## 1 Before Getting Started on the Exercises

Review the class examples we discussed this past week and make sure you understand what each command and operation does. Specifically, make sure you understand array creation and manipulation.

For each exercise submit

1. The Java code file containing the program.
2. A Word, LibreOffice, or text file containing at least 3 runs of the program.

## 2 Exercises

1. Create a program that does the following.
   (a) In the main, the program will ask the user for the size of the array. The main will create an array of integers of that size.
   (b) The main will call the method

   ```
   PopulateArray(int[] A)
   ```

   which will ask the user for each entry of the array one by one and insert the input values into the array.
   (c) The main will then call a method to print the array out to the screen with some spaces between the entries. We will update the method we developed in class. This method will take the array as a parameter but it will also take a width for the amount of space to use for each entry. The method header will look like the following,

   ```
   public static void PrintArray(int[] A, int width)
   ```

   For this method you will be using a printf instead of a print. Recall that with an integer you use a d in the *tag* to format the integer. So the following line

   ```
   System.out.printf("%5d", A[i]);
   ```

   will print the $i^{th}$ entry of $A$ with 5 spaces. You will need to update this so that the input width is used in place of the 5.
   (d) The main should then call a method

   ```
   ManipulateArray(int[] A)
   ```

   which does the following to each entry in the array. If the entry is even then the entry is replaced by half its value, if the entry is evenly divisible by 3 then the entry is replaced with one third its value, if the entry is neither divisible by 2 or 3 then 5 is added to the entry.
   (e) The main should then print out the contents of the array.
   (f) The main should then call the manipulate and the print methods two more times.

   A run of the program is below.

```
Input Array Size: 5
Input entry 1: 1
Input entry 2: 2
Input entry 3: 3
Input entry 4: 4
Input entry 5: 5
    1    2    3    4    5
    6    1    1    2   10
    3    6    6    1    5
    1    3    3    6   10
```

2. Recall the Nifty sequence that takes a number and if it is even it will divide it by 2 and if it is odd it will multiply by 3 and add 1. As we discussed before the sequence always seems to hit one at some point and then we stop the sequence. This exercise will take an input number from the user and instead of just printing the sequence out it will store the sequence in an array. Since the length of the sequence differs depending on the first number we will need to calculate the sequence first to determine its length, then create an array of just the right size, and then put the sequence into the array.

   Create a program that does the following.

   (a) In the main, the program will ask the user for the first number.

   (b) The main then will call the method

   ```
   public static int NiftySequenceLength(int n)
   ```

   that will return the length of the sequence what begins with the number $n$.

   (c) The main then creates the array of the correct size, loads $n$ into the first position and then calls,

   ```
   public static void PopulateArray(int[] A)
   ```

   which loads the sequence into the array.

   (d) Have the main call a print array method to print out the array. You can, of course, use the array printer that you wrote in the previous exercise.

   (e) Have the program call two more methods that you will write. One of these is to count the number of even numbers in the list and the other is to count the odd numbers in the list. The headers for these will look like,

   ```
   public static int CountEvens(int[] A)
   public static int CountOdds(int[] A)
   ```

   Have the main print these values out.

   A run of the program is below.

   **Program Run:**

   ```
   Input n: 12
     12    6    3  10    5  16    8    4    2    1
   Number of even numbers in the list: 7
   Number of odd numbers in the list: 3
   ```

3. Below is the start of a program that creates a random one-dimensional array of a size that is input by the user and a maximum entry size also specified by the user. The program also takes in an input of an integer `countdiv` which is a number used in the two counting functions described below. After the code segment and run there are instructions on what functions you should create. The main program and functions that are already written are not to be altered, you will simply be adding functions to the program.

