Abstraction,Abstract class,Abstract methods,Interface

-K.L.Madhavi

Abstraction

- **Abstraction in Java** is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user. abstraction is achieved by **interfaces** and **abstract classes**. We can achieve 100% abstraction using interfaces.
- Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Algorithm to implement abstraction in Java

1. Determine the classes or interfaces that will be part of the abstraction.
2. Create an abstract class or interface that defines the common behaviors and properties of these classes.
3. Define abstract methods within the abstract class or interface that do not have any implementation details.
4. Implement concrete classes that extend the abstract class or implement the interface.
5. Override the abstract methods in the concrete classes to provide their specific implementations.
6. Use the concrete classes to contain the program logic.

Java Abstract classes and Java Abstract methods

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A abstract method must always be redefined in the subclass, thus making overriding compulsory or making the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an abstract keyword.
6. There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the *new operator*.
7. An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

Continue…

The abstract keyword is a non-access modifier,used for classes and methods:

1.**Abstract class:**is a restricted class that cannot be used to create objects(to access it,it must be inherited from another class.

2.**Abstract method**:can only be used in a abstract class,and it does not have a body.The body is provided by the subclass.

An abstract class can have both abstract and regular methods:

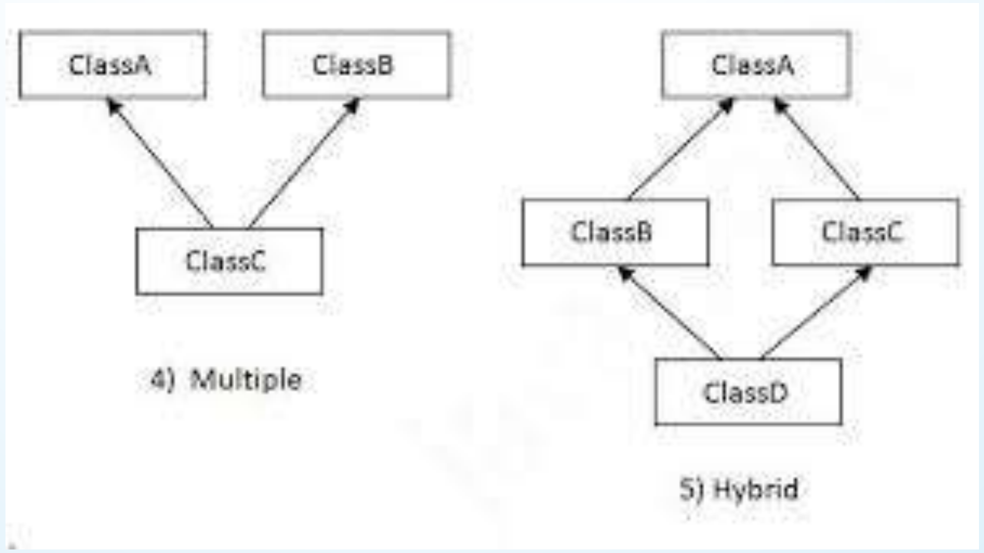
```
abstract class Animal {
    public abstract void animalSound();
    public void sleep()
   {
       System.out.println(“animal sound");
   }
}
```

```
// Abstract class
abstract class Animal
{
       // Abstract method (does not have a body)
       public abstract void animalSound();
      // Regular method
       public void sleep()
      {
           System.out.println(“sleeping");
       }
 }
// Subclass (inherit from Animal)
class Cat extends Animal
{
        public void animalSound()
       {
         // The body of animalSound() is provided here
          System.out.println(“The cat sounds”);
        }
}
Class Main
{
       public static void main(String args[])
       {
             Cat obj=new Cat();
             obj.animalSound();
             obj.sleep();
        }
}
```

Interface

Interfaces are another method of implementing abstraction in Java. The key difference is that, by using interfaces, we can achieve 100% abstraction in Java classes. In Java or any other language, interfaces include both methods and variables but lack a method body. Apart from abstraction, interfaces can also be used to implement inheritance in Java.It uses interface and implements keywords.

Through interface concept we can implement **multiple** and **hybrid inhertance** in java.



Multiple inheritance using interface

To achieve multiple inheritance we use interfaces.

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class.Only abstract methods can be used in interfaces.

Example Program:

```
interface Features
{
   void dialing();
   void messaging();
}
interface Addons
{
    void vcalling();
    void mms();
}
class SmartPhone implements Features,Addons
{
    public void dialing()
    {
        System.out.println("Smartphone is dialing…");
    }
    public void messaging()
    {
        System.out.println("Smartphone is messaging…");
    }
    public void vcalling()
    {
        System.out.println("Smartphone is in video call…");
    }
    public void mms()
    {
        System.out.println("Smartphone is sending mms…");
    }
}
```

```
Class Mobile
{
    public static void main(String args[])
    {
        SmartPhone SP=new SmartPhone;
        SP.dialing();
        SP.messaging();
        SP.vcalling();
        SP.mms();
     }
}
```

Hybrid interitance using interface

In general, the meaning of hybrid (mixture) is made of more than one thing. In Java, the hybrid inheritance is the composition of two or more types of inheritance. The main purpose of using hybrid inheritance is to modularize the code into well-defined classes. It also provides the code reusability.It can be achieved by using the following combinations:

- **Single and Multiple Inheritance (not supported but can be achieved through interface)**
- Multilevel and Hierarchical Inheritance
- Hierarchical and Single Inheritance
- Multiple and Multilevel Inheritance

```
class HumanBody
{
    public void displayHuman()
    {
        System.out.println("Method defined inside HumanBody class");
     }
}
interface Male
{
   public void show();
}
interface Female
{
   public void show();
}
public class Child extends HumanBody implements Male, Female
{
    public void show()
    {
        System.out.println("Implementation of show() method defined in interfaces Male and Female");
     }
     public void displayChild()
     {
        System.out.println("Method defined inside Child class");
     }
}
class Main{
     public static void main(String args[])
    {
        Child obj = new Child();
        System.out.println("Implementation of Hybrid Inheritance in Java");
        obj.show();
        obj.displayChild();
     }
}
```