Unit II Part A

Inheritance

Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

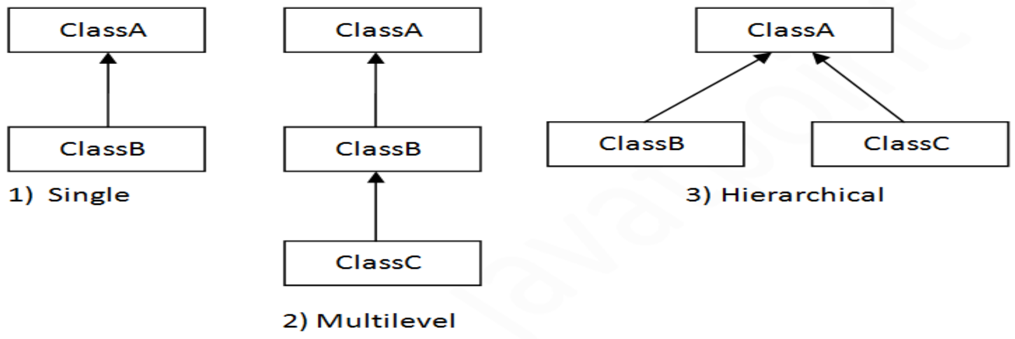
The syntax of Java Inheritance

1. **class Subclass-name extends Superclass-name**
2. **{**
3. **//methods and fields**
4. **}**

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

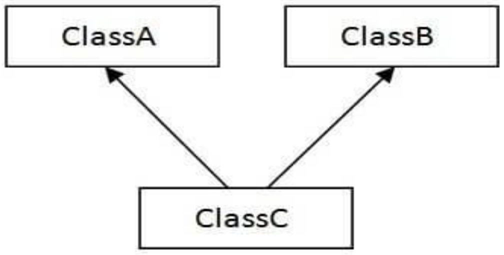
Types of inheritance in java

On the basis of class, there can **be three types of inheritance in java: single, multilevel and hierarchical**. In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

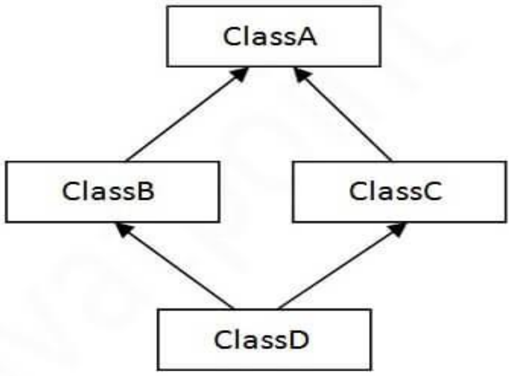


Multiple inheritance and Hybrid with multiple Inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

Single Inheritance Example

```
/*  Single Inheritance Example  */
class Animal
{
        void eat()
        {
                System.out.println("eating...");
        }
}
class Dog extends Animal
{
        void bark()
        {
                System.out.println("barking...");
        }
}
class TestInheritance
{
        public static void main(String args[])
        {
                Dog d=new Dog();
                d.bark();
                d.eat();
        }
}
```

Multilevel Inheritance Example

```
/*  Multilevel Inheritance Example  */
```

AKV

```
class Animal
{
        void eat()
        {
                System.out.println("eating...");
        }
}
class Dog extends Animal
{
        void bark()
        {
                System.out.println("barking...");
        }
}
class BabyDog extends Dog
{
        void weep()
        {
                System.out.println("weeping...");
        }
}
class TestInheritance2
{
        public static void main(String args[])
        {
                BabyDog d=new BabyDog();
                d.weep();
                d.bark();
                d.eat();
        }
}
```

AKV

Hierarchical Inheritance Example

```
/*  Hierarchical Inheritance Example  */
class Animal
{
        void eat()
        {
                System.out.println("eating...");
        }
}
class Dog extends Animal
{
        void bark()
        {
                System.out.println("barking...");
        }
}
```

```
class Cat extends Animal
{
	void meow()
	{
		System.out.println("meowing...");
	}
}
class TestInheritance3
{
	public static void main(String args[])
	{
		Cat c=new Cat();
		c.meow();
		c.eat();
		//c.bark();//Error
	}
}
```

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.

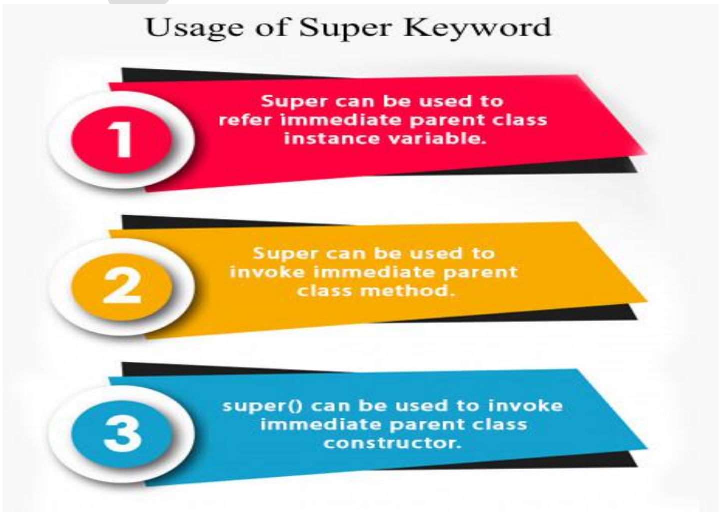
```
/*  Method Overriding Example  */
class Vehicle
{
	void run()
	{
		System.out.println("Vehicle is running");
	}
}
class TestOverriding extends Vehicle
{
	void run()
	{
		System.out.println("Bike is running safely");
	}
	public static void main(String args[])
	{
		TestOverriding obj = new TestOverriding();
		obj.run();
	}
}
```

