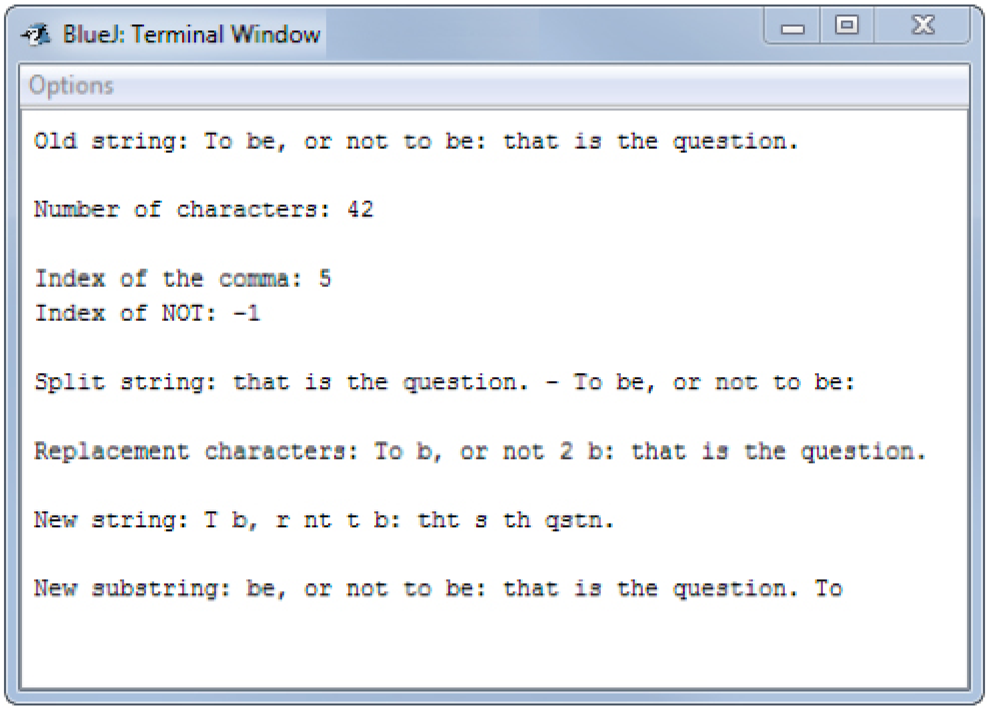
02.06 Desk Check: StringPractice.java

Look over the source code for the StringPractice.java program. You know the drill by now: look for the big picture first, then check the details.

Expected Output

When you compile and run the program, you should see the following output. Try to match up the displayed results with the corresponding segments of code.



```
Old string: To be, or not to be: that is the question.

Number of characters: 42

Index of the comma: 5
Index of NOT: -1

Split string: that is the question. - To be, or not to be:

Replacement characters: To b, or not 2 b: that is the question.

New string: T b, r nt t b: tht s th qstn.

New substring: be, or not to be: that is the question. To
```

Code Analysis

The following detailed analysis of the program will help you conduct a desk check. Be sure you understand the detailed explanation of each line of code and the syntax of each statement. Run the program and look at the output produced by each section of code.

String method: **length()**

The `length()` method will return an integer value representing the number of characters in a string. The method does not require any parameters (i.e. the parentheses are empty).

int	**length()** Returns the length of this string.

Sample code:

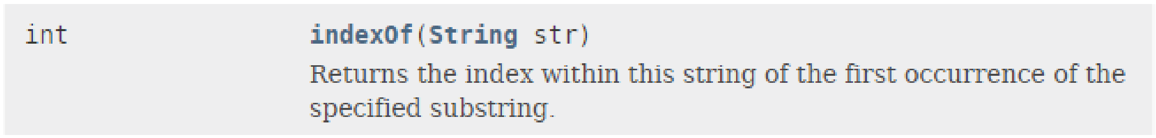
```
String oldString = "To be, or not to be: that is the question.";
System.out.println("Old string: " + oldString);

int stringLength = oldString.length();
System.out.println("Number of characters: " + stringLength);
```

How it works:

- Declares a String object `oldString` and assigns a String literal to it.
- Concatenates a String literal with a String object and prints output to the screen.
- Declares an int variable `stringLength`, invokes the `length()` method on the `oldString` object and assigns the value to the `stringLength` variable.
- Concatenates a String literal with the value indicating the length of the String object.

