Abstract classes

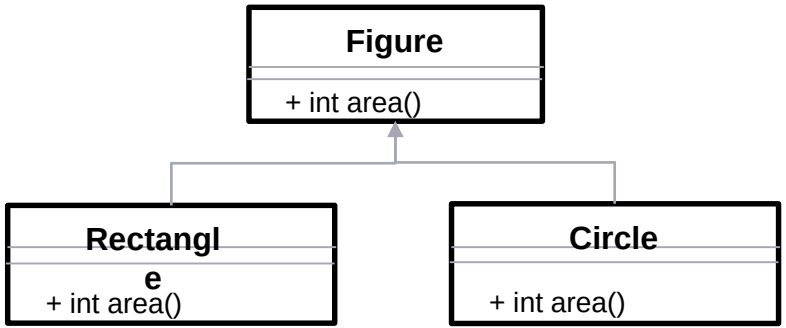
Agenda

1 **Abstract Classes**

Abstract classes

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.

Abstract Classes (Contd.).

- 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

Abstract Classes (Contd.).

- 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 Classes (Contd.).

- **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

Abstract Classes (Contd.).

- 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();
```

Abstract Classes (Contd.).

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());
  }
}
```

Quiz

What will be the output for the below code ?

```
class Gbase{
public abstract void testBase();
}
public class Sample extends GBase{
   public static void main() {
    Sample ob = new Sample();
    ob.testBase();
   }
}
```

Quiz(Contd.).

What will be the output for the below code ?

```
class abstract GBase{
public void testBase(){
System.out.println("Hello World");
}
}
public class Sample extends GBase{
   public static void main() {
    GBase ob = new GBase();
    ob.testBase();
   }
}
```


Assignment

Thank You