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

AKV

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
/*  Example : super is used to refer immediate parent class instance variable*/
   class Animal
  {
         String color="white";
   }
   class Dog extends Animal
         {
                  String color="black";
                  void printColor()
                  {
                           System.out.println(color);//prints color of Dog class
                           System.out.println(super.color);//prints color of Animal class
                  }
   }
   class TestSuper1
         {
                  public static void main(String args[])
                  {
                           Dog d=new Dog();
                           d.printColor();
                  }
         }
```

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
/*  Example : super can be used to invoke parent class method*/
   class Animal
         {
                  void eat()
                  {
                           System.out.println("eating...");
                  }
   }
   class Dog extends Animal
         {
                  void eat()
                  {
                           System.out.println("eating bread...");
                  }
                  void bark()
                  {
```

AKV

```
                        System.out.println("barking...");
                }
                void work()
                {
                        super.eat();
                        bark();
                }
    }
    class TestSuper2
        {
                public static void main(String args[])
                {
                        Dog d=new Dog();
                        d.work();
                }
        }
```

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor.

```
/*  Example : super is used to invoke parent class constructor*/
    class Animal
    {
                Animal(String a)
                {
                        System.out.println(a+" is created");
                }
    }
    class Dog extends Animal
    {
                Dog(String d)
                {
                        super("Animal");
                        System.out.println(d+" is created");
                }
    }
    class TestSuper3
   {
                public static void main(String args[])
                {
                        Dog d=new Dog("Dog");
                }
   }
```

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable

AKV

2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable (It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
/*  Example :  Java final Variable*/
class Bike
{
        final int speedlimit=90;//final variable
        void run()
        {
                speedlimit=400;
                System.out.println("Speedlimit="+speedlimit);
        }
        public static void main(String args[])
        {
                Bike obj=new  Bike();
                obj.run();
        }
}
```

2) Java final method

If you make any method as final, you cannot override it.

```
/*  Example :  Java final Method*/
class Bike
{
   final void run()
        {
                System.out.println("running");
        }
}

class Honda extends Bike
{
   void run()
```

AKV

```
        {
                System.out.println("running safely with 100kmph");
        }
   public static void main(String args[])
        {
                Honda honda= new Honda();
                honda.run();
   }
}
```

3) Java final class

If you make any class as final, you cannot extend it.

```
/*  Example :  Java final class*/
final class Bike
{
}
class Honda1 extends Bike
{
        void run()
        {
                System.out.println("running safely with 100kmph");
        }
        public static void main(String args[])
        {
                Honda1 honda= new Honda1();
                honda.run();
        }
 }
```

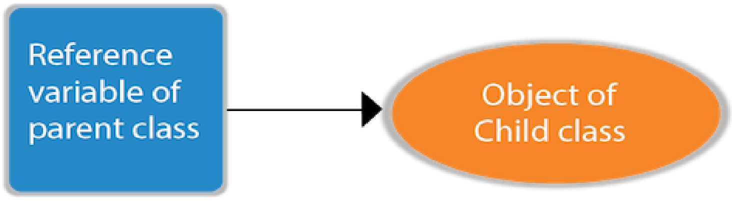
Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



```
class A{}
class B extends A{}
A a=new B();//upcasting
```

Static binding means when the type of object which is invoking the method is determined at compile time by the compiler. While **Dynamic binding** means when the type of object which is invoking the method is determined at run time by the compiler.

```
/* Dynamic method Dispatch*/
class Bike
{
        void run()
        {
                System.out.println("running");
        }
}
class Splendor extends Bike
{
        void run()
        {
                System.out.println("running safely with 60km");
        }
   public static void main(String args[])
        {
                Bike b = new Splendor();//upcasting
                b.run();
        }
}
```

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstraction in Java

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

AKV

internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

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

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method

```
/*  Example :  Java abstract class*/
abstract class Bike
{
	abstract void run();
}
class Honda2 extends Bike
{
	void run()
	{
		System.out.println("running safely..");
	}
	public static void main(String args[])
	{
		Bike obj;
		obj= new Honda2(); //Upcasting
		obj.run();
	}
}
```

AKV

```
/*  Example :  Java abstract class*/
abstract class Shape
{
        double d1,d2;
        Shape(double a,double b)
        {
                d1=a;
                d2=b;
        }
        abstract double area();
}
class Rectangle extends Shape
{
        Rectangle(double a,double b)
        {
                super(a,b);
        }
        double area()
        {
                return d1*d2;
        }
}
class Triangle extends Shape
{
        Triangle(double a,double b)
        {
                super(a,b);
        }
        double area()
        {
                return (d1*d2)/2;
        }
}
class TestAbstraction1
{
        public static void main(String args[])
        {
                Shape ref;
                ref=new Rectangle(5.0,10.0);
                System.out.println(ref.area());
                ref=new Triangle(1.0,2.0);
                System.out.println(ref.area());
        }
}
```

AKV