VIPS
Technical Campus
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

SCHOOL OF ENGINEERING AND TECHNOLOGY

Course : OBJECT ORIENTED PROGRAMMING

Paper Code: AIML-202,IIOT-202

Faculty : Dr. Shivanka

Assistant Professor

VIPS

INHERITANCE

- Inheritance is an OOPS concept by which a class can acquire the characteristics of another class. i.e it's a method of creating a new class based on already existing class.
- The class from which its inheriting is called base class and the class which is inheriting is called sub class.
- It is the inheriting properties of parent class or base class to sub class or child class.
- **subclass (child) - the class that inherits from another class**
- **superclass (parent) - the class being inherited from**
- To inherit from a class, use the extends keyword.

Why Do We Need Java Inheritance?

- Code Reusability: The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
- Method Overriding: Method Overriding is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.
- Abstraction: The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

How to use Syntax of Inheritance in Java?

- To inherit the parent class, a child class must include a keyword called **"extends."** The keyword "extends" enables the compiler to understand that the child class derives the functionalities and members of its parent class.
- To understand this in an easier way, let us verify the syntax for inheritance in Java.

Syntax:

```
class SubClass extends ParentClass
{
//DataMembers;
//Methods;
}
```

Types of Inheritance in Java

The different 5 types of Inheritance in java are as follows:

1. Single inheritance.
2. Multi-level inheritance.
3. Multiple inheritance.
4. Hierarchical Inheritance.
5. Hybrid Inheritance.

1. Single Inheritance

- In single inheritance, a single subclass extends from a single superclass.

For example :Class A inherits from class B.

Java Single Inheritance

Single Inheritance


```
class A
{
 int a, b;
 void display()
 {
 System.out.println("Inside class A values = "+a+" "+b);
 } }
class B extends A
{
 int c;
 void show() {
 System.out.println("Inside Class B values=" +a+ " "+b+" " +c);  }
}
```

```
class SingleInheritance
{
 public static void main(String args[])

{
 B obj = new B(); //derived class object
 obj.a=10;
 obj.b=20;
 obj.c=30;
 obj.display();
 obj.show();
 }
}
```

Save this program class name including main function SingleInheritance.java

Output :-

```
Inside class A values = 10
20
Inside Class B values=10
20 30
```

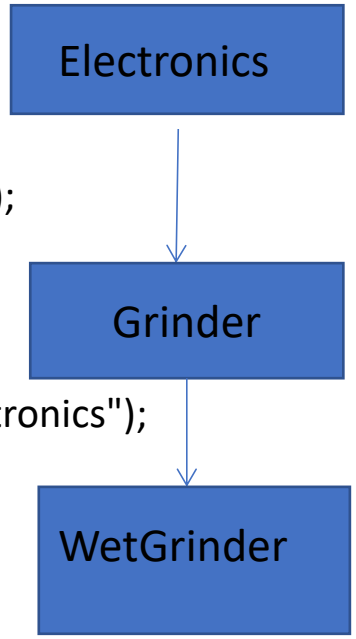
Multi-Level Inheritance in Java

- In Multi-Level Inheritance in Java, a class extends to another class that is already extended from another class.
- For example, if there is a class A that extends class B and class B extends from another class C, then this scenario is known to follow Multi-level Inheritance.

Java Multilevel Inheritance

MULTI LEVEL INHERITANCE


```
class Electronics {
public Electronics(){
System.out.println("Class Electronics");
}
public void deviceType() {
System.out.println("Device Type: Electronics");
}}
class Grinder extends Electronics {
public Grinder() {
System.out.println("Class Grinder");
}
public void category() {
System.out.println("Category - Grinder");
}}
```




```
class WetGrinder extends Grinder {
public WetGrinder() {
System.out.println("Class WetGrinder");
}
public void grinding_tech() {
System.out.println("Grinding Technology-
WetGrinder");
}
}
public class Tester {
public static void main(String[] arguments) {
WetGrinder wt= new WetGrinder();
wt.deviceType();
wt.category();
wt.grinding_tech();
}}
```

