**Reviewing Assignment**

Lab Assignment 5

|  |  |
| --- | --- |
| Started: | Oct 28, 2014 10:00 AM |
| Finished: | Oct 29, 2014 11:40 AM |

 1 of 1

**Lab Assignment 5     Total Grade: 18   (of possible 20 points)**

**Score: 18   (of possible 20 points)**

**Assignment 5 - Text Processing**

Select one option from below.  All (both) options are worth the same number of points.  The more advanced option(s) are provided for students who find the basic one too easy and want more of a challenge.   Make sure you have read and understood

* both ***modules A*** and ***B*** this week, and
* ***module 2R - Lab Homework Requirements***

before submitting this assignment. Hand in only one program, please.

OPTION A (Basic) Detecting a Key Character in a String

Understand the Application

We would like to demonstrate our ability to control strings and use methods.

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.  This is how we will do things in option A.

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.  There is a second class, called **StringBuffer** which *does* have this ability, but we don't really need to use it.  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.  This is what you will discover and practice in the current assignment:  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)

We will be careful at all stages:  input methods will only deal with user input, and not attempt to do computation.  Computations will not do any input or output.

The exception is always **main()**.  In **main()** we may do input and output directly if we are not required to use a method to do so by the spec.  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 also 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.

This program does not loop for different strings.  Once it processes a string, it ends.

All methods that are used will be static.

**Input Method Specs**

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()**.

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**

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, '$'.

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.

int countKey(String theString, char keyCharacter)

This method will take both a string and a character as parameters, and return the number of key character 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 four runs.    In at least one of the four runs, intentionally commit input errors to demonstrate both kinds of illegal input described above.

A (partial) sample run is given at the bottom of this page.

OPTION B-1 (Intermediate): Case Sensitivity

Augment the basic option as follows:

1. Make the match case-insensitive.
2. Add a fourth function that switches the case of the key character wherever it is found:  if an occurrence is lower-case, change it to upper-case, and vice versa.

**OPTION B-2 (Intermediate):  Words, Not Characters**

Make the key a string rather than a char.  Use Option A's case-sensitivity, as your match criterion, not Option B-1's case-insensitivity. Have a maximum key string length (make that a static class constant) and set it to 5.  Then, modify all the methods that take the char from option A so that they take a string in its place.    All methods have the same meaning as before, but now the thing to be counted, replaced or removed is the key string, not a key char.  (In the masking function, each occurrence of the string will be replaced by a single dollar-sign).  You can change the name of the methods as appropriate (**removeKey()**, rather than **removeCharacter()**, for example).

Sample Output for Option A

Here is an example of a partial run sample:

/\* ---------------------- Sample run ---------------------------------------

Please enter a SINGLE character to act as key: abc

Please enter a SINGLE character to act as key:

Please enter a SINGLE character to act as key: a

Please enter a phrase or sentence >= 4 characters:

He who laughs last, laughs fast, faster, FASTEST.

String with 'a' masked:

He who l$ughs l$st, l$ughs f$st, f$ster, FASTEST.

# as: 5

String with 'a' removed:

He who lughs lst, lughs fst, fster, FASTEST.

------------------------------------------------------------------------ \*/

**Answer**

* text/plain[foothillAssignment5.txt](https://myetudes.org/access/mneme/content/private/mneme/cff3240c-b51c-41f6-80dc-4db4530bdd05/submissions/14983921/572cfd65-7f79-45d5-0082-80c0c741c1f1/foothillAssignment5.txt)

[[https://myetudes.org/ambrosia_library/icons/collapse.gif](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/14983921/list) Model Answer](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/14983921/list)

/\* CS 1A Lab 5

 \* Instructor Solution

 \*/

import java.util.Scanner;

public class Foothill

{

   static final int MIN\_STR\_LEN = 4;

   static final char KEY\_CHAR = '$';

   static Scanner input;

   public static void main(String[] args)

   {

      String userString;

      char keyCharacter;

      input = new Scanner(System.in);

      // get the string from the user

      keyCharacter = getKeyCharacter();

      userString = getString();