   ```
   import java.util.Scanner;

   public class Lab08_03 {

       <<< INSERT NEW FUNCTIONS HERE >>>

       public static void main(String[] args) {
           Scanner keyboard = new Scanner(System.in);
           System.out.print("Input the array size: ");
           int arraySize = keyboard.nextInt();
           System.out.print("Input max entry size: ");
           int entrysize = keyboard.nextInt();
           System.out.print("Input the count division: ");
           int countdiv = keyboard.nextInt();

           int intArray[] = new int[arraySize];
           PopulateArray(intArray, entrysize);
   ```

```
        PrintArray(intArray, 4);

        System.out.println("The sum of the array is = " + SumArray(intArray));
        System.out.println("The average of the array is = " + AvgArray(intArray));
        System.out.println("The maximum of the array is = " + MaxArray(intArray));
        System.out.println("The minimum of the array is = " + MinArray(intArray));
        System.out.println("The number less than " + countdiv + " in the array is = "
            + CountLessArray(intArray, countdiv));
        System.out.println("The number greater than " + countdiv + " in the array is = "
            + CountGreaterArray(intArray, countdiv));
        System.out.println("The variance of the array is = " + VarianceArray(intArray));
        System.out.println("The standard deviation of the array is = " + StandardDeviationArray(intArray)
             );

        System.out.println();
        PrintArrayBarChart(intArray);
        System.out.println();

        int[] B = ReverseArray(intArray);
        PrintArray(intArray, 4);
        PrintArray(B, 4);
    }
}
```

**Program Run:**

```
Input the array size: 20
Input max entry size: 50
Input the count division: 15
  44  39   9  50  32  33  22  10  37  18  33  10  47  37  48  31  20   2  21  18
The sum of the array is = 561
The average of the array is = 28.05
The maximum of the array is = 50
The minimum of the array is = 2
The number less than 15 in the array is = 4
The number greater than 15 in the array is = 16
The variance of the array is = 204.89210526315787
The standard deviation of the array is = 14.314052719728187

********************************************
***************************************
*********
**************************************************
********************************
*********************************
**********************
**********
*************************************
******************
*********************************
**********
***********************************************
*************************************
************************************************
*******************************
********************
**
*********************
******************

  44  39   9  50  32  33  22  10  37  18  33  10  47  37  48  31  20   2  21  18
  18  21   2  20  31  48  37  47  10  33  18  37  10  22  33  32  50   9  39  44
```

The functions you are to create and what they do are described below.

(a) **public static void** `PopulateArray(`**int**`[] A,` **int** `n)`

This function will take in an array and integer $n$ and populate the array with random numbers between 1 and $n$.

(b) `public static void PrintArray(int[] Arr, int width)`

This method will take the array as a parameter but it will also take a width for the amount of space to use for each entry. You will simply need to use the one you created earlier.

(c) `public static void PrintArrayBarChart(int[] Arr)`

This function prints a bar chart of the array entries. The bars are horizontal and the number of * matches the array entry.

(d) `public static int SumArray(int[] Arr)`

This function calculates and returns the sum of the array entries.

(e) `public static double AvgArray(int[] Arr)`

This function calculates and returns the average of the array entries. If the size of the array is less than one return 0.

(f) `public static double VarianceArray(int[] Arr)`

This function calculates and returns the variance of the array entries. The variance of a list of numbers is defined to be

$$\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}$$

where $x_i$ represents the $i^{th}$ entry in the array and $\mu$ is the average (or mean) of the array entries. The sum is taken over all array entries and $n$ is the size of the array. If the size of the array is less than two the function should return 0.

(g) `public static double StandardDeviationArray(int[] Arr)`

This function calculates and returns the standard deviation of the array entries. The standard deviation of a list of numbers is defined to be the square root of the variance. As with the variance, if the size of the array is less than two return 0.

(h) `public static int MaxArray(int[] Arr)`

This function finds and returns the maximum value of the array.

(i) `public static int MinArray(int[] Arr)`

This function finds and returns the minimum value of the array.

(j) `public static int CountLessArray(int[] Arr, int n)`

This function finds and returns the number of array entries that are strictly less than $n$.

(k) `public static int CountGreaterArray(int[] Arr, int n)`

This function finds and returns the number of array entries that are strictly greater than $n$.

(l) `public static int[] ReverseArray(int[] Arr)`

This function returns an array of the same size as the one input with its entries in reverse order. Note that the return type is an array of integers `int[]`. To make a method return an array you need to create a new array inside the method and then return it at the end. Since this array has the same size as the one coming in as a parameter we need to use `Arr.length` as the size of the new array. The shell to this method is as follows.

```
public static int[] ReverseArray(int[] Arr) {
    int B[] = new int[Arr.length];

< Insert code that will store the reverse of the array Arr in B. >

    return B;
}
```

Be careful not to alter the contents of `Arr` since this will alter the contents of the array created in the main and we do not want to do that.

## 3 Challenge Exercise: The Vigenère Cipher

Challenge Exercises are optional, they will be graded as extra credit.

