth < 21 && Dealer.getNumCards() == 5)
180             dealer5Under21 = true;
181 
182         if (playerNatural21 && dealerNatural21)
183             return 0;
184         else if (playerNatural21)
185             return 1;
186         else if (dealerNatural21)
187             return 2;
188 
189         if (player5Under21 && !dealer5Under21)
190             return 1;
191         else if (dealer5Under21 && !player5Under21)
192             return 2;
193 
194         if (playerWorth == dealerWorth)
195             return 0;
196         else if (playerWorth > dealerWorth)
197             return 1;
198         else
199             return 2;
200     }
201 
202     public static void main(String[] args) {
203         Hand Dealer = new Hand();
204         Hand Player = new Hand();
205         Deck Cards = new Deck();
206         int playerwins = 0;
207         int dealerwins = 0;
208         int gamesplayed = 0;
209 
210         do {
211             int Selection = 0;
212             boolean DealerStay = false;
213             boolean playerbust = true;
214             boolean dealerbust = true;
215 
216             Cards.ShuffleDeck();
217 
218             gamesplayed++;
219 
220             System.out.println();
221             System.out.println("Game " + gamesplayed);
222             System.out.println("--------------------");
223             System.out.println();
224 
225             Player.clearHand();
226             Dealer.clearHand();
227 
228             Player.addToHand(Cards.dealCard());
```

```
229             Dealer.addToHand(Cards.dealCard());
230             Player.addToHand(Cards.dealCard());
231             Dealer.addToHand(Cards.dealCard());
232
233             do {
234                 PrintHands(Dealer, Player);
235                 Selection = Menu();
236
237                 if (Selection == 1)
238                     Player.addToHand(Cards.dealCard());
239
240                 playerbust = Bust(Player, false);
241                 DealerStay = DetermineDealerStand(Dealer, Player);
242
243                 if (!DealerStay)
244                     Dealer.addToHand(Cards.dealCard());
245
246                 dealerbust = Bust(Dealer, true);
247             } while (Selection != 2 && Player.getNumCards() < 5 && !dealerbust && !
                    playerbust);
248
249             while (!DetermineDealerStand(Dealer, Player)) {
250                 Dealer.addToHand(Cards.dealCard());
251             }
252
253             System.out.println();
254             PrintHandsUp(Dealer, Player);
255
256             int winner = Winner(Dealer, Player);
257             if (winner == 1) {
258                 playerwins++;
259                 System.out.println("Player Won");
260             } else if (winner == 2) {
261                 dealerwins++;
262                 System.out.println("Dealer Won");
263             } else {
264                 System.out.println("It was a Draw");
265             }
266         } while (PlayAgainMenu());
267
268         System.out.println();
269         System.out.println("Games Played = " + gamesplayed);
270         System.out.println("Player Wins = " + playerwins);
271         System.out.println("Dealer Wins = " + dealerwins);
272         System.out.println("Draws = " + (gamesplayed - dealerwins - playerwins));
273         System.out.println();
274     }
275 }
```

Program Output — BlackJack.java

```
Game 1
--------------------

Dealer:     *   7S
Player:    4D   7C

1. Hit
2. Stand

Selection: 1

Dealer:     *   7S   4S
```

```
Player:    4D   7C   8C

1. Hit
2. Stand

Selection: 2


Dealer:    5S   7S   4S   JC
Player:    4D   7C   8C
Player Won

Would you like to play another game?  (Y/N):  y

Game 2
--------------------

Dealer:     *   QS
Player:   10S   7S

1. Hit
2. Stand

Selection: 2


Dealer:    AH   QS
Player:   10S   7S
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 3
--------------------

Dealer:     *   3C
Player:    2S   8S

1. Hit
2. Stand

Selection: 1

Dealer:     *   3C   JS
Player:    2S   8S   4H

1. Hit
2. Stand

Selection: 1


Dealer:    3S   3C   JS
Player:    2S   8S   4H   KH
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 4
--------------------

Dealer:     *   JH
Player:    KD   2S
```

```
1. Hit
2. Stand

Selection: 1

Dealer:     *   JH   2C
Player:    KD   2S   3S

1. Hit
2. Stand

Selection: 1


Dealer:    5C   JH   2C
Player:    KD   2S   3S   8D
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 5
--------------------

Dealer:     *   QH
Player:   10D   2C

1. Hit
2. Stand

Selection: 1


Dealer:    QD   QH
Player:   10D   2C   JD
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 6
--------------------

Dealer:     *   4S
Player:    2C   7D

1. Hit
2. Stand

Selection: 1

Dealer:     *   4S  10D
Player:    2C   7D   6S

1. Hit
2. Stand

Selection: 1


Dealer:    8S   4S  10D
Player:    2C   7D   6S   9D
It was a Draw

Would you like to play another game?  (Y/N):  y
```

```
Game 7
--------------------

Dealer:     *   9S
Player:    8S   QH

1. Hit
2. Stand

Selection: 2


Dealer:    9D   9S   QC
Player:    8S   QH
Player Won

Would you like to play another game?  (Y/N):  y

Game 8
--------------------

Dealer:     *  10D
Player:    2D   6D

1. Hit
2. Stand

Selection: 1

Dealer:     *  10D   3C
Player:    2D   6D   4S

1. Hit
2. Stand

Selection: 1


Dealer:    5D  10D   3C
Player:    2D   6D   4S  10H
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 9
--------------------

Dealer:     *   AH
Player:    AS   AD

1. Hit
2. Stand

Selection: 1

Dealer:     *   AH   8C
Player:    AS   AD   6C

1. Hit
2. Stand

Selection: 1

Dealer:     *   AH   8C   9D
```

```
Player:    AS   AD   6C   4H

1. Hit
2. Stand

Selection: 1


Dealer:    4D   AH   8C   9D
Player:    AS   AD   6C   4H   5D
Player Won

Would you like to play another game?  (Y/N):  y

Game 10
--------------------

Dealer:     *   4D
Player:    JC   4C

1. Hit
2. Stand

Selection: 1


Dealer:    6H   4D
Player:    JC   4C   KS
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 11
--------------------

Dealer:     *   9H
Player:    5S   KD

1. Hit
2. Stand

Selection: 1

Dealer:     *   9H   2H
Player:    5S   KD   4C

1. Hit
2. Stand

Selection: 2


Dealer:    3S   9H   2H   6S
Player:    5S   KD   4C
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 12
--------------------

Dealer:     *   QD
Player:    4H   KH
```

```
1. Hit
2. Stand

Selection: 2


Dealer:    4C   QD   3S
Player:    4H   KH
Dealer Won

Would you like to play another game?  (Y/N):  y

Game 13
--------------------

Dealer:     *   QC
Player:    QH   8D

1. Hit
2. Stand

Selection: 2


Dealer:    5D   QC   3S   9D
Player:    QH   8D
Player Won

Would you like to play another game?  (Y/N):  y

Game 14
--------------------

Dealer:     *   5D
Player:    QH   JS

1. Hit
2. Stand

Selection: 2


Dealer:    KH   5D   7S
Player:    QH   JS
Player Won

Would you like to play another game?  (Y/N):  y

Game 15
--------------------

Dealer:     *   4C
Player:    JS  10S

1. Hit
2. Stand

Selection: 2


Dealer:    4S   4C   KD   6H
Player:    JS  10S
Player Won
```

```
Would you like to play another game?  (Y/N):  y

Game 16
--------------------

Dealer:     *   2C
Player:    2D   7H

1. Hit
2. Stand

Selection: 1

Dealer:     *   2C   6H
Player:    2D   7H   3C

1. Hit
2. Stand

Selection: 1

Dealer:     *   2C   6H   8S
Player:    2D   7H   3C   3H

1. Hit
2. Stand

Selection: 1


Dealer:    6D   2C   6H   8S
Player:    2D   7H   3C   3H   JS
It was a Draw

Would you like to play another game?  (Y/N):  y

Game 17
--------------------

Dealer:     *   JC
Player:   10S   8C

1. Hit
2. Stand

Selection: 2


Dealer:    4S   JC   JS
Player:   10S   8C
Player Won

Would you like to play another game?  (Y/N):  n

Games Played = 17
Player Wins = 7
Dealer Wins = 8
Draws = 2
```

Chapter 11

Searching & Sorting

11.1 Introduction

Two things we tend to do fairly often with one-dimensional arrays is search them and sort them, if the data in the arrays is of a type that we can sort. In these examples we will be using arrays of integers, but one could easily convert the code to work with floats, doubles, and strings. With a little more work, one co revise this code to work with a programmer-defined object, like the employee class from the chapter on objects. The algorithms for each type of search and sort can be adjusted to work with any data type that can be compared, that is, given any two values of the data type can we determine if one is less than the other or not? So in the case of numbers like integers and real numbers, this is easy, we use the less than comparison symbol, $<$. In the case of strings we use the string compareTo function. In the case of the Employee class, we would need to create our own method for this. We could use the compareTo on the employee name, or we could use $<$ on the wage or weekly pay, just to name a few. It would depend what we want to sort by.

There are many ways to search and sort a list of values, some are rather simplistic and others can be quite complex. In addition, some methods are faster than others. Here we look at two searching algorithms, the linear search and the binary search. We will consider four sorting algorithms, the bubble sort, a modification of the bubble sort, the insertion sort, and the selection sort. If you continue your study of computer science you will look at more sophisticated methods for both searching and sorting.

11.2 Searching Algorithms Example

Linear Search Algorithm: Look at each item in the array in turn, and check whether that item is the one you are looking for. If so, the search is finished and

return the array index of the match. If you look at every item without finding the one you want, then you can be sure that the item is not in the array, in this case return a -1 to signify that the object was not found. Writing this out in algorithmic form,

1. For each entry in the array check its equality with the target value.
   (a) If the target item matches return the array index the target item was in.
   (b) If the target item does not match move onto the next array entry.
2. If the end of the array is reached without a match then return -1 to indicate a failed search.

Binary Search Algorithm: This only works for an array that is sorted from smallest to largest. The basic operation is to look at the item in the middle of the range. If this item is greater than the target value N, then the second half of the range can be eliminated. If it is less than N, then the first half of the range can be eliminated. If the number in the middle just happens to be N exactly, then the search is finished and the method should return the index of the middle, where the target value is in the array. If the size of the range decreases to zero, then the number N does not occur in the array, and as with the linear search we return -1 to signify that the object was not found. Writing this out in algorithmic form,

1. Set the lowest value to 0 and the highest value to the last position of the array (length - 1). So we are always searching for the target N between the lowest and highest positions.
2. Calculate the location between the lowest and highest position (i.e. the middle of the list).
3. If N is equal to the entry in the middle position then we have found the target and return the middle position. The function will end at this point.
4. If N is less than the value in the middle position then if N is in the list it is in the first half of the list, so we discard the second the half of the list by setting highest to middle - 1.
5. If N is greater than the value in the middle position then if N is in the list it is in the second half of the list, so we discard the first the half of the list by setting lowest to middle + 1.
6. If N is not in the list then at some point the lowest value will exceed the highest value, in this case return -1 to indicate a failed search.

If we cut out the explanations we can revise the last algorithm to a form that is closer to the code we will write.

1. Set the lowest value to 0 and the highest value to the last position of the array (length - 1).
2. While the lowest value is smaller than the highest value do the following.
   (a) Calculate the middle location.
   (b) If N is equal to the entry in the middle position then return the middle position.
   (c) If N is less than the value in the middle position then set highest to middle - 1.
   (d) If N is greater than the value in the middle position then set lowest to middle + 1.
3. Return -1.

```
1  /*-
2   * ArraySearching
3   * Example showing the implementation of the linear search and the binary search.
4   * Author:  Don Spickler
5   * Date: 3/20/2011
6   */
7
8  public class ArraySearching {
9
10     public static void PrintIntArray(int A[]) {
11         for (int i = 0; i < A.length; i++) {
12             System.out.print(A[i] + "  ");
13         }
14         System.out.println();
15     }
16
17     /*-
18      * Searches the array A for the integer N. If N is not in the array, then -1
19      * is returned. If N is in the array, then the return value is the first
20      * integer i that satisfies A[i] == N.
21      */
22     static int linearSearch(int[] A, int N) {
23         for (int index = 0; index < A.length; index++) {
24             if (A[index] == N)
25                 return index; // N has been found at this index!
26         }
27         // If we get this far, then N has not been found
28         // anywhere in the array. Return a value of -1.
29         return -1;
30     }
31
32     /*-
33      * Searches the array A for the integer N. Precondition: A must be sorted
34      * into increasing order. Postcondition: If N is in the array, then the
35      * return value, i, satisfies A[i] == N. If N is not in the array, then the
36      * return value is -1.
37      */
38     static int binarySearch(int[] A, int N) {
39         int lowestPossibleLoc = 0;
40         int highestPossibleLoc = A.length - 1;
41         while (highestPossibleLoc >= lowestPossibleLoc) {
42             int middle = (lowestPossibleLoc + highestPossibleLoc) / 2;
43             if (A[middle] == N) {
44                 // N has been found at this index!
```

```
45                  return middle;
46              } else if (A[middle] > N) {
47                  // eliminate locations >= middle
48                  highestPossibleLoc = middle - 1;
49              } else {
50                  // eliminate locations <= middle
51                  lowestPossibleLoc = middle + 1;
52              }
53          }
54          // At this point, highestPossibleLoc < LowestPossibleLoc,
55          // which means that N is known to be not in the array. Return
56          // a -1 to indicate that N could not be found in the array.
57          return -1;
58      }
59
60      public static void main(String[] args) {
61          int arr1[] = { 3, 5, -2, 7, 10, 1, 5, 7, 3 };
62          PrintIntArray(arr1);
63
64          int pos = linearSearch(arr1, 1);
65          System.out.println(pos + "  " + arr1[pos]);
66          pos = linearSearch(arr1, 7);
67          System.out.println(pos + "  " + arr1[pos]);
68          pos = linearSearch(arr1, 9);
69          System.out.println(pos);
70          System.out.println();
71
72          int arr2[] = { 2, 4, 4, 4, 5, 6, 15, 18, 23, 42, 45, 57, 101 };
73          PrintIntArray(arr2);
74
75          pos = binarySearch(arr2, 15);
76          System.out.println(pos + "  " + arr2[pos]);
77          pos = binarySearch(arr2, 4);
78          System.out.println(pos + "  " + arr2[pos]);
79          pos = binarySearch(arr2, 100);
80          System.out.println(pos);
81          System.out.println();
82
83          // Doing a binary search on a non-sorted array will not
84          // work in general.
85          pos = binarySearch(arr1, 1);
86          System.out.println(pos);
87          System.out.println();
88      }
89  }
```

Program Output — ArraySearching.java

```
3  5  -2  7  10  1  5  7  3
5  1
3  7
-1

2  4  4  4  5  6  15  18  23  42  45  57  101
6  15
2  4
-1

-1
```

11.3 Sorting Algorithms Example

Bubble Sort Algorithm: The idea behind the bubble sort is to move through the list and compare each two consecutive entries in the array. That is, entries 0 and 1 then 1 and 2 then 2 and 3 and so on until the last two entries have been compared. At each comparison if the numbers are out of order then we interchange them. The result is that after one pass through the array the largest number in the array is in the last position of the array. In other words, the largest number in the array is now in its correct place in the sorted array. We now do the same process but we stop at the next to last position of the array (since the last position is already sorted). The second pass will put the second largest number in the next to last position and hence it is in its correct position. Make another pass up to the second to last position and so on. Once the last position sorted is the second position in the array we do the final comparison, and possible interchange, and we are done. In all we do the length of the array minus 1 passes.

1. For each entry index, i, from the next to last up to the second entry do the following
2. For each entry index, j, from the first up to $i - 1$ do the following
   (a) Compare the entry in the j^{th} position with the entry in the $j + 1^{st}$ position. If the two entries are out of order, interchange them.

Modified Bubble Sort Algorithm: If you examine the bubble sort above you probably notice that there are times when you are doing a process when you do not need to. If you have not noticed this, trace through the algorithm with an array that is already sorted. We can modify the bubble sort a little by tracking the last interchange we make. Notice that during each pass the array is sorted from the last interchange to the end. So instead of doing a pass for each entry we can do each pass up to the last interchange of the previous pass. This could conceivably save many passes through the array.

1. While a change in the order of any entries was done on the last pass do the following,
   (a) For each entry index, j, from the first up to the last changed position minus 1 do the following,
      i. Compare the entry in the j^{th} position with the entry in the $j + 1^{st}$ position. If the two entries are out of order, interchange them. Also, note that a change has been made and track the position of the last interchange for the next pass.

Insertion Sort Algorithm: The insertion sort works under the following principle, say we have an already sorted list and we want to insert another number into the list. One way to do this and retain a sorted list is to find the place the new number needs to go, move all of the larger numbers down one entry and put the new number in the empty spot. We can extend this idea to sorting an array as follows. Start with a single entry, entry index 0. An array with a single number in it is clearly sorted. Now take the second number in the array and insert it into the single entry array by moving the larger numbers down and inserting the new entry in order. Now we have a sorted array with two elements. Now take the third entry of the array and insert it by moving the larger entries down and inserting the new entry. Do the same with the next entry and so on until we process the entire array.

1. For each entry index, itemsSorted, from one to the length of the array minus 1 do the following,
   (a) Store the entry at the itemsSorted position of the array. Also store the location loc where we will store this number. Start the loc at the position of itemsSorted - 1.
   (b) While the entry in the array at position loc is larger than the one in the itemsSorted position move the loc position entry down one and decrement loc.
   (c) Place the new entry in location loc + 1.

Selection Sort Algorithm: The idea behind the selection sort is to find the largest element in the array and interchange it with the entry in the last position. Then find the largest entry in the array up to the next to last position and interchange it with the entry in the next to last position. Repeat the process up to position length - 2, and so on.

1. For each entry index, lastPlace, from the length minus one to index 1 do the following,
   (a) Find the largest value in the array from position 0 up to position lastPlace.
   (b) Interchange the maximum with the entry in lastPlace.

```
1  import java.util.Random;
2  import java.util.Scanner;
3
4  /*-
5   * ArraySorting
6   * Example showing the implementation of the Bubble Sort, Modified Bubble Sort,
7   * Insertion Sort, and the Selection Sort.
8   * Author:  Don Spickler
9   * Date: 3/20/2011
10  */
```

```
11
12 public class ArraySorting {
13
14     public static int[] CopyintArray(int A[]) {
15         int[] B = new int[A.length];
16
17         for (int i = 0; i < A.length; i++)
18             B[i] = A[i];
19
20         return B;
21     }
22
23     public static void PrintIntArray(int A[]) {
24         for (int i = 0; i < A.length; i++) {
25             System.out.print(A[i] + "  ");
26         }
27         System.out.println();
28     }
29
30     public static void BubbleSort(int A[]) {
31         for (int i = A.length - 1; i > 0; i--) {
32             for (int j = 0; j < i; j++) {
33                 if (A[j] > A[j + 1]) {
34                     int temp = A[j];
35                     A[j] = A[j + 1];
36                     A[j + 1] = temp;
37                 }
38             }
39         }
40     }
41
42     public static void ModifiedBubbleSort(int A[]) {
43         int lastchange = A.length - 1;
44         boolean changemade = true;
45         int pass = 1;
46
47         while (changemade) {
48             changemade = false;
49             int changed = 0;
50             int j = 0;
51             while (j < lastchange) {
52                 if (A[j] > A[j + 1]) {
53                     int temp = A[j];
54                     A[j] = A[j + 1];
55                     A[j + 1] = temp;
56                     changed = j;
57                     changemade = true;
58                 }
59                 j++;
60             }
61             lastchange = changed;
62         }
63     }
64
65     public static void insertionSort(int[] A) {
66         for (int itemsSorted = 1; itemsSorted < A.length; itemsSorted++) {
67             int temp = A[itemsSorted];
68             int loc = itemsSorted - 1;
69             while (loc >= 0 && A[loc] > temp) {
70                 A[loc + 1] = A[loc];
71                 loc = loc - 1;
72             }
73             A[loc + 1] = temp;
74         }
```

```
75      }
76
77      public static void selectionSort(int[] A) {
78          for (int lastPlace = A.length - 1; lastPlace > 0; lastPlace--) {
79              int maxLoc = 0;
80              for (int j = 1; j <= lastPlace; j++)
81                  if (A[j] > A[maxLoc])
82                      maxLoc = j;
83
84              int temp = A[maxLoc];
85              A[maxLoc] = A[lastPlace];
86              A[lastPlace] = temp;
87          }
88      }
89
90      public static void main(String[] args) {
91          Scanner keyboard = new Scanner(System.in);
92          Random generator = new Random();
93          System.out.print("Input the array size: ");
94          int arraySize = keyboard.nextInt();
95          System.out.print("Input the maximum random integer: ");
96          int intSize = keyboard.nextInt();
97          System.out.println();
98
99          int arr[] = new int[arraySize];
100         int arr2[];
101
102         for (int i = 0; i < arr.length; i++)
103             arr[i] = generator.nextInt(intSize) + 1;
104
105         arr2 = CopyintArray(arr);
106         PrintIntArray(arr2);
107         BubbleSort(arr2);
108         PrintIntArray(arr2);
109         System.out.println();
110
111         arr2 = CopyintArray(arr);
112         PrintIntArray(arr2);
113         ModifiedBubbleSort(arr2);
114         PrintIntArray(arr2);
115         System.out.println();
116
117         arr2 = CopyintArray(arr);
118         PrintIntArray(arr2);
119         insertionSort(arr2);
120         PrintIntArray(arr2);
121         System.out.println();
122
123         arr2 = CopyintArray(arr);
124         PrintIntArray(arr2);
125         selectionSort(arr2);
126         PrintIntArray(arr2);
127         System.out.println();
128     }
129 }
```

Program Output — ArraySorting.java

```
Input the array size: 20
Input the maximum random integer: 1000

215  226  390  449  933  471  102  686  570  295  57  378  686  847  472  869  18  65  202
      523
```

```
18  57  65  102  202  215  226  295  378  390  449  471  472  523  570  686  686  847  869
      933

215  226  390  449  933  471  102  686  570  295  57  378  686  847  472  869  18  65  202
      523
18  57  65  102  202  215  226  295  378  390  449  471  472  523  570  686  686  847  869
      933

215  226  390  449  933  471  102  686  570  295  57  378  686  847  472  869  18  65  202
      523
18  57  65  102  202  215  226  295  378  390  449  471  472  523  570  686  686  847  869
      933

215  226  390  449  933  471  102  686  570  295  57  378  686  847  472  869  18  65  202
      523
18  57  65  102  202  215  226  295  378  390  449  471  472  523  570  686  686  847  869
      933
```

Chapter 12

Array Lists

12.1 Introduction

Another array-type object in Java is the ArrayList. The ArrayList works a lot like an array but has some very nice features that the array does not have. For example, the ArrayList will automatically resize itself if needed. With an array, if you use all of the sell locations then you are done, or you need to create a bigger array and copy all of the elements over from the smaller array to the larger array. This can be done but it is a bit of a hassle. Although the array and the ArrayList are very similar in style, their syntax is quite different.

12.2 Basic ArrayList Manipulation Example

To create an array list we start off with the data type `ArrayList`, followed by the type of data to be stored in the ArrayList in angled brackets, then the variable name, assignment to a new `ArrayList` of the same type, and finally empty parentheses. For example, to create an ArrayList of integers, named `A`, we use the syntax.

```
ArrayList<Integer> A = new ArrayList<Integer>();
```

Note the use of `Integer` and not `int`. With ArrayLists the data type must be class structure and not a native data type. So for the native data types we use the *wrapper classes* for those types. So to make an array list of ints we use Integer. The wrapper classes for the native numeric types are,

Native Type	Wrapper Class
boolean	Boolean
char	Character

Native Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Note that the String is already a class structure so it does not need a wrapper class. We can create an ArrayList out of any class type, so we can create an ArrayList of Scanners, Strings, Integers, Employees, Cards, Decks, Hands, and even ArrayLists.

Once the ArrayList is created we can add items to the list with the `add` method. For example,

```
A.add(3);
A.add(17);
A.add(32);
A.add(-5);
A.add(8);
A.add(2120);
```

adds the integers 3, 17, 32, -5, 8, and 2120 to the list in that order. As with arrays, the ArrayList uses an index to reference the items in the list and this index starts at 0, as it does for arrays. So in our example, 3 is at index 0, 17 at index 1, and so on. We can retrieve a data item from the list by using the `get` method with the index of the item. So `A.get(1)` will retrieve the item at index 1, that is, the 17.

Note that we constructed a method for printing out an ArrayList that gets each element, one by one, and sends it to the print statement. We can also print out an ArrayList by simply putting the ArrayList into a print or println statement, like `System.out.println(A)`.

The ArrayList has its own searching methods. The statement `A.indexOf(32)` will find the first occurrence of 32 in the list and return the index to that occurrence. If the value being searched for is not in the list, the `indexOf` method will return -1, just like our searching algorithms we implemented on arrays. You can find the last occurrence by using the `lastIndexOf` method in place of the `indexOf` method.

You can also sort an ArrayList in a single command, using the `Collections` class. The statement, `Collections.sort(A)` will sort the ArrayList `A`. Sorting ArrayLists that contain numeric or string data types is just this simple. To sort a user-defined data type, like our Employee or Card requires a little more work. Specifically, we need to implement the Comparable interface. We will look at an example of this later on. Note that the `Collections.sort(A)` will sort the ArrayList in

ascending order, smallest to largest. We can reverse the sort to get largest to smallest by the command, `Collections.sort(A, Collections.reverseOrder())`.

To remove an item at a specified index we use the `remove` method. The statement `A.remove(3)` will remove the item at index 3 and it will move the later items forward in the list. So index 3 will now contain the item that was in index 4, and so on. We can also use the `remove` method to remove a specific item. So if the parameter in this call is an object, the program will search for that object and remove it if it is in the list. So the statement `A.remove((Integer) 8)` will find the item 8 in the list and if it is there it will remove it. Note that this removes the item 8 and not the element at index 8. To remove the item at index 8 we would use, `A.remove(8)`.

There is one more command in this example, and that is the clone method. Remember with arrays we do not use the assignment statement to make a copy of the array. We create another array of the same size and copy the array contents one by one to the new array. The same has to be done with an ArrayList, but this is built into the ArrayList class, it is cloning. So the statement,

```
ArrayList<Integer> D = (ArrayList<Integer>) A.clone();
```

will create a clone of the ArrayList `A` and assign it to the ArrayList `D`. Now `A` and `D` are two different ArrayLists with the same contents.

If we take a closer look at our printing method, notice that the parameter is

```
ArrayList Arr
```

That is, there is no type associated with the ArrayList, so the method will take in an ArrayList of any type. This is why it worked when printing out integers, doubles, and strings.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3
4  /*-
5   * ArrayList001
6   * Example showing the basic use of an ArrayList.
7   * Author:  Don Spickler
8   * Date: 3/20/2011
9   */
10
11 public class ArrayList001 {
12
13     public static void printArrayList(ArrayList A) {
14         for (int i = 0; i < A.size(); i++)
15             System.out.print(A.get(i) + "  ");
16
17         System.out.println();
18     }
19
20     public static void main(String[] args) {
21         ArrayList<Integer> A = new ArrayList<Integer>();
22
23         A.add(3);
24         A.add(17);
```

```
25          A.add(32);
26          A.add(-5);
27          A.add(8);
28          A.add(2120);
29
30          System.out.println(A.size());
31          System.out.println();
32
33          printArrayList(A);
34
35          System.out.println();
36          System.out.println(A.get(2) + A.get(1) * A.get(4));
37          System.out.println(A.indexOf(32));
38
39          Collections.sort(A);
40          printArrayList(A);
41
42          System.out.println(A.indexOf(32));
43          System.out.println(A.indexOf(1001));
44
45          A.remove(3);
46          printArrayList(A);
47          A.remove((Integer) 8);
48          printArrayList(A);
49
50          A.add(23);
51          A.add(7);
52          A.add(2);
53          A.add(-15);
54          A.add(82);
55          A.add(21);
56
57          printArrayList(A);
58          Collections.sort(A);
59          printArrayList(A);
60
61          System.out.println(A);
62
63          System.out.println();
64          System.out.println("---------------------------");
65          System.out.println();
66
67          ArrayList<String> B = new ArrayList<String>();
68
69          B.add("John");
70          B.add("Sue");
71          B.add("Kim");
72          B.add("Sam");
73          B.add("Mike");
74          B.add("Don");
75          B.add("Dan");
76          B.add("Jack");
77          B.add("Jane");
78
79          printArrayList(B);
80          Collections.sort(B);
81          printArrayList(B);
82          B.remove("Kim");
83          printArrayList(B);
84          B.remove(3);
85          printArrayList(B);
86
87          System.out.println(B.indexOf("John"));
88          System.out.println(B.indexOf("Simon"));
```

```
89
90          System.out.println(B);
91
92          System.out.println();
93          System.out.println("---------------------------");
94          System.out.println();
95
96          ArrayList<Double> C = new ArrayList<Double>();
97
98          for (int i = 0; i < 5; i++)
99              C.add(Math.random());
100
101         printArrayList(C);
102         Collections.sort(C);
103         printArrayList(C);
104
105         System.out.println(C);
106
107         System.out.println();
108         System.out.println("---------------------------");
109         System.out.println();
110
111         Collections.sort(A);
112         printArrayList(A);
113         Collections.sort(A, Collections.reverseOrder());
114         printArrayList(A);
115
116         System.out.println();
117         System.out.println("---------------------------");
118         System.out.println();
119
120         // clone copies an object, the cast is needed to "convert" the data to
121         // the ArrayList.
122         ArrayList<Integer> D = (ArrayList<Integer>) A.clone();
123
124         System.out.println(A);
125         System.out.println(D);
126         Collections.sort(D);
127         System.out.println(A);
128         System.out.println(D);
129         D.remove(2);
130         D.remove(5);
131         System.out.println(A);
132         System.out.println(D);
133     }
134 }
```

Program Output — ArrayList001.java

```
6

3  17  32  -5  8  2120

168
2
-5  3  8  17  32  2120
4
-1
-5  3  8  32  2120
-5  3  32  2120
-5  3  32  2120  23  7  2  -15  82  21
-15  -5  2  3  7  21  23  32  82  2120
[-15, -5, 2, 3, 7, 21, 23, 32, 82, 2120]
```

```
---------------------------

John  Sue  Kim  Sam  Mike  Don  Dan  Jack  Jane
Dan  Don  Jack  Jane  John  Kim  Mike  Sam  Sue
Dan  Don  Jack  Jane  John  Mike  Sam  Sue
Dan  Don  Jack  John  Mike  Sam  Sue
3
-1
[Dan, Don, Jack, John, Mike, Sam, Sue]

---------------------------

0.9575829181659298  0.9448645608091782  0.3133759640147388  0.7114314558717817
    0.4304279228994208
0.3133759640147388  0.4304279228994208  0.7114314558717817  0.9448645608091782
    0.9575829181659298
[0.3133759640147388, 0.4304279228994208, 0.7114314558717817, 0.9448645608091782,
    0.9575829181659298]

---------------------------

-15  -5  2  3  7  21  23  32  82  2120
2120  82  32  23  21  7  3  2  -5  -15

---------------------------

[2120, 82, 32, 23, 21, 7, 3, 2, -5, -15]
[2120, 82, 32, 23, 21, 7, 3, 2, -5, -15]
[2120, 82, 32, 23, 21, 7, 3, 2, -5, -15]
[-15, -5, 2, 3, 7, 21, 23, 32, 82, 2120]
[2120, 82, 32, 23, 21, 7, 3, 2, -5, -15]
[-15, -5, 3, 7, 21, 32, 82, 2120]
```

12.3 Basic ArrayList Manipulation Example #2

This example shows a little more with passing ArrayLists to methods. Notice that the call simply uses the ArrayList name, just like the array. Pay close attention to the parameter list in the method headers in this example. The parameter,

`ArrayList<Integer> Arr`

will expect an ArrayList of Integers but the parameter,

`ArrayList Arr`

will take an ArrayList of any type.

```
1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 /*-
5  * ArrayList002
6  * Another example showing the basic use of an ArrayList.
7  * Author:  Don Spickler
8  * Date: 3/20/2011
9  */
```

```
10
11 public class ArrayList002 {
12
13     public static void PopulateArray(ArrayList<Integer> A, int numelts) {
14         Scanner keyboard = new Scanner(System.in);
15         for (int i = 0; i < numelts; i++) {
16             System.out.print("Input entry " + (i + 1) + ": ");
17             A.add(keyboard.nextInt());
18         }
19     }
20
21     public static void PrintArray(ArrayList Arr) {
22         for (int i = 0; i < Arr.size(); i++) {
23             System.out.print(Arr.get(i) + "  ");
24         }
25         System.out.println();
26     }
27
28     public static int SumArray(ArrayList<Integer> Arr) {
29         int sum = 0;
30         for (int i = 0; i < Arr.size(); i++) {
31             sum += Arr.get(i);
32         }
33         return sum;
34     }
35
36     public static void main(String[] args) {
37         Scanner keyboard = new Scanner(System.in);
38         System.out.print("Input the array size: ");
39         int arraySize = keyboard.nextInt();
40
41         ArrayList<Integer> intArray = new ArrayList<Integer>();
42
43         PopulateArray(intArray, arraySize);
44         PrintArray(intArray);
45         System.out.println("The sum of the array is = " + SumArray(intArray));
46     }
47 }
```

Program Output **ArrayList002.java**

```
Input the array size: 5
Input entry 1: 2
Input entry 2: 6
Input entry 3: 8
Input entry 4: 4
Input entry 5: 1
2  6  8  4  1
The sum of the array is = 21
```

12.4 ArrayList with User-Defined Types Example

In this example we create an ArrayList of Employee types.

```
1 /*-
2  * Employee
3  * Employee class stores information about a company employee, name,
4  * wage, and hours worked for the week.
5  * Author:  Don Spickler
```

```
 * Date: 3/20/2011
 */

public class Employee {
    private String first;
    private String last;
    private double wage;
    private double hours_worked;

    public Employee(String fn, String ln, double w, double hw) {
        first = fn;
        last = ln;

        if (w > 0)
            wage = w;
        else
            wage = 0;

        if (hw > 0)
            hours_worked = hw;
        else
            hours_worked = 0;
    }

    public String getName() {
        return first + " " + last;
    }

    public String getFormalName() {
        return last + ", " + first;
    }

    public double getWage() {
        return wage;
    }

    public double getHoursWorked() {
        return hours_worked;
    }

    public void setFirstName(String n) {
        first = n;
    }

    public void setLastName(String n) {
        last = n;
    }

    public void setWage(double w) {
        if (w > 0)
            wage = w;
        else
            wage = 0;
    }

    public void getHoursWorked(double hw) {
        if (hw > 0)
            hours_worked = hw;
        else
            hours_worked = 0;
    }

    public double pay() {
        double payment = 0;
```

```
70          if (hours_worked > 40)
71              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
72          else
73              payment = wage * hours_worked;
74
75          return payment;
76      }
77  }
```

```
1   import java.util.ArrayList;
2   import java.util.Scanner;
3
4   /*-
5    * ArrayList003
6    * Example showing the use of an ArrayList with user-defined data type entries.
7    * Author:  Don Spickler
8    * Date: 3/20/2011
9    */
10
11  public class ArrayList003 {
12
13      public static void InputEmployees(ArrayList<Employee> A, int sz) {
14          Scanner keyboard = new Scanner(System.in);
15
16          for (int i = 0; i < sz; i++) {
17              System.out.print("Employee " + (i + 1) + " first name: ");
18              String fname = keyboard.nextLine();
19              System.out.print("Employee " + (i + 1) + " last name: ");
20              String lname = keyboard.nextLine();
21              System.out.print("Employee " + (i + 1) + " wage: ");
22              double wage = keyboard.nextDouble();
23              System.out.print("Employee " + (i + 1) + " hours worked: ");
24              double hours = keyboard.nextDouble();
25
26              A.add(new Employee(fname, lname, wage, hours));
27
28              System.out.println(); // Put space between inputs.
29              String clear = keyboard.nextLine(); // clear the end of line.
30          }
31      }
32
33      public static void PrintPayReport(ArrayList<Employee> A) {
34          for (int i = 0; i < A.size(); i++) {
35              PrintRecord(A.get(i));
36          }
37      }
38
39      public static double CalculateTotalPay(ArrayList<Employee> A) {
40          double totalpay = 0;
41          for (int i = 0; i < A.size(); i++) {
42              totalpay += A.get(i).pay();
43          }
44          return totalpay;
45      }
46
47      public static void PrintRecord(Employee employee) {
48          System.out.println("Name: " + employee.getFormalName());
49          System.out.println("Wage: " + employee.getWage());
50          System.out.println("Hours Worked: " + employee.getHoursWorked());
51          System.out.printf("Pay: %.2f \n", employee.pay());
52          System.out.println();
53      }
54
55      public static void main(String[] args) {
```

```
56          Scanner keyboard = new Scanner(System.in);
57          System.out.print("Input the number of employees: ");
58          int arraySize = keyboard.nextInt();
59
60          ArrayList<Employee> company = new ArrayList<Employee>();
61
62          InputEmployees(company, arraySize);
63          PrintPayReport(company);
64          double totpay = CalculateTotalPay(company);
65          System.out.printf("Company payout = %.2f \n", totpay);
66      }
67  }
```

Program Output **ArrayList003.java**

```
Input the number of employees: 3
Employee 1 first name: Don
Employee 1 last name: Spickler
Employee 1 wage: 12.50
Employee 1 hours worked: 48

Employee 2 first name: Jane
Employee 2 last name: Doe
Employee 2 wage: 15.63
Employee 2 hours worked: 37

Employee 3 first name: Jack
Employee 3 last name: Frost
Employee 3 wage: 14.25
Employee 3 hours worked: 28

Name: Spickler, Don
Wage: 12.5
Hours Worked: 48.0
Pay: 650.00

Name: Doe, Jane
Wage: 15.63
Hours Worked: 37.0
Pay: 578.31

Name: Frost, Jack
Wage: 14.25
Hours Worked: 28.0
Pay: 399.00

Company payout = 1627.31
```

12.5 ArrayList and the Collections Class Example

This example is a slight update of the previous example. Notice two additions to the Employee class. We added two methods, `compareTo` and `toString`.

The `compareTo` method was created so that we could use the `Collections` class to sort the ArrayList of Employees. When you construct a `compareTo` method it must return an int so that the integer is less than zero if the caller object is

smaller, zero if they are equal and greater than one if the parameter object is smaller. For example, if we have two employees `e1` and `e2`, then if `e1.compareTo(e2)` is negative then `e1` is less than `e2`. If `e1.compareTo(e2)` is zero then `e1` is equal to `e2`. Finally, if `e1.compareTo(e2)` is positive then `e1` is greater than `e2`. Looking at our definition,

```
this.getFormalName().toUpperCase().compareTo(((Employee) e2)
    .getFormalName().toUpperCase())
```

The reserved word `this` represents `e1`, so we are constructing the formal names of both `e1` and `e2`, converting them to uppercase and then using the string `compareTo` method to compare them. In all, it is doing alphabetical order. Note that we also needed to put `implements Comparable` after the class name at the top. `Comparable` is something called an interface, we will not discuss what interfaces are in this set of notes.

