**Unit 2: Using Objects**

**Topic 7 Lab 1: String Methods**

| **Name:** |  | | |
| --- | --- | --- | --- |

**Create a new IntelliJ project, e.g. LASTNAMEU2T7Lab1,**

**and add a runner class (name it whatever you want) with a main method.**

| **Exploring String Methods** | | |
| --- | --- | --- |
| A String holds characters in a sequence.  Each character is at a position, or **index,** which starts with **0**. An **index** is a number associated with a position in a String (sort of like an array or list, although a string is *not* an array or list). Example:  **String str = "AP CSA is awesome!"**    For example, the character “C” is at **index** **3**, and the character at **index** **11** is a “w”.  The **length** of a String is the number of characters in it *including any spaces or special characters*, which takes us to the first String method that you need to know: **length(), which returns an int**    All methods above are *instance* methods, so you call them on the String object itself.  **Example:**  String **myString** = "AP CSA is awesome!"; // creating a String object  int len = **myString**.length(); // calling the length() method on the object  System.out.println(len);  **Prints: 18** (**not** **17**, because it’s 0-indexed; the first character is at index 0 and the last at index 17 -- which is *actually* 18 in length! If you don’t believe it, count the boxes above for *AP CSA is awesome!*) | | |
| **Determine** the output of this code **without** running it in IntelliJ. Note the escape sequences in 6 & 7!  String str1 = "Hello!"; System.out.println(str1.length());  String str2 = " H e l l o ! ";  int len = str2.length();  System.out.println(len);  String str3 = "Is tax 8.5%?";  System.out.println(str3.length());  String str4 = "43";  int len2 = str4.length();  System.out.println(len2);  String str5 = "";  System.out.println(str5.length());  String str6 = "Cat says \"MEOW\"!";  System.out.println(str6.length());  String str7 = "\\\"\"\n\\\n";  System.out.println(str7.length()); | **Capture your predictions:** | |
| **CONFIRM!** Now, run in your IDE to confirm that you counted correctly. **Look carefully at #6 & 7!** | | |
| How are the *two* characters that make up an escape sequence (e.g. \") counted in terms of length()? | [*Check*](#_epnwkrjrbaug) | |
| Write some code to create a String object holding the string **good morning**, then use the length method on your String object to obtain the string's length. Store the length in a variable strLen, and then print out strLen. | | |
| **Paste your code here:** [confirm](#_5f334yhfkaqc) | | |

**EXPLORATION CONTINUES ON NEXT PAGE**

| On to the next method! Let’s do indexOf, since it also returns an int  **Review the Explanations & Examples:**    **Example 1:**  String myString = "AP CSA is awesome!";  int index = myString.indexOf("**is**");  System.out.println(index);  **Prints: 7**    **Example 2:**  String myString = "AP CSA is awesome!";  int index = myString.indexOf("**A is a**");  System.out.println(index);  **Prints: 5**    **Example 3:**  String myString = "AP CSA is awesome!";  int index = myString.indexOf("**A**");  System.out.println(index);  **Prints: 0** *Note the explanation of this method:*    **Example 4:**  String myString = "AP CSA is awesome!";  int index = myString.indexOf("**a**");  System.out.println(index);  **Prints: 10**    **Example 5:**  String myString = "AP CSA is awesome!";  int index = myString.indexOf("**B**");  System.out.println(index);  **Prints: -1** *Note the explanation of this method:* |
| --- |

**EXPLORATION CONTINUES ON NEXT PAGE**

| **AP EXAM Pro Tip!**  For the problem below, **grab a piece of paper and pencil (seriously!)** and write the String, and *number each character below*, like this:    **You will have to do this all by hand on the AP Exam -- may as well practice now :)** | | |
| --- | --- | --- |
| **1. TAKE OUT A SCRAP PIECE OF PAPER!**  **Write the following string down:**  Hello how are you today?  **Then write the index of each character underneath, similar to this example for the string**  AP CSA is awesome! | | |
| **2. Now determine the output of the following code** using your scrap paper and your brains only:  String str1 = "Hello how are you today?";  System.out.println(str1.length());  System.out.println(str1.indexOf("H"));  System.out.println(str1.indexOf("h"));  System.out.println(str1.indexOf("l"));  System.out.println(str1.indexOf("o"));  System.out.println(str1.indexOf("?"));  System.out.println(str1.indexOf("ll"));  System.out.println(str1.indexOf("ello"));  System.out.println(str1.indexOf("lo ho"));  System.out.println(str1.indexOf("a"));  System.out.println(str1.indexOf("W"));  System.out.println(str1.indexOf(" "));  System.out.println(str1.indexOf("how are"));  System.out.println(str1.indexOf("howare"));  System.out.println(str1.indexOf("Are"));  System.out.println(str1.indexOf("Hello how are you today?"));  System.out.println(str1.indexOf(""));  int x = str1.indexOf("e") + str1.indexOf("E");  System.out.println(x);  String str2 = "how are you?";  System.out.println(str1.indexOf(str2)); | **Jot down the printed output that you expect:**  **CONFIRM your answers by copying/pasting/running in your IDE. Any surprises?**  **What do *you* think is up with this one: "" (Mr. Miller isn't so sure himself):** [*Check answers*](#_i4btzkv9mgwx) | |
| Write some code to find and print the index of “**you**” in the string below:  String str1 = "Hello how are you today?";  Then write code to find and print the index of “**You**” (capital Y).  Run it and make sure the output makes sense!  **Paste your code below:** | | |
| [confirm](#_4a9doo9bd45i) | | |
| Are “you” and “You” the same in terms of indexOf? |  | |