The Vigenère cipher was developed in the mid sixteenth century by Blaise de Vigenère. When Vigenère was twenty six years old he went on a two-year diplomatic mission to Rome. There is where he was first exposed to the world of cryptography. Building on the work of Leon Battista Alberti(1404–1472) (often called the Father of Western Cryptography), Johannes Trithemius, and Giovanni Porta, Vigenère developed several cryptographic methods and steganographic techniques. In 1585 he published *Traictè des Chiffres*, which contained his contributions to the field. Vigenère developed much more sophisticated ciphers then the standard repeated keyword cipher we will discuss here. We will briefly discuss some of the autokey ciphers that Vigenère published in the Traictè des Chiffres, which can be more difficult to break than the classical repeated keyword.

Alberti was one the leading figures of the Renaissance; a painter, composer, poet and philosopher. He was also the author of the first scientific analysis of perspective, a treatise on the housefly, and a funeral oration for his dog. He is probably best known as an architect, having designed Rome's first Trevi Fountain and having written "De Re Aedificatoria", the first printed book on architecture, which acted as a catalyst for the transition from Gothic to Renaissance design.[1]

The Vigenère cipher is a polyalphabetic cipher which is a variation of the Caesar shift cipher. As the name implies it uses several different alphabetic substitutions in place of the one, as in the monoalphabetic ciphers.

The Vigenère cipher was one of those "quantum leaps" in cryptography, of which there are several in history, where it seems that a completely secure method of encryption had been found. The Vigenère cipher was considered to be unbreakable, and it achieved the title of the *chiffre indéchiffrable.* It was not until 1863, nearly 300 years after its creation, when F. W. Kasiski found a method for cryptanalyzing the cipher and later the method of coincidence analysis made the determination of the keyword length even easier.

The Vigenère cipher encryption is implemented as follows,

1. A keyword is chosen for the cipher, this is the key to both encryption and decryption.
2. Each letter is converted to a shift value using the standard A–Z as 0–25, as in the chart below. The keyword then becomes a shift vector.

   For example, if the keyword is VIGENERE then the shift vector is $(21, 8, 6, 4, 13, 4, 17, 4)$.
3. The plaintext message is written out character for character and the keyword (or shift vector) is written below it, character for character and the keyword is repeated as many times as is needed to cover the message.
4. Each character in the plaintext message is shifted by the amount as its corresponding keyword letter to create the ciphertext.

[1]The Black Chamber by Simon Singh (2015) `http://www.simonsingh.net/The_Black_Chamber/`

Table 1: Vigenère Cipher Shift Coding

| **Letter** | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Shift** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

| **Letter** | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Shift** | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

The decryption process is the same except that the shift done in the last step is reversed to take the cuphertext back to the plaintext.

**Example:** Lets say that we want to send the message,

Meet me at Cool Beans tomorrow afternoon.

First we convert the message as follows, we convert it to all uppercase,

MEET ME AT COOL BEANS TOMORROW AFTERNOON.

then we remove anything that is not an alphabetic character,

MEETMEATCOOLBEANSTOMORROWAFTERNOON

Lets say that we use the key

coffee

We do the same to the key as we did with the message so it gets converted to

COFFEE

We then convert the characters to numbers, A to 0, B to 1, C to 2, . . . , Z to 25. So our message is coded as

12 4 4 19 12 4 0 19 2 14 14 11 1 4 0 13 18 19 14 12 14 17 17 14 22 0 5 19 4 17 13 14 14 13

We do the same with the key.

2 14 5 5 4 4

Now we repeat the key so that it has the same number of numbers as the message, and we place it next to the message. We then add entry by entry and take the result mod 26. The table below shows the process.

| **Message** | **Key** | **Sum** | **Mod 26** |
|---|---|---|---|
| 12 | 2 | 14 | 14 |
| 4 | 14 | 18 | 18 |
| 4 | 5 | 9 | 9 |
| 19 | 5 | 24 | 24 |
| 12 | 4 | 16 | 16 |
| 4 | 4 | 8 | 8 |
| 0 | 2 | 2 | 2 |
| 19 | 14 | 33 | 7 |
| 2 | 5 | 7 | 7 |
| 14 | 5 | 19 | 19 |
| 14 | 4 | 18 | 18 |
| 11 | 4 | 15 | 15 |
| 1 | 2 | 3 | 3 |
| 4 | 14 | 18 | 18 |
| 0 | 5 | 5 | 5 |

