## الجلسة الرابعة - برمجة 3

**الغاية من الجلسة:** الواجهات Interfaces وتعددية الأشكال Polymorphism.

في عالم برمجة الحاسوب، تُعتبر الواجهات (Interfaces) أداة قوية ومرنة تُستخدم لتعريف مجموعة من الطرق دون توفير تنفيذ فعلي لهذه الطرق (دون تحقيق). تعتبر الواجهات جزءاً أساسياً من تقنيات البرمجة الكائنية الموجهة للواجهات(Object-Oriented Programming) ، وهي تسهل فهم البرمجيات وجعلها أكثر مرونة وقابلية لإعادة الاستخدام.

**تعريف الواجهات :** الواجهة في جافا تمثل مجموعة من الطرق (methods) التي يجب على أي كائن ينفذ (implements) هذه الواجهة تنفيذها. بمعنى آخر، تحدد الواجهة السلوك الذي يجب أن يتبعه الكائن الذي ينفذها دون تقديم أي تفاصيل حول كيفية تنفيذ هذا السلوك.

تُستخدم الواجهات في تعريف العقود (contracts) بين الكلاسات، حيث يمكن للكلاسات المختلفة تنفيذ نفس الواجهة بطرق مختلفة وفقاً لاحتياجات التطبيق.

لكي يتمكن كلاس ما من تنفيذ واجهة في جافا، يجب عليه استخدام الكلمة المفتاحية "implements" متبوعة باسم الواجهة التي يُريد تنفيذها. بمجرد تنفيذ الواجهة، يجب على الكلاس توفير تعريف لجميع الطرق الموجودة في الواجهة.

### مثال على الواجهات وآلية استخدامها:

يرغب مطور تطبيق في إنشاء هيكل بيانات للأشياء التي يمكن بيعها ونقلها. يهدف الهيكل إلى توفير معلومات عن الأشياء المباعة وإمكانية نقلها بطريقة مرنة وآمنة.

يجب على المطور تعريف مجموعة من الخصائص والسلوكيات المشتركة لجميع الأشياء المباعة والقابلة للنقل. يتم ذلك من خلال واجهات مختلفة، حيث يتم تنفيذ هذه الواجهات بواسطة الكلاسات المعنية دون الحاجة إلى ذكر التفاصيل المحددة للتنفيذ في الواجهة.

بناءً على هذه المتطلبات، يتم تصميم الهيكل كالتالي:

1. يتضمن واجهة Sellable معلومات حول الأشياء المباعة مثل الوصف والسعر المدرج وأقل سعر يتم قبوله.
2. تقدم واجهة Transportable الوصول إلى معلومات حول الأشياء التي يمكن نقلها، مثل الوزن ومعلومة عما إذا كانت الشيء خطرًا أم لا.

```
// Interface for objects that can be sold.
public interface Sellable {
  /* description of the object*/
public String description();
/*list price in cents*/
public int listPrice();
/*lowest price in cents we will accept*/
public int lowestPrice();
}
// Interface for objects that can be transported.
public interface Transportable {
```

```
  /** weight in grams*/
public int weight();
/** whether the object is hazardous*/
 public boolean isHazardous();
}
```

نريد الآن تعريف واجهة Trackable والتي تعبر عن المنتجات القابلة للتتبع والتي ترث الواجهتين السابقتين وتضيف طرق إضافية مثل طريقة تعيد id المنتج، وطريقة للتأكد من وصول المنتج إلى الهدف أم لا، وما هي حالة المنتج status والتي تعبر عن حالة المنتج هل هو سليم أم مكسور أم مفقود ... الخ،

```
public interface Trackable extends Sellable, Transportable {
  /**
   * Returns the product ID associated with the item.
   * @return the product ID
   */
  String getProductId();

  /**
   * Checks if the item has arrived at its destination.
   * @return true if the item has arrived, false otherwise
   */
  boolean hasArrived();

  /**
   * Returns the status of the item.
   * @return the status of the item
   */
  String getStatus();

  /**
   * Sets the status of the item.
   * @param status the new status of the item
   */
```

```
    void setStatus(String status);
}
```

