**Programming Project 2**

**Due Friday, October 16 at 11:59pm Cleveland time**

***IMPORTANT:****Read the****Do's and Dont's****in the****Course Honor Policy****found on Canvas.*

**I. Overview**

This second programming assignment is to give you practice writing loops and manipulating strings and arrays. **Do not use arrays on a problem involving Strings, and do not use Strings in a problem involving arrays.**

**II. Code Readability (20% of your project grade)**

As with the last project, you are required to have code that is easy for another person to read and understand. To accomplish that, the class will have certain rules about how your code should look.

**To receive the full readability marks, your code must follow the following guideline:**

* You should place a comment at the top of the file that contains your name and a short description of the purpose of the class.
* You should place a *short* comment before (directly above) each method describing the method. The comment should be only described *what* the method does, not *how* it does it. Do not simply copy the descriptions below for your comments.
* You should place a short comment directly above each field (should you really have any?) indicating the purpose of the variable.
* You should place a short comment above or to the right of each local variable explaining the purpose of the variable. *Variables that are only used as loop indexes do not need to have comments.*
* You should place a short comment above each loop explaining how the loop works. Ideally, you should list the goal of the loop, any required precondition for the loop, and if you can, a good iteration subgoal for the loop.
* Any other complicated code such as code that has lots of if statements or variables should contain *short* comments to help the reader. The comments can either be above the code fragment or to the right, aligned in a column.
* Remember to use good style: everything should be indented nicely, variables should have good names, there should be a blank line between each method.

**III. Program Testing Document (20% of your project grade)**

As with the previous project, you are required to submit a document demonstrating that you thoroughly tested your program. In this case, it means documenting tests for each of the methods listed below. If you are unable to complete a method above, you should still describe tests that would test the method had it been completed.

Hints for testing loops. Your tests need to, at the minimum cover the following cases:

1. **Test 0, test 1, test many:** This means you have to test cases where the parameters, if they are integers, are 0, 1 or some value other than 1. If the parameters are strings, you have to test strings of length 0, 1, and more than 1. If the strings must contain certain data, you need to test cases where they contain 0, 1, and more than 1 of the desired data.
2. **Test first, last, and middle:** In cases where you have to search in or modify a string, you need to test cases where the item to be found or modified is the first character of the string, the last character of the string, or somewhere in the middle of the string.

**What must go in the report:** For each method below, your report should describe, in English, what "*test 0, 1, many*" and "*test first, middle, last*" mean for each of the methods. Then, you should list the specific tests that you will do, what the expected output is, and then (if you completed the method) a cut-and-paste from DrJava showing the actual test.

**JUnit**: you recently learned about JUnit in lab. JUnit will be required for future homework, and you are welcome to use it with this homework. If you choose to write JUnit tests for your code, you do not need to include the actual tests in your report. Your JUnit file should include comments with each test that link to the report and indicate what you are testing. For example, if your report indicates that the method requires a test with a string of length 0, your JUnit class should have such a test and a comment on the test noting that it is the test of a length 0 string that your report described. Try to organize the JUnit class and report to make it easy for a reader to jump back and forth between the report and the tests.

**IV. Java Programming (60% of your grade)**

**Guidelines for the program**:

* All methods listed below must be **public and static**.
* If your code is using a loop to modify or create a string, you need to use the **StringBuilder class from the API**.
* Keep the loops simple but also efficient. Remember that you want **each loop to do only one "task**" while also avoiding unnecessary traversals of the data.
* No additional methods are needed. However, you may write **additional *private* helper methods**, but you still need to have efficient and simple algorithms. Do not write helper methods that do a significant number of *unnecessary* traversals of the data.
* ***Important***: you must **not use either break or continue** in your code. These two commands are often used to compensate for a poorly designed loop. Likewise, you must not write code that mimics what break does. Instead, re-write your loop so that the loop logic does not need break-type behavior.
* While it may be tempting to hunt through the API to find classes and methods that can shorten your code, you may not do that. The first reason is that this homework is an exercise in writing loops, not in using the API. The second reason is that in a lot of cases, the API methods may shorten the *writing* of your code but increase its *running time*. The **only classes** and methods you can use are listed below. Note: if a method you want to use from the API is not listed, you should *not* implement the method yourself so you can use it. Rather, you should look for the solution that does not need that method.

You *are allowed* to use the following methods from the Java API:

* + class String
    - length
    - charAt
  + class StringBuilder
    - length
    - charAt
    - append
    - toString
  + class Character
    - any method

Create a class called HW2 that contains the following methods:

1. replaceFirstK: takes a String, two chars, and an int as input and returns a String that is the same as the parameter String except that first int occurrences of the first char parameter are replaced with the second char parameter.