      System.out.println("\nString with '" + keyCharacter + "' masked: ");

      System.out.println( " " + maskCharacter(userString, keyCharacter) );

      System.out.println( "\n# "  + keyCharacter + "s: "

         + countKey(userString, keyCharacter) );

      System.out.println("\nString with '" + keyCharacter + "' removed: ");

      System.out.println( " " + removeCharacter(userString, keyCharacter) );

      input.close();

   }

  // method definitions:

   public static char getKeyCharacter()

   {

      String theString;

      // get the string from the user

      do

      {

         System.out.print("Please enter a SINGLE character to act as key: ");

         theString = input.nextLine();

      }

      while (theString.length() !=  1);

      return theString.charAt(0);

   }

   public static String getString()

   {

      String theString;

      // get the string from the user

      do

      {

         System.out.print("Please enter a phrase or sentence "

               + ">= "+ MIN\_STR\_LEN + " characters: " );

         theString = input.nextLine();

      }

      while (theString.length() < MIN\_STR\_LEN);

      return theString;

   }

   public static String maskCharacter(String theString, char keyCharacter)

   {

      String resultString;

      resultString = "";  // we build up the result starting from ""

      for (int k = 0; k < theString.length(); k++)

      {

         if ( theString.charAt(k) == keyCharacter )

               resultString = resultString

            + KEY\_CHAR;

         else

               resultString  = resultString

            + theString.charAt(k);

      }

      return resultString;

   }

   public static String removeCharacter(String theString, char keyCharacter)

   {

      String resultString;

      resultString = "";

      for (int k = 0; k < theString.length(); k++)

      {

         if ( theString.charAt(k) == keyCharacter)

            continue;

         resultString  = resultString

               + theString.charAt(k);

      }

      return resultString;

   }

   public static int countKey(String theString, char keyCharacter)

   {

      int numKeys, k;

      for (k =  numKeys = 0; k < theString.length(); k++)

      {

         if ( theString.charAt(k) == keyCharacter)

            numKeys++;

      }

      return numKeys;

   }

}

/\* ---------------------- Sample run ---------------------------------------

Please enter a SINGLE character to act as key: asdf

Please enter a SINGLE character to act as key: r

Please enter a phrase or sentence >= 4 characters: 3

Please enter a phrase or sentence >= 4 characters: ttt

Please enter a phrase or sentence >= 4 characters: Are we REALLY there yet??#@$

String with 'r' masked:

 A$e we REALLY the$e yet??#@$

# rs: 2

String with 'r' removed:

 Ae we REALLY thee yet??#@$

(more runs supplied by student)

------------------------------------------------------------------------ \*/

**Comments**

Congratulations! You made it to the halfway point, Dmitri.  
  
GENERAL / MAIN() / RUN  
\* You followed the assignment spec well with regards to method signatures.  
\* You are properly using parameter passing and functional returns to pass values between functions.  No globals were used to communicate info to/from method.  
\* You have enough runs and show invalid input as well as valid.  
\* Your variable names are descriptive.  
- All methods after main() are indented too far.  (-.5)  
- Both do/while loops are double indented. The braces should line up with the keyword.  (-1.5)

INPUT METHODS  
\* Good use of loops to ensure that the input is valid.  
- However, both of your user input methods could be made more straightforward and simpler. Had the loop condition been the length test, some of the logic inside the loop could have been eliminated. Take a look at how this was done in the sample solution.   
  
\* Your communication with the user is clear and helpful.  
  
MASKCHARACTER() and REMOVECHARACTER()  
\* Good loops to build the strings.  
\* You are not altering the parameter. Creating a new string variable is the right choice.  
  
COUNTKEY()  
\* You are successfully counting the number of key characters.