The other method we added was `toString`. This was done so that we could use a print statement on an Employee type. In the main we changed the `PrintPayReport` method from calling `PrintRecord` to `System.out.println(A.get(i))`. If a print or println is called with an object (like `A.get(i)` which is an Employee) the system calls the `toString` method to get a string representation of the object and then simply prints the string to the console screen.

```
1  /*-
2   * Employee
3   * Employee class stores information about a company employee, name,
4   * wage, and hours worked for the week.
5   * Author:  Don Spickler
6   * Date: 3/20/2011
7   */
8
9  public class Employee implements Comparable {
10     private String first;
11     private String last;
12     private double wage;
13     private double hours_worked;
14
15     public Employee(String fn, String ln, double w, double hw) {
16         first = fn;
17         last = ln;
18
19         if (w > 0)
20             wage = w;
21         else
22             wage = 0;
23
24         if (hw > 0)
25             hours_worked = hw;
26         else
27             hours_worked = 0;
28     }
29
30     public String getName() {
31         return first + " " + last;
32     }
33
```

```
34      public String getFormalName() {
35          return last + ", " + first;
36      }
37
38      public double getWage() {
39          return wage;
40      }
41
42      public double getHoursWorked() {
43          return hours_worked;
44      }
45
46      public void setFirstName(String n) {
47          first = n;
48      }
49
50      public void setLastName(String n) {
51          last = n;
52      }
53
54      public void setWage(double w) {
55          if (w > 0)
56              wage = w;
57          else
58              wage = 0;
59      }
60
61      public void getHoursWorked(double hw) {
62          if (hw > 0)
63              hours_worked = hw;
64          else
65              hours_worked = 0;
66      }
67
68      public double pay() {
69          double payment = 0;
70          if (hours_worked > 40)
71              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
72          else
73              payment = wage * hours_worked;
74
75          return payment;
76      }
77
78      public String toString() {
79          String retstr = "";
80          retstr += "Name: " + getFormalName() + "\n";
81          retstr += "Wage: " + getWage() + "\n";
82          retstr += "Hours Worked: " + getHoursWorked() + "\n";
83          retstr += "Pay: " + (Math.round(pay() * 100.0) / 100.0) + "\n";
84
85          return retstr;
86      }
87
88      public int compareTo(Object e2) {
89          return this.getFormalName().toUpperCase().compareTo(((Employee) e2).getFormalName
                ().toUpperCase());
90      }
91  }
```

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Scanner;
4
```

```
5  /*-
6   * ArrayList004
7   * Example showing the use of an ArrayList with user-defined data type entries.
8   * This program adds in the sorting of the ArrayList using the sort method in
9   * the Collections class in Java.  Note the addition of the compareTo method
10  * in the Employee class.
11  * Author:  Don Spickler
12  * Date: 3/20/2011
13  */
14
15 public class ArrayList004 {
16
17     public static void InputEmployees(ArrayList<Employee> A, int sz) {
18         Scanner keyboard = new Scanner(System.in);
19
20         for (int i = 0; i < sz; i++) {
21             System.out.print("Employee " + (i + 1) + " first name: ");
22             String fname = keyboard.nextLine();
23             System.out.print("Employee " + (i + 1) + " last name: ");
24             String lname = keyboard.nextLine();
25             System.out.print("Employee " + (i + 1) + " wage: ");
26             double wage = keyboard.nextDouble();
27             System.out.print("Employee " + (i + 1) + " hours worked: ");
28             double hours = keyboard.nextDouble();
29
30             A.add(new Employee(fname, lname, wage, hours));
31
32             System.out.println(); // Put space between inputs.
33             String clear = keyboard.nextLine(); // clear the end of line.
34         }
35     }
36
37     public static void PrintPayReport(ArrayList A) {
38         for (int i = 0; i < A.size(); i++) {
39             System.out.println(A.get(i));
40         }
41     }
42
43     public static double CalculateTotalPay(ArrayList<Employee> A) {
44         double totalpay = 0;
45         for (int i = 0; i < A.size(); i++) {
46             totalpay += A.get(i).pay();
47         }
48         return totalpay;
49     }
50
51     public static void main(String[] args) {
52         Scanner keyboard = new Scanner(System.in);
53         System.out.print("Input the number of employees: ");
54         int arraySize = keyboard.nextInt();
55
56         ArrayList<Employee> company = new ArrayList<Employee>();
57         InputEmployees(company, arraySize);
58
59         Collections.sort(company);
60
61         PrintPayReport(company);
62         System.out.printf("Company payout = %.2f \n", CalculateTotalPay(company));
63     }
64 }
```

Program Output **ArrayList004.java**

```
Input the number of employees: 3
Employee 1 first name: Don
Employee 1 last name: Spickler
Employee 1 wage: 12.50
Employee 1 hours worked: 48

Employee 2 first name: Jane
Employee 2 last name: Doe
Employee 2 wage: 15.63
Employee 2 hours worked: 37

Employee 3 first name: Jack
Employee 3 last name: Frost
Employee 3 wage: 14.25
Employee 3 hours worked: 28

Name: Doe, Jane
Wage: 15.63
Hours Worked: 37.0
Pay: 578.31

Name: Frost, Jack
Wage: 14.25
Hours Worked: 28.0
Pay: 399.0

Name: Spickler, Don
Wage: 12.5
Hours Worked: 48.0
Pay: 650.0

Company payout = 1627.31
```

12.6 Basic Statistics Calculations Example

This example generates random integers and loads them into an ArrayList of integers, then does some basic statistics on the list of numbers.

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.Scanner;
4
5  /*-
6   * ArrayList005
7   * Example showing the calculation of some basic statistics using
8   * the entries in the ArrayList.
9   * Author:  Don Spickler
10  * Date: 3/20/2011
11  */
12
13 public class ArrayList005 {
14
15     public static void PopulateArray(ArrayList<Integer> A, int sz, int max) {
16         for (int i = 0; i < sz; i++) {
17             A.add((int) (Math.random() * max) + 1);
18         }
19     }
20
21     public static void PrintArray(ArrayList<Integer> Arr, int width) {
```

```
22          String format = "%" + width + "d";
23          for (int i = 0; i < Arr.size(); i++) {
24              System.out.printf(format, Arr.get(i));
25          }
26          System.out.println();
27      }
28
29      public static void PrintArrayBarChart(ArrayList<Integer> Arr) {
30          for (int i = 0; i < Arr.size(); i++) {
31              for (int j = 0; j < Arr.get(i); j++)
32                  System.out.print("*");
33
34              System.out.println();
35          }
36      }
37
38      public static int SumArray(ArrayList<Integer> Arr) {
39          int sum = 0;
40          for (int i = 0; i < Arr.size(); i++) {
41              sum += Arr.get(i);
42          }
43          return sum;
44      }
45
46      public static double AvgArray(ArrayList<Integer> Arr) {
47          int sum = SumArray(Arr);
48          if (Arr.size() > 0)
49              return 1.0 * sum / Arr.size();
50          else
51              return 0;
52      }
53
54      public static double VarianceArray(ArrayList<Integer> Arr) {
55          if (Arr.size() >= 2) {
56              double avg = AvgArray(Arr);
57
58              double sum = 0;
59              for (int i = 0; i < Arr.size(); i++) {
60                  sum += (Arr.get(i) - avg) * (Arr.get(i) - avg);
61              }
62              return sum / (Arr.size() - 1);
63          } else
64              return 0;
65      }
66
67      public static double StandardDeviationArray(ArrayList<Integer> Arr) {
68          if (Arr.size() >= 2)
69              return Math.sqrt(VarianceArray(Arr));
70          else
71              return 0;
72      }
73
74      public static int MaxArray(ArrayList<Integer> Arr) {
75          if (Arr.size() > 0) {
76              int max = Arr.get(0);
77              for (int i = 0; i < Arr.size(); i++) {
78                  if (Arr.get(i) > max)
79                      max = Arr.get(i);
80              }
81              return max;
82          } else
83              return 0;
84      }
85
```

```
86      public static int MinArray(ArrayList<Integer> Arr) {
87          if (Arr.size() > 0) {
88              int min = Arr.get(0);
89              for (int i = 0; i < Arr.size(); i++) {
90                  if (Arr.get(i) < min)
91                      min = Arr.get(i);
92              }
93              return min;
94          } else
95              return 0;
96      }
97
98      public static int CountLessArray(ArrayList<Integer> Arr, int n) {
99          int count = 0;
100         for (int i = 0; i < Arr.size(); i++) {
101             if (Arr.get(i) < n)
102                 count++;
103         }
104         return count;
105     }
106
107     public static int CountGreaterArray(ArrayList<Integer> Arr, int n) {
108         int count = 0;
109         for (int i = 0; i < Arr.size(); i++) {
110             if (Arr.get(i) > n)
111                 count++;
112         }
113         return count;
114     }
115
116     public static void main(String[] args) {
117         Scanner keyboard = new Scanner(System.in);
118         System.out.print("Input the array size: ");
119         int arraySize = keyboard.nextInt();
120         System.out.print("Input max entry size: ");
121         int entrysize = keyboard.nextInt();
122         System.out.print("Input the count division: ");
123         int countdiv = keyboard.nextInt();
124
125         ArrayList<Integer> intArray = new ArrayList<Integer>();
126         PopulateArray(intArray, arraySize, entrysize);
127         PrintArray(intArray, 4);
128
129         System.out.println("The sum of the array is = " + SumArray(intArray));
130         System.out.println("The average of the array is = " + AvgArray(intArray));
131         System.out.println("The maximum of the array is = " + MaxArray(intArray));
132         System.out.println("The minimum of the array is = " + MinArray(intArray));
133         System.out.println(
134                 "The number less than " + countdiv + " in the array is = " +
                        CountLessArray(intArray, countdiv));
135         System.out.println(
136                 "The number greater than " + countdiv + " in the array is = " +
                        CountGreaterArray(intArray, countdiv));
137         System.out.println("The variance of the array is = " + VarianceArray(intArray));
138         System.out.println("The standard deviation of the array is = " +
                StandardDeviationArray(intArray));
139
140         System.out.println();
141         PrintArrayBarChart(intArray);
142         System.out.println();
143
144         // Copy intArray to B.
145         ArrayList<Integer> B = (ArrayList<Integer>) intArray.clone();
146
```

```
147         Collections.reverse(B);
148         PrintArray(intArray, 4);
149         PrintArray(B, 4);
150     }
151 }
```

Program Output **ArrayList005.java**

```
Input the array size: 25
Input max entry size: 50
Input the count division: 5
  10  13  28   2  26  48  44  13   6   7  19  17  39  41  13  15  20  45  21   6  48  28
      28  26  23
The sum of the array is = 586
The average of the array is = 23.44
The maximum of the array is = 48
The minimum of the array is = 2
The number less than 5 in the array is = 1
The number greater than 5 in the array is = 24
The variance of the array is = 196.50666666666666
The standard deviation of the array is = 14.018083558984326

**********
**********
**************************
**
************************
**********************************************
*****************************************
**********
******
*******
******************
*****************
************************************
****************************************
**********
**************
******************
******************************************
**********************
******
**********************************************
**************************
**************************
************************
**********************

  10  13  28   2  26  48  44  13   6   7  19  17  39  41  13  15  20  45  21   6  48  28
      28  26  23
  23  26  28  28  48   6  21  45  20  15  13  41  39  17  19   7   6  13  44  48  26   2
      28  13  10
```

12.7 Untyped ArrayList Example

In the previous examples most of our ArrayLists had a type, like Integer, Double, String, or Employee. On the other had, our print method from the first examples

did not have a type, the parameter was simply `ArrayList A`. When we do not give an ArrayList a type it is called *untyped*. An untyped ArrayList really has the type Object. In Java, everything is an Object, so for `ArrayList A`, `A` is an ArrayList of Objects. This can come in handy but you have to be careful when working with them. The neat thing is that this ArrayList `A` can store anything in any cell. So your first entry could be an Integer, your second entry could be a String, your third entry could be an Employee, and your fourth entry could be a Card.

What you need to be careful about is that you need to cast these objects to their type before you use them. So even if `A.get(1)` is a double you need to use `(Double) A.get(1)` before you can use it as a double. Trace through this program very carefully to see how all of the output was generated.

```
1  import java.util.ArrayList;
2
3  /*-
4   * ArrayList006
5   * Example showing the use of an untyped ArrayList.
6   * Author:  Don Spickler
7   * Date: 3/20/2011
8   */
9
10 public class ArrayList006 {
11
12     public static void printList(ArrayList A) {
13         for (int i = 0; i < A.size(); i++) {
14             System.out.println(A.get(i));
15         }
16     }
17
18     public static double sumList(ArrayList A) {
19         double sum = 0;
20         for (int i = 0; i < A.size(); i++) {
21             try {
22                 sum += (Double) A.get(i);
23             } catch (Exception e) {
24             }
25         }
26         return sum;
27     }
28
29     public static double sumList2(ArrayList A) {
30         double sum = 0;
31         for (int i = 0; i < A.size(); i++) {
32             try {
33                 String entstr = "" + A.get(i);
34                 sum += Double.parseDouble(entstr);
35             } catch (Exception e) {
36             }
37         }
38
39         return sum;
40     }
41
42     public static void main(String[] args) {
43         ArrayList genList = new ArrayList();
44
45         genList.add("This is a string.");
46         genList.add(3.14159);
47         genList.add(27);
```

```
48          genList.add('A');
49          genList.add(-4);
50          genList.add("Another String");
51          genList.add(17);
52          genList.add(Math.E);
53
54          printList(genList);
55          System.out.println();
56
57          double ans = (Double) genList.get(1) + (Integer) genList.get(2);
58          System.out.println(ans);
59
60          System.out.println();
61          String ans2 = (String) genList.get(0) + (Integer) genList.get(2);
62          System.out.println(ans2);
63
64          // String ans3 = (String) genList.get(1) + (String) genList.get(2);
65
66          System.out.println();
67          System.out.println(sumList(genList));
68
69          System.out.println();
70          System.out.println(sumList2(genList));
71
72          System.out.println(genList);
73          genList.remove(5);
74          System.out.println(genList);
75
76          genList.remove("This");
77          System.out.println(genList);
78
79          genList.remove("This is a string.");
80          System.out.println(genList);
81
82          genList.add(17.6);
83          genList.add(Math.PI);
84          genList.add(17.6);
85          genList.add("Java is cool.");
86          genList.add(-55);
87          genList.add("The last entry.");
88          System.out.println(genList);
89
90          System.out.println();
91          System.out.println(sumList(genList));
92
93          System.out.println();
94          System.out.println(sumList2(genList));
95
96          ArrayList list2 = new ArrayList();
97
98          list2 = (ArrayList) genList.clone();
99
100         System.out.println();
101         System.out.println(genList);
102         System.out.println(list2);
103
104         list2.remove(Math.PI);
105         list2.remove("Java is cool.");
106         list2.remove(17.6);
107         list2.remove((Integer) (-4));
108
109         System.out.println();
110         System.out.println(genList);
111         System.out.println(list2);
```

```
112     }
113 }
```

Program Output — ArrayList006.java

```
This is a string.
3.14159
27
A
-4
Another String
17
2.718281828459045

30.14159

This is a string.27

5.859871828459045

45.85987182845905
[This is a string., 3.14159, 27, A, -4, Another String, 17, 2.718281828459045]
[This is a string., 3.14159, 27, A, -4, 17, 2.718281828459045]
[This is a string., 3.14159, 27, A, -4, 17, 2.718281828459045]
[3.14159, 27, A, -4, 17, 2.718281828459045]
[3.14159, 27, A, -4, 17, 2.718281828459045, 17.6, 3.141592653589793, 17.6, Java is cool.,
    -55, The last entry.]

44.20146448204884

29.201464482048834

[3.14159, 27, A, -4, 17, 2.718281828459045, 17.6, 3.141592653589793, 17.6, Java is cool.,
    -55, The last entry.]
[3.14159, 27, A, -4, 17, 2.718281828459045, 17.6, 3.141592653589793, 17.6, Java is cool.,
    -55, The last entry.]

[3.14159, 27, A, -4, 17, 2.718281828459045, 17.6, 3.141592653589793, 17.6, Java is cool.,
    -55, The last entry.]
[3.14159, 27, A, 17, 2.718281828459045, 17.6, -55, The last entry.]
```

Chapter 13

Files

13.1 Introduction

Java has a lot of facilities for file reading, writing and manipulation. We will look at just a couple of these. There are essentially two types of files, text files and binary files. A text file is one you can open up in Microsoft Notepad or GEdit or any word processor. It is readable and consists of keyboard characters. A binary file is simply a string of 1's and 0's and although you can open them in a text editor, they will not be readable. Files like the java code files you have been writing are text files. Although you have been opening them in Eclipse, you could open them in a text editor or word processor and they will look the same, without the syntax highlighting that Eclipse does. Files like executable files are binary files. If you try to open one in a text editor you will see a lot of garbage looking symbols. If you do this, be careful not to alter and save the executable file. If you do, it will not run.

13.2 Basic File Reading Example

In our first example, we are reading in a text file that contains five integers. We then load these integers into an ArrayList in the program. Before we can read anything from a file we need to associate the file with a File variable in the program. The line,

```
File infile = new File("DataFile.txt");
```

does that association. Now `infile` is linked to the file `DataFile.txt` which resides in the same folder as our project. To gain access to the file, we use a Scanner. The line,

```
Scanner input = new Scanner(infile);
```

creates a Scanner that is linked to the file. Now the file is essentually like the keyboard. We can read from the file in the same manner as we did from the keyboard, using methods like `nextInt` or `nextLine`. For example, `input.nextInt()` reads the next integer from the file, just like we read the next integer from the keyboard. Remember back to keyboard inputs, if we use `nextInt` but type in a double we get an error. The same thing can happen here so we need to be careful that we read the correct data type out of the file or the program could crash. If we want to read the entire file into the program we either need to know how many items are in the file or in some way figure out when the file is ended. In this example we use the technique of reading until we cannot read any more data. When we try to read past the end of a file, an exception is thrown. So we simply put it in a try-catch block. Notice in the output, we get an exception on the read of item 6, since there are only 5 items in the file. If we did not do this print out of the exception our program would simply read in the entire file into the ArrayList and then print it out. We put the exception print out in the program to illustrate the method.

Recall that we said that closing the keyboard was not all that critical, when dealing with files it is very important that we close the file when we are finished with it. The command `input.close()` will close the file.

```
1  import java.io.File;
2  import java.util.ArrayList;
3  import java.util.Scanner;
4
5  /*-
6   * FileExample001
7   * Example showing how to read a file until an exception occurs
8   * when reading past the end of the file.
9   * Author:  Don Spickler
10  * Date: 3/24/2011
11  */
12
13 public class FileExample001 {
14
15     public static void main(String[] args) {
16         ArrayList filecontents = new ArrayList();
17
18         try {
19             File infile = new File("DataFile.txt");
20             Scanner input = new Scanner(infile);
21
22             int i = 1;
23             try {
24                 while (true) {
25                     filecontents.add(input.nextInt());
26                     i++;
27                 }
28             } catch (Exception ex) {
29                 System.out.println("Exception at " + i + "  ---  " + ex.getMessage());
30             }
31
32             input.close();
33         } catch (Exception e) {
34             System.out.println("Could not open file for input.");
35         }
36
```

```
37          for (int i = 0; i < filecontents.size(); i++)
38              System.out.println(filecontents.get(i));
39      }
40  }
```

DataFile.txt

```
1  12
2  32
3  43
4  55
5  132
```

Program Output **FileExample001.java**

```
Exception at 6  ---  null
12
32
43
55
132
```

13.3 Basic File Writing Example

Writing to a file is just as easy. We link the output file to a variable in the same way,

```
File outfile = new File("DataFile.txt");
```

Then we gain access not with a Scanner, since that is for reading, but with a PrintWriter,

```
PrintWriter output = new PrintWriter(outfile);
```

Now we can use `output` just like `System.out`. That is we can do print and println commands on `output` and the results will go to the file and not to the console screen.

```
output.println((int) (Math.random() * 100) + 1);
```

When we are finished writing to the file we need to flush it and close it,

```
output.flush();
output.close();
```

The flush method flushes the output buffer and then close will close the file. When you do a print or println to a file, the prints do not go immediately to the file, they are held in a memory location, an output buffer, until the disk is ready to be written to and then the data is written to the file. So if you close the file before the data is written you may be missing some information that was to be written.

```java
import java.io.File;
import java.io.PrintWriter;

/*-
 * FileExample002
 * Example showing how to write to a file.
 * Author:  Don Spickler
 * Date: 3/24/2011
 */

public class FileExample002 {

    public static void main(String[] args) {
        try {
            File outfile = new File("DataFile.txt");
            PrintWriter output = new PrintWriter(outfile);

            for (int i = 0; i < 25; i++)
                output.println((int) (Math.random() * 100) + 1);

            output.flush();
            output.close();
        } catch (Exception e) {
            System.out.println("Could not open file for output.");
        }
    }
}
```

DataFile.txt

```
57
90
19
42
6
8
44
47
4
61
67
93
61
65
90
97
72
85
67
41
3
4
68
63
8
```

13.4 File Copy Example

This example simply reads in a text file and saves the contents to another text file.

```java
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;

/*-
 * FileExample003
 * Example showing how to copy a file.
 * Author:  Don Spickler
 * Date: 3/24/2011
 */

public class FileExample003 {

    public static void main(String[] args) {
        try {
            File infile = new File("DataFile.txt");
            Scanner input = new Scanner(infile);
            File outfile = new File("DataFileCopy.txt");
            PrintWriter output = new PrintWriter(outfile);

            try {
                while (true) {
                    output.println(input.nextLine());
                }
            } catch (Exception ex) {
            }

            input.close();
            output.flush();
            output.close();
        } catch (Exception e) {
            System.out.println("Could not open files.");
        }
    }
}
```

DataFile.txt

```
Since the discovery of public key cryptography in 1976, cryptography has relied
    extensively on algebraic number theory, specifically on the properties of rings and
    fields.  The world's e-commerce system would not exist without the RSA algorithm, the
    most commonly used encryption and authentication algorithm, which utilizes these
    algebraic structures.  Furthermore, the Advanced Encryption Standard (AES), one of the
     most common symmetric cyphers, attributes its speed and security to the structure of
    factor rings, similar to those we have studied.

While these methods are secure by today's standards, with the ever increasing
    computational power of the computer, they will eventually need to be replaced.
    Research by Nobel Prize winners, David J. Wineland and Serge Haroche, suggests that if
     the quantum computer is successfully built then new cryptographic algorithms must be
    found.  In short, none of the current cryptographic methods currently used in practice
     will stand up to the computational power of the quantum computer.

With this realization, there is a substantial amount of research being done on post-
    quantum cryptography; cryptographic methods that will not succumb to the power of the
    quantum computer.  Many of the most promising methods being developed utilize the
    multiplicative group of units in ring structures, such as the ones we have studied.
    My research has produced concise formulas for determining the cyclic group
    decomposition of the multiplicative group of units for a large class of quotient rings
     closely related to those currently being used and those that are proposed for future
    cryptographic methods.
```

DataFileCopy.txt

```
1 Since the discovery of public key cryptography in 1976, cryptography has relied
      extensively on algebraic number theory, specifically on the properties of rings and
      fields.  The world's e-commerce system would not exist without the RSA algorithm, the
      most commonly used encryption and authentication algorithm, which utilizes these
      algebraic structures.  Furthermore, the Advanced Encryption Standard (AES), one of the
       most common symmetric cyphers, attributes its speed and security to the structure of
      factor rings, similar to those we have studied.
2
3 While these methods are secure by today's standards, with the ever increasing
      computational power of the computer, they will eventually need to be replaced.
      Research by Nobel Prize winners, David J. Wineland and Serge Haroche, suggests that if
       the quantum computer is successfully built then new cryptographic algorithms must be
      found.  In short, none of the current cryptographic methods currently used in practice
       will stand up to the computational power of the quantum computer.
4
5 With this realization, there is a substantial amount of research being done on post-
      quantum cryptography; cryptographic methods that will not succumb to the power of the
      quantum computer.  Many of the most promising methods being developed utilize the
      multiplicative group of units in ring structures, such as the ones we have studied.
      My research has produced concise formulas for determining the cyclic group
      decomposition of the multiplicative group of units for a large class of quotient rings
       closely related to those currently being used and those that are proposed for future
      cryptographic methods.
```

13.5 File Attributes Example

This example shows you how to get file and disk information from the file system.

```
1  import java.io.File;
2
3  /*-
4   * FileExample004
5   * Example showing how to get file attributes.
6   * Author:  Don Spickler
7   * Date: 3/24/2011
8   */
9
10 public class FileExample004 {
11
12     public static void main(String[] args) {
13         File infile = new File("DataFile.txt");
14
15         try {
16             System.out.println("Absolute Path: " + infile.getAbsolutePath());
17             System.out.println("Canonical Path: " + infile.getCanonicalPath());
18             System.out.println("Name: " + infile.getName());
19             System.out.println("Path: " + infile.getPath());
20             System.out.println("Parent: " + infile.getParent());
21             System.out.println("Length: " + infile.length());
22
23             System.out.println("Total Space: " + infile.getTotalSpace());
24             System.out.println("Usable Space: " + infile.getUsableSpace());
25             System.out.println("Free Space: " + infile.getFreeSpace());
26
27         } catch (Exception e) {
28             System.out.println("Error: " + e.getMessage());
29         }
30     }
```

```
31 }
```

```
1 Since the discovery of public key cryptography in 1976, cryptography has relied
      extensively on algebraic number theory, specifically on the properties of rings and
      fields.  The world's e-commerce system would not exist without the RSA algorithm, the
      most commonly used encryption and authentication algorithm, which utilizes these
      algebraic structures.  Furthermore, the Advanced Encryption Standard (AES), one of the
       most common symmetric cyphers, attributes its speed and security to the structure of
      factor rings, similar to those we have studied.
2
3 While these methods are secure by today's standards, with the ever increasing
      computational power of the computer, they will eventually need to be replaced.
      Research by Nobel Prize winners, David J. Wineland and Serge Haroche, suggests that if
       the quantum computer is successfully built then new cryptographic algorithms must be
      found.  In short, none of the current cryptographic methods currently used in practice
       will stand up to the computational power of the quantum computer.
4
5 With this realization, there is a substantial amount of research being done on post-
      quantum cryptography; cryptographic methods that will not succumb to the power of the
      quantum computer.  Many of the most promising methods being developed utilize the
      multiplicative group of units in ring structures, such as the ones we have studied.
      My research has produced concise formulas for determining the cyclic group
      decomposition of the multiplicative group of units for a large class of quotient rings
       closely related to those currently being used and those that are proposed for future
      cryptographic methods.
```

Program Output **FileExample004.java**

```
Absolute Path: C:\Users\despickler\Documents\Classes\IntroJava\AllJavaCodeExamples\
    FileExample004\DataFile.txt
Canonical Path: C:\Users\despickler\Documents\Classes\IntroJava\AllJavaCodeExamples\
    FileExample004\DataFile.txt
Name: DataFile.txt
Path: DataFile.txt
Parent: null
Length: 1652
Total Space: 500000878592
Usable Space: 301126500352
Free Space: 301126500352
```

13.6 File Contents Counts Example

This example shows you how to get line and word counts from a file. Notice the use of the `split` method for strings that will return an array of string split over the given substring.

```
1  import java.io.File;
2  import java.util.Scanner;
3
4  /*-
5   * FileExample005
6   * Example showing how to find the word counts, line count, non-blank line count,
7   * character and non-space character counts.  Note that replacing the infile line
8   * with the commented line above it will run the program on the program code file,
9   * that is, this file.
10  * Author:  Don Spickler
```

```
 * Date: 3/24/2011
 */

public class FileExample005 {

    public static void main(String[] args) {
        int wordcount = 0;
        long charcount = 0;
        long charcountnospace = 0;
        long lines = 0;
        long nonblanklines = 0;

        try {
            // File infile = new File("src\\FileExample005.java");
            File infile = new File("DataFile.txt");
            Scanner input = new Scanner(infile);

            try {
                while (true) {
                    String str = input.nextLine();
                    lines++;
                    str = str.trim();
                    charcount += str.length();

                    if (!str.equalsIgnoreCase("")) {
                        str = str.replaceAll("  ", " ");
                        String words[] = str.split(" ");
                        wordcount += words.length;
                        nonblanklines++;
                    }
                    str = str.replaceAll(" ", "");
                    charcountnospace += str.length();
                }
            } catch (Exception ex) {
            }

            input.close();

            System.out.println("Character Count: " + charcount);
            System.out.println("Character Count (No Spaces): " + charcountnospace);
            System.out.println("Word Count: " + wordcount);
            System.out.println("Lines: " + lines);
            System.out.println("Non Blank Lines: " + nonblanklines);
        } catch (Exception e) {
            System.out.println("Could not open file.");
        }
    }
}
```

DataFile.txt

```
Since the discovery of public key cryptography in 1976, cryptography has relied
    extensively on algebraic number theory, specifically on the properties of rings and
    fields.  The world's e-commerce system would not exist without the RSA algorithm, the
    most commonly used encryption and authentication algorithm, which utilizes these
    algebraic structures.  Furthermore, the Advanced Encryption Standard (AES), one of the
     most common symmetric cyphers, attributes its speed and security to the structure of
    factor rings, similar to those we have studied.

While these methods are secure by today's standards, with the ever increasing
    computational power of the computer, they will eventually need to be replaced.
    Research by Nobel Prize winners, David J. Wineland and Serge Haroche, suggests that if
     the quantum computer is successfully built then new cryptographic algorithms must be
```

```
    found.  In short, none of the current cryptographic methods currently used in practice
     will stand up to the computational power of the quantum computer.
4
5 With this realization, there is a substantial amount of research being done on post-
    quantum cryptography; cryptographic methods that will not succumb to the power of the
    quantum computer.  Many of the most promising methods being developed utilize the
    multiplicative group of units in ring structures, such as the ones we have studied.
    My research has produced concise formulas for determining the cyclic group
    decomposition of the multiplicative group of units for a large class of quotient rings
     closely related to those currently being used and those that are proposed for future
    cryptographic methods.
```

Program Output **FileExample005.java**

```
Character Count: 1638
Character Count (No Spaces): 1389
Word Count: 246
Lines: 5
Non Blank Lines: 3
```

13.7 Text File Reformatting Example

This is a slightly more advanced example showing how to reformat a text file into a file that has at most 80 characters per line and the lines at their maximum length in the sense that none of the words from the beginning of a subsequent line can be moved up to the previous line and still remain 80 characters or less.

```
1  import java.io.File;
2  import java.io.PrintWriter;
3  import java.util.Scanner;
4
5  /*-
6   * FileExample006
7   * More advanced example showing how to reformat a text file into a file that
8   * has at most 80 characters per line and the lines at their maximum length
9   * in the sense that none of the words from the beginning of a subsequent line
10  * can be moved up to the previous line and still remain 80 characters or less.
11  * Author:  Don Spickler
12  * Date: 3/24/2011
13  */
14
15 public class FileExample006 {
16
17     public static void main(String[] args) {
18         int width = 80;
19         String fileString = "";
20         String fileStringFormatted = "";
21         String lineFormatted = "";
22
23         try {
24             File infile = new File("DataFile.txt");
25             Scanner input = new Scanner(infile);
26             File outfile = new File("DataFileFormatted.txt");
27             PrintWriter output = new PrintWriter(outfile);
28
29             try {
30                 while (true) {
```

```
31                      String str = input.nextLine().trim();
32
33                      if (str.equalsIgnoreCase(""))
34                          fileString += "\n\n";
35                      else
36                          fileString += str + " ";
37                  }
38              } catch (Exception ex) {
39              }
40
41              String paragraphs[] = fileString.split("\n");
42
43              for (int i = 0; i < paragraphs.length; i++) {
44                  String paragraph = paragraphs[i].trim();
45
46                  if (paragraph.equalsIgnoreCase("")) {
47                      fileStringFormatted += "\n";
48                  } else {
49                      paragraph = paragraph.replaceAll("  ", " ");
50                      String words[] = paragraph.split(" ");
51                      int wordpos = 0;
52
53                      while (wordpos < words.length) {
54                          if (lineFormatted.length() + words[wordpos].length() < width) {
55                              lineFormatted += words[wordpos] + " ";
56                              wordpos++;
57                          } else {
58                              fileStringFormatted += lineFormatted.trim() + "\n";
59                              lineFormatted = "";
60                          }
61                      }
62
63                      if (!lineFormatted.equalsIgnoreCase("")) {
64                          fileStringFormatted += lineFormatted.trim() + "\n";
65                          lineFormatted = "";
66                      }
67                  }
68              }
69
70              output.println(fileStringFormatted);
71
72              output.flush();
73              output.close();
74              input.close();
75          } catch (Exception e) {
76              System.out.println("Error: " + e.getMessage());
77          }
78      }
79  }
```

DataFile.txt

```
1  Since the discovery of public key cryptography
2  in 1976, cryptography has relied
3  extensively on algebraic number theory, specifically on the properties of rings and fields
       .
4  The world's e-commerce system would not exist without the RSA algorithm, the most commonly
        used encryption
5  and authentication algorithm, which utilizes
6  these algebraic structures.  Furthermore, the Advanced Encryption Standard (AES),
7  one of the most common symmetric cyphers, attributes its
8  speed and security to the structure of factor rings, similar to those we have studied.
9
```

```
10 While these methods are secure by today's
11 standards, with the ever increasing computational power of the computer, they will
       eventually
12 need to be replaced.  Research by Nobel Prize winners, David J. Wineland and Serge Haroche
       ,
13 suggests that if the quantum computer is
14 successfully built then new cryptographic algorithms must be found.  In short,
15 none of the current cryptographic methods currently used in practice will stand up to the
16 computational power of the quantum computer.
17
18 With this realization, there is a substantial amount of research being done on
19 post-quantum cryptography; cryptographic methods that will not succumb
20 to the power of the quantum computer.
21 Many of the most promising methods being developed utilize the multiplicative group of
       units in ring structures,
22 such as the ones we have studied.  My research has
23 produced concise formulas for determining the cyclic group decomposition
24 of the multiplicative group of units for a large class of quotient rings closely related
       to those currently being
25 used and those that are proposed for future cryptographic methods.
```

DataFileFormatted.txt

```
1  Since the discovery of public key cryptography in 1976, cryptography has relied
2  extensively on algebraic number theory, specifically on the properties of rings
3  and fields. The world's e-commerce system would not exist without the RSA
4  algorithm, the most commonly used encryption and authentication algorithm,
5  which utilizes these algebraic structures. Furthermore, the Advanced Encryption
6  Standard (AES), one of the most common symmetric cyphers, attributes its speed
7  and security to the structure of factor rings, similar to those we have
8  studied.
9
10 While these methods are secure by today's standards, with the ever increasing
11 computational power of the computer, they will eventually need to be replaced.
12 Research by Nobel Prize winners, David J. Wineland and Serge Haroche, suggests
13 that if the quantum computer is successfully built then new cryptographic
14 algorithms must be found. In short, none of the current cryptographic methods
15 currently used in practice will stand up to the computational power of the
16 quantum computer.
17
18 With this realization, there is a substantial amount of research being done on
19 post-quantum cryptography; cryptographic methods that will not succumb to the
20 power of the quantum computer. Many of the most promising methods being
21 developed utilize the multiplicative group of units in ring structures, such as
22 the ones we have studied. My research has produced concise formulas for
23 determining the cyclic group decomposition of the multiplicative group of units
24 for a large class of quotient rings closely related to those currently being
25 used and those that are proposed for future cryptographic methods.
```

13.8 Binary File Writing Example

Writing to a binary file is fairly easy, again this is only one way of many to do this. What we are going to do is put everything we want to write into an ArrayList and then use Java's built in ArrayList writer to write it to the disk. To do this we first create and load the information into the ArrayList and then link the file to a file output stream,

```
FileOutputStream fileOut = new FileOutputStream("BinaryFile.
   bin");
```

then make the stream ready for objects,

```
ObjectOutputStream output = new ObjectOutputStream(fileOut);
```

write the ArrayList,

```
output.writeObject(A);
```

then close everything up,

```
output.close();
fileOut.close();
```

```
1  import java.io.FileOutputStream;
2  import java.io.ObjectOutputStream;
3  import java.util.ArrayList;
4
5  /*-
6   * FileExample007
7   * Basic example of writing to a binary file.  Note that we are writing
8   * a non-typed ArrayList to the file.
9   * Author:  Don Spickler
10  * Date: 3/24/2011
11  */
12
13 public class FileExample007 {
14
15     public static void main(String[] args) {
16         ArrayList A = new ArrayList();
17
18         A.add(12.3);
19         A.add(Math.PI);
20         A.add("A string thing.");
21         A.add(25);
22
23         try {
24             FileOutputStream fileOut = new FileOutputStream("BinaryFile.bin");
25             ObjectOutputStream output = new ObjectOutputStream(fileOut);
26             output.writeObject(A);
27             output.close();
28             fileOut.close();
29         } catch (Exception e) {
30             System.out.println("Error: " + e.getMessage());
31         }
32     }
33 }
```

13.9 Binary File Read Example

Reading a binary file that is stored as in the last example is read in the same way it was written. Link the file to an input stream,

```
FileInputStream fileIn = new FileInputStream("BinaryFile.bin
    ");
```

set it for objects,

```
ObjectInputStream input = new ObjectInputStream(fileIn);
```

read and cast the ArrayList,

```
A = (ArrayList) input.readObject();
```

then close everything up,

```
input.close();
fileIn.close();
```

```
1  import java.io.FileInputStream;
2  import java.io.ObjectInputStream;
3  import java.util.ArrayList;
4
5  /*-
6   * FileExample008
7   * Basic example of reading from a binary file.  Note that we are reading
8   * a non-typed ArrayList object from the file, hence we need to cast the
9   * object.
10  * Author:  Don Spickler
11  * Date: 3/24/2011
12  */
13
14 public class FileExample008 {
15
16     public static void main(String[] args) {
17         ArrayList A = new ArrayList();
18
19         try {
20             FileInputStream fileIn = new FileInputStream("BinaryFile.bin");
21             ObjectInputStream input = new ObjectInputStream(fileIn);
22             A = (ArrayList) input.readObject();
23             input.close();
24             fileIn.close();
25         } catch (Exception e) {
26             System.out.println("Error: " + e.getMessage());
27         }
28
29         System.out.println(A);
30     }
31 }
```

Program Output **FileExample008.java**

```
[12.3, 3.141592653589793, A string thing., 25]
```

13.10 Binary File Example with Programmer Created Objects

This example reads and writes a binary file of Employee data types. The main does the reading and writing the same way as in the previous examples. Before we can apply this method to a user-defined data type, like Employee, we need to make the Employee class Serializable. To do this you need to do two things, first you need to put the following import command at the top.

```
import java.io.Serializable;
```

Then you need to implement Serializable for the class by placing

```
implements Serializable
```

after the class name.

```
1  import java.io.Serializable;
2
3  /*-
4   * Employee
5   * Employee class stores information about a company employee, name,
6   * wage, and hours worked for the week. Note the use of Serializable
7   * so that we can read and write the object contents and the overloaded
8   * toString method that allows the main program to use a printline on
9   * an Employee Object.
10  * Author:  Don Spickler
11  * Date: 3/20/2011
12  */
13
14 public class Employee implements Serializable {
15     private String firstname;
16     private String lastname;
17     private double wage;
18     private double hours_worked;
19
20     public Employee(String fn, String ln, double w, double hw) {
21         firstname = fn;
22         lastname = ln;
23
24         if (w > 0)
25             wage = w;
26         else
27             wage = 0;
28
29         if (hw > 0)
30             hours_worked = hw;
31         else
32             hours_worked = 0;
33     }
34
35     public String getName() {
36         return firstname + " " + lastname;
37     }
38
39     public String getFormalName() {
40         return lastname + ", " + firstname;
41     }
```

```
42
43      public double getWage() {
44          return wage;
45      }
46
47      public double getHoursWorked() {
48          return hours_worked;
49      }
50
51      public void setName(String fn, String ln) {
52          firstname = fn;
53          lastname = ln;
54      }
55
56      public void setWage(double w) {
57          if (w > 0)
58              wage = w;
59          else
60              wage = 0;
61      }
62
63      public void getHoursWorked(double hw) {
64          if (hw > 0)
65              hours_worked = hw;
66          else
67              hours_worked = 0;
68      }
69
70      public double pay() {
71          double payment = 0;
72          if (hours_worked > 40)
73              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
74          else
75              payment = wage * hours_worked;
76
77          return payment;
78      }
79
80      public String toString() {
81          String retstr = "";
82          retstr += "Name: " + getFormalName() + "\n";
83          retstr += "Wage: " + getWage() + "\n";
84          retstr += "Hours Worked: " + getHoursWorked() + "\n";
85          retstr += "Pay: " + (Math.round(pay() * 100.0) / 100.0) + "\n";
86
87          return retstr;
88      }
89  }
```

```
1   import java.util.ArrayList;
2   import java.util.Scanner;
3   import java.io.FileOutputStream;
4   import java.io.ObjectOutputStream;
5   import java.io.FileInputStream;
6   import java.io.ObjectInputStream;
7
8   /*-
9    * FileExample009
10   * More advanced example showing the reading and writing of a binary file
11   * that contains an ArrayList of user-defined objects.
12   * Author:  Don Spickler
13   * Date: 3/24/2011
14   */
15
```

```
16 public class FileExample009 {
17
18     public static void AddEmployee(ArrayList<Employee> A){
19         Scanner keyboard = new Scanner(System.in);
20         System.out.print("First Name: ");
21         String fn = keyboard.nextLine();
22         System.out.print("Last Name: ");
23         String ln = keyboard.nextLine();
24         System.out.print("Wage: ");
25         double wage = keyboard.nextDouble();
26         System.out.print("Hours Worked: ");
27         double hours = keyboard.nextDouble();
28
29         A.add(new Employee(fn, ln, wage, hours));
30
31         String clear = keyboard.nextLine();
32         System.out.println();
33     }
34
35     public static void main(String[] args) {
36         ArrayList<Employee> company = new ArrayList<Employee>();
37
38         for (int i = 0; i < 3; i++)
39             AddEmployee(company);
40
41         for (int i = 0; i < company.size(); i++)
42             System.out.println(company.get(i));
43
44         try {
45             FileOutputStream fileOut = new FileOutputStream("CompanyRecords.bin");
46             ObjectOutputStream output = new ObjectOutputStream(fileOut);
47             output.writeObject(company);
48             output.close();
49             fileOut.close();
50         } catch (Exception e) {
51             System.out.println("Error: " + e.getMessage());
52         }
53
54         company.clear();
55         System.out.println();
56         System.out.println(company.size());
57         System.out.println();
58
59         try {
60             FileInputStream fileIn = new FileInputStream("CompanyRecords.bin");
61             ObjectInputStream input = new ObjectInputStream(fileIn);
62             company = (ArrayList<Employee>) input.readObject();
63             input.close();
64             fileIn.close();
65         } catch (Exception e) {
66             System.out.println("Error: " + e.getMessage());
67         }
68
69         for (int i = 0; i < company.size(); i++)
70             System.out.println(company.get(i));
71     }
72 }
```

