**CS 273 Laboratory 2: String Operations**

**Part A: Run the initial project; Examine the Java file**

* Open up the lab2 project.
* Compile and run Lab2App. When running the program, you may have to resize the window to see the whole thing. The window should contain several text fields. You can type something into the first one, then hit Enter. The text that you typed will immediately be replicated in all the other text boxes. Some of the windows might have different colors.
* Right-click or double-click on Lab2.java to open the editor window. Notice there are 12 methods defined in Lab2.java. In each one, you are given a parameter called input, which is the String that the user has typed in.
* Each method returns a String, which is the String that will appear in one of the text boxes.
* Whenever you hit the **Enter** key, the program calls all of the methods, and puts the resulting strings in the respective output text boxes. In addition, it sets the color of each window to indicate whether the result is correct according to Prof. Vegdahl's implementation of the same function.
  + Note: the text in the output boxes is surrounded by a pair of double-quotes so that you can see where the string begins and ends—so that you can see “invisible” spaces.
* The color meanings are:
* **White:** Correct
* **Yellow:** Incorrect
* **Red:** Your version did something illegal in Java, such as accessing a string out-of-bounds.
* **Black:** Your version did something illegal, but so did Vegdahl’s. This is probably because the operation was not possible, like taking the second word of a string that only has one.
* **Light blue:** Prof. Vegdahl’s version did something illegal, and yours did not. This could be because you are doing more error-checking than Vegdahl’s. However, it could also be because you did something wrong, too.

In summary:

* White and black are good, because you got the same behavior as Prof. Vegdahl.
* Yellow and red are bad, because the professor got a legitimate answer and you got a different answer or no answer at all.
* Light blue requires closer examination.

**Part B: Write the Java code to implement the various operations:**

The remainder of this lab will be one where you fill in the code for the methods Lab2.java. In each case, the code that you write should replace the statement

return input;

In all cases, you should first trim the string. Trimming removes all whitespace such as spaces and tabs from the beginning and the end. Note, most String methods, including trim(), return the updated String. So, you want to store the result of the method in a String and then use that updated String value going forward.

You should test each method with several input strings before you claim that it is done correctly.  To test your changes, right click on Lab2.java and select Compile. Then right click on Lab2StartApp and select the main method.

For some inputs, the method won’t make sense. For example, you cannot remove ten characters from a string that only has seven. In these cases, it is generally OK for the operation to “bomb”.

If you get stuck on a problem, feel free to move ahead and come back to it.

**The methods are described below:**

**Part 1:**

stringTrim: Produce a trimmed string. That is, one where the whitespace at the beginning and end is removed.

stringUpperCase: Produce a (trimmed) string where all lowercase characters are converted to uppercase.

**checkpoint 1 (10 points): Have your lab instructor or assistant confirm that your stringTrim and stringUppercase are working properly.**

**Part 2:**

stringReplacePeriodWithQuestion: Produce a string in which every period is replaced by a question mark. For example, if the input is “hello.”, then the output should be “hello?”.

**checkpoint 2 (10 points): Have your lab instructor or assistant confirm that your stringReplacePeriodWithQuestion is working properly.**

**Part 3:**

stringAppendLength: Add the length of the trimmed string to the end of the (trimmed) string in parentheses.

**checkpoint 3 (10 points): Have your lab instructor or assistant confirm that your stringAppendLength is working properly.**

**Part 4:**

stringMinusFirstTenChars: Produce a (trimmed) string in which the first 10 characters have been removed. You do not have to handle the case where the string’s length is less than 10.

**checkpoint 4 (10 points): Have your lab instructor or assistant confirm that your stringMinusFirstTenChars is working properly.**

**Part 5:**

stringMinusLastTenChars: Produce a (trimmed) string in which the last 10 characters have been removed. You do not have to handle the case where the string’s length is less than 10.

**checkpoint 5 (10 points): Have your lab instructor or assistant confirm that your stringMinusLastTenChars is working properly.**

**Part 6:**

stringInsert: Produce a (trimmed) string where two ‘-’ characters are inserted: one after the first character in the string, and one after the second. For example, when given abcdef, it would produce a-b-cdef. You do not have to handle the case where the string’s length is less than two.