والآن نعرف المنتج الذي يحقق الواجهة السابقة Trackable:

```
public class Perfume implements Trackable {
    private String description;
    private int price;
    private int lowestPrice;
    private int weight;
    private boolean hazardous;
    private String productId;
    private boolean arrived;
    private String status;

    // Constructor
    public Perfume(String description, int price, int weight, boolean hazardous, String productId, boolean arrived,
String status) {
        this.description = description;
        this.price = price;
        this.lowestPrice = price/2;
        this.weight = weight;
        this.hazardous = hazardous;
        this.productId = productId;
        this.arrived = false;
        this.arrived = arrived;
        this.status = status;
    }

    // Implementing methods from the Trackable interface
    @Override
```

```
public String getProductId() {
    return productId;
}

@Override
public boolean hasArrived() {
    return arrived;
}

@Override
public String getStatus() {
    return status;
}

@Override
public void setStatus(String status) {
    this.status = status;
}

// Implementing methods from the Sellable interface
@Override
public String description() {
    return description;
}

@Override
public int listPrice() {
    return price;
}
```

```
    @Override
    public int lowestPrice() {
        return price/2;
    }

    // Implementing methods from the Transportable interface
    @Override
    public int weight() {
        return weight;
    }

    @Override
    public boolean isHazardous() {
        return hazardous;
    }

    // Other methods specific to Perfume class
    // Getters and setters
    // Additional business logic طرق إضافية
}
```

**والأن نعرف الصف الـ Main:**

```
public class Main {
    public static void main(String[] args) {
        // Create a new Perfume object
        Perfume perfume = new Perfume("Fancy Perfume", 5000, 50, false, "P12345", false, “Lost”);

        // Print information about the perfume
        System.out.println("Product ID: " + perfume.getProductId());
        System.out.println("Description: " + perfume.description());
        System.out.println("List Price: " + perfume.listPrice() + " cents");
```

```
        System.out.println("Lowest Price: " + perfume.lowestPrice() + " cents");
        System.out.println("Weight: " + perfume.weight() + " grams");
        System.out.println("Is Hazardous: " + perfume.isHazardous());
        System.out.println("Arrived: " + perfume.hasArrived());
        System.out.println("Status: " + perfume.getStatus());
    }
}
```

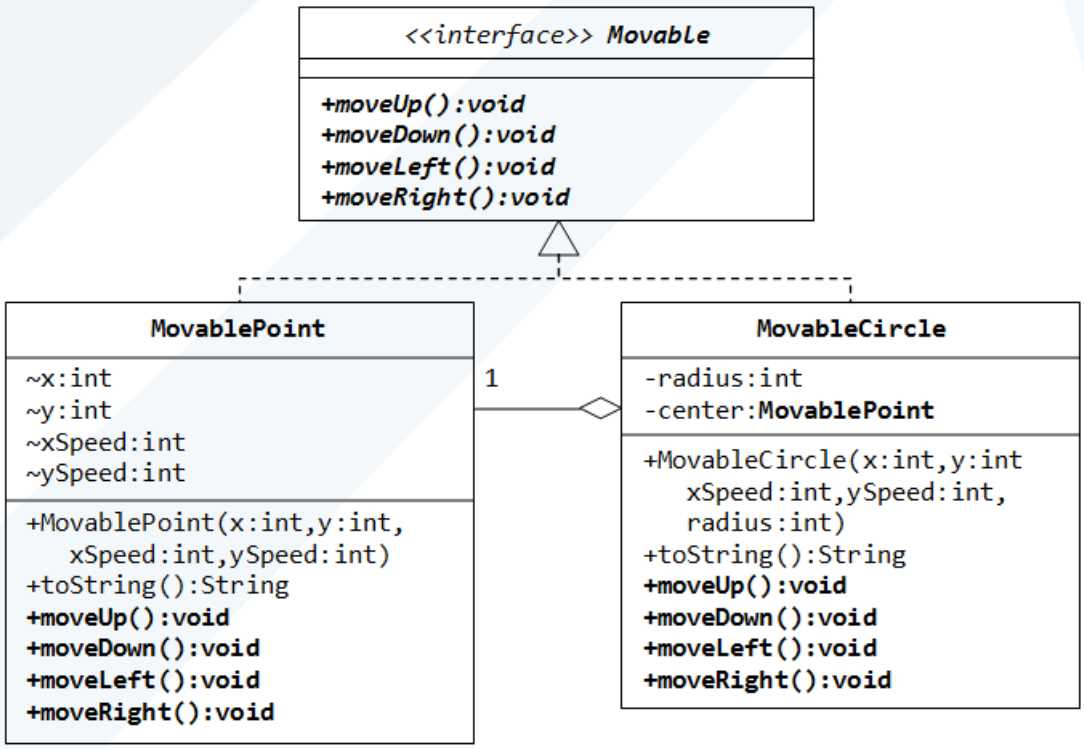
يمكننا أن نحقق تعددية الأشكال Polymorphism في الصف السابق بجعل نمط العطر هو Trackable:

```
Trackable perfume = new Perfume("Fancy Perfume", 5000, 50, false, "P12345", false, "Lost");
```

