Abstraction in OOP

When we generally talk about abstraction the software language is an example of Abstraction. Let's take an example and write a statement as-

$x = y + z;$

In the above statement, we are adding two variables that are stored in two different locations and then storing the result in a new location. So, what happens next? As you might know, there are registers, instruction sets, program counters, storage units, etc., involved. When we refer to abstraction in Java, we are talking about abstraction in object-oriented programming (OOP) and how it is achieved. The concept of abstraction in OOP starts right at the moment when a class is getting conceived. Abstraction is applied everywhere in software and OOP.

What is Java Abstraction?

Abstraction is nothing but the quality of dealing with ideas rather than events. It basically deals with hiding the internal details and showing the essential things to the user.

If you look at the above gif, you can see when you get a call, we get an option to either pick it up or just reject it. But in reality, there is a lot of code that runs in the background. So here, you don't know

the internal processing of how a call is generated, that's the beauty of abstraction. You can achieve abstraction in two ways:

a) Abstract Class

b) Interface

Let's understand these concepts in more detail.

Abstract Class

Abstract class contains the 'abstract' keyword. But what exactly it means? If you make the class abstract, it cannot be instantiated, i.e you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.

Note: *You can achieve 0–100% abstraction using an abstract class.*

To use an abstract class, you have to inherit it from the base class. Here, you have to provide implementations for the abstract methods, else it will become an abstract class.

Let's look at the syntax of an abstract class:

```
Abstract class Sports {   // abstract class sports

Abstract void jump();    // abstract method

}
```

Interface

An interface in Java is a collection of abstract methods and static constants. As you might know in an interface, each method is public and abstract but it does not contain any constructor. Along with abstraction, the interface also helps to achieve multiple inheritance in Java.

Note: *You can achieve 100% abstraction using interfaces.*

Basically, Interface is a group of related methods with empty bodies. Let us understand interfaces by taking an example of a Shape' interface with its related methods.

```
public interface shape{

public void draw();

public double getArea();

}
```

These methods need to be present in every 'shape', right? But their working is going to be different.

Let's assume that you want to draw a shape, say circle, square, rectangle etc. You already know, each shape possess its own dimensions like radius, height, and width. Say I want to draw a circle and calculate its area. Considering the same, I have created two methods in the above code i.e. draw() and getArea(). Now,

using these methods, I can draw any shape and calculate the area by implementing its interface.

Now, let's look into the functionality as to how you can implement this interface.

In order to implement this interface, the name of your class would change to any of the Shape, let's say "Circle". So, to implement the class interface, I will make use of 'implement' keyword:

```
public class Circle implements Shape{

private double radius;

public Circle(double r){

this.radius = r;

}

void draw(){
```

```
System.out.println("Drawing Circle");

}

public double getArea(){

return Math.PI*this.radius*this.radius;

}

public double getRadius(){

return this.radius;

}

}

public class Test{

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

```
Shape c = new Circle(8);

c.draw();

System.out.println("Area="+c.getArea());

}

}
```

In the above example, I have specified functionalities to the different methods and calculated area of a circle. Here, on implementing an interface, it allows a class to become more formal about the behavior that it provides. You can also create another class, say ‘Rectangle’ class which can inherit the same interface ‘shape’ with different functionalities.

Real-time Example of abstraction in java

Suppose we have Sport as an interface. Here, implementation will be provided by classes called “Badminton” and “Football”. In a real scenario, an end user will not be aware of the implementation class. Hence, an object of the implementation class can be provided by the factory method. Factory method can be used to create an object of implementation class based on some criterion.

Let’s implement the same and create an interface called Sport.java.

```
public Interface Sport{

void play();

}

//Now, we will create class named "Badminton"

public class Badminton implements Sport {

@Override
```

```
public void play() {

System.out.println("Playing badminton");

}

}

//Next let's create our last class “Football”

public class Football implements Sport {

@Override

public void play() {

System.out.println("Playing football");

}
```

The last step is to create a main class named “SportInterface”.

```
public SportInterface{

public static void main(String args[]){

// your code here

}

}
```

When you execute the above program, the output will be as shown below:

```
Playing badminton

--------------------

Playing football
```

I hope you guys are clear with the interface and how you can achieve abstraction using it. Now, let's conclude this article by comparing Abstraction and Encapsulation.

Abstraction vs Encapsulation

Abstraction	Encapsulation
Solves the problem in design level	Solves the problem in the implementation level
Used for hiding unwanted data and giving relevant results	Encapsulation means hiding the code and data into a single unit to protect data from the outside world
Outer layout – used in terms of design	Inner layout – used in terms of implementation

I hope you understood the difference between Abstraction and Encapsulation.