Advanced Programming

# Lab: String Theory

As in all lab assignments, you are expected to fully comment your code according to the published style guide. Comments are to be written before coding begins.

## *Background*

- Strings can be used to represent words, which in turn can make a program more user-friendly.

- Strings are objects that represent character sequences.

Defining a `String` variable creates a pointer that can point to a `String` object. The following code defines the `String` variable `myString.`

```
String myString;
```

This action is similar to typing `“int x;”`.

Since `myString` does not point to a `String` object, it is null. A null string is not the same as a string with no characters! A null string is ‘no string’.

Just like any other object, a `String` object must be created before it can be used. We can do this using the constructor. The following code constructs a `String` object and sets its value to “String Theory”.

```
String myString = new String(“String Theory”);
```

- Enclosing text in quotes creates new instances.

Enclosing text in quotes creates a string literal. Java automatically constructs a `String` object when it sees text in quotes. We usually use string literals to assign string pointers to strings rather than calling the `String` constructor. Creating the `String` “String Theory” using string literals looks like this:

```
String myString = “String Theory”;
```

- Use escape sequences to create strings containing ", \, or newline characters.

Escape sequences are strings containing the ‘\’ character and are used to create strings that contain non-printing characters such as the newline character or the tab character. For example, to specify a string that has characters on two lines, such as

String
Theory

we must use the newline character. This string would be defined as:

```
String myString = “String \n Theory”;
```

Other escape sequences are the tab (\t), the quote (\”) and, of course, the backslash character itself (\\).

- Different instances may contain the same characters, so don't compare with == as you would with primitive types like `int` or `double`.

The `==` operator is used to compare primitive types and object references. It is never used to compare objects, only their references. If you create the following two string objects:

```
String string1 = “String Theory”;
String string2 = new String (“String Theory”);
```

then the statement `string1==string2` will return `false`. (Because Java creates the string object when a literal is used, if we had written `String string2 = “String Theory”`, java would have assigned the pointers to the same object and the == comparison would have returned `true.`)

If you want to compare the actual strings (not the references) you must use the `String` class method `equals`. For the above example,

```
string1.equals(string2)
```

would return `true.` Remember that this is the Karel the Robot equivalent to telling a `UrRobot` named `karel` to turn left using the message `karel.turnLeft()`.

- Strings are immutable—no methods will change the sequence of characters.

The `String` class contains no mutator (setter) methods. Once a string is constructed, it cannot be changed. When you manipulate a string, you end up creating a new `String` object. This has implications when passing a `String` object to a method – the method cannot change the string it was passed.

- "ab" + "cd" will create the new string "abcd".

The + operator applied to strings concatenates the two strings.

- Remember that the characters in a string are numbered starting from 0!

Here are some messages that the `String` class supports. While the `String` class supports many other messages, we will **only** use the following:

- `boolean` **`equals`**`(Object other)`
- `int` **`compareTo`**`(Object other)` `// return value < 0 if this is less than other`
  `// return value = 0 if this is equal to other`
  `// return value > 0 if this is greater than other`
- `int` **`length`**`()`
- `String` **`substring`**`(int from, int pastEnd)` `// returns the substring beginning at from and ending at pastEnd-1`
- `String` **`substring`**`(int from)` `// returns substring(from, length())`
- `int` **`indexOf`**`(String s)` `// returns the index of the first occurrence of s; // returns -1 if not found`
- `String toLowerCase()` `// returns the lower case representation`

## Background: WordGameDisplay

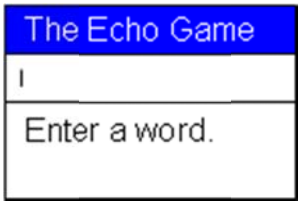
For this lab, you will need to download WordGameDisplay.java, Main.java, and testDoc.txt to a new directory. The `WordGameDisplay` class will be our user interface for simple word games. You should *not* need to modify it. Here are the messages it supports.

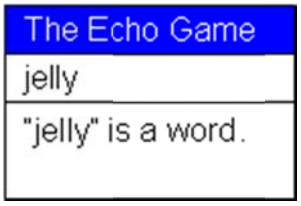
```
void setTitle(String title)

String getGuess()

void setText(String text)

String loadString(String fileName)
```

## Exercise 1: The Echo Game ... Echo Game ... Echo Game ...

Let's try using a `WordGameDisplay`. Create a class called `WordGame`. In its constructor, create a new `WordGameDisplay` and store it in an instance variable.

