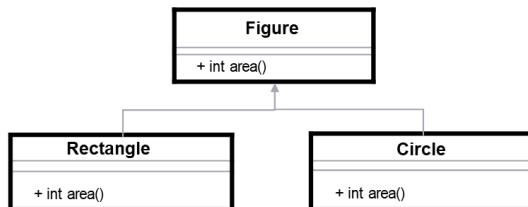


# Java Day5

## Abstract Classes

- Let us see the below example of Figure class extended by Rectangle and Circle.



- In the above example area() for Figure being more generic we cannot define it. At the level of rectangle or Circle we can give the formula for area.
- Often, you would want to define a superclass that declares the structure of a given abstraction without providing the implementation of every method
- The objective is to:
  - Create a superclass that only defines a generalized form that will be shared by all of its subclasses
  - leaving it to each subclass to provide for its own specific implementations
  - Such a class determines the nature of the methods that the subclasses **must implement**
  - Such a superclass is unable to create a meaningful implementation for a method or methods
- The class **Figure** in the previous example is such a superclass.
  - Figure is a pure geometrical abstraction
  - You have only kinds of figures like **Rectangle**, **Triangle** etc. which actually are subclasses of class **Figure**
  - The class **Figure** has no implementation for the **area( )** method, as there is no way to determine the area of a **Figure**
  - The **Figure** class is therefore a partially defined class with no implementation for the **area( )** method
  - The definition of **area()** is simply a placeholder
- The importance of abstract classes:**  
they define a generalized form (possibly some generalized methods with no implementations) *that will be shared by all of its subclasses*, so that *each subclass can provide specific implementations* of such methods.

- **abstract method** – It's a method declaration with no definition
- a mechanism which shall ensure that a subclass must compulsorily override such methods.
- Abstract method in a superclass has to be overridden by all its subclasses.
- The subclasses cannot make use of the abstract method that they inherit directly (without overriding these methods).
- These methods are sometimes referred to as subclasses' responsibility as they have no implementation specified in the superclass
  
- To use an abstract method, use this general form: **abstract type name(parameter-list);**
- Abstract methods do not have a body
- Abstract methods are therefore characterized by the lack of the opening and closing braces that is customary for any other normal method
- This is a crucial benchmark for identifying an abstract class
- area method of Figure class made Abstract.  

```
public abstract int area();
```

Any class that contains one or more abstract methods **must** also be declared abstract

- It is perfectly acceptable for an abstract class to implement a concrete method
- You cannot create objects of an abstract class
- That is, an abstract class cannot be instantiated with the new keyword
- Any subclass of an abstract class must *either implement all of the abstract methods in the superclass, or be itself declared abstract.*

## Revised Figure Class – using abstract

- There is no meaningful concept of area( ) for an undefined two-dimensional geometrical abstraction such as a Figure.
- The following version of the program declares area( ) as abstract inside class Figure.
- This implies that class Figure be declared abstract, and all subclasses derived from class Figure must override area( ).

## Improved Version of the Figure Class Hierarchy

```
abstract class Figure{  
    double dimension1;  
    double dimension2;  
    Figure(double x, double y){  
        dimension1 = x;  
        dimension2 = y;  
    }  
    abstract double area();  
}
```

## Improved Version of the Figure Class Hierarchy (Contd.).

```
class Rectangle extends Figure{  
    Rectangle(double x, double y){  
        super(x,y);  
    }  
  
    double area(){  
        System.out.print("Area of rectangle is :");  
        return dimension1 * dimension2;  
    }  
}  
  
class Triangle extends Figure{  
    Triangle(double x, double y){  
        super(x,y);  
    }  
  
    double area(){  
        System.out.print("Area for triangle is :");  
        return dimension1 * dimension2 / 2;  
    }  
}
```

## Improved Version of the Figure Class Hierarchy (Contd.).

```
class FindArea{  
    public static void main(String args[]){  
        Figure fig;  
        Rectangle r = new Rectangle(9,5);  
        Triangle t = new Triangle(10,8);  
        fig = r;  
        System.out.println("Area of rectangle is :" + fig.area());  
        fig = t;  
        System.out.println("Area of triangle is :" + fig.area());  
    }  
}
```

