



# **SNS COLLEGE OF TECHNOLOGY**



**Coimbatore-35.**

**An Autonomous Institution**

**Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A++’ Grade  
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
COURSE CODE & NAME : 23CST205 - Object Oriented Programming Using Java**

**II YEAR/ III SEMESTER**

**UNIT – II INTRODUCTION TO JAVA**

**Topic: BASICS OF JAVA PROGRAMMING- JAVA METHODS**



# Methods in Java

## 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:

Object Oriented Programming concepts–

Objects/23CST205 - Object Oriented Programming Using

Java/Ms R.ARUNA/AP/CSE/SNSCT



# Methods in Java



## Example

Create a method inside Main:

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

## Example Explained

- `myMethod()` is the name of the method
- `static` means that the method belongs to the Main class and not an object of the Main 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

Object Oriented Programming concepts–



# Methods in Java

## 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 Main {
    static void myMethod() {
        System.out.println("I just got executed!");
    }

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

// Outputs "I just got executed!"



# Methods in Java



A method can also be called multiple times:

## Example

```
public class Main {
    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!
```

Object Oriented Programming concepts–Objects/23CST205 -  
Object Oriented Programming Using Java/MS

R.ARUNA/AP/CSE/SNSCT



# 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:



# Java Method Parameters



## 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` and `Anja` are **arguments**.





# Java Method Parameters



## Multiple Parameters

You can have 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
```

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.





# Java Method Parameters

## 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 Main {
    static int myMethod(int x) {
        return 5 + x;
    }

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



# Java Method Parameters

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

## Example

```
public class Main {  
    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)
```



# Java Method Parameters

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

## Example

```
public class Main {  
    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)
```

Object Oriented Programming concepts–Objects/23CST205 - Object Oriented Programming Using Java/Ms R.ARUNA/AP/CSE/SNSCT