Program Output — FileExample009.java

```
First Name: Don
Last Name: Spickler
Wage: 10.25
```

```
Hours Worked: 53

First Name: Jane
Last Name: Doe
Wage: 15.32
Hours Worked: 37

First Name: Jack
Last Name: Frost
Wage: 17.50
Hours Worked: 35

Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84

Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5


0

Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84

Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5
```

13.11 Binary File Example: Programmer Created Objects and Sorting

In this example we update the last example to incorporate sorting of the ArrayList. We simply add in the `Comparable` to the `Serializable` in the class header and add in the `compareTo` method we discussed several examples ago.

```
1 import java.io.Serializable;
2
3 /*-
4  * Employee
5  * Employee class stores information about a company employee, name,
```

```
 * wage, and hours worked for the week. Note the use of Serializable
 * so that we can read and write the object contents, Comparable so
 * that we can use the Collections class to sort the array, and the
 * overloaded toString method that allows the main program to use a
 * printline on an Employee Object.
 * Author:  Don Spickler
 * Date: 3/20/2011
 */

public class Employee implements Serializable, Comparable {
    private String firstname;
    private String lastname;
    private double wage;
    private double hours_worked;

    public Employee(String fn, String ln, double w, double hw) {
        firstname = fn;
        lastname = ln;

        if (w > 0)
            wage = w;
        else
            wage = 0;

        if (hw > 0)
            hours_worked = hw;
        else
            hours_worked = 0;
    }

    public String getName() {
        return firstname + " " + lastname;
    }

    public String getFormalName() {
        return lastname + ", " + firstname;
    }

    public double getWage() {
        return wage;
    }

    public double getHoursWorked() {
        return hours_worked;
    }

    public void setName(String fn, String ln) {
        firstname = fn;
        lastname = ln;
    }

    public void setWage(double w) {
        if (w > 0)
            wage = w;
        else
            wage = 0;
    }

    public void getHoursWorked(double hw) {
        if (hw > 0)
            hours_worked = hw;
        else
            hours_worked = 0;
    }
```

```
70
71      public double pay() {
72          double payment = 0;
73          if (hours_worked > 40)
74              payment = wage * 40 + (hours_worked - 40) * wage * 1.5;
75          else
76              payment = wage * hours_worked;
77
78          return payment;
79      }
80
81      public String toString() {
82          String retstr = "";
83          retstr += "Name: " + getFormalName() + "\n";
84          retstr += "Wage: " + getWage() + "\n";
85          retstr += "Hours Worked: " + getHoursWorked() + "\n";
86          retstr += "Pay: " + (Math.round(pay() * 100.0) / 100.0) + "\n";
87
88          return retstr;
89      }
90
91      public int compareTo(Object e2) {
92          return this.getFormalName().toUpperCase().compareTo(((Employee) e2).getFormalName
                ().toUpperCase());
93      }
94  }
```

```
1   import java.io.FileInputStream;
2   import java.io.FileOutputStream;
3   import java.io.ObjectInputStream;
4   import java.io.ObjectOutputStream;
5   import java.util.ArrayList;
6   import java.util.Collections;
7   import java.util.Scanner;
8
9   /*-
10   * FileExample009
11   * More advanced example showing the reading and writing of a binary file
12   * that contains an ArrayList of user-defined objects and the use of the
13   * Collections object for sorting the list.
14   * Author:  Don Spickler
15   * Date: 3/24/2011
16   */
17
18  public class FileExample010 {
19
20      public static void AddEmployee(ArrayList<Employee> A) {
21          Scanner keyboard = new Scanner(System.in);
22          System.out.print("First Name: ");
23          String fn = keyboard.nextLine();
24          System.out.print("Last Name: ");
25          String ln = keyboard.nextLine();
26          System.out.print("Wage: ");
27          double wage = keyboard.nextDouble();
28          System.out.print("Hours Worked: ");
29          double hours = keyboard.nextDouble();
30
31          A.add(new Employee(fn, ln, wage, hours));
32
33          String clear = keyboard.nextLine();
34          System.out.println();
35      }
36
37      public static void SaveFile(ArrayList<Employee> company) {
```

```
38          try {
39              FileOutputStream fileOut = new FileOutputStream("CompanyRecords.bin");
40              ObjectOutputStream output = new ObjectOutputStream(fileOut);
41              output.writeObject(company);
42              output.close();
43              fileOut.close();
44          } catch (Exception e) {
45              System.out.println("Error: " + e.getMessage());
46          }
47      }
48
49      public static ArrayList<Employee> OpenFile() {
50          try {
51              FileInputStream fileIn = new FileInputStream("CompanyRecords.bin");
52              ObjectInputStream input = new ObjectInputStream(fileIn);
53              ArrayList<Employee> company = (ArrayList<Employee>) input.readObject();
54              input.close();
55              fileIn.close();
56              return company;
57          } catch (Exception e) {
58              System.out.println("Error: " + e.getMessage());
59          }
60
61          return null;
62      }
63
64      public static void main(String[] args) {
65          ArrayList<Employee> company = new ArrayList<Employee>();
66
67          for (int i = 0; i < 3; i++)
68              AddEmployee(company);
69
70          System.out.println("======================");
71
72          for (int i = 0; i < company.size(); i++)
73              System.out.println(company.get(i));
74
75          System.out.println("----------------------");
76
77          SaveFile(company);
78
79          company.clear();
80          System.out.println(company.size());
81
82          System.out.println("----------------------");
83
84          company = OpenFile();
85
86          for (int i = 0; i < company.size(); i++)
87              System.out.println(company.get(i));
88
89          System.out.println("----------------------");
90
91          Collections.sort(company);
92
93          for (int i = 0; i < company.size(); i++)
94              System.out.println(company.get(i));
95
96          System.out.println("----------------------");
97
98          SaveFile(company);
99          company = OpenFile();
100
101         for (int i = 0; i < company.size(); i++)
```

```
102             System.out.println(company.get(i));
103
104         System.out.println("----------------------");
105     }
106 }
```

Program Output — FileExample010.java

```
First Name: Don
Last Name: Spickler
Wage: 10.25
Hours Worked: 53

First Name: Jane
Last Name: Doe
Wage: 15.32
Hours Worked: 37

First Name: Jack
Last Name: Frost
Wage: 17.50
Hours Worked: 35

======================
Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84

Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5

----------------------
0
----------------------
Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84

Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5

----------------------
Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84
```

```
Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5

Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

----------------------
Name: Doe, Jane
Wage: 15.32
Hours Worked: 37.0
Pay: 566.84

Name: Frost, Jack
Wage: 17.5
Hours Worked: 35.0
Pay: 612.5

Name: Spickler, Don
Wage: 10.25
Hours Worked: 53.0
Pay: 609.88

----------------------
```

Chapter 14

Recursion

14.1 Introduction

Recursion is when a method calls itself, or when a method calls another method, then another, and another but at some point it comes back to the original method. This first type, when a method calls itself directly is called *simple recursion*, and when there is a chain of calls that cycle back to a method it is called *mutual recursion*. Whenever you have a calculation, or any other process, that can be completed by defining it in terms of itself you have a recursive definition of the calculation or process and it can be implemented as a recursive method. For example, the Fibonacci sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... it is constructed by starting with two 1's and then every number after that is the sum of the two numbers before it. So $2 = 1 + 1$, $3 = 2 + 1$, $5 = 3 + 2$, $8 = 5 + 3$, ... If we call F_n the n^{th} Fibonacci number then we can write $F_n = F_{n-1} + F_{n-2}$. Hence we can define a Fibonacci number in terms of itself. So this is a recursive definition of a Fibonacci number.

Recursion has many uses both inside and outside mathematics. In fact, most parsers incorporate mutual recursion. So when you compile and run a program, your compiler will be using recursion to compile your program. Similarly, when you open up a web page your browser will take the HTML (and other language components) of the web page and parse it into the page you actually see. If you have never done this, you may want to view a page source from a web page to see what your browser gets from the web and subsequently changes into the nice graphical page you see.

14.2 Recursive Method Example Showing the Call Stack

When you call a method from itself, each call creates a new version of the method and stores all of the parameter values for the new method, without deleting the values from the calling method. You can visualize these as their own box in the following diagram.

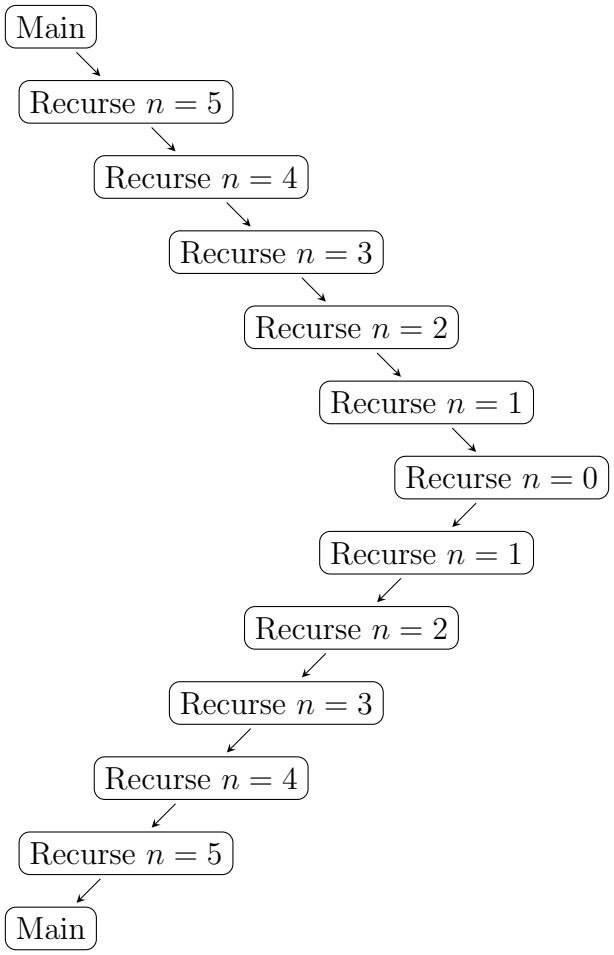


Figure 14.1: Call Stack for the Recursive method in this example.

So the main calls Recurse with a value of $n = 5$ for the parameter. Then inside Recurse we call Recurse with parameter $n - 1$, so $n = 4$. It, in turn, calls Recurse again on $n - 1$, so $n = 3$, and so on. Note that when $n = 0$ the if condition is false and so we do not call Recurse again but instead write the "Out Recurse" statement. Now we are at the end of the $n = 0$ call to Recurse, so as always, we go back to the calling method, which was Recurse with $n = 1$. Then the Recurse with $n = 1$ writes

out its "Out Recurse" statement and finishes. So we go back to Recurse with $n = 2$, and so on until we finish up Recurse with $n = 5$ and then go back to the main.

Just like with loops, we must make sure that there is a stopping condition on the recursion. In our example here it is when the value of n reaches 0. If we did not have the if statement in this program we would continue to recurse indefinitely, which is of course, called infinite recursion. As with an infinite loop, you will be able to tell if you have an infinite recursion by your program not responding in a reasonable amount of time. If you let an infinite recursion run it will eventually stop because each time you recurse you store information for each call, so at some point you will run out of memory.

```
1  /*-
2   * Recursion000
3   * Example showing the call stack for the recursive calls.
4   * Author:  Don Spickler
5   * Date: 3/7/2011
6   */
7
8  public class Recursion000 {
9
10     public static void Recurse(int n) {
11         System.out.println("In Recurse n = " + n);
12         if (n > 0)
13             Recurse(n - 1);
14         System.out.println("Out Recurse n = " + n);
15     }
16
17     public static void main(String[] args) {
18         Recurse(5);
19     }
20 }
```

Program Output **Recursion001.java**

```
In Recurse n = 5
In Recurse n = 4
In Recurse n = 3
In Recurse n = 2
In Recurse n = 1
In Recurse n = 0
Out Recurse n = 0
Out Recurse n = 1
Out Recurse n = 2
Out Recurse n = 3
Out Recurse n = 4
Out Recurse n = 5
```

14.3 Basic Recursive Methods Example

In this example we do some recursive calculations. We saw in the introduction that the Fibonacci sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... is recursive with the recursion formula $F_n = F_{n-1} + F_{n-2}$. So in our code we will calculate the n^{th}

Fibonacci number by adding the recursive call on $n-1$ and the recursive call on $n-2$. The stopping condition is that $F_1 = 1$ and $F_2 = 1$. We will also do this calculation iteratively, which can be done by simply keeping track of the last two values.

Another calculation that can be done both recursively and iteratively is the calculation of the factorial. The factorial is defined for any non-negative integer n to be,

$$n! = \begin{cases} n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1 & \text{when } n > 0 \\ 1 & \text{when } n = 0 \end{cases}$$

So this can be calculated iteratively by an accumulator that multiplies instead of adds. We can also formulate this recursively, notice that $(n-1) \cdot (n-2) \cdots 2 \cdot 1 = (n-1)!$, so,

$$n! = \begin{cases} n \cdot (n-1)! & \text{when } n > 0 \\ 1 & \text{when } n = 0 \end{cases}$$

Determining if a number is divisible by 3 or 9 is also a recursive process. You can determine if a number is divisible by 3 if the sum of all of its digits is divisible by 3. The same is true for 9, you can determine if a number is divisible by 9 if the sum of all of its digits is divisible by 9. For example, take the number 78391752, the sum of its digits is, 42. So if 42 is divisible by 3 then so is 78391752. Now we know the 42 is divisible by 3 but if this number was larger it might not be so easy to see. We can apply the same process to 42. We want to determine if 42 is divisible by 3 (so we can answer the same question about 78391752), so add up the digits of 42 and we get 6, which is clearly divisible by 3. Since 6 is divisible by 3, so is 42 and so is 78391752. To put this into an algorithm, we will simply add all the digits of the numbers until we get to a single digit number. If that single digit number is either 3, 6, or 9, then the original number was divisible by 3. In the case of divisibility by 9, we follow the same process but the single digit number we get at the end must be 9 in order to have divisibility by 9. So we can see that 78391752 is not divisible by 9.

```
1  /*-
2   * Recursion001
3   * Examples of recursive methods, factorial, Fibonacci numbers,
4   * division by 3 and division by 9.
5   * Author:  Don Spickler
6   * Date: 3/7/2011
7   */
8
9  public class Recursion001 {
10
11     // Factorial: Recursive implementation n! = n*(n-1)!
12     public static long factorialRecursive(long n) {
13         if (n == 0)
14             return 1;
15
16         return n * factorialRecursive(n - 1);
17     }
18
19     // Factorial: Iterative implementation n! = 1*2*3*...*n
20     public static long factorialIterative(long n) {
21         long num = 1;
```

```
22          for (int i = 1; i <= n; i++)
23              num = num * i;
24
25          return num;
26      }
27
28      // Fibonacci Numbers: Recursive implementation Fn = Fn-1 + Fn-2
29      public static int FibonacciRecursive(int n) {
30          if ((n == 1) || (n == 2))
31              return 1;
32
33          return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);
34      }
35
36      // Fibonacci Numbers: Iterative implementation Fn = Fn-1 + Fn-2
37      public static long FibonacciIterative(long n) {
38          long backOne = 1;
39          long backTwo = 1;
40          long num = 1;
41          for (int i = 3; i <= n; i++) {
42              num = backOne + backTwo;
43              backTwo = backOne;
44              backOne = num;
45          }
46          return num;
47      }
48
49      public static int add_digits(int n) {
50          int sum = 0;
51          while (n > 0) {
52              sum += (n % 10);
53              n = n / 10;
54          }
55          return sum;
56      }
57
58      // Divisibility by 9 recursive
59      public static boolean isDiv9(int num) {
60          if (num == 9)
61              return true;
62          else if (num > 9)
63              return isDiv9(add_digits(num));
64
65          return false;
66      }
67
68      // Divisibility by 3 recursive
69      public static boolean isDiv3(int num) {
70          if (num == 3 || num == 6 || num == 9)
71              return true;
72          else if (num > 9)
73              return isDiv3(add_digits(num));
74
75          return false;
76      }
77
78      public static void main(String[] args) {
79          for (int i = 0; i <= 10; i++)
80              System.out.println("" + i + "! = " + factorialRecursive(i) + " = " +
                    factorialIterative(i));
81
82          System.out.println();
83
84          for (int i = 1; i <= 10; i++)
```

```
85              System.out.println("F" + i + " = " + FibonacciRecursive(i) + " = " +
                    FibonacciIterative(i));
86
87          System.out.println();
88          System.out.println(isDiv3(78391752));
89          System.out.println(isDiv3(78391750));
90          System.out.println(isDiv9(78391752));
91          System.out.println(isDiv9(78391755));
92      }
93  }
```

Program Output **Recursion001.java**

```
0! = 1 = 1
1! = 1 = 1
2! = 2 = 2
3! = 6 = 6
4! = 24 = 24
5! = 120 = 120
6! = 720 = 720
7! = 5040 = 5040
8! = 40320 = 40320
9! = 362880 = 362880
10! = 3628800 = 3628800

F1 = 1 = 1
F2 = 1 = 1
F3 = 2 = 2
F4 = 3 = 3
F5 = 5 = 5
F6 = 8 = 8
F7 = 13 = 13
F8 = 21 = 21
F9 = 34 = 34
F10 = 55 = 55

true
false
false
true
```

14.4 The Towers of Hanoi Example

The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower) is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.

[1]Image taken from Wikimedia Commons.

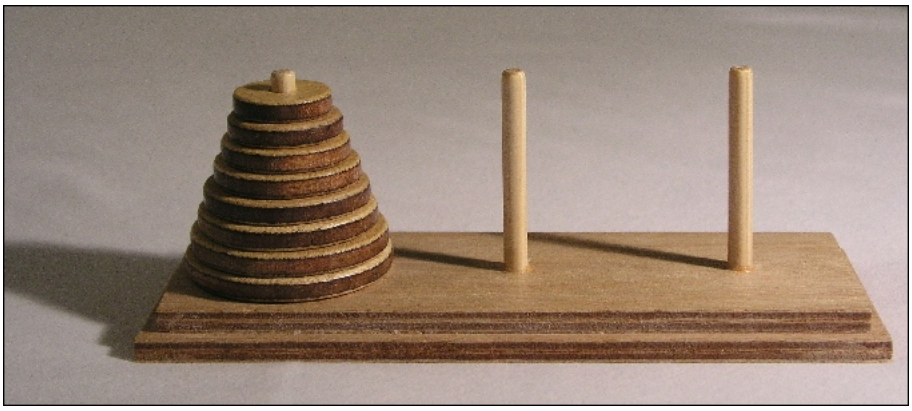


Figure 14.2: Towers of Hanoi Game[1]

2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3. No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

The puzzle was invented by the French mathematician Edouard Lucas in 1883. There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the immutable rules of the Brahma, since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. According to the legend, when the last move of the puzzle will be completed, the world will end. It is not clear whether Lucas invented this legend or was inspired by it.

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them $2^{64} - 1$ seconds or roughly 585 billion years or 18,446,744,073,709,551,615 turns to finish, or about 127 times the current age of the sun.

There are many variations on this legend. For instance, in some tellings, the temple is a monastery and the priests are monks. The temple or monastery may be said to be in different parts of the world — including Hanoi, Vietnam, and may be associated with any religion. In some versions, other elements are introduced, such as the fact that the tower was created at the beginning of the world, or that the priests

or monks may make only one move per day.[2]

We can solve this puzzle recursively. Let's label the pegs as A, B, and C, from left to right. We want to move all the disks from peg A to peg B, under the rules of the game. This leaves peg C as a spare or temporary peg. To solve this problem we could solve the problem for $n-1$ disks to move them from peg A to peg C, then move the largest disk from peg A to peg B, then solve the problem again on $n-1$ disks to move them from peg C to peg B, and we are done.

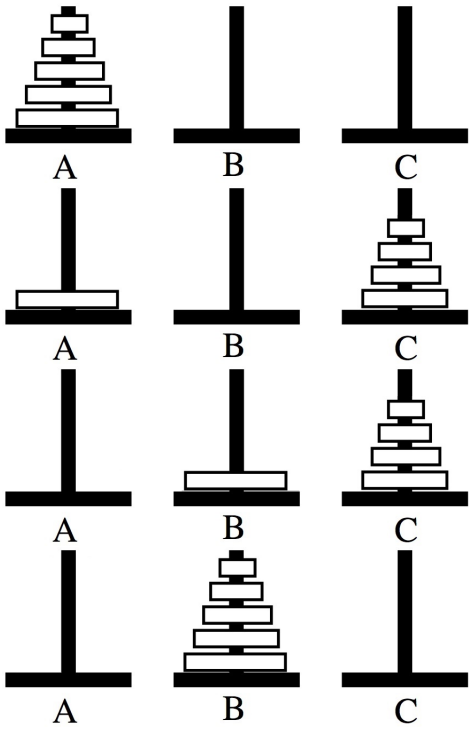


Figure 14.3: Towers of Hanoi Game Recursive Solution

Let's say that we construct a recursive method `hanoi` that solves the problem for n disks. Then this method would need to bring in as parameters the number of disks, the beginning peg letter, the ending peg letter and the spare or temporary peg letter. If the number of disks is greater than 1 the method must call itself with $n-1$ disks moving them from the beginning peg to the temporary peg, then move a disk (which will be the largest) from the beginning peg to the ending peg, and then call itself again with $n-1$ disks moving them from the temporary peg to the ending peg. If the number of disks is 1, the solution is very easy, we move the disk from the beginning peg to the ending peg. That is it, a very short and elegant solution to a game that could require many individual moves.

```
1 import java.util.Scanner;
```

[2]Taken from Wikipedi: https://en.wikipedia.org/wiki/Tower_of_Hanoi

```
2
3  /*-
4   * Recursion002
5   * The Towers of Hanoi
6   * Author:  Don Spickler
7   * Date: 3/7/2011
8   */
9
10 public class Recursion002 {
11
12     public static void hanoi(int n, String begin, String end, String temp) {
13         if (n == 1)
14             System.out.println("move " + begin + " to " + end);
15         else {
16             hanoi(n - 1, begin, temp, end);
17             System.out.println("move " + begin + " to " + end);
18             hanoi(n - 1, temp, end, begin);
19         }
20     }
21
22     public static void main(String[] args) {
23         Scanner keyboard = new Scanner(System.in);
24
25         System.out.print("Input the number of disks: ");
26         int numDisks = keyboard.nextInt();
27         hanoi(numDisks, "A", "B", "C");
28     }
29 }
```

Program Output — Recursion002.java Run #1

```
Input the number of disks: 3
move A to B
move A to C
move B to C
move A to B
move C to A
move C to B
move A to B
```

Program Output — Recursion002.java Run #2

```
Input the number of disks: 5
move A to B
move A to C
move B to C
move A to B
move C to A
move C to B
move A to B
move A to C
move B to C
move B to A
move C to A
move B to C
move A to B
move A to C
move B to C
move A to B
move C to A
move C to B
move A to B
```

```
move C to A
move B to C
move B to A
move C to A
move C to B
move A to B
move A to C
move B to C
move A to B
move C to A
move C to B
move A to B
```

Chapter 15

Introduction to Graphical User Interfaces

15.1 Introduction

Most of the applications that you use in your computer or mobile device are not console programs, like the ones we have been writing. Although, a lot of computations that are done "under the hood" while your operating system is operating are essentially console programs. They may not produce output to a console screen but they produce textual or binary output for other system-level processes. On the application level, like word processors or spreadsheets, you are working in a *Graphical User Interface* (GUI). A Graphical User Interface operates inside a window or frame, it has controls like menus, toolbars, buttons, text boxes, drop-down selection, and the list goes on.

In this chapter, we will look at how to create these frames and panels and draw basic graphical shapes onto them. In the next chapter we will go through the creation of a text editor we call JavaPad, which will be like Microsoft Notepad or GEdit in Linux.

These chapters are designed to give you a feel for how to do graphical user interface programming in Java, using the Java Swing interface. The area of graphical user interface programming, human computer interaction, and object oriented programming is huge and many institutions have entire classes on these topics. So we will only be scratching the surface of this topic. On the other hand, if you have made it this far, you have the needed programming background and you are ready to begin looking into GUI programming in Java. If this sparks your interest, please pick up a book or look at web resources on Java Swing.

15.2 Setting up a JFrame

The first step to GUI programming is creating a frame to draw graphics to or place controls on. In Java this is called a JFrame. Notice the class header,

```
public class IntroGUI001 extends JFrame
```

This `extends` reserved word means that our program is going to be a `JFrame`, that is, built got GUI. Technically, what this means is that our program is inheriting, and extending, the basic `JFrame` functionality. The main is fairly simple, we create an instance of our program `prog`, the constructor does nothing. We set the title to "GUI", we position and size the frame, it is 700 pixels wide, 500 pixels high, and is positioned 20 pixels in and 20 pixels down from the upper left corner of the of our screen. We make it visible and we move it to the front, on top of the windows that are already on the screen. Since we did not draw anything to the frame or place any controls on it, it is not all that interesting. Click the X in the title bar to close the program.

```
1  import javax.swing.JFrame;
2
3  /*-
4   * IntroGUI001
5   * Shows the basic setup for the creation of a JFrame.
6   * Author:  Don Spickler
7   * Date: 7/6/2016
8   */
9
10 public class IntroGUI001 extends JFrame {
11
12     public static void main(String[] args) {
13         IntroGUI001 prog = new IntroGUI001(args);
14         prog.setTitle("GUI");
15
16         prog.setBounds(20, 20, 700, 500);
17         prog.setVisible(true);
18         prog.toFront();
19     }
20
21     public IntroGUI001(String[] args) {
22     }
23 }
```

15.3 Drawing on the JFrame

When a frame or other object it created, Java must draw (or paint) the object. So if we wanted to draw something to the frame all we need to do is put the drawing code into a `paint` method. Here you are really overriding the internal `paint` method. The `paint` method must have the these exact parameters and the first line must be `super.paint(g);` This line tells Java to "paint" the frame and then we are going to draw on top of it. The `setColor` method sets the current drawing color. The

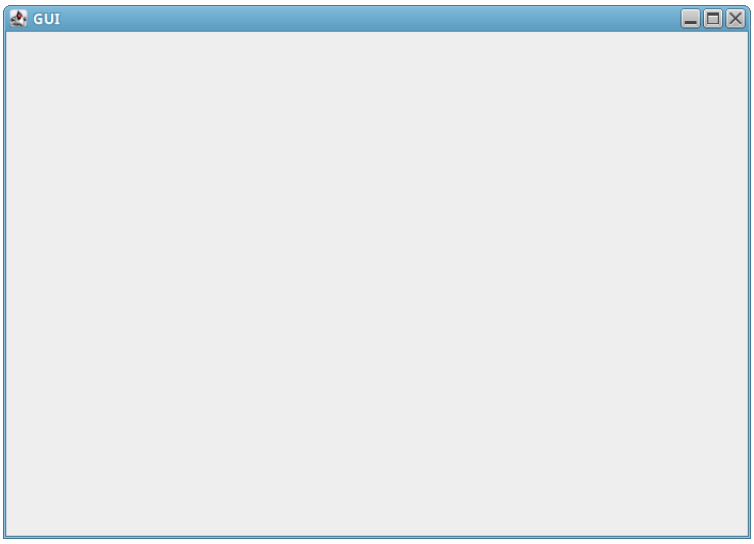


Figure 15.1: IntroGUI001.java Output

`Color` class has some built in colors and you can make your own color by using red, green, and blue values, we will see this later. The `fillOval` method takes in four integers, the first two are the (x, y) pixel position of the upper left corner and the last two are the width and height of the *invisible* rectangle that contains the oval. The oval will be drawn so that it is tangent to the midpoints of the sides of the rectangle. If the specified rectangle is a square then the oval is a circle.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI002
6   * Shows the basic setup for the creation of a JFrame and
7   * draws two circles to the frame by overriding the paint method.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class IntroGUI002 extends JFrame {
13
14     public static void main(String[] args) {
15         IntroGUI002 prog = new IntroGUI002(args);
16         prog.setTitle("GUI");
17
18         prog.setBounds(20, 20, 700, 500);
19         prog.setVisible(true);
20         prog.toFront();
21     }
22
23     public IntroGUI002(String[] args) {
24     }
25
26     public void paint(Graphics g) {
27         super.paint(g);
28
29         g.setColor(Color.black);
```

```
30          g.fillOval(60, 40, 220, 220);
31          g.setColor(Color.red);
32          g.fillOval(80, 60, 180, 180);
33      }
34  }
```

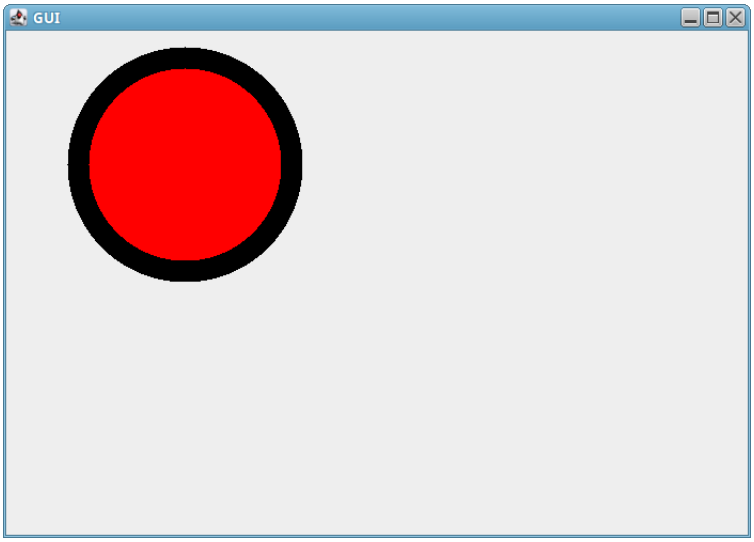


Figure 15.2: IntroGUI002.java Output

15.4 Lines and Colors

In the previous example we saw that the Java `Color` class had built in colors, we can also define our own color using the syntax `new Color(r, g, b)` where r, g, and b are integers between 0 and 255. The value 0 means no color of that component and 255 means full intensity of that color component. So $(255, 0, 0)$ is bright red, $(150, 150, 150)$ is gray, $(255, 255, 0)$ is bright yellow, and $(255, 255, 255)$ is bright white. If you have played around with the color editor in Microsoft Paint, or other painting program this should be familiar. The `drawLine` method takes in four integers, the first two are the (x, y) pixel position of one endpoint of the line and the last two are the (x, y) pixel position of the other endpoint of the line.

```
1  import java.awt.*;
2  import java.util.Random;
3  import javax.swing.*;
4
5  /*-
6   * IntroGUI003
7   * Draws 300 random lines in random colors inside the box [0, 300] X [0, 300].
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
```

```
12 public class IntroGUI003 extends JFrame {
13
14     private Random gen = new Random();
15
16     public static void main(String[] args) {
17         IntroGUI003 prog = new IntroGUI003(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
24
25     public IntroGUI003(String[] args) {
26     }
27
28     public void paint(Graphics g) {
29         super.paint(g);
30
31         for (int i = 0; i < 300; i++) {
32             g.setColor(new Color(gen.nextInt(255), gen.nextInt(255), gen.nextInt(255)));
33             g.drawLine(gen.nextInt(300), gen.nextInt(300), gen.nextInt(300), gen.nextInt
                   (300));
34         }
35     }
36 }
```

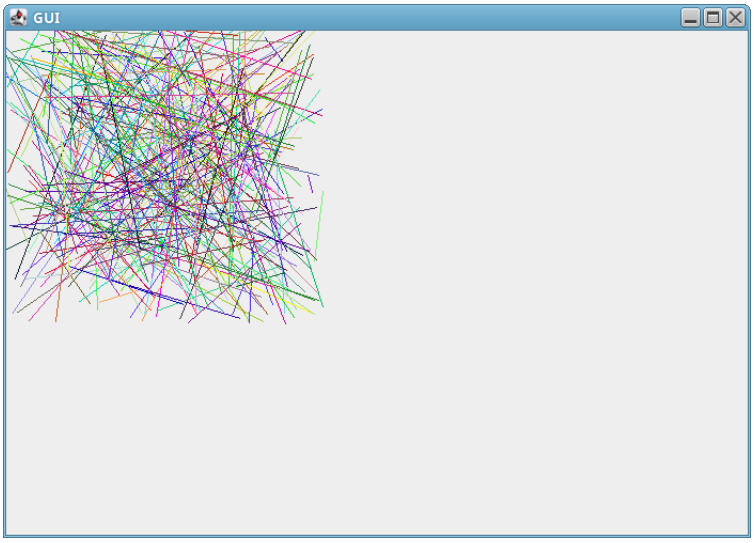


Figure 15.3: IntroGUI003.java Output

15.5 Getting the JFrame Bounds

In the last example, the lines did not fill the screen since we graphed them in the 300 by 300 square in the upper left corner, and the JFrame was larger than that. We can use the entire frame if we knew how wide and how high it is. We specified this in the

main but if we resize the frame we will need to get the new width and height. This is what the three lines,

```
Rectangle bounds = prog.getBounds();
int x = (int) bounds.getWidth();
int y = (int) bounds.getHeight();
```

will do. A JFrame has bounds that can be extracted and stored in a `Rectangle` object, which is internally defined in Java. We can then simply ask it for the height and width of the rectangle. Note that the height and width are doubles so we need to cast them to ints. When you run this example, resize the frame to see what happens. This shows you that when a frame is resized the paint method is called.

```
1  import java.awt.*;
2  import java.util.Random;
3  import javax.swing.*;
4
5  /*-
6   * IntroGUI004
7   * Draws 300 random lines in random colors inside the frame.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class IntroGUI004 extends JFrame {
13
14     private Random gen = new Random();
15     private static IntroGUI004 prog;
16
17     public static void main(String[] args) {
18         prog = new IntroGUI004(args);
19         prog.setTitle("GUI");
20
21         prog.setBounds(20, 20, 700, 500);
22         prog.setVisible(true);
23         prog.toFront();
24     }
25
26     public IntroGUI004(String[] args) {
27     }
28
29     public void paint(Graphics g) {
30         super.paint(g);
31
32         Rectangle bounds = prog.getBounds();
33         int x = (int) bounds.getWidth();
34         int y = (int) bounds.getHeight();
35
36         for (int i = 0; i < 300; i++) {
37             g.setColor(new Color(gen.nextInt(255), gen.nextInt(255), gen.nextInt(255)));
38             g.drawLine(gen.nextInt(x), gen.nextInt(y), gen.nextInt(x), gen.nextInt(y));
39         }
40     }
41 }
```

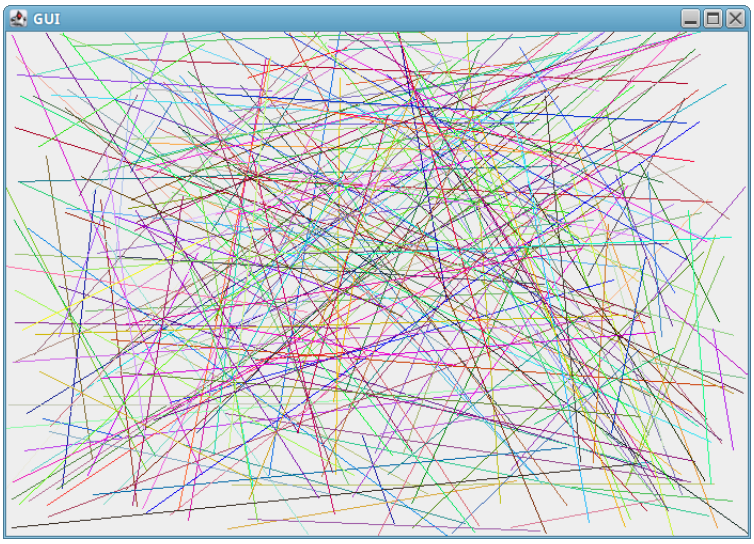


Figure 15.4: IntroGUI004.java Output

15.6 A Little More Geometry

We have seen the `fillOval` method in the previous examples, the `drawOval` method is the same except that it just draws the edge and does not fill it in. The same is true for the `fillRect` and `drawRect` methods, as with the oval, the parameters are the coordinates to the upper left corner, followed by the width and height of the rectangle.

Notice that the oval in this example is cut off. This is because the coordinates that we are specifying are pixel positions on the frame, which starts in the upper left corner of the frame, under the title bar. You may have noticed the lines in the previous example looked like they were going behind the title bar as well. So when we draw objects to the frame we need to make sure that we stay out of the title bar. This is not the best way to go since a title bar can be different in size depending on your platform. It is better to draw on a panel (called a JPanel) and then put the panel on the frame. We do this in the next example.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI005
6   * Draws some geometric objects on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI005 extends JFrame {
12
13     private static IntroGUI005 prog;
14
```

```
15      public static void main(String[] args) {
16          prog = new IntroGUI005(args);
17          prog.setTitle("GUI");
18
19          prog.setBounds(20, 20, 700, 500);
20          prog.setVisible(true);
21          prog.toFront();
22      }
23
24      public IntroGUI005(String[] args) {
25      }
26
27      public void paint(Graphics g) {
28          super.paint(g);
29
30          g.setColor(Color.red);
31          g.fillOval(70, 50, 200, 50);
32          g.setColor(Color.green);
33          g.drawOval(10, 20, 100, 200);
34          g.drawRect(100, 200, 50, 50);
35          g.setColor(Color.blue);
36          g.fillRect(170, 200, 50, 50);
37
38          g.setColor(Color.red);
39          g.fillRect(170, 130, 50, 50);
40          g.setColor(Color.black);
41          g.drawRect(170, 130, 50, 50);
42      }
43  }
```

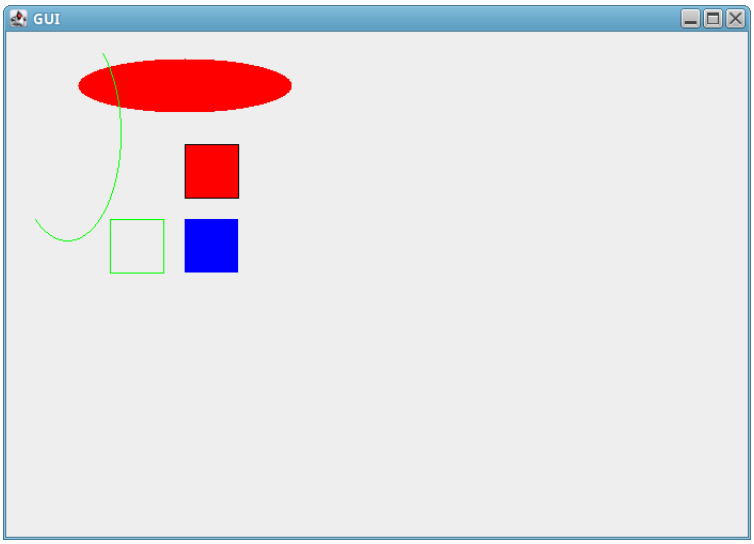


Figure 15.5: IntroGUI005.java Output

15.7 Adding a JPanel

This example gets around the positioning problem we had in the previous example. We do exactly the same drawing commands but this time we do them on a JPanel (in its paint method) and place the panel on the frame. Now the $(0, 0)$ pixel position is in the upper left corner of the visible area, hence no clipping of the oval.

Our JPanel is called GraphicsJPanel and it extends the Java JPanel, so our panel is a JPanel that overrides the paint method. Note that the code of the panel paint method is exactly the same as the one we had in the frame in the previous example. The only other addition is the line `setBackground(Color.white);` in the constructor of our GraphicsJPanel, which as the name suggests, sets the background color of the panel to white.

In the main program, the frame, we create a new GraphicsJPanel by

```
canvas = new GraphicsJPanel();
```

and then we place it on the frame with,

```
getContentPane().setLayout(new BorderLayout());
getContentPane().add(canvas, BorderLayout.CENTER);
```

The ContentPane is the viewing area of the frame and `getContentPane()` gives you access to it. Setting the layout to a BorderLayout creates 5 areas to the pane, North, South, East, West, and Center. If any of these are not filled the area collapses to nothing.