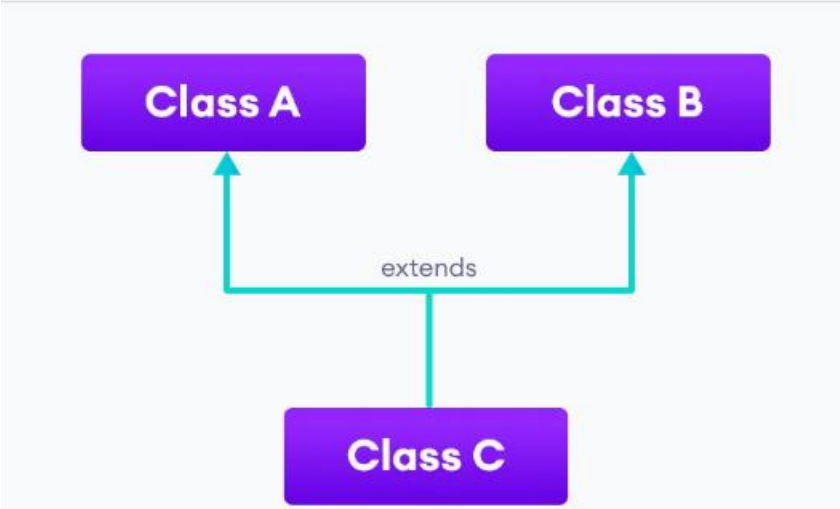
Output :-
Class Electronics
Class Grinder
Class WetGrinder

Device Type: Electronics
Category: Grinder
GrindingTechnology:
WetGrinder

Multiple Inheritance in Java

- Defining derived class from numerous base classes or superclass is known as 'Multiple Inheritance'. In this case, there is more than one superclass, and there can be one or more subclasses.

- Multiple inheritances are available in object-oriented programming with C++, but it is not available in Java.



Java Multiple Inheritance

Why Multiple Inheritance is not supported in Java?

- Note: Java doesn't support multiple inheritance. However, we can achieve multiple inheritance using interfaces.
- Let's consider a case in Inheritance. Consider a class A, class B and class C. Now, let class C extend class A and class B. Now, consider a method read() in both class A and class B. The method read() in class A is different from the method read() in class B. But, while inheritance happens, the compiler has difficulty in deciding on which read() to inherit. So, in order to avoid such kind of ambiguity, multiple inheritance is not supported in Java.

Interface in Java

- An interface in Java is a **blueprint of a behaviour**. A Java interface contains static constants and abstract methods.

- The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship.

Characterstics of Interface

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the behaviour.
- Interface **do not have constructor**.
- Represent behaviour as interface unless every sub-type of the class is guarantee to have that behaviour.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared **abstract.**

A Java library example is Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Syntax:

```
interface {


    // declare constant fields

    // declare methods that abstract

    // by default.

}
```

To declare an interface, use the **interface keyword**. It is used to provide total abstraction. That m

Why do we use an Interface?

- It is used to achieve total abstraction.
- Since java does not support **multiple inheritances** in the case of class, by using an **interface** it can achieve multiple inheritances.
- Any class can extend only 1 class but can any class implement infinite number of interface.
- It is also used to achieve **loose coupling**.
- Interfaces are used to implement **abstraction.** So the question arises why use interfaces when we have abstract classes?
- The reason is, abstract classes may contain **non-final variables**, whereas **variables** in the **interface are final, public and static.**

```
// A simple interface


interface Player
{
    final int id = 10;
//final+public+static
    int move();
}
```

Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

Multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:

Example:-

```
interface Printable{  //1st Interface
void print();  }
interface Showable{  //2nd interface
void print();  }
  class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }  }
```

Output : Hello

Interface Inheritance

A class implements an interface, but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output:-

Hello

Welcome

Difference Between Class and Interface

S.No	Class	Interface
1	In class, you can instantiate variables and create an object.	In an interface, you can't instantiate variables and create an object
2	Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
3	The access specifiers used with classes are private, protected, and public.	In Interface only one specifier is used- Public

Abstract class in Java

1. Abstraction is a process of hiding the implementation details and showing only functionality to the use.
2. 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. It cannot be instantiated.