**EXPLORATION CONTINUES ON NEXT PAGE**

| On to the next method, the substring(int from, int to) method, which returns a String.  Note that this is the first of two **overloaded** substring methods (same name, different signature).  **Review the Explanation & Examples:**    **Example 1:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0, 4**);  System.out.println(subString);  **Prints:** "**AP C**"    **substring(0, 4) makes a *new* String made up of the characters at index 0, 1, 2, 3 (but NOT 4)** | |
| --- | --- |
| **Using the same string as in the example:**  String myString = "AP CSA is awesome!";  **Write some code to obtain the substring containing characters "awe" from** myString **and store it in a new String object named** aweStr**; print out** aweStr **to make sure it is correct.** | |
| 1 [check](#_lmzyicoex9za) | |
| **So what happens with the original myString? Does using the substring method affect it?**  Find out! Copy/paste/run the following in your IDE:  String myString = "AP CSA is awesome!";  String subString = myString.substring(0, 4);  System.out.println(subString);  System.out.println(myString); | Did performing myString.substring() *modify* myString?  ***Read on to confirm!*** |
| **AS YOU HOPEFULLY NOTICED:** The substring method returns a new String object containing the substring; **the *original* String (myString) is *not* affected in any way**!  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0, 4**);  System.out.println(subString);  System.out.println(myString);  **Prints:** "**AP CS**"  "**AP CSA is awesome!**" // myString hasn’t changed!   | In fact, Strings are **immutable** objects, which means all Strings methods invoked on a particular String do **not** change the original String object itself. | | --- | | |
| **Example 2:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**7, 9**);  System.out.println(subString);  **Prints:** "**is**"    **Example 3:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**6, 10**);  System.out.println(subString);  **Prints:** " **is** " **(note the spaces!)**    **Example 4:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**12, 13**);  System.out.println(subString);  **Prints:** "**e**"    **Calling substring(12, 13) makes a new String that includes the character at index 12 ONLY!**  **This is how you obtain a *single* character at a specific index!**   | A string identical to the *single character substring* at position index can be created by calling substring(index, index + 1). | | --- |   **Example 5:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0, 17**);  System.out.println(subString);  **Prints:** "**AP CSA is awesome**" (no "!" included)    **Example 6:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0, 18**);  System.out.println(subString);  **Prints:** "**AP CSA is awesome!**" (prints the entire string)    **Example 7:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0,** **19**);  System.out.println(subString);    **This happens: Kaboom!**    **Explanation:**   | A String object has index values from 0 to length – 1. **Attempting to access indices outside this range will result in a runtime error called IndexOutOfBoundsException!** | | --- |   **Comment:**  If you notice in Example 7, using 19 as the to argument tells Java to access index (19 - 1), or 18, but 18 is ***outside*** the index range of myString, which only goes to 17.  In Example 6, using 18 was OK though, since 18 - 1 = 17, and 17 is the *last* valid index of the string. | |
| **Using the same string as in the example:**  String myString = "AP CSA is awesome!";  **Write some code to obtain the substring containing the *single* character "S" from** myString **and store it in a new String object named** s**; print out** s **to make sure it is correct.**  **Write some more code to obtain the substring containing "awesome!" (including !) from** myString **and store it in a new String object named** awe**; print out** awe **to make sure it is correct.** | |
| 2 [check](#_jdociwxmlx0) | |

| **Refer to your scrap paper from before, where you should have written down the index values of each character in the string** "Hello how are you today?"  **Determine** the output of this code using your scrap paper and brains:  String origStr = "Hello how are you today?";  String str1 = origStr.substring(0, 13);  System.out.println(str1);  String str2 = origStr.substring(8, 11);  System.out.println(str2);  String str3 = origStr.substring(14, 15);  System.out.println(str3);  String str4 = origStr.substring(0, 1);  System.out.println(str4);  String str5 = origStr.substring(9, 10);  System.out.println(str5);  String str6 = origStr.substring(18, 23);  System.out.println(str6);  String str7 = origStr.substring(20, 24);  System.out.println(str7);  String str8 = origStr.substring(10, 26);  System.out.println(str8); | **Jot down the expected output:**  **CONFIRM your answers by copying/pasting/running in your IDE. Note what IntelliJ tells you about the last one in the compiler!** [*Check*](#_oh8vuwkvdb3p) | |
| --- | --- | --- |
| Using this string:  String origStr = "Hello how are you today?";  Write some code that will retrieve the substring “**how**” from origStr above and store it in a new variable, str9. Then print out str9 to make sure! | | |
| 3 [confirm](#_yoebmnoitfr3) | | |
| **DEBUG! A student tried the previous problem and wrote 10 instead of 9 for the second parameter:**  String origStr = "Hello how are you today?";  String str9 = origStr.substring**(6, 10**);  System.out.println(str9);  And got this:    So he assumed he was correct. What’s going on? Was he correct? | **Explain!** [check](#_246mclxuym2h) | |
| Here’s a string:  String blah = "What's for dinner?";  Write a line of code that will retrieve the substring dinner from blah and store it in a new variable, ugh. Print ugh to make sure that “dinner” is stored in ugh. | | |
| [*Check*](#_wvkg1b2hftu7) *4* | | |

| **Let’s kick it up a notch!**  What you see below is an example of **method chaining**, a fairly common practice in programming when one method call returns an object that you immediately want to perform another method on. It’s cool and handy! | | |
| --- | --- | --- |
| What will this print?  String yum = "What’s for dinner?";  String din = yum.substring(3, 12).substring(5, 7);  System.out.println(din); | | **Prediction:** *Copy/paste/run**to confirm!* |

##### *(*[*click here for explanation on what’s happening*](#_ul01wf8stjpw)*)*

| **Try this one!** Again, note the method chaining; go left to right, and reindex each new String | | |
| --- | --- | --- |
| What will this print?  String today = "FRIDAY!";  String x = today.substring(3, 7).substring(0, 3).substring(1, 3);  System.out.println(x); | | **Prediction:** *Copy/paste/run to confirm!* |

| On to the next method, the substring(int from) method, which also returns a String.    **Example 1:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**3**);  System.out.println(subString);  **Prints:** "**CSA is awesome!**"    **substring(3) makes a new String made up of the characters starting at index 3, and going all the way to the end. It’s *equivalent* to substring(3, myString.length()) which is myString.substring(3, 18)**  **Example 2:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**10**);  System.out.println(subString);  **Prints:** "**awesome!**"    **Example 3:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**16**);  System.out.println(subString);  **Prints:** "**e!**"    **Example 4:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**0**);  System.out.println(subString);  **Prints:** "**AP CSA is awesome!**"    **Example 5:**  String myString = "AP CSA is awesome!";  String subString = myString.substring(**17**);  System.out.println(subString);  **Prints:** "**!**" | |
| --- | --- |

