# OBJECT ORIENTED PROGRAMMING WITH JAVA

## BCA II SEM(NEP)

## NOTES

**Prepared By**

**Mr. Prabhu Kichadi,** BE, MTECH

**9880437187**

# UNIT – II

**Inheritance and Polymorphism:**

Inheritance in java, Super and sub class, Types of inheritance, Overriding, Polymorphism, Dynamic binding, Abstract class, Interface in java, Packages in java - defining and importing user defined packages.

## Inheritance in java:

Inheritance is a process of acquiring properties from one class to another class. It is one of the key concepts of oops.

Inheritance is a key concept in object-oriented programming that allows a class to be based on another class, called the superclass or parent class. The subclass or child class inherits the fields and methods of the superclass, allowing it to reuse and extend the functionality of the superclass.

In Java, inheritance is implemented using the extends keyword. A subclass can inherit the members of the superclass, including public, protected, and package-private fields and methods. Private members are not inherited.

**Super/Parent/Base Class**: A class which provides properties to another class is called super class.

**Sub/Child/Derived Class:** A class which inherits properties from another class is called Sub class.

**Advantages of inheritance:**

**Code reuse:** Inheritance enables the reuse of code from existing classes, allowing programmers to avoid duplicating code in multiple classes.

E**xtensibility**: Inheritance allows new classes to be built upon existing classes, making it possible to add new features and functionality to an application without modifying the original code.

**Easy maintenance:** as classes are separated maintenance becomes easy to developers.

**Polymorphism:** Inheritance supports polymorphism, which is the ability of objects to take on many forms. Subclasses can be used in place of their parent classes, enabling the creation of more flexible and adaptable programs.

**Modularity:** Inheritance facilitates modularity by dividing complex systems into smaller, more manageable classes. Each class can focus on a specific set of tasks, making it easier to understand, test, and maintain the code.

**Efficiency:** Inheritance can improve program performance by reducing the amount of code that needs to be written and executed. This can lead to faster execution times and reduced memory usage.

## Inheritance Syntax:

```
class SuperClass
{
        //parent class members
}

class Subclass extends Superclass
{
   // subclass members
}
```

## Example:

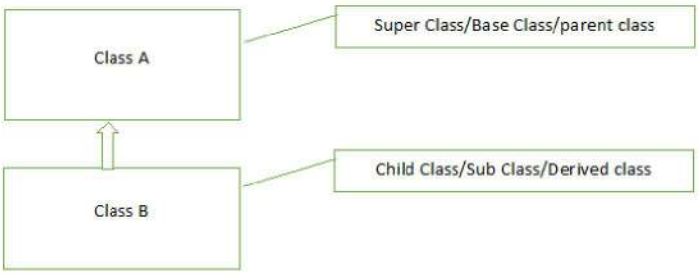
```
class Animal
{
   public void speak()
   {
      System.out.println("Animal speaks");
   }
}
class Dog extends Animal
{
   public void bark()
   {
      System.out.println("Dog barks");
   }
}
public class MainClass
{
   public static void main(String[] args)
   {
      Dog dog = new Dog();
      dog.speak(); // Output: Animal speaks
      dog.bark(); // Output: Dog barks
   }
}
```

In this example, Dog is a subclass of Animal. The extends keyword in the Dog class declaration indicates that Dog inherits from Animal. The speak() method is inherited from Animal and can be called on a Dog object, along with the bark() method that is specific to the Dog class.

## Super and sub class:

**A superclass** is a class that is being extended by a subclass. A superclass can provide a set of common properties and behavior that can be inherited by multiple subclasses.

**A subclass** is a class that inherits properties and behavior from a superclass, which is a class that is being extended. The subclass can add its own properties and behavior while retaining the characteristics of its superclass.

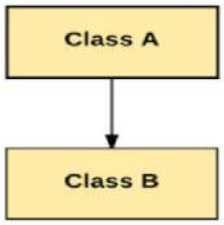



## Types of inheritance:

1. **Single inheritance**
2. **Multilevel inheritance:**
3. **Hierarchical inheritance:**
4. **Multiple inheritance:**
5. **Hybrid inheritance:**

**1. Single inheritance:** A subclass can inherit properties and methods from a single superclass. This is the most common type of inheritance.