- **Points to Remember**

❑An abstract class must be declared with an **abstract** keyword.

❑It can have **abstract and non-abstract methods**.

❑It cannot be **instantiated**.

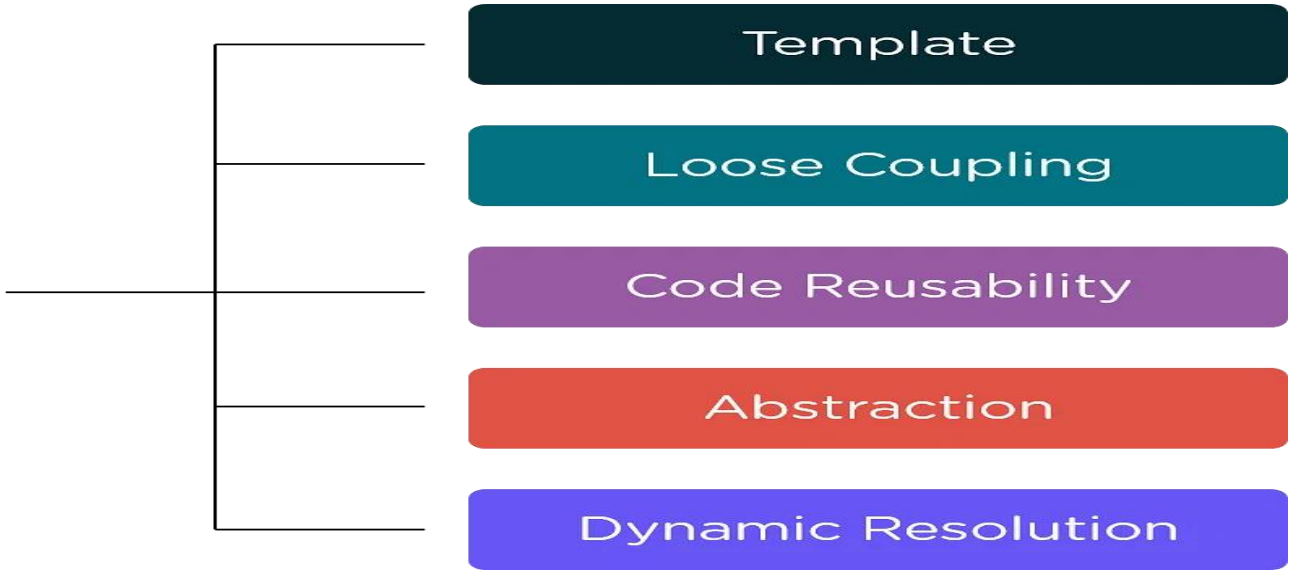
❑It can have **constructors and static methods** also.

❑It can have **final methods** which will force the **subclass not to change the body** of the method.

Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be Instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Features of Abstract Class



Features of Abstract Class

Template

The abstract class in Java enables the best way to execute the process of data abstraction by providing the developers with the option of hiding the code implementation. It also presents the end-user with a template that explains the methods involved.

Loose Coupling

Data abstraction in Java enables loose coupling, by reducing the dependencies at an exponential level.

Code Reusability

Using an abstract class in the code saves time. We can call the abstract method wherever the method is necessary. Abstract class avoids the process of writing the same code again.

Abstraction

Data abstraction in Java helps the developers hide the code complications from the end-user by reducing the project's complete characteristics to only the necessary components.