| Here’s a string:  String blah2 = "What's for dinner?";  Write a line of code using the ***single parameter* substring method** that will retrieve the substring dinner? (including ?) from blah2 and store it in a new variable, ugh2. Print ugh2 to make sure that “dinner?” is stored in ugh2.  Then write a line of code using the ***two-parameter* substring method** that will retrieve the same substring dinner? (including ?) from blah2 and store it in a new variable, ugh3. Print ugh3 to make sure that “dinner?” is stored in ugh3. | | |
| --- | --- | --- |
| 4 [confirm](#_gfcdvnetglp8) | | |
| **Without using IntelliJ, what will the following print?**  *(Recall the AP Exam pro tip --* ***write all strings down*** *and label index numbers!)*  String tired = "I'm tired!";  String end = "Of substrings!";  String str0 = end.substring(3);  System.out.println(str0);  String str1 = tired.substring(4) + end.substring(3);  System.out.println(str1);  String str2 = tired.substring(9) + tired.substring(6);  System.out.println(str2);  String str3 = end.substring(0, 1) + " " + end.substring(1, 2);  System.out.println(str3);  String str4 = end.substring(6).substring(2);  System.out.println(str4);  **// Tricky!**  String fire = "CRACKLE!";  String witch = "CACKLE!";  String str5 = fire.substring(2);  System.out.println(str5);  String str6 = witch.substring(witch.length() - 4);  System.out.println(str6);  int loc = str5.indexOf(str6);  System.out.println(loc);  String str7 = fire.substring(loc, loc + 3);  System.out.println("the hidden word is: " + str7); | | **Expected output:**  **CONFIRM your answers by copying/pasting/ running in your IDE.** [*Check*](#_rmz4do8j57zx) |
| **Free Style!** Write some code to test out the substring method on your own. Come up with your own strings! Try out **BOTH** versions of substring: the one with **TWO** parameters and the one with just **ONE**. Make sure you see the difference!  **Copy/paste your free style code below:** | | |
|  | | |

| **Practice AP Question!** | | |
| --- | --- | --- |
| **Your Answer:** |  | |

##### [*Check your answer!*](#_60be2s6k78yt)

**EXPLORATION CONTINUES ON NEXT PAGE**

| On to the next method, the boolean equals(String other) method, which returns a boolean     | **Careful!**  equals returns true if the two Strings represent the *same sequence of characters* (e.g. “cat” and “cat”). This is ***not*** a test to see if two variables *reference* the same String *object*, however; that’s what == is for.  When you want to see if two Strings have the same characters (i.e. their character strings are equal), you **MUST USE** **equals** -- do ***not*** use **==** for this!  **equals** is for ***content*** comparison (i.e. same sequence of letters)  **==** is for ***reference*** comparison (i.e. two variables reference the same object in memory) | | --- | | |
| --- | --- | --- |

| **Determine** the output of this code without an IDE.  String str1 = "Hello!";  String str2 = "Hello!";  System.out.println(str1.equals(str2));  String str3 = "hello!";  System.out.println(str1.equals(str3));  String str4 = "HELLO!";  System.out.println(str1.equals(str4));  String str5 = "Hello";  System.out.println(str1.equals(str5));  String str6 = "Hello! ";  System.out.println(str1.equals(str6)); | **Jot down what you expect the output to be:**  **CONFIRM your answers by copying/pasting/running in your IDE.** | |
| --- | --- | --- |
| **What did you discover about *case sensitivity* and equals?** | [*Check!*](#_qqzf5hp5pg12) | |

| **Just to see why…**  **🤯 Be forewarned: what you might experience below may be wildly unintuitive and the explanation that follows might be confusing -- that’s ok, the thing you need to take away from this is you should NOT EVER** **use “==” to see if two strings *have the same character sequence*.** | | |
| --- | --- | --- |
| **8.** Let’s explore why you should **NOT EVER** use **==** to see if two strings have the same character sequence:  String str1 = "Hello!";  String str2 = "Hello!";  String str3 = new String("Hello!");  String str4 = new String("Hello!");  System.out.println(str1 == str2);  System.out.println(str1 == str3);  System.out.println(str1 == str4);  System.out.println(str3 == str4); | **Capture your predictions:**  **CONFIRM your answers by copying/pasting/ running in your IDE.** | |

| **MORAL OF THE STORY:** When you want to see if two Strings have the same characters (i.e. their character strings are equal), you **MUST USE** **equals**. Do ***NOT*** use **==** for this! |
| --- |

##### *optional read:* [*Wait what? I demand an explanation!*](#_zdjbg8vt64yo) *(we will discuss this more later)*

| **Free Style!** Write some code to test out the equals method on your own. Come up with your own strings! Try out equals vs. == with strings. Make sure you see why **NOT** to use == for comparing strings!  **Copy/paste your free style code below:** | | |
| --- | --- | --- |
|  | | |

##### 

##### **EXPLORATION CONTINUES ON NEXT PAGE**

##### 

| On to the **final** method (finally!), the int compareTo(String other) method, which returns an int    **The method compareTo compares two strings *alphabetically*, character by character!**  If the strings are ***equal*** (i.e. if using equals on them returns true), compareTo returns **0**.  If the first String (the one you are performing the method on) is alphabetically ordered **before** the second String (the argument of compareTo), it returns a **negative** number.  If the first String is alphabetically ordered **after** the second String, it returns a **positive** number.  **Here’s a fun little graphic:**    **Heads up!** You will see some strange positive and negative numbers; the actual value that compareTo returns **does not matter** -- all you care about is: *positive*, *negative*, or 0 (equal)?  (But if you’re curious, it’s the distance in the first letter that is different, e.g. A is 7 letters away from H.) | |
| --- | --- |
| **Example:**  String str1 = "maybe";  String str2 = "apple";  String str3 = "zebra";  String str4 = "maybe";  String str5 = "Maybe";  System.out.println(str1.compareTo(str2));  System.out.println(str2.compareTo(str1));  System.out.println(str1.compareTo(str4));  System.out.println(str1.equals(str4));  System.out.println(str1.compareTo(str3));  System.out.println(str3.compareTo(str1));  System.out.println(str2.compareTo(str3));  System.out.println(str3.compareTo(str2));  **Weirdness alert!**  System.out.println(str1.compareTo(str5)); | **Prints:**  12: **positive** because str1 ("maybe") comes **AFTER** str2 ("apple") alphabetically.  -12: **negative** because str2 ("apple") comes **BEFORE** str1 ("maybe") alphabetically.  0: **zero** because str1 ("maybe") and str4 ("maybe") are equal in terms of their character sequence  (i.e. str1.equals(str2) is *true*)  true: because the two strings match exactly! Note that equals will be true for two strings whenever compareTo is 0.  -13: **negative** because str1 ("maybe") comes **BEFORE** str3 ("zebra") alphabetically.  13: **positive** because str3 ("zebra") comes **AFTER** str1 ("maybe") alphabetically.  -25: **negative** because str2 ("apple") comes **BEFORE** str3 ("zebra") alphabetically.  25: **positive** because str3 ("zebra") comes **AFTER** str2 ("apple") alphabetically.  32: **positive** because str1 ("maybe") with a lowercase **m** is decidedly **AFTER** str5 ("Maybe") with a capital M. When comparing strings, *capitals* come *before* *lowercase*. So just like equals, compareTo ***IS CASE SENSITIVE****!* So be careful :) |
| **Again, we don’t care about the actual value itself (-25, 13, -12, whatever), we just care if it’s positive, negative, or 0.** However**,** you might notice that there *is* an alphabetical relationship between the values and the words (e.g. 25 is the distance between “a” and “z” in the alphabet) -- this is cool to know and note, but not generally necessary for using the compareTo method. | |