**Ex:**

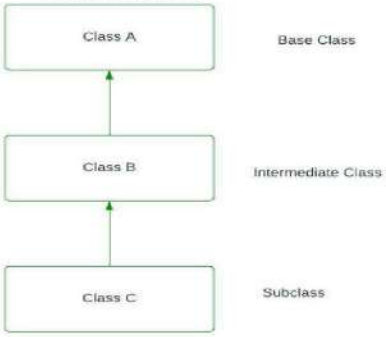



```
public class Animal
{
  public void eat()
  {
    System.out.println("I can eat");
  }
}
```

```
public class Dog extends Animal
{
  public void bark()
  {
    System.out.println("I can bark");
  }
}
```

In this example, Dog is a subclass of Animal, and it inherits the eat() method from Animal.

2. **Multilevel inheritance:**A subclass can inherit properties and methods from a superclass, which in turn inherits properties and methods from another superclass.

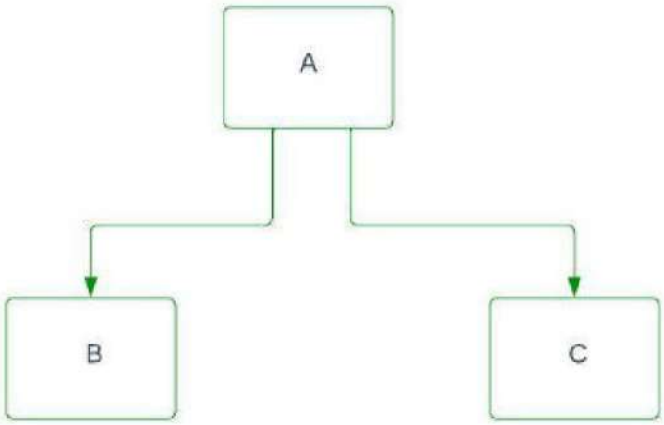



```
public class Animal
{
  public void eat()
  {
    System.out.println("I can eat");
  }
}
public class Dog extends Animal
{
  public void bark()
  {
    System.out.println("I can bark");
  }
}
public class Bulldog extends Dog
{
  public void guard()
  {
    System.out.println("I can guard");
  }
}
```

In this example, Bulldog is a subclass of Dog, which is a subclass of Animal. Bulldog inherits the eat() and bark() methods from Dog, which in turn inherits the eat() method from Animal.

### 3. Hierarchical inheritance:

Multiple subclasses can inherit properties and methods from a single superclass.

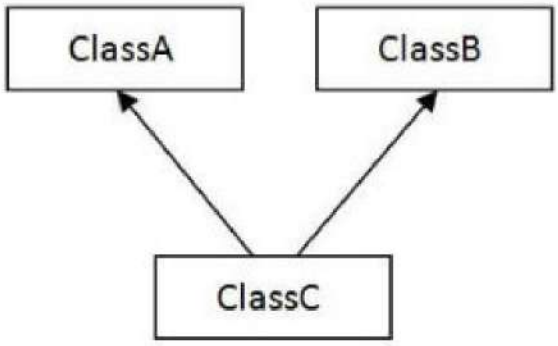



```
public class Animal
{
  public void eat()
  {
    System.out.println("I can eat");
  }
}

public class Dog extends Animal
{
  public void bark()
  {
    System.out.println("I can bark");
  }
}

public class Cat extends Animal
{
  public void meow()
  {
    System.out.println("I can meow");
  }
}
```

In this example, Dog and Cat are subclasses of Animal. Both Dog and Cat inherit the eat() method from Animal.