Now write a method called `echo`. Calling this method should set the title of the display window to "The Echo Game" and should prompt the user to enter a word. When the user has typed a word and hit enter, the window should "echo" back what the user typed, as shown below. (Yes, the word must appear in quotes.)



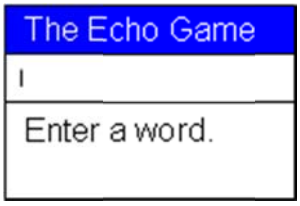



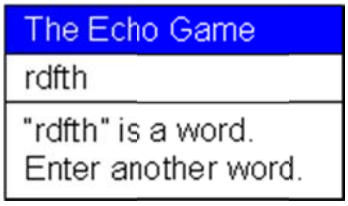

*Before entering a word* *After entering a word*

Make sure this fun game works correctly before going on. Then go ahead and make your game even *more* fun (is that even possible?) by making the game repeat *forever*, prompting the user for a new word each time. You can use the following construct to program this infinite loop.

```
while (true)
{
... your lousy code ...
}
```

Your game should now appear as follows. (Yes, the prompt for another word must appear on a new line.)




*Before entering the first word* *After entering each word*

## Exercise 2: Only letters!

Suppose we are only interested in the letters contained in a string. We want to throw away all spaces, numbers and punctuation and we want to change all capital letters to lower case. In this exercise, we will develop a method that takes in a `String` and produces an output `String` consisting of only the letters in the original string, all lower case.

It turns out the `String` class `compareTo` method can help us determine if a given letter is in the set of lower case letters. When `compareTo` is called on a `String` object that contains only one letter and its parameter is also only one letter, the return value will be the lexographical distance between the letters. Suppose the `String aLetter` contains only the letter "c". Then the following code:

```
int diff = aLetter.compareTo("a");
```

will return the number 2. That is, the letter "c" is two letters beyond the letter "a". What set of numbers will be produced when `aLetter` is selected from the set ["a", "b", … "z"]?

Our strategy is:

1. Create an empty `String` object that will hold the output string.
2. Convert the input string to lower case.
3. Traverse the input string and compare each character to see if it is in the set ["a", "b", … "z"]. If it is, add it to the output string, otherwise discard it. We can traverse the input string using a loop like the following:

```
for(int i = 0; i < inputString.length(); i++)
{
     ... your most excellent code ...
}
```

4. Return the resulting `String` object.

Add a **private** method to your `WordGame` class called `onlyLetters` which takes in a `String` object and returns a String containing only the letters present in the input, converted to lower case.

Add a method called `lowerCaseLetters` to your `WordGame` class. Calling this method should set the title of the display to "Only Lower Case", and prompt the user for a string. It should then convert the string using the `onlyLetters` method and display the result.

## Exercise 3: Palindromes

A *palindrome* is a string that reads the same both forward and backwards. Some examples of palindromes are:

"Mom"
"racecar"
"A man a plan, a canal, Panama!"

And my all-time favorite:

"Go hang a salami, I'm a lasagna hog!"

Let's see if we can design and implement a program that tests if an input string is a palindrome.

The first method we will implement is a private helper method that will examine an input string consisting of **only lower case letters** and return true if the input string is a palindrome. Call this method `isPalindrome`, and add it to your `WordGame` class.

First, we need to make some definitions. A string with no characters (an empty string) is a palindrome. Also, a string with only one character is a palindrome.

Check it out! A palindrome is a string whose first and last letters are the same, and sandwiched in the middle is a palindrome! If you think recursively, this will lead you to an efficient recursive solution to detecting a palindrome.

Another algorithm is to reverse the string and compare it to itself. If the string is equal to the reverse of itself, then it is a palindrome.

Or, you can start at the beginning and compare the first and last characters, then the second and next to last, etc. until you get to the middle. If the characters are equal, then the string is a palindrome.

Write the private helper method `isPalindrome in the WordGame` class.