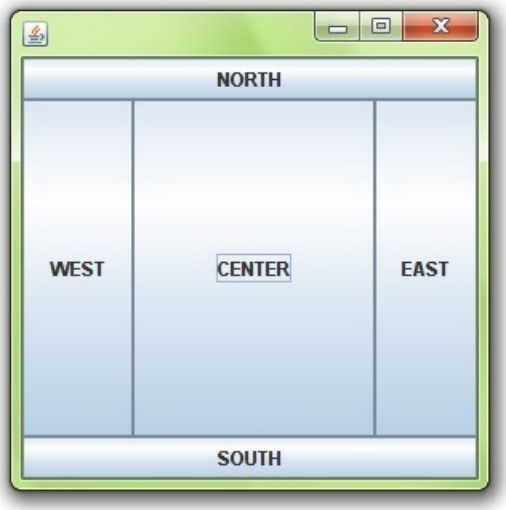


Figure 15.6: Border Layout

So a nice way to put a panel on the entire pane is to place it in the center of a

BorderLayout, which is what the last line of code does. There are lots of other layouts and ways to position controls, we will see some of these in the next chapter.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Draws some geometric objects on the panel.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.red);
21         g.fillOval(70, 50, 200, 50);
22         g.setColor(Color.green);
23         g.drawOval(10, 20, 100, 200);
24         g.drawRect(100, 200, 50, 50);
25         g.setColor(Color.blue);
26         g.fillRect(170, 200, 50, 50);
27
28         g.setColor(Color.red);
29         g.fillRect(170, 130, 50, 50);
30         g.setColor(Color.black);
31         g.drawRect(170, 130, 50, 50);
32     }
33 }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI006
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI006 extends JFrame {
12
13     private static IntroGUI006 prog;
14     private GraphicsJPanel canvas;
15
16     public static void main(String[] args) {
17         prog = new IntroGUI006(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
24
25     public IntroGUI006(String[] args) {
26         canvas = new GraphicsJPanel();
27
```

```
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

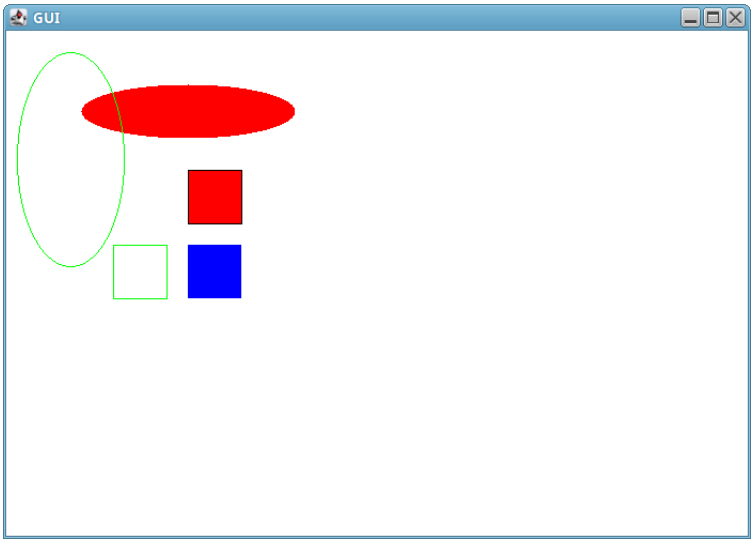


Figure 15.7: IntroGUI006.java Output

15.8 Clearing a Rectangle

This example simply shows the use of the `clearRect` method, as with the rectangle drawing methods the parameters are the coordinates to the upper left corner followed by the width and height of the rectangle.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the clear rectangle method.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.red);
21         g.fillRect(50, 70, 250, 350);
```

```
22          g.clearRect(100, 100, 100, 100);
23      }
24  }
```

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   /*-
5    * IntroGUI007
6    * Creates a special JPanel for graphing and places the panel on the frame.
7    * Author:  Don Spickler
8    * Date: 7/6/2016
9    */
10
11  public class IntroGUI007 extends JFrame {
12
13      private static IntroGUI007 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI007(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI007(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

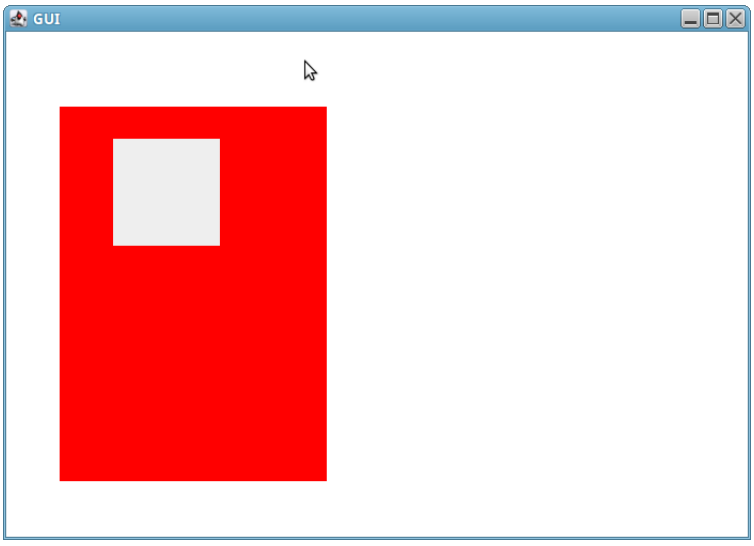


Figure 15.8: IntroGUI007.java Output

15.9 Polygons

To draw a polygon we first set up two arrays of integers one for the x-values and one for the y-values. So in this example the points we will be connecting into a polygon are $(20, 100)$, $(120, 200)$, $(290, 20)$, $(20, 50)$, and $(80, 210)$. Once the arrays are set up we draw the polygon by sending the x and y arrays and the number of points to draw. Note that the `drawPolygon` will close the loop by linking the last point up to the first point.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the drawPolygon method.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         int [] xvalues = new int [5];
21         int [] yvalues = new int [5];
22
23         xvalues[0] = 20;
24         xvalues[1] = 120;
25         xvalues[2] = 290;
26         xvalues[3] = 20;
27         xvalues[4] = 80;
28
29         yvalues[0] = 100;
30         yvalues[1] = 200;
31         yvalues[2] = 20;
32         yvalues[3] = 50;
33         yvalues[4] = 210;
34
35         g.setColor(Color.red);
36         g.drawPolygon(xvalues, yvalues, 5);
37     }
38 }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI008
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI008 extends JFrame {
12
```

```
13      private static IntroGUI008 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI008(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI008(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

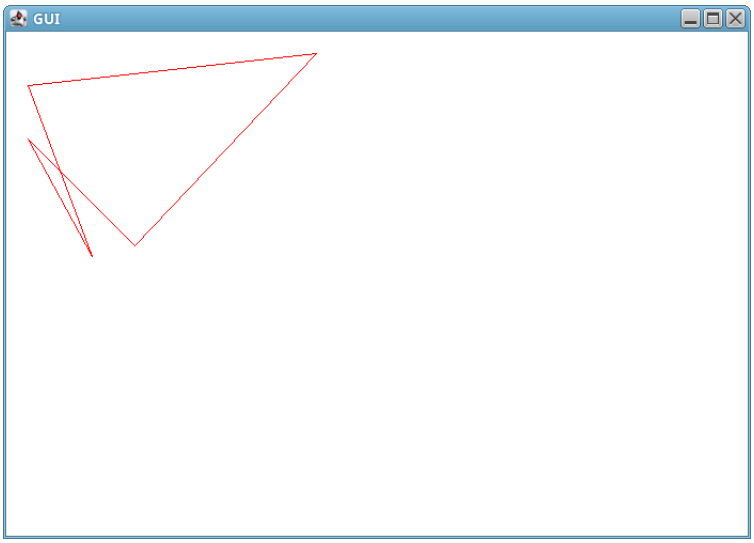


Figure 15.9: IntroGUI008.java Output

15.10 Filling Polygons

This is the same as the last example except that we use `fillPolygon` in place of `drawPolygon`.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the fillPolygon method.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
```

```
 */

public class GraphicsJPanel extends JPanel {

    public GraphicsJPanel() {
        setBackground(Color.white);
    }

    public void paint(Graphics g) {
        super.paint(g);

        int [] xvalues = new int [5];
        int [] yvalues = new int [5];

        xvalues[0] = 20;
        xvalues[1] = 120;
        xvalues[2] = 290;
        xvalues[3] = 20;
        xvalues[4] = 80;

        yvalues[0] = 100;
        yvalues[1] = 200;
        yvalues[2] = 20;
        yvalues[3] = 50;
        yvalues[4] = 210;

        g.setColor(Color.red);
        g.fillPolygon(xvalues, yvalues, 5);
    }
}
```

```
import java.awt.*;
import javax.swing.*;

/*-
 * IntroGUI009
 * Creates a special JPanel for graphing and places the panel on the frame.
 * Author:  Don Spickler
 * Date: 7/6/2016
 */

public class IntroGUI009 extends JFrame {

    private static IntroGUI009 prog;
    private GraphicsJPanel canvas;

    public static void main(String[] args) {
        prog = new IntroGUI009(args);
        prog.setTitle("GUI");

        prog.setBounds(20, 20, 700, 500);
        prog.setVisible(true);
        prog.toFront();
    }

    public IntroGUI009(String[] args) {
        canvas = new GraphicsJPanel();

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
    }
}
```

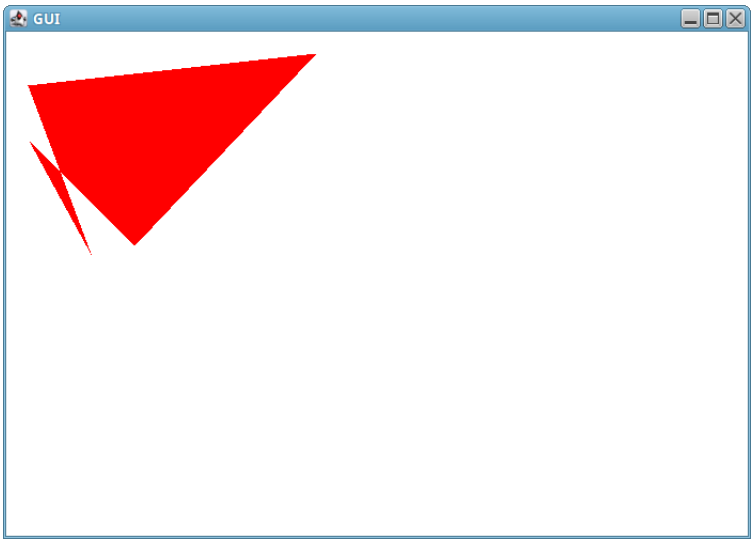


Figure 15.10: IntroGUI009.java Output

15.11 Drawing Text

This example shows some basic manipulation and plotting of text. The `drawString` method takes in three parameters, the first is the string to be plotted, the next two are the (x, y) pixel position of the lower left corner to the box that the string is drawn, that is, the baseline of the leftmost character. The text is drawn in the current color and the currently selected font. So the first string is in the default font. To change the font we use the `setFont` method, so

```
g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 50));
```

changes the font to a sans serif type font that is bold and 50 pixels in height and the line

```
g.setFont(new Font(Font.SANS_SERIF, Font.BOLD | Font.ITALIC,
    70));
```

sets the font, again to sans serif, that is both bold and italic, and is 70 pixels in height.

The Font class has several types of faces, like sans serif, built into it and you can use the other fonts that are installed on your computer but we will not do that here. The font style can be plain, bold, italic, or underlined or any combination of these. If you want to use multiple styles just put a vertical bar between them. The vertical bar is a bit-wise OR, it will tell Java to use both styles. This type of option selection is very common, especially in C and C++.

```
1 import java.awt.*;
```

```
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the process of selecting fonts and drawing text to a graphics panel.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.red);
21         g.drawString("Hi there", 100, 200);
22
23         g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 50));
24         g.drawString("Big Font", 50, 70);
25         g.setFont(new Font(Font.SANS_SERIF, Font.BOLD | Font.ITALIC, 70));
26         g.drawString("Big Slant Font", 50, 170);
27     }
28 }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI010
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI010 extends JFrame {
12
13     private static IntroGUI010 prog;
14     private GraphicsJPanel canvas;
15
16     public static void main(String[] args) {
17         prog = new IntroGUI010(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
24
25     public IntroGUI010(String[] args) {
26         canvas = new GraphicsJPanel();
27
28         getContentPane().setLayout(new BorderLayout());
29         getContentPane().add(canvas, BorderLayout.CENTER);
30     }
31 }
```

Figure 15.11: IntroGUI010.java Output

15.12 Drawing Text and Objects

This is simply another example of drawing text and geometrical objects.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the process of selecting fonts and drawing text to a graphics panel.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.black);
21         g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 30));
22         g.drawString("Green Triangle", 50, 70);
23
24         int [] trixvalues = new int [3];
25         int [] triyvalues = new int [3];
26
27         trixvalues[0] = 20;
28         trixvalues[1] = 120;
29         trixvalues[2] = 290;
30
31         triyvalues[0] = 100;
32         triyvalues[1] = 200;
33         triyvalues[2] = 110;
```

```
34
35          g.setColor(Color.green);
36          g.fillPolygon(trixvalues, triyvalues, 3);
37
38          g.setColor(Color.black);
39          g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 30));
40          g.drawString("Red Square", 350, 70);
41          g.setColor(Color.red);
42          g.fillRect(360, 100, 150, 150);
43      }
44  }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI011
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI011 extends JFrame {
12
13     private static IntroGUI011 prog;
14     private GraphicsJPanel canvas;
15
16     public static void main(String[] args) {
17         prog = new IntroGUI011(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
24
25     public IntroGUI011(String[] args) {
26         canvas = new GraphicsJPanel();
27
28         getContentPane().setLayout(new BorderLayout());
29         getContentPane().add(canvas, BorderLayout.CENTER);
30     }
31 }
```

15.13 More About Color

As we saw above, the syntax to create a custom color was `new Color(r, g, b)` where r, g, and b are integers between 0 and 255. The value 0 means no color of that component and 255 means full intensity of that color component. We can also use floats in place of the integers. If we use floats then the range is from 0.0 to 1.0, 0 means no color of that component and 1 means full intensity of that color component. This style is more in line with graphics programming interfaces like OpenGL. So $(1, 0, 0)$ is bright red, $(0.5, 0.5, 0.5)$ is gray, $(1, 1, 0)$ is bright yellow, and $(1, 1, 1)$ is bright white. Notice that we use the `f` type caster so that Java sees these as floats and not integers or doubles.

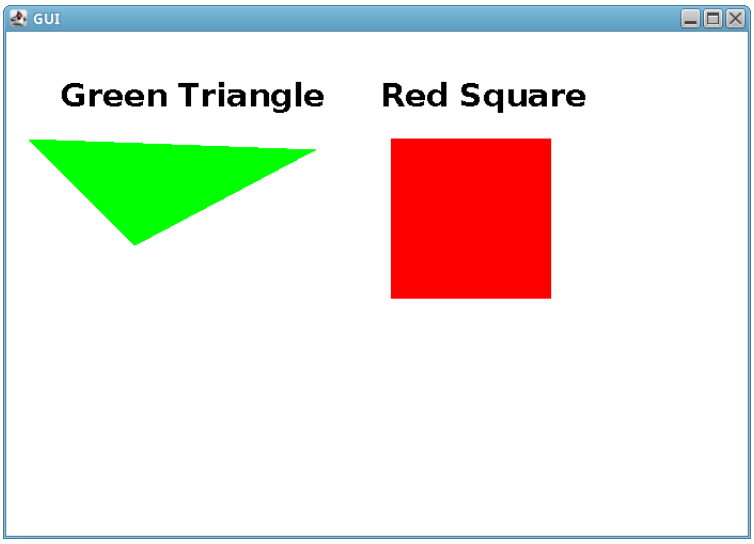


Figure 15.12: IntroGUI011.java Output

With the custom color creation we can add a fourth component, called the alpha channel. This controls transparency of the color. Note in the bottom two squares that the lines are showing through, this is because we let the color be semi-transparent. In the alpha channel, 0.0 (or 0) means completely transparent and 1.0 (or 255) means completely opaque. Finally, the bottom rectangles show how to use transparency to create a gradient.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows more options for color and the use of transparency.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.black);
21         for (int i = 0; i < 120; i += 10)
22             g.drawLine(i, 0, i, 300);
23
24         g.setColor(Color.red);
25         g.fillRect(0, 0, 50, 50);
26         g.setColor(Color.green);
27         g.fillRect(60, 0, 50, 50);
28
29         g.setColor(Color.blue);
```

```
30          g.fillRect(0, 60, 50, 50);
31          g.setColor(Color.magenta);
32          g.fillRect(60, 60, 50, 50);
33
34          g.setColor(new Color(0, 255, 0));
35          g.fillRect(0, 120, 50, 50);
36          g.setColor(new Color(0, 255, 255));
37          g.fillRect(60, 120, 50, 50);
38
39          g.setColor(new Color(1f, 0f, 0f));
40          g.fillRect(0, 180, 50, 50);
41          g.setColor(new Color(1f, 0.5f, 0f));
42          g.fillRect(60, 180, 50, 50);
43
44          g.setColor(new Color(1f, 0f, 0f, 0.5f));
45          g.fillRect(0, 240, 50, 50);
46          g.setColor(new Color(255, 100, 0, 130));
47          g.fillRect(60, 240, 50, 50);
48
49          for (int i = 0; i < 120; i += 10) {
50              g.setColor(new Color(1f, 0f, 0f, i / 120f));
51              g.fillRect(i, 300, 50, 50);
52          }
53
54          for (int i = 0; i < 120; i++) {
55              g.setColor(new Color(1f, 0f, 0f, i / 120f));
56              g.drawLine(i, 360, i, 400);
57          }
58      }
59  }
```

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   /*-
5    * IntroGUI012
6    * Creates a special JPanel for graphing and places the panel on the frame.
7    * Author:  Don Spickler
8    * Date: 7/6/2016
9    */
10
11  public class IntroGUI012 extends JFrame {
12
13      private static IntroGUI012 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI012(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI012(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

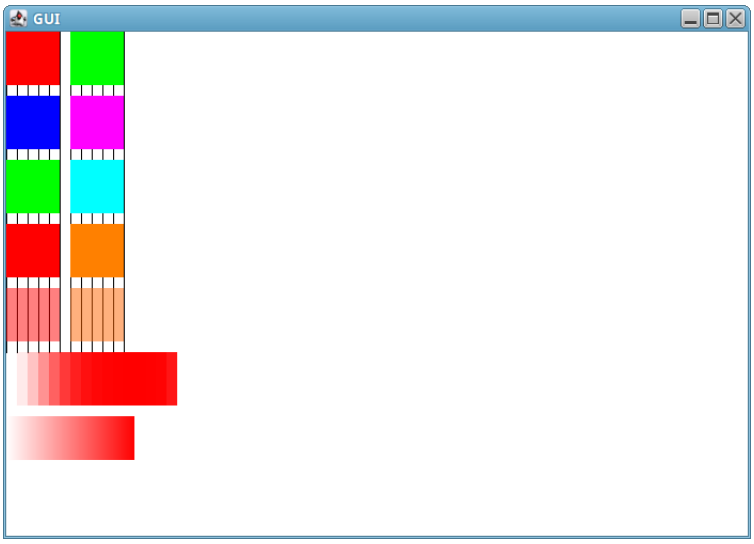


Figure 15.13: IntroGUI012.java Output

15.14 Translation

The $(0, 0)$ pixel position is by default in the upper left corner of the panel. You can move it using the `translate` method, which translates the origin to the specified pixel position. So in this example, the origin has been translated to the pixel position $(200, 200)$.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Shows the translate method.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.translate(200, 200);
21         g.drawLine(0, 200, 0, -200);
22         g.drawLine(-200, 0, 200, 0);
23     }
24 }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
```

```
4  /*-
5   * IntroGUI013
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI013 extends JFrame {
12
13     private static IntroGUI013 prog;
14     private GraphicsJPanel canvas;
15
16     public static void main(String[] args) {
17         prog = new IntroGUI013(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
24
25     public IntroGUI013(String[] args) {
26         canvas = new GraphicsJPanel();
27
28         getContentPane().setLayout(new BorderLayout());
29         getContentPane().add(canvas, BorderLayout.CENTER);
30     }
31 }
```

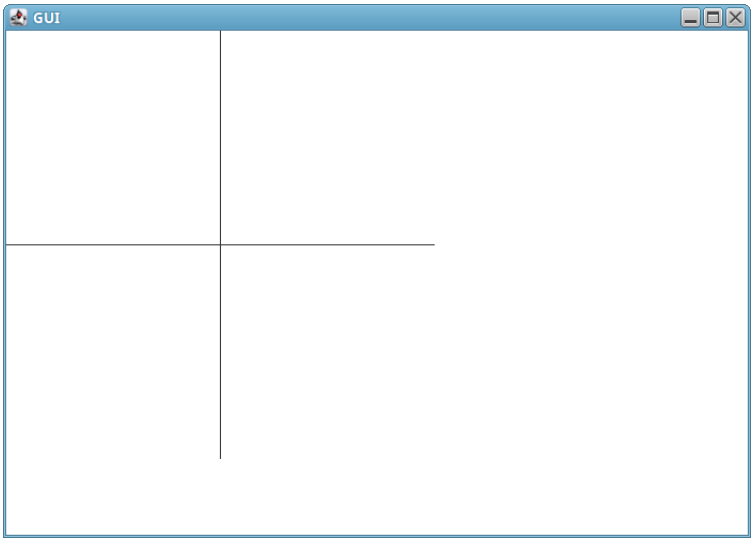


Figure 15.14: IntroGUI013.java Output

15.15 Plotting a Function

This example updates the last example by adding in the plot of the function $f(x) = \sin(x)$ to the axes.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Plots the function y = sin(x).
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.black);
21         g.translate(200, 200);
22         g.drawLine(0, 200, 0, -200);
23         g.drawLine(-200, 0, 200, 0);
24
25         g.setColor(Color.red);
26
27         double lastx = -100;
28         double lasty = -100;
29
30         for (double x = -10; x <= 10; x += 0.1) {
31             double y = Math.sin(x);
32             if (lastx > -90) {
33                 g.drawLine((int) (20 * lastx), (int) (20 * lasty), (int) (20 * x), (int)
                       (20 * y));
34             }
35             lastx = x;
36             lasty = y;
37         }
38     }
39 }
```

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * IntroGUI014
6   * Creates a special JPanel for graphing and places the panel on the frame.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class IntroGUI014 extends JFrame {
12
13     private static IntroGUI014 prog;
14     private GraphicsJPanel canvas;
15
16     public static void main(String[] args) {
17         prog = new IntroGUI014(args);
18         prog.setTitle("GUI");
19
20         prog.setBounds(20, 20, 700, 500);
21         prog.setVisible(true);
22         prog.toFront();
23     }
```

```
24
25      public IntroGUI014(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

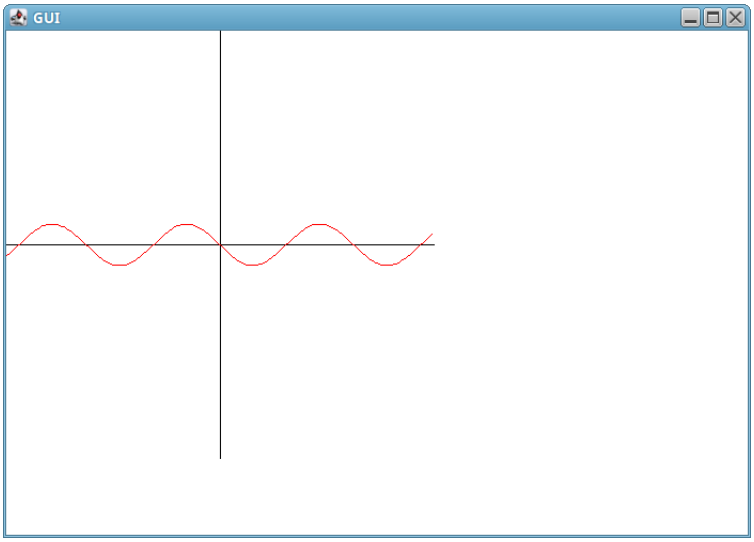


Figure 15.15: IntroGUI014.java Output

15.16 Plotting a Hexagon

This example plots a hexagon on the coordinate axes.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /*-
5   * GraphicsJPanel
6   * Plots a hexagon.
7   * Author:  Don Spickler
8   * Date: 7/6/2016
9   */
10
11 public class GraphicsJPanel extends JPanel {
12
13     public GraphicsJPanel() {
14         setBackground(Color.white);
15     }
16
17     public void paint(Graphics g) {
18         super.paint(g);
19
20         g.setColor(Color.black);
21         g.translate(200, 200);
```

```
22          g.drawLine(0, 200, 0, -200);
23          g.drawLine(-200, 0, 200, 0);
24
25          g.setColor(Color.red);
26
27          for (int i = 0; i <= 6; i++) {
28              double x = Math.cos(2 * Math.PI * ((double) i / 6));
29              double y = Math.sin(2 * Math.PI * ((double) i / 6));
30              double nextx = Math.cos(2 * Math.PI * ((double) (i + 1) / 6));
31              double nexty = Math.sin(2 * Math.PI * ((double) (i + 1) / 6));
32              g.drawLine((int) (100 * x), (int) (100 * y), (int) (100 * nextx), (int) (100 *
                    nexty));
33          }
34      }
35  }
```

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   /*-
5    * IntroGUI015
6    * Creates a special JPanel for graphing and places the panel on the frame.
7    * Author:  Don Spickler
8    * Date: 7/6/2016
9    */
10
11  public class IntroGUI015 extends JFrame {
12
13      private static IntroGUI015 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI015(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI015(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

15.17 Freehand Drawing Example

This example allows the user to draw freehand on the panel using their mouse. When the user clicks and drags the mouse on the panel a black line will trace out the path that is made. To implement this we first need to link up the mouse to the program, specifically the panel. This is similar to the way we hooked up the keyboard to a program so that we could take keyboard input. We will not be using a Scanner as we did with the keyboard, we will be using listeners, specifically the MouseListener and

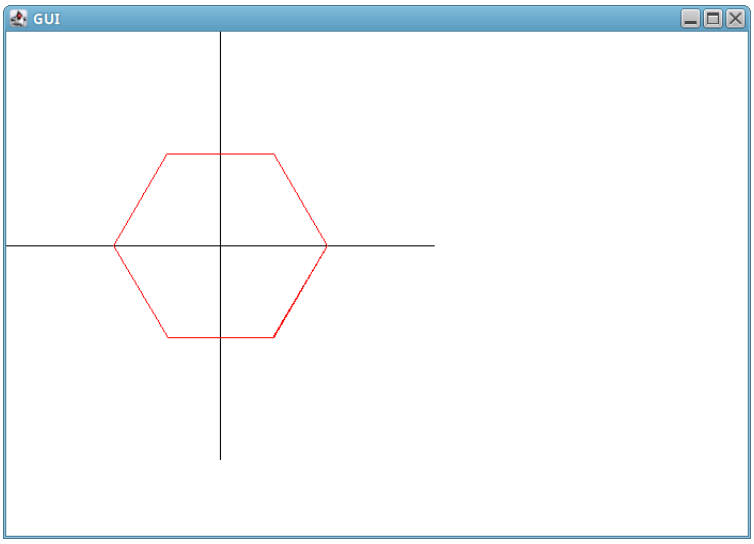


Figure 15.16: IntroGUI015.java Output

the MouseMotionListener. In Java Swing, a listener listens for changes that are made with input devises. So the mouse listener detects buttons being pressed or released and the mouse motion listener detects movement and dragging of the mouse. When a change is made to an input device the system *fires an event*, these events are picked up by the listener and allow the programmer run code when an event occurs.

There are three things you need to do to link the listener up to your program, specifically the panel.

1. Add the listeners to the class header with an implements command, for example, `implements MouseListener, MouseMotionListener`. Multiple listeners can be added by placing a comma between them in the list.

2. Add the listener to the panel in the constructor, for example,

   ```
   addMouseListener(this);
   addMouseMotionListener(this);
   ```

   The `this` simply refers to the panel.

3. Override the required methods for each listener you include, even if they are empty. For example, if you include the MouseListener and the MouseMotionListener you need to include the methods, mousePressed, mouseDragged, mouseEntered, mouseExited, mouseClicked, mouseMoved, and mouseReleased. As you can see in the example code we only implemented mousePressed and mouseDragged, the rest are empty. Note that all of these bring in one parameter, a MouseEvent, which holds information about the event.

The mousePressed event is fired if any of the mouse buttons are pressed, that is, just pressed down. The mouseReleased event is when a button is let up and the mouseClicked event is fired when the button is both pressed and released. The mouseEntered event is fired when the mouse enters the control, that is, it enters the panel from a position outside the panel. The mouseExited event is fired when the mouse pointer leaves the control. The mouseMoved event is fired anytime the mouse is moved, so this event is fired hundreds of times during a simple move of the mouse. The mouseDragged event is fired when the mouse is moved and a button is down.

In this example, we have two integer variables, `prevX` and `prevY` that track the previous mouse position. So when the mouse button is pressed we get the position of the mouse using the `getX()` and `getY()` methods from the mouse event. If the mouse is moved while the button is down then the mouseDragged event is fired, in which case we get the current position of the mouse, draw a line from the previous position to the current position and then set the previous position to the current position. In all, this creates a lot of small straight lines on the panel that track the mouse positions and draw out the path made by the mouse.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * GraphicsJPanel
7   * Freehand drawing panel.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class GraphicsJPanel extends JPanel implements MouseListener, MouseMotionListener {
13
14     private int prevX, prevY; // The previous location of the mouse.
15
16     public GraphicsJPanel() {
17         setBackground(Color.white);
18         addMouseListener(this);
19         addMouseMotionListener(this);
20     }
21
22     public void mousePressed(MouseEvent evt) {
23         prevX = evt.getX(); // x-coordinate where the user clicked.
24         prevY = evt.getY(); // y-coordinate where the user clicked.
25     }
26
27     public void mouseDragged(MouseEvent evt) {
28         int x = evt.getX(); // x-coordinate of mouse.
29         int y = evt.getY(); // y-coordinate of mouse.
30
31         Graphics g = getGraphics();
32         g.drawLine(prevX, prevY, x, y);
33
34         // Get ready for the next line segment in the curve.
35         prevX = x;
36         prevY = y;
37     }
38
39     // Some empty routines. These are required by the MouseListener
```

```
40      // and MouseMotionListener interfaces.
41      public void mouseEntered(MouseEvent evt) {
42      }
43
44      public void mouseExited(MouseEvent evt) {
45      }
46
47      public void mouseClicked(MouseEvent evt) {
48      }
49
50      public void mouseMoved(MouseEvent evt) {
51      }
52
53      public void mouseReleased(MouseEvent evt) {
54      }
55  }
```

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   /*-
5    * IntroGUI016
6    * Creates a special JPanel for graphing and places the panel on the frame.
7    * Author:  Don Spickler
8    * Date: 7/6/2016
9    */
10
11  public class IntroGUI016 extends JFrame {
12
13      private static IntroGUI016 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI016(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI016(String[] args) {
26          canvas = new GraphicsJPanel();
27
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

15.18 Freehand Drawing Example #2

In the previous example, you could draw on the panel with any of the mouse buttons. In this example, we restrict the drawing to be done only if the left button is depressed and we set up a click of the right mouse button to clear the drawing area. In the mousePressed event we check to see if button number 1 (the left one) was the one that was pressed, if so we set the dragging boolean flag to true. Then in the mouseDragged event if dragging is false we exit the event and if it is true we continue to the drawing

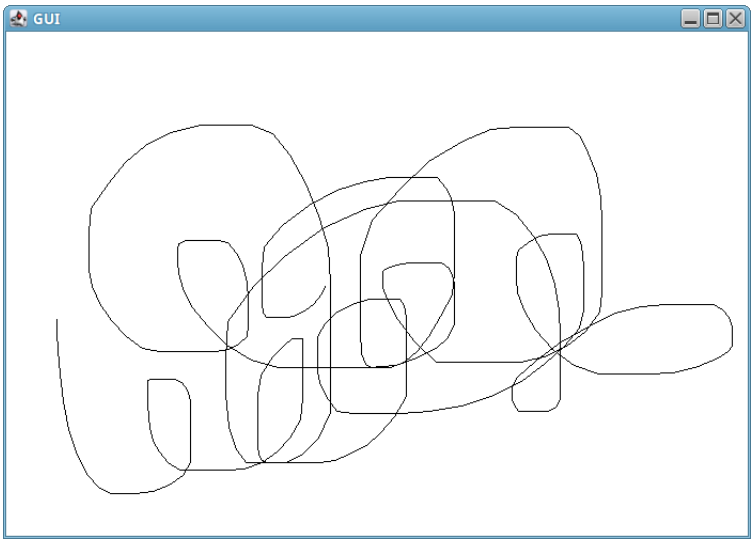


Figure 15.17: IntroGUI016.java Output

code. In the mouseClicked event we check if the button was number 3 (the right one) and if so we draw a filled rectangle over the entire panel, erasing the current contents. Also note that we need to implement the mouseReleased event to stop the dragging.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * GraphicsJPanel
7   * Freehand drawing panel.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class GraphicsJPanel extends JPanel implements MouseListener, MouseMotionListener {
13
14     private int prevX, prevY; // The previous location of the mouse.
15     private boolean dragging = false;
16
17     public GraphicsJPanel() {
18         setBackground(Color.white);
19         addMouseListener(this);
20         addMouseMotionListener(this);
21     }
22
23     public void mousePressed(MouseEvent evt) {
24         //  Only draw if the left mouse button was depressed.
25         if (evt.getButton() == MouseEvent.BUTTON1)
26             dragging = true;
27
28         prevX = evt.getX(); // x-coordinate where the user clicked.
29         prevY = evt.getY(); // y-coordinate where the user clicked.
30     }
31
32     public void mouseDragged(MouseEvent evt) {
33         if (!dragging)
```

```
34              return;
35
36          int x = evt.getX(); // x-coordinate of mouse.
37          int y = evt.getY(); // y-coordinate of mouse.
38
39          Graphics g = getGraphics();
40          g.drawLine(prevX, prevY, x, y);
41
42          // Get ready for the next line segment in the curve.
43          prevX = x;
44          prevY = y;
45      }
46
47      public void mouseEntered(MouseEvent evt) {
48      }
49
50      public void mouseExited(MouseEvent evt) {
51      }
52
53      public void mouseClicked(MouseEvent evt) {
54          // If right mouse button clicked, clear the screen
55          if (evt.getButton() == MouseEvent.BUTTON3) {
56              Graphics g = getGraphics();
57
58              g.setColor(Color.WHITE);
59              g.fillRect(0, 0, getWidth(), getHeight());
60          }
61      }
62
63      public void mouseMoved(MouseEvent evt) {
64      }
65
66      public void mouseReleased(MouseEvent evt) {
67          dragging = false;
68      }
69  }
```

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   /*-
5    * IntroGUI017
6    * Creates a special JPanel for graphing and places the panel on the frame.
7    * Author:  Don Spickler
8    * Date: 7/6/2016
9    */
10
11  public class IntroGUI017 extends JFrame {
12
13      private static IntroGUI017 prog;
14      private GraphicsJPanel canvas;
15
16      public static void main(String[] args) {
17          prog = new IntroGUI017(args);
18          prog.setTitle("GUI");
19
20          prog.setBounds(20, 20, 700, 500);
21          prog.setVisible(true);
22          prog.toFront();
23      }
24
25      public IntroGUI017(String[] args) {
26          canvas = new GraphicsJPanel();
27
```

```
28          getContentPane().setLayout(new BorderLayout());
29          getContentPane().add(canvas, BorderLayout.CENTER);
30      }
31  }
```

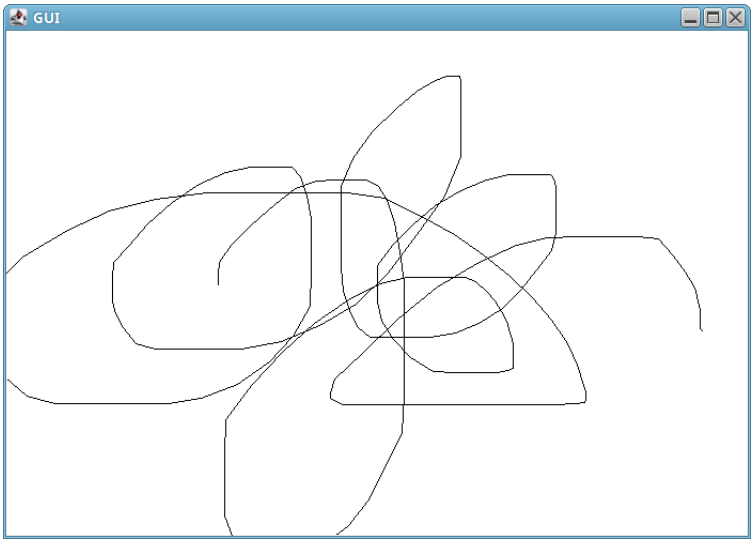


Figure 15.18: IntroGUI017.java Output

15.19 Freehand Drawing Example with Selections

This example is an extension of the previous example. We place selection boxes on the right that when clicked in change the drawing color. In addition, the Clr box will clear the drawing area. We also restrict the drawing area so that the user does not draw into the menu of colored boxes. To make the selection area we simply added colored squares to the graphics area and updated the mouseClicked method to track where the current position of the click was, if it is in a selection box we change the color of the drawColor variable.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * GraphicsJPanel
7   * Freehand drawing panel with color selection and restricted drawing area.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class GraphicsJPanel extends JPanel implements MouseListener, MouseMotionListener {
13
14     private int prevX, prevY; // The previous location of the mouse.
15     private boolean dragging = false;
```

```
16      private Color drawColor;
17      private int drawWidth = 500;
18      private int drawHeight = 400;
19
20      public GraphicsJPanel() {
21          setBackground(Color.white);
22          addMouseListener(this);
23          addMouseMotionListener(this);
24      }
25
26      public void paint(Graphics g) {
27          super.paint(g);
28
29          g.setColor(Color.WHITE);
30          g.fillRect(0, 0, drawWidth, drawHeight);
31          g.setColor(Color.BLACK);
32          g.drawRect(0, 0, drawWidth, drawHeight);
33
34          g.setColor(Color.BLACK);
35          g.drawRect(0, 0, drawWidth - 50, drawHeight);
36
37          g.setColor(Color.BLACK);
38          g.fillRect(drawWidth - 50, 0, 50, 50);
39          g.setColor(Color.BLACK);
40          g.drawRect(drawWidth - 50, 0, 50, 50);
41
42          g.setColor(Color.RED);
43          g.fillRect(drawWidth - 50, 50, 50, 50);
44          g.setColor(Color.BLACK);
45          g.drawRect(drawWidth - 50, 50, 50, 50);
46
47          g.setColor(Color.GREEN);
48          g.fillRect(drawWidth - 50, 100, 50, 50);
49          g.setColor(Color.BLACK);
50          g.drawRect(drawWidth - 50, 100, 50, 50);
51
52          g.setColor(Color.BLUE);
53          g.fillRect(drawWidth - 50, 150, 50, 50);
54          g.setColor(Color.BLACK);
55          g.drawRect(drawWidth - 50, 150, 50, 50);
56
57          g.setColor(Color.YELLOW);
58          g.fillRect(drawWidth - 50, 200, 50, 50);
59          g.setColor(Color.BLACK);
60          g.drawRect(drawWidth - 50, 200, 50, 50);
61
62          g.setColor(Color.GRAY);
63          g.fillRect(drawWidth - 50, 250, 50, 50);
64          g.setColor(Color.BLACK);
65          g.drawRect(drawWidth - 50, 250, 50, 50);
66
67          g.drawRect(drawWidth - 50, 300, 50, 50);
68          g.setFont(new Font(Font.SANS_SERIF, Font.PLAIN, 20));
69          g.drawString("Clr", drawWidth - 40, 335);
70
71          drawColor = Color.BLACK;
72      }
73
74      public void mousePressed(MouseEvent evt) {
75          // Only draw if the left mouse button was depressed.
76          if (evt.getButton() == MouseEvent.BUTTON1)
77              dragging = true;
78
79          prevX = evt.getX(); // x-coordinate where the user clicked.
```

