Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body). Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

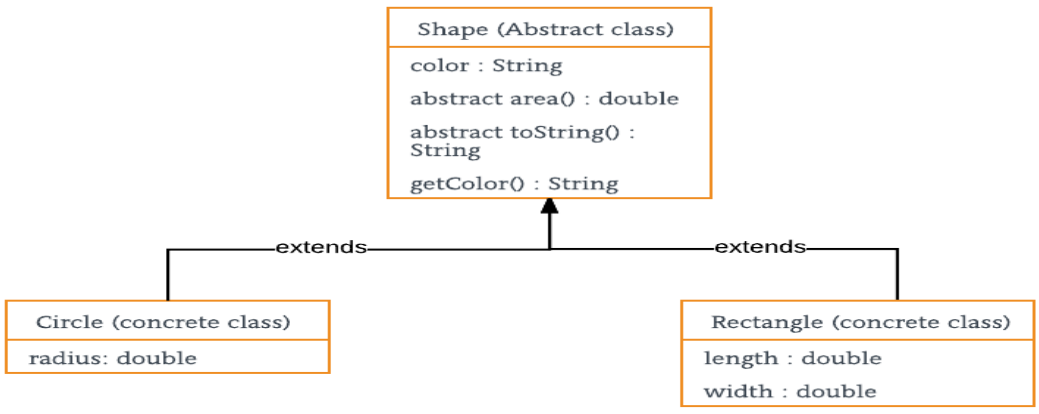
Abstract class (0 to 100%)

Interface (100%)

When to use abstract classes and abstract methods with an example

There are situations in which we will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes we will want to create a superclass that only defines a generalization form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.

Consider a classic “shape” example, perhaps used in a computer-aided design system or game simulation. The 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 – each of which may have additional characteristics and behaviors. For example, certain shapes can be flipped. Some behaviors may be different, such as when you want to calculate the area of a shape. The type hierarchy embodies both the similarities and differences between the shapes.



```
// concept of Abstraction
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();
    }
}
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();
    }
}
public class Test
{
    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());
    }
}
```

Output:

```
Shape constructor called
Circle constructor called
```

Shape constructor called

Rectangle constructor called

Circle color is Redand area is : 15.205308443374602

Rectangle color is Yellowand area is : 8.0

Advantages of Abstraction

It reduces the complexity of viewing the things.

Avoids code duplication and increases reusability.

Helps to increase security of an application or program as only important details are provided to the user.