University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Object-Oriented Programming

Polymorphism, Abstract Classes and Interfaces

11 avril 2024

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester : S4

Plan

Introduction

Polymorphism is a fundamental concept in Java and other object-oriented programming languages, allowing for actions to behave differently based on the actual object that is performing the action

What is Polymorphism ?

- The term "polymorphism" originates from the Greek words "poly" (many) and "morph" (form).
- In Java, it refers to the ability of a single interface to control access to a general class of actions.
- You can specify a general set of stack routines that all share the same names
- The concept of polymorphism is often expressed by the phrase “one interface, multiple methods

Types of Polymorphism in Java

1. Compile-time Polymorphism (Static Polymorphism)

- Achieved through method overloading.
- Java does not support operator overloading.
- Example :
  - `void display(int a)`
  - `void display(int a, int b)`

2. Runtime Polymorphism (Dynamic Polymorphism)

- Achieved through method overriding.
- Requires inheritance.
- Example :
  - In a superclass `Animal`, a method `makeSound()` is defined.
  - The subclass `Dog` overrides `makeSound()` to provide a specific implementation.

Example : Compile-time Polymorphism

```
public class Calculator {

  public int add(int a, int b) {
    return a + b;
  }

  public double add(double a, double b) {
    return a + b;
  }
}
```

Example : Runtime Polymorphism

```
class Animal {
  void sound() {
    System.out.println("Some sound");
  }
}
class Lion extends Animal {
  @Override
  void sound() {
    System.out.println("Roar");
  }
}
class Snake extends Animal {
  @Override
  void sound() {
    System.out.println("Hiss");
  }
}
```

```
public class TestPolymorphism {
 public static void main(String[] args) {
    Animal myAnimal = new Animal();
    Animal myLion = new Lion();
    Animal mySnake = new Snake();

    myAnimal.sound(); // Outputs: Some sound
    myLion.sound();   // Outputs: Roar
    mySnake.sound();  // Outputs: Hiss
  }
}
```

1. Polymorphism
2. Abstract classes
3. Introduction to Interfaces
   - Characteristics of Interfaces
   - Defining an Interface
   - Implementing an Interface

What are Abstract Classes ?

- Abstract classes are classes that cannot be instantiated on their own.
- They are used to provide a base for subclasses to build upon.
- Abstract classes can include abstract methods, which are method declarations without an implementation.

Characteristics of Abstract Classes

Key Features

- **Instantiation :** Cannot create instances directly.
- **Subclassing :** Must be subclassed by concrete classes.
- **Abstract Methods :** Can contain abstract methods that *must* be implemented by subclasses.

Purpose

- Provides a template for future specific classes.
- Helps to avoid redundancy and enhance reusability.

Rules for Abstract Classes

- An abstract class may contain both abstract and non-abstract methods.
- Abstract methods do not specify a body and only provide a method signature.
- If a class includes even one abstract method, the class must be declared abstract.

Using Abstract Classes

- Abstract classes are crucial for situations where a general framework needs to be established, and specific behaviors need to be enforced.
- Subclasses of an abstract class must implement all abstract methods, but they can also override other methods.

Example

```
// Abstract class defining common functionality for shapes
public abstract class Shape {

  // Abstract method - subclasses must provide implementation
  public abstract double calculateArea();

  // Non-abstract method with default implementation (can be
        overridden)
  public void printDetails() {
    System.out.println("This is a shape.");
  }
}
```

```java
public class Circle extends Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }
  // Implementation for calculateArea() specific to circles
  @Override
  public double calculateArea() {
    return Math.PI * radius * radius;
  }
  // Overriding printDetails() to provide specific information
        for circles
  @Override
  public void printDetails() {
    System.out.println("This is a circle with radius: " +
          radius);
  }
}
```

```
public class Square extends Shape {
  private double sideLength;

  public Square(double sideLength) {
    this.sideLength = sideLength;
  }

    // Implementation for calculateArea() specific to squares
  @Override
  public double calculateArea() {
    return sideLength * sideLength;
  }
  }
```

```java
public class Rectangle extends Shape {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  @Override
  public double calculateArea() {
    return width * height;
  }

  @Override
  public void printDetails() {
    System.out.println("This is a rectangle with width: " +
           width + " and height: " + height);
  }
}
```

```
public class Main {
  public static void main(String[] args) {
/*You cannot directly create an object of the abstract class
      Shape*/
    Circle circle = new Circle(5);
    Square square = new Square(4);
    Rectangle rectangle = new Rectangle(6, 3);

    System.out.println("Circle Area: " +
          circle.calculateArea());
    System.out.println("Square Area: " +
          square.calculateArea());
    System.out.println("Rectangle Area: " +
          rectangle.calculateArea());

    circle.printDetails();
    square.printDetails();
    rectangle.printDetails();
  }
}
```

What are Interfaces ?

- Interfaces in Java are a blueprint of a class. They have static constants and abstract methods.
- Java interfaces specify what a class must do but not how it does it.
- They are implemented by classes which then define the methods' behavior.

Characteristics of Interfaces

Key Features

- **Abstract Methods :** All methods in interfaces are implicitly abstract and public.
- **Constants :** All fields are public, static, and final (constant values).
- **Implementation :** A class can implement multiple interfaces.

Why Use Interfaces ?

- To achieve abstraction.
- To support the functionality of multiple inheritance.
- To separate the method definition from the method implementation.

Defining an Interface

- Syntax to define an interface is similar to class.
- Example :

Simple Interface

```
public interface Vehicle  {
    void cleanVehicle();
    int getNumberOfWheels();
}
```

- This 'Vehicle' interface can be implemented by any class that pertains to a mode of transport that needs cleaning and uses wheels.

Implementing an Interface

- A class implements an interface using the '**implements**' keyword.
- It must provide a body for all abstract methods from the interface.

```
public class Car implements Vehicle {
    public void cleanVehicle() {
      System.out.println("Cleaning the vehicle");
    }
    public int getNumberOfWheels() {
      return 4;
    }
  }
```

- 'Car' class implements the 'Vehicle' interface and provides implementation for the cleaning and wheel count methods.

Example

```
public interface Drawable {
  double PI = 3.14159; // implicitly public, static, and final

  void draw(); // implicitly public and abstract
  default void printMessage() {
    System.out.println("This is a drawable object.");
  }
}

public interface Resizable {
  void resize(int newSize);
}

public abstract class Shape {
  public abstract double getArea();
  public abstract double getPerimeter();
}
```

```
public class Circle extends Shape implements Drawable,
      Resizable {
  private double radius;
  public Circle(double radius)
  {  this.radius = radius; }

  @Override
  public void draw()
  {   System.out.println("Draw a circle, radius:" + radius);}
  @Override
  public double getArea()
  {  return Math.PI * radius * radius;  }
  @Override
  public double getPerimeter()
  {  return 2 * Math.PI * radius;}
  public double getRadius()
  {  return radius; }
  @Override
  public void printMessage()
  {  System.out.println("This is a circle.");}
}
```

```
Circle c = new Circle(5.0);

c.draw();

System.out.println("Area: " + c.getArea());

System.out.println("Perimeter: " + c.getPerimeter());

System.out.println("Radius: " + c.getRadius());

c.printMessage();
```

Questions ?