Dynamic Resolution

Using the support of dynamic method resolution, developers can solve multiple problems with the help of one abstract method.

The Syntax for Abstract Class

To declare an abstract class, use the access modifier first, then the "abstract" keyword, and the class name shown below.

//Syntax:

<Access_Modifier> abstract class <Class_Name> {

//Data_Members;

//Statements;

//Methods;}

Abstract Class Example


```
abstract public class AbstractDemoVehicle {
    int no_of_tyres;
    abstract void start();//without defining or body
}
class car extends AbstractDemoVehicle{
    void start()
    {System.out.println("Start with key");
    }
    class scooter extends AbstractDemoVehicle{
    void start(){System.out.println("Start with kick");
    }}
```

```
  public static void main(String[] args) {
        // TODO code application logic here
  //AbstractDemoVehicle a=new
AbstractDemoVehicle();
  //It will show error because class is abstact
  car c=new car();
  c.start();
  scooter x=new scooter();
 x.start();}

}
```

Output:-
Start with key
Start with kick

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

Simply, **abstract class** achieves partial abstraction **(0 to 100%)** whereas **interface** achieves fully abstraction **(100%).**

Difference between abstract class and Interface

Abstract Class	Interface
1) Abstract class can have **abstract and non-abstract methods**.	Interface can have **only abstract methods**. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't** support **multiple inheritance**.	Interface supports **multiple inheritance**.
3) Abstract class can have **final, non-final, static and non-static** variables.	Interface has only **static and final** variables.
4) Abstract class can provide the **implementation** of interface.	Interface **can't** provide the **implementation** of abstract class.
5) The abstract keyword is used to declare **abstract class**.	The interface keyword is used to declare **interface**.
6) An abstract class can **extend another Java class** and implement multiple Java interfaces.	An interface can **extend another** Java interface only.
7) An abstract class can be extended using keyword **"extends".**	An interface can be implemented using keyword **"implements"**.
8) A Java abstract class can have class members like **private, protected, etc.**	Members of a Java interface are **public by default.**
9)**Example:-**public abstract class Shape{	9)**Example:-**public interface Drawable{
public abstract void draw();}	void draw();}

Java 8 Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method Example in java 8 Default method:

Save File: TestInterfaceDefault.java

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Output:

```
drawing rectangle
default method
```

Java 8 Static Method in Interface

Since Java 8, we can have static method in interface.
Let's see an example:

File: TestInterfaceStatic.java

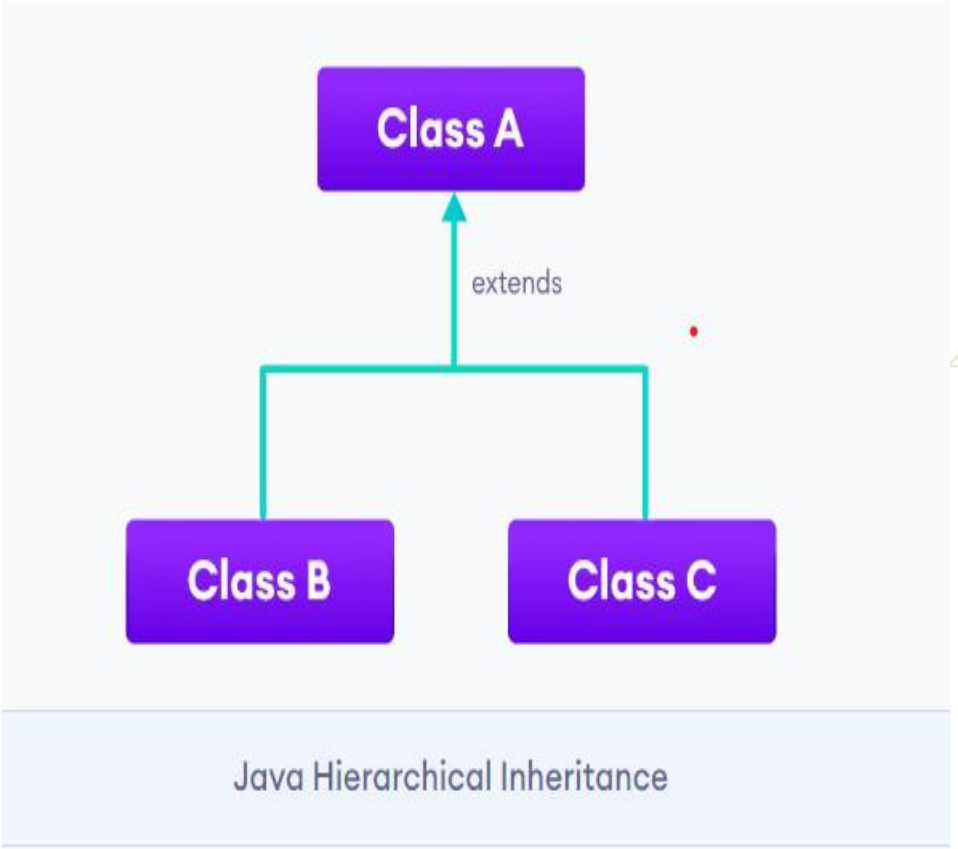
```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Output:

```
drawing rectangle
27
```

Hierarchical Inheritance in Java

- In hierarchical inheritance, multiple subclasses extend from a single superclass.
- For example, consider a parent class Car. Now, consider child classes Audi, BMW and Mercedes. In Hierarchical Inheritance in Java, class Audi, class BMW and class Mercedes, all these three extend class Car.



Java Hierarchical Inheritance

Hierarchical Inheritance


```
public class ClassH1
{
   public void dispH1()
   {  System.out.println("disp() method of
ClassH1"); }}
public class ClassH2 extends ClassH1
{  public void dispH2()
   {   System.out.println("disp() method of
ClassH2");   }}
public class ClassH3 extends ClassH1
{public void dispH3()
   {System.out.println("disp() method of
ClassH3");   }} }}
public class ClassH4 extends ClassH1
{public void dispH4()
   {System.out.println("disp() method of
ClassH4");   }}
      }}
```

```
public class HierarchicalInheritanceTest
{
   public static void main(String args[])
   { //Assigning ClassH2 object to ClassH2 reference
      ClassH2 h2 = new ClassH2();
      //call dispH2() method of ClassH2
      h2.dispH2();
      //call dispH1() method of ClassH1
      h2.dispH1();//Assigning ClassH3 object to ClassH3 reference
      ClassH3 h3 = new ClassH3();
      //call dispH3() method of ClassH3
      h3.dispH3();
      //call dispH1() method of ClassH1
      h3.dispH1() //Assigning ClassH4 object to ClassH4 reference
      ClassH4 h4 = new ClassH4();
      //call dispH4() method of ClassH4
      h4.dispH4();
      //call dispH1() method of ClassH1
      h4.dispH1();   }}
```

Output:

```
disp() method of ClassH2
disp() method of ClassH1
disp() method of ClassH3
disp() method of ClassH1
disp() method of ClassH4
disp() method of ClassH1
```

Super keyword in Java

Super keyword usage in inheritance, always refers to its immediate as an object.

There are three usages of super keyword in Java:

1. We can invoke the superclass variables.

2. We can invoke the superclass methods.

3. We can invoke the superclass constructor.

Example for Super Keyword in Java:

```
class Superclass
{  int i =20;
void display()
{ System.out.println("Superclass display method"); } }
class Subclass extends Superclass
{ int i = 100;
void display()
 {  super.display();
 System.out.println("Subclass display method");
 System.out.println(" i value ="+i);
 System.out.println("superclass i value ="+super.i);
 } }
class SuperUse{ public static void main(String args[])
{ Subclass obj = new Subclass();
 obj.display();
} }
```