Lab2 - Fun with Strings

Reminder: You must first have your completed lab checked off by your TA (either in-person or by e-mail). Afterwards, you must submit your lab via Brightspace. Failure to do either will result in a grade of 0.

Goals

In this Lab you will:

1. Write a program that accepts text-based *commands* that have an effect inside the program.
2. Explore various manipulations on an instance of type `java.lang.String`.

Resources

You will want to refer to each and all of the following to complete this Lab:

1. **Lab Overview** (this document).
2. **StringFunLab.java**, the **Starter Code** for this Lab (attached).
3. **Sample Exchange**, showing a sample input and output exchange between the program and a user running it on the command line, once the program is correctly written (attached).

Background

Before Microsoft Windows and Apple Macintosh made Graphical User Interfaces (`GUI`s) common, computer users exclusively interacted with programs through text - offering menus or text prompts entirely in text, and then accepting input typed by the user on a keyboard. Though dated, such programs can still be written today, and tend to be much simpler for beginner programmers to make.

When you are done with today's Lab, you will have a program that holds a single `String` in memory, and allows the user to make alterations to that `String`. We do this by repeatedly asking the user to type a **command**. For instance, if the user enters "`printString`", we will simply print the in-memory `String` to `System.out`. If they type "`reverseString`", we will update the in-memory String to be the mirror image of what it used to be. You can see a list of the commands you will implement in the next section.

Some commands take **arguments**, which just means there are two or more command "words" entered at the same time. The first is the `command` itself and the rest affect what the `command` does. For instance, if the user enters "`enterNewString bob`", "bob" is an argument to the command `enterNewString` - it will replace the in-memory `String` to be "bob" instead.

If the user enters a `command` that isn't recognized, or provides the wrong number or type of arguments, we don't need to guess what to do. Instead, we can simply tell the user, "Error, Invalid Command", and then print a small help snippet explaining the valid commands and how to use them. A function doing this has already been written for you in the **Starter Code**.

Finally, note that in Java, `String`s are "immutable" - which means none of the methods provided by `String` will actually alter the `String` itself, but will instead create a new `String` with the

requested change applied. To carry over the result, you have to reassign using `=`.

For instance suppose one of the commands was to capitalize all letters in the `String`. If the `String` were stored in variable `str`, you would say:

```
str = str.toUpperCase();
```

Merely calling `str.toUpperCase();` would not be sufficient, and `str` would end up unaffected.

What To Do, And How

Add code to `StringFunLab.java` so that all of the commands in the following section are successfully implemented. If the user enters a command incorrectly, or an incorrect command, notify them of their error, and print the help snippet (see **Starter Code**, this has been partially already handled for you).

Static functions already exist for each of the commands, you simply need to fill in the functions with code. We've already done this for you for `quit`, `enterNewString`, and `printString`.

Some of these commands can be easily implemented using operations on `String`s that we've seen in lecture. Others will require you to be a little creative.

When you are done, if you enter the exact same commands as in the **Sample Exchange**, you should see the exact same results.

Table of Commands

For the below, the Example always assumes you are applying the command to a `String` with value "abc def" as the initial value.

Command	**Description**	**Example**	**Example Result**
`searchText [text]`	Finds the first occurrence of [text] in the `String`, and prints the index in the `String` where it was found; or prints -1. The `String` is left untouched.	`searchText c`	"abc def" (i.e. unchanged - also prints "2")
`removeText [text]`	Removes any occurrence of [text] in the `String`	`removeText bc`	"a def"
`addText [i] [text]`	The first argument is an integer between 0 and the length of the string inclusive; adds [text] at that location in the string	`addText 3 zzz`	"abczzz def"

`reverseText`	Makes the string into its mirror image	`reverseText`	"fed cba"
`reverseEachWord`	Like reverseText but applies to each command word individually	`reverseEachWord`	"cba fed"
`printString`	Prints the current value of the `String` to `stdout`	`printString`	"abc def" (i.e. unchanged - also prints "abc def")
`enterNewString [text]`	Overwrites whatever the string was with [text] instead.	`enterNewString zzz`	"zzz"
`quit`	Special command - exits the program	`quit`	N/A