```
80          prevY = evt.getY(); // y-coordinate where the user clicked.
81      }
82
83      public void mouseDragged(MouseEvent evt) {
84          if (!dragging)
85              return;
86
87          int x = evt.getX(); // x-coordinate of mouse.
88          int y = evt.getY(); // y=coordinate of mouse.
89
90          Graphics g = getGraphics();
91          g.setColor(drawColor);
92
93          // Make sure that we are in the drawing region and then draw the line.
94          if ((x > 0) && (x < drawWidth - 50) && (y > 0) && (y < drawHeight) && (prevX > 0)
                && (prevX < drawWidth - 50)
95                  && (prevY > 0) && (prevY < drawHeight))
96              g.drawLine(prevX, prevY, x, y);
97
98          // Get ready for the next line segment in the curve.
99          prevX = x;
100         prevY = y;
101     }
102
103     public void mouseReleased(MouseEvent evt) {
104         dragging = false;
105     }
106
107     public void mouseClicked(MouseEvent evt) {
108         // If right mouse button clicked, clear the screen.
109         if (evt.getButton() == MouseEvent.BUTTON3) {
110             Graphics g = getGraphics();
111
112             g.setColor(Color.WHITE);
113             g.fillRect(0, 0, drawWidth - 50, drawHeight);
114             g.setColor(Color.BLACK);
115             g.drawRect(0, 0, drawWidth - 50, drawHeight);
116         }
117
118         // If left mouse button clicked, change the color.
119         if (evt.getButton() == MouseEvent.BUTTON1) {
120             int x = evt.getX(); // x-coordinate of mouse.
121             int y = evt.getY(); // y-coordinate of mouse.
122
123             if (x > drawWidth - 50 && x < drawWidth) {
124                 if (y > 0 && y < 50)
125                     drawColor = Color.BLACK;
126                 else if (y > 50 && y < 100)
127                     drawColor = Color.RED;
128                 else if (y > 100 && y < 150)
129                     drawColor = Color.GREEN;
130                 else if (y > 150 && y < 200)
131                     drawColor = Color.BLUE;
132                 else if (y > 200 && y < 250)
133                     drawColor = Color.YELLOW;
134                 else if (y > 250 && y < 300)
135                     drawColor = Color.GRAY;
136                 else if (y > 300 && y < 350) {
137                     Graphics g = getGraphics();
138
139                     g.setColor(Color.WHITE);
140                     g.fillRect(0, 0, drawWidth - 50, drawHeight);
141                     g.setColor(Color.BLACK);
142                     g.drawRect(0, 0, drawWidth - 50, drawHeight);
```

```
                }
            }
        }
    }

    // Some empty routines. These are required by the MouseListener
    // and MouseMotionListener interfaces.
    public void mouseEntered(MouseEvent evt) {
    }

    public void mouseExited(MouseEvent evt) {
    }

    public void mouseMoved(MouseEvent evt) {
    }
}
```

```
import java.awt.*;
import javax.swing.*;

/*-
 * IntroGUI018
 * Creates a special JPanel for graphing and places the panel on the frame.
 * Author:  Don Spickler
 * Date: 7/6/2016
 */

public class IntroGUI018 extends JFrame {

    private static IntroGUI018 prog;
    private GraphicsJPanel canvas;

    public static void main(String[] args) {
        prog = new IntroGUI018(args);
        prog.setTitle("GUI");

        prog.setBounds(20, 20, 700, 500);
        prog.setVisible(true);
        prog.toFront();
    }

    public IntroGUI018(String[] args) {
        canvas = new GraphicsJPanel();

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
    }
}
```

15.20 Freehand Drawing Example with Selections and Fixed Size

We made only two changes from the last example to this one. In the last example the drawing area did not match the window size, which looked a bit strange. So we altered the size of the window and added the command,

```
prog.setResizable(false);
```

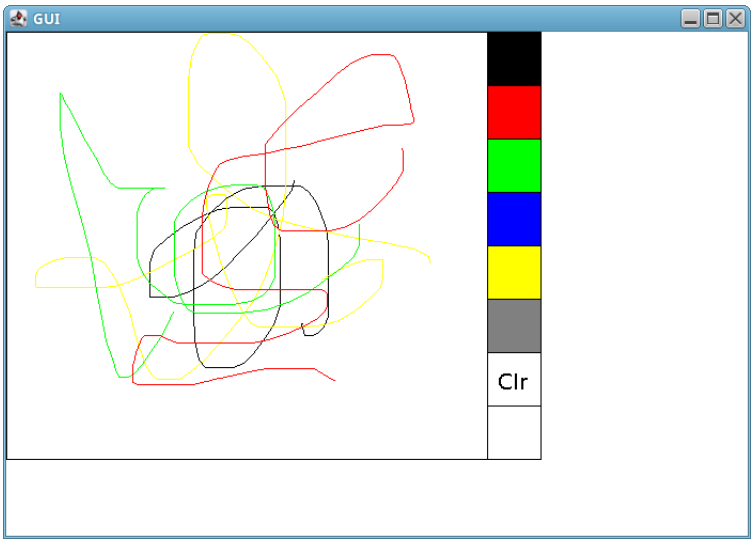


Figure 15.19: IntroGUI018.java Output

to the main, which made the window fixed in size.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * GraphicsJPanel
7   * Freehand drawing panel with color selection and restricted drawing area.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class GraphicsJPanel extends JPanel implements MouseListener, MouseMotionListener {
13
14     private int prevX, prevY; // The previous location of the mouse.
15     private boolean dragging = false;
16     private Color drawColor;
17     private int drawWidth = 500;
18     private int drawHeight = 400;
19
20     public GraphicsJPanel() {
21         setBackground(Color.white);
22         addMouseListener(this);
23         addMouseMotionListener(this);
24     }
25
26     public void paint(Graphics g) {
27         super.paint(g);
28
29         g.setColor(Color.WHITE);
30         g.fillRect(0, 0, drawWidth, drawHeight);
31         g.setColor(Color.BLACK);
32         g.drawRect(0, 0, drawWidth, drawHeight);
33
34         g.setColor(Color.BLACK);
35         g.drawRect(0, 0, drawWidth - 50, drawHeight);
36
```

```
37          g.setColor(Color.BLACK);
38          g.fillRect(drawWidth - 50, 0, 50, 50);
39          g.setColor(Color.BLACK);
40          g.drawRect(drawWidth - 50, 0, 50, 50);
41
42          g.setColor(Color.RED);
43          g.fillRect(drawWidth - 50, 50, 50, 50);
44          g.setColor(Color.BLACK);
45          g.drawRect(drawWidth - 50, 50, 50, 50);
46
47          g.setColor(Color.GREEN);
48          g.fillRect(drawWidth - 50, 100, 50, 50);
49          g.setColor(Color.BLACK);
50          g.drawRect(drawWidth - 50, 100, 50, 50);
51
52          g.setColor(Color.BLUE);
53          g.fillRect(drawWidth - 50, 150, 50, 50);
54          g.setColor(Color.BLACK);
55          g.drawRect(drawWidth - 50, 150, 50, 50);
56
57          g.setColor(Color.YELLOW);
58          g.fillRect(drawWidth - 50, 200, 50, 50);
59          g.setColor(Color.BLACK);
60          g.drawRect(drawWidth - 50, 200, 50, 50);
61
62          g.setColor(Color.GRAY);
63          g.fillRect(drawWidth - 50, 250, 50, 50);
64          g.setColor(Color.BLACK);
65          g.drawRect(drawWidth - 50, 250, 50, 50);
66
67          g.drawRect(drawWidth - 50, 300, 50, 50);
68          g.setFont(new Font(Font.SANS_SERIF, Font.PLAIN, 17));
69          g.drawString("Clear", drawWidth - 47, 333);
70
71          drawColor = Color.BLACK;
72      }
73
74      public void mousePressed(MouseEvent evt) {
75          // Only draw if the left mouse button was depressed.
76          if (evt.getButton() == MouseEvent.BUTTON1)
77              dragging = true;
78
79          prevX = evt.getX(); // x-coordinate where the user clicked.
80          prevY = evt.getY(); // y-coordinate where the user clicked.
81      }
82
83      public void mouseDragged(MouseEvent evt) {
84          if (!dragging)
85              return;
86
87          int x = evt.getX(); // x-coordinate of mouse.
88          int y = evt.getY(); // y=coordinate of mouse.
89
90          Graphics g = getGraphics();
91          g.setColor(drawColor);
92
93          // Make sure that we are in the drawing region and then draw the line.
94          if ((x > 0) && (x < drawWidth - 50) && (y > 0) && (y < drawHeight) && (prevX > 0)
                && (prevX < drawWidth - 50)
95                  && (prevY > 0) && (prevY < drawHeight))
96              g.drawLine(prevX, prevY, x, y);
97
98          // Get ready for the next line segment in the curve.
99          prevX = x;
```

```
100         prevY = y;
101     }
102
103     public void mouseReleased(MouseEvent evt) {
104         dragging = false;
105     }
106
107     public void mouseClicked(MouseEvent evt) {
108         // If right mouse button clicked, clear the screen.
109         if (evt.getButton() == MouseEvent.BUTTON3) {
110             Graphics g = getGraphics();
111
112             g.setColor(Color.WHITE);
113             g.fillRect(0, 0, drawWidth - 50, drawHeight);
114             g.setColor(Color.BLACK);
115             g.drawRect(0, 0, drawWidth - 50, drawHeight);
116         }
117
118         // If left mouse button clicked, change the color.
119         if (evt.getButton() == MouseEvent.BUTTON1) {
120             int x = evt.getX(); // x-coordinate of mouse.
121             int y = evt.getY(); // y-coordinate of mouse.
122
123             if (x > drawWidth - 50 && x < drawWidth) {
124                 if (y > 0 && y < 50)
125                     drawColor = Color.BLACK;
126                 else if (y > 50 && y < 100)
127                     drawColor = Color.RED;
128                 else if (y > 100 && y < 150)
129                     drawColor = Color.GREEN;
130                 else if (y > 150 && y < 200)
131                     drawColor = Color.BLUE;
132                 else if (y > 200 && y < 250)
133                     drawColor = Color.YELLOW;
134                 else if (y > 250 && y < 300)
135                     drawColor = Color.GRAY;
136                 else if (y > 300 && y < 350) {
137                     Graphics g = getGraphics();
138
139                     g.setColor(Color.WHITE);
140                     g.fillRect(0, 0, drawWidth - 50, drawHeight);
141                     g.setColor(Color.BLACK);
142                     g.drawRect(0, 0, drawWidth - 50, drawHeight);
143                 }
144             }
145         }
146     }
147
148     // Some empty routines. These are required by the MouseListener
149     // and MouseMotionListener interfaces.
150     public void mouseEntered(MouseEvent evt) {
151     }
152
153     public void mouseExited(MouseEvent evt) {
154     }
155
156     public void mouseMoved(MouseEvent evt) {
157     }
158 }
```

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 /*-
```

```
 * IntroGUI019
 * Creates a special JPanel for graphing and places the panel on the frame.
 * Author:  Don Spickler
 * Date: 7/6/2016
 */

public class IntroGUI019 extends JFrame {

    private static IntroGUI019 prog;
    private GraphicsJPanel canvas;

    public static void main(String[] args) {
        prog = new IntroGUI019(args);
        prog.setTitle("GUI");

        prog.setBounds(20, 20, 507, 429);
        prog.setResizable(false);
        prog.setVisible(true);
        prog.toFront();
    }

    public IntroGUI019(String[] args) {
        canvas = new GraphicsJPanel();

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(canvas, BorderLayout.CENTER);
    }
}
```

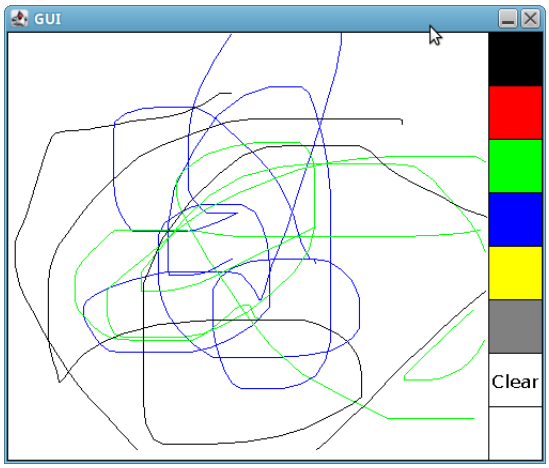


Figure 15.20: IntroGUI019.java Output

15.21 Freehand Drawing Example with Buttons

In our final example of this chapter we show how to incorporate buttons into our application. We will create three buttons, OK, Clear, and Random. The OK button will close the program, the Clear button will clear the graphics area and the Random button will draw 300 random lines.

The GraphicsJPanel has the freehand drawing code from the last several examples and it has a new method `randomLines` that produces the 300 random lines, exactly like in our beginning examples. The major difference between this and the previous examples is in the constructor of the JFrame and the creation of buttons. To create a button (JButton) we use the following style,

```
JButton OK_Button = new JButton("OK");
OK_Button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        System.exit(0);
    }
});
```

The first line creates the button and assigns the caption of the button. The next 5 lines, which is really a single command, assigns to the button an action. If the button is pressed this action will take place. The action for the OK button is simply `System.exit(0);` which, as its name implies, shuts down the program. The rest of the code around this action just creates an ActionListener for the button. As with the mouse, we need to create a listener for the component to pick up any events. For a button, these are ActionListeners. The Clear button will call `canvas.clearScreen();` if pressed, and this method clears the screen. The Random button will call `canvas.randomLines();` if pressed, and draws 300 lines to the screen.

When you create a button, or anything else, it does not immediately show up on the frame, you need to add the controls to panels and get them into the content pane. After our three buttons are created we add them to a JPanel called `buttons`, we make the border of this panel a black line, and then add it to the South area of the border layout.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.Random;
4  import javax.swing.*;
5
6  /*-
7   * GraphicsJPanel
8   * Freehand drawing panel and random line graphing.
9   * Author:  Don Spickler
10  * Date: 7/6/2016
11  */
12
13 public class GraphicsJPanel extends JPanel implements MouseListener, MouseMotionListener {
14
15     private int prevX, prevY; // The previous location of the mouse.
16
17     public GraphicsJPanel() {
18         setBackground(Color.white);
19         addMouseListener(this);
20         addMouseMotionListener(this);
21     }
22
```

```
23      public void clearScreen() {
24          Graphics g = getGraphics();
25
26          g.setColor(Color.WHITE);
27          g.fillRect(0, 0, getWidth(), getHeight());
28      }
29
30      public void randomLines() {
31          Graphics g = getGraphics();
32          Random gen = new Random();
33
34          int drawWidth = getWidth();
35          int drawHeight = getHeight();
36
37          for (int i = 0; i < 300; i++) {
38              g.setColor(new Color(gen.nextInt(255), gen.nextInt(255), gen.nextInt(255)));
39              g.drawLine(gen.nextInt(drawWidth), gen.nextInt(drawHeight), gen.nextInt(
                    drawWidth),
40                      gen.nextInt(drawHeight));
41          }
42      }
43
44      public void paint(Graphics g) {
45          super.paint(g);
46          clearScreen();
47      }
48
49      public void mousePressed(MouseEvent evt) {
50          prevX = evt.getX(); // x-coordinate where the user clicked.
51          prevY = evt.getY(); // y-coordinate where the user clicked.
52      }
53
54      public void mouseDragged(MouseEvent evt) {
55          int x = evt.getX(); // x-coordinate of mouse.
56          int y = evt.getY(); // y-coordinate of mouse.
57
58          Graphics g = getGraphics();
59          g.drawLine(prevX, prevY, x, y);
60
61          // Get ready for the next line segment in the curve.
62          prevX = x;
63          prevY = y;
64      }
65
66      // Some empty routines. These are required by the MouseListener
67      // and MouseMotionListener interfaces.
68
69      public void mouseReleased(MouseEvent evt) {
70      }
71
72      public void mouseClicked(MouseEvent evt) {
73      }
74
75      public void mouseEntered(MouseEvent evt) {
76      }
77
78      public void mouseExited(MouseEvent evt) {
79      }
80
81      public void mouseMoved(MouseEvent evt) {
82      }
83  }
```

```
1  import java.awt.*;
```

```
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.border.*;
5
6  /*-
7   * IntroGUI020
8   * Creates a special JPanel for graphing and places the panel on the frame.
9   * Author:  Don Spickler
10  * Date: 7/6/2016
11  */
12
13 public class IntroGUI020 extends JFrame {
14
15     private static IntroGUI020 prog;
16     private GraphicsJPanel canvas;
17
18     public static void main(String[] args) {
19         prog = new IntroGUI020(args);
20         prog.setTitle("GUI");
21
22         prog.setBounds(20, 20, 507, 429);
23         prog.setVisible(true);
24         prog.toFront();
25     }
26
27     public IntroGUI020(String[] args) {
28         canvas = new GraphicsJPanel();
29
30         JButton OK_Button = new JButton("OK");
31         OK_Button.addActionListener(new ActionListener() {
32             public void actionPerformed(ActionEvent evt) {
33                 System.exit(0);
34             }
35         });
36
37         JButton Cancel_Button = new JButton("Clear");
38         Cancel_Button.addActionListener(new ActionListener() {
39             public void actionPerformed(ActionEvent evt) {
40                 canvas.clearScreen();
41             }
42         });
43
44         JButton Random_Button = new JButton("Random");
45         Random_Button.addActionListener(new ActionListener() {
46             public void actionPerformed(ActionEvent evt) {
47                 canvas.randomLines();
48             }
49         });
50
51         JPanel buttons = new JPanel();
52         buttons.add(OK_Button);
53         buttons.add(Cancel_Button);
54         buttons.add(Random_Button);
55         buttons.setBorder(new LineBorder(Color.BLACK));
56
57         getContentPane().setLayout(new BorderLayout());
58         getContentPane().add(canvas, BorderLayout.CENTER);
59         getContentPane().add(buttons, BorderLayout.SOUTH);
60     }
61 }
```

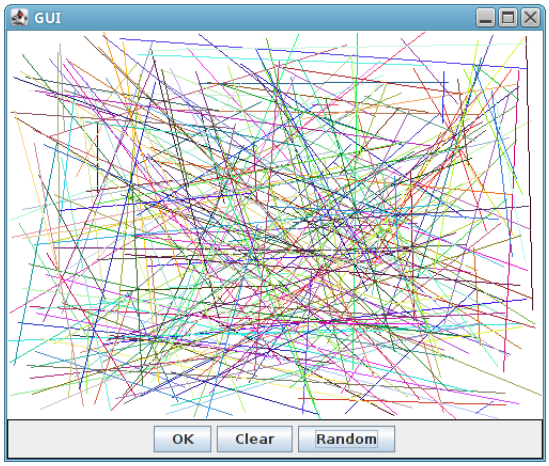


Figure 15.21: IntroGUI020.java Output

Chapter 16

Graphical User Interface Example: JavaPad

16.1 Introduction

As with the last chapter, this series of examples is simply to give you a feel for how a GUI application is put together and the things involved in its creation. As with the last chapter we hope that you will gain enough background to be able to make alterations to the this code but for those interested in crating other applications from scratch, please consult a text or online source on Java Swing. This application is a simple text editor for editing, opening, saving, and printing text documents. It has added features of displaying word and character counts, undo and redo, cut, copy, paste, select all, toggle for line wrapping and an about screen. In addition, we will be constructing menu systems and a toolbar for the selection of these features.

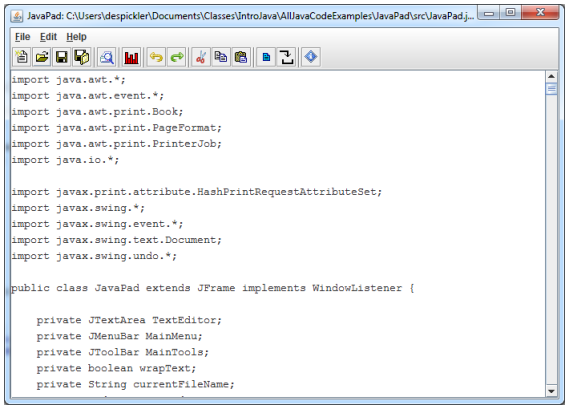


Figure 16.1: JavaPad Application

In this series of examples we start out with the most basic JFrame and keep

building on to it until we have our final application.

16.2 JavaPad Step #1: The JFrame

We first create a JFrame for the application. Note that this is similar to the first example from the previous chapter. The only difference is that we include a WindowListener that can track window events. Although we do not use them in this application, except to close the program, it is good practice to incorporate them in more professional applications.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * JFrame shell.
8   * Author:  Don Spickler
9   * Date: 7/6/2016
10  */
11
12 public class JavaPad extends JFrame implements WindowListener {
13
14     public static void main(String[] args) {
15         JavaPad prog = new JavaPad(args);
16
17         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
18         prog.setBounds(20, 20, 700, 500);
19         prog.setVisible(true);
20         prog.toFront();
21     }
22
23     public JavaPad(String[] args) {
24         addWindowListener(this);
25     }
26
27     public void windowActivated(WindowEvent e) {
28     }
29
30     public void windowClosed(WindowEvent e) {
31     }
32
33     public void windowClosing(WindowEvent e) {
34         System.exit(0);
35     }
36
37     public void windowDeactivated(WindowEvent e) {
38     }
39
40     public void windowDeiconified(WindowEvent e) {
41     }
42
43     public void windowIconified(WindowEvent e) {
44     }
45
46     public void windowOpened(WindowEvent e) {
47     }
48 }
```

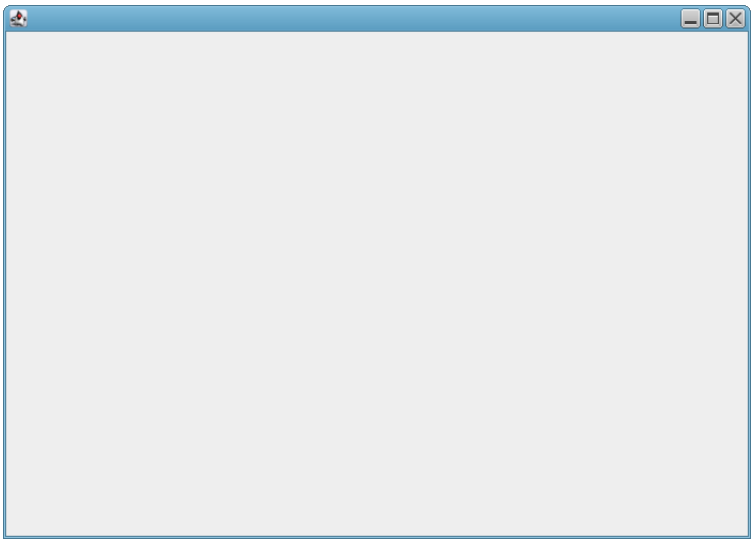


Figure 16.2: JavaPad.java Output: Draft #1

16.3 JavaPad Step #2: Add the JTextArea

All that was done here was adding in the JTextArea, which is a Java Swing control that allows us to type in text. If you run this you will be able to type in the text to the control. The code is straightforward, create a new JTextArea and place it in the center of the BorderLayout in the content pane.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * Added to the application:
8   * 1. The JTextArea
9   *
10  * Author:  Don Spickler
11  * Date: 7/6/2016
12  */
13
14 public class JavaPad extends JFrame implements WindowListener {
15
16     private JTextArea TextEditor;
17
18     public static void main(String[] args) {
19         JavaPad prog = new JavaPad(args);
20
21         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
22         prog.setBounds(20, 20, 700, 500);
23         prog.setVisible(true);
24         prog.toFront();
25     }
26
27     public JavaPad(String[] args) {
28         addWindowListener(this);
```

```
29          setTitle("JavaPad");
30
31          TextEditor = new JTextArea();
32
33          getContentPane().setLayout(new BorderLayout());
34          getContentPane().add(TextEditor, BorderLayout.CENTER);
35      }
36
37      public void windowActivated(WindowEvent e) {
38      }
39
40      public void windowClosed(WindowEvent e) {
41      }
42
43      public void windowClosing(WindowEvent e) {
44          System.exit(0);
45      }
46
47      public void windowDeactivated(WindowEvent e) {
48      }
49
50      public void windowDeiconified(WindowEvent e) {
51      }
52
53      public void windowIconified(WindowEvent e) {
54      }
55
56      public void windowOpened(WindowEvent e) {
57      }
58  }
```

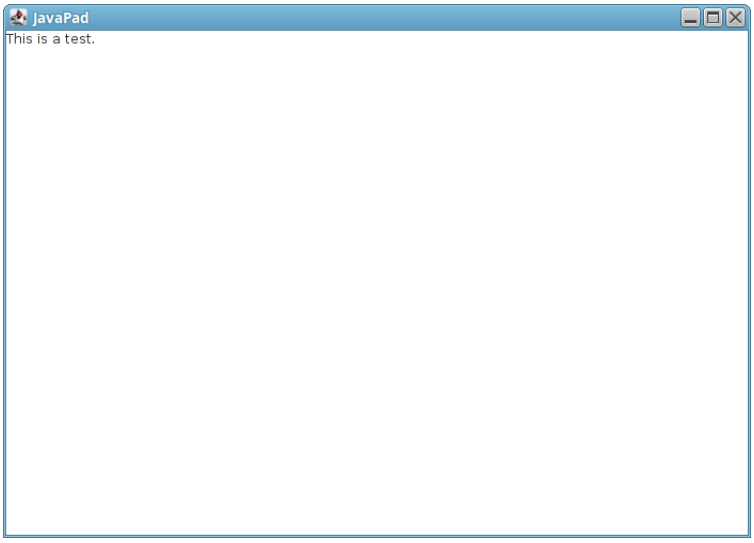


Figure 16.3: JavaPad.java Output: Draft #2

16.4 JavaPad Step #3: Add in a Shell of a Menu

This adds a menu, not with a lot of functionality at this point, to the application. To create a menu, we do the following,

1. Create a new JMenuBar.
2. Create JMenu items for each menu, like File, Edit, and Help.
3. Create JMenuItem items for each feature.
4. Add the JMenuItems to the appropriate JMenus.
5. Add the JMenus to the JMenuBar.
6. Use the setJMenuBar method to put the menu on the application.

Note that once this is done we can update the menu by adding more JMenuItems and adding them to the JMenus. We do the menu creation in a method called `createMenu`, the first line creates the JMenuBar object. The next set of commands create the JMenus, we first make a new JMenu and then we set the mnemonic, which is the underlined character in the menu that is used with the Alt key for keyboard selection of the menu item.

```
JMenu FileMenu = new JMenu("File");
FileMenu.setMnemonic('F');
```

Next we create the JMenuItems, each has a similar structure to the buttons we looked at in the last chapter. Create a new one, with caption, set its mnemonic and then associate an action listener with the menu item. This action listener calls, `windowClosing(null);` which shuts down the program.

```
JMenuItem ExitMenuItem = new JMenuItem("Exit");
ExitMenuItem.setMnemonic('x');
ExitMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        windowClosing(null);
    }
});
```

To add the menu item to the menu we use the add method,

```
FileMenu.add(ExitMenuItem);
```

To add the menu to the menu bar we use the add method,

`MainMenu.add(FileMenu);`

To add the menu bar to the application we use the setJMenuBar method in the constructor,

`setJMenuBar(MainMenu);`

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * Added to the application:
8   * 1. The JTextArea
9   * 2. The Menu
10  *
11  * Author:  Don Spickler
12  * Date: 7/6/2016
13  */
14
15 public class JavaPad extends JFrame implements WindowListener {
16
17     private JTextArea TextEditor;
18     private JMenuBar MainMenu;
19
20     public static void main(String[] args) {
21         JavaPad prog = new JavaPad(args);
22
23         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
24         prog.setBounds(20, 20, 700, 500);
25         prog.setVisible(true);
26         prog.toFront();
27     }
28
29     public JavaPad(String[] args) {
30         addWindowListener(this);
31         setTitle("JavaPad");
32
33         TextEditor = new JTextArea();
34
35         getContentPane().setLayout(new BorderLayout());
36         getContentPane().add(TextEditor, BorderLayout.CENTER);
37
38         createMenu();
39         setJMenuBar(MainMenu);
40     }
41
42     private void createMenu() {
43         MainMenu = new JMenuBar();
44
45         JMenu FileMenu = new JMenu("File");
46         FileMenu.setMnemonic('F');
47
48         JMenu EditMenu = new JMenu("Edit");
49         EditMenu.setMnemonic('E');
50
51         JMenu HelpMenu = new JMenu("Help");
52         HelpMenu.setMnemonic('H');
53
54         JMenuItem ExitMenuItem = new JMenuItem("Exit");
55         ExitMenuItem.setMnemonic('x');
```

```
56          ExitMenuItem.addActionListener(new ActionListener() {
57              public void actionPerformed(ActionEvent evt) {
58                  windowClosing(null);
59              }
60          });
61
62          JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
63          AboutMenuItem.setMnemonic('A');
64          AboutMenuItem.addActionListener(new ActionListener() {
65              public void actionPerformed(ActionEvent evt) {
66              }
67          });
68
69          FileMenu.add(ExitMenuItem);
70
71          HelpMenu.add(AboutMenuItem);
72
73          MainMenu.add(FileMenu);
74          MainMenu.add(EditMenu);
75          MainMenu.add(HelpMenu);
76      }
77
78      public void windowActivated(WindowEvent e) {
79      }
80
81      public void windowClosed(WindowEvent e) {
82      }
83
84      public void windowClosing(WindowEvent e) {
85          System.exit(0);
86      }
87
88      public void windowDeactivated(WindowEvent e) {
89      }
90
91      public void windowDeiconified(WindowEvent e) {
92      }
93
94      public void windowIconified(WindowEvent e) {
95      }
96
97      public void windowOpened(WindowEvent e) {
98      }
99  }
```

16.5 JavaPad Step #4: Add in the About Screen

Making an about screen is fairly easy if you do not want anything too fancy. We created a method, `onShowAbout` the the action listener calls and looks like the following.

```
private void onShowAbout() {
    JOptionPane.showMessageDialog(this, "JavaPad\nWritten by
       : Don Spickler\nCopyright 2016", "About JavaPad",
       JOptionPane.INFORMATION_MESSAGE);
    }
```

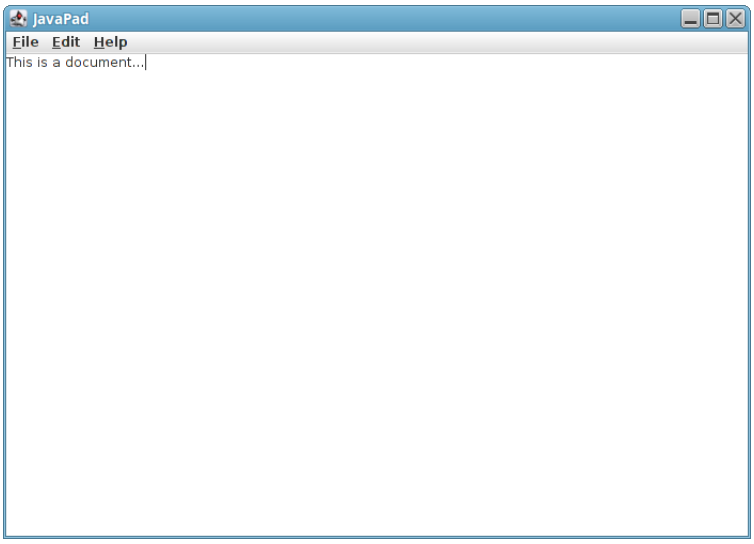


Figure 16.4: JavaPad.java Output: Draft #3

Java Swing has built in message dialog boxes that you can use for simple interaction with the user. It also has input dialogs for getting some simple information from the user. Here we use a message dialog to display our program's copyright information. The `this` refers to the JFrame and it is the parent of the message box, the next string is the message, the next is the dialog title, and the final is the message type, which determines the icon used in the display. Other message types include warnings and errors, with different icons.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * Added to the application:
8   * 1. The JTextArea
9   * 2. The Menu
10  * 3. The About Screen
11  *
12  * Author:  Don Spickler
13  * Date: 7/6/2016
14  */
15
16 public class JavaPad extends JFrame implements WindowListener {
17
18     private JTextArea TextEditor;
19     private JMenuBar MainMenu;
20
21     public static void main(String[] args) {
22         JavaPad prog = new JavaPad(args);
23
24         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
25         prog.setBounds(20, 20, 700, 500);
26         prog.setVisible(true);
27         prog.toFront();
```

```
28      }
29
30      public JavaPad(String[] args) {
31          addWindowListener(this);
32          setTitle("JavaPad");
33
34          TextEditor = new JTextArea();
35
36          getContentPane().setLayout(new BorderLayout());
37          getContentPane().add(TextEditor, BorderLayout.CENTER);
38
39          createMenu();
40          setJMenuBar(MainMenu);
41      }
42
43      private void createMenu() {
44          MainMenu = new JMenuBar();
45
46          JMenu FileMenu = new JMenu("File");
47          FileMenu.setMnemonic('F');
48
49          JMenu EditMenu = new JMenu("Edit");
50          EditMenu.setMnemonic('E');
51
52          JMenu HelpMenu = new JMenu("Help");
53          HelpMenu.setMnemonic('H');
54
55          JMenuItem ExitMenuItem = new JMenuItem("Exit");
56          ExitMenuItem.setMnemonic('x');
57          ExitMenuItem.addActionListener(new ActionListener() {
58              public void actionPerformed(ActionEvent evt) {
59                  windowClosing(null);
60              }
61          });
62
63          JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
64          AboutMenuItem.setMnemonic('A');
65          AboutMenuItem.addActionListener(new ActionListener() {
66              public void actionPerformed(ActionEvent evt) {
67                  onShowAbout();
68              }
69          });
70
71          FileMenu.add(ExitMenuItem);
72
73          HelpMenu.add(AboutMenuItem);
74
75          MainMenu.add(FileMenu);
76          MainMenu.add(EditMenu);
77          MainMenu.add(HelpMenu);
78      }
79
80      private void onShowAbout() {
81          JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad", JOptionPane.INFORMATION_MESSAGE);
82      }
83
84      public void windowActivated(WindowEvent e) {
85      }
86
87      public void windowClosed(WindowEvent e) {
88      }
89
90      public void windowClosing(WindowEvent e) {
```

```
91          System.exit(0);
92      }
93
94      public void windowDeactivated(WindowEvent e) {
95      }
96
97      public void windowDeiconified(WindowEvent e) {
98      }
99
100     public void windowIconified(WindowEvent e) {
101     }
102
103     public void windowOpened(WindowEvent e) {
104     }
105 }
```

Figure 16.5: JavaPad.java Output: Draft #4

16.6 JavaPad Step #5: Add in Cut, Copy, and Paste

These are simply three new menu items, so we add them as we did before. The only difference here is that instead of adding a mnemonic we will add an accelerator, that is, a keystroke combination that will select the menu item. The one used for Cut is Ctrl+X. These also show up automatically on the menu.

```
JMenuItem CutMenuItem = new JMenuItem("Cut");
CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X',
   InputEvent.CTRL_DOWN_MASK));
CutMenuItem.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        TextEditor.cut();
    }
});
```

Cut, copy, and paste are all built into the JTextArea control, so we need only call the appropriate method.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * Added to the application:
8   * 1. The JTextArea
9   * 2. The Menu
10  * 3. The About Screen
11  * 4. Added Cut, Copy, and Paste
12  *
13  * Author:  Don Spickler
14  * Date: 7/6/2016
15  */
16
17 public class JavaPad extends JFrame implements WindowListener {
18
19     private JTextArea TextEditor;
20     private JMenuBar MainMenu;
21
22     public static void main(String[] args) {
23         JavaPad prog = new JavaPad(args);
24
25         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
26         prog.setBounds(20, 20, 700, 500);
27         prog.setVisible(true);
28         prog.toFront();
29     }
30
31     public JavaPad(String[] args) {
32         addWindowListener(this);
33         setTitle("JavaPad");
34
35         TextEditor = new JTextArea();
36
37         getContentPane().setLayout(new BorderLayout());
38         getContentPane().add(TextEditor, BorderLayout.CENTER);
39
40         createMenu();
41         setJMenuBar(MainMenu);
42     }
43
44     private void createMenu() {
45         MainMenu = new JMenuBar();
46
47         JMenu FileMenu = new JMenu("File");
48         FileMenu.setMnemonic('F');
49
50         JMenu EditMenu = new JMenu("Edit");
51         EditMenu.setMnemonic('E');
52
53         JMenu HelpMenu = new JMenu("Help");
54         HelpMenu.setMnemonic('H');
55
56         JMenuItem ExitMenuItem = new JMenuItem("Exit");
57         ExitMenuItem.setMnemonic('x');
58         ExitMenuItem.addActionListener(new ActionListener() {
59             public void actionPerformed(ActionEvent evt) {
60                 windowClosing(null);
61             }
62         });
63
64         JMenuItem CutMenuItem = new JMenuItem("Cut");
```

```
65          CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
66          CutMenuItem.addActionListener(new ActionListener() {
67              public void actionPerformed(ActionEvent evt) {
68                  TextEditor.cut();
69              }
70          });
71
72          JMenuItem CopyMenuItem = new JMenuItem("Copy");
73          CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
74          CopyMenuItem.addActionListener(new ActionListener() {
75              public void actionPerformed(ActionEvent evt) {
76                  TextEditor.copy();
77              }
78          });
79
80          JMenuItem PasteMenuItem = new JMenuItem("Paste");
81          PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
82          PasteMenuItem.addActionListener(new ActionListener() {
83              public void actionPerformed(ActionEvent evt) {
84                  TextEditor.paste();
85              }
86          });
87
88          JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
89          AboutMenuItem.setMnemonic('A');
90          AboutMenuItem.addActionListener(new ActionListener() {
91              public void actionPerformed(ActionEvent evt) {
92                  onShowAbout();
93              }
94          });
95
96          FileMenu.add(ExitMenuItem);
97
98          EditMenu.add(CutMenuItem);
99          EditMenu.add(CopyMenuItem);
100         EditMenu.add(PasteMenuItem);
101
102         HelpMenu.add(AboutMenuItem);
103
104         MainMenu.add(FileMenu);
105         MainMenu.add(EditMenu);
106         MainMenu.add(HelpMenu);
107     }
108
109     private void onShowAbout() {
110         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad",
111                 JOptionPane.INFORMATION_MESSAGE);
112     }
113
114     public void windowActivated(WindowEvent e) {
115     }
116
117     public void windowClosed(WindowEvent e) {
118     }
119
120     public void windowClosing(WindowEvent e) {
121         System.exit(0);
122     }
123
124     public void windowDeactivated(WindowEvent e) {
```

```
125     }
126
127     public void windowDeiconified(WindowEvent e) {
128     }
129
130     public void windowIconified(WindowEvent e) {
131     }
132
133     public void windowOpened(WindowEvent e) {
134     }
135 }
```

16.7 JavaPad Step #6: Add in Select All

As with cut, copy, and paste, select all is built into the JTextArea control, so we need only call the method by `TextEditor.selectAll();`.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  /*-
6   * JavaPad
7   * Added to the application:
8   * 1. The JTextArea
9   * 2. The Menu
10  * 3. The About Screen
11  * 4. Added Cut, Copy, and Paste
12  * 5. Add in Select All
13  *
14  * Author:  Don Spickler
15  * Date: 7/6/2016
16  */
17
18 public class JavaPad extends JFrame implements WindowListener {
19
20     private JTextArea TextEditor;
21     private JMenuBar MainMenu;
22
23     public static void main(String[] args) {
24         JavaPad prog = new JavaPad(args);
25
26         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
27         prog.setBounds(20, 20, 700, 500);
28         prog.setVisible(true);
29         prog.toFront();
30     }
31
32     public JavaPad(String[] args) {
33         addWindowListener(this);
34         setTitle("JavaPad");
35
36         TextEditor = new JTextArea();
37         JScrollPane TextEditorSP = new JScrollPane(TextEditor);
38
39         getContentPane().setLayout(new BorderLayout());
40         getContentPane().add(TextEditorSP, BorderLayout.CENTER);
41
42         createMenu();
```

```
43          setJMenuBar(MainMenu);
44      }
45
46      private void createMenu() {
47          MainMenu = new JMenuBar();
48
49          JMenu FileMenu = new JMenu("File");
50          FileMenu.setMnemonic('F');
51
52          JMenu EditMenu = new JMenu("Edit");
53          EditMenu.setMnemonic('E');
54
55          JMenu HelpMenu = new JMenu("Help");
56          HelpMenu.setMnemonic('H');
57
58          JMenuItem ExitMenuItem = new JMenuItem("Exit");
59          ExitMenuItem.setMnemonic('x');
60          ExitMenuItem.addActionListener(new ActionListener() {
61              public void actionPerformed(ActionEvent evt) {
62                  windowClosing(null);
63              }
64          });
65
66          JMenuItem CutMenuItem = new JMenuItem("Cut");
67          CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
68          CutMenuItem.addActionListener(new ActionListener() {
69              public void actionPerformed(ActionEvent evt) {
70                  TextEditor.cut();
71              }
72          });
73
74          JMenuItem CopyMenuItem = new JMenuItem("Copy");
75          CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
76          CopyMenuItem.addActionListener(new ActionListener() {
77              public void actionPerformed(ActionEvent evt) {
78                  TextEditor.copy();
79              }
80          });
81
82          JMenuItem PasteMenuItem = new JMenuItem("Paste");
83          PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
84          PasteMenuItem.addActionListener(new ActionListener() {
85              public void actionPerformed(ActionEvent evt) {
86                  TextEditor.paste();
87              }
88          });
89
90          JMenuItem SelectAllMenuItem = new JMenuItem("Select All");
91          SelectAllMenuItem.setAccelerator(KeyStroke.getKeyStroke('A', InputEvent.
                CTRL_DOWN_MASK));
92          SelectAllMenuItem.addActionListener(new ActionListener() {
93              public void actionPerformed(ActionEvent evt) {
94                  TextEditor.selectAll();
95              }
96          });
97
98          JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
99          AboutMenuItem.setMnemonic('A');
100         AboutMenuItem.addActionListener(new ActionListener() {
101             public void actionPerformed(ActionEvent evt) {
102                 onShowAbout();
```

```
103             }
104         });
105
106         FileMenu.add(ExitMenuItem);
107
108         EditMenu.add(CutMenuItem);
109         EditMenu.add(CopyMenuItem);
110         EditMenu.add(PasteMenuItem);
111         EditMenu.addSeparator();
112         EditMenu.add(SelectAllMenuItem);
113
114         HelpMenu.add(AboutMenuItem);
115
116         MainMenu.add(FileMenu);
117         MainMenu.add(EditMenu);
118         MainMenu.add(HelpMenu);
119     }
120
121     private void onShowAbout() {
122         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad",
123                 JOptionPane.INFORMATION_MESSAGE);
124     }
125
126     public void windowActivated(WindowEvent e) {
127     }
128
129     public void windowClosed(WindowEvent e) {
130     }
131
132     public void windowClosing(WindowEvent e) {
133         System.exit(0);
134     }
135
136     public void windowDeactivated(WindowEvent e) {
137     }
138
139     public void windowDeiconified(WindowEvent e) {
140     }
141
142     public void windowIconified(WindowEvent e) {
143     }
144
145     public void windowOpened(WindowEvent e) {
146     }
147 }
```

16.8 JavaPad Step #7: Add in File Transfer

The additions we made in the above examples have been fairly short, from here on out the addition are going to be lengthy and we will be glossing over a lot of details. We added a new file (class) to the project `SimpleFileFilter` which is in the appendices. All this does is allow us to easily control the file types that are in the file selection dialog. At the bottom of a file selector there is usually a drop-down that allows the user to filter the file types for easier selection. This `SimpleFileFilter` allows us to update that drop-down for our own application.

