Polymorphism Implementation:

Question: Create a Java program that demonstrates polymorphism using a superclass named **Shape** and its subclasses **Circle**, **Rectangle**, and **Square**. Each subclass should override a method named **area()** to calculate and return the area of the shape.

```
// Superclass Shape
abstract class Shape {
    abstract double area();
}

// Subclass Circle
class Circle extends Shape {
    double radius;
    Circle(double radius) { this.radius = radius; }
    double area() { return Math.PI * radius * radius; }
}

// Subclass Rectangle
class Rectangle extends Shape {
    double length, width;
    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    double area() { return length * width; }
}

// Subclass Square
class Square extends Rectangle {
    Square(double side) { super(side, side); }
}

// Main class to demonstrate polymorphism
public class PolymorphismDemo {
    public static void main(String[] args) {
        Shape[] shapes = new Shape[]{
            new Circle(5), new Rectangle(4, 5), new Square(6)
        };

        for (Shape shape : shapes) {
            System.out.println("Area: " + shape.area());
        }
    }
}
```

Abstract Class Usage:

Question: Define an abstract class **Vehicle** with an abstract method **numberOfWheels()**. Create two subclasses, **Bike** and **Car**, which implement the **numberOfWheels** method. Demonstrate the usage of these classes in a main method.

```
// Abstract class Vehicle
abstract class Vehicle {
    abstract int numberOfWheels();
}

// Subclass Bike
class Bike extends Vehicle {
    int numberOfWheels() { return 2; }
}

// Subclass Car
class Car extends Vehicle {
    int numberOfWheels() { return 4; }
}

// Main class to demonstrate usage of abstract class
public class AbstractClassDemo {
    public static void main(String[] args) {
        Vehicle bike = new Bike();
        Vehicle car = new Car();
        System.out.println("Bike       wheels:       "       +
bike.numberOfWheels());
        System.out.println("Car       wheels:       "       +
car.numberOfWheels());
    }
}
```

Interface Implementation:

Question: Create an interface named **Animal** with a method **makeSound()**. Implement this interface in two classes, **Dog** and **Cat**, and define how each animal makes a sound. Write a main method to demonstrate polymorphic behavior of the **makeSound** method.

```
// Interface Animal
interface Animal {
    void makeSound();
}

// Class Dog implementing Animal
class Dog implements Animal {
    public void makeSound() { System.out.println("Bark"); }
}

// Class Cat implementing Animal
class Cat implements Animal {
    public void makeSound() { System.out.println("Meow"); }
}

// Main class to demonstrate interface implementation
public class InterfaceDemo {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.makeSound();
        cat.makeSound();
    }
}
```

Comparable Interface Task:

Question: Implement the **Comparable** interface in a **Student** class that has properties like **name** and **grade**. Write a method to sort an array of **Student** objects based on their grades.

```
// Class Student implementing Comparable
class Student implements Comparable<Student> {
    String name;
    int grade;
    Student(String name, int grade) {
        this.name = name;
        this.grade = grade;
    }

    public int compareTo(Student other) {
        return this.grade - other.grade;
    }
}

// Main class to sort Student objects
import java.util.Arrays;
public class ComparableDemo {
    public static void main(String[] args) {
        Student[] students = {
            new Student("Alice", 90),
            new Student("Bob", 85),
            new Student("Charlie", 95)
        };
        Arrays.sort(students);
        for (Student s : students) {
            System.out.println(s.name + ": " + s.grade);
        }
    }
}
```

Inner Class Challenge:

Question: Create a class **OuterClass** with a non-static inner class **InnerClass**. The **InnerClass** should have a method that returns an integer. Demonstrate how to instantiate the **InnerClass** from within the **OuterClass** and how to access the method of **InnerClass**.

```
// Class with Inner Class
public class OuterClass {
    class InnerClass {
        int innerMethod() { return 5; }
    }

    void demonstrateInner() {
        InnerClass inner = new InnerClass();
        System.out.println("Inner method returns: " +
inner.innerMethod());
    }

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.demonstrateInner();
    }
}
```

Static Inner Class Task:

Question: Define a static inner class **NestedClass** inside a class **ParentClass**. Add a static method in **NestedClass** that returns a string message. Show how to call this method from a main method without creating an instance of **ParentClass**.

```
// Parent Class with a Static Inner Class
public class ParentClass {
    static class NestedClass {
        static String getMessage() { return "Hello from Nested
Class"; }
    }

    public static void main(String[] args) {
        System.out.println(NestedClass.getMessage());
    }
}
```