

# Introduction to Methods

by Sophia



## WHAT'S COVERED

In this lesson, you will learn how to use methods in Java. Specifically, this lesson covers:

## Table of Contents

- [1. Methods](#)
- [2. More Built-In Methods](#)
  - [2a. toUpperCase\(\) method and toLowerCase\(\) method](#)

## 1. Methods

So far, the tutorials have focused on variables and values used in Java code. Moving forward, you will focus on how to use that data as a basis for accomplishing a task. Operators have a limited capacity to do the work needed. However, methods can be used and are essential when more complex actions are needed. A **method** is a named chunk of code that can be called upon or invoked to perform a certain predefined set of actions. Methods correspond to an action or a behavior when programming. You have already observed and have used a couple of common methods, such as `print()` and `println()` to display text in the console. You have also used `nextLine()` and `nextInt()` to read user input.



### HINT

These are examples of the many prebuilt methods that are available for use and are located in the Java library. Not all methods are prebuilt in the Java library. It is also possible, and sometimes necessary, to create and use methods on our own.

A method needs access to the values needed to carry out its tasks. Most methods take one or more parameters to pass values of the appropriate data types into the method. This ensures that the code in the body of the method can use them. Depending on the nature of the task, a method may return, or send back, a value of a particular type that represents the outcome of the task. Using a method is referred to as "calling a method."

When discussing methods in Java, both of the terms "parameter" and "arguments" are used to refer to the values passed to a method. A **parameter** is a special kind of variable that provides a name to a piece of data being passed. It is a channel, through which the value is passed. The name of the parameter is then treated like a local variable in the body of the method. The code in the method accesses the passed data via this

variable. An **argument** refers to the variable or literal value that is passed through the parameter when the method is called.

➞ **EXAMPLE** Here is an example of calling a method from the Java library that you have already used:

```
public class Hello {  
    public static void main(String[] args) {  
        // Call the println() method with a String argument  
        System.out.println("Hello, world!");  
    }  
}
```

In this code, the `println()` method, which is provided by the `System` class's output stream, is named `out`, and displays the value passed to it. The `println()` method has the return type `void`, meaning that it does not return a value. Therefore, there is no variable and equal sign to the left of the method call to save a value.

The basic pattern for the **definition of a method** is:

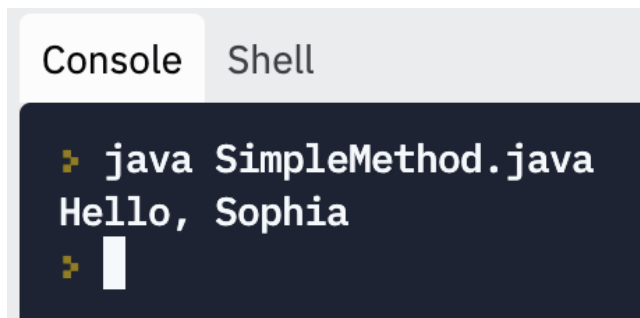
```
static ReturnType MethodName(Parameters...)
```

The return type indicates the type of data, which may include `int`, `double`, `char`, or `String` returned by the method. If a method doesn't return any data, then it will have the return type `void`. Typically, the return value is assigned to a variable of the appropriate type, though this is not the only way to handle a return value. Future tutorials will discuss this in greater detail.

The following program defines a method called `sayHello()`. It creates a greeting that is then printed out. For the time being, accept that the keyword `static` needs to start a line that defines a method, that can be called on its own. Compare how the `sayHello()` method is defined below and how it is called:

```
class SimpleMethod {  
  
    /* This method takes a String parameter with the name of the  
       person to be greeted. It concatenates "Hello, " and the  
       name as a single String and returns that String */  
    static String sayHello(String name) {  
        return "Hello, " + name;  
    }  
  
    public static void main(String[] args) {  
        String person = "Sophia";  
        /* Call the sayHello() method passing in String with name as the  
           argument. The returned String is assigned to the String  
           variable greeting */  
        String greeting = sayHello(person);  
        // Print out the greeting  
        System.out.println(greeting);  
    }  
}
```

When running in Replit, the output from this program looks like this:



```
Console Shell
❏ java SimpleMethod.java
Hello, Sophia
❏
```

Don't worry if all of the pieces involved in creating a method are not making sense yet. The important thing is that you understand what methods do. When you do review Java documentation, you will see the arguments often shortened as "args" in documentation about Java, and in the parameter for the `main()` method.

