OOPs in Java – Complete Notes with Examples

What is OOP?

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which encapsulate data and behaviour.

Class

- A class is a user defined blueprint or prototype from which objects are created.
- A class is a collection of fields, methods and constructor.

Object

- It is a basic unit of Object Oriented Programming and represents the real-life entities.
- When an object of a class is created, the class is said to be **instantiated**.
- Created using the **new** keyword.

Example Class:

```
class Car {                                          // class
   String brand;                                     // variables/fields
   int speed;

   Car(String brand, int speed) {                    // Constructor
      this.brand = brand;
      this.speed = speed;
   }

   void displayInfo() {                              // Method
      System.out.println("Brand: " + brand + ", Speed: " + speed);
   }
}
```

Example of Object Creation:

```
public class Main {

   public static void main(String[] args) {
      Car c1 = new Car("BMW", 200);
      c1.displayInfo();
   }

}
```

Constructor

- Object creation syntax:

```
    Test t = new Test();
```

- **Test** → class Name
- **t** → Reference variables
- **=** → assignment operator
- **new** → keyword used to create object
- **Test ()** --→ constructor

- ; → statement terminator
- When we create new instance (Object) of a class using new keyword, a constructor for that class is called.
- Constructors are used to initialize the instance variables of a class.

Rules to declare Constructor

- Constructor name class name must be same.
- Constructor is able to take parameters.
- Constructor not allowed explicit return type (return type declaration not possible).

Types of Constructors

1. Default Constructor

- If we don't write constructor for a class then compiler generates one constructor for you that constructor is called default constructor, And it is not visible in code.
- Compiler generates Default constructor inside the class when we are not providing any type of constructor (0-arg or parameterized).
- The compiler generated default constructor is always 0-argumnetconstructor with empty implementation (empty body).

2. User Defined Constructor

- Inside the class if we are declaring at least one constructor that is called user defined constructor.
- The main objective of constructor is to initialize some values to instance variables during object creation time.
- Constructors are used to write the functionality of project that functionality is executed during object creation.
- Inside the class it is possible to declare multiple constructors.

Core Principles of OOP:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

Inheritance

- The process of acquiring fields(variables) and methods(behaviours) from one class to another class is called inheritance.
- The main objective of inheritance is code extensibility whenever we are extending class automatically the code is reused.
- In inheritance one class giving the properties and behaviour & another class is taking the properties and behaviour.
- Class which gives the properties is called as **Parent/Super/Base** class.
- Class which takes the properties is called as **Child/Sub/Derived** class.
- Inheritance is also known as is-a relationship. By using extends keyword we are achieving inheritance concept.
- **extends** keyword used to achieve inheritance & it is providing relationship between two classes.
- In java parent class is giving properties to child class and Child is acquiring properties from Parent.
- To reduce length of the code and redundancy of the code sun people introduced inheritance concept.
- With child objects we can call parent class fields or methods.

Types of Inheritance

There are 5 types of inheritance:

1. Single Inheritance:

- One class has one and only one direct super class is called single inheritance.
- In absence of direct super class, every class is implicitly a sub class of **Object** class in java.

```
class A {

    void show() {
        System.out.println("Class A");
    }

}
class B extends A {

    void display() {
         System.out.println("Class B");
    }

}
```

2. Multi Level Inheritance:

- One sub class is extending parent class and that sub class will become parent of next extended class.
- This flow is called as multi level inheritance.

```
class Parent {

    void m1() {
        System.out.println("Class Parent");
    }

}

class Child extends Parent {

    void m2() {
         System.out.println("Class Child");
    }

}
```

```
class GrandChild extends Child {

    void m3() {
        System.out.println("Class Grand Child");
    }

}
```

3. Hierarchical Inheritance:

- More than one sub class is extending a single parent is called as Hierarchical inheritance.
- Class Son and Daughter both are extending Parent.

```
class Parent {

    void m1() {
        System.out.println("Class Parent");
    }

}

class Son extends Parent {

    void m2() {
        System.out.println("Class Son");
    }

}

class Daughter extends Parent {

    void m3() {
        System.out.println("Class Daughter");
    }

}
```

4. Multiple Inheritance:

- One sub class is extending more than one super class is called as multiple inheritance.
- But **Java does not support multiple inheritance with classes** as it creates ambiguity issue for the JVM.
- This ambiguity issue occurs if both parent classes have same method and if we try to call that method with child object then JVM will be confused whether to access the method from parent 1 or parent 2.
- This ambiguity issue is also called as Diamond problem.
- Class Son and Daughter both are extending Parent.

```
class Parent1 {

    void m1() {
        System.out.println("Class Parent 1");
    }

}

class Parent2 {

    void m1() {
        System.out.println("Class Parent 2");
    }

}

class Child extends Parent1, Parent2{

    void m3() {
        System.out.println("Class Daughter");
    }

}
```

5. Hybrid Inheritance:

- This is a combination of two or more types of inheritance. Possible with interfaces.

Polymorphism:

- Polymorphism is the ability of an object to take on many forms
- Polymorphism shows same functionality (method name same) with different logics execution.
- The ability to appear in more forms is called polymorphism.
- Polymorphism is a Greek word poly means many and morphism means forms.

There are two types of polymorphism in java

1. **Compile time polymorphism / static binding / early binding**
   Method execution decided at compilation time
   Example: - method overloading.
2. **Runtime polymorphism/dynamic binding/late binding.**
   Method execution decided at runtime.
   Example: - method overriding.

