OOPs Concepts in JAVA

One Day Workshop On Object oriented Design at Jamia Millia Islamia, Delhi

Conducted by

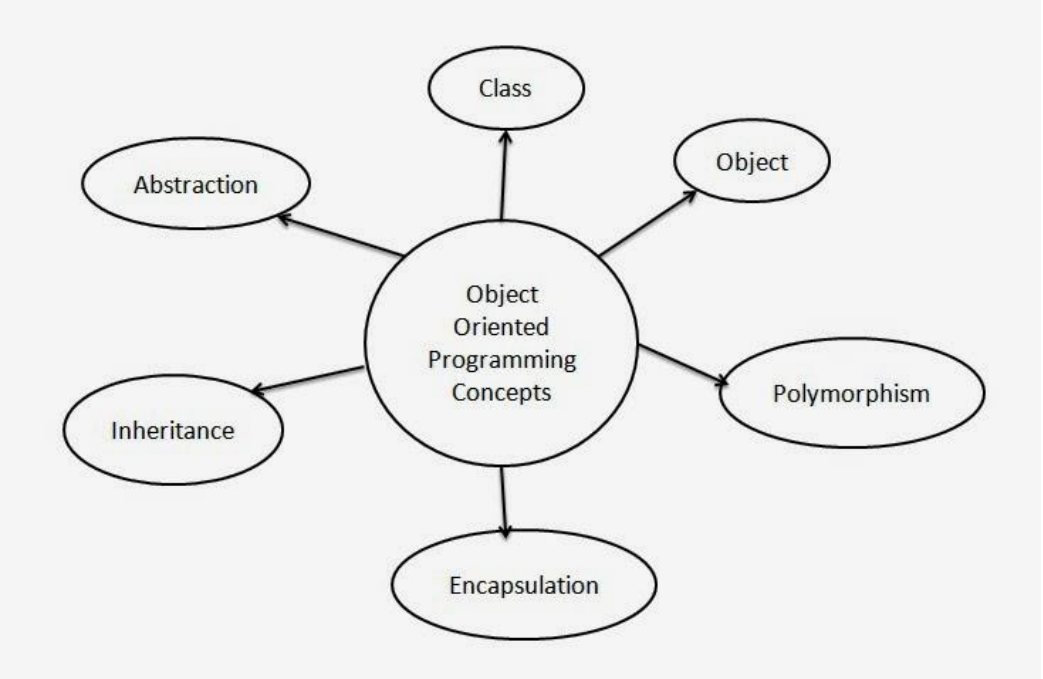
Adil Raja (SDE-1),Genpact.pvt.ltd

Md Kashif Alam(SDE-1,Digital),TCS

OOPs (Object Oriented Programming System)

Java is an Object-Oriented Language. **Object Oriented Programming** is a methodology to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction



Object and Class in JAVA

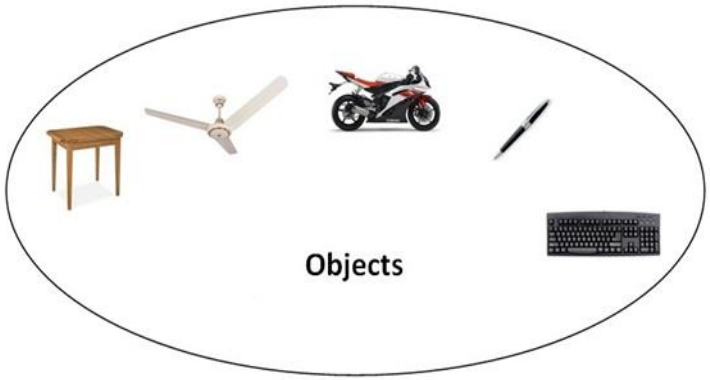
In this chapter, we will look into the concepts - Classes and Objects.

Object − Objects have states and behaviours.

