Java Abstraction

By Umesh Sir

Contact us 7758094241

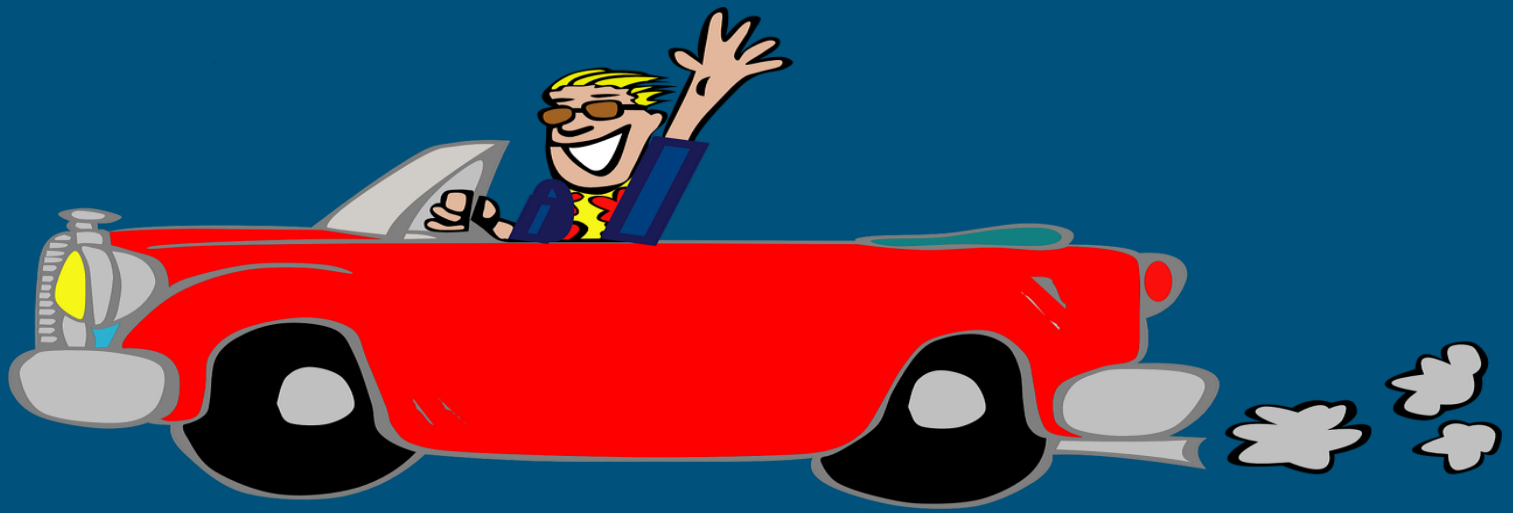
Abstraction

- ★ What is mean by Abstraction?
- ★ Abstract Class
- ★ Usage of abstract classes in Real-world Examples.
- ★ Interface
- ★ Why use Java interface?

★ What is mean by Abstraction?

- Abstraction means hiding lower-level details and exposing only the essential and relevant details to the users.

Real-world examples :- Let's consider a Car, which abstracts the internal details and exposes to the driver only those details that are relevant to the interaction of the driver with the Car.



Abstraction Real World Example

- Example 2: Consider an ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know the internal details about ATM.

- Abstraction exposes to the user only those things that are relevant to them and hides the remainder of the details. In OOP terms, we say that an object should expose to the users only a set of high-level; operations, while the internal implementation of those operations is hidden

- In Java, the abstraction is achieved by Interfaces and Abstract classes. We can achieve 100% abstraction using Interfaces And (0 to 100% using Abstract Class).

- Abstract Class

- Interface

- abstarct Methods - it ends with semi colon. (void show(); and int add(int val1, int val2);)

Interface

An interface in Java is a blueprint of a class. It has 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 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.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have default and static methods in an interface.

Since Java 9, we can have private methods in an interface.

★ Why use Java interface?

1] Inteface helps us to decouple the code like we can change the implimentation any time from one class to another class.

```
inteface Language{

                String getLanguage();

}

Class Marathi impl Language{

String getLanguage(){   }

}
```

1] Why Multiple Inheritance supported in case of Interface and not supported in case of class?

2] Relationship between Class and Interface (Interface -> Implements -> Class .... Inteface -> extends -> Inteface .... Class -> extends -> class)

Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

File: TestInterface1.java

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

An interface is a contract (or a protocol, or a common understanding) of what the classes can do. When a class implements a certain interface, it promises to provide implementation to all the abstract methods declared in the interface. Interface defines a set of common behaviors. The classes implement the interface agree to these behaviors and provide their own implementation to the behaviors. This allows you to program at the interface, instead of the actual implementation. One of the main usage of interface is provide a communication contract between two objects. If you know a class implements an interface, then you know that class contains concrete implementations of the methods declared in that interface, and you are guaranteed to be able to invoke these methods safely. In other words, two objects can communicate based on the contract defined in the interface, instead of their specific implementation.