- If we want achieve overloading concept one class is enough.
- It is possible to overload any number of methods in single java class.

1.Compile time polymorphism [Method Overloading]

- If java class allows two or more methods with same name but different number of arguments such type of methods are called overloaded methods.
- We can overload the methods in three ways in java language

a. By passing different number of arguments to the same methods.

```
void m1(int a){
}
void m1(int a, int b){
}
```

b. Provide the same number of arguments with different data types.

```
void m1(int a){
}
void m1(char ch){
}
```

c. Provide the same number of arguments with different order of arguments.

```
void m1(int a, char b){
}

void m1(char b, int a){
}
```

- If we want achieve overloading concept one class is enough.
- It is possible to overload any number of methods in single java class.

Types of overloading: -

- Method overloading explicitly by the programmer
- Constructor overloading.

2.Runtime polymorphism [Method Overriding]:

- If we want to achieve method overriding we need two classes with parent and child relationship.
- The parent class method contains some implementation (logics).
- If child is satisfied use parent class method.
- If the child class not satisfied (required own implementation) then override the method in Child class.
- A subclass has the same method as declared in the super class it is known as method overriding.
- The parent class method is called ===> **overridden method**
- The child class method is called ===> **overriding method**
- Constructor cannot be overridden
- **@Override** annotation is optional but recommended

While overriding methods must follow these rules: -

- While overriding child class method signature & parent class method signatures must be same otherwise, we will get compilation error.
- The return types of overridden method & overriding method must be same.
- While overriding the methods it is possible to maintains same level permission or increasing order but not decreasing order, if you are trying to reduce the permission compiler generates error message "attempting to assign weaker access privileges".
- You will not be able to override final methods. (Final methods are preventing overriding).
- Static methods are bounded with class hence we are unable to override static methods.
- If sub class defines a static method with same signature as parent class static method, then it doesn't indicate overriding, it's called as **method hiding**.

```
class Animal {
    void sound() {
        System.out.println("Animal Sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog Barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog(); // Upcasting
        a.sound();
    }
}
```

Encapsulation

- The process of binding the data and code in a single unit is called as Encapsulation.
- It also **restricts direct access** to some of the object's components, which is a form of **data hiding**.
- Think of it like a capsule that contains medicine—you don't need to know what's inside to use it safely.

How to Achieve Encapsulation in Java:

- Declare class variables as **private.**
- Provide public **getter** and **setter** methods to access and update the values**.**

```
public class Employee {
    // Private data members
    private String name;
    private int age;

    // Public getter and setter methods
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        if(age > 18) {
            this.age = age;
        } else {
            System.out.println("Age must be greater than 18.");
        }
    }
}
```

Access Modifiers

There are total 4 access modifiers in Java.

1. **public**
    - public modifier is applicable for classes, methods, variables and constructors.
    - Public members are accessible across the project in all packages and classes.
2. **private**
    - private modifier is applicable for methods, variables and constructors.
    - We cannot use private at class level.
    - private members are accessible only within the same class.
3. **protected**
    - protected modifier is applicable for methods, variables and constructors.
    - We cannot use protected at class level.
    - protected members are accessible within the same package and also in sub classes of other packages.
4. **default**
    - default modifier is applicable for classes, methods, variables and constructors.
    - Do not mention default, if we do not mention any access modifier then by default it will at as **default.**
    - default members are accessible within the same package.

Abstraction

There are two types of methods in java based on signature.

a. Normal methods

b. Abstract methods

Normal methods: - (component method/concrete method)

- Normal method is a method which contains method declaration as well as method implementation.

Example: -

```
void m1() --->method declaration
{
body; --->method implementation
}
```

Abstract methods: -

- The method which is having only method declaration but not method implementations such type of methods are called abstract Methods.
- In java every abstract method must end with ";".

Example:

```
abstract void m1 (); ----> method declaration
```

Based on above representation of methods the classes are divided into two types

1) Normal classes.

2) Abstract classes.

Normal classes: -

- Normal class is an ordinary java class it contains only normal methods.

Example: -

```
class Test //normal class
{
void m1(){  //normal method
body;
}
}
```

Abstract class: -

- If any abstract method inside the class that class must be abstract.
- A class can be abstract even if it doesn't have any abstract methods.

Abstract modifier: -

- Abstract modifier is applicable for methods and classes but not for variables.
- To represent particular class is abstract class and particular method is abstract method to the compiler use abstract modifier.
- The abstract class contains declaration of abstract methods, it says abstract class partially implemented class hence for partially implemented classes object creation is not possible.
- If we are trying to create object of abstract class compiler generate error message "class is abstract con not be instantiated".
- Abstract class may contain abstract methods or may not contains abstract methods but object creation is not possible.

Interfaces

- Interface is also one of the type of class it contains only abstract methods. And Interfaces not alternative for abstract class it is extension for abstract classes.
- The abstract class contains at least one abstract method but the interface contains only abstract methods.
- Interfaces giving the information about the functionalities and these are not giving the information about internal implementation.
- Inside the source file it is possible to declare any number of interfaces. And we are declaring the interfaces by using interface keyword.

- Interface contains abstract method, for these abstract methods provide the implementation in the implementation classes.
- Implementation class is nothing but the class that implements particular interface.
- While providing implementation of interface methods that implementation methods must be public methods otherwise compiler generate error message "attempting to assign weaker access privileges".
- By default the methods which are in interface are public abstract.
- The interface contains constants and these constants by default public static final.
- For the interfaces object creation is not possible.