If we consider a Student ,

its state is - Name, Roll No

its behaviour is - SetName, SetRollNo,



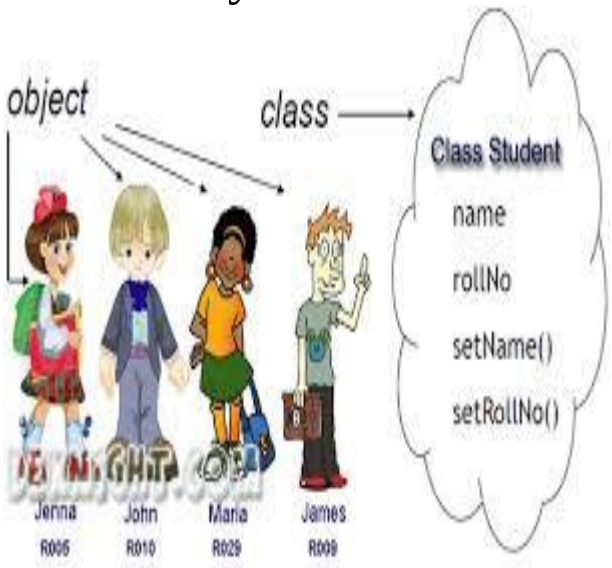
If you compare the software object with a real-world object, they have very similar characteristics. Software objects also have a state and a behaviour. A software object's state is stored in fields(DATA) and behaviour is shown via methods.

Object and Class in JAVA

Class − A *class* is a template or blueprint from which objects are created. A class can be defined as a template that describes the behaviour/state that the object of its type support. It is a group of objects that has common properties.

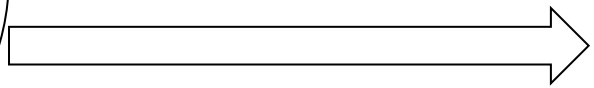
You declare a class by specifying the class keyword followed by a non-reserved identifier that names it. A pair of matching open and close brace characters ({ and }) follow and delimit the class's body.

```
class <class_name>{
    data member;
    method;
}
```

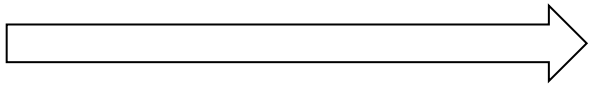


Object and Class in JAVA

```
public class Student {
String name;
int rollno;
void SetName(String nm)
{
name = nm;
}
void SetRollNo(int rno)
{
rollno = rno;
}
void DisplayInfo()
{
System.out.println(name +" is student with Roll number "+rollno);
}
public static void main(String[] args) {
// TODO Auto-generated method stub
Student s1 =new Student();
Student s2 =new Student();
s1.SetName("John");
s2.SetName("Maria");
s1.SetRollNo(1002);
s2.SetRollNo(1005);
s1.DisplayInfo();
s2.DisplayInfo();
}

}
```



class



object

Java Package in JAVA

A **java package** is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.

Access modifiers in JAVA

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:

- private
- default
- protected
- Public

1) **private access modifier**
The private access modifier is accessible only within class.

2) **default access modifier**
If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.

3) **protected access modifier**
The protected access modifier is accessible within package and outside the package but through inheritance only. The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

4) **public access modifier**
The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Constructor in JAVA

- Constructor in java is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor

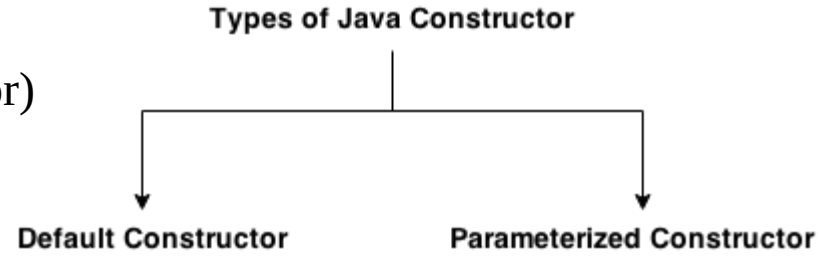
Rules for creating java constructor

There are basically two rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

There are two types of constructors:

- Default constructor (no-arg constructor)
- Parameterized constructor



Constructor in JAVA

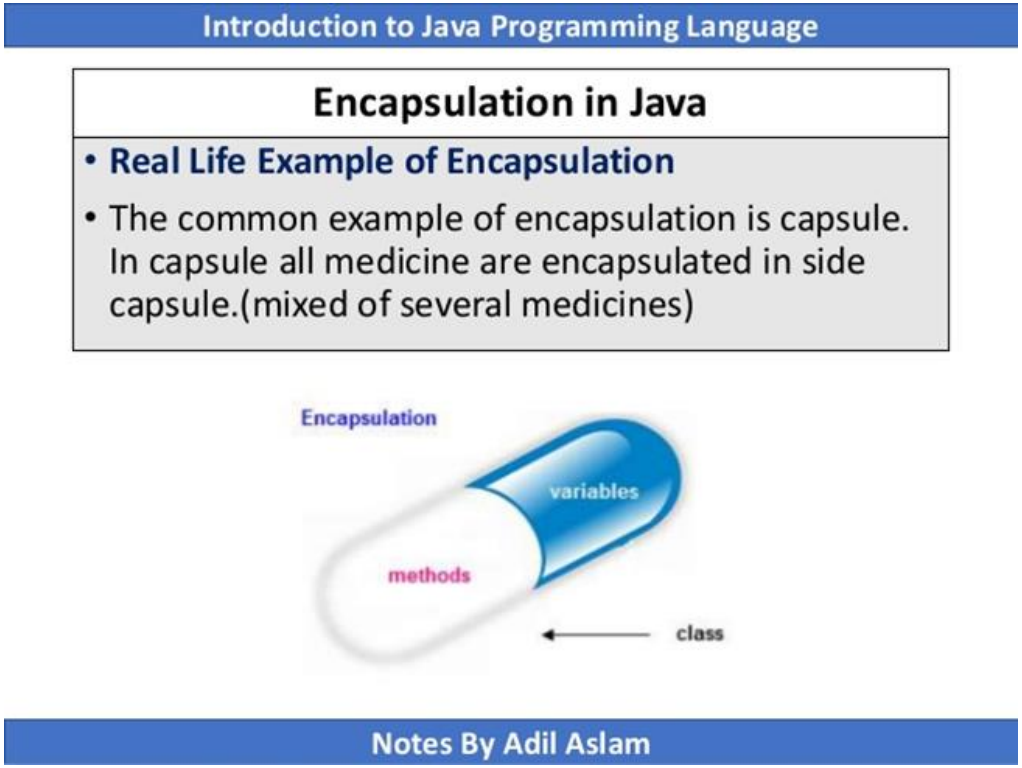
Default Constructor

```
package classobject;

public class Constructors {
String name;
public Constructors() {
System.out.println(" This is default constructor");
  }

  public Constructors(String nm) {
    // This constructor has one parameter, name.
   name = nm;
   System.out.println(" This is constructor with Parameter Name "+name);
  }
public static void main(String[] args) {
// TODO Auto-generated method stu
Constructors c1 = new Constructors();
Constructors c2 = new Constructors("Anja
 }
}
```

Parameterised Constructor

Encapsulation in JAVA

Encapsulation is one of the four fundamental OOP concepts. **Encapsulation in java** is a *process of wrapping code and data together into a single unit*, for example capsule i.e. mixed of several medicines. The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class.