String method: **indexOf(String str)**

The `indexOf()` method returns an integer value representing the index position in the string where the substring was found. It takes a single String parameter representing the value to be matched. Remember, the index of the first character of a String object is 0. If the string isn't found, a -1 is returned. The method is case sensitive.



int	**indexOf**(**String** str) Returns the index within this string of the first occurrence of the specified substring.

Sample code:

```
int indexOfComma = oldString.indexOf(",");
System.out.println("Index of the comma: " + indexOfComma);
int indexOfNot = oldString.indexOf("NOT");
System.out.println("Index of NOT: " + indexOfNot);
```

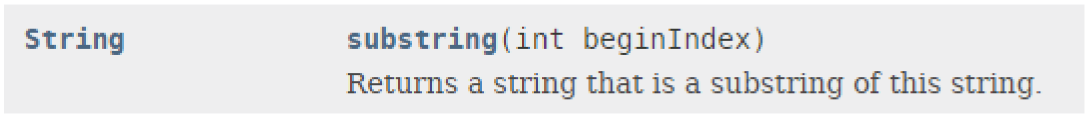
How it works:

- Declares an integer variable `indexOfComma`. Invokes the `indexOf()` method on the `oldString` object to determine the position of the first comma in the `oldString` object and assigns that value to the `indexOfComma` variable.

- Declares an integer variable `indexOfNot`. Invokes the `indexOf()` method on the `oldString` object to determine the position of "NOT" within the oldString object and assigns that value to the `indexOfComma` variable.

String method: **substring(int start)** and **substring(int start, int end)**

The substring() method takes either one or two int parameters. It returns a substring value from the current String object. If one parameter is provided, it indicates the starting index position and will return all characters from that index to the end of the string. The two-parameter version indicates the starting (inclusive) index position and ending (exclusive) index position.



String	substring(int beginIndex) Returns a string that is a substring of this string.

Sample code:

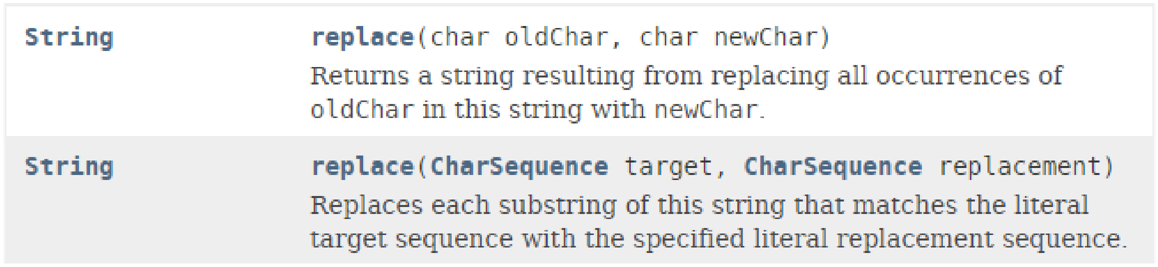
```
int halfwayPoint = stringLength / 2;
String firstHalf = oldString.substring(0, halfwayPoint);
String secondHalf = oldString.substring(halfwayPoint);
String splitString = secondHalf + " - " + firstHalf;
System.out.println("Split string: " + splitString);
```

How it works:

- Declares `halfwayPoint` to be a variable of type int. The `stringLength` is divided by 2 and assigned to `halfwayPoint`.
- Declares `firstHalf` to be a String object. The `substring()` method with two parameters is invoked on the `oldString` object and the first half of the `oldString` object is assigned to the `firstHalf` String reference variable.
- Declares `secondHalf` to be a String object. The `substring()` method with one parameter is invoked on the `oldString` object and the second half of the `oldString` object is assigned to the `secondHalf` String reference variable.
- Declares `splitString` to be a String object. Concatenates the `secondHalf` String object, a hyphen, and the `firstHalf` String object together as a single String literal.
- Concatenates a String literal with the String `splitString` and displays the output to the screen.