## **Abstraction**

```
package com.Day5.Abstraction;

abstract class Shape{
    String color;
    abstract void draw();
    public void getColor(String color){
        this.color=color;
    }
}
class Rectangle extends Shape{
    void draw(){
        System.out.println("Drawing Rectangle...");
    }
}
class Square extends Shape{
    void draw(){
        System.out.println("Drawing Square....");
    }
}

public class AbstractionDemo {
    public static void main(String[] args) {
        Shape rectangle=new Rectangle();
        rectangle.draw();

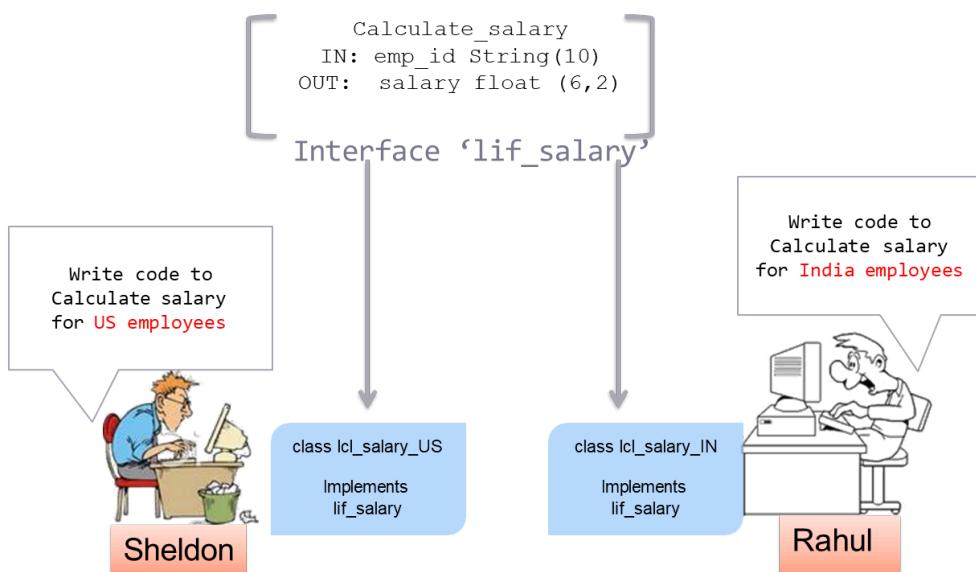
    }
}
```

## What is an Interface?

An interface is a named collection of method declarations (without implementations)

- An interface can also include constant declarations
  - An interface is syntactically similar to an abstract class
  - An interface is a collection of abstract methods and final variables
  - A class implements an interface using the **implements** clause
- 
- An interface defines a protocol of behavior
  - An interface lays the specification of what a class is supposed to do
  - How the behavior is implemented is the responsibility of each implementing class
  - Any class that implements an interface adheres to the protocol defined by the interface, and in the process, implements the specification laid down by the interface

## Interface: Example



## Why interfaces are required ?

- Interfaces allow you to implement common behaviors in different classes that are not related to each other
- Interfaces are used to describe behaviors that are not specific to any particular kind of object, but common to several kinds of objects
- Defining an interface has the advantage that an interface definition stands apart from any class or class hierarchy
- This makes it possible for any number of independent classes to implement the interface
- Thus, an interface is a means of specifying a consistent specification, the implementation of which can be different across many independent and unrelated classes to suit the respective needs of such classes
- Interfaces reduce coupling between components in your software
- Java does not support multiple inheritance
- This is a constraint in class design, as a class cannot achieve the functionality of two or more classes at a time
- Interfaces help us make up for this loss as a class can implement more than one interface at a time
- Thus, interfaces enable you to create richer classes and at the same time the classes need not be related

## Interface members

- All the methods that are declared within an interface are always, by default, *public* and *abstract*
- Any variable declared within an interface is always, by default, *public static* and *final*

## Interface

```
package com.Day5.Abstraction;  
interface Payable{  
    void generatePayslip();  
}  
interface Bonus{
```

```
void calculateBonus();
}

class Contractor implements Payable,Bonus{

    @Override
    public void generatePayslip() {
        System.out.println("Contractor salary will be calculated on day basis");
    }

    @Override
    public void calculateBonus() {

    }
}

class FullTimeEmployee implements Payable{
    public void generatePayslip(){
        System.out.println("Full time employees salary will be calculated on monthly basis");
    }
}

public class InterfaceDemo {
    public static void main(String[] args) {
        Payable contractor=new Contractor();
        contractor.generatePayslip();
        Payable FTE=new FullTimeEmployee();
        FTE.generatePayslip();
    }
}
```