Strings and Regular Expressions

7/7 questions correct

Excellent!

Retake

Next (/learn/data-structures-optimizing-performance/supplement/CJcnF/did-you-have-trouble-with-any-question-s-on-the-practice-quiz)

1.

How many String objects are created in the following piece of code?

```
String s1 = "This is a string";
String s2 = new String("This is a string");
String s3 = s2;
String s4 = "This is a string";
```

(Give a numeric answer. For example, if you think one String object is created, type 1, not one.)

2

Well done!
There are two objects created: one interned String created on the first line, and a second String object created on the second line. The third line creates another reference to the String created on the second line (but not a new object), while the fourth line creates another reference to the interned string (but not a new object).

2.

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 the other questions in this quiz) without running the code.

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

- Equal
- Not equal

Well done!
The == operator compares object references. Since two objects are created, their references are not the same. To compare the actual characters in the Strings, you need to use the .equals method.

3.

Consider the line of code:

```
String string1 = new String("Coursera");
```

What line(s) of code can you use to assign the char 's' to the variable letter?

(Select all correct options)

- [ ] `char letter = string1.charAt(4);`

Well done!
The charAt method will return the character in the String at that index, starting with index 0.

- [ ] `char letter = string1.charAt(5);`

Well done!
The charAt method will return the character in the String at that index, starting with index 0. So the letter at index 5 is actually 'e'.

- [ ] `char letter = string1[4];`

Well done!
Strings are objects that store arrays of characters, but not actually arrays. You cannot reference individual characters using array notation.

- [ ] `char letter = string1[5];`

Well done!
Strings are objects that store arrays of characters, but not actually arrays. You cannot reference individual characters using array notation.

4.

What does the following code print?

Note that in the videos we might have mistakenly used the method "append", but the correct name for this method is "concat" which is shown below in the code. There is no method "append" in the Java string class.

```
String s = "Hello";
s.concat(" World!");
System.out.println(s);
```

- Hello

Well done!
Strings are immutable. So the call to s.concat returns a new string (which is ignored), but does not modify s.

- Hello World!
- Hello World!Hello

5.

Consider the following String declaration:

```
String text = "Practice, practice, practice!";
```

What is returned by the call:

```
text.indexOf("prac");
```

(Type your answer as a number, not a word. For example, if you think two is the answer, type 2 rather than two.)

10

Well done!
The substring "prac" first occurs at index position 10.

6.

What are the contents of the array 'words' at the end of the following code

```
String text = "Hurray!!#! It's Friday! finally...";
String[] words = text.split("!+");
```

- `["Hurray", "#", " It's Friday", " finally..."]`

Well done!
Notice that all of the characters between one or more ! are preserved as separate Strings.

- ○ `["Hurray!!", "#!", " It's Friday!", " finally..."]`
- ○ `["Hurray", "", "#", " It's Friday", " finally..."]`
- ○ `["Hurray!!#! It's Friday! finally..."]`

✔ 7.

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

"this is a test.23,54,390."

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

["this is a test", "23", "54", "390"]

(Select all correct choices.)

- ☐ `d.getTokens("[a-z0-9 ]+");`

Well done!
This will preserve any contiguous String of lowercase letters, spaces or digits. Notice that if we had spaces between the numbers, those spaces would also be included in the final list. I.e. if the text had been "this is a test. 23, 54, 390." the list returned would be ["this is a test", " 23", " 54", " 390"].

- ☐ `d.getTokens("[^.,]+");`

Well done!
This will preserve any contiguous String of characters that are not . or , Notice that if we had spaces between the numbers, those spaces would also be included in the final list. I.e. if the text had been "this is a test. 23, 54, 390." the list returned would be ["this is a test", " 23", " 54", " 390"]

- ☐ `d.getTokens("[a-z0-9]+");`

Well done!
This is preserve only contiguous strings of lowercase letters and digits, not spaces. So the words in the sentence will be broken apart, as follows: ["this", "is", "a", "test", "23", "54", "390"]

- ☐ `d.getTokens("[^ ]+");`

Well done!

This will preserve contiguous Strings of characters which are not spaces. But those characters do include punctuation. So it would return ["this", "is", "a", "test.23,54,390"]

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

Well done!
This will preserve any contiguous String of lowercase letters and spaces, or continuous String of digits. Notice that if we had spaces between the numbers, those spaces would **not** be preserved and the numbers would still be split as shown.