Website

https://www.curiousinfotechsolutions.com



# Abstraction in Java

# Java Programming

## Day-13

Email id:
**info@curiousinfotechsolutions.com**

# Syllabus Day-13



- **Abstraction in Java**
  - **Allow us to create a general idea of what the problem is and how to solve it.**
- How can implement Data Abstraction in Java
- **Abstract classes and Abstract methods**
- **Encapsulation vs Data Abstraction**
- **Advantages of Abstraction**
- **Example**

# Abstraction in Java





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

# How can implement Data Abstraction in Java

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

# Abstract classes and Abstract methods



- An abstract class is a class that is declared with an abstract keyword. class Shape -> public class Shape -> abstract class Shape
- An abstract method **is a method** that is declared without implementation.
- An abstract class may or may not have all abstract methods. Some of them can be ***concrete methods***.
- A **method defined abstract** must always be *redefined in the subclass*, thus making overriding compulsory OR either make the subclass itself abstract.

Contd..

# Contd.. *Abstract classes and Abstract methods*




- Any class that contains one or more abstract methods must also be declared with an abstract keyword.
  - abstract void ReadData(); abstract void WriteData();
- There can be no object of an abstract class. That is, an abstract class can not be directly instantiated with the *new operator*.
- An abstract class can have parameterized constructors and the default constructor is always present in an abstract class.

# Encapsulation vs Data Abstraction





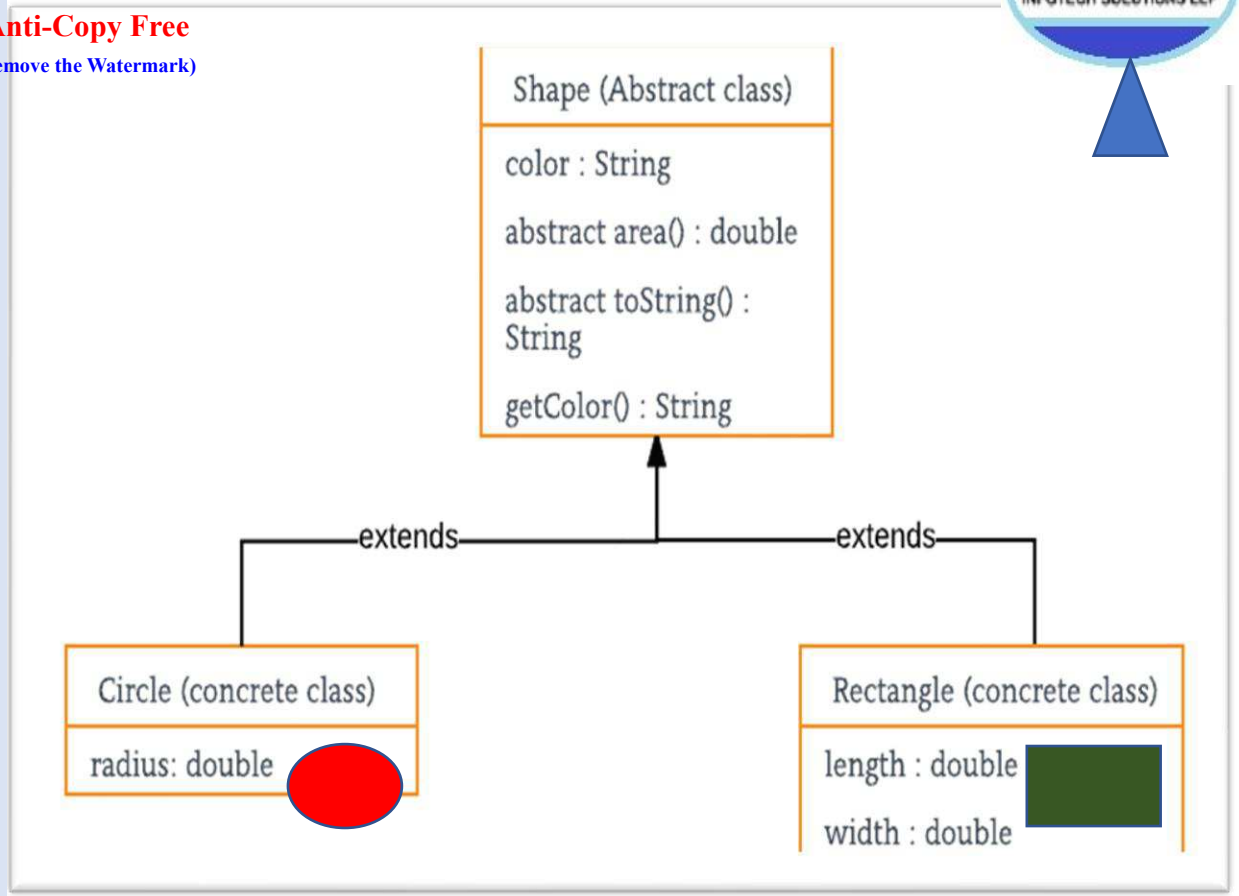
- Encapsulation is data hiding (information hiding)
  - *while Abstraction is detailed hiding (implementation hiding).*
- While encapsulation groups together data and methods that act upon the data,
  - *data abstraction deal with exposing the interface to the user and hiding the details of implementation.*