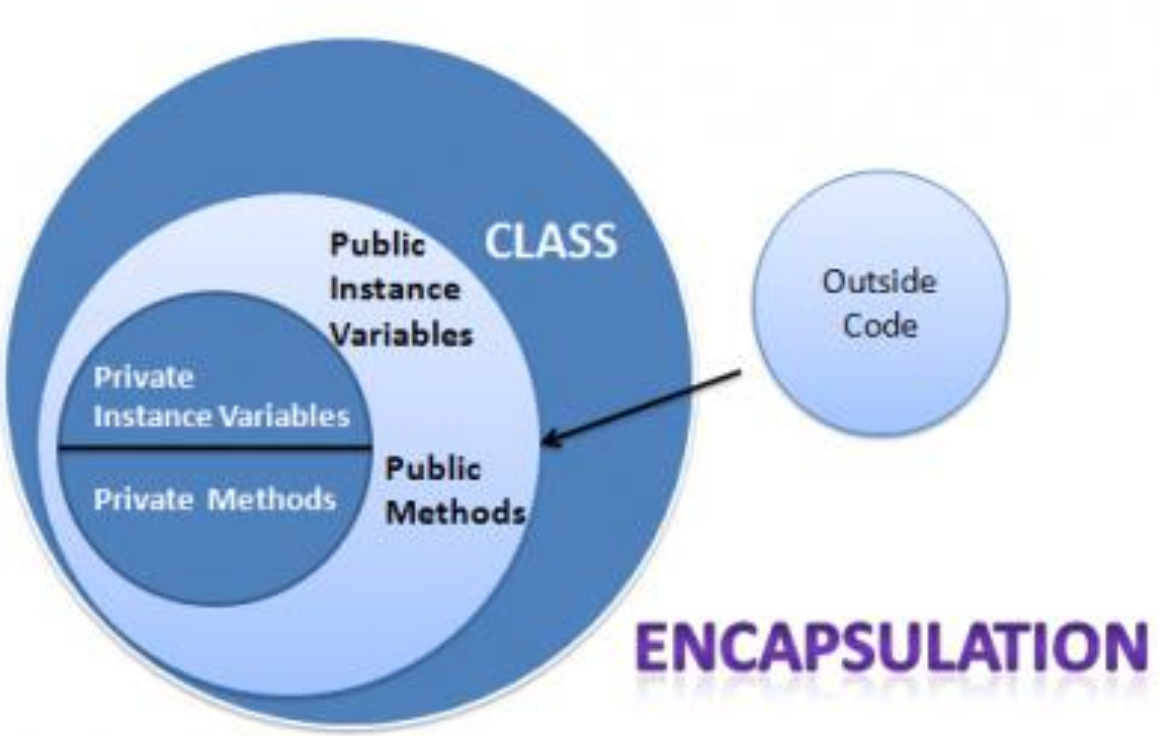


Encapsulation in JAVA

To achieve encapsulation in Java −

•Declare the variables of a class as private.

•Provide public setter and getter methods to modify and view the variables values.

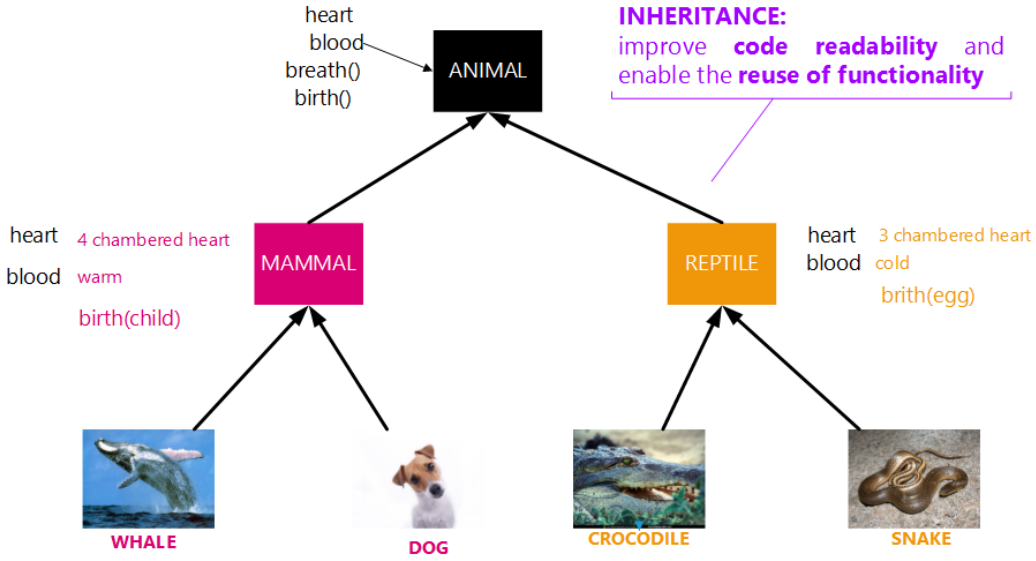


Encapsulation in JAVA

```
public class EncapTest {
   private String name;
   public String getName() {
      return name;
   }
   public void setName(String newName) {
      name = newName;
   }

public static void main(String[] args) {
EncapTest encap = new EncapTest();
    encap.setName("James");
    System.out.println("Entered Name is "+encap.getName());
}
}
```

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviours of parent object.
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 parent class, and you can add new methods and fields also. Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.



Inheritance in Java

Why use inheritance in java?

