

@Author: Nyakama Mahle

Date: 11/08/2023

Abstraction in Java

Abstraction in Java is a fundamental concept in object-oriented programming that allows you to create more manageable and understandable software by focusing on the essential features of an object while hiding unnecessary details. It provides a way to represent real-world entities and their behaviors in a simplified manner within your code.

In Java, abstraction is achieved through two main mechanisms: abstract classes and interfaces.

Abstract Classes:

An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed by other classes. It serves as a blueprint for creating more specific classes. Abstract classes can have both abstract (unimplemented) methods and concrete (implemented) methods. By declaring methods as abstract, you indicate that the subclass must provide a concrete implementation for those methods.

Example:

```
// abstract class
abstract class Shape {
    private String name;

    // concrete method
    public String getName() {
        return name;
    }

    //Concrete method
    public void setShape(String name) {
        this.name = name;
    }

    // concrete method
    public void setDimension(double dimension) {
        System.out.println(" Dimension of a shape : " + dimension);
    }

    // abstract method
    abstract void print();
}
```

```
public class Rectangle extends Shape {  
    private double width;  
    private double length;  
    public double getWidth() {  
        return width;  
    }  
  
    public double getLength() {  
        return length;  
    }  
    // method overloading  
    public void setShape(String name , double width , double length) {  
        super.setShape(name);  
        this.width = width;  
        this.length = length;  
    }  
    // method overloading  
    public void setDimension(double width , double length) {  
        this.width = width;  
        this.length = length;  
    }  
    // method overriding  
    @Override  
    void print() {  
        System.out.println("Name " + getName());  
        System.out.println("Width : " + getWidth());  
        System.out.println("Length : " + getLength());  
    }  
}
```

```
public class Cube extends Rectangle {

    private double height;

    public double getHeight() {
        return height;
    }

    // method overloading
    public void setShape(String name, double width, double length, double height) {
        super.setShape(name);
        super.setShape(name, width, length);
        this.height = height;
    }

    // method overloading
    public void setDimension(double width, double length, double height) {
        super.setDimension(width, length);
        if (height >= 0.0) {
            this.height = height;
        }
    }

    // method overriding
    @Override
    void print() {
        super.print();
        System.out.println("Height : " + getHeight());
    }

}
```

```

public class App {

    public static void main(String[] args) {

        // Create a Rectangle object

        Rectangle rectangle = new Rectangle();

        rectangle.setShape("Rectangle", 5.0, 10.0);

        rectangle.print();


        // Create a Cube object

        Cube cube = new Cube();

        cube.setShape("Cube", 4.0, 4.0, 4.0);

        cube.print();

    }

}

```

1. Abstract Class Shape:

- The Shape class is declared as abstract, indicating that it cannot be instantiated directly.
- Abstract methods are declared within the abstract class (print() in this case), which are intended to be overridden by its subclasses.
- The getName() method and the setShape(String name) method provide a clear interface for setting and retrieving the shape's name.

2. Inheritance:

- The Rectangle class inherits from the abstract Shape class, which signifies an "is-a" relationship between a rectangle and a shape.
- The Cube` class inherits from the `Rectangle` class, further building upon the inheritance hierarchy.

3. Method Overriding:

- Both the `Rectangle` and `Cube` classes override the `print()` method from the `Shape` class, providing their own implementations. This demonstrates the ability of subclasses to provide specialized behavior.

4. Method Overloading:

- The `setShape` and `setDimension` methods in both `Rectangle` and `Cube` classes are overloaded. This means that there are multiple methods with the same name but different parameter lists. It allows flexibility in how dimensions and attributes of shapes are set.

