


Method parameters - Method declaration and calling

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.

Parameter



```
public static void displayValue(int num){  
    System.out.println("The value is " + num);  
}
```

When a **parameter** is passed to the method, it is called an **argument**.

The values we passed to the method.

Argument



```
displayValue(5);
```



```
displayValue(5);
```

The argument **5** is copied into
the parameter variable num

```
public static void displayValue(int num){  
    System.out.println("The value is " + num);  
}
```


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 Main {  
    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
```


Example

```
public class Main {  
    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, Anja are **arguments**.

You can have multiple parameters, as many parameters as you like:

Example

```
public class Main {  
    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
```

What is method overloading and how do you achieve it?

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)
```

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

Method Overloading improves code readability and re-usability.

It's easier to remember one method name instead of remembering multiple method names.

Example

Without method overloading: we have two methods with different names, which are doing the same thing

```
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);  
}
```


Example

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

```
static int plusMethod(int x, int y) {  
    return x + y;  
}
```

We overload the method named **plusMethod()** to work for both: int and double

```
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);  
}
```


Method Overloading Rules

- Parameters of the overloaded method must be different (number of parameters or data types of parameters)
- Return Type of the overloaded method can be same or different
- Method can be overloaded any number of times
- Any method can be overloaded