### مثال آخر على استخدام الواجهات:

لدينا واجهة قابلة للتحريك (Movable) تحتوي على عمليات مشتركة للكائنات القابلة للتحريك مثل الانتقال لأعلى ولأسفل ولليمين ولليسار.

الآن سنقوم بتحقيق هذه الواجهة في كلاسين MovablePoint وMovableCircle .




```
// Interface representing common operations for movable objects
```

```
interface Movable {
   void moveUp();
   void moveDown();
   void moveLeft();
   void moveRight();
}

// Implementation of Movable interface for a movable point
class MovablePoint implements Movable {
   private int x;
   private int y;

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

   @Override
   public void moveUp() {
      y += ySpeed;
   }

   @Override
   public void moveDown() {
      y -= ySpeed;
   }

   @Override
   public void moveLeft() {
      x -= xSpeed;
   }
```

```
    @Override
    public void moveRight() {
        x += xSpeed;
    }

    @Override
    public String toString() {
        return "MovablePoint{" + "x=" + x +  ", y=" + y + '}';
    }
}

// Implementation of Movable interface for a movable circle
class MovableCircle implements Movable {
    private MovablePoint center;
    private int radius;

    public MovableCircle(int x, int y, int radius) {
        this.center = new MovablePoint(x, y);
        this.radius = radius;
    }

    @Override
    public void moveUp() {
        center.moveUp();
    }

    @Override
    public void moveDown() {
        center.moveDown();
    }

    @Override
```

```
    public void moveLeft() {
        center.moveLeft();
    }

    @Override
    public void moveRight() {
        center.moveRight();
    }

    @Override
    public String toString() {
        return "MovableCircle{" +
                "center=" + center +
                ", radius=" + radius +
                '}';
    }
}

// Main class to demonstrate usage of Movable interface
public class Main {
    public static void main(String[] args) {
        // Create a movable point
        MovablePoint point = new MovablePoint(1, 1);
        System.out.println("Original point: " + point);
        point.moveUp();
        point.moveRight();
        System.out.println("Point after moving up and right: " + point);

        // Create a movable circle
        MovableCircle circle = new MovableCircle(0, 0, 3);
        System.out.println("Original circle: " + circle);
        circle.moveDown();
```

```
        circle.moveLeft();
        System.out.println("Circle after moving down and left: " + circle);
    }
}
```

طلب إضافي: ماذا لو كان لدينا مستطيلاً قابلاً للتحريك MovableRectangle؟ أضف الكود المناسب.

**انتهت الجلسة**