الجلسة الثامنة - برمجة 3

الغاية من الجلسة: تعددية الأشكال Polymorphism

في برمجة Java فإن تعددية الأشكال Polymorphism هي مفهوم يتيح للكائنات أن يكون لها أشكال متعددة، أو أن تظهر سلوكاً مختلفاً استناداً إلى السياق الذي يُستخدم فيه.

في سياق البرمجة الكائنية، يمكن تحقيق التعددية عبر استخدام مفهومين رئيسيين: الوراثة وتجاوز الطرق. بالوراثة، يمكن لكائن أن يرث واصفات (سمات) وسلوكيات من كائن أب أساسي، بينما قد يتم تعديل السلوكيات لتناسب الحاجة في كائن فرعي. على سبيل المثال، يمكن أن يكون لدينا كائن "شكل"، والذي يمكن أن يكون له أشكال متعددة مثل "دائرة"، "مربع"، و "مثلث"، حيث يرث كل كائن من هذه الأشكال السمات والسلوكيات العامة للكائن "شكل"، ويقوم كل منها بتطبيق السلوكيات بطريقة مخصصة للشكل الخاص به.

يمكن أن نرى التعددية عبر وجود واجهة "شكل" التي تحتوي على طريقة "calculateArea()" ، والتي يتم تحقيقها في كل من الأصناف "دائرة"، "مربع"، و "مثلث". يتيح ذلك لكل شكل أن يظهر سلوكاً مختلفاً عندما يستدعى"calculateArea()" ، وذلك استناداً إلى تعبيره الفريد عن طريقة حساب مساحته.

مثال:

اكتب برنامجاً يُدعى "ShapeManager" يحتوي على ثلاثة طرق لحساب مساحة الأشكال الهندسية: دائرة، مربع، ومثلث. تمثل كل من الدوائر والمربعات والمثلثات كائنات منفصلة. يجب على كل طريقة في "ShapeManager" استقبال كائن من الشكل المناسب (دائرة، مربع، أو مثلث) وحساب مساحتها وإرجاع قيمة المساحة كنتيجة.

```
public interface Shape {
  double calculateArea();
}

public class Circle implements Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  @Override
  public double calculateArea() {
```

الصف Math يحوي ثابت ستاتيكي PI وقيمته هي 3.14

```
    return Math.PI * radius * radius;
  }
}
```

```
public class Square implements Shape {
  private double side;

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

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

public class Triangle implements Shape {
//القاعدة والارتفاع
  private double base;
  private double height;

  public Triangle(double base, double height) {
    this.base = base;
    this.height = height;
  }

  @Override
  public double calculateArea() {
//مساحة المثلث هي نصف القاعدة في الارتفاع
    return 0.5 * base * height;
  }
}
```

ملاحظة: مساحة المربع هي طول الضلع في طول الضلع

```
public class ShapeManager {
  public double calculateArea(Shape shape) {
    return shape.calculateArea();
  }
}
```

كما ترى فإننا في الـShapeManager نمرر للطريقة غرض من نمط الواجهة السابقة وبالتالي حسب الغرض الممرر تكون المساحة أي أنه في حال مررنا غرضاً من دائرة بالتالي الطريقة calculateArea تحسب مساحة الدائرة الممررة، بينما في حال مررنا غرضاً من نمط مربع بالتالي الطريقة سوف تحسب مساحة المثلث.

والآن ننشئ الصف Main من أجل إنشاء الأغراض واستدعاء الطرق السابقة:

```
public class Main {
  public static void main(String[] args) {
    // Create instances of different shapes
    Circle circle = new Circle(5); // Circle with radius 5
    Square square = new Square(4); // Square with side length 4
    Triangle triangle = new Triangle(3, 6); // Triangle with base 3 and height 6

    // Create a ShapeManager instance
    ShapeManager shapeManager = new ShapeManager();

    // Calculate and print the area of each shape using the ShapeManager
    double circleArea = shapeManager.calculateArea(circle);
    System.out.println("Area of circle: " + circleArea);

    double squareArea = shapeManager.calculateArea(square);
    System.out.println("Area of square: " + squareArea);

    double triangleArea = shapeManager.calculateArea(triangle);
    System.out.println("Area of triangle: " + triangleArea);
  }
}
```

سؤال: كيف يمكن أن يكون شكل الكود لو لم نستخدم تعددية الأشكال؟

كنا سوف نضطر إلى أن نضع طريقة مساحة من أجل كل شكل من الأشكال الثلاث، كما الكود الآتي:

```
public class ShapeManager {
  public double calculateCircleArea(Circle circle) {
    // Calculate area of circle
  }

  public double calculateSquareArea(Square square) {
    // Calculate area of square
  }

  public double calculateTriangleArea(Triangle triangle) {
    // Calculate area of triangle
  }
}
```

مثال آخر:

اكتب الصف PointDrawer والذي يحوي دالة drawPoint تُستخدم لرسم نقطة معينة في الفضاء ثلاثي الأبعاد. استخدم مفهوم الـ Polymorphism لتحقيق ذلك. تأكد من توضيح كيف يمكن لهذه الدالة أن تُستخدم لرسم نقطة في الفضاء ثنائي الأبعاد أيضاً.

```
interface Point {
  void draw();
}

// صف يمثل نقطة في الفضاء ثلاثي الأبعاد
class ThreeDimensionalPoint implements Point {
  private int x, y, z;

  public ThreeDimensionalPoint(int x, int y, int z) {
    this.x = x;
    this.y = y;
```

```
    this.z = z;
  }

  @Override
  public void draw() {
    System.out.println("Drawing a 3D point at (" + x + ", " + y + ", " + z + ")");
  }
}

// صف يمثل نقطة في الفضاء ثنائي الأبعاد
class TwoDimensionalPoint implements Point {
  private int x, y;

  public TwoDimensionalPoint(int x, int y) {
    this.x = x;
    this.y = y;
  }

  @Override
  public void draw() {
    System.out.println("Drawing a 2D point at (" + x + ", " + y + ")");
  }
}

// صف لرسم النقاط
class PointDrawer {
  // طريقة لرسم النقاط باستخدام مفهوم تعددية الأشكال
  public static void drawPoint(Point point) {
    point.draw();
  }
}
```

```
// الصف الرئيسي Main
public class Main {
    public static void main(String[] args) {
        // إنشاء أغراض من الصفوف المختلفة
        Point point3D = new ThreeDimensionalPoint(10, 20, 30);
        Point point2D = new TwoDimensionalPoint(5, 8);

        PointManager.drawPoint(point3D);
        PointManager.drawPoint(point2D);
    }
}
```

انتهت الجلسة
