CS5800 Homework 05

Subham Panda

017314921

GitHub link

Decorator Problem

```
package decoratorcode;

// Concrete decorator class for additional toppings
public class AdditionalTopping extends ToppingDecorator {
    private String toppingName;
    private double toppingPrice;

    public AdditionalTopping(FoodItem foodItem, String toppingName, double
toppingPrice) {
        super(foodItem);
        this.toppingName = toppingName;
        this.toppingPrice = toppingPrice;
    }

    @Override
    public double getCost() {
        return super.getCost() + toppingPrice;
    }
}
```

```
package decoratorcode;

// Concrete component class for base food items
public class BaseFoodItem implements FoodItem {
    private String name;
    private double price;

    public BaseFoodItem(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public double getCost() {
        return price;
    }
}
```

```java
package decoratorcode;

// Interface for food items
public interface FoodItem {
    double getCost();
}
```

```java
package decoratorcode;

// Class representing customer's loyalty status
public class LoyaltyStatus {
    private double discountRate;

    public LoyaltyStatus(double discountRate) {
        this.discountRate = discountRate;
    }

    public double applyDiscount(double totalCost) {
        return totalCost * (1 - discountRate);
    }
}
```

```java
package decoratorcode;

import decoratorcode.FoodItem;

import java.util.ArrayList;
import java.util.List;

// Class representing customer's order
public class Order {
    private List<FoodItem> items = new ArrayList<>();

    public void addItem(FoodItem item) {
        items.add(item);
    }

    public double calculateTotalCost() {
        double totalCost = 0;
        for (FoodItem item : items) {
            totalCost += item.getCost();
        }
        return totalCost;
    }
}
```

```java
package decoratorcode;

import decoratorcode.FoodItem;

// Decorator class for toppings
public abstract class ToppingDecorator implements FoodItem {
    protected FoodItem foodItem;

    public ToppingDecorator(FoodItem foodItem) {
        this.foodItem = foodItem;
    }

    @Override
    public double getCost() {
        return foodItem.getCost();
    }
}
```

```java
package tests;

import decoratorcode.*;
import org.junit.Test;
import static org.junit.Assert.*;


public class RestaurantTest {

    @Test
    public void testBaseFoodItem() {
        FoodItem sushi = new BaseFoodItem("Sushi", 10.0);
        assertEquals(10.0, sushi.getCost(), 0.01);
    }

    @Test
    public void testAdditionalTopping() {
        FoodItem sushi = new BaseFoodItem("Sushi", 10.0);
        FoodItem topping = new AdditionalTopping(sushi, "Soy Sauce", 2.0);
        assertEquals(12.0, topping.getCost(), 0.01);
    }

    @Test
    public void testOrder() {
        Order order = new Order();
        FoodItem sushi = new BaseFoodItem("Sushi", 10.0);
        FoodItem topping = new BaseFoodItem("Imitation Crab", 4.5);
        order.addItem(sushi);
        order.addItem(topping);
        assertEquals(14.5, order.calculateTotalCost(), 0.01);
    }

    @Test
    public void testLoyaltyStatus() {
```

```
        LoyaltyStatus loyaltyStatus = new LoyaltyStatus(0.1);
        assertEquals(90.0, loyaltyStatus.applyDiscount(100.0), 0.01);
    }
}
```

```
import decoratorcode.*;

// Driver program
public class Driver {
    public static void main(String[] args) {
        // Create food items
        FoodItem sushi = new BaseFoodItem("Sushi", 15.0);
        FoodItem riceCake = new BaseFoodItem("Rice Cake", 4.5);

        // Add toppings
        FoodItem sushiWithCrab = new AdditionalTopping(sushi, "Imitation
Crab", 4.5);
        FoodItem sushiWithCrabCrunchy = new AdditionalTopping(sushiWithCrab,
"Crunchy", 2.5);

        // Create order
        Order order = new Order();
        order.addItem(sushiWithCrabCrunchy);
        order.addItem(riceCake);

        // Calculate total cost
        double totalCost = order.calculateTotalCost();
        System.out.println("Total cost before discount: $" + totalCost);

        // Apply discount based on loyalty status
        LoyaltyStatus loyaltyStatus = new LoyaltyStatus(0.1); // 10% discount
for example
        double discountedCost = loyaltyStatus.applyDiscount(totalCost);
        System.out.println("Total cost after discount: $" + discountedCost);
    }
}
```

Output

Driver

```
/Users/subhampanda/Library/Java/JavaVirtualMachines/openjdk-22/Contents/Home/bin/java
Total cost before discount: $26.5
Total cost after discount: $23.85

Process finished with exit code 0
```

TestCase