# Advantages of Abstraction

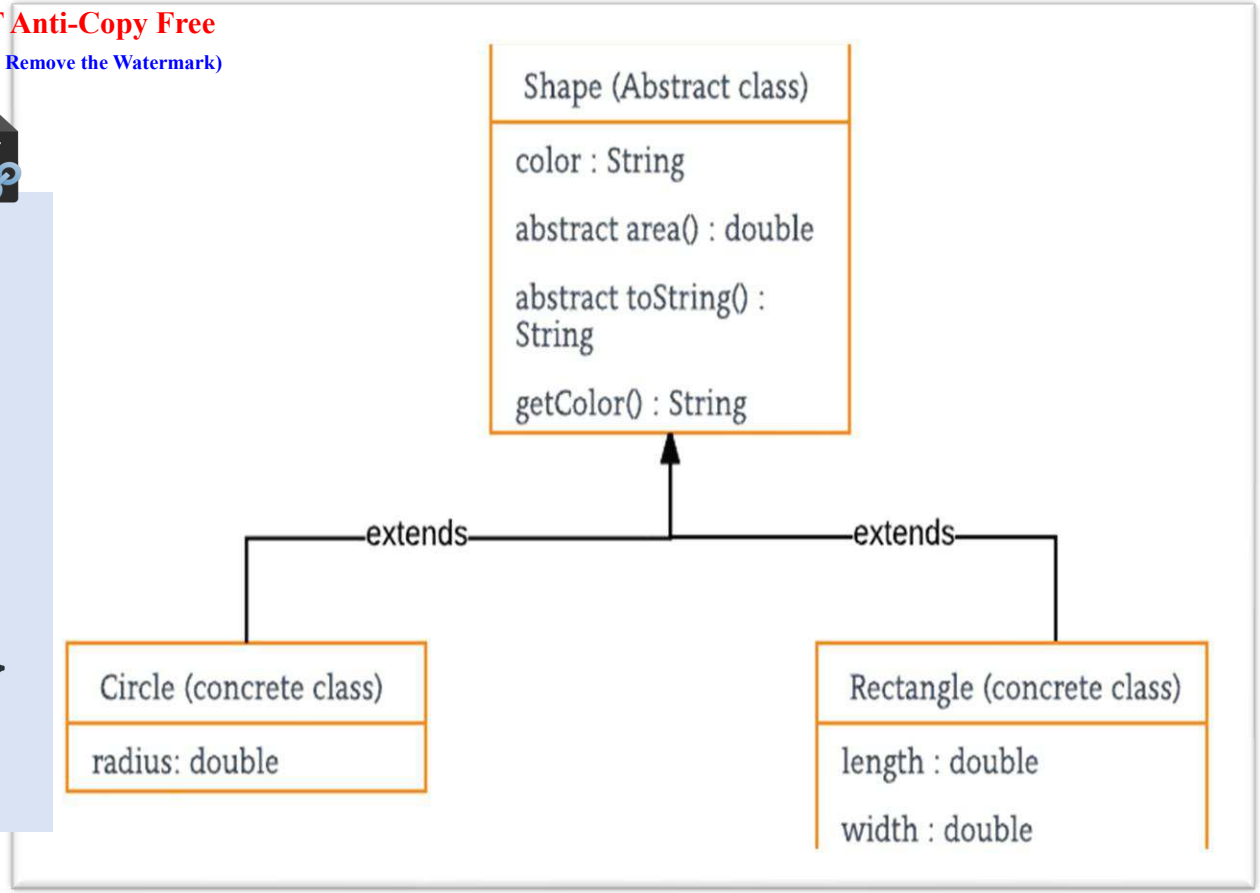



- It reduces the complexity *of viewing the things*.
- Avoids code duplication and increases reusability.
- Helps to increase the security of an application or program as only important details are provided to the user.

