Programming Project 2

Due Friday, March 1 at 11:59pm

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. Arrays are not needed, and *and you should not use arrays in your assignment*.

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 describe *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 the 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 were introduced to JUnit in a recent lab. JUnit will be required for future homeworks, 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 (55% 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 or lots of unnecessary extra memory.
- 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 almost always are 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 shoud look for the solution that does not need that method. You *are allowed* to use the following 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 static methods:

1. **samePrefix** takes two String s and one int parameter and returns a boolean:
   Let x be the input int value (though you should use a better name). The method should return true if the first x characters of each input String are exactly the same.

   samePrefix("this is a test", "this is a trial", 11) should return true
   samePrefix("this is a test", "this is a trial", 12) should return false

samePrefix("this is a test", "This is a trial", 4) should return false
samePrefix("this is a test", "this is a test", 100) should return false

2. **matchingParentheses** takes a String as input and returns a boolean:
   The method should return true if all the parentheses (if any) in the input are properly matched. Any closing parenthesis ')' should be preceded by a matching open parenthesis '(' and there should be exactly one open parenthesis for each closed parenthesis.

   matchingParentheses("This is a (test (of the) (matching)) parentheses") should return true
   matchingParentheses("The (second closing) parenthesis) does not match") should return false
3. **removeEveryKthWord**: takes a String and an int as input and returns a String. The input number is a value *k* such that the output string is the same as the input string except that every _k_th word plus the whitespace immediately following that word is removed. If the input is not a positive number, the output will be the same as the original string.

   removeEveryKthWord("Four score and seven years ago our fore fathers", 3) should return "Four score seven years our fore "
   removeEveryKthWord(" Every Who down in Whoville liked Christmas a lot!", 2) should return " Every down Whoville Christmas lot!"
4. **flipEachK** should take a String and an int as input and returns a String.
   Let *k* be the name of the input integer. The output string should contain the first *k* characters of the input string in order, the next *k* characters of the input string in reverse order, the next *k* characters of the input string in normal order, and so on. The output string should have all the characters of the input string and no others.

   flipEachK("abcdefghijklmn", 4) should return "abcdhgfeijklnm"
5. **reverseDigits** should take a String as input and output a String.
   The output String should be identical to the input String except all digits (0 through 9) of the input String are in reverse order (but same locations) in the output String.

   reverseDigits("0 the d1gits of the2 string3 4 are8 reversed 9!") should return "9 the d8gits of the4 string3 2 are1 reversed 0!"
6. **replaceText** takes two Strings as input and returns a String.
   Each input string will have text with zero or more substrings marked by matching parentheses. The returned string should contain the contents of the first string, except that each substring that is inside parentheses in the first input string is replaced by the matching substring that is inside parentheses in the second input string. (So the first substring of string 1 is replaced by the first substring of string 2, the second by the second, and so on.) If the first input string has more substrings in matching parentheses than the second string does, the substrings are simply removed. If the second string has more substrings in matching

parentheses, these are ignored. Any nested and matched parentheses (matched parentheses inside the outer set of matched parentheses) should be treated as regular characters.

replaceText("a (simple) programming (example)", "(cool) (problem)") should return "a cool programming problem"
replaceText("a ((nested) example) with (three) replacements (to (handle))", "the replacements are (answer) and (really (two) not three)") should return "an answer with really (two) not three replacements "

Note: As in the examples, the behavior for unmatched parentheses is up to you, but your method should not crash.

7. **Extra Credit (10 points): reverseAll** should take a String and return a String
All substrings inside matching parentheses should be reversed. If the reversed portion contains matching parentheses, these should be "re-reversed" and so on. The parentheses should still be correctly matched and nested about the different affected substrings.

   reverseAll("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") should return "a b ((h ((p q) (m n o) l k j i) r s t) (f g) e d c) (w v u) x y z"

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.