Programming Assignment 5: Text Processing

Total Points (50 points) – Due Monday, October 26th at 11:59 PM

This programming assignment is intended to demonstrate your knowledge of the following:

- Writing a while loop
- Write methods and calling methods

Text Processing [50 points]

There are times when a program has to search for and replace certain characters in a string with other characters. This program will look for an individual character, called the ***key character***, inside a ***target string***. It will then perform various functions such as replacing the key character with a dollar-sign ($) wherever it occurs in the target string. For example, if the key character were 'a' and the string were "*He who laughs last, laughs fast, faster, FASTEST."* then the operation of replacing the 'a' by a dollar-sign would result in the new string: "*He who l$ughs l$st, l$ughs f$st, f$ster, FASTEST."*

As you can see, only the lower-case 'a' was detected and replaced. The letter 'a' and 'A' are considered different characters. This is called "*case-sensitivity*" and this entire program spec is **case-sensitive.**
This was only one possible task we might perform. Another would be to **remove** all instances of the key character rather than replace each with a dollar-sign. Yet a third might be to **count** the number of key characters. We are going to do it all.

Modifying vs. Rebuilding

Whenever we deal with strings, we have to decide whether we are going to modify an existing string or create a second string which has the desired changes. Since one cannot modify any string in the **String** class, the first option does not really exist. We will not attempt to touch the original string in our operations. Instead, we will declare a new string object and build that up in stages, replacing or removing the desired characters of the original string by simply taking action on the new string we are building.

When I say we "build" a string, I mean that we initialize the string to be empty, "", and then use ***concatenation*** to replace it with ever longer versions of itself. You will use concatenation to build a string by repeatedly tagging new characters onto its end. This technique has some advantages that allow it to be used for a variety of purposes, so it's good to learn now.

It might seem like we are breaking the rule that **String** objects cannot be modified, but we will not be changing anything. Instead, we will be replacing what the String reference points to, with a slightly longer version of the old String at every phase. For example, consider this statement, which appends an exclamation point to the end of a string, **myStr**:

```
myStr = myStr + "!";
```

This statement uses the old value of **myStr** on the RHS, then completely throws away the old value on the LHS and replaces it with the new, longer, **String**. This is similar to a more familiar kind of numeric statement:

```
n = n + 3;
```

where we replace the old contents of **n** with new contents.

The Methods

We will be writing methods. Some will get input from the user (which take no parameters), and others will take arguments: the **String** and/or ***key character***. Depending on the method we write, it will return one of the following types: a **String**, a **char** or an **int**. For example, one of the methods we write will take the ***key character*** and the ***target string*** as parameters and will return a new **String** which has all the occurrences of the key character replaced by dollar-signs.
Its signature would look like this:

```
public static String maskCharacter(String theString, char keyCharacter)
```

In our spec, this week, we will use input methods to get the input (not **main()**), but we will allow **main()** to do the output directly.

The Program Spec

Ask the user to enter both a ***key character*** and a ***target string*** (phrase, sentence, etc.). Then, show the user three things:

1. The target string with the key character replaced by dollar-signs.
2. The target string with the key character removed.
3. The number of occurrences of the key character (case sensitive) in the target **string.**

Clarification and Other Requirements

- This program does not loop for different strings. Once it processes a string, the program ends.
- Here, "**character**" means ***any*** printable character. They can be letters, numbers or even special symbols.
- Each target input string should be complex with a mix of characters, special symbols, numbers to show how the program handles each category.

All methods that are used will be static.

Input Method Specs

1. **char getKeyCharacter()**

   This method requests a single character from the user and continues to ask for it until the user gets it right: this method will test to make sure the user only types one single character. 0, 2, 3 or more characters will be flagged as an error, and the method will keep at the user until he types just one character. You are not required to use a **char** as an input variable -- in fact, you cannot solve the problem using a char as input (you must think about this and make the appropriate choice here). What matters is that a **char** is returned, as a functional return, to the client, **main()**.

2. **String getString()**

   This method requests a string from the user and continues to ask for it until the user gets it right: this method will test to make sure the user only types a string that has at least 4 characters. Make this minimum size a constant (**final**), and use that symbolic constant, not the literal (4) wherever it is needed. The acquired string will be returned as a functional return.

Processing Method Specs

You must write the methods below from scratch. Do not rely on any other built-in or pre-existing methods that appear to provide any of this functionality for you. There is a high value to you in practicing such logic.

1. **String maskCharacter(String theString, char keyCharacter)**
   This method will take both a string and a character as parameters and return a new string that has each occurrence of the key character replaced by a dollar-sign, '$'.
2. **String removeCharacter(String theString, char keyCharacter)**
   This method will take both a string and a character as parameters and return a new string that has each occurrence of the key character removed, but all other characters left intact.
3. **int countKey(String theString, char keyCharacter)**
   This method will take both a string and a character as parameters, and return the number of key characters that appear in the string (case sensitive).

Input Errors

Whenever the user makes an input error, keep at them until they get it right. Do not return from an input method until you have acquired a legal value, even if it takes years.

Test Run Requirements:

Submit at least three runs. In at least one of the three runs, intentionally commit input errors to demonstrate both kinds of illegal input described above.

CLASS NAME. Your program class should be called *Text Processing.java.*

Sample Run

```
Please enter a SINGLE character to act as key: dgkfpo
Please enter a SINGLE character to act as key: O*(U
Please enter a SINGLE character to act as key: P
Please enter a phrase or sentence >= 4 and <= 500 characters:
sdf
Please enter a phrase or sentence >= 4 and <= 500 characters:
sfd sf jko POIJ  JKOLP  lkjsdf psadfjP sdfj erP sdfp

String with 'P' masked:
sfd sf jko $OIJ  JKOL$  lkjsdf psadfj$ sdfj er$ sdfp

String with 'P' removed:
sfd sf jko OIJ  JKOL  lkjsdf psadfj sdfj er sdfp

# Ps: 4
```

Submission Instructions

- Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. I will run the programs myself to see the output.
- Make sure the run "matches" your source. If the run you submit could not have come from the source you submit, it will be graded as if you did not hand in a run.
- Use the Assignment Submission link to submit the source code file.
- Submit the following file:
    - Text Processing.java
- Do not submit **.class** files.

Standard program header

Each programming assignment should have the following header, with italicized text, appropriately replaced.

```
/*
 * Class: CS1A
* Description: (Give a brief description of Assignment 5)
 * Due date:
 * Name: (your name)
 * File name: TextProcessing.java
```