Week 1 feedback and quiz answers

(read only after passing the programming assignment and the quiz)

Congratulations String Master!

Congratulations on completing Week 1 of Data Structures: Measuring and Optimizing Performance! To complete Week 1 it takes not only a sound understanding of strings and how to manipulate them, but also a lot of perseverance, diligence, and patience, all important skills needed for being successful in the field of Computer Science! If you found yourself struggling with this Week's assignment or quiz, that's okay! Before going on to Week 2, think about what you struggled with and try to narrow in on where and why you got stuck. Did you have a lot of syntax issues when you compiled your program? Maybe you should review Java syntax before diving into Week 2. Did you struggle with coming up with the algorithm to solve the question? Maybe try writing out on paper your algorithm and running through possible test cases before coding on a computer. If you struggled with Week 1's quiz, try out these helpful tips:

1. In order to not be overwhelmed by code, try drawing pictures and memory models in order to see what happens as the code is executed. Keep track of the different variables as they increase, decrease, and change over time. If the code still appears overwhelming, look at the code again line by line and make sure you completely understand how each line contributes to the code at large.

2. The programming quizzes focus a lot on concepts, so if there is a question on the quiz that you continuously get wrong or only guessed the correct answer, go back and look at the programming assignment and the lessons. Concepts in Computer Science overlap and will show up again, so make sure you understand the core of what is going on.

If working with strings and regular expressions really interested you, continue on learning by exploring other String methods and making your own projects regarding strings! If this week did not interest you, don't worry because next week you will be learning about a new topic: Efficiency!

Before you begin next week, here are some more helpful hints:

Have you contributed to the discussion forum? If not, try to contribute to the forum and post your questions and/or answers to other people's questions. Professors and mentors are constantly looking at this forum and it's great to see learners helping other learners. Additionally, even if you think you might not know the answer, contribute anyways and hopefully you all can help each other understand concepts and find solutions.

Refer a friend to this course! The more learners the better because more people can contribute to discussions.

The rest of this reading contains the explanations for the answers to this week's quiz questions. Correct answers are in **bold**. Incorrect answers are shown in *italics*.

Good luck and have fun learning!

Sincerely,

Our MOOC Team

Quiz answers and explanations

Correct answers are in **bold**. Incorrect answers are shown in *italics*.

1. What will be printed by the following code? Of course, you can run this code and get the question right, but we want you to answer this question (and all of these questions) without running the code.

```
String s1 = new String("String 1");
String s2 = "String 1";
if (s1 == s2) {
    System.out.println("Equal");
}
else {
    System.out.println("Not equal");
}
```

- **Not equal (correct)**

The == operator compares object references. Since two objects are created (one interned and one other String), their references are not the same. To compare the actual characters in the Strings, you need to use the .equals method.

- *Equal (incorrect)*

The == operator compares object references. Since two objects are created (one interned and one other String), their references are not the same. To compare the actual characters in the Strings, you need to use the .equals method.

2. Which of the following lines of code correctly assign a String containing the text "My String" to the variable 'text'?

(Select all correct options.)

-

```
String text = "My ";String s2 = "String";text = text + s2;
```

(Correct) This will append "My " with "String" to create a new String, which is stored back in the variable text.

-

```
String s1 = "My String";
String text = s1;
```

(Correct) In the the first line, s1 references a String object containing the text "My String". In the second line, the variable text is created and set to reference the same object.

-

```
String text = "My ";
text.concat("String");
```

(Note that in our videos we might have accidentally called the method append--the correct method name is "concat").

(Incorrect) Strings are immutable. So the call to text.concat returns a new string (which is ignored), but does not modify text.

-

```
String text = new String("My ");text + new String("String");
```

(*Incorrect*) Strings are immutable. So the second line returns a new String (which is ignored), but does not modify text.

3. What best describes the functionality of the following method?

```
public int mystery(String s)
{
    char[] letters = s.toCharArray();
    int x = 0;
    for (int i = 0; i < letters.length; i++) {
        if (letters[i] == ' ') {
```

```
        if (letters[i] == ' ') {
            letters[i] = '_';
            x++;
        }
    }
    return x;
}
```

- **It counts the number of spaces in a given string, and returns the count. It does not change the original String. (Correct)**

This is correct. toCharArray returns a copy of the array of characters in a string, so modifying that array will not modify the original String.

- *It counts the number of spaces in a given String and returns the count. It also replaces all spaces in the given String with underscore characters ('_'). (Incorrect)*

Remember that Strings are immutable. toCharArray returns a copy of the array of characters in a string, so modifying that array will not modify the original String.

- *It counts the number of total characters in the String and returns the count. It also replaces all spaces in the given String with underscore characters ('_'). (Incorrect)*

Remember that Strings are immutable. toCharArray returns a copy of the array of characters in a string, so modifying that array will not modify the original String. Also, notice the if statement that is causing x to increment only in certain cases.

4. Assume you have a String variable s that stores the text "%one%%two%%%three%%%%"

Which of the following calls to s.spilt will return the String array: ["%", "%%", "%%%", "%%%%"]

(Select all correct options.)

- *s.split("%+"); (Incorrect)*

This will do the opposite of what we want, splitting on the sequences of dollar signs and leaving the text (and empty strings).

- **s.split("[a-z]+"); (Correct)**

This will work because it will split the string on any sequence of one or more lowercase characters.

- **s.split("one|two|three"); (Correct)**

This will work, but it is not as general as the other solution in this list.

- *s.split("[one,two,three]"); (Incorrect)*

This will split the string on any one of the literal characters of [onetwhr,]. So the array will have

This will split the string on any one of the literal characters of [oheiwnf,]. So the array will have the "$", "", "$", and "" as desired, but it will also have a bunch of empty strings.

5. Assume that you have a Document object stored in the variable d, whose whole text string is

"one (1), two (2), three (3)"

Which of the following calls to getTokens will return the list of Strings given here:

["one", "(1)", "two", "(2)", "three", "(3)"]

Notice the commas are not preserved in the strings in the list.

(Select all correct options.)

- **d.getTokens("[^,]+"); (Correct)**

Notice there is a space after the comma in the character class.

This will work and split the string at contiguous sequences of commas and/or spaces.

- *d.getTokens("[^,]+"); (Incorrect)*

Notice there is NO space after the comma in the character class.

This will not allow us to obtain the desired result because we want to split the String at the spaces and since spaces are legally included as part of each token, this will only split at commas.

- **d.getTokens("[a-z()0-9]+"); (Correct)**

Hint: The ( ) will be treated as literal tokens inside the character set.

In this case getTokens is looking for continuous sequences of lowercase letters, digits or open/closed parens, but NOT spaces or commas, which is what we want.

- **d.getTokens("[a-z]+|[()0-9]+"); (Correct)**

Hint: The ( ) will be treated as literal tokens inside the character set.

In this case getTokens is looking for continuous sequences of lowercase letters or continuous sequences of digits and open or closed parentheses, which again will work in this case.