Hints and Notes

The hard part of this Lab has actually already been done for you with all of the code in the **Starter Code**. Your job is to fill in the different static functions with the appropriate code. Look for the TODO tags and fill in the missing code.

```
import java.util.Scanner;

public class StringFunLab
{
    public static final String START_STRING = "abc def";

    /**
      * Prints the provided `errorMessage`, then reminds the
      * user what all the commands are and how to use them.
      * You do not need to edit this function.
      */
    public static void printHelp(String errorMessage)
    {
        System.err.println(errorMessage);
        System.err.println("Usage - enter one of the following Commands:");
        System.err.println();
        System.err.println("searchText [text]: prints the index of [text] in "
                           + "the String, leaves it unchanged.");
        System.err.println("removeText [text] - deletes all occurences of "
                           + "[text] in the String.");
        System.err.println("addText [i] [text] - first argument is an integer "
                           + "between 0 and and the length of the string; "
                           + "adds [text] at that location in the string.");
        System.err.println("reverseText [no argument] - makes the string into "
                           + "its mirror image.");
        System.err.println("reverseEachWord [no argument] - like reverseText "
                           + "but applies to each command word individually.");
        System.err.println("printString [no argument] - prints the current "
                            + "value of the string.");
        System.err.println("enterNewString [text] - overwites whatever the "
                           + "string was with [text] instead.");
        System.err.println("quit [no argument] - exits the program.");

        System.err.println("");
    }

    /**
      * Implements the "printString" command.  This command doesn't change
      * theString, so we just return what it was.
      */
    public static String printString(String oldString, String[] commandWords)
    {
        System.out.println(oldString);
        return oldString;
    }

    /**
      * A useful function, combines all of the words in `words` starting from
      * `index`.  You do not need to edit this function.
      */
    private static String combineWordsFrom(String[] words, int index)
    {
        String newString = "";
        for(int i = index; i < words.length; i+=1)
```

```
        {
            if(i > index)
            {
                newString += " ";
            }
            newString += words[i];
        }
        return newString;
    }

    public static String searchText(String oldString, String[] commandWords)
    {
        // TODO: You should search the string "oldString" for the text found in "
            commandWords" starting from index 1 to the end of the string array
        // 1. First check if the length of commandWords is less than 2. If it is, then
             print an error using printHelp and return null.
        // 2. Get the text you need to search oldString for. Use combineWordsFrom(
            commandWords,1) to get the appropriate text.
        // 3. Print out the index where the text is found in oldString (or -1 if not
            found). Use the String function indexOf for this.
        // 4. Return null (since searchText does not change the current string).
    }

    public static String removeText(String oldString, String[] commandWords)
    {
        // TODO: You should remove from "oldString" all instances of the text found in
             "commandWords" starting from index 1 to the end of the string array
        // 1. First check if the length of commandWords is less than 2. If it is, then
             print an error using printHelp and return null.
        // 2. Get the text you need to remove from oldString. Use combineWordsFrom(
            commandWords,1) to get the appropriate text.
        // 3. Remove the text from oldString using the String function replaceAll.
        // 4. Return the String that is returned from the replaceAll function.
    }

    public static String addText(String oldString, String[] commandWords)
    {
        // TODO: You should add text to "oldString" at a specific index
        // 1. First check if the length of commandWords is less than 3. If it is, then
             print an error using printHelp and return null.
        // 2. Get the text you need to add to oldString. Use combineWordsFrom(
            commandWords,2) to get the appropriate text.
        // 3. Get the index using "Integer.parseInt(commandWords[1])".
        // 4. Check if the index is in the correct range (greater than 0, and less
            than or equal to oldString's length). If not, print an error using
            printHelp and return null.
        // 5. Return oldString.substring(parameters) + text + oldString.substring(
            parameters) using the correct parameters for substring.
    }

    private static String reverseString(String s)
    {
        // TODO: Make the utility function reverseString which reverses a single
            string
```

```
    // 1. Create a new StringBuilder object, and initialize it with "s"
    // 2. Call reverse() on the StringBuilder object, then toString()
    // 3. Return the resulting String from step 2
    // Note: An alternative way of doing this is to traverse the string in reverse
        using a for loop and the function charAt, and to build the string in
       reverse character by character
}

public static String reverseText(String oldString, String[] commandWords)
{
    // This function is already completed, as long as reverseString is correctly
       implemented.
    // Note: It was a design choice to not error out if arguments were erroneously
        included.

    return reverseString(oldString);
}

public static String reverseEachWord(String oldString, String[] commandWords)
{
    // TODO: Reverse each individual word in oldString using the reverseString
       function which you filled in
    // 1. Call split(" ") on oldString (split up the string on space), and store
       the result in a String array
    // 2. Create a new empty string called result (i.e., String result = "")
    // 3. Create a for loop and iterate over each element of the String array from
        step 1
    // 4. Within the loop, if the index is greater than 0, add a blank space to
       result (i.e., result += " ")
    // 5. Within the loop, add the reversed version of the array String to result
       (i.e., result += reverseString(array[i]))
    // 6. Return result
    // Note: It was a design choice to not error out if arguments were erroneously
        included.
}

/**
  * Implements the "enterNewString" command.  You probably want to
  * create a similar function for each string command.  Note that
  * `oldString` is ignored by this particular command/function.
  */
public static String enterNewString(String oldString, String[] commandWords)
{
    // This command needs an argument after the command; if it's missing
    // that's an error.
    if (commandWords.length < 2)
    {
        printHelp("enterNewString - requires an argument");
        return null;
    }

    return combineWordsFrom(commandWords, 1);
}
```

```
/**
  * This method should return the result of applying the string command
  * represented by `commandWords`, or else null (if the command was not
  * recognized or the wrong arguments were supplied).  When returning
  * null, an appropriate error message should first be printed (use
  * printHelp()).
  */
public static String handleStringCommand(String oldString, String[] commandWords)
{
    if ("searchText".equals(commandWords[0]))
    {
        return searchText(oldString, commandWords);
    }
    else if ("removeText".equals(commandWords[0]))
    {
        return removeText(oldString, commandWords);
    }
    else if ("addText".equals(commandWords[0]))
    {
        return addText(oldString, commandWords);
    }
    else if ("reverseText".equals(commandWords[0]))
    {
        return reverseText(oldString, commandWords);
    }
    else if ("reverseEachWord".equals(commandWords[0]))
    {
        return reverseEachWord(oldString, commandWords);
    }
    else if ("printString".equals(commandWords[0]))
    {
        return printString(oldString, commandWords);
    }
    else if ("enterNewString".equals(commandWords[0]))
    {
        return enterNewString(oldString, commandWords);
    }
    else
    {
        printHelp("Unknown command: "+commandWords[0]);
        return null;
    }
}

public static void main(String args[])
{
    String theString = START_STRING;

    Scanner s = new Scanner(System.in);
    boolean run = true;

    do
    {
        System.out.print("Enter Command: ");
```

```
            String line = s.nextLine();
            String[] commandWords = line.split(" ");

            // Handle "quit" specially.
            if ("quit".equals(commandWords[0]))
            {
                run = false;
            }
            else
            {
                // Everything besides quit should be a command that alters
                // `theString`.
                String newString = handleStringCommand(theString, commandWords);
                if (newString != null)
                {
                    theString = newString;
                }
            }
        } while(run == true);
    }
}
```