Now, write the method `palindrome` in the WordGame class. Calling this method should set the title of the display area to "Palindrome" and prompt the user for a string. It should then test the string to see if it is a palindrome, and then it should print out the original input string and "is a palindrome" if the input string is a palindrome or it should print out the original input string and "is not a palindrome' if the original input string is not a palindrome.

Now, make your game even more fun by making your game repeat *forever*, each time asking the user for a new string and adding the result to the display. Make sure the display shows a complete history of the game.

Finally, modify your game so that it will stop when the user inputs an empty string.

## Exercise 4: Counting Occurrences

Certain words (and phrases) occur more often than others in English language prose. We could use information about the probability of occurrence for performing encryption or compression. Compression is a particularly important application since it reduces the bandwidth required to send a message. Counting the occurrences of words is the first step in creating a Huffman code for a message.

In this exercise, you will write a series of methods that will read an input text file into a `String` and then determine the number of times an inputted word or phrase occurs in the file.

The first method you will write will open a text file that you specify and read the contents into one gigantic string. It will return this string. The second method will process a `String` passed as a parameter, and determine the number of occurrences of a second `String` also passed as a parameter. It will return the count. The third, and final, method will prompt the user for a file, and then repeatedly ask for a word or phrase and output the number of occurrences. The third method will stop when an empty string is input.

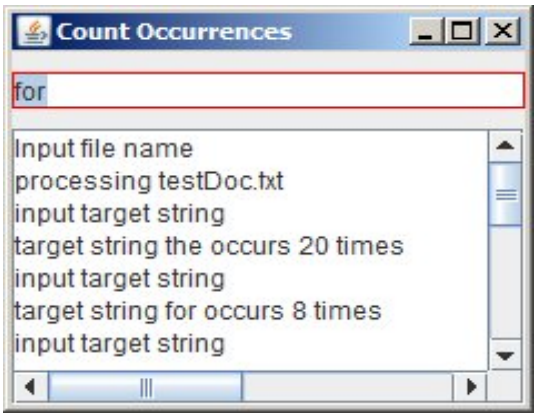
Begin by adding the private method `readInput` to your `WordGame` class. The method `readInput` will take in a `String` that contains the input file name. It will return a `String` consisting of all of the text data in the file. The method will throw an `IOException` if anything goes wrong, and it will terminate the program. An outline of the `readInput` method is given below. Fill in the missing line(s) with your code!

```
/**
* readInput takes in a string containing a file name and
* then wraps a BufferedReader around a FileReader for that
* file.  If the file is not found, or any other IO error
* occurs, an exception is thrown.
* @param filename is the input file name
* @return a String containing the input data
*/
private String readInput(String fileName)
{
     try
     {
          BufferedReader in = new BufferedReader(new
                    FileReader(fileName));
          // read a line of text into the variable line
          // end of line characters are discarded
          // you may want to put them back!
          String line = in.readLine();
          String lines = "";
          while (line != null)
          {
               // add code to append line to lines

               line = in.readLine();
          }
          in.close();

          return lines;
     }
     // in case of error ...
     catch(IOException e)
     {
          RuntimeException up = new RuntimeException(e);
          throw up;
     }
}
```

After you have tested `readInput` using the provided file testDoc.txt, add a private method called `countOccurrences` to `WordGames.` The method `countOccurrences` should take in a `String` object that represents the string to be processed, and a second `String` object that represents the target word or phrase to be counted. It will then scan the entire input string looking for and counting occurrences of the target string. When the method completes, it should return the count as an `int`.

Finally, add a method called `countIt` to your `WordGames` class. Calling `countIt` should set the display title to "Count Occurrences" and then prompt the user for an input file. After

successfully opening the input file, your method should repeatedly ask the user for a word or phrase and then echo the word or phrase to the display, adding the number of occurences. Below is an example run of the program.




## Additional Credit Opportunities (To earn above 95%)

If you finish early, you may work on the following until the last class day for the assignment. (You earn the 5% by remaining productive and working steadily on one or more of these tasks.)

1. Add a menu that allows you to choose which game you will play
2. Design and implement a word game of your choosing. Some ideas are Jumble and Hangman.

## Scoring

I may look at your code at any time during the development. You will lose points if **you are coding any method that is not commented.** It is not correct to comment after the fact.

During check-off, I will examine one or more of your methods, and then I will ask you to run one or more of your word games.