Next, you will focus on a couple of other methods that are part of the Java library. These library methods are already defined and available for use, so you will not have to define them. Unlike the `sayHello()` method covered previously, most library methods are not static. They cannot be used by themselves. Methods do need to be called by means of an **object**. Future tutorials will cover classes and objects in much more detail.

For now, keep in mind that many calls to Java methods look like this example using the `length()` method:

#### ➞ EXAMPLE

```
int length = name.length();
```

In this case, `name` is a `String` variable that holds the `String` data for which we want to get the length. The `length()` method doesn't take any parameters, so the parentheses are empty. The `length()` method returns an `int` value representing the number of characters in the `String` stored in the `name` variable.

➞ EXAMPLE Here is a brief program to give the complete context (the code should be in a file named `StringLength.java`):

```
class StringLength {
    public static void main(String[] args) {
        // This String object contains the name Sophia
        String name = "Sophia";
        // The length() method can be used to get the length
        // (number of characters) in a String.
        int length = name.length();
        System.out.println("The name " + name + " has a length of " +
            length + ".");
    }
}
```

When running in Replit, the program produces the following output:

```
Console Shell
❯ java StringLength.java
The name Sophia has a length of 6.
❯
```

Most methods use parameters. Here is another example using a method named `charAt()`. This method works on a `String` and gets the character at the requested position. The example requests the first letter in the `String` by passing in the argument `0`. While people usually start counting at `1`, Java usually starts counting at `0`.

➞ **EXAMPLE** Using a method named `charAt()`:

```
class FirstLetter {
    public static void main(String[] args) {
        String name = "Sophia";
        /* The charAt() method takes a parameter indicating the
           position at which to get the character. Passing 0 as
           the argument gets the first character in the String. */
        char firstLetter = name.charAt(0);
        System.out.println("The first letter in " + name +
            " is " + firstLetter + ".");
    }
}
```

Running this program, saved in a file named `FirstLetter.java`, should produce this result:

```
Console Shell
❯ java FirstLetter.java
The first letter in Sophia is S.
❯
```

Now that you have had a chance to see a couple of the built-in methods in Java, try your hand at it.



**Directions:** Using Replit, try using the `length()` and `charAt()` methods that you just learned in the examples above. Remember that these methods only work on `String` data.



Java requires that all code be part of a class. The name of the class and the name of the source code file must match. That means that a program in a class named `SecondLetter` must be defined in a file named `SecondLetter.java`.



Every Java program has a `main()` method that runs when the program starts. The previous examples show

you how to set up the `main()` method in the class for the program.

It's easy and quite normal to make mistakes at first, so work slowly and be patient with yourself. As you write more Java code, you will develop a sense for how the language works—and where you are likely to make mistakes.



#### TERMS TO KNOW

#### Method

Methods are a piece of code that runs when it is called. We have the ability to pass in data into those functions which are called parameters.

#### Parameter

A parameter is the actual variable name(s) when we define a function definition.

#### Method Definition

The method definition is the first line when we create a new function to be used in a program.

#### Argument

An argument is the actual value(s) that is being passed into the method when it is being called.

#### Object

An object is an instance of a class that has properties and methods that are encapsulated or part of it.

---

## 2. More Built-In Methods

In the previous sample that used the Java `charAt()` method, the first character in a `String` was at position, or index, 0. Many methods that work on `String` data use this zero-based indexing.

For example, consider the 12-character `String` "Java is fun!"

J	a	v	a		i	s		f	u	n	!
0	1	2	3	4	5	6	7	8	9	10	11

The `indexOf()` method returns the index of the first occurrence of a character (`char`) in a `String` value.