**checkpoint 6 (10 points): Have your lab instructor or assistant confirm that your stringInsert is working properly.**

**Part 7:**

stringFirstWord: Produce a string that contains only the first word of the original (trimmed) input. (A "word" being a consecutive sequence of non-blank characters. You may assume that the input contains no whitespace other than spaces.) For example, if the input is butter! and jam, then the output should be butter!. You may assume there are at least two words. For the extra credit checkpoint below, you could update this to handle just one word of input, but come back and do that (if you want) after you’ve completed the other checkpoints.

**checkpoint 7 (10 points): Have your lab instructor or assistant confirm that your stringFirstWord is working properly.**

**Part 8:**

stringLowerCaseExceptLastWord: Produce a (trimmed) string in which all letters are converted to lowercase, except those in the last word. The last word should remain unchanged. For example, if the input is Fish Have grEEn LeGs, then the output should be fish have green LeGs.

**checkpoint 8 (10 points): Have your lab instructor or assistant confirm that your stringLowerCaseExceptLastWord is working properly.**

**Part 9:**

stringMiddleEllipsis: Produce a (trimmed) string three dots are inserted exactly in the middle of the string. If the string has an odd number of characters, then insert the dots after the middle character.

For example, if the input is muffin, the output should be muf...fin. If the input is barbed-wire, the output should be barbed...-wire. Can you think of better test strings that make it even easier to quickly tell whether the “…” is in the right place?

Complete this task without using an if statement. Hint: try adding 1 to the string’s length before dividing it by 2. Be prepared to explain why this works! It might help to write out a few examples to show the instructor or TA.

**checkpoint 9 (10 points): Have your lab instructor or assistant confirm that your stringMiddleEllipsis is working properly, and show them your explanation (with examples) of why it works.**

**Part 10:**

stringAllButSecondToLastWord: Produce a (trimmed) string that is identical to the input, except that the next-to-last word (and the space that follows it) is removed. For example, if the input is I love chewy bubble gum, then the output should be I love chewy gum.

Hint: your code from Part 8 already separates the last word from the rest of the string. How could you use that technique again here?

**checkpoint 10 (10 points): Have your lab instructor or assistant confirm that your stringAllButSecondToLastWord is working properly.**

**Extra Credit**

The remaining portion of the lab is optional and can be completed in any order for extra credit.

stringFirstThreeQuarters: Produce a (trimmed) string that contains only the first 75% of the characters that are in the input. If the number of characters is not an exact multiple of 4, "round up" to the whole number that is above 75%, but is as close as possible.

For example, if the input is the 12-character string Jacob Marley, then the return value should be the 9-character string Jacob Mar. If the input is the 10-character string Jackhammer, then the return value should be the 8-character string Jackhamm.

Complete this task without using an if statement.

**checkpoint EC1 (5 points): Have your lab instructor or assistant confirm that your stringFirstThreeQuarters is working properly.**

If you'd like, perform some extra error-checking to prevent your operations from doing illegal things. Some of these may require that you use a condition expression -- or if statement -- which we have not yet studied. If you do these correctly, you'll get white text boxes for all kinds of interesting test cases. You can work on some or all of the following in any order.

1. For stringMinusFirstTenChars, make sure that there are at least 10 characters; if not, remove all the characters that are there.
2. For stringMinusLastTenChars, make sure that there are at least 10 characters to lop off; if not, remove all the characters that are there.
3. For stringInsert, make sure there are at least two characters. You determine what should be done if there are not at least two characters.
4. For stringAllButSecondToLastWord, if there is only one word (or no words), it should return the original (trimmed) string.
5. Any other error checking you can think of that improves the robustness of the program. You can get a maximum of two extra credit points regardless of how many methods you update for this one.

If you check the “w/bounds-checks” check-box, it will show the professor’s algorithm that has been enhanced to do the above extra credit (1-4). Otherwise, it will just do the “basic” behavior, which bombs in some situations.

**Checkpoint EC2 (2 points for each correctly updated method): Demonstrate your improvements to your lab instructor or assistant.**