OOPs (Object-Oriented Programming)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

An Object can be defined as an instance of a class. It contains an address and takes up some space in memory.
Example: A dog is an object because it has states like color, name, breed, etc.

Class

A class can also be defined as a blueprint from which you can create an individual object. Class does not consume any space.

Constructor

A Constructor in Java is a block of codes like the method. It is called when an instance of the class is created. Every time when we create an object by using the new keyword, 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.

Rules for Creating Java Constructor

There are the following rules for defining a 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.

Note:

We can use access modifiers while declaring a constructor. It controls the object creation. A constructor may be private, protected, public or default.

Types of Java Constructors

There are two types of constructors in Java:

1. Default Constructor (No-arg constructor)
2. Parameterized Constructor

Default Constructor

When a constructor does not have any parameter, is known as default constructor.

syntax:

```
< class_name >(){}
```

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Parameterized Constructor

A constructor that has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

```
class Student {
    int id;
    String name;
    int age;

    Student(int i, String n) {
        id = i;
        name = n;
    }

    Student(int i, String n, int a) {
```

```
        id = i;
        name = n;
        age = a;
    }

    void display() {
        System.out.println(id + " " + name + " " + age);
    }
}

public class Main {
    public static void main(String args[]) {
        Student s1 = new Student(111, "Karan");
        Student s2 = new Student(222, "Aryan", 25);
        s1.display();
        s2.display();
    }
}```

    ## Difference Between Constructor and Method in Java

There are many differences between constructors and methods. They are given below.

|Java Constructor|Java Method|
|---|---|
|A constructor is used to initialize the state of an object.|A method is used to expose the behavior of an object.|
|A constructor must not have a return type.|A method must have a return type.|
|The constructor is invoked implicitly.|The method is invoked explicitly.|
|The Java compiler provides a default constructor if we do not have any constructor in a class.|The method is not provided by the compiler in any case.|
|The constructor's name must be same as the class name.|The method name may or may not be same as the class name.|

## Java Copy Constructor

Java does not support the copy constructor. However, we can copy the values from one object to another, like a copy constructor in C++.

There are the following three ways to copy the values of one object into another:

- By Using a Constructor
- By Assigning the Values of One Object to Another
```

- By Using the clone() Method of the Object Class

```java
//Using constructor
class Student {
    int id;
    String name;

    Student(int i, String n) {
        id = i;
        name = n;
    }

    Student(Student s) {
        id = s.id;
        name = s.name;
    }

    void display() {
        System.out.println(id + " " + name);
    }
}

public class Main {
    public static void main(String args[]) {
        Student s1 = new Student(111, "Karan");
        Student s2 = new Student(s1);
        s1.display();
        s2.display();
    }
}```

## Ways to Create an Object of a Class

## 1. **Using the `new` Keyword** (Most Common Way)

### Description:

This is the most straightforward and widely used method to create an object.
### Example:

```java
class MyClass {
    void display() {
        System.out.println("Hello from MyClass");
    }
```
```

```
}

public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass(); // Object created using new
        obj.display();
    }
}```


## 2. **Using `Class.forName()` Method (Reflection)**

### Description:

Used when the class name is stored as a string or loaded dynamically at runtime.
### Example:

```java
class MyClass {
    void show() {
        System.out.println("Created using Class.forName");
    }
}

public class Main {
    public static void main(String[] args) throws Exception {
        MyClass obj = (MyClass) Class.forName("MyClass").newInstance(); // Deprecated since Java 9
        obj.show();
    }
}```

## 3. **Using `clone()` Method**

### Description:

Creates a copy of an existing object. The class must implement `Cloneable`.
### Example:

```java
class MyClass implements Cloneable {
    int x = 10;

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
```

```
    }
}

public class Main {
    public static void main(String[] args) throws CloneNotSupportedException {
        MyClass obj1 = new MyClass();
        MyClass obj2 = (MyClass) obj1.clone(); // Cloning obj1
        System.out.println("Cloned object x = " + obj2.x);
    }
}```

## 4. **Using Factory Method**

### Description:

A method that returns a new object instance. Used for abstraction and control over instantiation.
### Example:

```java
class MyClass {
    private MyClass() {
        // private constructor
    }

    public static MyClass createObject() {
        return new MyClass(); // Factory method
    }

    public void hello() {
        System.out.println("Created via factory method");
    }
}

public class Main {
    public static void main(String[] args) {
        MyClass obj = MyClass.createObject(); // Factory method call
        obj.hello();
    }
}```


## Anonymous Object

Anonymous objects are objects that are instantiated without storing their reference in a variable. They are used for one-time operations (e.g., method

```
calls) and are discarded immediately after use.

- No reference variable: Cannot reuse the object.
- Created & used instantly: Saves memory for short-lived tasks.
- Common in event handling (e.g., button clicks).

```java
class MyClass {
    void show() {
        System.out.println("Anonymous object called me!");
    }
}

public class Main {
    public static void main(String[] args) {
        new MyClass().show();  // anonymous object
    }
}
```

`this` Keyword

The `this` keyword refers to **the current object** inside a class.

Uses of `this`:

1. To refer to current class instance variables.
2. To invoke current class methods or constructors.
3. To return the current object.

Example:

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;         // this refers to current object's id
        this.name = name;     // this refers to current object's name
    }
}```
## Static Variables, Static Methods, and Static Blocks

###  Static Variable
- Shared across all instances.
```
```

- Memory is allocated only once (at class loading time).

```java
class Employee {
    int id;
    String name;
    static String company = "TechCorp"; // static variable

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

Static Method

- Belongs to the class, not object.
- Can **only access static members**.

```
class Utility {
    static int add(int a, int b) {
        return a + b;
    }
}
```

```
class Init {
    static {
        System.out.println("Static block executed");
    }

    static void display() {
        System.out.println("Static method");
    }
}
```

```
Static block executed
Static method
```

Instance Variables and Instance Methods

Instance Variables

- Declared inside a class but **outside methods**.
- Each object gets its own copy.

Instance Methods

- Operate on instance variables.
- Need an object to be called.

```
class Car {
    String model;  // instance variable

    void setModel(String model) {
        this.model = model;
    }

    void getModel() {
        System.out.println("Model: " + model);
    }
}
```

Initialization Blocks

- Blocks used to **initialize common code** for all constructors.
- Two types:
  - **Instance Initialization Block
  - **Static Initialization Block

Instance Initialization Block (IIB)

- Runs **every time** an object is created.
- Executes **before the constructor**.

```
class Demo {
    {
        System.out.println("Instance block called");
    }

    Demo() {
        System.out.println("Constructor called");
    }
}
```

Static Block

- Executes **once** when the class is loaded.
- Used to initialize static data.

```
class Init {
    static {
        System.out.println("Static block executed");
    }

    static void display() {
        System.out.println("Static method");
    }
}
```