| **Determine** whether each statement will print a **POSITIVE** number, **NEGATIVE** number, or **0** (you don’t have to figure out exactly what the number will be, just if it’s positive, negative, or 0):  String str1 = "Hello John!";  String str2 = "My name is Jack.";  String str3 = "Hello";  String str4 = "Hello Jack";  String str5 = "My name is jack.";  String str6 = "Hello";  System.out.println(str1.compareTo(str2));  System.out.println(str2.compareTo(str1));  System.out.println(str1.compareTo(str4));  System.out.println(str1.compareTo(str3));  System.out.println(str3.compareTo(str6));  System.out.println(str2.compareTo(str5));  System.out.println(str2.equals(str5)); | **Capture whether you expect each comparison to be a POSITIVE number, a NEGATIVE number, or 0 (don't worry about predicting the exact numeric value if positive or negative):**  **Now, run in IntelliJ to confirm!** [*Check with explanations*](#_ugh5xdyw25u4) *9* | |
| --- | --- | --- |

| **Free Style!** Write some code to test out the compareTo method on your own. Come up with your own strings! Make sure to try reversing the order of the strings.  **Copy/paste your free style code below:** | | |
| --- | --- | --- |
|  | | |

**Exploration continues on the next page**

| **String Info!** |
| --- |
| Clear out all the test code so far from your main method.  Then, write a short program that asks the user to enter a string. Print out how long their string is, as well as show them the "first half" and the "second half" of the word. For example, if the user enters "apples", the length is 6, the first half is "app" and the second half is "les". If the user enters an odd-length word, like "apple," include the extra letter in the *second* half: "ap" and "ple"  Then ask them to enter a *second* string. After they enter their second string, tell the user which of the strings entered was longer, or if they have the same length. Then, tell them if the two strings are the same sequence of characters (i.e. if they are equal or not), and if not, which comes first alphabetically.  Lastly, inform the user at what index the second string is found inside the first string, or "not found" if not found. For example, if the user enters "bananas'' for the first string and "nana" for the second string, you should print index 2. If the user enters "apples" for the first string and "nana" for the second string, you should print "not found".  **Your program's code should include the use of each of the following string methods *at least once each*:**  **X**   |  | length | | --- | --- | |  | indexOf | |  | substring (*two*-parameter method) | |  | substring (*one*-parameter version) | |  | equals | |  | compareTo |   **Test case 1: Test case 2:**    **Test case 3: Test case 4:** |
| **Copy/paste your entire program below:** |
|  |
| **Insert a screenshot showing the output from *another* test case that you came up with on your own:** |
|  |

###### Want to compare solutions?

###### [Here is how Mr. Miller did it](#_y96x0amew0mo)

**Done!**

Submit in Google Classroom:



### Answer ([back](#_129vrdk4xo8m))

| **Predict** (by counting them up manually!) the output of this code. Note the escape sequences in 6 & 7.  String str1 = "Hello!"; System.out.println(str1.length());  String str2 = " H e l l o ! ";  int len = str2.length();  System.out.println(len);  String str3 = "Is tax 8.5%?";  System.out.println(str3.length());  String str4 = "43";  int len2 = str4.length();  System.out.println(len2);  String str5 = "";  System.out.println(str5.length());  String str6 = "Cat says \"MEOW\"!";  System.out.println(str6.length());  String str7 = "\\\"\"\n\\\n";  System.out.println(str7.length()); | 6 (punctuation marks like ! count)  13 (spaces in between *and on the ends* count)  12 (every letter, number, *or* symbol counts!)  2 (numbers are characters, too!)  0 (the "empty string" has no length)  **16** (see note below)  **6** (see note below; this String is made of 6 escape sequences, note that a new line counts as a character!) | |
| --- | --- | --- |
| How are the *two* characters that make up an escape sequence (e.g. \") counted in terms of length()? | Each *pair* of escape sequence characters  (\\, \", \n) is counted as 1 towards the String’s length | |

### Answer ([back](#_xy7429pkvtsh))

| **1. Your scrap paper should look like this:**   | **H** | **e** | **l** | **l** | **o** |  | **h** | **o** | **w** |  | **a** | **r** | **e** |  | **y** | **o** | **u** |  | **t** | **o** | **d** | **a** | **y** | **?** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2. Determine** the output of this code.  String str1 = "Hello how are you today?";  System.out.println(str1.length());  System.out.println(str1.indexOf("H"));  System.out.println(str1.indexOf("h"));  System.out.println(str1.indexOf("l"));  System.out.println(str1.indexOf("o"));  System.out.println(str1.indexOf("?"));  System.out.println(str1.indexOf("ll"));  System.out.println(str1.indexOf("ello"));  System.out.println(str1.indexOf("lo ho"));  System.out.println(str1.indexOf("a"));  System.out.println(str1.indexOf("W"));  System.out.println(str1.indexOf(" "));  System.out.println(str1.indexOf("how are"));  System.out.println(str1.indexOf("howare"));  System.out.println(str1.indexOf("Are"));  System.out.println(str1.indexOf("Hello how are you today?"));  System.out.println(str1.indexOf(""));  int x = str1.indexOf("e") + str1.indexOf("E");  System.out.println(x);  String str2 = "how are you?";  System.out.println(str1.indexOf(str2)); | 24  0 (first character)  6 (h not the same as H)  2 (finds first instance)  4  23 (last character)  2  1  3  10 (finds first instance)  -1 (not in string)  5 (finds first instance)  6  -1 (not exact!)  -1 (case matters!)  0 (the string is found  inside of itself!)  0 (um yeah, Mr. Miller  isn't too sure either)  0 (indexOf("e") returns  1 and indexOf("E")  returns -1,  and 1 + -1 = 0!)  -1 (careful about the ? in str2, it doesn’t match where it is in str1. If there wasn’t a ? in str2, it would have printed 6) | |