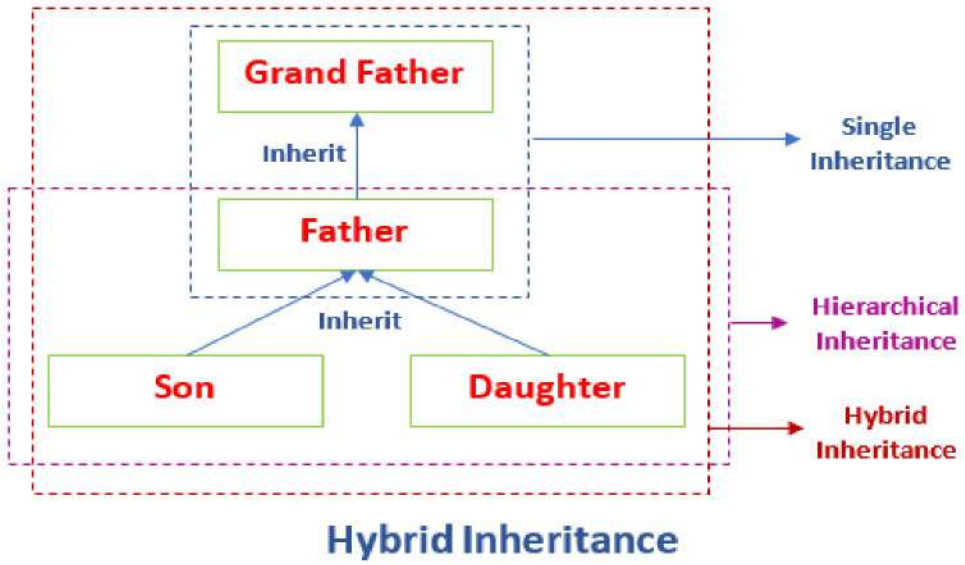
**4. Multiple inheritance:**

A subclass can inherit properties and methods from multiple super classes. **Java does not support multiple inheritance. But using interfaces we can achieve multiple inheritance in java.**




```
public class Vehicle
{
  public void drive()
  {
    System.out.println("I can drive");
  }
}
public class Radio
{
  public void playMusic()
  {
    System.out.println("I can play music");
  }
}

public class Car extends Vehicle, Radio
{
  public void honk()
  {
    System.out.println("Beep beep!");
  }
}
```

**5. Hybrid inheritance:** This is a combination of two or more types of inheritance.




Hybrid Inheritance

```
//parent class
class GrandFather
{
	public void showG()
	{
		System.out.println("He is grandfather.");
	}
}
//inherits GrandFather properties
class Father extends GrandFather
{
	public void showF()
	{
		System.out.println("He is father.");
	}
}
//inherits Father properties
class Son extends Father
{
	public void showS()
	{
		System.out.println("He is son.");
	}
}
```

```
//inherits Father properties
public class Daughter extends Father
{
        public void showD()
        {
                System.out.println("She is daughter.");
        }
}
```

**Polymorphism & Overriding:** Polymorphism is the concept of an object which takes multiple forms on a single message. Object behaves different on different levels of inheritance.

Polymorphism allows a single variable of a superclass type to refer to objects of its own type or any of its subclasses. When the method is called on an object, Java automatically determines which version of the method to call based on the type of the object at runtime.

**Types of polymorphism:**
**1. Compile Time Polymorphism (Early Binding-Static Binding).**
**2. Run time Polymorphism (Late Binding-Dynamic Binding).**

**1. Compile Time Polymorphism (Early Binding-Static Binding).- Method Overloading:**
When a class provides multiple methods with the same name but different parameters, it is called method overloading. Java automatically selects the correct version of the method to call based on the arguments passed to the method.

refers to the ability of a programming language to determine which method to call at **compile time, based on the type and number of arguments provided to the method.**

**Ex:**

```
public class Calculator
{
   public int add(int x, int y)
   {
      return x + y;
   }

   public double add(double x, double y)
   {
```

```
    return x + y;
  }

  public int add(int x, int y, int z)
  {
    return x + y + z;
  }
}

class MainClass
{

      public static void main(String args[])
      {
      Calculator calc = new Calculator();
      int sum1 = calc.add(2, 3);         // calls the first add method, returns 5
      double sum2 = calc.add(2.5, 3.5);   // calls the second add method, returns
      int sum3 = calc.add(2, 3, 4);       // calls the third add method, returns 9
      }
}
```

**2. Run time Polymorphism (Late Binding-Dynamic Binding). – Method Overriding:**
When a subclass provides its own implementation of a method that is already defined in its superclass, it is called method overriding. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.

