Abstraction in Java

Difficulty Level : Medium Last Updated : 17 Sep, 2021

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

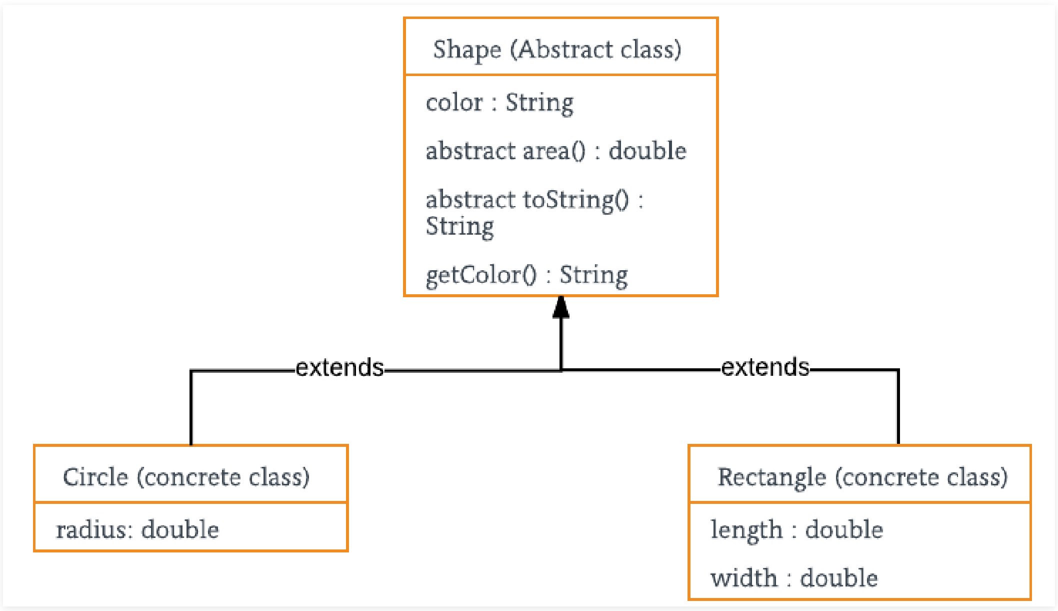
Abstract classes and Abstract methods :

1. An abstract class is a class that is declared with an abstract keyword.
2. An abstract method is a method that is declared without 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 the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an 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 the default constructor is always present in an abstract class.

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.



```java
// Java program to illustrate the
// concept of Abstraction
abstract class Shape {
    String color;

    // these are abstract methods
    abstract double area();
    public abstract String toString();

    // abstract class can have the 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.getColor()
            + "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.getColor()
            + "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
```

Encapsulation vs Data Abstraction

1. Encapsulation is data hiding(information hiding) while Abstraction is detailed hiding(implementation hiding).
2. While encapsulation groups together data and methods that act upon the data, data abstraction deal with exposing the interface to the user and hiding the details of implementation.

Advantages of Abstraction

1. It reduces the complexity of viewing the things.
2. Avoids code duplication and increases reusability.
3. Helps to increase the security of an application or program as only important details are provided to the user.

Related articles :

- Interfaces in java
- Abstract classes in java
- Difference between abstract class and interface
- abstract keyword in java