String method: **replace(String old, String new)**

The replace() method returns a String with the old characters replaced by the new characters. The method takes two String parameters. The first indicates the old string value to replace. The second is the new string to use as the replacement. This method is case sensitive. All instances of the old string found within the original string will be replaced. If the old string is not found, no changes occur.



String	**replace**(char oldChar, char newChar) Returns a string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**replace**(**CharSequence** target, **CharSequence** replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.

Sample code:

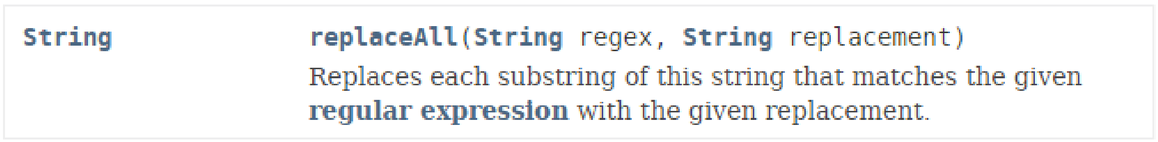
```
String replaceChars = oldString.replace("to", "2");
replaceChars = replaceChars.replace("be", "b");
System.out.println("Replacement characters: " + replaceChars);
```

How it works:

- Declares `replaceChars` to be a String object. The replace() method is invoked on the `oldString` object to replace the word "to" with the number "2". The modified String object is assigned to `replaceChars`.
- The `replace() method` is invoked on the `replaceChars` String object to replace the word "be" with the letter "b".
- Concatenates a String literal with the String object `replaceChars` and prints the modified output to the screen.

String method: **replaceAll(String chars, String new)**

The `replaceAll() method returns` a new String with the replacements. The method takes two String parameters and replaces each character listed with the first parameter with new replacement string literal.



String	**replaceAll**(**String** regex, **String** replacement) Replaces each substring of this string that matches the given **regular expression** with the given replacement.

Sample code:

```
String newString1 = oldString.replaceAll("[aeiou]", "");
System.out.println("New string: " + newString1);
```

How it works:

- Declares `newString1` to be String object. The `replaceAll()` method is invoked on the oldString object and all vowels are removed by replacing them with nothing (an empty pair of double quotation marks).
- Concatenates a String literal with the String object `newString1` and prints the output to the screen.

Using Multiple Methods

It is common to use a combination of methods to achieve a particular goal. Sometimes you need to manipulate a string literal by pulling it apart. The following code shows how to use the `indexOf()` and `substring()` methods to move the first word in a string literal to the end.

Sample code:

```
int positionOfSpace = oldString.indexOf(' ');
String substring1 = oldString.substring(0, positionOfSpace);
String substring2 = oldString.substring(positionOfSpace + 1);
String newString2 = substring2 + " " + substring1;
System.out.println("New substring: " + newString2);
```

How it works:

- Declares `positionOfSpace` to be a variable of type int. Invokes the `indexOf()` method on the `oldString` object to determine the position of the first blank space in the oldString object and assigns that value to the `positionOfSpace` variable.
- Declares `substring1` to be a String object. Invokes the substring() method on the `oldString` object to extract the first word (beginning up to space but not including the space) from the string of characters. Assigns these characters to the `substring1` object.

- Declares `substring2` to be a String object. Invokes the `substring()` method on the `oldString` object to extract the portion of the string from the character after the first blank space to the end of the string of characters. Assigns these characters to the `substring2` object.
- Declares `newString2` to be a String object. Concatenates `substring2`, a blank space, and `substring1` together and prints and assigns the revised string of characters to the `newString2` object.
- Concatenates a String literal with the `newString2` object and prints the output to the screen.

Modifications

As usual, the best way to extend your knowledge of programming is to play with code by making modifications and observing the results. However, be sure that when you make a modification you understand the reasons for any changes you observe. Some changes may not make sense in the context of the program, but they will still teach you something important.

- Change the String literal assigned to the `oldString` object.
- Change the parameters of each method to correspond with the new String object.
- Here's a challenge for you. Using only the methods covered in this Desk Check, how could you determine the number of vowels in a given String literal? Hint: can you find the length of the string with vowels and without? If you have those two values, how can you find the number of vowels?

Print