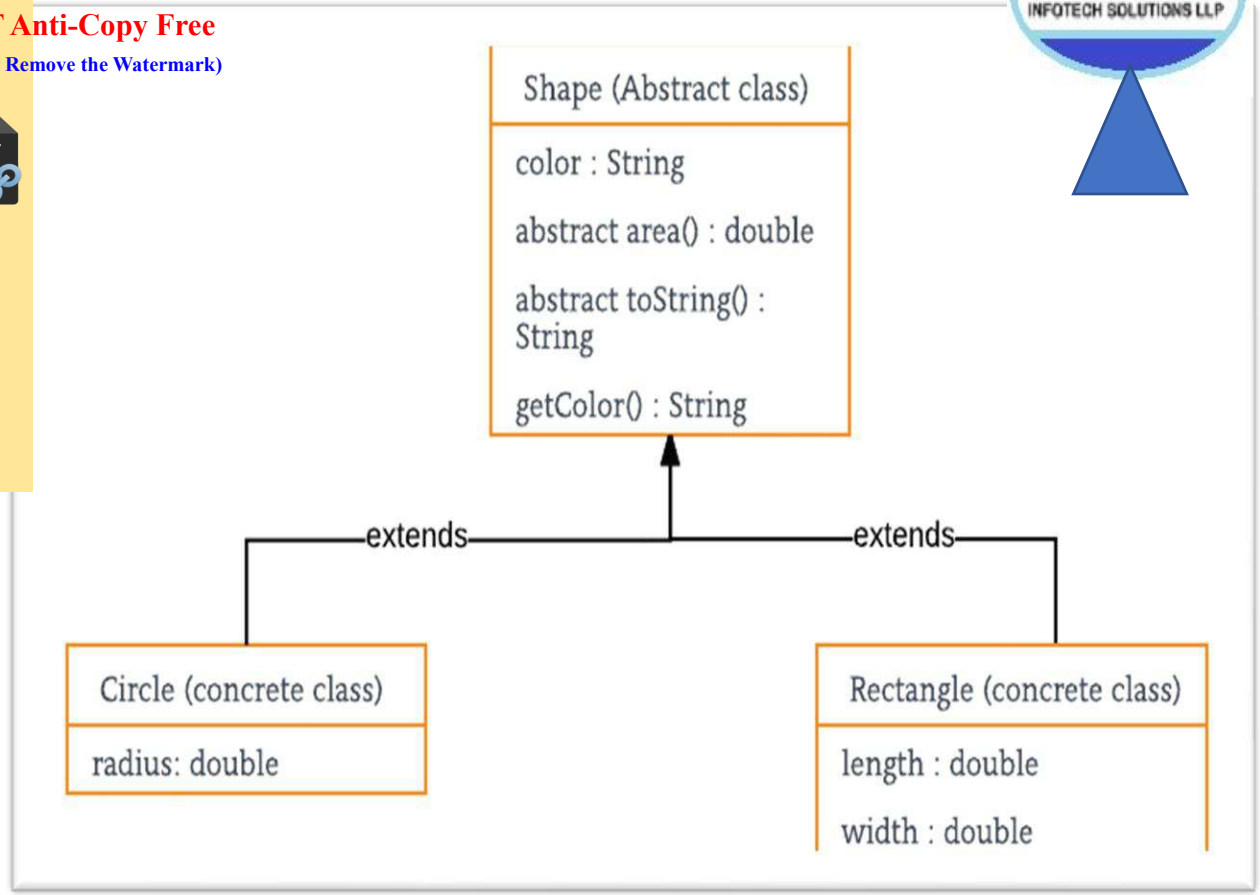
- Consider a classic **shape** used in a computer-aided design system or game simulation.
- The base type is **shape** and each shape has a color, size, and so on.
- From this, specific types of **shapes** are derived (inherited) - *circle, square, triangle,* and so on each of which may have additional characteristics and behaviors.
- For example, certain shapes can be flipped. Some behaviors may be different, such as when we want to calculate the area of a shape.
- The type hierarchy embodies both the similarities and differences between the shapes.