Abstract Class

Java Abstract Class Basics

1. Abstract Class

An abstract class is a class that is declared abstract means it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

Abstract Class Definition Rules:

1.1] Abstract classes cannot be instantiated directly.

1.2] Abstract classes may be defined with any number, including zero, of abstract and non-abstract methods.

1.3] Abstract classes may not be marked as private or final.

1.4] An abstract class that extends another abstract class inherits all of its abstract methods as its own abstract methods.

1.5] The first concrete class that extends an abstract class must provide an implementation for all of the inherited abstract methods.

```java
public class AbstractClassCompleteExample {

    public static void main(String[] args) {
        Animals animals = new Cat();
        animals.sound();

        animals = new Dog();
        animals.sound();
    }
}

abstract class Animals{
    private String name;

    // All kind of animals eat food so make this common to all animals
    public void eat(){
        System.out.println(" Eating ..........");
    }

    // The aninals make different sounds. They will provide their own implementation
    abstract void sound();
}
```

```java
class Cat extends Animals{

    @Override
    void sound() {
        System.out.println("Meoww Meoww ........");
    }
}

class Dog extends Animals {

    @Override
    void sound() {
         System.out.println("Woof Woof ........");
    }
}
```

2. Abstract Method

An abstract method is a method that is declared without an implementation.

Abstract Method Definition Rules:

2.1] Abstract methods may only be defined in abstract classes.

2.2] Abstract methods may not be declared private or final.

2.3] Abstract methods must not provide a method body/implementation in the abstract class for which is it declared.

2.4] Implementing an abstract method in a subclass follows the same rules for overriding a method.

Example: The below `AbstractShape` class contains two abstract methods - `draw()` and `moveTo()`;

```
abstract class AbstractShape{
    abstract void draw();
    abstract void moveTo(double deltaX, double deltaY);
}
```

3. If a class includes abstract methods, then the class itself must be declared abstract

Example:

```
public abstract class AbstractClassExample {
   // declare fields
   // declare nonabstract methods
   abstract void abstractMethod();
}
```

4. When an abstract class is subclassed

The subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

For example :

```
abstract class AbstractShape{
    abstract void draw();
}

//The abstract method draw in type Shape can only be defined by an abstract class
abstract class Shape extends AbstractShape{

    @Override
    abstract void draw();

    /*@Override
    void draw() {
        // TODO Auto-generated method stub

    }*/
}
```

6. When an Abstract Class Implements an Interface

When we create the Java class that implements an interface must implement all of the interface's methods. It is possible, however, to define a class that does not implement all of the interface's methods, provided that the class is declared to be abstract.

For example:

```
interface A {
   void a();

   void b();

   void c();

   void d();
}

// The abstract class can also be used to provide some implementation of the interface.
//In such case, the end user may not be forced to override all the methods of the interface.
abstract class B implements A {
    public void c() {
        System.out.println("I am c");
    }
}
```

```
class M extends B {
    public void a() {
        System.out.println("I am a");
    }

    public void b() {
        System.out.println("I am b");
    }

    public void d() {
        System.out.println("I am d");
    }
}

class Test5 {
    public static void main(String args[]) {
        final A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
    }
}
```

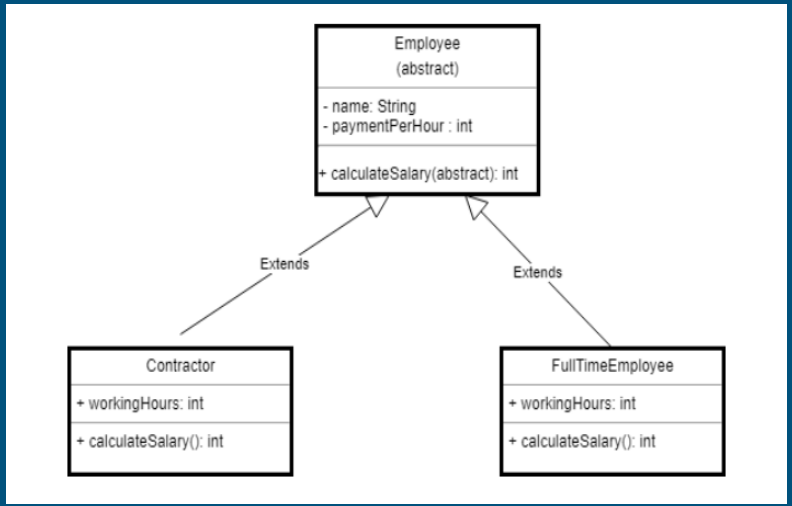
★ Usage of abstract classes in Real-world Examples

It can have both concrete method and abstract method. We can't create object of Abstract Class.