This is an example of running the program, including the text that prints out when using `System.out.print` and `System.out.println` and the actual text the user typed into the program (formatted **`like this`** below):

Enter Command: **help**
Unknown command: help
Usage - enter one of the following Commands:

searchText [text]: prints the index of [text] in the String, leaves it unchanged.
removeText [text] - deletes all occurences of [text] in the String.
addText [i] [text] - first argument is an integer between 0 and and the length of the string; adds [text] at that location in the string.
reverseText [no argument] - makes the string into its mirror image.
reverseEachWord [no argument] - like reverseText but applies to each command word individually.
printString [no argument] - prints the current value of the string.
enterNewString [text] - overwites whatever the string was with [text] instead.
quit [no argument] - exits the program.

Enter Command: **printString**
abc def
Enter Command: **searchText de**
4
Enter Command: **addText 4 zzz**
Enter Command: **printString**
abc zzzdef
Enter Command: **reverseText**
Enter Command: **printString**
fedzzz cba
Enter Command: **reverseEachWord**
Enter Command: **printString**
zzzdef abc
Enter Command: **removeText zde**
Enter Command: **printString**
zzf abc
Enter Command: **enterNewString programming**
Enter Command: **printString**
programming
Enter Command: **addText -1 invalid**
addText - invalid index (must be int between 0 and string length): -1
Usage - enter one of the following Commands:

searchText [text]: prints the index of [text] in the String, leaves it unchanged.
removeText [text] - deletes all occurences of [text] in the String.

addText [i] [text] - first argument is an integer between 0 and and the length of the string; adds [text] at that location in the string.
reverseText [no argument] - makes the string into its mirror image.
reverseEachWord [no argument] - like reverseText but applies to each command word individually.
printString [no argument] - prints the current value of the string.
enterNewString [text] - overwites whatever the string was with [text] instead.
quit [no argument] - exits the program.

Enter Command: **removeText**
removeText - requires an argument
Usage - enter one of the following Commands:

searchText [text]: prints the index of [text] in the String, leaves it unchanged.
removeText [text] - deletes all occurences of [text] in the String.
addText [i] [text] - first argument is an integer between 0 and and the length of the string; adds [text] at that location in the string.
reverseText [no argument] - makes the string into its mirror image.
reverseEachWord [no argument] - like reverseText but applies to each command word individually.
printString [no argument] - prints the current value of the string.
enterNewString [text] - overwites whatever the string was with [text] instead.
quit [no argument] - exits the program.

Enter Command: **quit**