### WHAT ([back](#_2nlszwlmu1jz))

****

== is for ***reference*** comparison (i.e. two variables reference the same object in memory), and so two objects are considered "==" **only if** they refer to the *SAME STRING OBJECT --* ***not*** *whether those string objects happen to have the same character sequence.* We will discuss this more in detail later.

| **8.**  String str1 = "Hello!";  String str2 = "Hello!";  String str3 = new String("Hello!");  String str4 = new String("Hello!");  System.out.println(str1 == str2);  System.out.println(str1 == str3);  System.out.println(str1 == str4);  System.out.println(str3 == str4); | **true -** this is actually **very surprising!** If == means that two variables refer to the *same* object, shouldn’t (str1 == str2) be *FALSE* since str1 and str2 are clearly initialized to two different string literals? You would think, however, in Java, with STRING LITERALS, it actually caches them in such a way that makes == *true* for two string objects that are MADE FROM THE SAME EXACT STRING LITERAL: “Hello!”. **CRAZY and confusing and it’s why we don’t ever mess with == for content comparison!**  **false** - For the reason you might suspect; str1 and str3 refer to two different String objects, they are not ==. Note that because str3 is made using the String constructor *rather than* a String literal, Java does NOT apply the weird rule described above.  **false** - For the same reason as the previous false.  **false** - For the same reason, although this one might be the most obvious to see; clearly str3 and str4 are two different String objects, so they are not ==. | |
| --- | --- | --- |

**Great, so this is all very confounding.**

**What’s the moral of the story? This:**

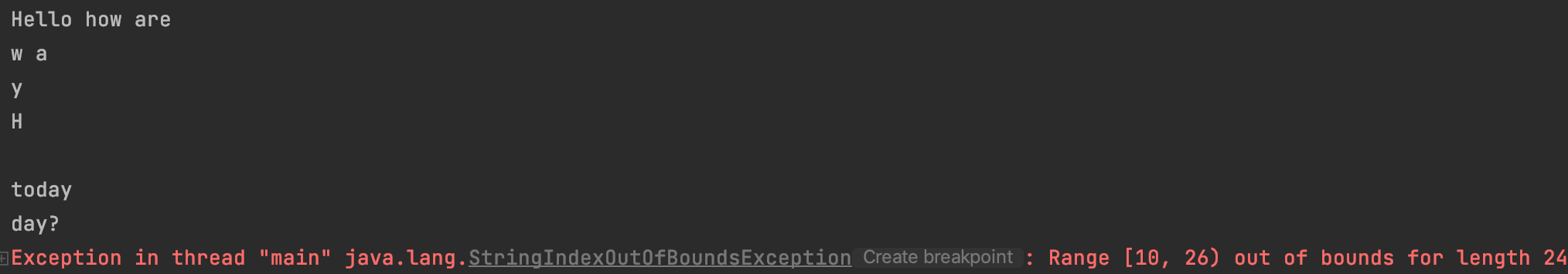
| When you want to see if two Strings have the same characters (i.e. their character strings are equal), you **MUST USE** **equals**. Do ***NOT*** use **==** for this! |
| --- |

### Solution ([back](#_rluik3ed35ad))

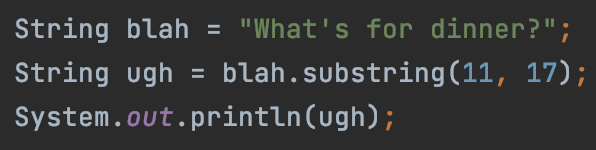
| **H** | **e** | **l** | **l** | **o** |  | **h** | **o** | **w** |  | **a** | **r** | **e** |  | **y** | **o** | **u** |  | **t** | **o** | **d** | **a** | **y** | **?** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |

| **3. Determine** the output of this code  String origStr = "Hello how are you?";  String str1 = origStr.substring(0, 13);  System.out.println(str1);  String str2 = origStr.substring(8, 11);  System.out.println(str2);  String str3 = origStr.substring(14, 15);  System.out.println(str3);  String str4 = origStr.substring(0, 1);  System.out.println(str4);  String str5 = origStr.substring(9, 10);  System.out.println(str5);  String str6 = origStr.substring(18, 23);  System.out.println(str6);  String str7 = origStr.substring(20, 24);  System.out.println(str7);  String str8 = origStr.substring(10, 26);  System.out.println(str8); | Hello how are  w a  y  H  ← single space  today  day?  **IndexOutofBoundsError!**  (26 - 1 = 25, which exceeds the last index of 23) | |
| --- | --- | --- |

If you run the code, you should see that the "outOfBoundsException" is a **runtime** error because the program started to execute (note that other values are printed out before the crash occurs):



### Answer ([back](#_ra25lkddfd1d))



****

### Answer ([back](#_jb1tj0fbjtbk))

**This prints: "or"**

**EXPLANATION of Method Chaining**

| What will this print?  String yum = "What’s for dinner?";  String din = **yum.substring(3, 12).substring(5, 7)**;  System.out.println(din); | | |
| --- | --- | --- |

| **W** | **h** | **a** | **t** | ’ | **s** |  | **f** | **o** | **r** |  | **d** | **i** | **n** | **n** | **e** | **r** | **?** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

In the code segment above, the *first* method, **substring(3, 12)**, is called on **yum**, which returns the new String object: "**t’s for d**"

This *new* String,"**t’s for d**"**,** becomes the object on which the *second* method (the “chained” method), **substring(5, 7)**, is called; **substring(5, 7)** is ***not*** called on **yum**, but rather the String *result* of **yum.substring(3, 12)**, which is "**t’s for d**".