In the JavaPad file we made many additions. First we added New, Open, Save, and Save As to the menu system and linked their action listeners to methods for each. We also added methods to take care of setting up the dialog file selectors (for example, `OpenTextFile`). These use the `SimpleFileFilter` and the JFileChooser, which does most of the work. If a legitimate filename is returned from the JFileChooser we either read the file or write to the file, depending on which method we are in. The reader is set up to read in chunks of 10,000 characters from the text file at a time until the file is read. The writer, on the other hand, will dump the entire contents of the JTextArea to the file in one write. We have also incorporated a method to check to see if a file already exists, so we can warn the user if they are about to overwrite a file that already exists.

In addition, we added a menu option to toggle line wrapping and we updated the titlebar to display the filename and path of the selected file.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.io.*;
4  import javax.swing.*;
5
6  /*-
7   * JavaPad
8   * Added to the application:
9   * 1. The JTextArea
10  * 2. The Menu
11  * 3. The About Screen
12  * 4. Cut, Copy, and Paste
13  * 5. Select All
14  * 6. File transfer and choosing
15  *
16  * Author:  Don Spickler
17  * Date: 7/6/2016
18  */
19
20 public class JavaPad extends JFrame implements WindowListener {
21
22     private JTextArea TextEditor;
23     private JMenuBar MainMenu;
24     private boolean wrapText;
25     private String currentFileName;
26
27     public static void main(String[] args) {
28         JavaPad prog = new JavaPad(args);
29
30         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
31         prog.setBounds(20, 20, 700, 500);
32         prog.setVisible(true);
33         prog.toFront();
34     }
35
36     public JavaPad(String[] args) {
37         addWindowListener(this);
38         currentFileName = "";
39         updateTitle();
40
41         TextEditor = new JTextArea();
42         JScrollPane TextEditorSP = new JScrollPane(TextEditor);
43
44         wrapText = true;
```

```
45          TextEditor.setLineWrap(wrapText);
46          TextEditor.setWrapStyleWord(wrapText);
47
48          getContentPane().setLayout(new BorderLayout());
49          getContentPane().add(TextEditorSP, BorderLayout.CENTER);
50
51          createMenu();
52          setJMenuBar(MainMenu);
53      }
54
55      private void updateTitle() {
56          if (currentFileName == "")
57              setTitle("JavaPad: Untitled");
58          else
59              setTitle("JavaPad: " + currentFileName);
60      }
61
62      private void createMenu() {
63          MainMenu = new JMenuBar();
64
65          JMenu FileMenu = new JMenu("File");
66          FileMenu.setMnemonic('F');
67
68          JMenu EditMenu = new JMenu("Edit");
69          EditMenu.setMnemonic('E');
70
71          JMenu HelpMenu = new JMenu("Help");
72          HelpMenu.setMnemonic('H');
73
74          JMenuItem NewMenuItem = new JMenuItem("New");
75          NewMenuItem.setMnemonic('n');
76          NewMenuItem.setAccelerator(KeyStroke.getKeyStroke('N', InputEvent.CTRL_DOWN_MASK))
                ;
77          NewMenuItem.addActionListener(new ActionListener() {
78              public void actionPerformed(ActionEvent evt) {
79                  onNew();
80              }
81          });
82
83          JMenuItem OpenMenuItem = new JMenuItem("Open...");
84          OpenMenuItem.setMnemonic('o');
85          OpenMenuItem.setAccelerator(KeyStroke.getKeyStroke('O', InputEvent.CTRL_DOWN_MASK)
                );
86          OpenMenuItem.addActionListener(new ActionListener() {
87              public void actionPerformed(ActionEvent evt) {
88                  onOpen();
89              }
90          });
91
92          JMenuItem SaveAsMenuItem = new JMenuItem("Save As...");
93          SaveAsMenuItem.setMnemonic('a');
94          SaveAsMenuItem.addActionListener(new ActionListener() {
95              public void actionPerformed(ActionEvent evt) {
96                  onSaveAs();
97              }
98          });
99
100         JMenuItem SaveMenuItem = new JMenuItem("Save");
101         SaveMenuItem.setMnemonic('s');
102         SaveMenuItem.setAccelerator(KeyStroke.getKeyStroke('S', InputEvent.CTRL_DOWN_MASK)
                );
103         SaveMenuItem.addActionListener(new ActionListener() {
104             public void actionPerformed(ActionEvent evt) {
105                 onSave();
```

```
106             }
107         });
108 
109         JMenuItem ExitMenuItem = new JMenuItem("Exit");
110         ExitMenuItem.setMnemonic('x');
111         ExitMenuItem.addActionListener(new ActionListener() {
112             public void actionPerformed(ActionEvent evt) {
113                 windowClosing(null);
114             }
115         });
116 
117         JMenuItem CutMenuItem = new JMenuItem("Cut");
118         CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
119         CutMenuItem.addActionListener(new ActionListener() {
120             public void actionPerformed(ActionEvent evt) {
121                 TextEditor.cut();
122             }
123         });
124 
125         JMenuItem CopyMenuItem = new JMenuItem("Copy");
126         CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
127         CopyMenuItem.addActionListener(new ActionListener() {
128             public void actionPerformed(ActionEvent evt) {
129                 TextEditor.copy();
130             }
131         });
132 
133         JMenuItem PasteMenuItem = new JMenuItem("Paste");
134         PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
135         PasteMenuItem.addActionListener(new ActionListener() {
136             public void actionPerformed(ActionEvent evt) {
137                 TextEditor.paste();
138             }
139         });
140 
141         JMenuItem SelectAllMenuItem = new JMenuItem("Select All");
142         SelectAllMenuItem.setAccelerator(KeyStroke.getKeyStroke('A', InputEvent.
                CTRL_DOWN_MASK));
143         SelectAllMenuItem.addActionListener(new ActionListener() {
144             public void actionPerformed(ActionEvent evt) {
145                 TextEditor.selectAll();
146             }
147         });
148 
149         JMenuItem ToggleLineWrapMenuItem = new JMenuItem("Toggle Line Wrap Mode");
150         ToggleLineWrapMenuItem.addActionListener(new ActionListener() {
151             public void actionPerformed(ActionEvent evt) {
152                 wrapText = !wrapText;
153                 TextEditor.setLineWrap(wrapText);
154                 TextEditor.setWrapStyleWord(wrapText);
155             }
156         });
157 
158         JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
159         AboutMenuItem.setMnemonic('A');
160         AboutMenuItem.addActionListener(new ActionListener() {
161             public void actionPerformed(ActionEvent evt) {
162                 onShowAbout();
163             }
164         });
165 
```

```
166         FileMenu.add(NewMenuItem);
167         FileMenu.add(OpenMenuItem);
168         FileMenu.addSeparator();
169         FileMenu.add(SaveMenuItem);
170         FileMenu.add(SaveAsMenuItem);
171         FileMenu.addSeparator();
172         FileMenu.add(ExitMenuItem);
173 
174         EditMenu.add(CutMenuItem);
175         EditMenu.add(CopyMenuItem);
176         EditMenu.add(PasteMenuItem);
177         EditMenu.addSeparator();
178         EditMenu.add(SelectAllMenuItem);
179         EditMenu.addSeparator();
180         EditMenu.add(ToggleLineWrapMenuItem);
181 
182         HelpMenu.add(AboutMenuItem);
183 
184         MainMenu.add(FileMenu);
185         MainMenu.add(EditMenu);
186         MainMenu.add(HelpMenu);
187     }
188 
189     private void onNew() {
190         boolean oktogo = true;
191 
192         if (!TextEditor.getText().isEmpty()) {
193             oktogo = false;
194             int ans = JOptionPane.showConfirmDialog(this,
195                     "This will clear all of the text in the editor.\nDo you wish to delete
                         the current text?",
196                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                         WARNING_MESSAGE);
197             if (ans == JOptionPane.YES_OPTION)
198                 oktogo = true;
199         }
200 
201         if (!oktogo)
202             return;
203 
204         TextEditor.setText("");
205         currentFileName = "";
206         updateTitle();
207     }
208 
209     private void onOpen() {
210         boolean oktogo = true;
211 
212         if (!TextEditor.getText().isEmpty()) {
213             oktogo = false;
214             int ans = JOptionPane.showConfirmDialog(this,
215                     "This will replace all of the text in the editor with the file
                         contents.\nDo you wish to overwrite the current text?",
216                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                         WARNING_MESSAGE);
217             if (ans == JOptionPane.YES_OPTION)
218                 oktogo = true;
219         }
220 
221         if (!oktogo)
222             return;
223 
224         String filetext = OpenTextFile("Text Files", "txt");
225         TextEditor.setText(filetext);
```

```
226         TextEditor.setCaretPosition(0);
227         updateTitle();
228     }
229
230     private void onSaveAs() {
231         onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
232         updateTitle();
233     }
234
235     private void onSave() {
236         if (currentFileName == "")
237             onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
238         else
239             saveFile(currentFileName, TextEditor.getText());
240
241         updateTitle();
242     }
243
244     private void onShowAbout() {
245         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad",
246                 JOptionPane.INFORMATION_MESSAGE);
247     }
248
249     public void windowActivated(WindowEvent e) {
250     }
251
252     public void windowClosed(WindowEvent e) {
253     }
254
255     public void windowClosing(WindowEvent e) {
256         System.exit(0);
257     }
258
259     public void windowDeactivated(WindowEvent e) {
260     }
261
262     public void windowDeiconified(WindowEvent e) {
263     }
264
265     public void windowIconified(WindowEvent e) {
266     }
267
268     public void windowOpened(WindowEvent e) {
269     }
270
271     public String OpenTextFile(String FileTypeName, String ext) {
272         String[] fileTypes = new String[] { ext };
273         String filename = "";
274         JFileChooser fc = new JFileChooser();
275         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
276         fc.addChoosableFileFilter(FileFilter);
277         fc.setFileFilter(FileFilter);
278         int option = fc.showOpenDialog(this);
279         if (option == JFileChooser.APPROVE_OPTION) {
280             if (fc.getSelectedFile() != null)
281                 filename = fc.getSelectedFile().getPath();
282         } else
283             return "";
284
285         String InputText = "";
286         try {
287             BufferedReader br = new BufferedReader(new FileReader(filename));
```

```
288             char[] cbuf = new char[10000];
289
290             int numchars = br.read(cbuf, 0, 10000);
291             while (numchars > 0) {
292                 InputText += (new String(cbuf, 0, numchars));
293                 numchars = br.read(cbuf, 0, 10000);
294             }
295             br.close();
296             currentFileName = filename;
297         } catch (Exception e) {
298             JOptionPane.showMessageDialog(this, "IO Error:  " + e.toString() + "\nCould
                    not open file.", "IO Error",
299                     JOptionPane.WARNING_MESSAGE);
300             InputText = "";
301             currentFileName = "";
302         }
303         return InputText;
304     }
305
306     public void onTextFileSaveAs(String InfoString, String FileTypeName, String ext) {
307         String[] fileTypes = new String[] { ext };
308         JFileChooser fc = new JFileChooser();
309         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
310         fc.addChoosableFileFilter(FileFilter);
311         fc.setFileFilter(FileFilter);
312         fc.setDialogTitle("Save As");
313         int option = fc.showSaveDialog(this);
314         if (option == JFileChooser.APPROVE_OPTION) {
315             if (fc.getSelectedFile() != null) {
316                 String filename = fc.getSelectedFile().getPath();
317                 filename = filename.trim();
318                 if (!filename.toLowerCase().endsWith("." + ext))
319                     filename = filename + "." + ext;
320
321                 boolean okToSave = true;
322                 if (FileExists(filename)) {
323                     okToSave = false;
324                     int ans = JOptionPane.showConfirmDialog(this,
325                             "The file " + filename + " already exists. \nDo you wish to
                                overwrite the file?",
326                             "Overwrite File", JOptionPane.YES_NO_OPTION, JOptionPane.
                                QUESTION_MESSAGE);
327                     if (ans == JOptionPane.YES_OPTION)
328                         okToSave = true;
329                 }
330
331                 if (okToSave) {
332                     saveFile(filename, InfoString);
333                 }
334             }
335         }
336     }
337
338     public boolean FileExists(String testFilename) {
339         boolean retval = false;
340         try {
341             BufferedReader br = new BufferedReader(new FileReader(testFilename));
342             br.close();
343             retval = true;
344         } catch (Exception e) {
345             retval = false;
346         }
347         return retval;
```

```
348     }
349
350     private void saveFile(String filename, String infoStr) {
351         try {
352             PrintWriter outputfile = new PrintWriter(new FileWriter(filename));
353             outputfile.print(infoStr);
354             outputfile.close();
355             currentFileName = filename;
356         } catch (Exception e) {
357             JOptionPane.showMessageDialog(this, "The file could not be saved.", "IO Error"
                    ,
358                     JOptionPane.WARNING_MESSAGE);
359             File f = new File(filename);
360             try {
361                 f.delete();
362             } catch (Exception delex) {
363             }
364             currentFileName = "";
365         }
366     }
367 }
```

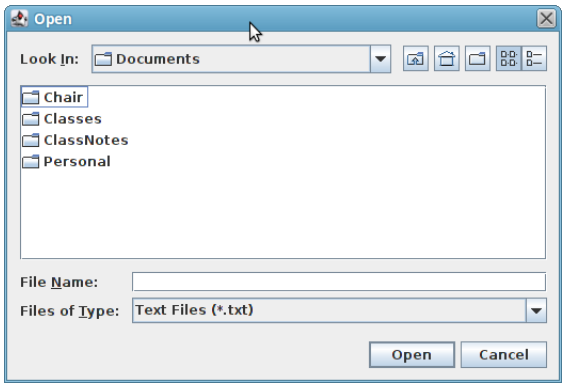


Figure 16.6: JavaPad.java Output: Draft #7

16.9 JavaPad Step #8: Add in Undo, Redo, and File Properties

In this iteration we add in the ability to undo and redo edits as well as an option to print out the number of lines, words, and characters in the text area. The undo and redo are controlled by an UndoManager that is built into Java Swing. We set the limit to -1 meaning that there is no limit on the number of undos. Then we extract the document object from the editor and link it to the UndoManager which will then track changes made to the document. We then add a KeyListener to the editor so it will understand and respond to Ctrl+Z and Ctrl+Y, the standard keystrokes for undo and redo. We then add these options to the menu and link them to methods for undoing and redoing edits.

We also added in a menu item for document properties to print out the number of lines, words, and characters in the text area. This item is associated with the method `onShowStatistics`. This method uses string splitting to make the counts easier to do. When the counts are done we construct JPanels using JLabels for each count, and another panel to put these in and finally put this into the message portion of a message dialog.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.io.*;
4
5  import javax.swing.*;
6  import javax.swing.event.*;
7  import javax.swing.text.Document;
8  import javax.swing.undo.*;
9
10 /*-
11  * JavaPad
12  * Added to the application:
13  * 1. The JTextArea
14  * 2. The Menu
15  * 3. The About Screen
16  * 4. Cut, Copy, and Paste
17  * 5. Select All
18  * 6. File transfer and choosing
19  * 7. Undo, Redo, and file properties
20  *
21  * Author:  Don Spickler
22  * Date: 7/6/2016
23  */
24
25 public class JavaPad extends JFrame implements WindowListener {
26
27     private JTextArea TextEditor;
28     private JMenuBar MainMenu;
29     private boolean wrapText;
30     private String currentFileName;
31     private UndoManager undoManager;
32
33     public static void main(String[] args) {
34         JavaPad prog = new JavaPad(args);
35
36         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
37         prog.setBounds(20, 20, 700, 500);
38         prog.setVisible(true);
39         prog.toFront();
40     }
41
42     public JavaPad(String[] args) {
43         addWindowListener(this);
44         currentFileName = "";
45         updateTitle();
46
47         TextEditor = new JTextArea();
48         JScrollPane TextEditorSP = new JScrollPane(TextEditor);
49
50         wrapText = true;
51         TextEditor.setLineWrap(wrapText);
52         TextEditor.setWrapStyleWord(wrapText);
53
54         getContentPane().setLayout(new BorderLayout());
55         getContentPane().add(TextEditorSP, BorderLayout.CENTER);
```

```
56
57          createMenu();
58          setJMenuBar(MainMenu);
59
60          undoManager = new UndoManager();
61          undoManager.setLimit(-1); // Unlimited
62
63          Document notedoc = TextEditor.getDocument();
64          notedoc.addUndoableEditListener(new UndoableEditListener() {
65              public void undoableEditHappened(UndoableEditEvent e) {
66                  undoManager.addEdit(e.getEdit());
67              }
68          });
69
70          TextEditor.addKeyListener(new KeyListener() {
71              public void keyPressed(KeyEvent e) {
72
73                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
74                      if (e.getKeyCode() == KeyEvent.VK_Z) {
75                          try {
76                              if (undoManager.canUndo()) {
77                                  undoManager.undo();
78                              }
79                          } catch (CannotUndoException exp) {
80                          }
81                      }
82
83                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
84                      if (e.getKeyCode() == KeyEvent.VK_Y) {
85                          try {
86                              if (undoManager.canRedo()) {
87                                  undoManager.redo();
88                              }
89                          } catch (CannotRedoException exp) {
90                          }
91                      }
92              }
93
94              public void keyTyped(KeyEvent e) {
95              }
96
97              public void keyReleased(KeyEvent e) {
98              }
99          });
100
101     }
102
103     private void updateTitle() {
104         if (currentFileName == "")
105             setTitle("JavaPad: Untitled");
106         else
107             setTitle("JavaPad: " + currentFileName);
108     }
109
110     private void createMenu() {
111         MainMenu = new JMenuBar();
112
113         JMenu FileMenu = new JMenu("File");
114         FileMenu.setMnemonic('F');
115
116         JMenu EditMenu = new JMenu("Edit");
117         EditMenu.setMnemonic('E');
118
119         JMenu HelpMenu = new JMenu("Help");
```

```
120         HelpMenu.setMnemonic('H');
121
122         JMenuItem NewMenuItem = new JMenuItem("New");
123         NewMenuItem.setMnemonic('n');
124         NewMenuItem.setAccelerator(KeyStroke.getKeyStroke('N', InputEvent.CTRL_DOWN_MASK))
                ;
125         NewMenuItem.addActionListener(new ActionListener() {
126             public void actionPerformed(ActionEvent evt) {
127                 onNew();
128             }
129         });
130
131         JMenuItem OpenMenuItem = new JMenuItem("Open...");
132         OpenMenuItem.setMnemonic('o');
133         OpenMenuItem.setAccelerator(KeyStroke.getKeyStroke('O', InputEvent.CTRL_DOWN_MASK)
                );
134         OpenMenuItem.addActionListener(new ActionListener() {
135             public void actionPerformed(ActionEvent evt) {
136                 onOpen();
137             }
138         });
139
140         JMenuItem SaveAsMenuItem = new JMenuItem("Save As...");
141         SaveAsMenuItem.setMnemonic('a');
142         SaveAsMenuItem.addActionListener(new ActionListener() {
143             public void actionPerformed(ActionEvent evt) {
144                 onSaveAs();
145             }
146         });
147
148         JMenuItem SaveMenuItem = new JMenuItem("Save");
149         SaveMenuItem.setMnemonic('s');
150         SaveMenuItem.setAccelerator(KeyStroke.getKeyStroke('S', InputEvent.CTRL_DOWN_MASK)
                );
151         SaveMenuItem.addActionListener(new ActionListener() {
152             public void actionPerformed(ActionEvent evt) {
153                 onSave();
154             }
155         });
156
157         JMenuItem StatsMenuItem = new JMenuItem("Properties...");
158         StatsMenuItem.addActionListener(new ActionListener() {
159             public void actionPerformed(ActionEvent evt) {
160                 onShowStatistics();
161             }
162         });
163
164         JMenuItem ExitMenuItem = new JMenuItem("Exit");
165         ExitMenuItem.setMnemonic('x');
166         ExitMenuItem.addActionListener(new ActionListener() {
167             public void actionPerformed(ActionEvent evt) {
168                 windowClosing(null);
169             }
170         });
171
172         JMenuItem UndoMenuItem = new JMenuItem("Undo");
173         UndoMenuItem.addActionListener(new ActionListener() {
174             public void actionPerformed(ActionEvent evt) {
175                 onUndo();
176             }
177         });
178
179         JMenuItem RedoMenuItem = new JMenuItem("Redo");
180         RedoMenuItem.addActionListener(new ActionListener() {
```

```
181             public void actionPerformed(ActionEvent evt) {
182                 onRedo();
183             }
184         });
185 
186         JMenuItem CutMenuItem = new JMenuItem("Cut");
187         CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
188         CutMenuItem.addActionListener(new ActionListener() {
189             public void actionPerformed(ActionEvent evt) {
190                 TextEditor.cut();
191             }
192         });
193 
194         JMenuItem CopyMenuItem = new JMenuItem("Copy");
195         CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
196         CopyMenuItem.addActionListener(new ActionListener() {
197             public void actionPerformed(ActionEvent evt) {
198                 TextEditor.copy();
199             }
200         });
201 
202         JMenuItem PasteMenuItem = new JMenuItem("Paste");
203         PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
204         PasteMenuItem.addActionListener(new ActionListener() {
205             public void actionPerformed(ActionEvent evt) {
206                 TextEditor.paste();
207             }
208         });
209 
210         JMenuItem SelectAllMenuItem = new JMenuItem("Select All");
211         SelectAllMenuItem.setAccelerator(KeyStroke.getKeyStroke('A', InputEvent.
                CTRL_DOWN_MASK));
212         SelectAllMenuItem.addActionListener(new ActionListener() {
213             public void actionPerformed(ActionEvent evt) {
214                 TextEditor.selectAll();
215             }
216         });
217 
218         JMenuItem ToggleLineWrapMenuItem = new JMenuItem("Toggle Line Wrap Mode");
219         ToggleLineWrapMenuItem.addActionListener(new ActionListener() {
220             public void actionPerformed(ActionEvent evt) {
221                 wrapText = !wrapText;
222                 TextEditor.setLineWrap(wrapText);
223                 TextEditor.setWrapStyleWord(wrapText);
224             }
225         });
226 
227         JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
228         AboutMenuItem.setMnemonic('A');
229         AboutMenuItem.addActionListener(new ActionListener() {
230             public void actionPerformed(ActionEvent evt) {
231                 onShowAbout();
232             }
233         });
234 
235         FileMenu.add(NewMenuItem);
236         FileMenu.add(OpenMenuItem);
237         FileMenu.addSeparator();
238         FileMenu.add(SaveMenuItem);
239         FileMenu.add(SaveAsMenuItem);
240         FileMenu.addSeparator();
```

```
241         FileMenu.add(StatsMenuItem);
242         FileMenu.addSeparator();
243         FileMenu.add(ExitMenuItem);
244 
245         EditMenu.add(UndoMenuItem);
246         EditMenu.add(RedoMenuItem);
247         EditMenu.addSeparator();
248         EditMenu.add(CutMenuItem);
249         EditMenu.add(CopyMenuItem);
250         EditMenu.add(PasteMenuItem);
251         EditMenu.addSeparator();
252         EditMenu.add(SelectAllMenuItem);
253         EditMenu.addSeparator();
254         EditMenu.add(ToggleLineWrapMenuItem);
255 
256         HelpMenu.add(AboutMenuItem);
257 
258         MainMenu.add(FileMenu);
259         MainMenu.add(EditMenu);
260         MainMenu.add(HelpMenu);
261     }
262 
263     private void onNew() {
264         boolean oktogo = true;
265 
266         if (!TextEditor.getText().isEmpty()) {
267             oktogo = false;
268             int ans = JOptionPane.showConfirmDialog(this,
269                     "This will clear all of the text in the editor.\nDo you wish to delete
                          the current text?",
270                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                         WARNING_MESSAGE);
271             if (ans == JOptionPane.YES_OPTION)
272                 oktogo = true;
273         }
274 
275         if (!oktogo)
276             return;
277 
278         TextEditor.setText("");
279         currentFileName = "";
280         updateTitle();
281     }
282 
283     private void onOpen() {
284         boolean oktogo = true;
285 
286         if (!TextEditor.getText().isEmpty()) {
287             oktogo = false;
288             int ans = JOptionPane.showConfirmDialog(this,
289                     "This will replace all of the text in the editor with the file
                         contents.\nDo you wish to overwrite the current text?",
290                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                         WARNING_MESSAGE);
291             if (ans == JOptionPane.YES_OPTION)
292                 oktogo = true;
293         }
294 
295         if (!oktogo)
296             return;
297 
298         String filetext = OpenTextFile("Text Files", "txt");
299         TextEditor.setText(filetext);
300         TextEditor.setCaretPosition(0);
```

```
301         updateTitle();
302     }
303 
304     private void onSaveAs() {
305         onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
306         updateTitle();
307     }
308 
309     private void onSave() {
310         if (currentFileName == "")
311             onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
312         else
313             saveFile(currentFileName, TextEditor.getText());
314 
315         updateTitle();
316     }
317 
318     private void onUndo() {
319         try {
320             if (undoManager.canUndo()) {
321                 undoManager.undo();
322             }
323         } catch (CannotUndoException exp) {
324         }
325         TextEditor.requestFocus();
326     }
327 
328     private void onRedo() {
329         try {
330             if (undoManager.canRedo()) {
331                 undoManager.redo();
332             }
333         } catch (CannotUndoException exp) {
334         }
335         TextEditor.requestFocus();
336     }
337 
338     private void onShowAbout() {
339         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad",
340                 JOptionPane.INFORMATION_MESSAGE);
341     }
342 
343     public void windowActivated(WindowEvent e) {
344     }
345 
346     public void windowClosed(WindowEvent e) {
347     }
348 
349     public void windowClosing(WindowEvent e) {
350         System.exit(0);
351     }
352 
353     public void windowDeactivated(WindowEvent e) {
354     }
355 
356     public void windowDeiconified(WindowEvent e) {
357     }
358 
359     public void windowIconified(WindowEvent e) {
360     }
361 
362     public void windowOpened(WindowEvent e) {
363     }
```

```
364
365     public String OpenTextFile(String FileTypeName, String ext) {
366         String[] fileTypes = new String[] { ext };
367         String filename = "";
368         JFileChooser fc = new JFileChooser();
369         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
370         fc.addChoosableFileFilter(FileFilter);
371         fc.setFileFilter(FileFilter);
372         int option = fc.showOpenDialog(this);
373         if (option == JFileChooser.APPROVE_OPTION) {
374             if (fc.getSelectedFile() != null)
375                 filename = fc.getSelectedFile().getPath();
376         } else
377             return "";
378
379         String InputText = "";
380         try {
381             BufferedReader br = new BufferedReader(new FileReader(filename));
382             char[] cbuf = new char[10000];
383
384             int numchars = br.read(cbuf, 0, 10000);
385             while (numchars > 0) {
386                 InputText += (new String(cbuf, 0, numchars));
387                 numchars = br.read(cbuf, 0, 10000);
388             }
389             br.close();
390             currentFileName = filename;
391         } catch (Exception e) {
392             JOptionPane.showMessageDialog(this, "IO Error:  " + e.toString() + "\nCould
                    not open file.", "IO Error",
393                     JOptionPane.WARNING_MESSAGE);
394             InputText = "";
395             currentFileName = "";
396         }
397         return InputText;
398     }
399
400     public void onTextFileSaveAs(String InfoString, String FileTypeName, String ext) {
401         String[] fileTypes = new String[] { ext };
402         JFileChooser fc = new JFileChooser();
403         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
404         fc.addChoosableFileFilter(FileFilter);
405         fc.setFileFilter(FileFilter);
406         fc.setDialogTitle("Save As");
407         int option = fc.showSaveDialog(this);
408         if (option == JFileChooser.APPROVE_OPTION) {
409             if (fc.getSelectedFile() != null) {
410                 String filename = fc.getSelectedFile().getPath();
411                 filename = filename.trim();
412                 if (!filename.toLowerCase().endsWith("." + ext))
413                     filename = filename + "." + ext;
414
415                 boolean okToSave = true;
416                 if (FileExists(filename)) {
417                     okToSave = false;
418                     int ans = JOptionPane.showConfirmDialog(this,
419                             "The file " + filename + " already exists. \nDo you wish to
                                overwrite the file?",
420                             "Overwrite File", JOptionPane.YES_NO_OPTION, JOptionPane.
                                QUESTION_MESSAGE);
421                     if (ans == JOptionPane.YES_OPTION)
422                         okToSave = true;
```

```
423                 }
424
425                 if (okToSave) {
426                     saveFile(filename, InfoString);
427                 }
428             }
429         }
430     }
431
432     public boolean FileExists(String testFilename) {
433         boolean retval = false;
434         try {
435             BufferedReader br = new BufferedReader(new FileReader(testFilename));
436             br.close();
437             retval = true;
438         } catch (Exception e) {
439             retval = false;
440         }
441         return retval;
442     }
443
444     private void saveFile(String filename, String infoStr) {
445         try {
446             PrintWriter outputfile = new PrintWriter(new FileWriter(filename));
447             outputfile.print(infoStr);
448             outputfile.close();
449             currentFileName = filename;
450         } catch (Exception e) {
451             JOptionPane.showMessageDialog(this, "The file could not be saved.", "IO Error"
                    ,
452                     JOptionPane.WARNING_MESSAGE);
453             File f = new File(filename);
454             try {
455                 f.delete();
456             } catch (Exception delex) {
457             }
458             currentFileName = "";
459         }
460     }
461
462     public void onShowStatistics() {
463         JPanel statsPanel = new JPanel();
464         statsPanel.setLayout(new BoxLayout(statsPanel, BoxLayout.Y_AXIS));
465         String inputText = TextEditor.getText();
466
467         String inputTextForLines = TextEditor.getText();
468         String[] splitInput = inputTextForLines.split("\n");
469         int numlines = splitInput.length;
470
471         if (numlines == 1)
472             if (splitInput[0].trim().length() == 0)
473                 numlines = 0;
474
475         int numnonblanklines = 0;
476         if (numlines > 0)
477             for (int i = 0; i < splitInput.length; i++)
478                 if (splitInput[i].trim().length() > 0)
479                     numnonblanklines++;
480
481         inputText = inputText.replaceAll("\t", "");
482         inputText = inputText.replaceAll("\n", "");
483         int numchars = inputText.length();
484         String inputTextTrim = inputText.replaceAll(" ", "");
485         int numcharsNoSpace = inputTextTrim.length();
```

```
486
487         JPanel numLinesPanel = new JPanel();
488         numLinesPanel.setLayout(new BoxLayout(numLinesPanel, BoxLayout.X_AXIS));
489         numLinesPanel.add(new JLabel("Number of Lines: " + numlines));
490         numLinesPanel.add(Box.createHorizontalGlue());
491
492         JPanel numNonBlankLinesPanel = new JPanel();
493         numNonBlankLinesPanel.setLayout(new BoxLayout(numNonBlankLinesPanel, BoxLayout.
                X_AXIS));
494         numNonBlankLinesPanel.add(new JLabel("Number of Non-Blank Lines: " +
                numnonblanklines));
495         numNonBlankLinesPanel.add(Box.createHorizontalGlue());
496
497         JPanel numcharsPanel = new JPanel();
498         numcharsPanel.setLayout(new BoxLayout(numcharsPanel, BoxLayout.X_AXIS));
499         numcharsPanel.add(new JLabel("Number of Characters (Including Spaces): " +
                numchars));
500         numcharsPanel.add(Box.createHorizontalGlue());
501
502         JPanel numcharsPanelNoSpace = new JPanel();
503         numcharsPanelNoSpace.setLayout(new BoxLayout(numcharsPanelNoSpace, BoxLayout.
                X_AXIS));
504         numcharsPanelNoSpace.add(new JLabel("Number of Characters (Excluding Spaces): " +
                numcharsNoSpace));
505         numcharsPanelNoSpace.add(Box.createHorizontalGlue());
506
507         String inputTextForWords = TextEditor.getText();
508         inputTextForWords = inputTextForWords.replaceAll("\n", " ");
509         inputTextForWords = inputTextForWords.replaceAll("\t", " ");
510         String[] words = inputTextForWords.split(" ");
511
512         int numwords = 0;
513         for (int i = 0; i < words.length; i++)
514             if (!words[i].trim().equalsIgnoreCase(""))
515                 numwords++;
516
517         JPanel numwordsPanel = new JPanel();
518         numwordsPanel.setLayout(new BoxLayout(numwordsPanel, BoxLayout.X_AXIS));
519         numwordsPanel.add(new JLabel("Number of Words: " + numwords));
520         numwordsPanel.add(Box.createHorizontalGlue());
521
522         statsPanel.add(numLinesPanel);
523         statsPanel.add(numNonBlankLinesPanel);
524         statsPanel.add(numcharsPanel);
525         statsPanel.add(numcharsPanelNoSpace);
526         statsPanel.add(numwordsPanel);
527
528         JOptionPane.showMessageDialog(this, statsPanel, "Statistics", JOptionPane.
                PLAIN_MESSAGE);
529     }
530 }
```

16.10 JavaPad Step #9: Printing

This is another lengthy addition. Creating a link to a printer and constructing a print preview dialog can be a daunting task. When you are creating your own programs that are complex and you want a feature you do not know how to create, the internet is a good place to go to find example code and components you can add to your program.

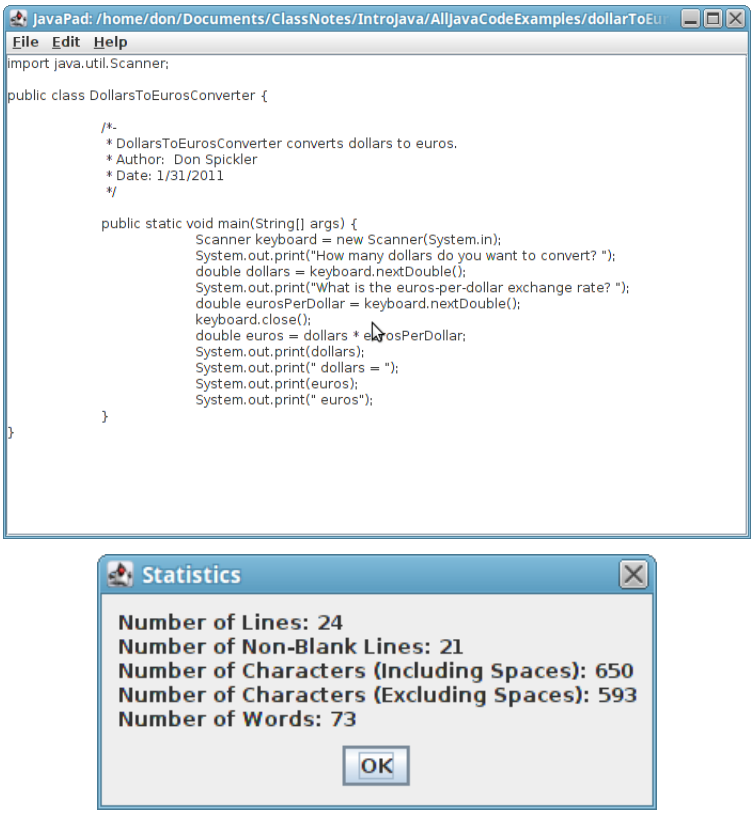


Figure 16.7: JavaPad.java Output: Draft #8

If you want a particular feature, it is possible that someone else has already written it in the same language you are working and was kind enough to post it on a forum or blog. In this case you do not need to write the code but instead just integrate it into your application or alter it to your needs. As is the case for the PrintPreview class we use here. It is a combination of about five print preview examples from the web, in addition to fixing a couple errors from that code and making it easily extendable to add in option panels on the left or right. We have added three more classes to the application. The PrintPreview class that is the main print preview dialog, the TextPrinter class that reformats the text to fit in the printable region of the page and breaks the pages where needed, which is more difficult than it sounds, and the ToolbarButton class for creating the buttons for the toolbar in the PrintPreview dialog. We will use the ToolbarButton class again in the last draft of this construction to add a toolbar to our application. These classes can be found in the appendices.

