Java Methods

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**.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. 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 parentheses **()** 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!"

A method can also be called multiple times:

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!

# Java Method Parameters

## Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.

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

When a **parameter** is passed to the method, it is called an **argument**. So, from the example above: fname is a **parameter**, while Liam, Jenny and Anja are **arguments**.

## Multiple Parameters

You can have as many parameters as you like:

### Example

public class MyClass {

static void myMethod(String fname, int age) {

System.out.println(fname + " is " + age);

}

public static void main(String[] args) {

myMethod("Liam", 5);

myMethod("Jenny", 8);

myMethod("Anja", 31);

}

}

// Liam is 5

// Jenny is 8

// Anja is 31

Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

## Return Values

The void keyword, used in the examples above, 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)

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)

You can also store the result in a variable (recommended, as it is easier to read and maintain):

### 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)

## A Method with If...Else

It is common to use if...else statements inside methods:

### Example

public class MyClass {

// Create a checkAge() method with an integer variable called **age**

static void checkAge(int age) {

// If age is less than 18, print "access denied"

if (age < 18) {

System.out.println("Access denied - You are not old enough!");

// If age is greater than 18, print "access granted"

} else {

System.out.println("Access granted - You are old enough!");

}

}

public static void main(String[] args) {

checkAge(20); // Call the checkAge method and pass along an age of 20

}

}

// Outputs "Access granted - You are old enough!"

# Java Method Overloading

## Method Overloading

With**method overloading**, multiple methods can have the same name with different parameters:

### Example

int myMethod(int x)

float myMethod(float x)

double myMethod(double x, double y)

Consider the following example, which have two methods that add numbers of different type:

### Example

static int plusMethodInt(int x, int y) {

return x + y;

}

static double plusMethodDouble(double x, double y) {

return x + y;

}

public static void main(String[] args) {

int myNum1 = plusMethodInt(8, 5);

double myNum2 = plusMethodDouble(4.3, 6.26);

System.out.println("int: " + myNum1);

System.out.println("double: " + myNum2);

}

Instead of defining two methods that should do the same thing, it is better to overload one.

In the example below, we overload the plusMethod method to work for both int and double:

### Example

static int plusMethod(int x, int y) {

return x + y;

}

static double plusMethod(double x, double y) {

return x + y;

}

public static void main(String[] args) {

int myNum1 = plusMethod(8, 5);

double myNum2 = plusMethod(4.3, 6.26);

System.out.println("int: " + myNum1);

System.out.println("double: " + myNum2);

}

**Note:** Multiple methods can have the same name as long as the number and/or type of parameters are different.