**Ex:**

```
class Animal
{
  public void makeSound()
  {
    System.out.println("Animal is making a sound.");
  }
}
class Cat extends Animal
{
  public void makeSound()
  {
    System.out.println("Meow!");
  }
}
```

```
class Dog extends Animal
{
   public void makeSound()
   {
      System.out.println("Woof!");
   }
}

class MainClass
{
  public static void main(String[] args)
  {
        Animal animal = new Animal();
        Animal cat = new Cat();
        Animal dog = new Dog();

        animal.makeSound();  // prints "Animal is making a sound."
        cat.makeSound();     // prints "Meow!"
        dog.makeSound();     // prints "Woof!"

   }
}
```

In this example, the makeSound() method is called on an Animal object, a Cat object, and a Dog object. Java determines which version of the method to call at runtime, based on the type of the object that the method is called on.

**Abstraction:** Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
Ways to achieve Abstraction.

There are two ways to achieve abstraction in java

1. **Abstract class (0 to 100%)**
2. **Interface (100%)**

**Abstract class:** If the class is specified with abstract keyword that class will become abstract class. an abstract class is a class that cannot be instantiated, but can be subclassed.

- An abstract class can contain both abstract and non-abstract methods. An abstract method is a method that does not have an implementation and is declared using the abstract keyword.
- A non-abstract method, on the other hand, has an implementation and can be called directly.
- an abstract class is a class that cannot be instantiated, but can be subclassed.

```
class Shape
{
        public abstract void draw();

}
class Circle extends Shape
{
        public void draw()
        {
                System.out.println("Drawing circle");
        }
}
class Rectangle extends Shape
{
        public void draw()
        {
                System.out.println("Drawing Rectangle");
        }
}
class MainClass
{
    public static void main(String[] args)
        {
                Shape s1 = new Circle();
                s1.draw();

                s1 = new Rectangle();
                s1.draw();
        }
}
```

**Interface in java**: an interface is a collection of abstract methods and constants that define a contract for a class to implement.

- By default all the methods in interface are public, abstract
- All the data members are public, static and final.

- It is similar to an abstract class, but unlike an abstract class, it cannot contain any implementation code.

- An interface is declared using the **interface** keyword, and its methods are declared using the abstract keyword (although the abstract keyword is optional, as all methods in an interface are implicitly abstract).

- The **implementation class** must provide definition for all the abstract methods of an interface.

- Using **implements** keyword a class can inherit interface.

**Ex:**

```
interface Animal
{
   void makeSound();
}
class Cat implements Animal
{
   public void makeSound()
         {
      System.out.println("Meow!");
   }
}
class Dog implements Animal
{
   public void makeSound()
         {
      System.out.println("wow!");
   }
}
class MainClass
{
         public static void main(String[] args)
         {
```

```
            Animal a1 = new Cat();
            a1.makeSound();

            Animal a2 = new Dog();
            a2.makeSound();
        }
}
```

**Output:**

## Packages in java - defining and importing user defined packages.

a package is a collection of related classes, interfaces, and sub-packages. It provides a way to organize and encapsulate code, and also helps to avoid naming conflicts.

**Steps:**

**Creating Package:**

1. To define a package in Java, you use the package keyword followed by the name of the package.
2. The package statement should be the first line of code in your Java source file, before any imports or class declarations.

**example:**

```
package com.example.myapp;

public class MyClass
{
   // class code here
}
```

4. Compilation command would be

> javac -d . ProgramName.java

5. Execution Command would be

>java com.example.myapp.MyClass

**Importing a class from a package:**

6. instead using Fully Qualified ClassName to refer a class from package we can use import statement to access the class:

```
imort com.example.myapp.MyClass;
class Test
{
        public static void main(String[] args)
        {
                MyClass m1 = new MyClass();

        }
}
```

7. any number of import statements can used in a program to access any number of classes.



**Note: Refer all Java Lab journal programs on this unit.**