Intro to objects tutorial

In this tutorial, you'll practice creating strings and calling methods on the `String` class. By the end of this tutorial, you'll have written code that:

- Uses the `new` operator to create objects of the `String` class.
- Calls some common string methods to interrogate and manipulate strings.
- Properly compares two string values.

Getting started

To get started, follow these steps:

1. Import the project into IntelliJ IDEA.
2. Once the project is open, navigate to the project folder `src/main/java/com/techelevator`.
3. Open the `Tutorial.java` file by double-clicking on the filename.

You'll see the starter code for this project. All of your work will be in the `main` method of this class.

Step One: Create strings

In the first part of this tutorial, you'll create new strings in several ways. `String` is a *reference* type, which means that the data that makes up a string is stored in Heap memory, and a reference (the memory address or location) of that data is stored in the `String` variable on the Stack. Whenever you need to create a new instance of a reference type, you use the `new` operator.

From a `char` array

At its heart, a string is an array of characters. As such, you can create a new string from an array of characters with the `new` operator. Under the `Step 1:` comment in your code, type or paste the following:

```
// Create a new string from an array of characters
char[] helloChars = new char[] {'h', 'e', 'l', 'l', 'o', '!'};
String greeting = new String(helloChars);
System.out.println("Greeting: " + greeting);
```

In that code, you used the `new` operator to allocate a chunk of Heap memory large enough to store a string containing the characters `h`, `e`, `l`, `l`, `o`, and `!`. The `new` operator returned the address of the newly created string, and you stored that address into the `greeting` variable.

Run the program, and you'll see `Greeting: hello!` on the console.

From a string literal

Another way to create a new string is to pass a *string literal*, which is enclosed in double-quotes. Under your previous code, type this:

```
// You can also create a string by passing in a literal value, in double-quotes
String salutation = new String("Welcome my friend");
System.out.println("Salutation: " + salutation);
```

When you run the program, your output looks like this:

```
Greeting: hello!
Salutation: Welcome my friend
```

Using the string literal syntax

Since strings are so frequently used, Java allows you to *assign* a string literal to a variable. The Java compiler recognizes this as a request for new memory.

Add the following code:

```
// Java allows you to skip the *new* operator when creating a new String
String toast = "May the compiler rise up to meet you.";
System.out.println("Toast: " + toast);
```

This is by far the most common method of creating a new string in Java. Your output now looks like this:

```
Greeting: hello!
Salutation: Welcome my friend
Toast: May the compiler rise up to meet you.
```

Step Two: Try out some `String` methods

Next, you'll practice using some of the more common methods on the `String` class. These methods either give you information about a string, or use a string to create a new string that has been changed in some way.

First, you'll ask the user to type in a sentence, and then you'll call methods to get variations of that sentence to display to the user.

> In these examples, the user is typing "The quick brown fox jumps over the lazy dog." You can type this or any other sentence at the prompt. Feel free to experiment; change the input sentence and see how the output changes.

Under the `Step 2:` comment, type the following:

```
// Prompt the user to enter a sentence
System.out.print("Please type a sentence: ");
```

```
Scanner scanner = new Scanner(System.in);
String sentence = scanner.nextLine();
```

This code prompts the user to type a message and waits for the user to type something and press **Enter**. Then `sentence` will hold the address of the string value the user typed.

First, echo the value back to the user:

```
// Print the sentence back to the user
System.out.println(sentence);
```

If you run the program, you'll see the message `Please type a sentence:` on the console. The program is waiting for you to type something. Click in the console window, and type a sentence, then press the **Enter** key. You'll see the same sentence printed on the next line.

`toUpperCase()` and `toLowerCase()`

Use these methods to transform the original sentence by either making every character uppercase or lowercase. Then print the results to the user:

```
// Print the sentence in all upper-case
String uppercaseSentence = sentence.toUpperCase();
System.out.println(uppercaseSentence);

// Print the sentence in all lower-case
System.out.println(sentence.toLowerCase());
```

Run the program, and you'll see something like this:

```
Please type a sentence: The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
the quick brown fox jumped over the lazy dog.
```

Find the length of the first word

The `length()` method returns how many characters long the string is.

The `indexOf()` method allows you to find the first occurrence of a smaller string within the string.

You'll use these two methods to report the length of the first word in the sentence to the user. Type this code:

```
// Find the first space character
int firstSpace = sentence.indexOf(" ");
// Report the length of the first word
```

```
if (firstSpace == -1) {
    // IndexOf returns -1 when the string is not found.
    // If there is no space, assume the whole sentence is one word.
    System.out.println("The first word is " + sentence.length() + " characters
long.");
} else {
    // Report the length of the first word
    System.out.println("The first word is " + firstSpace + " characters long.");
}
```

You called `indexOf()` to find the space character. This method returns the index of the first occurrence of that string within `sentence`. If the `" "` string is *not* found inside sentence, `indexOf()` returns -1.

If there is no space character—meaning `indexOf()` returned -1—then you can assume the whole sentence is one word. So you used the `length()` method to report the length of `sentence`.

If `indexOf()` returns a non-negative number, the space was found. You report that value as the length of the first word. For example, if the sentence starts with "The quick", then `indexOf()` returns 3 (the fourth character). Three is the length of the first word.