So you can think of it working like this:

String din = "**t’s for d**"**.substring(5, 7)**;

And "**t’s for d**" is an entirely new String, so we need to reindex it:

| **t** | ’ | **s** |  | **f** | **o** | **r** |  | **d** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

And we can see that calling **substring(5, 7)** on this string will produce the output you see: **or**

##### 

### Answers ([back](#_96vvlhihzza)) Your handwritten string objects should be indexed like this:

**tired:**

| **I** | ' | **m** |  | **t** | **i** | **r** | **e** | **d** | **!** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**end:**

| **O** | **f** |  | **s** | **u** | **b** | **s** | **t** | **r** | **i** | **n** | **g** | **s** | **!** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

**fire:**

| **C** | **R** | **A** | **C** | **K** | **L** | **E** | **!** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

**witch:**

| **C** | **A** | **C** | **K** | **L** | **E** | **!** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

| String tired = "I'm tired!";  String end = "Of substrings!";  String str0 = end.substring(3);  System.out.println(str0);  String str1 = tired.substring(4) + end.substring(3);  System.out.println(str1);  String str2 = tired.substring(9) + tired.substring(6);  System.out.println(str2);  String str3 = end.substring(0, 1) + " " + end.substring(1, 2);  System.out.println(str3);  String str4 = end.substring(6).substring(2);  System.out.println(str4);  **// TRICKY!**  String fire = "CRACKLE!";  String witch = "CACKLE!";  String str5 = fire.substring(2);  System.out.println(str5);  String str6 = witch.substring(witch.length() - 4);  System.out.println(str6);  int loc = str5.indexOf(str6);  System.out.println(loc);  String str7 = fire.substring(loc, loc + 3);  System.out.println("the hidden word is: " + str7); | | **substrings!**  **tired!substrings!**  **!red!**  **O f**  **rings!**  **ACKLE!**  **KLE!**  **2**  **the hidden word is: ACK** |
| --- | --- | --- |

### AP Answer ([back](#_7e215kuib3pf))

**Answer: A**

If you missed this one, try it with any two strings!

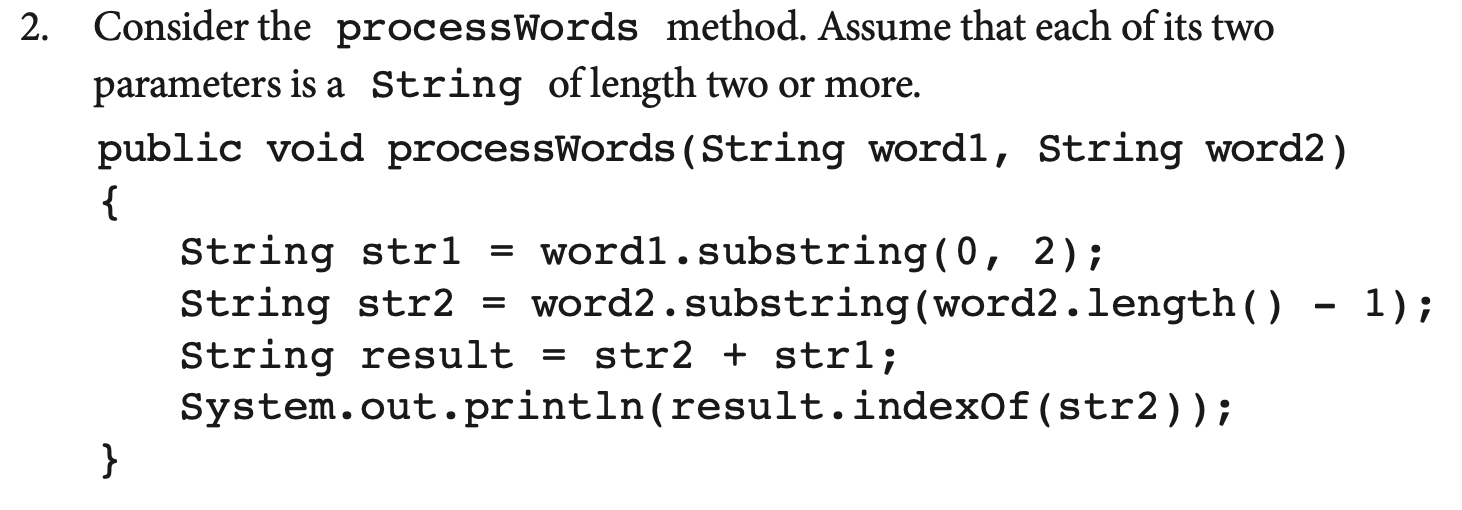
Assume word1 is, say, "apple", and word2 is, say, "banana"

word1: word2:

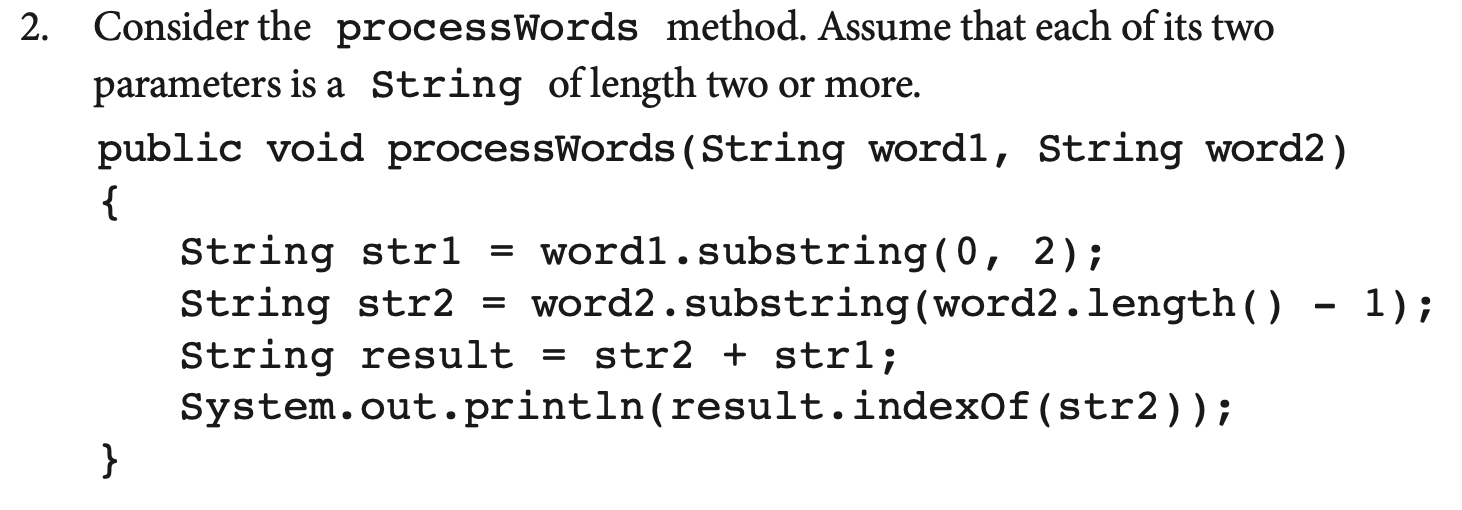
a p p l e b a n a n a

0 1 2 3 4 0 1 2 3 4 5

This line of code would store "**ap**" in str1:

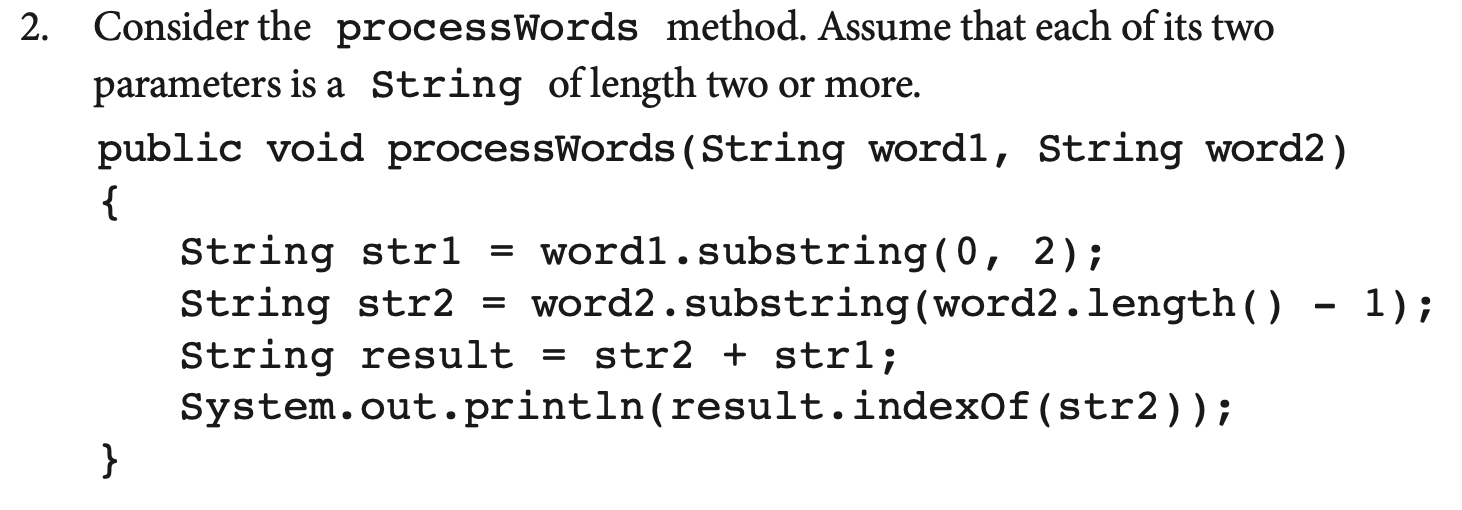


This line of code would store "**a**" in word2:

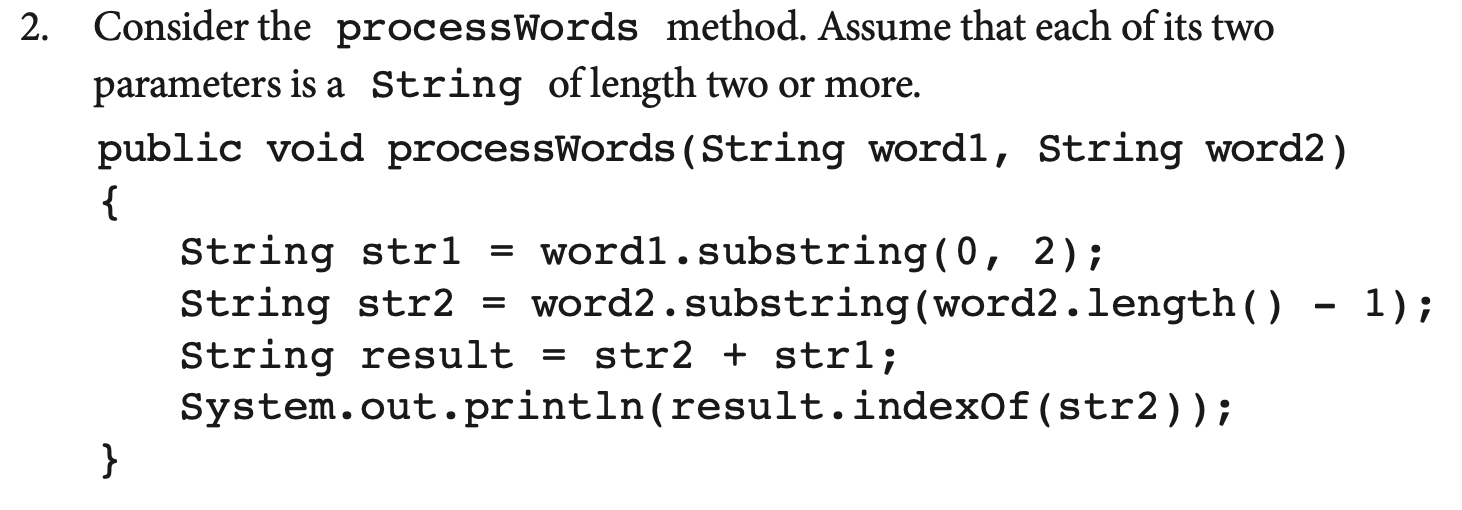


Note that word2.substring(word2.length() - 1) would be word2.substring(5), since word2 has a length of 6, which makes a substring containing just the last letter ("a")

This line of code would concatenate them, str2 ("a") followed by str1 ("ap"), storing "**aap**" in result:



This line of code would print where str2 occurs in result; in other words, where "a" appears in "aap" -- this occurs at index 0 (also index 1, but first occurrence returned):