- Reusability -- facility to use public methods of base class without rewriting the same
- Extensibility -- extending the base class logic as per business logic of the derived class
- Data hiding -- base class can decide to keep some data private so that it cannot be altered by the derived class
- Overriding--With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Inheritance in Java

Syntax of Java Inheritance

class Subclass-name **extends** Superclass-name

```
{
  //methods and fields
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

```
public class Animal {
}
public class Mammal extends Animal {
}
public class Reptile extends Animal {
}
public class Dog extends Mammal {
}
```

Inheritance in Java

```
package Inheritance;
class Animal {
public void heart() {
        System.out.println("We have heart");
    }}
class Mammal extends Animal {
 public void MammaryGlands() {
        System.out.println("We have Mammary glands!");
    }}
 class Reptile extends Animal {
 public void Hiss() {
        System.out.println("Hiss Hiss!!");
    }}
public class dog extends Mammal {
 public void bark() {
        System.out.println("Dogs bark Bow Bow!");
    }
public static void main(String[] args) {
// TODO Auto-generated method stub
dog d = new dog();
d.bark();
d.MammaryGlands();
d.heart();
}}
```

super keyword in Java

```
package classobject;
 class Bike4 {
 int speed=50;
}
class superkwrd extends Bike4{
  int speed=100;
  void display(){
   System.out.println(super.speed);//will print speed of Vehicle now
  }
  public static void main(String args[]){
  superkwrd b=new superkwrd();
  b.display();
  }
}
```

super keyword in Java

- The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of java super Keyword

- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.

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:

- variable
- method
- Class

1) Java final variable

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

2) Java final method

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

3) Java final class

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

this keyword in Java

In java, ‘this’ is a **reference variable** that refers to the current object.

Usage of java this keyword
Here is given the 6 usage of java this keyword.

- this keyword can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- this keyword can be used to invoke current class method (implicitly)
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this keyword can also be used to return the current class instance.

this keyword in Java

```
public class thiskwrd {
  int id;
  String name;
  thiskwrd(int id,String name){
  this.id = id;
  this.name = name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    thiskwrd s1 = new thiskwrd(111,"Karan");
    thiskwrd s2 = new thiskwrd(222,"Aryan");
    s1.display();
    s2.display();
 }
}
```

Polymorphism in JAVA?

- Polymorphism in java is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

- There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Compile-time Polymorphism	Run-time Polymorphism
Is implemented through method **overloading**.	Is implemented through method **overriding**.
Is executed at the compile-time since the compiler knows which method to execute depending on the number of parameters and their data types.	Is executed at run-time since the compiler does not know the method to be executed, whether it is the base class method that will be called or the derived class method.
Is referred to as **static** polymorphism.	Is referred to as **dynamic** polymorphism.

Method Overloading in JAVA?

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading.**
Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b (int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

Method Overloading in JAVA?

Example of Method Overloading by changing the no. of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
class Calculation{
    void sum(int a,int b){System.out.println(a+b);}
    void sum(int a,int b,int c){System.out.println(a+b+c);}
    public static void main(String args[]){
    Calculation obj=new Calculation();
    obj.sum(10,10,10);
    obj.sum(20,20);
      }
    }
```

Method Overloading in JAVA?

Example of Method Overloading by changing data type of argument

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments**.**

```
class Calculation2{
  void sum(int a,int b){System.out.println(a+b);}
  void sum(double a,double b){System.out.println(a+b);}

  public static void main(String args[]){
  Calculation2 obj=new Calculation2();
  obj.sum(10.5,10.5);
  obj.sum(20,20);
  }
}
```

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.

Rules for Java Method Overriding

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).

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.

Rules for Java Method Overriding

- method must have same name as in the parent class
- method must have same parameter as in the parent class.
- must be IS-A relationship (inheritance).

Method Overriding in JAVA?

Example of method overriding
In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}

class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Abstraction in JAVA?

As per dictionary, abstraction is the quality of dealing with ideas rather than events. Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.

In other words, the user will have the information on what the object does instead of how it does it.

In Java, abstraction is achieved using Abstract classes and interfaces.

Abstraction in JAVA?

Abstract Class
A class which contains the abstract keyword in its declaration is known as abstract class.

Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)

But, if a class has at least one abstract method, then the class must be declared abstract.
If a class is declared abstract, it cannot be instantiated.

To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.

If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstraction in JAVA?

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```