# Day 12

# Java OOP Concepts - Method & Constructor

## Method

- 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.
- We can call/invoke the methods through object reference.

**Why use methods?**

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

## How to Create a Method?

To create a method in Java, you need to define it within a class.

**Syntax:**

```
returnType methodName(parameters) {
    // method body
}
```

**Example:**

```
public class Example {
    // Method without parameters and return value
    void printHello() {
        System.out.println("Hello, World!");
    }
}
```

## How to Call a Method?

To call a method, you use the method name followed by parentheses. If the method requires parameters, you pass them inside the parentheses.

**Example:**

```
public class Example {
    void printHello() {
        System.out.println("Hello, World!");
    }

    public static void main(String[] args) {
        Example obj = new Example();
        obj.printHello(); // Calling the method
    }
}
```

## Different Ways to Create Methods in Java

1. **No Parameters and No Return Value**: A method that performs an action but doesn't take any inputs or return any outputs.
2. **No Parameters and Returns a Value**: A method that doesn't take any inputs but returns a result.
3. **Takes Parameters and No Return Value**: A method that takes inputs but doesn't return anything.
4. **Takes Parameters and Returns a Value**: A method that takes inputs and returns a result based on those inputs.

## Passing arguments in 2 ways

1) Call by Value
2) Call by Reference

**Explanation:**

1. **Call by Value:**
    - The method callByValue accepts a primitive data type (int data).
    - When the callByValue method is called, a copy of the value is passed to the method.
    - Modifications inside the method affect only the local copy, not the original value.
    - After the method execution, the original data value in the main method remains unchanged.
2. **Call by Reference (Using Object Reference):**
    - The method callByReference accepts an object of class Example as its parameter.

- When the callByReference method is called, the reference to the object is passed to the method.
- Modifications inside the method affect the actual object, because the reference points to the same memory location.
- After the method execution, the changes made inside the method are reflected in the original object.

## Constructor

- A constructor is a block of code similar to the method.
- It is called when an object is created.
- At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

**There are two types of constructors in Java:**

1) Default constructor
2) Parameterized constructor

**Note:** It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

### Rules for creating Java constructor

**There are two rules defined for the constructor.**

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

## this & this()

- **this**: Refers to the current object or instance variables/methods.
- **this():** Calls another constructor from the same class.

**this keyword:**

- **Purpose**: Refers to the current instance (object) of the class.
- **Use cases**:
  - It is used to differentiate between instance variables and parameters when they have the same name.
  - It can be used to invoke instance methods or access instance variables from within the class.

- **Example**:

```
class Example {
    int x;

    Example(int x) {
        this.x = x; // Refers to the instance variable 'x'
    }

    void print() {
        System.out.println(this.x); // Refers to the instance variable 'x'
    }
}

public class Main {
    public static void main(String[] args) {
        Example obj = new Example(5);
        obj.print(); // Outputs: 5
    }
}
```

**this() constructor call:**

- **Purpose**: Calls another constructor from the same class (constructor chaining).
- **Use case**: It is used to avoid code duplication by calling one constructor from another, especially when multiple constructors exist with different parameters.
- **Important rule**: this() must always be the first statement inside a constructor.
- **Example**:

```
class Example {
    int x;
    int y;

    Example() {
        this(10, 20); // Calls the parameterized constructor
        System.out.println("Default constructor");
    }

    Example(int x, int y) {
        this.x = x;
        this.y = y;
        System.out.println("Parameterized constructor");
    }
}

public class Main {
    public static void main(String[] args) {
        Example obj = new Example(); // Calls default constructor
    }
}
```

## Differences Between Java Method and Constructor

### Method:

1. A method can have any name.
2. A method may or may not return a value.
3. If a method does not return a value, it must be specified as void.
4. A method can accept parameters or arguments.
5. Methods need to be explicitly invoked or called through an object.
6. Methods are used to define and execute logic.

### Constructor:

1. A constructor must have the same name as the class.
2. A constructor never returns a value (not even void).
3. There is no need to specify void for a constructor.
4. A constructor can accept parameters or arguments.
5. A constructor is automatically invoked when an object is created.
6. Constructors are used to initialize the values of variables.

# Lab Assignments

**1. Library Management System**

**Objective:** Create a simple library management system to track books and their availability.

- **Class Book:**
  - Fields: String title, String author, boolean isAvailable
  - Constructors:
    - A default constructor.
    - A parameterized constructor to initialize all fields.
  - Methods:
    - borrowBook(): Sets isAvailable to false if the book is available.
    - returnBook(): Sets isAvailable to true.
    - printBookDetails(): Prints details of the book.
- **Lab Task:**
  - Create a few Book objects using both constructors.
  - Test borrowing and returning books, and print their details.

**2. Bank Account Simulation**

**Objective:** Simulate basic banking operations such as deposit and withdrawal.

- **Class BankAccount:**
  - Fields: String accountNumber, double balance
  - Constructors:
    - A default constructor.
    - A parameterized constructor to initialize accountNumber and balance.
  - Methods:
    - deposit(double amount): Adds the amount to the balance.
    - withdraw(double amount): Subtracts the amount from the balance if sufficient funds are available.
    - printAccountSummary(): Prints the account number and balance.
- **Lab Task:**
  - Create a BankAccount object and test deposit and withdrawal operations.
  - Print the account summary after each operation.

**3. Student Grade Management**

**Objective:** Manage student grades and calculate average grades.

- **Class Student:**
    - Fields: String name, int[] grades
    - Constructors:
        - A default constructor.
        - A parameterized constructor to initialize name and grades.
    - Methods:
        - addGrade(int grade): Adds a new grade to the grades array.
        - calculateAverage(): Returns the average of grades.
        - printStudentDetails(): Prints the student's name and average grade.
- **Lab Task:**
    - Create a Student object with some grades.
    - Add more grades, calculate the average, and print the student's details.

**4. Product Inventory System**

**Objective:** Manage a product inventory with details and stock levels.

- **Class Product:**
    - Fields: String name, double price, int stock
    - Constructors:
        - A default constructor.
        - A parameterized constructor to initialize name, price, and stock.
    - Methods:
        - updateStock(int quantity): Updates the stock level by adding the quantity.
        - sellProduct(int quantity): Decreases stock by the quantity if sufficient stock is available.
        - printProductDetails(): Prints the product details including stock level.
- **Lab Task:**
    - Create a Product object and test updating stock and selling products.
    - Print the product details after each operation.

**5. Appointment Scheduling**

**Objective:** Schedule and manage appointments.

- **Class Appointment:**
    - Fields: String description, String date, String time
    - Constructors:
        - A default constructor.
        - A parameterized constructor to initialize description, date, and time.
    - Methods:
        - updateTime(String newTime): Updates the appointment time.
        - printAppointmentDetails(): Prints the appointment details.
- **Lab Task:**
    - Create an Appointment object with a description, date, and time.
    - Update the time of the appointment and print the appointment details.