 1 of 1

import java.util.Scanner;

public class Foothill

{

static final Scanner inputStream = new Scanner(System.in);

public static void main(String[] args)

{

char keyCharacter = getKeyCharacter();

String targetString = getString();

String maskedString = maskCharacter(targetString, keyCharacter);

String removedCharStr = removeCharacter(targetString, keyCharacter);

int keyCount = countKey(targetString, keyCharacter);

System.out.println("\nString with '" + keyCharacter + "' masked: ");

System.out.println(" " + maskedString);

System.out.println("\n# " + keyCharacter + "s: " + keyCount);

System.out.println("\nString with '" + keyCharacter + "' removed: ");

System.out.println(" " + removedCharStr);

}

public static char getKeyCharacter()

{

char keyCharacter = ' ';

String prompt, userInput;

boolean corInput = false;

do

{

prompt = "\nPlease enter a SINGLE character to act as key: ";

System.out.print(prompt);

userInput = inputStream.nextLine();

if (userInput.length() == 1)

{

keyCharacter = userInput.charAt(0);

corInput = true;

}

}

while(!corInput);

return keyCharacter;

}

public static String getString()

{

String targetString = " ";

final int MINIMUM\_LENGTH = 4;

String prompt, userInput;

boolean corInput = false;

do

{

prompt = "\nPlease enter a phrase or sentence >= "

+ MINIMUM\_LENGTH + "characters:\n";

System.out.print(prompt);

userInput = inputStream.nextLine();

if (userInput.length() >= MINIMUM\_LENGTH)

{

targetString = userInput;

corInput = true;

}

}

while(!corInput);

return targetString;

}

public static String maskCharacter(String theString, char keyCharacter)

{

String maskedString = "";

for (int position = 0; position < theString.length(); position++)

{

char currentChar = theString.charAt(position);

if (currentChar == keyCharacter)

maskedString += '$';

else

maskedString += currentChar;

}

return maskedString;

}

public static String removeCharacter(String theString, char keyCharacter)

{

String removedCharStr = "";

for (int position = 0; position < theString.length(); position++)

{

char currentChar = theString.charAt(position);

if (currentChar != keyCharacter)

removedCharStr += currentChar;

}

return removedCharStr;

}

public static int countKey(String theString, char keyCharacter)

{

int keyCount = 0;

for (int position = 0; position < theString.length(); position++)

{

char currentChar = theString.charAt(position);

if (currentChar == keyCharacter)

keyCount ++;

}

return keyCount;

}

}

/\*----------paste of run from console window------------

RUN 1

Please enter a SINGLE character to act as key: a

Please enter a phrase or sentence >= 4characters:

He who laughs last, laughs fast, faster, LAST, FASTEST.

String with 'a' masked:

He who l$ughs l$st, l$ughs f$st, f$ster, LAST, FASTEST.

# as: 5

String with 'a' removed:

He who lughs lst, lughs fst, fster, LAST, FASTEST.

RUN 2

Please enter a phrase or sentence >= 4characters:

He who laughs last, laughs fast, faster, LAST, FASTEST.

String with 'A' masked:

He who laughs last, laughs fast, faster, L$ST, F$STEST.

# As: 2

String with 'A' removed:

He who laughs last, laughs fast, faster, LST, FSTEST.

RUN 3

Please enter a SINGLE character to act as key: e

Please enter a phrase or sentence >= 4characters:

Add a fourth function that switches the case of the key character wherever it is

found: if an occurrence is lower-case, change it to upper-case, and vice versa

String with 'e' masked:

Add a fourth function that switch$s th$ cas$ of th$ k$y charact$r wh$r$v$r it i

s found: if an occurr$nc$ is low$r-cas$, chang$ it to upp$r-cas$, and vic$ v$rs

a

# es: 18

String with 'e' removed:

Add a fourth function that switchs th cas of th ky charactr whrvr it is found:

if an occurrnc is lowr-cas, chang it to uppr-cas, and vic vrsa

RUN 4

Please enter a SINGLE character to act as key: test

Please enter a SINGLE character to act as key:

Please enter a SINGLE character to act as key: l

Please enter a phrase or sentence >= 4characters:

He who laughs last, laughs fast, faster, LAST.

String with 'l' masked:

He who $aughs $ast, $aughs fast, faster, LAST.

# ls: 3

String with 'l' removed:

He who aughs ast, aughs fast, faster, LAST.

----------------------------------------------------- \*/