1 Objectives

1. Practice using Arrays and Strings in Java.
2. Practice with user input and output.
3. Practice with data types, and ASCII representations.

2 Pre-Reading

In this lab, you will let a user transform some words into “leet speak” and 2 different forms of encrypted strings. You will also get experience seeing and using the ASCII representations of characters within strings.

Before you begin, consider that in Java, the type called String is really just an array of char types. However, you can’t generally access a String element-by-element like you can with an array, e.g. myString[4];. But, we can convert a String to a char[] and vice versa!

For example, it is valid to use

```java
String myString = "hello";
char[] myStringArr = myString.toCharArray();
System.out.println("The character at index 2 is: " + myStringArr[2]);
// Output: The character at index 2 is: l
```

Similarly, we can create a string from a character array as follows:

```java
char[] myCharArr = { 'h', 'e', 'l', 'l', 'o', ' ', 'o', ' ', ':', ')' };
String myString = new String(myCharArr);
System.out.println(myString);
// Output: hello :) 
```

In the above example, notice a separate element for each character, and the space character, ' ', is required to be one of them if we want to see some spacing. Also, we need the new keyword to convert the character array to a String. You will use this way of accessing string characters to substitute their values!

For the second part of the lab, we will be taking into account that all characters are represented by ASCII codes. That is, when we see the character ‘a’ on screen, it is really an integer that gets translated from the numeric value 97. You can see the full ASCII table in Figure 1.

For the second part of the lab, we will be taking into account that all characters are represented by ASCII codes. That is, when we see the character ‘a’ on screen, it is really an integer that gets translated from the numeric value 97. You can see the full ASCII table in Figure 1.

Now, when we recall that the char type in Java takes exactly one byte, we can see why! Since a single byte is 8 bits, when we write it in base-2, the largest possible value is 255. So it only takes the lowest 7 bits of a char to represent every possible ASCII value. But this is where more “advanced” character sets like extended ASCII or Unicode come into play so that we can use special characters from other languages, emojis, and other things as “text” elements. We will discuss base-2, bytes, bits, in class, but you should keep in mind the different ways that you can represent one single number!
3 Tasks

1. Maintain two arrays for use in “translating” regular text to “leet speak”:
   (a) One array will contain the standard characters:
       ```java
       char[] letters = {'a', 'e', 'l', 'o', 's', 't'};
       ```
   (b) The other will contain the corresponding symbols:
       ```java
       char[] symbols = {'@', '3', '1', '0', '$', '7'};
       ```

2. Prompt the user to enter a full sentence with all lower-case characters, then store it in a variable to be used later.
   (a) You can (optionally) manually convert the string to all lower-case with the `String` method called `toLowerCase()`, for example:
       ```java
       String one = "Hello, World!";
       String two = one.toLowerCase();
       System.out.println(two); // Output: hello, world!
       ```

3. Convert the input string to an array of characters.

4. Then, use a loop to iterate the array of characters and, for each one, check to see if it is in the `letters` array: if it is, replace with the corresponding element (i.e. same index) in the `symbols` array.
   (a) For example, if the user inputs `leet speak` the output should be `1337$p3@k`. 

---

**Figure 1:** The ASCII table
5. Next, “reset” the character array to match the original input string – you can just use the `.toCharArray()` command again.

6. Now, prompt the user for a positive integer, store it in a variable. This will be a shift for a basic “Caesar Cipher”.

7. Finally, loop the character array and, if the character is not a space, display:

   (a) Show the character itself
   (b) Show the corresponding ASCII code by converting it to an `int`. See the following example code of type casting between `int` and `char`:
   ```java
   char myChar = 'd';
   int myInt = 5;
   int combined = myChar + myInt;
   System.out.println(myChar); // d
   System.out.println((int) myChar); // 100
   // NB: when doing addition, they become an int if you don't specify
   System.out.println(myChar + myInt) // 105
   System.out.println((char) combined); // i
   ```
   (c) Show the result of adding the user’s integer to the ASCII code and converting back to a char.
      i. When you add, you want to “wrap around” from z back to a. So to accommodate this, you can use the modulus operator, `%` along with some clever arithmetic.
      ii. For example, if the user enters 5 for the shift value, and you have the character x (ASCII value 120), the result should be c (ASCII value 99); likewise, if the character is b (ASCII value 97) the result would be g (ASCII value 103).

8. Either with a new for-loop, or in the one above, change the elements of the character array by adding the integer to each one. Then display the “shifted” sentence to the user. This is known as the “Caesar Cipher”, and was an early form of sending encrypted messages; you just need to know what the shift value was in the beginning, then subtract it out from the message you receive!

4 Submission

Upload your source code, properly documented, to the course MyClasses page.