In the main program file (JavaPad.java), in addition to adding a menu item for printing we added the method `onPrintPreview` that sets up the print job using TextPrinter and sends it to the preview dialog PrintPreview. From there, the PrintPreview class handles everything for sending the previewed print job to a printer.

```
1 import java.awt.*;
```

```
2  import java.awt.event.*;
3  import java.awt.print.Book;
4  import java.awt.print.PageFormat;
5  import java.awt.print.PrinterJob;
6  import java.io.*;
7
8  import javax.print.attribute.HashPrintRequestAttributeSet;
9  import javax.swing.*;
10 import javax.swing.event.*;
11 import javax.swing.text.Document;
12 import javax.swing.undo.*;
13
14 /*-
15  * JavaPad
16  * Added to the application:
17  * 1. The JTextArea
18  * 2. The Menu
19  * 3. The About Screen
20  * 4. Cut, Copy, and Paste
21  * 5. Select All
22  * 6. File transfer and choosing
23  * 7. Undo, Redo, and file properties
24  * 8. Print Preview and Printing
25  *
26  * Author:  Don Spickler
27  * Date: 7/6/2016
28  */
29
30 public class JavaPad extends JFrame implements WindowListener {
31
32     private JTextArea TextEditor;
33     private JMenuBar MainMenu;
34     private boolean wrapText;
35     private String currentFileName;
36     private UndoManager undoManager;
37
38     public static void main(String[] args) {
39         JavaPad prog = new JavaPad(args);
40
41         prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
42         prog.setBounds(20, 20, 700, 500);
43         prog.setVisible(true);
44         prog.toFront();
45     }
46
47     public JavaPad(String[] args) {
48         addWindowListener(this);
49         currentFileName = "";
50         updateTitle();
51
52         TextEditor = new JTextArea();
53         JScrollPane TextEditorSP = new JScrollPane(TextEditor);
54
55         wrapText = true;
56         TextEditor.setLineWrap(wrapText);
57         TextEditor.setWrapStyleWord(wrapText);
58
59         getContentPane().setLayout(new BorderLayout());
60         getContentPane().add(TextEditorSP, BorderLayout.CENTER);
61
62         createMenu();
63         setJMenuBar(MainMenu);
64
65         undoManager = new UndoManager();
```

```
66          undoManager.setLimit(-1); // Unlimited
67
68          Document notedoc = TextEditor.getDocument();
69          notedoc.addUndoableEditListener(new UndoableEditListener() {
70              public void undoableEditHappened(UndoableEditEvent e) {
71                  undoManager.addEdit(e.getEdit());
72              }
73          });
74
75          TextEditor.addKeyListener(new KeyListener() {
76              public void keyPressed(KeyEvent e) {
77
78                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
79                      if (e.getKeyCode() == KeyEvent.VK_Z) {
80                          try {
81                              if (undoManager.canUndo()) {
82                                  undoManager.undo();
83                              }
84                          } catch (CannotUndoException exp) {
85                          }
86                      }
87
88                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
89                      if (e.getKeyCode() == KeyEvent.VK_Y) {
90                          try {
91                              if (undoManager.canRedo()) {
92                                  undoManager.redo();
93                              }
94                          } catch (CannotRedoException exp) {
95                          }
96                      }
97              }
98
99              public void keyTyped(KeyEvent e) {
100             }
101
102             public void keyReleased(KeyEvent e) {
103             }
104         });
105
106     }
107
108     private void updateTitle() {
109         if (currentFileName == "")
110             setTitle("JavaPad: Untitled");
111         else
112             setTitle("JavaPad: " + currentFileName);
113     }
114
115     private void createMenu() {
116         MainMenu = new JMenuBar();
117
118         JMenu FileMenu = new JMenu("File");
119         FileMenu.setMnemonic('F');
120
121         JMenu EditMenu = new JMenu("Edit");
122         EditMenu.setMnemonic('E');
123
124         JMenu HelpMenu = new JMenu("Help");
125         HelpMenu.setMnemonic('H');
126
127         JMenuItem NewMenuItem = new JMenuItem("New");
128         NewMenuItem.setMnemonic('n');
129         NewMenuItem.setAccelerator(KeyStroke.getKeyStroke('N', InputEvent.CTRL_DOWN_MASK))
```

```
        ;
130         NewMenuItem.addActionListener(new ActionListener() {
131             public void actionPerformed(ActionEvent evt) {
132                 onNew();
133             }
134         });
135
136         JMenuItem OpenMenuItem = new JMenuItem("Open...");
137         OpenMenuItem.setMnemonic('o');
138         OpenMenuItem.setAccelerator(KeyStroke.getKeyStroke('O', InputEvent.CTRL_DOWN_MASK)
        );
139         OpenMenuItem.addActionListener(new ActionListener() {
140             public void actionPerformed(ActionEvent evt) {
141                 onOpen();
142             }
143         });
144
145         JMenuItem SaveAsMenuItem = new JMenuItem("Save As...");
146         SaveAsMenuItem.setMnemonic('a');
147         SaveAsMenuItem.addActionListener(new ActionListener() {
148             public void actionPerformed(ActionEvent evt) {
149                 onSaveAs();
150             }
151         });
152
153         JMenuItem SaveMenuItem = new JMenuItem("Save");
154         SaveMenuItem.setMnemonic('s');
155         SaveMenuItem.setAccelerator(KeyStroke.getKeyStroke('S', InputEvent.CTRL_DOWN_MASK)
        );
156         SaveMenuItem.addActionListener(new ActionListener() {
157             public void actionPerformed(ActionEvent evt) {
158                 onSave();
159             }
160         });
161
162         JMenuItem PrintMenuItem = new JMenuItem("Print...");
163         PrintMenuItem.setMnemonic('p');
164         PrintMenuItem.setAccelerator(KeyStroke.getKeyStroke('P', InputEvent.CTRL_DOWN_MASK
        ));
165         PrintMenuItem.addActionListener(new ActionListener() {
166             public void actionPerformed(ActionEvent evt) {
167                 onPrintPreview();
168             }
169         });
170
171         JMenuItem StatsMenuItem = new JMenuItem("Properties...");
172         StatsMenuItem.addActionListener(new ActionListener() {
173             public void actionPerformed(ActionEvent evt) {
174                 onShowStatistics();
175             }
176         });
177
178         JMenuItem ExitMenuItem = new JMenuItem("Exit");
179         ExitMenuItem.setMnemonic('x');
180         ExitMenuItem.addActionListener(new ActionListener() {
181             public void actionPerformed(ActionEvent evt) {
182                 windowClosing(null);
183             }
184         });
185
186         JMenuItem UndoMenuItem = new JMenuItem("Undo");
187         UndoMenuItem.addActionListener(new ActionListener() {
188             public void actionPerformed(ActionEvent evt) {
189                 onUndo();
```

```
190             }
191         });
192
193         JMenuItem RedoMenuItem = new JMenuItem("Redo");
194         RedoMenuItem.addActionListener(new ActionListener() {
195             public void actionPerformed(ActionEvent evt) {
196                 onRedo();
197             }
198         });
199
200         JMenuItem CutMenuItem = new JMenuItem("Cut");
201         CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
202         CutMenuItem.addActionListener(new ActionListener() {
203             public void actionPerformed(ActionEvent evt) {
204                 TextEditor.cut();
205             }
206         });
207
208         JMenuItem CopyMenuItem = new JMenuItem("Copy");
209         CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
210         CopyMenuItem.addActionListener(new ActionListener() {
211             public void actionPerformed(ActionEvent evt) {
212                 TextEditor.copy();
213             }
214         });
215
216         JMenuItem PasteMenuItem = new JMenuItem("Paste");
217         PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
218         PasteMenuItem.addActionListener(new ActionListener() {
219             public void actionPerformed(ActionEvent evt) {
220                 TextEditor.paste();
221             }
222         });
223
224         JMenuItem SelectAllMenuItem = new JMenuItem("Select All");
225         SelectAllMenuItem.setAccelerator(KeyStroke.getKeyStroke('A', InputEvent.
                CTRL_DOWN_MASK));
226         SelectAllMenuItem.addActionListener(new ActionListener() {
227             public void actionPerformed(ActionEvent evt) {
228                 TextEditor.selectAll();
229             }
230         });
231
232         JMenuItem ToggleLineWrapMenuItem = new JMenuItem("Toggle Line Wrap Mode");
233         ToggleLineWrapMenuItem.addActionListener(new ActionListener() {
234             public void actionPerformed(ActionEvent evt) {
235                 wrapText = !wrapText;
236                 TextEditor.setLineWrap(wrapText);
237                 TextEditor.setWrapStyleWord(wrapText);
238             }
239         });
240
241         JMenuItem AboutMenuItem = new JMenuItem("About JavaPad");
242         AboutMenuItem.setMnemonic('A');
243         AboutMenuItem.addActionListener(new ActionListener() {
244             public void actionPerformed(ActionEvent evt) {
245                 onShowAbout();
246             }
247         });
248
249         FileMenu.add(NewMenuItem);
```

```
250         FileMenu.add(OpenMenuItem);
251         FileMenu.addSeparator();
252         FileMenu.add(SaveMenuItem);
253         FileMenu.add(SaveAsMenuItem);
254         FileMenu.addSeparator();
255         FileMenu.add(PrintMenuItem);
256         FileMenu.addSeparator();
257         FileMenu.add(StatsMenuItem);
258         FileMenu.addSeparator();
259         FileMenu.add(ExitMenuItem);
260
261         EditMenu.add(UndoMenuItem);
262         EditMenu.add(RedoMenuItem);
263         EditMenu.addSeparator();
264         EditMenu.add(CutMenuItem);
265         EditMenu.add(CopyMenuItem);
266         EditMenu.add(PasteMenuItem);
267         EditMenu.addSeparator();
268         EditMenu.add(SelectAllMenuItem);
269         EditMenu.addSeparator();
270         EditMenu.add(ToggleLineWrapMenuItem);
271
272         HelpMenu.add(AboutMenuItem);
273
274         MainMenu.add(FileMenu);
275         MainMenu.add(EditMenu);
276         MainMenu.add(HelpMenu);
277     }
278
279     private void onNew() {
280         boolean oktogo = true;
281
282         if (!TextEditor.getText().isEmpty()) {
283             oktogo = false;
284             int ans = JOptionPane.showConfirmDialog(this,
285                     "This will clear all of the text in the editor.\nDo you wish to delete
                            the current text?",
286                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                            WARNING_MESSAGE);
287             if (ans == JOptionPane.YES_OPTION)
288                 oktogo = true;
289         }
290
291         if (!oktogo)
292             return;
293
294         TextEditor.setText("");
295         currentFileName = "";
296         updateTitle();
297     }
298
299     private void onOpen() {
300         boolean oktogo = true;
301
302         if (!TextEditor.getText().isEmpty()) {
303             oktogo = false;
304             int ans = JOptionPane.showConfirmDialog(this,
305                     "This will replace all of the text in the editor with the file
                            contents.\nDo you wish to overwrite the current text?",
306                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                            WARNING_MESSAGE);
307             if (ans == JOptionPane.YES_OPTION)
308                 oktogo = true;
309         }
```

```
310
311         if (!oktogo)
312             return;
313
314         String filetext = OpenTextFile("Text Files", "txt");
315         TextEditor.setText(filetext);
316         TextEditor.setCaretPosition(0);
317         updateTitle();
318     }
319
320     private void onSaveAs() {
321         onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
322         updateTitle();
323     }
324
325     private void onSave() {
326         if (currentFileName == "")
327             onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
328         else
329             saveFile(currentFileName, TextEditor.getText());
330
331         updateTitle();
332     }
333
334     private void onUndo() {
335         try {
336             if (undoManager.canUndo()) {
337                 undoManager.undo();
338             }
339         } catch (CannotUndoException exp) {
340         }
341         TextEditor.requestFocus();
342     }
343
344     private void onRedo() {
345         try {
346             if (undoManager.canRedo()) {
347                 undoManager.redo();
348             }
349         } catch (CannotUndoException exp) {
350         }
351         TextEditor.requestFocus();
352     }
353
354     private void onShowAbout() {
355         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
              2016", "About JavaPad",
356                 JOptionPane.INFORMATION_MESSAGE);
357     }
358
359     public void windowActivated(WindowEvent e) {
360     }
361
362     public void windowClosed(WindowEvent e) {
363     }
364
365     public void windowClosing(WindowEvent e) {
366         System.exit(0);
367     }
368
369     public void windowDeactivated(WindowEvent e) {
370     }
371
372     public void windowDeiconified(WindowEvent e) {
```

```
373     }
374
375     public void windowIconified(WindowEvent e) {
376     }
377
378     public void windowOpened(WindowEvent e) {
379     }
380
381     public String OpenTextFile(String FileTypeName, String ext) {
382         String[] fileTypes = new String[] { ext };
383         String filename = "";
384         JFileChooser fc = new JFileChooser();
385         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
386         fc.addChoosableFileFilter(FileFilter);
387         fc.setFileFilter(FileFilter);
388         int option = fc.showOpenDialog(this);
389         if (option == JFileChooser.APPROVE_OPTION) {
390             if (fc.getSelectedFile() != null)
391                 filename = fc.getSelectedFile().getPath();
392         } else
393             return "";
394
395         String InputText = "";
396         try {
397             BufferedReader br = new BufferedReader(new FileReader(filename));
398             char[] cbuf = new char[10000];
399
400             int numchars = br.read(cbuf, 0, 10000);
401             while (numchars > 0) {
402                 InputText += (new String(cbuf, 0, numchars));
403                 numchars = br.read(cbuf, 0, 10000);
404             }
405             br.close();
406             currentFileName = filename;
407         } catch (Exception e) {
408             JOptionPane.showMessageDialog(this, "IO Error:  " + e.toString() + "\nCould
                    not open file.", "IO Error",
409                     JOptionPane.WARNING_MESSAGE);
410             InputText = "";
411             currentFileName = "";
412         }
413         return InputText;
414     }
415
416     public void onTextFileSaveAs(String InfoString, String FileTypeName, String ext) {
417         String[] fileTypes = new String[] { ext };
418         JFileChooser fc = new JFileChooser();
419         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
420         fc.addChoosableFileFilter(FileFilter);
421         fc.setFileFilter(FileFilter);
422         fc.setDialogTitle("Save As");
423         int option = fc.showSaveDialog(this);
424         if (option == JFileChooser.APPROVE_OPTION) {
425             if (fc.getSelectedFile() != null) {
426                 String filename = fc.getSelectedFile().getPath();
427                 filename = filename.trim();
428                 if (!filename.toLowerCase().endsWith("." + ext))
429                     filename = filename + "." + ext;
430
431                 boolean okToSave = true;
432                 if (FileExists(filename)) {
433                     okToSave = false;
```

```
434                     int ans = JOptionPane.showConfirmDialog(this,
435                             "The file " + filename + " already exists. \nDo you wish to
                                overwrite the file?",
436                             "Overwrite File", JOptionPane.YES_NO_OPTION, JOptionPane.
                                QUESTION_MESSAGE);
437                     if (ans == JOptionPane.YES_OPTION)
438                         okToSave = true;
439                 }
440
441                 if (okToSave) {
442                     saveFile(filename, InfoString);
443                 }
444             }
445         }
446     }
447
448     public boolean FileExists(String testFilename) {
449         boolean retval = false;
450         try {
451             BufferedReader br = new BufferedReader(new FileReader(testFilename));
452             br.close();
453             retval = true;
454         } catch (Exception e) {
455             retval = false;
456         }
457         return retval;
458     }
459
460     private void saveFile(String filename, String infoStr) {
461         try {
462             PrintWriter outputfile = new PrintWriter(new FileWriter(filename));
463             outputfile.print(infoStr);
464             outputfile.close();
465             currentFileName = filename;
466         } catch (Exception e) {
467             JOptionPane.showMessageDialog(this, "The file could not be saved.", "IO Error"
                    ,
468                     JOptionPane.WARNING_MESSAGE);
469             File f = new File(filename);
470             try {
471                 f.delete();
472             } catch (Exception delex) {
473             }
474             currentFileName = "";
475         }
476     }
477
478     public void onShowStatistics() {
479         JPanel statsPanel = new JPanel();
480         statsPanel.setLayout(new BoxLayout(statsPanel, BoxLayout.Y_AXIS));
481         String inputText = TextEditor.getText();
482
483         String inputTextForLines = TextEditor.getText();
484         String[] splitInput = inputTextForLines.split("\n");
485         int numlines = splitInput.length;
486
487         if (numlines == 1)
488             if (splitInput[0].trim().length() == 0)
489                 numlines = 0;
490
491         int numnonblanklines = 0;
492         if (numlines > 0)
493             for (int i = 0; i < splitInput.length; i++)
494                 if (splitInput[i].trim().length() > 0)
```

```
495                     numnonblanklines++;
496
497         inputText = inputText.replaceAll("\t", "");
498         inputText = inputText.replaceAll("\n", "");
499         int numchars = inputText.length();
500         String inputTextTrim = inputText.replaceAll(" ", "");
501         int numcharsNoSpace = inputTextTrim.length();
502
503         JPanel numLinesPanel = new JPanel();
504         numLinesPanel.setLayout(new BoxLayout(numLinesPanel, BoxLayout.X_AXIS));
505         numLinesPanel.add(new JLabel("Number of Lines: " + numlines));
506         numLinesPanel.add(Box.createHorizontalGlue());
507
508         JPanel numNonBlankLinesPanel = new JPanel();
509         numNonBlankLinesPanel.setLayout(new BoxLayout(numNonBlankLinesPanel, BoxLayout.
                X_AXIS));
510         numNonBlankLinesPanel.add(new JLabel("Number of Non-Blank Lines: " +
                numnonblanklines));
511         numNonBlankLinesPanel.add(Box.createHorizontalGlue());
512
513         JPanel numcharsPanel = new JPanel();
514         numcharsPanel.setLayout(new BoxLayout(numcharsPanel, BoxLayout.X_AXIS));
515         numcharsPanel.add(new JLabel("Number of Characters (Including Spaces): " +
                numchars));
516         numcharsPanel.add(Box.createHorizontalGlue());
517
518         JPanel numcharsPanelNoSpace = new JPanel();
519         numcharsPanelNoSpace.setLayout(new BoxLayout(numcharsPanelNoSpace, BoxLayout.
                X_AXIS));
520         numcharsPanelNoSpace.add(new JLabel("Number of Characters (Excluding Spaces): " +
                numcharsNoSpace));
521         numcharsPanelNoSpace.add(Box.createHorizontalGlue());
522
523         String inputTextForWords = TextEditor.getText();
524         inputTextForWords = inputTextForWords.replaceAll("\n", " ");
525         inputTextForWords = inputTextForWords.replaceAll("\t", " ");
526         String[] words = inputTextForWords.split(" ");
527
528         int numwords = 0;
529         for (int i = 0; i < words.length; i++)
530             if (!words[i].trim().equalsIgnoreCase(""))
531                 numwords++;
532
533         JPanel numwordsPanel = new JPanel();
534         numwordsPanel.setLayout(new BoxLayout(numwordsPanel, BoxLayout.X_AXIS));
535         numwordsPanel.add(new JLabel("Number of Words: " + numwords));
536         numwordsPanel.add(Box.createHorizontalGlue());
537
538         statsPanel.add(numLinesPanel);
539         statsPanel.add(numNonBlankLinesPanel);
540         statsPanel.add(numcharsPanel);
541         statsPanel.add(numcharsPanelNoSpace);
542         statsPanel.add(numwordsPanel);
543
544         JOptionPane.showMessageDialog(this, statsPanel, "Statistics", JOptionPane.
                PLAIN_MESSAGE);
545     }
546
547     public void onPrintPreview() {
548         if (TextEditor.getText().trim().length() == 0)
549             return;
550
551         try {
552             PrinterJob pj = PrinterJob.getPrinterJob();
```

```
553             HashPrintRequestAttributeSet pra = new HashPrintRequestAttributeSet();
554             Book bk = new Book();
555             PageFormat thisFormat = pj.defaultPage();
556             String printtitle = currentFileName;
557             if (printtitle == "")
558                 printtitle = "Untitled";
559
560             TextPrinter tp = new TextPrinter(thisFormat, printtitle, "", TextEditor.
                    getText(), 10, TextEditor.getFont(),
561                     TextEditor.getTabSize());
562             PrintPreview np = new PrintPreview(tp, thisFormat, "Print Preview");
563         } catch (Exception e) {
564             JOptionPane.showMessageDialog(this, "Cannot print current document.", "Print
                    Error",
565                     JOptionPane.WARNING_MESSAGE);
566         }
567     }
568
569 }
```

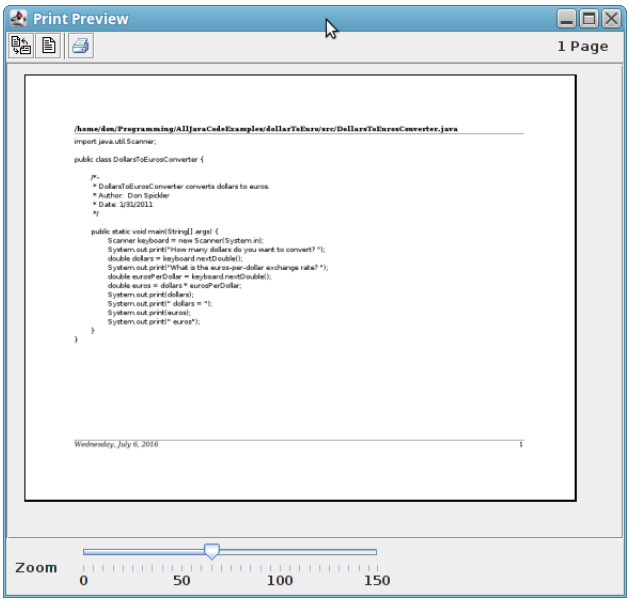


Figure 16.8: JavaPad.java Output: Draft #9

16.11 JavaPad Step #10: Toolbar

In addition to adding a toolbar to the application we changed the font on the text editor to be more like a monospaced typewriter font. We also changed the default number of spaces used for a tab character to 4 to make the display of program code a little easier to read. Toolbars are fairly easy to create, essentially they are just menus and their code is very similar to a menu. To create a toolbar, we do the following,

1. Create a new JToolBar.

2. Create ToolbarButton items for each feature.
3. Add the ToolbarButton to the JToolBar.
4. Add the JToolBar to the application by placing it in the North part of the boarder layout for the content pane.

Creating the JToolBar is done with,

```
MainTools = new JToolBar("JavaPad Tools");
MainTools.setFloatable(false);
```

Note that if we set Floatable to true the user will be able to grab the toolbar and remove it from the docking structure. We decided to keep it fixed. We create the ToolbarButton with the commands,

```
ToolbarButton NewTool = new ToolbarButton(new ImageIcon(cl.
   getResource("FileNewImage.GIF")), "New", 25, 25);
NewTool.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        onNew();
    }
});
```

Here the image FileNewImage.GIF will be used on the toolbar, it will have a tool tip of New and the size of the button will be 25 X 25 pixels. We then associate an action with the button that calls the method `onNew`, just like the menu did. A word about images. Each button in the toolbar needs to have an image, these are created by the programmer and loaded into the application. I usually have a subfolder of the project folder (same level as src and bin) that contains all of my tool images. Most of them I simply took from s screen shots of other applications and altered them slightly. You can bring in other file types for the images, like jpeg, png and bmp, but gif and png tend to work better since you can make a color transparent in these types. The folder that you place all of your images in must be added to the project properties for the Java Build Path. Select the project properties, click on the Java Build Path and then the Source tab. Click the Add Folder... button and add in the images folder.

Once this is done then the class loader can find the folder. Note the use of the variable `cl` in the ImageIcon creation above. The variable `cl` is a class loader type with the declaration,

```
public ClassLoader cl = this.getClass().getClassLoader();
```

Using this setup, the program will be able to find the images if they are in the project folder or if they are in a java archive file (JAR). This makes it convenient if you are going to distribute your application. To add the button to the toolbar, just use the add method,

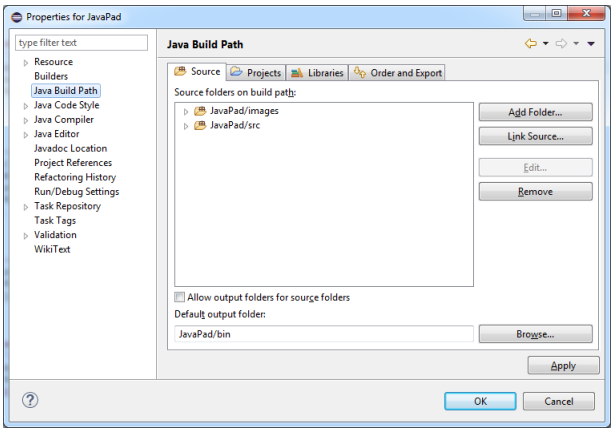


Figure 16.9: Image Folder Inclusion

```
MainTools.add(NewTool);
```

Finally, to put the toolbar on the application add it to the North, and we are done.

```
getContentPane().add(MainTools, BorderLayout.NORTH);
```

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.awt.print.*;
4  import java.io.*;
5
6  import javax.print.attribute.HashPrintRequestAttributeSet;
7  import javax.swing.*;
8  import javax.swing.event.*;
9  import javax.swing.text.*;
10 import javax.swing.undo.*;
11
12 /*-
13  * JavaPad
14  * Added to the application:
15  * 1. The JTextArea
16  * 2. The Menu
17  * 3. The About Screen
18  * 4. Cut, Copy, and Paste
19  * 5. Select All
20  * 6. File transfer and choosing
21  * 7. Undo, Redo, and file properties
22  * 8. Print Preview and Printing
23  * 9. Toolbar
24  *
25  * Author:  Don Spickler
26  * Date: 7/6/2016
27  */
28
29 public class JavaPad extends JFrame implements WindowListener {
30
31     private JTextArea TextEditor;
32     private JMenuBar MainMenu;
33     private JToolBar MainTools;
34     private boolean wrapText;
35     private String currentFileName;
```

```
36      private UndoManager undoManager;
37      public ClassLoader cl = this.getClass().getClassLoader();
38
39      public static void main(String[] args) {
40          JavaPad prog = new JavaPad(args);
41
42          prog.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
43          prog.setBounds(20, 20, 700, 500);
44          prog.setVisible(true);
45          prog.toFront();
46      }
47
48      public JavaPad(String[] args) {
49          addWindowListener(this);
50          currentFileName = "";
51          updateTitle();
52
53          TextEditor = new JTextArea();
54          JScrollPane TextEditorSP = new JScrollPane(TextEditor);
55
56          wrapText = true;
57          TextEditor.setLineWrap(wrapText);
58          TextEditor.setWrapStyleWord(wrapText);
59          TextEditor.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 14));
60          TextEditor.setTabSize(4);
61
62          createMenu();
63          createToolBar();
64          setJMenuBar(MainMenu);
65
66          getContentPane().setLayout(new BorderLayout());
67          getContentPane().add(TextEditorSP, BorderLayout.CENTER);
68          getContentPane().add(MainTools, BorderLayout.NORTH);
69
70          undoManager = new UndoManager();
71          undoManager.setLimit(-1); // Unlimited
72
73          Document notedoc = TextEditor.getDocument();
74          notedoc.addUndoableEditListener(new UndoableEditListener() {
75              public void undoableEditHappened(UndoableEditEvent e) {
76                  undoManager.addEdit(e.getEdit());
77              }
78          });
79
80          TextEditor.addKeyListener(new KeyListener() {
81              public void keyPressed(KeyEvent e) {
82
83                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
84                      if (e.getKeyCode() == KeyEvent.VK_Z) {
85                          try {
86                              if (undoManager.canUndo()) {
87                                  undoManager.undo();
88                              }
89                          } catch (CannotUndoException exp) {
90                          }
91                      }
92
93                  if (e.getModifiersEx() == KeyEvent.CTRL_DOWN_MASK)
94                      if (e.getKeyCode() == KeyEvent.VK_Y) {
95                          try {
96                              if (undoManager.canRedo()) {
97                                  undoManager.redo();
98                              }
99                          } catch (CannotRedoException exp) {
```

```
100                         }
101                     }
102             }
103
104             public void keyTyped(KeyEvent e) {
105             }
106
107             public void keyReleased(KeyEvent e) {
108             }
109         });
110
111     }
112
113     private void updateTitle() {
114         if (currentFileName == "")
115             setTitle("JavaPad: Untitled");
116         else
117             setTitle("JavaPad: " + currentFileName);
118     }
119
120     private void createMenu() {
121         MainMenu = new JMenuBar();
122
123         JMenu FileMenu = new JMenu("File");
124         FileMenu.setMnemonic('F');
125
126         JMenu EditMenu = new JMenu("Edit");
127         EditMenu.setMnemonic('E');
128
129         JMenu HelpMenu = new JMenu("Help");
130         HelpMenu.setMnemonic('H');
131
132         JMenuItem NewMenuItem = new JMenuItem("New", new ImageIcon(cl.getResource("
                FileNewImage.GIF")));
133         NewMenuItem.setMnemonic('n');
134         NewMenuItem.setAccelerator(KeyStroke.getKeyStroke('N', InputEvent.CTRL_DOWN_MASK))
                ;
135         NewMenuItem.addActionListener(new ActionListener() {
136             public void actionPerformed(ActionEvent evt) {
137                 onNew();
138             }
139         });
140
141         JMenuItem OpenMenuItem = new JMenuItem("Open...", new ImageIcon(cl.getResource("
                FileOpenImage.GIF")));
142         OpenMenuItem.setMnemonic('o');
143         OpenMenuItem.setAccelerator(KeyStroke.getKeyStroke('O', InputEvent.CTRL_DOWN_MASK)
                );
144         OpenMenuItem.addActionListener(new ActionListener() {
145             public void actionPerformed(ActionEvent evt) {
146                 onOpen();
147             }
148         });
149
150         JMenuItem SaveAsMenuItem = new JMenuItem("Save As...", new ImageIcon(cl.
                getResource("FileSaveAsImage.GIF")));
151         SaveAsMenuItem.setMnemonic('a');
152         SaveAsMenuItem.addActionListener(new ActionListener() {
153             public void actionPerformed(ActionEvent evt) {
154                 onSaveAs();
155             }
156         });
157
158         JMenuItem SaveMenuItem = new JMenuItem("Save", new ImageIcon(cl.getResource("
```

```
            FileSaveImage.GIF")));
159         SaveMenuItem.setMnemonic('s');
160         SaveMenuItem.setAccelerator(KeyStroke.getKeyStroke('S', InputEvent.CTRL_DOWN_MASK)
                );
161         SaveMenuItem.addActionListener(new ActionListener() {
162             public void actionPerformed(ActionEvent evt) {
163                 onSave();
164             }
165         });
166
167         JMenuItem PrintMenuItem = new JMenuItem("Print...", new ImageIcon(cl.getResource("
                PrintPreviewIcon.GIF")));
168         PrintMenuItem.setMnemonic('p');
169         PrintMenuItem.setAccelerator(KeyStroke.getKeyStroke('P', InputEvent.CTRL_DOWN_MASK
                ));
170         PrintMenuItem.addActionListener(new ActionListener() {
171             public void actionPerformed(ActionEvent evt) {
172                 onPrintPreview();
173             }
174         });
175
176         JMenuItem StatsMenuItem = new JMenuItem("Properties...", new ImageIcon(cl.
                getResource("BarChartImage.GIF")));
177         StatsMenuItem.addActionListener(new ActionListener() {
178             public void actionPerformed(ActionEvent evt) {
179                 onShowStatistics();
180             }
181         });
182
183         JMenuItem ExitMenuItem = new JMenuItem("Exit", new ImageIcon(cl.getResource("
                RemoveImage.GIF")));
184         ExitMenuItem.setMnemonic('x');
185         ExitMenuItem.addActionListener(new ActionListener() {
186             public void actionPerformed(ActionEvent evt) {
187                 windowClosing(null);
188             }
189         });
190
191         JMenuItem UndoMenuItem = new JMenuItem("Undo", new ImageIcon(cl.getResource("Undo.
                GIF")));
192         UndoMenuItem.addActionListener(new ActionListener() {
193             public void actionPerformed(ActionEvent evt) {
194                 onUndo();
195             }
196         });
197
198         JMenuItem RedoMenuItem = new JMenuItem("Redo", new ImageIcon(cl.getResource("Redo.
                GIF")));
199         RedoMenuItem.addActionListener(new ActionListener() {
200             public void actionPerformed(ActionEvent evt) {
201                 onRedo();
202             }
203         });
204
205         JMenuItem CutMenuItem = new JMenuItem("Cut", new ImageIcon(cl.getResource("
                CutImage.GIF")));
206         CutMenuItem.setAccelerator(KeyStroke.getKeyStroke('X', InputEvent.CTRL_DOWN_MASK))
                ;
207         CutMenuItem.addActionListener(new ActionListener() {
208             public void actionPerformed(ActionEvent evt) {
209                 TextEditor.cut();
210             }
211         });
212
```

```
213         JMenuItem CopyMenuItem = new JMenuItem("Copy", new ImageIcon(cl.getResource("
                CopyImage.GIF")));
214         CopyMenuItem.setAccelerator(KeyStroke.getKeyStroke('C', InputEvent.CTRL_DOWN_MASK)
                );
215         CopyMenuItem.addActionListener(new ActionListener() {
216             public void actionPerformed(ActionEvent evt) {
217                 TextEditor.copy();
218             }
219         });
220
221         JMenuItem PasteMenuItem = new JMenuItem("Paste", new ImageIcon(cl.getResource("
                PasteImage.GIF")));
222         PasteMenuItem.setAccelerator(KeyStroke.getKeyStroke('V', InputEvent.CTRL_DOWN_MASK
                ));
223         PasteMenuItem.addActionListener(new ActionListener() {
224             public void actionPerformed(ActionEvent evt) {
225                 TextEditor.paste();
226             }
227         });
228
229         JMenuItem SelectAllMenuItem = new JMenuItem("Select All", new ImageIcon(cl.
                getResource("CopyAllImage.GIF")));
230         SelectAllMenuItem.setAccelerator(KeyStroke.getKeyStroke('A', InputEvent.
                CTRL_DOWN_MASK));
231         SelectAllMenuItem.addActionListener(new ActionListener() {
232             public void actionPerformed(ActionEvent evt) {
233                 TextEditor.requestFocus();
234                 TextEditor.selectAll();
235             }
236         });
237
238         JMenuItem ToggleLineWrapMenuItem = new JMenuItem("Toggle Line Wrap Mode",
239                 new ImageIcon(cl.getResource("NewLineToSpaceIcon.GIF")));
240         ToggleLineWrapMenuItem.addActionListener(new ActionListener() {
241             public void actionPerformed(ActionEvent evt) {
242                 wrapText = !wrapText;
243                 TextEditor.setLineWrap(wrapText);
244                 TextEditor.setWrapStyleWord(wrapText);
245             }
246         });
247
248         JMenuItem AboutMenuItem = new JMenuItem("About JavaPad", new ImageIcon(cl.
                getResource("About.PNG")));
249         AboutMenuItem.setMnemonic('A');
250         AboutMenuItem.addActionListener(new ActionListener() {
251             public void actionPerformed(ActionEvent evt) {
252                 onShowAbout();
253             }
254         });
255
256         FileMenu.add(NewMenuItem);
257         FileMenu.add(OpenMenuItem);
258         FileMenu.addSeparator();
259         FileMenu.add(SaveMenuItem);
260         FileMenu.add(SaveAsMenuItem);
261         FileMenu.addSeparator();
262         FileMenu.add(PrintMenuItem);
263         FileMenu.addSeparator();
264         FileMenu.add(StatsMenuItem);
265         FileMenu.addSeparator();
266         FileMenu.add(ExitMenuItem);
267
268         EditMenu.add(UndoMenuItem);
269         EditMenu.add(RedoMenuItem);
```

```
270         EditMenu.addSeparator();
271         EditMenu.add(CutMenuItem);
272         EditMenu.add(CopyMenuItem);
273         EditMenu.add(PasteMenuItem);
274         EditMenu.addSeparator();
275         EditMenu.add(SelectAllMenuItem);
276         EditMenu.add(ToggleLineWrapMenuItem);
277
278         HelpMenu.add(AboutMenuItem);
279
280         MainMenu.add(FileMenu);
281         MainMenu.add(EditMenu);
282         MainMenu.add(HelpMenu);
283     }
284
285     private void createToolBar() {
286         MainTools = new JToolBar("JavaPad Tools");
287         MainTools.setFloatable(false);
288
289         // Define tools
290
291         ToolbarButton NewTool = new ToolbarButton(new ImageIcon(cl.getResource("
                FileNewImage.GIF")), "New", 25, 25);
292         NewTool.addActionListener(new ActionListener() {
293             public void actionPerformed(ActionEvent evt) {
294                 onNew();
295             }
296         });
297
298         ToolbarButton OpenTool = new ToolbarButton(new ImageIcon(cl.getResource("
                FileOpenImage.GIF")), "Open...", 25,
299                 25);
300         OpenTool.addActionListener(new ActionListener() {
301             public void actionPerformed(ActionEvent evt) {
302                 onOpen();
303             }
304         });
305
306         ToolbarButton SaveAsTool = new ToolbarButton(new ImageIcon(cl.getResource("
                FileSaveAsImage.GIF")), "Save As...",
307                 25, 25);
308         SaveAsTool.addActionListener(new ActionListener() {
309             public void actionPerformed(ActionEvent evt) {
310                 onSaveAs();
311             }
312         });
313
314         ToolbarButton SaveTool = new ToolbarButton(new ImageIcon(cl.getResource("
                FileSaveImage.GIF")), "Save", 25, 25);
315         SaveTool.addActionListener(new ActionListener() {
316             public void actionPerformed(ActionEvent evt) {
317                 onSave();
318             }
319         });
320
321         ToolbarButton PrintTool = new ToolbarButton(new ImageIcon(cl.getResource("
                PrintPreviewIcon.GIF")), "Print...",
322                 25, 25);
323         PrintTool.addActionListener(new ActionListener() {
324             public void actionPerformed(ActionEvent evt) {
325                 onPrintPreview();
326             }
327         });
328
```

```
329         ToolbarButton PropertiesTool = new ToolbarButton(new ImageIcon(cl.getResource("
                BarChartImage.GIF")),
330                 "Properties...", 25, 25);
331         PropertiesTool.addActionListener(new ActionListener() {
332             public void actionPerformed(ActionEvent evt) {
333                 onShowStatistics();
334             }
335         });
336 
337         ToolbarButton UndoTool = new ToolbarButton(new ImageIcon(cl.getResource("Undo.GIF"
                )), "Undo", 25, 25);
338         UndoTool.addActionListener(new ActionListener() {
339             public void actionPerformed(ActionEvent evt) {
340                 onUndo();
341             }
342         });
343 
344         ToolbarButton RedoTool = new ToolbarButton(new ImageIcon(cl.getResource("Redo.GIF"
                )), "Redo", 25, 25);
345         RedoTool.addActionListener(new ActionListener() {
346             public void actionPerformed(ActionEvent evt) {
347                 onRedo();
348             }
349         });
350 
351         ToolbarButton CutTool = new ToolbarButton(new ImageIcon(cl.getResource("CutImage.
                GIF")), "Cut", 25, 25);
352         CutTool.addActionListener(new ActionListener() {
353             public void actionPerformed(ActionEvent evt) {
354                 TextEditor.cut();
355             }
356         });
357 
358         ToolbarButton CopyTool = new ToolbarButton(new ImageIcon(cl.getResource("CopyImage
                .GIF")), "Copy", 25, 25);
359         CopyTool.addActionListener(new ActionListener() {
360             public void actionPerformed(ActionEvent evt) {
361                 TextEditor.copy();
362             }
363         });
364 
365         ToolbarButton PasteTool = new ToolbarButton(new ImageIcon(cl.getResource("
                PasteImage.GIF")), "Paste", 25, 25);
366         PasteTool.addActionListener(new ActionListener() {
367             public void actionPerformed(ActionEvent evt) {
368                 TextEditor.paste();
369             }
370         });
371 
372         ToolbarButton SelectAllTool = new ToolbarButton(new ImageIcon(cl.getResource("
                CopyAllImage.GIF")), "Select All",
373                 25, 25);
374         SelectAllTool.addActionListener(new ActionListener() {
375             public void actionPerformed(ActionEvent evt) {
376                 TextEditor.requestFocus();
377                 TextEditor.selectAll();
378             }
379         });
380 
381         ToolbarButton ToggleLineWrapTool = new ToolbarButton(new ImageIcon(cl.getResource(
                "NewLineToSpaceIcon.GIF")),
382                 "Toggle Line Wrap Mode", 25, 25);
383         ToggleLineWrapTool.addActionListener(new ActionListener() {
384             public void actionPerformed(ActionEvent evt) {
```

```
385                 wrapText = !wrapText;
386                 TextEditor.setLineWrap(wrapText);
387                 TextEditor.setWrapStyleWord(wrapText);
388             }
389         });
390 
391         ToolbarButton AboutTool = new ToolbarButton(new ImageIcon(cl.getResource("About.
                PNG")), "About JavaPad", 25,
392                 25);
393         AboutTool.addActionListener(new ActionListener() {
394             public void actionPerformed(ActionEvent evt) {
395                 onShowAbout();
396             }
397         });
398 
399         // Add tools to toolbar
400 
401         MainTools.add(NewTool);
402         MainTools.add(OpenTool);
403         MainTools.add(SaveTool);
404         MainTools.add(SaveAsTool);
405         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
406         MainTools.add(PrintTool);
407         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
408         MainTools.add(PropertiesTool);
409         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
410         MainTools.add(UndoTool);
411         MainTools.add(RedoTool);
412         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
413         MainTools.add(CutTool);
414         MainTools.add(CopyTool);
415         MainTools.add(PasteTool);
416         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
417         MainTools.add(SelectAllTool);
418         MainTools.add(ToggleLineWrapTool);
419         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
420         MainTools.add(AboutTool);
421     }
422 
423     private void onNew() {
424         boolean oktogo = true;
425 
426         if (!TextEditor.getText().isEmpty()) {
427             oktogo = false;
428             int ans = JOptionPane.showConfirmDialog(this,
429                     "This will clear all of the text in the editor.\nDo you wish to delete
                            the current text?",
430                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                            WARNING_MESSAGE);
431             if (ans == JOptionPane.YES_OPTION)
432                 oktogo = true;
433         }
434 
435         if (!oktogo)
436             return;
437 
438         TextEditor.setText("");
439         currentFileName = "";
```

```
440         updateTitle();
441     }
442 
443     private void onOpen() {
444         boolean oktogo = true;
445 
446         if (!TextEditor.getText().isEmpty()) {
447             oktogo = false;
448             int ans = JOptionPane.showConfirmDialog(this,
449                     "This will replace all of the text in the editor with the file
                        contents.\nDo you wish to overwrite the current text?",
450                     "Overwrite Text", JOptionPane.YES_NO_OPTION, JOptionPane.
                        WARNING_MESSAGE);
451             if (ans == JOptionPane.YES_OPTION)
452                 oktogo = true;
453         }
454 
455         if (!oktogo)
456             return;
457 
458         String filetext = OpenTextFile("Text Files", "txt");
459         TextEditor.setText(filetext);
460         TextEditor.setCaretPosition(0);
461         updateTitle();
462     }
463 
464     private void onSaveAs() {
465         onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
466         updateTitle();
467     }
468 
469     private void onSave() {
470         if (currentFileName == "")
471             onTextFileSaveAs(TextEditor.getText(), "Text Files", "txt");
472         else
473             saveFile(currentFileName, TextEditor.getText());
474 
475         updateTitle();
476     }
477 
478     private void onUndo() {
479         try {
480             if (undoManager.canUndo()) {
481                 undoManager.undo();
482             }
483         } catch (CannotUndoException exp) {
484         }
485         TextEditor.requestFocus();
486     }
487 
488     private void onRedo() {
489         try {
490             if (undoManager.canRedo()) {
491                 undoManager.redo();
492             }
493         } catch (CannotUndoException exp) {
494         }
495         TextEditor.requestFocus();
496     }
497 
498     private void onShowAbout() {
499         JOptionPane.showMessageDialog(this, "JavaPad\nWritten by: Don Spickler\nCopyright
                2016", "About JavaPad",
500                 JOptionPane.INFORMATION_MESSAGE);
```