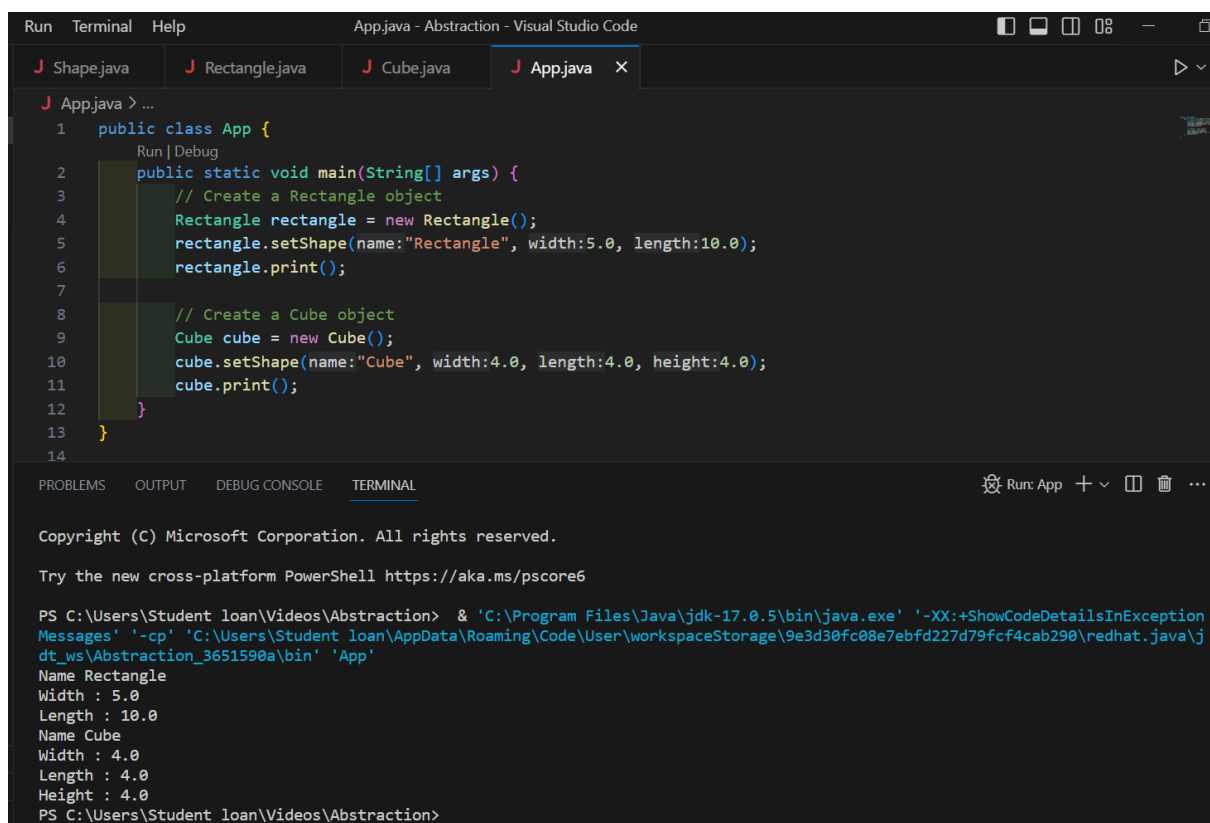
5. Access Modifiers:

- The use of access modifiers like `private`, `public`, and `protected` helps in controlling the visibility of members (variables and methods) of classes. This encapsulation is a form of abstraction, as it hides the internal details of the classes.

6. Abstract vs. Concrete Methods:

- The `Shape` class defines an abstract method `print()`, which must be implemented by its concrete subclasses. This enforces a certain behavior in subclasses without providing a complete implementation in the base class.

In summary, the provided code demonstrates the concepts of abstraction, inheritance, method overloading, method overriding, access modifiers, encapsulation, and other fundamental principles of object-oriented programming. These concepts collectively allow you to model and manipulate different shapes while abstracting away the complex implementation details.



The screenshot shows the Visual Studio Code editor with a project named 'App.java - Abstraction'. The editor has four tabs: 'Shape.java', 'Rectangle.java', 'Cube.java', and 'App.java'. The 'App.java' tab is active, showing the following code:

```
1 public class App {
2     public static void main(String[] args) {
3         // Create a Rectangle object
4         Rectangle rectangle = new Rectangle();
5         rectangle.setShape(name:"Rectangle", width:5.0, length:10.0);
6         rectangle.print();
7
8         // Create a Cube object
9         Cube cube = new Cube();
10        cube.setShape(name:"Cube", width:4.0, length:4.0, height:4.0);
11        cube.print();
12    }
13 }
14
```

Below the editor, the 'TERMINAL' panel is open, displaying the output of the Java application. The terminal shows the following text:

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\Student loan\Videos\Abstraction> & 'C:\Program Files\Java\jdk-17.0.5\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Student loan\AppData\Roaming\Code\User\workspaceStorage\9e3d30fc08e7ebfd227d79fcf4cab290\redhat.java\jdt_ws\Abstraction_3651590a\bin' 'App'
Name Rectangle
Width : 5.0
Length : 10.0
Name Cube
Width : 4.0
Length : 4.0
Height : 4.0
PS C:\Users\Student loan\Videos\Abstraction>
```