Example 1: Employee, Contractor, and FullTimeEmployee Example

In this example, we create an abstract Employee class and which contains the abstract calculateSalary() method. Let the subclasses extend the Employee class and implement a calculateSalary() method.

Let's create Contractor and FullTimeEmployee classes as we know that the salary structure for a contractor and full-time employees are different so let these classes override and implement a calculateSalary() method.



Step 1: Let's first create the superclass Employee. Note the usage of abstract keyword in this class definition. This marks the class to be abstract, which means it can not be instantiated directly. We define a method called calculateSalary() as an abstract method. This way you leave the implementation of this method to the inheritors of the Employee class.

Step 2: The Contractor class inherits all properties from its parent abstract Employee class but has to provide its own implementation to calculateSalary() method. In this case, we multiply the value of payment per hour with given working hours.

```
public abstract class Employee {

    private String name;
    private int paymentPerHour;

    public Employee(String name, int paymentPerHour) {
        this.name = name;
        this.paymentPerHour = paymentPerHour;
    }

    public abstract int calculateSalary();
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getPaymentPerHour() {
        return paymentPerHour;
    }
    public void setPaymentPerHour(int paymentPerHour) {
        this.paymentPerHour = paymentPerHour;
    }
}
```

```
public class Contractor extends Employee {


    private int workingHours;
    public Contractor(String name, int paymentPerHour, int workingHours) {
        super(name, paymentPerHour);
        this.workingHours = workingHours;
    }
    @Override
    public int calculateSalary() {
        return getPaymentPerHour() * workingHours;
    }
}
```

Step 3: The FullTimeEmployee also has its own implementation of calculateSalary() method. In this case, we just multiply by a constant value of 8 hours.

Step 4: Let's create an AbstractionDemo class to test implementation of Abstraction with the below code:

```
public class FullTimeEmployee extends Employee {
    public FullTimeEmployee(String name, int paymentPerHour) {
        super(name, paymentPerHour);
    }
    @Override
    public int calculateSalary() {
        return getPaymentPerHour() * 8;
    }
}
```

```
public class AbstractionDemo {

    public static void main(String[] args) {

        Employee contractor = new Contractor("contractor", 10, 10);
        Employee fullTimeEmployee = new FullTimeEmployee("full time employee", 8);
        System.out.println(contractor.calculateSalary());
        System.out.println(fullTimeEmployee.calculateSalary());
    }
}
```

Example 2: Drawing Shapes Example

- Consider the second example Shapes base type is “shape” and each shape has a color, size, and so on. From this, specific types of shapes are derived(inherited)-circle, square, triangle, and so on.
- The areas for these shapes are different so make the area() method abstract and let the subclasses override and implement.

```
abstract class Shape {
    String color;

    // these are abstract methods
    abstract double area();

    public abstract String toString();

    // abstract class can have constructor
    public Shape(String color) {
        System.out.println("Shape constructor called");
        this.color = color;
    }

    // this is a concrete method
    public String getColor() {
        return color;
    }
}
```

```
class Circle extends Shape {
    double radius;

    public Circle(String color, double radius) {

        // calling Shape constructor
        super(color);
        System.out.println("Circle constructor called");
        this.radius = radius;
    }

    @Override
    double area() {
        return Math.PI * Math.pow(radius, 2);
    }

    @Override
    public String toString() {
        return "Circle color is " + super.color + "and area is : " + area();
    }

}
```

```java
class Rectangle extends Shape {

    double length;
    double width;

    public Rectangle(String color, double length, double width) {
        // calling Shape constructor
        super(color);
        System.out.println("Rectangle constructor called");
        this.length = length;
        this.width = width;
    }

    @Override
    double area() {
        return length * width;
    }

    @Override
    public String toString() {
        return "Rectangle color is " + super.color + "and area is : " + area();
    }

}
```

```java
public class AbstractionTest {
    public static void main(String[] args) {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

Summary :-

Let's summarize what we have discussed in this article:

1] An abstract class is a class that is declared with abstract keyword.

2] An abstract method is a method that is declared without an implementation.

3] An abstract class may or may not have all abstract methods. Some of them can be concrete methods

4] A method defined abstract must always be redefined in the subclass, thus making overriding compulsory OR either make subclass itself abstract.

5] Any class that contains one or more abstract methods must also be declared with 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 default constructor is always present in an abstract class.

Thank you