Here is an example of the program:

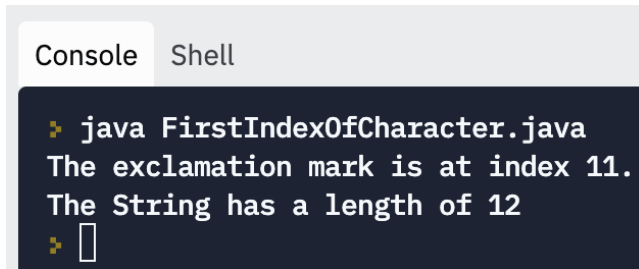
```
class FirstIndexOfCharacter {
    public static void main(String[] args) {
        String motto = "Java is fun!";
        // Find the index of the 1st occurrence of the !.
        // The index of the 1st occurrence is zero-based
        int index = motto.indexOf('!');
        System.out.println("The exclamation mark is at index " + index +
            ".");
        // The length of the String is the actual # of characters
    }
}
```

```

    int length = motto.length();
    System.out.println("The String has a length of " + length);
}
}

```

Running the program should produce this output:



```

Console  Shell
❯ java FirstIndexOfCharacter.java
The exclamation mark is at index 11.
The String has a length of 12
❯

```

## 2a. toUpperCase() method and toLowerCase() method

These two methods do what their names indicate. They convert a String value to the corresponding uppercase or lowercase form.

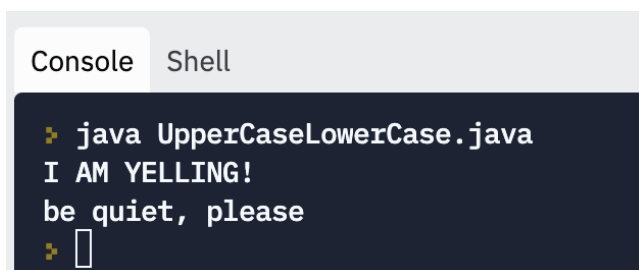
The following program illustrates how to convert a string value case:

```

class UpperCaseLowerCase {
    public static void main(String[] args) {
        String statement = "I am yelling!";
        String yelling = statement.toUpperCase();
        System.out.println(yelling);
        // Re-use the statement variable, no data type specified
        statement = "BE QUIET, PLEASE";
        // Overwrite uppercase with lowercase version
        statement = statement.toLowerCase();
        System.out.println(statement);
    }
}

```

Running the program should produce the following output:



```

Console  Shell
❯ java UpperCaseLowerCase.java
I AM YELLING!
be quiet, please
❯

```



HINT

The toUpperCase() and toLowerCase() methods have no effect on characters that are not letters.



TRY IT

**Directions:** Using the length() and the charAt() methods covered above, write a program that displays the

last character in a String variable using the following approach:

- Determine the length of the value in the String variable.
- Ensure the last index of the last character is 1 less than the length, since Java starts counting the positions in the String at 0.
- Use `charAt()` to get the character (char) in the last position.
- Use a char variable for the result, keeping in mind that `charAt()` returns a char value.

## REFLECT

The output from the program should look something like this (yourString value may vary):

```
Console Shell
> java LastCharacter.java
Text: The quick brown fox jumps over the lazy dog
Last character: g
>
```

Here is a screenshot of the code that produces the output shown:

```
LastCharacter.java x Console Shell
1 class LastCharacter {
2     public static void main(String[] args) {
3         String text = "The quick brown fox jumps over the lazy dog";
4         int length = text.length();
5         int lastIndex = length - 1;
6         char lastCharacter = text.charAt(lastIndex);
7         System.out.println("Text: " + text);
8         System.out.println("Last character: " + lastCharacter);
9     }
10 }
11
```

## SUMMARY

In this lesson, you learned the basics about Java **methods** and how to use them. You explored examples of methods, including **built-in methods** in the Java library used when working with String values, such as **toUpperCase() method** and **toLowerCase() method**, which convert a String value to the corresponding uppercase or lowercase form.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source [cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf](https://cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf)

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source [py4e.com/html3/](https://py4e.com/html3/)

## TERMS TO KNOW

### Argument

An argument is the actual value(s) that is being passed into the method when it is being called.

### Method

Methods are a piece of code that runs when it is called. We have the ability to pass in data into those

functions which are called parameters.

**Method Definition**

The method definition is the first line when we create a new function to be used in a program.

**Object**

An object is an instance of a class that has properties and methods that are encapsulated or part of it.

**Parameter**

A parameter is the actual variable name(s) when we define a function definition.