| Message | Key | Sum | Mod 26 |
|---|---|---|---|
| 13 | 5 | 18 | 18 |
| 18 | 4 | 22 | 22 |
| 19 | 4 | 23 | 23 |
| 14 | 2 | 16 | 16 |
| 12 | 14 | 26 | 0 |
| 14 | 5 | 19 | 19 |
| 17 | 5 | 22 | 22 |
| 17 | 4 | 21 | 21 |
| 14 | 4 | 18 | 18 |
| 22 | 2 | 24 | 24 |
| 0 | 14 | 14 | 14 |
| 5 | 5 | 10 | 10 |
| 19 | 5 | 24 | 24 |
| 4 | 4 | 8 | 8 |
| 17 | 4 | 21 | 21 |
| 13 | 2 | 15 | 15 |
| 14 | 14 | 28 | 2 |
| 14 | 5 | 19 | 19 |
| 13 | 5 | 18 | 18 |

We then take the last column of numbers and convert these to letters again using the 0 to A, 1 to B, 2 to C, . . . , 25 to Z. So our ciphertext is

OSJYQICHHTSPDSFSWXQATWVSYOKYIVPCTS

If we wanted to decrypt this ciphertext we would simply reverse the process and subtract the key from the ciphertext letters.

For this exercise you will construct a program that will encrypt and decrypt messages using the Vigenère Cipher. First construct the following methods,

1. `public static char[] ProcessMessage(String message)`

   This method will take in a string and return an array of characters that are all uppercase and any characters removed that are not alphabetic. So the input of

   Meet me at Cool Beans tomorrow afternoon.

   will get converted to an array of the characters

   MEETMEATCOOLBEANSTOMORROWAFTERNOON

   One command that will come in handy here is in the Character class. If ch is a character then the command Character.isAlphabetic(ch) will return true if ch is an alphabetic character and false otherwise.

2. `public static int[] ConvertCharArrayToIntegerArray(char[] A)`

   This will convert an array of characters to an array of integers using A to 0, B to 1, C to 2, . . . , Z to 25.

3. `public static char[] ConvertIntegerArrayToCharArray(int[] A)`

   This will convert an array of integers to an array of characters using 0 to A, 1 to B, 2 to C, . . . , 25 to Z.

4. `public static String ConvertCharArrayToString(char[] A)`

   This will convert an array of characters to a string.

5. `public static int[] RepeatArray(int[] A, int newLength)`

   This will take an array of integers and make a new array of the length newLength and will put the array entries of A into the new array repeating A as many times as needed. For example, if we start with the array

   | | | | | |
   |---|---|---|---|---|
   | 3 | 7 | 13 | 4 | 2 |

   and give a new size of 18 the result will be the array,

   | | | | | | | | | | | | | | | | | | |
   |---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
   | 3 | 7 | 13 | 4 | 2 | 3 | 7 | 13 | 4 | 2 | 3 | 7 | 13 | 4 | 2 | 3 | 7 | 13 |

6. `public static int[] VigenereEncrypt(int[] Message, int[] Key)`

   This will take the two integer arrays (assumed to be the same length), add them together entry by entry, modulo 26, and return the new array.

7. `public static int[] VigenereDecrypt(int[] Cipher, int[] Key)`

   This will take the two integer arrays (assumed to be the same length), subtract the Key from the Cipher, modulo 26, and return the new array. Here you need to be careful because `%` does not work the way you want it to with negative numbers.

Now write a main program that uses these to encrypt and decrypt a message using the Vigenère Cipher. The program should first ask the user if they want to encrypt or decrypt. Then it will ask for the message or ciphertext and key. It will then convert the message or ciphertext to uppercase with only characters being used. Same for the key. It will then convert the characters to arrays of integers, do the encryption or decryption, and then convert the result to a string and print it to the screen. Some examples are below,

```
Encode or Decode (E/D): e
Input the message: Meet me at Cool Beans tomorrow afternoon.
Input the keyword: coffee
Ciphertext: OSJYQICHHTSPDSFSWXQATWVSYOKYIVPCTS

Encode or Decode (E/D): d
Input the ciphertext: OSJYQICHHTSPDSFSWXQATWVSYOKYIVPCTS
Input the keyword: coffee
Plaintext: MEETMEATCOOLBEANSTOMORROWAFTERNOON

Encode or Decode (E/D): e
Input the message: Not much of a challenge.
Input the keyword: Vigenere
Ciphertext: IWZQHGYSAIILNPCIIOK
```