```
501     }
502
503     public void windowActivated(WindowEvent e) {
504     }
505
506     public void windowClosed(WindowEvent e) {
507     }
508
509     public void windowClosing(WindowEvent e) {
510         System.exit(0);
511     }
512
513     public void windowDeactivated(WindowEvent e) {
514     }
515
516     public void windowDeiconified(WindowEvent e) {
517     }
518
519     public void windowIconified(WindowEvent e) {
520     }
521
522     public void windowOpened(WindowEvent e) {
523     }
524
525     public String OpenTextFile(String FileTypeName, String ext) {
526         String[] fileTypes = new String[] { ext };
527         String filename = "";
528         JFileChooser fc = new JFileChooser();
529         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
              " + ext + ")");
530         fc.addChoosableFileFilter(FileFilter);
531         fc.setFileFilter(FileFilter);
532         int option = fc.showOpenDialog(this);
533         if (option == JFileChooser.APPROVE_OPTION) {
534             if (fc.getSelectedFile() != null)
535                 filename = fc.getSelectedFile().getPath();
536         } else
537             return "";
538
539         String InputText = "";
540         try {
541             BufferedReader br = new BufferedReader(new FileReader(filename));
542             char[] cbuf = new char[10000];
543
544             int numchars = br.read(cbuf, 0, 10000);
545             while (numchars > 0) {
546                 InputText += (new String(cbuf, 0, numchars));
547                 numchars = br.read(cbuf, 0, 10000);
548             }
549             br.close();
550             currentFileName = filename;
551         } catch (Exception e) {
552             JOptionPane.showMessageDialog(this, "IO Error:  " + e.toString() + "\nCould
                  not open file.", "IO Error",
553                     JOptionPane.WARNING_MESSAGE);
554             InputText = "";
555             currentFileName = "";
556         }
557         return InputText;
558     }
559
560     public void onTextFileSaveAs(String InfoString, String FileTypeName, String ext) {
561         String[] fileTypes = new String[] { ext };
562         JFileChooser fc = new JFileChooser();
```

```
563         SimpleFileFilter FileFilter = new SimpleFileFilter(fileTypes, FileTypeName + " (*.
                " + ext + ")");
564         fc.addChoosableFileFilter(FileFilter);
565         fc.setFileFilter(FileFilter);
566         fc.setDialogTitle("Save As");
567         int option = fc.showSaveDialog(this);
568         if (option == JFileChooser.APPROVE_OPTION) {
569             if (fc.getSelectedFile() != null) {
570                 String filename = fc.getSelectedFile().getPath();
571                 filename = filename.trim();
572                 if (!filename.toLowerCase().endsWith("." + ext))
573                     filename = filename + "." + ext;
574 
575                 boolean okToSave = true;
576                 if (FileExists(filename)) {
577                     okToSave = false;
578                     int ans = JOptionPane.showConfirmDialog(this,
579                             "The file " + filename + " already exists. \nDo you wish to
                                overwrite the file?",
580                             "Overwrite File", JOptionPane.YES_NO_OPTION, JOptionPane.
                                QUESTION_MESSAGE);
581                     if (ans == JOptionPane.YES_OPTION)
582                         okToSave = true;
583                 }
584 
585                 if (okToSave) {
586                     saveFile(filename, InfoString);
587                 }
588             }
589         }
590     }
591 
592     public boolean FileExists(String testFilename) {
593         boolean retval = false;
594         try {
595             BufferedReader br = new BufferedReader(new FileReader(testFilename));
596             br.close();
597             retval = true;
598         } catch (Exception e) {
599             retval = false;
600         }
601         return retval;
602     }
603 
604     private void saveFile(String filename, String infoStr) {
605         try {
606             PrintWriter outputfile = new PrintWriter(new FileWriter(filename));
607             outputfile.print(infoStr);
608             outputfile.close();
609             currentFileName = filename;
610         } catch (Exception e) {
611             JOptionPane.showMessageDialog(this, "The file could not be saved.", "IO Error"
                    ,
612                     JOptionPane.WARNING_MESSAGE);
613             File f = new File(filename);
614             try {
615                 f.delete();
616             } catch (Exception delex) {
617             }
618             currentFileName = "";
619         }
620     }
621 
622     public void onShowStatistics() {
```

```
623         JPanel statsPanel = new JPanel();
624         statsPanel.setLayout(new BoxLayout(statsPanel, BoxLayout.Y_AXIS));
625         String inputText = TextEditor.getText();
626 
627         String inputTextForLines = TextEditor.getText();
628         String[] splitInput = inputTextForLines.split("\n");
629         int numlines = splitInput.length;
630 
631         if (numlines == 1)
632             if (splitInput[0].trim().length() == 0)
633                 numlines = 0;
634 
635         int numnonblanklines = 0;
636         if (numlines > 0)
637             for (int i = 0; i < splitInput.length; i++)
638                 if (splitInput[i].trim().length() > 0)
639                     numnonblanklines++;
640 
641         inputText = inputText.replaceAll("\t", "");
642         inputText = inputText.replaceAll("\n", "");
643         int numchars = inputText.length();
644         String inputTextTrim = inputText.replaceAll(" ", "");
645         int numcharsNoSpace = inputTextTrim.length();
646 
647         JPanel numLinesPanel = new JPanel();
648         numLinesPanel.setLayout(new BoxLayout(numLinesPanel, BoxLayout.X_AXIS));
649         numLinesPanel.add(new JLabel("Number of Lines: " + numlines));
650         numLinesPanel.add(Box.createHorizontalGlue());
651 
652         JPanel numNonBlankLinesPanel = new JPanel();
653         numNonBlankLinesPanel.setLayout(new BoxLayout(numNonBlankLinesPanel, BoxLayout.
                X_AXIS));
654         numNonBlankLinesPanel.add(new JLabel("Number of Non-Blank Lines: " +
                numnonblanklines));
655         numNonBlankLinesPanel.add(Box.createHorizontalGlue());
656 
657         JPanel numcharsPanel = new JPanel();
658         numcharsPanel.setLayout(new BoxLayout(numcharsPanel, BoxLayout.X_AXIS));
659         numcharsPanel.add(new JLabel("Number of Characters (Including Spaces): " +
                numchars));
660         numcharsPanel.add(Box.createHorizontalGlue());
661 
662         JPanel numcharsPanelNoSpace = new JPanel();
663         numcharsPanelNoSpace.setLayout(new BoxLayout(numcharsPanelNoSpace, BoxLayout.
                X_AXIS));
664         numcharsPanelNoSpace.add(new JLabel("Number of Characters (Excluding Spaces): " +
                numcharsNoSpace));
665         numcharsPanelNoSpace.add(Box.createHorizontalGlue());
666 
667         String inputTextForWords = TextEditor.getText();
668         inputTextForWords = inputTextForWords.replaceAll("\n", " ");
669         inputTextForWords = inputTextForWords.replaceAll("\t", " ");
670         String[] words = inputTextForWords.split(" ");
671 
672         int numwords = 0;
673         for (int i = 0; i < words.length; i++)
674             if (!words[i].trim().equalsIgnoreCase(""))
675                 numwords++;
676 
677         JPanel numwordsPanel = new JPanel();
678         numwordsPanel.setLayout(new BoxLayout(numwordsPanel, BoxLayout.X_AXIS));
679         numwordsPanel.add(new JLabel("Number of Words: " + numwords));
680         numwordsPanel.add(Box.createHorizontalGlue());
681 
```

```
682             statsPanel.add(numLinesPanel);
683             statsPanel.add(numNonBlankLinesPanel);
684             statsPanel.add(numcharsPanel);
685             statsPanel.add(numcharsPanelNoSpace);
686             statsPanel.add(numwordsPanel);
687
688             JOptionPane.showMessageDialog(this, statsPanel, "Statistics", JOptionPane.
                    PLAIN_MESSAGE);
689         }
690
691         public void onPrintPreview() {
692             if (TextEditor.getText().trim().length() == 0)
693                 return;
694
695             try {
696                 PrinterJob pj = PrinterJob.getPrinterJob();
697                 HashPrintRequestAttributeSet pra = new HashPrintRequestAttributeSet();
698                 Book bk = new Book();
699                 PageFormat thisFormat = pj.defaultPage();
700                 String printtitle = currentFileName;
701                 if (printtitle == "")
702                     printtitle = "Untitled";
703
704                 TextPrinter tp = new TextPrinter(thisFormat, printtitle, "", TextEditor.
                        getText(), 10, TextEditor.getFont(),
705                             TextEditor.getTabSize());
706                 PrintPreview np = new PrintPreview(tp, thisFormat, "Print Preview");
707             } catch (Exception e) {
708                 JOptionPane.showMessageDialog(this, "Cannot print current document.", "Print
                        Error",
709                         JOptionPane.WARNING_MESSAGE);
710             }
711         }
712 }
```

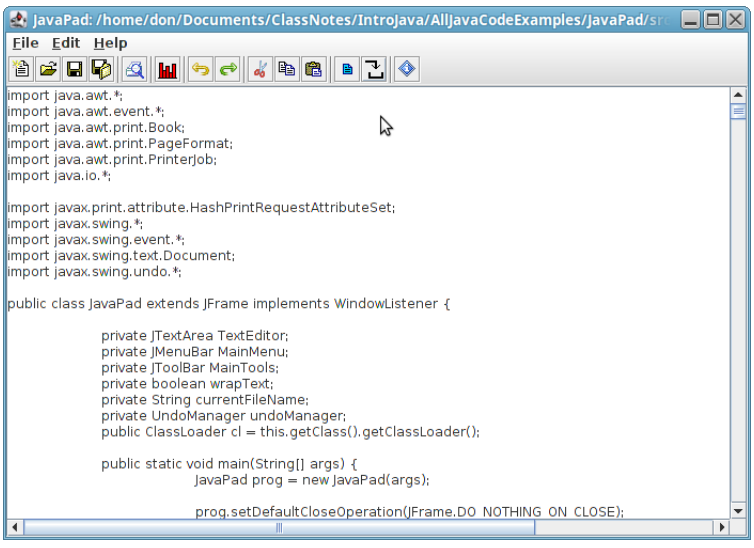


Figure 16.10: JavaPad.java Output: Final

Appendix A

JavaPad: SimpleFileFilter.java

The SimpleFileFilter class extends the FileFilter class to make the addition of filters for file choosers easier to manage. For example,

```
JFileChooser fc = new JFileChooser();
String[] TextFileTypes = {"txt"};
String[] CodeFileTypes = {"cpp", "java"};
SimpleFileFilter TextFileFilter = new SimpleFileFilter(TextFileTypes, "Text Files (*.txt)"
    );
SimpleFileFilter CodeFileFilter = new SimpleFileFilter(CodeFileTypes, "Code Files (*.cpp,
    *.java)");
fc.addChoosableFileFilter(TextFileFilter);
fc.addChoosableFileFilter(CodeFileFilter);
fc.setFileFilter(TextFileFilter);
int option = fc.showOpenDialog(null);
```

Will add in two filters on top of the all files filter, one for text files and the other for C++ and Java files. The one that is selected when the dialog opens is the text files.

```
import javax.swing.filechooser.*;
import java.io.File;

/*-
 * SimpleFileFilter
 * Extends the FileFilter class to make the addition of filters
 * for file choosers easier to manage.
 *
 * Author:  Don Spickler
 */

public class SimpleFileFilter extends FileFilter {

    String[] extensions;
    String description;

    public SimpleFileFilter(String ext) {
        this(new String[] { ext }, null);
    }

    public SimpleFileFilter(String[] exts, String descr) {
        extensions = new String[exts.length];
        for (int i = exts.length - 1; i >= 0; i--) {
            extensions[i] = exts[i].toLowerCase();
```

```
25          }
26          description = (descr == null ? exts[0] + " files" : descr);
27      }
28
29      public boolean accept(File f) {
30          if (f.isDirectory()) {
31              return true;
32          }
33
34          String name = f.getName().toLowerCase();
35          for (int i = extensions.length - 1; i >= 0; i--) {
36              if (name.endsWith(extensions[i])) {
37                  return true;
38              }
39          }
40          return false;
41      }
42
43      public String getDescription() {
44          return description;
45      }
46  }
```

Appendix B

JavaPad: PrintPreview.java

Creates a Print Preview dialog box for the specified print job. The dialog box has features for scaling the view, landscape and portrait selection, page setup and printing. The content pane uses a border layout that utilizes only the north, south and center, so that inherited classes can use the east and west for option panels. The following is an example of the use of the print preview dialog. In the code,

```
1 PrinterJob pj = PrinterJob.getPrinterJob();
2 HashPrintRequestAttributeSet pra = new HashPrintRequestAttributeSet();
3 Book bk = new Book();
4 PageFormat thisFormat = pj.defaultPage();
5 String printtitle = "Document Title";
6 TextPrinter tp = new TextPrinter(thisFormat, printtitle, "", TextEditor.getText(), 10,
      TextEditor.getFont(), TextEditor.getTabSize());
7 PrintPreview np = new PrintPreview(tp, thisFormat, "Print Preview");
```

the TextPrinter class is a Pageable and Printable interface to reformat text to fit on the printable area of the page and to manage page breaks, and the TextEditor is a JTextArea object.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.awt.image.*;
4  import java.awt.print.*;
5
6  import javax.swing.*;
7  import javax.swing.border.*;
8  import javax.swing.event.*;
9
10 /*-
11  * PrintPreview
12  * Creates a Print Preview dialog box for the specified print job.
13  * The dialog box has features for scaling the view, landscape and
14  * portrait selection, page setup and printing.  The content pane
15  * uses a border layout that utilizes only the north, south and center,
16  * so that inherited classes can use the east and west for option panels.
17  *
18  * Author:  Don Spickler
19  */
20
21 public class PrintPreview extends JFrame implements Runnable {
```

```
22      protected JScrollPane displayArea;
23      protected int m_wPage;
24      protected int m_hPage;
25      protected int width;
26      protected int height;
27      protected Printable m_target = null;
28      protected Book m_book = null;
29      protected PreviewContainer m_preview;
30      protected PageFormat pp_pf = null;
31      protected JSlider zoomSlide;
32      protected JLabel pageLabel;
33      protected JToolBar MainTools;
34      protected String PrintJobName = "Java Printing";
35      public ClassLoader cl = this.getClass().getClassLoader();
36      protected ToolbarButton PrintTool;
37
38      // private static final double DOTS_PER_INCH = 72.0;
39
40      public void Warning(String message, String title) {
41          try {
42              JOptionPane.showMessageDialog(this, message, title, JOptionPane.
                    WARNING_MESSAGE);
43          } catch (Exception ex) {
44          }
45      }
46
47      public void Message(String message, String title) {
48          try {
49              JOptionPane.showMessageDialog(this, message, title, JOptionPane.
                    INFORMATION_MESSAGE);
50          } catch (Exception ex) {
51          }
52      }
53
54      protected void getThePreviewPages() {
55          m_wPage = (int) (pp_pf.getWidth());
56          m_hPage = (int) (pp_pf.getHeight());
57          int scale = getDisplayScale();
58          width = (int) Math.ceil(m_wPage * scale / 100);
59          height = (int) Math.ceil(m_hPage * scale / 100);
60
61          int pageIndex = 0;
62          try {
63              while (true) {
64
65                  int mb = 1024 * 1024;
66
67                  boolean show = false;
68                  if (show) {
69                      // Getting the runtime reference from system
70                      Runtime runtime = Runtime.getRuntime();
71
72                      System.out.println("##### Heap utilization statistics [MB] #####");
73
74                      System.out.println("Page:" + pageIndex);
75
76                      // Print used memory
77                      System.out.println("Used Memory:" + (runtime.totalMemory() - runtime.
                            freeMemory()) / mb);
78
79                      // Print free memory
80                      System.out.println("Free Memory:" + runtime.freeMemory() / mb);
81
82                      // Print total available memory
```

```
83                      System.out.println("Total Memory:" + runtime.totalMemory() / mb);
84
85                      // Print Maximum available memory
86                      System.out.println("Max Memory:" + runtime.maxMemory() / mb);
87                      System.out.println();
88                  }
89
90                  BufferedImage img = new BufferedImage(m_wPage, m_hPage, BufferedImage.
                        TYPE_INT_RGB);
91                  Graphics g = img.getGraphics();
92                  g.setColor(Color.white);
93                  g.fillRect(0, 0, m_wPage, m_hPage);
94
95                  if (m_target != null) {
96                      if (m_target.print(g, pp_pf, pageIndex) != Printable.PAGE_EXISTS)
97                          break;
98                  } else if (m_book != null) {
99                      try {
100                         if (m_book.getPrintable(pageIndex).print(g, m_book.getPageFormat(
                                pageIndex),
101                                 pageIndex) != Printable.PAGE_EXISTS)
102                             break;
103                     } catch (Exception e) {
104                         break;
105                     }
106                 }
107
108                 PagePreview pp = new PagePreview(width, height, img);
109                 m_preview.add(pp);
110                 pageIndex++;
111             }
112
113             PrintTool.setEnabled(true);
114             if (pageIndex == 1)
115                 pageLabel.setText(pageIndex + " Page   ");
116             else if (pageIndex > 1)
117                 pageLabel.setText(pageIndex + " Pages   ");
118             else {
119                 pageLabel.setText("");
120                 PrintTool.setEnabled(false);
121             }
122         } catch (Exception e) {
123             pageLabel.setText("");
124             PrintTool.setEnabled(false);
125         }
126     }
127
128     protected void previewThePages() {
129         if (displayArea != null)
130             displayArea.setVisible(false);
131
132         if (m_preview == null)
133             m_preview = new PreviewContainer();
134         else
135             m_preview.removeAll();
136
137         getThePreviewPages();
138
139         displayArea = new JScrollPane(m_preview);
140         JScrollBar verticalScrollBar = displayArea.getVerticalScrollBar();
141         verticalScrollBar.setUnitIncrement(35);
142         getContentPane().add(displayArea, BorderLayout.CENTER);
143         setVisible(true);
144         System.gc();
```

```
145     }
146
147     public int getDisplayScale() {
148         return zoomSlide.getValue();
149     }
150
151     public PrintPreview(Book bk, String title) {
152         super(title);
153
154         m_book = bk;
155         pp_pf = m_book.getPageFormat(0);
156
157         for (int i = 0; i < m_book.getNumberOfPages(); i++) {
158             m_book.getPageFormat(i).setPaper(pp_pf.getPaper());
159             m_book.getPageFormat(i).setOrientation(pp_pf.getOrientation());
160         }
161
162         Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
163         PrintPreviewSetup(title, (int) (0.75 * screen.width), (int) (0.75 * screen.height)
                ;
164     }
165
166     public PrintPreview(Book bk, String title, int wd, int ht) {
167         super(title);
168         m_book = bk;
169         pp_pf = m_book.getPageFormat(0);
170
171         for (int i = 0; i < m_book.getNumberOfPages(); i++) {
172             m_book.getPageFormat(i).setPaper(pp_pf.getPaper());
173             m_book.getPageFormat(i).setOrientation(pp_pf.getOrientation());
174         }
175
176         PrintPreviewSetup(title, wd, ht);
177     }
178
179     public PrintPreview(Printable target, PageFormat pf, String title) {
180         super(title);
181
182         m_target = target;
183         pp_pf = pf;
184         Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
185         PrintPreviewSetup(title, (int) (0.75 * screen.width), (int) (0.75 * screen.height)
                ;
186     }
187
188     public PrintPreview(Printable target, PageFormat pf, String title, int wd, int ht) {
189         super(title);
190         m_target = target;
191         pp_pf = pf;
192         PrintPreviewSetup(title, wd, ht);
193     }
194
195     public void PrintPreviewSetup(String title, int wd, int ht) {
196         PrinterJob prnJob = PrinterJob.getPrinterJob();
197
198         if (pp_pf.getHeight() == 0 || pp_pf.getWidth() == 0) {
199             return;
200         }
201         setSize(600, 400);
202
203         displayArea = null;
204         m_preview = null;
205
206         createToolBar();
```

```
207
208         zoomSlide = new JSlider();
209         zoomSlide.setMinimum(0);
210         zoomSlide.setMaximum(150);
211         zoomSlide.setMajorTickSpacing(5);
212         zoomSlide.setSnapToTicks(true);
213         zoomSlide.setValue(100);
214         zoomSlide.setLabelTable(zoomSlide.createStandardLabels(50, 0));
215         zoomSlide.setPaintLabels(true);
216         zoomSlide.setPaintTicks(true);
217
218         Dimension tempd = zoomSlide.getPreferredSize();
219         int sliderLength = 300;
220         zoomSlide.setPreferredSize(new Dimension(sliderLength, (int) tempd.getHeight() +
                10));
221         zoomSlide.setMaximumSize(new Dimension(sliderLength, (int) tempd.getHeight() + 10)
                );
222         zoomSlide.addChangeListener(new ChangeListener() {
223             public void stateChanged(ChangeEvent e) {
224                 Thread runner = new Thread(PrintPreview.this);
225                 runner.start();
226             }
227         });
228
229         JPanel zoomSlidePanel = new JPanel();
230         zoomSlidePanel.setLayout(new BoxLayout(zoomSlidePanel, BoxLayout.X_AXIS));
231         zoomSlidePanel.add(new JLabel("  Zoom   "));
232         zoomSlidePanel.add(zoomSlide);
233         zoomSlidePanel.add(Box.createHorizontalGlue());
234
235         getContentPane().add(MainTools, BorderLayout.NORTH);
236         getContentPane().add(zoomSlidePanel, BorderLayout.SOUTH);
237         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
238         setSize(wd, ht);
239         setVisible(true);
240         previewThePages();
241     }
242
243     public void run() {
244         int scale = getDisplayScale();
245         width = (int) (m_wPage * scale / 100);
246         height = (int) (m_hPage * scale / 100);
247
248         Component[] comps = m_preview.getComponents();
249         for (int k = 0; k < comps.length; k++) {
250             if (!(comps[k] instanceof PagePreview))
251                 continue;
252             PagePreview pp = (PagePreview) comps[k];
253             pp.setScaledSize(width, height);
254         }
255         m_preview.doLayout();
256         m_preview.getParent().getParent().validate();
257     }
258
259     protected void createToolBar() {
260         MainTools = new JToolBar("Print Preview Tools");
261         MainTools.setFloatable(false);
262         MainTools.setOpaque(false);
263         Dimension tempd;
264
265         PrintTool = new ToolbarButton(new ImageIcon(cl.getResource("PrintIcon.GIF")), "
                Print...", 25, 25);
266         PrintTool.addActionListener(new ActionListener() {
267             public void actionPerformed(ActionEvent evt) {
```

```
268                 try {
269                     PrinterJob prnJob = PrinterJob.getPrinterJob();
270                     if (m_target != null) {
271                         prnJob.setPrintable(m_target, pp_pf);
272                     } else if (m_book != null) {
273                         prnJob.setPageable(m_book);
274                     }
275 
276                     if (prnJob.printDialog()) {
277                         setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
278                         prnJob.setJobName(PrintJobName);
279                         prnJob.print();
280                         setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
281                     }
282 
283                 } catch (Exception ex) {
284                     Warning("There was an error in the printing system.", "Printer Error")
                         ;
285                 }
286             }
287         });
288 
289         ToolbarButton PageTool = new ToolbarButton(new ImageIcon(cl.getResource("
                FileTextImage.GIF")), "Page Setup...",
290                 25, 25);
291         PageTool.addActionListener(new ActionListener() {
292             public void actionPerformed(ActionEvent evt) {
293                 try {
294                     PrinterJob prnJob = PrinterJob.getPrinterJob();
295 
296                     pp_pf = prnJob.pageDialog(pp_pf);
297 
298                     if (m_book != null) {
299                         for (int i = 0; i < m_book.getNumberOfPages(); i++)
300                             m_book.getPageFormat(i).setPaper(pp_pf.getPaper());
301                     }
302 
303                     previewThePages();
304                 } catch (Exception ex) {
305                 }
306             }
307         });
308 
309         ToolbarButton PorLanTool = new ToolbarButton(new ImageIcon(cl.getResource("
                PorLanImage.GIF")),
310                 "Toggle Between Portrait and Landscape Orientation", 25, 25);
311         PorLanTool.addActionListener(new ActionListener() {
312             public void actionPerformed(ActionEvent evt) {
313                 if (pp_pf.getOrientation() == PageFormat.PORTRAIT) {
314                     pp_pf.setOrientation(PageFormat.LANDSCAPE);
315                 } else {
316                     pp_pf.setOrientation(PageFormat.PORTRAIT);
317                 }
318 
319                 if (m_book != null) {
320                     for (int i = 0; i < m_book.getNumberOfPages(); i++)
321                         m_book.getPageFormat(i).setOrientation(pp_pf.getOrientation());
322                 }
323 
324                 previewThePages();
325             }
326         });
327 
328         ToolbarButton CloseTool = new ToolbarButton(new ImageIcon(cl.getResource("
```

```
            RemoveImage.GIF")), "Close Preview",
329             25, 25);
330         CloseTool.addActionListener(new ActionListener() {
331             public void actionPerformed(ActionEvent evt) {
332                 dispose();
333             }
334         });
335 
336         // Add tools to toolbar
337 
338         MainTools.add(PorLanTool);
339         MainTools.add(PageTool);
340         MainTools.add(new JLabel(new ImageIcon(cl.getResource("ToolSeparatorImage.GIF"))))
                ;
341         MainTools.add(PrintTool);
342         MainTools.add(javax.swing.Box.createHorizontalGlue());
343         pageLabel = new JLabel();
344         MainTools.add(pageLabel);
345     }
346 
347     class PreviewContainer extends JPanel {
348         protected int H_GAP = 16;
349         protected int V_GAP = 10;
350 
351         public Dimension getPreferredSize() {
352             int n = getComponentCount();
353             if (n == 0)
354                 return new Dimension(H_GAP, V_GAP);
355             Component comp = getComponent(0);
356             Dimension dc = comp.getPreferredSize();
357             int w = dc.width;
358             int h = dc.height;
359 
360             Dimension dp = getParent().getSize();
361             int nCol = Math.max((dp.width - H_GAP) / (w + H_GAP), 1);
362             int nRow = n / nCol;
363             if (nRow * nCol < n)
364                 nRow++;
365 
366             int ww = nCol * (w + H_GAP) + H_GAP;
367             int hh = nRow * (h + V_GAP) + V_GAP;
368             Insets ins = getInsets();
369             return new Dimension(ww + ins.left + ins.right, hh + ins.top + ins.bottom);
370         }
371 
372         public Dimension getMaximumSize() {
373             return getPreferredSize();
374         }
375 
376         public Dimension getMinimumSize() {
377             return getPreferredSize();
378         }
379 
380         public void doLayout() {
381             Insets ins = getInsets();
382             int x = ins.left + H_GAP;
383             int y = ins.top + V_GAP;
384 
385             int n = getComponentCount();
386             if (n == 0)
387                 return;
388             Component comp = getComponent(0);
389             Dimension dc = comp.getPreferredSize();
390             int w = dc.width;
```

```
391             int h = dc.height;
392
393             Dimension dp = getParent().getSize();
394             int nCol = Math.max((dp.width - H_GAP) / (w + H_GAP), 1);
395             int nRow = n / nCol;
396             if (nRow * nCol < n)
397                 nRow++;
398
399             int index = 0;
400             for (int k = 0; k < nRow; k++) {
401                 for (int m = 0; m < nCol; m++) {
402                     if (index >= n)
403                         return;
404                     comp = getComponent(index++);
405                     comp.setBounds(x, y, w, h);
406                     x += w + H_GAP;
407                 }
408                 y += h + V_GAP;
409                 x = ins.left + H_GAP;
410             }
411         }
412     }
413
414     class PagePreview extends JPanel {
415         protected int m_w;
416         protected int m_h;
417         protected Image m_source;
418         protected Image m_img;
419
420         public PagePreview(int w, int h, Image source) {
421             m_w = w;
422             m_h = h;
423
424             if (m_w <= 0)
425                 m_w = 1;
426             if (m_h <= 0)
427                 m_h = 1;
428
429             m_source = source;
430             m_img = m_source.getScaledInstance(m_w, m_h, Image.SCALE_SMOOTH);
431             m_img.flush();
432             setBackground(Color.white);
433             setBorder(new MatteBorder(1, 1, 2, 2, Color.black));
434         }
435
436         public void setScaledSize(int w, int h) {
437             m_w = w;
438             m_h = h;
439
440             if (m_w <= 0)
441                 m_w = 1;
442             if (m_h <= 0)
443                 m_h = 1;
444
445             m_img = m_source.getScaledInstance(m_w, m_h, Image.SCALE_SMOOTH);
446             repaint();
447         }
448
449         public Dimension getPreferredSize() {
450             Insets ins = getInsets();
451             return new Dimension(m_w + ins.left + ins.right, m_h + ins.top + ins.bottom);
452         }
453
454         public Dimension getMaximumSize() {
```

```
455             return getPreferredSize();
456         }
457
458         public Dimension getMinimumSize() {
459             return getPreferredSize();
460         }
461
462         public void paint(Graphics g) {
463             g.setColor(getBackground());
464             g.fillRect(0, 0, getWidth(), getHeight());
465             g.drawImage(m_img, 0, 0, this);
466             paintBorder(g);
467         }
468     }
469 }
```

Appendix C

JavaPad: TextPrinter.java

The TextPrinter class is a Pageable and Printable interface to reformat text to fit on the printable area of the page and to manage page breaks. It has several constructors for specifying the headers, fonts, and page formats.

```
1  import java.awt.*;
2  import java.awt.print.*;
3  import java.text.*;
4  import java.util.*;
5
6  /*-
7   * TextPrinter
8   * Pageable and Printable interface to reformat text to fit on the
9   * printable area of the page and to manage page breaks.
10  *
11  * Author:  Don Spickler
12  */
13
14 public class TextPrinter implements Pageable, Printable {
15     private PageFormat htmlFormat;
16     private String LeftHeader = "";
17     private String RightHeader = "";
18     private int FontSize = 10;
19     private String NoteText;
20
21     private int PageNumber = 0;
22     private int MaxPageNumber = Integer.MAX_VALUE;
23     private int PagePos = 0;
24     private boolean NewPage = true;
25
26     private Font boldFont;
27     private Font plainFont;
28     private Font italicFont;
29     private Font textFont;
30
31     private int tabSize = 2;
32
33     private int left;
34     private int right;
35     private int top;
36     private int bottom;
37     private int height;
38     private int width;
```

```
39
40      private int headerStart;
41      private int pageStart;
42      private int pageEnd;
43
44      private int horPosAdjust = 1;
45
46      public TextPrinter(PageFormat pf, String leftHead, String rightHead, String message,
            int fs) {
47          NoteText = message;
48          htmlFormat = pf;
49          LeftHeader = leftHead;
50          RightHeader = rightHead;
51          FontSize = fs;
52          textFont = new Font(Font.SERIF, Font.PLAIN, FontSize);
53      }
54
55      public TextPrinter(PageFormat pf, String leftHead, String rightHead, String message,
            int fs, Font TextFont) {
56          NoteText = message;
57          htmlFormat = pf;
58          LeftHeader = leftHead;
59          RightHeader = rightHead;
60          FontSize = fs;
61          textFont = new Font(TextFont.getFontName(), TextFont.getStyle(), FontSize);
62      }
63
64      public TextPrinter(PageFormat pf, String leftHead, String rightHead, String message,
            int fs, Font TextFont,
65              int tabsize) {
66          NoteText = message;
67          htmlFormat = pf;
68          LeftHeader = leftHead;
69          RightHeader = rightHead;
70          FontSize = fs;
71          textFont = new Font(TextFont.getFontName(), TextFont.getStyle(), FontSize);
72          tabSize = tabsize;
73      }
74
75      public PageFormat getPageFormat() {
76          return htmlFormat;
77      }
78
79      public void setPageFormat(PageFormat pf) {
80          htmlFormat = pf;
81      }
82
83      public int getFontSize() {
84          return FontSize;
85      }
86
87      public void setFontSize(int sz) {
88          FontSize = sz;
89      }
90
91      public Font getTextFont() {
92          return textFont;
93      }
94
95      public void setTextFont(Font ft) {
96          textFont = ft;
97      }
98
99      private void drawText(String txt, Graphics2D g2, int pageIndex, int sx, int sy) {
```

```
100         txt += "  ";
101         int cx = sx;
102         int cy = sy;
103         String[] lines = txt.split("\n");
104         g2.setFont(textFont);
105         FontMetrics fm = g2.getFontMetrics();
106 
107         String tabString = "";
108         if (tabSize < 0)
109             tabSize = 0;
110 
111         for (int i = 0; i < tabSize; i++)
112             tabString += " ";
113 
114         for (int line = 0; line < lines.length; line++) {
115             String thisline = lines[line];
116 
117             if ((thisline != null) && (thisline.trim().length() > 0)) {
118                 thisline = thisline.replaceAll("\t", tabString);
119 
120                 if (thisline.length() > 0) {
121                     boolean done = false;
122                     while (!done) {
123                         if (thisline.length() == 0)
124                             done = true;
125                         else {
126                             if (thisline.charAt(thisline.length() - 1) == ' ')
127                                 thisline = thisline.substring(0, thisline.length() - 1);
128                             else
129                                 done = true;
130                         }
131                     }
132                 }
133 
134                 String[] words = thisline.split(" ");
135                 NewPage = false;
136                 for (int word = 0; word < words.length; word++) {
137                     NewPage = false;
138                     if (cx + fm.stringWidth(words[word]) <= right) {
139                         if (PageNumber == pageIndex) {
140                             g2.drawString(words[word], cx, cy - fm.getDescent() -
                                    horPosAdjust);
141                         }
142                         cx = cx + fm.stringWidth(words[word] + " ");
143                     } else {
144                         PagePos += fm.getHeight();
145                         cy = PagePos;
146                         cx = left;
147                         if (PagePos > pageEnd) {
148                             PagePos = pageStart;
149                             cy = PagePos;
150                             cx = left;
151                             NewPage = true;
152                             PageNumber++;
153                         }
154 
155                         if (PageNumber == pageIndex) {
156                             g2.drawString(words[word], cx, cy - fm.getDescent() -
                                    horPosAdjust);
157                         }
158                         cx = cx + fm.stringWidth(words[word] + " ");
159                     }
160                 }
161             }
```

```
162
163             if (line < lines.length - 1) {
164                 PagePos += fm.getHeight();
165                 cy = PagePos;
166                 cx = left;
167                 if (PagePos > pageEnd) {
168                     PagePos = pageStart;
169                     cy = PagePos;
170                     cx = left;
171                     NewPage = true;
172                     PageNumber++;
173                 }
174             }
175         }
176     }
177
178     public int print(Graphics g, PageFormat pageFormat, int pageIndex) throws
            PrinterException {
179         Graphics2D g2 = (Graphics2D) g;
180         g2.setStroke(new BasicStroke(0.5f));
181         NewPage = true;
182
183         if ((pageIndex < 0) || (pageIndex >= getNumberOfPages()))
184             return NO_SUCH_PAGE;
185
186         if (NoteText == null)
187             return NO_SUCH_PAGE;
188
189         if (pageIndex > MaxPageNumber)
190             return NO_SUCH_PAGE;
191
192         PageNumber = 0;
193
194         left = (int) pageFormat.getImageableX();
195         right = (int) (pageFormat.getImageableX() + pageFormat.getImageableWidth());
196         top = (int) pageFormat.getImageableY();
197         bottom = (int) (pageFormat.getImageableY() + pageFormat.getImageableHeight());
198         height = (int) pageFormat.getImageableHeight();
199         width = (int) pageFormat.getImageableWidth();
200
201         boldFont = new Font(Font.SERIF, Font.BOLD, FontSize);
202         plainFont = new Font(Font.SERIF, Font.PLAIN, FontSize);
203         italicFont = new Font(Font.SERIF, Font.ITALIC, FontSize);
204
205         g2.setColor(Color.WHITE);
206         g2.drawRect(0, 0, (int) pageFormat.getWidth(), (int) pageFormat.getHeight());
207         g2.fillRect(0, 0, (int) pageFormat.getWidth(), (int) pageFormat.getHeight());
208         g2.setColor(Color.BLACK);
209
210         g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.
                VALUE_ANTIALIAS_ON);
211         g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, RenderingHints.
                VALUE_TEXT_ANTIALIAS_ON);
212
213         g2.setFont(plainFont);
214         FontMetrics fm = g2.getFontMetrics();
215
216         headerStart = (int) (top + 1.5 * fm.getHeight());
217         pageEnd = (int) (bottom - 1.5 * fm.getHeight());
218
219         // Draw Header
220
221         g2.setFont(boldFont);
222         fm = g2.getFontMetrics();
```

```
223         int strwd = fm.stringWidth(RightHeader);
224         g2.drawString(RightHeader, right - strwd, top + fm.getHeight() - fm.getDescent());
225         g2.drawLine(left, top + fm.getHeight(), right, top + fm.getHeight());
226 
227         g2.setFont(boldFont);
228         fm = g2.getFontMetrics();
229         g2.drawString(LeftHeader, left, top + fm.getHeight() - fm.getDescent());
230 
231         g2.setFont(textFont);
232         fm = g2.getFontMetrics();
233         pageStart = headerStart + fm.getHeight();
234         PagePos = pageStart;
235 
236         // Draw Report
237 
238         if (NoteText.trim().length() != 0)
239             drawText(NoteText, g2, pageIndex, left, PagePos);
240 
241         // Draw Footer
242 
243         g2.setFont(plainFont);
244         fm = g2.getFontMetrics();
245         g2.drawLine(left, bottom - fm.getHeight(), right, bottom - fm.getHeight());
246 
247         String pagenum = "" + (pageIndex + 1);
248         strwd = fm.stringWidth(pagenum);
249         g2.setFont(plainFont);
250         g2.drawString(pagenum, right - strwd, bottom - fm.getDescent());
251 
252         Date now = new Date();
253         SimpleDateFormat df = new SimpleDateFormat("EEEE, MMMM d, yyyy");
254 
255         g2.setFont(italicFont);
256         fm = g2.getFontMetrics();
257         g2.drawString(df.format(now), left, bottom - fm.getDescent());
258 
259         MaxPageNumber = PageNumber;
260 
261         // Place at end to remove ending blank page.
262         if (NewPage)
263             MaxPageNumber--;
264 
265         return PAGE_EXISTS;
266     }
267 
268     public int getNumberOfPages() {
269         return 9999;
270     }
271 
272     public PageFormat getPageFormat(int pageIndex) throws IndexOutOfBoundsException {
273         return htmlFormat;
274     }
275 
276     public Printable getPrintable(int pageIndex) throws IndexOutOfBoundsException {
277         return this;
278     }
279 }
```

Appendix D

JavaPad: ToolbarButton.java

The ToolbarButton class extends the JButton class to make the creation of buttons specifically for a toolbar easier. It has several constructors for specifying the button image, text, tooltip, and size.

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /*-
5   * ToolbarButton
6   * Extends JButton to make the creation of buttons specifically for a toolbar easier.
7   *
8   * Author:  Don Spickler
9   */
10
11 public class ToolbarButton extends JButton {
12     public ToolbarButton(Icon btnImage, String btnText, String ToolTip, int width, int
           height) {
13         setIcon(btnImage);
14         setText(btnText);
15         setVerticalTextPosition(SwingConstants.BOTTOM);
16         setHorizontalTextPosition(SwingConstants.CENTER);
17         setIconTextGap(0);
18         setToolTipText(ToolTip);
19         setPreferredSize(new Dimension(width, height));
20         setMaximumSize(new Dimension(width, height));
21     }
22
23     public ToolbarButton(Icon btnImage, String btnText, String ToolTip) {
24         setIcon(btnImage);
25         setText(btnText);
26         setVerticalTextPosition(SwingConstants.BOTTOM);
27         setHorizontalTextPosition(SwingConstants.CENTER);
28         setIconTextGap(0);
29         setToolTipText(ToolTip);
30     }
31
32     public ToolbarButton(Icon btnImage, String ToolTip) {
33         setIcon(btnImage);
34         setToolTipText(ToolTip);
35     }
36
37     public ToolbarButton(Icon btnImage, String ToolTip, int width, int height) {
```

```
38          setIcon(btnImage);
39          setToolTipText(ToolTip);
40          setPreferredSize(new Dimension(width, height));
41          setMaximumSize(new Dimension(width, height));
42      }
43
44      public ToolbarButton(Icon btnImage) {
45          setIcon(btnImage);
46      }
47
48      public ToolbarButton(String btnText) {
49          setText(btnText);
50      }
51  }
```

Appendix E

Portion of the ASCII Table

#	Character	#	Character	#	Character
33	!	65	A	97	a
34	“	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	’	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z

#	Character	#	Character	#	Character
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		
64	@	96	`		