Java Methods

‹ Previous Next ›

A method is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parantheses **()**. Java provides some pre-defined methods, such as `System.out.println()` , but you can also create your own methods to perform certain actions:

Example

Create a method inside MyClass:

```
public class MyClass {
  static void myMethod() {
    // code to be executed
  }
}
```

Example Explained

- `myMethod()` is the name of the method
- `static` means that the method belongs to the MyClass class and not an object of the MyClass class. You will learn more about objects and how to access methods through objects later in this tutorial.

- `void` means that this method does not have a return value. You will learn more about return values later in this chapter

---

Call a Method

To call a method in Java, write the method's name followed by two parantheses **()** and a semicolon**;**

In the following example, `myMethod()` is used to print a text (the action), when it is called:

Example

Inside `main`, call the `myMethod()` method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "I just got executed!"
```

Run example »

A method can also be called multiple times, if you want:

Example

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
```

```
    myMethod();
    myMethod();
    myMethod();
  }
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Run example »

Method Parameters

Information can be passed to functions as parameter.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The following example has a method that takes a `String` called `fname` as parameter. When the method is called, we pass along a first name, which is used inside the method to print the full name:

Example

```
public class MyClass {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
```

```
// Jenny Refsnes
// Anja Refsnes
```

Run example »

Return Values

The `void` keyword indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as `int`, `char`, etc.) instead of `void`, and use the `return` keyword inside the method:

Example

```
public class MyClass {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

Run example »

This example returns the sum of a method's **two parameters**:

Example

```
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }
```

```
  public static void main(String[] args) {
    System.out.println(myMethod(5, 3));
  }
}
// Outputs 8 (5 + 3)
```

Run example »

You can also store the result in a variable (recommended):

Example

```
public class MyClass {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
// Outputs 8 (5 + 3)
```

Run example »

‹ Previous Next ›