**So this prints 0 *-- no matter what strings you choose for word1 and word2!***

### Answer ([back](#_585m6anvqjq1))

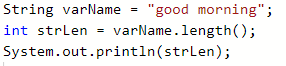
**equals** is for ***content*** comparison (i.e. same sequence of letters) -- **they MUST BE EXACTLY the same!**

| **7. Predict** the output of this code without Coding Rooms.  String str1 = "Hello!";  String str2 = "Hello!";  System.out.println(str1.equals(str2));  String str3 = "hello!";  System.out.println(str1.equals(str3));  String str4 = "HELLO!";  System.out.println(str1.equals(str4));  String str5 = "Hello";  System.out.println(str1.equals(str5));  String str6 = "Hello! ";  System.out.println(str1.equals(str6)); | **true -** the two strings have the *same exact* sequence of characters  **false -** "Hello!" Is **NOT** equal to "bello!" because the capitalization doesn’t match  **false -** "Hello!" Is **NOT** equal to "HELLO!" because the capitalization doesn’t match  **false -** "Hello!" Is **NOT** equal to "Hello" because of a missing exclamation point  **false -** "Hello!" Is **NOT** equal to "Hello! " because of the extra space | |
| --- | --- | --- |
| **What did you discover about case sensitivity and equals?** | equals **IS** case sensitive!!!  "Hello!" and "hello!" are **NOT** "equal" | |

### Answers ([back](#_obp86csfi57u))

| String str1 = "Hello John!";  String str2 = "My name is Jack.";  String str3 = "Hello";  String str4 = "Hello Jack";  String str5 = "My name is jack.";  String str6 = "Hello";  System.out.println(str1.compareTo(str2));  System.out.println(str2.compareTo(str1));  System.out.println(str1.compareTo(str4));  System.out.println(str1.compareTo(str3));  System.out.println(str3.compareTo(str6));  System.out.println(str2.compareTo(str5));  System.out.println(str2.equals(str5)); | Again, don't worry about the exact numeric values, we just care about if the result is positive, negative, or zero:  -5: **negative** because str1 ("**H**ello John!") comes **BEFORE** str2 ("**M**y name is Jack.") alphabetically.  5: **positive** because str2  ("**M**y name is Jack.") comes **AFTER** str2 ("**H**ello John!") alphabetically.  14: **positive** because str1 ("Hello J**o**hn!") comes **AFTER** str4 ("Hello J**a**ck") alphabetically. The comparison happens between the *first* *different* character: “o” and “a”  6: **positive** because str1 ("Hello **J**ohn!") comes **AFTER** str3 ("Hello") alphabetically.  The comparison is happening between “ ” (space) and *nothing*, and nothing comes before something alphabetically.  0: **zero** because str3 ("Hello") and str6 ("Hello") are equal in terms of their character sequence.  -32: **negative** because str2 ("My name is **J**ack.") comes **BEFORE** str5 ("My name is **j**ack.") alphabetically. The comparison happens between the *first* *different* characters: “J” and “j”, and since *capital* letters come *before* *lowercase* letters alphabetically, J is “before” j  false: because as you learned, equals is case sensitive, and so "My name is **J**ack." and "My name is **j**ack." are **NOT** equal! | |
| --- | --- | --- |

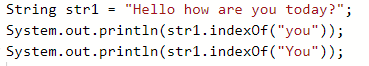
### Solution ([back](#_gwjrhpd8smmt))



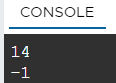
**YOU SHOULD GET:**



### Solution ([back](#_7nqj7wdzm80t))

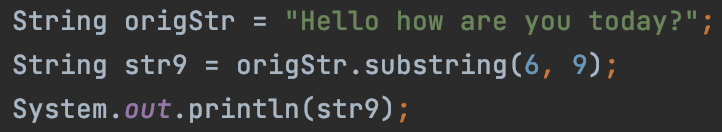
****

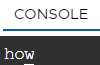
**Output:**

****

**“You” and “you” are NOT the same! “You” is *not* found (-1) but “you” is found (at index 14)**

### Solution ([back](#_7rfst82sc2vm))





### Check ([back](#_lg3seglnvpne))

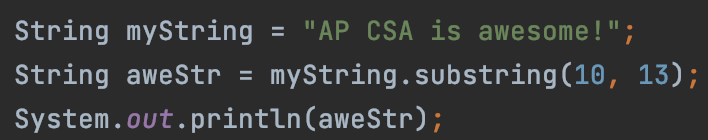
| **DEBUG! A student tried the previous problem and wrote 10 instead of 9 for the second parameter:**  String origStr = "Hello how are you today?";  String str9 = origStr.substring**(6, 10**);  System.out.println(str9);  And got this:    So he assumed he was correct. What’s going on? Was he correct? | **Explain!**  He was **NOT** correct because his answer includes the **space** after “how” in the original string:  how\_  Instead of:  how  ***He just didn’t notice it in the output!*** | |
| --- | --- | --- |

### Confirm ([back](#_e5uvq8rgmxt8))

We want "**awe**" which are the characters at index 10, 11, and 12, so we use the substring method with 10 and 13, since it goes *from* 10 up through *13 - 1,* or 12, which is what we want:



So your code should look like this:

****

### Confirm ([back](#_wn2idzfc9upx))

| **Using the same string as in the example:**  String myString = "AP CSA is awesome!";  **Write some code to obtain the substring containing the *single* character "S" from** myString **and store it in a new String object named** s**; print out** s **to make sure it is correct.**  **Write some more code to obtain the substring containing "awesome!" (including !) from** myString **and store it in a new String object named** awe**; print out** awe **to make sure it is correct.** | |
| --- | --- |
| 2 check | |

### Confirm ([back](#_6obamgxfnbk1))

| Here’s a string:  String blah2 = "What's for dinner?";  Write a line of code using the ***single parameter* substring method** that will retrieve the substring dinner? (including ?) from blah2 and store it in a new variable, ugh2. Print ugh2 to make sure that “dinner?” is stored in ugh2.  Then write a line of code using the ***two-parameter* substring method** that will retrieve the same substring dinner? (including ?) from blah2 and store it in a new variable, ugh3. Print ugh3 to make sure that “dinner?” is stored in ugh3. | | |
| --- | --- | --- |
|  | | |

### Sample solution ([back](#_7h38ucg271vr))