- // Java program to illustrate the
- // concept of Abstraction
- abstract class Shape {}
- class Circle extends Shape {}
- class Rectangle extends Shape {}
- public class Main {}



- // Java program to illustrate the
- // concept of Abstraction
- abstract class Shape {}
- class Circle extends Shape {}
- class Rectangle extends Shape {}

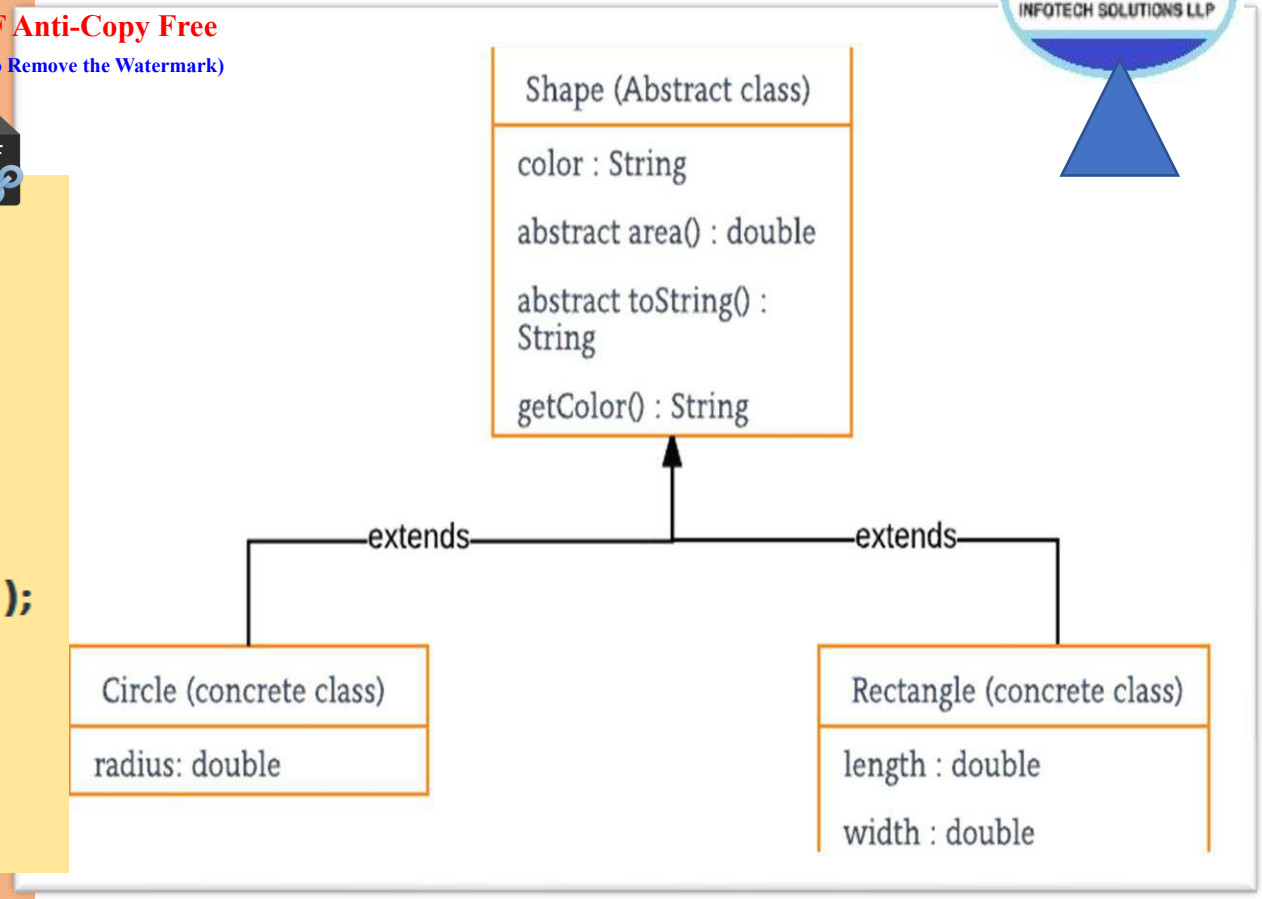



```
public class Main {
  public static void main(String[] args)
    {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);
        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

Circle color is Red and area is : 15.205308443374602
Rectangle color is Yellow and area is : 8.0

- // 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 {}
- class Rectangle extends Shape {}
- public class Main {}

- // 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 {}
- class Rectangle extends Shape {}

```
public class Main {
  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());
    }
}
```

```
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 Rectangle extends Shape {}

```
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();
    }
}
```

```
public class Main {
  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());
    }
}
```

```
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 Rectangle extends Shape {
   double length, 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();
   }
}
```

```
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();
    }
}
```

```
public class Main {
  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());
    }
}
```