Output:

```
Please type a sentence: The quick brown fox jumped over the lazy dog.
...
The first word is 3 characters long.
```

or

```
Please type a sentence: Eureka!
...
The first word is 7 characters long.
```

Find and replace inside a string

Now you'll use the `replace()` method to locate a substring inside the string, and replace it with another substring.

Add the following code:

```
// Replace the word "the" with "the one and only"
System.out.println(sentence.replace("the", "the one and only"));
```

This finds *every* occurrence of the string "the", replaces it with the string "the one and only", and returns the resulting string. Remember that the original string isn't modified.

Output:

```
Please type a sentence: The quick brown fox jumped over the lazy dog.
...
The quick brown fox jumped over the one and only lazy dog.
```

Try it again with the initial "The" non-capitalized:

```
Please type a sentence: the quick brown fox jumped over the lazy dog.
...
the one and only quick brown fox jumped over the one and only lazy dog.
```

Did you notice the difference? In the first example, the word *The* wasn't a match for the `replace()` method, which searches for *the*. In most string methods that involved searching or comparing, the case of the letters in the word matter. "T" isn't equal to "t."

Split the sentence into words and reassemble it

Now you'll change the form of `sentence` from a string into an array of strings, each element representing one word. To do that, use the `split()` method:

```
// list the words (split)
String[] words = sentence.split(" ");
System.out.println("The words in this sentence:");
for (int i = 0; i < words.length; i++){
    System.out.println(words[i]);
}
```

In the first line, `split()` looks for the *delimiter*, or *separator*, within the string, and "breaks" the string up every time it finds that delimiter. You passed in a single space (" ") character as the delimiter. The return value from that method is an array of the substrings that fall between the delimiters. The delimiters are stripped away, and don't end up in the result array.

The final three lines use a *for* loop to list the individual words on the console. The output looks like this:

```
Please type a sentence: The quick brown fox jumped over the lazy dog.
...
The words in this sentence:
The
quick
brown
fox
jumped
over
the
```

```
lazy
dog.
```

Now take that array of strings (`String[]`) called `words`, and *join* each string together to form a single string again. Like `split()`, the `join()` method requires a *delimiter* parameter. In this case, the delimiter is *inserted* between each string element as the new string is formed.

Add this code:

```
// Re-assemble the sentence with a new delimiter
String dashSentence = String.join("-->", words);
System.out.println(dashSentence);
```

Output:

```
Please type a sentence: The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
...
The-->quick-->brown-->fox-->jumped-->over-->the-->lazy-->dog.
```

Show the original string

You've called methods to uppercase, lowercase, replace, split, and join the string that's referenced by the `sentence` variable. As a reminder that *strings are immutable*, print out the original string that was typed by the user:

```
// Print the initial sentence. Notice it has not changed.
System.out.println(sentence);
```

When you run the program, you'll see that the printed string is the same string that you first typed into the console. This is a reminder that a string may not be changed in place. All the methods you called take the original string, manipulate its characters, and return a new string, without modifying the original.

Step Three: Compare strings

In the final section of this tutorial, your program will ask the user to type a secret phrase, and then check to see if the user typed the correct value.

Since `String` is a *reference* type, if you use the `==` operator to compare two strings, you won't compare the actual strings, but comparing the reference (or memory address) of the strings. This is usually *not* what you want.

First, prompt the user to enter the (not-so-well-kept) secret word. Under the `Step 3:` comment in your code, add this code:

```
// ***********  Step 3: Compare Strings  *************
String secretWord = "Secret!";
System.out.print("Enter the secret word (hint: it's '" + secretWord + "') ");
String userSecretWord = scanner.nextLine();
```

When this code runs and the user types a word, that word will be in `userSecretWord`.

Now, compare these strings using both the `==` operator and the `equals()` method to see the difference:

```
// Compare using ==
boolean matchEqualityOperator = secretWord == userSecretWord;
System.out.println("Using '==': " + matchEqualityOperator);

// Compare using equals()
boolean matchEquals = secretWord.equals(userSecretWord);
System.out.println("Using '.equals(): " + matchEquals);
```

When you run the program, you'll see this:

```
Enter the secret word (hint: it's `Secret!') Secret!
Using '==': false
Using '.equals(): true
```

`secretWord` and `userSecretWord` may have the same value, but they're in two different memory locations, so when you used `==` to compare, the result was `false`. Only `equals()` compares the actual values.

But what if you type "secret!"? If you try that, you'll find that the result is `false`. The `equals()` method respects case, so an uppercase "S" and lowercase "s" are considered different. If you would rather consider uppercase and lowercase letters as the same, use `equalsIgnoreCase()`:

```
// Compare using equalsIgnoreCase()
boolean matchEqualsIgnoreCase = secretWord.equalsIgnoreCase(userSecretWord);
System.out.println("Using '.equalsIgnoreCase(): " + matchEqualsIgnoreCase);
```

This time if you type "SECRET!", "secret!" or "Secret!", the result will be `true`, indicating a match.

Next steps

You've begun to learn about the `String` class and some of its methods. You can explore more string methods in the Java documentation.

Try combining the methods you have learn to solve more complex problems. For example:

- Can you find the number of words in a sentence? (Hint: use `split()` and the `length` property of the resulting array)
- Can you get the last character in a string? (Hint: use `substring()` and `length()`)