> HW2.replaceFirstK("Mississippi River", 'i', 'I', 3)

"MIssIssIppi River"

> HW2.replaceFirstK("Missouri River", 'r', '\*', 3)

"Missou\*i Rive\*"

1. allChars: takes two char parameters. Creates a String containing all characters, in order, from the first char parameter (inclusive) to the last (inclusive).

> HW2.allChars('d', 'm')

"defghijklm"

1. showCharOfString: takes two String parameters. Outputs a new String that is the same as the first String except that: for each character of the first String, if the character is not in the second String, replace that character with the underscore ('\_').

> HW2.showCharOfString("Missouri River", "s SR!r")

"\_\_ss\_\_r\_ R\_\_\_r"

1. hangman: Takes a String and an int and returns a boolean. The method plays the game of hangman where the String is the word the player must guess and the int is the maximum number of "bad guesses" allowed. The game should work as follows:
   1. Creates a StringBuilder that stores all the letters guessed by the player. The StringBuilder should initially be empty.
   2. Has a loop that does the following:
      1. Calls the showCharOfString on the parameter String and a string containing all the guessed letters.
      2. Uses System.out.println to print the String returned from the previous step as well as the total number of "bad guesses" so far. (*For fun, you can print a graphical hangman instead of the number of bad guesses.*)
      3. Uses javax.swing.JOptionPane.showInputDialog to get the next guessed letter from the user. The method should not crash if the user does not enter appropriate data.
      4. If the letter has not yet been guessed, adds the letter to the StringBuilder containing the guessed letters and if the letter is not in the input String, increases the count of "bad guesses" by one.
   3. The loop repeats until either: all letters of the parameter String are guessed or the number of "bad guesses" equals the int parameter.

Finally, the method returns true if the player guessed all the letters of the input String without reaching the limit on "bad guesses", and false if the "bad guesses" limit was reached. You should organize your loop so the return statement is after the loop terminates.

1. hiddenString: takes an array of char and a String as input parameters and and returns an boolean. The method returns true if we can find the input string inside the array by starting at any position of the array and reading either forwards or backwards.

> HW2.hiddenString(new char[]{'a','b','r','a','c','a','d','a'}, "acad")

true

> HW2.hiddenString(new char[]{'a','b','r','a','c','a','d','a'}, "carb")

true

> HW2.hiddenString(new char[]{'a','b','r','a','c','a','d','a'}, "brad")

False

1. hiddenString: takes a 2-dimensional array of char and a String as input parameters and returns a boolean. The method returns true if we can find the input string inside the array by starting at any position of the array and reading in any straight direction (up, down, left, right, or diagonal).

> HW2.hiddenString(new char[][]{{'a', 'b', 'c'},{'r','c','a','d'},{'b','r'}}, "bcc")

true

> HW2.hiddenString(new char[][]{{'a', 'b', 'c'},{'r','c','a','d'},{'b','r'}}, "ace")

false

> HW2.hiddenString(new char[][]{{'a', 'b', 'c'},{'r','c','a','d'},{'b','r'}}, "cad")

True

>HW2.hiddenString(new char[][]{{'a', 'b', 'c','d','e'},{'f','g','h','i','j'},{'k','l','m'},{'n','o','p','q'},{'r'}}, "bcc")

HW2.hiddenString(new char[][]{{'a', 'f', 'j','n','q','s'},{'b','g','k','o','r'},{'c','h','l'},{'d','i','m','p'},{'e'}}, "bcc")

HW2.hiddenString(new char[][]{{'a', 'b','a', 'c'},{'e','r','c','a','d'},{'b','r'}}, "ace")

False

1. **Extra Credit (up to 10%)**

capitalizeWords takes a String as input and returns a String as output. Any word in the input string (a word is defined as a sequence of Enlish letters plus the hypen -) that contains any capitalized letter will have all its letters capitalized in the output string.

> HW2.capitalizeWords("Guess what?? There are twenty-sIx letters in the English alphABEt!")

HW2.capitalizeWords("Guess what?? There are aRe Are twenty-sIx letters in the English aBac zAb Zab zab alphABEt! zaB")

"GUESS what?? THERE are TWENTY-SIX letters in the ENGLISH ALPHABET!"

**V. Submitting Your Project**

Submit the .java file (not the .class files or the .java~ files) of your class plus the testing report on Canvas.

Guess what?? There are twenty-sIx letters in the English alphABEt!