**Structural Design Patterns**

**Restaurant**

**Decorator**

You are building a software system for a restaurant that needs to keep track of customer orders. Customers can order various food items, such as burgers, fries, and hot dogs. Each food item has a base price and can have additional toppings or add-ons, which increase the price of the item.

Your task is to design a system that allows the restaurant to:

1. Create new food items with different base prices and toppings.
2. Calculate the total cost of a customer's order, including the cost of each item and any toppings.
3. Apply discounts to the total order cost based on the customer's loyalty status.

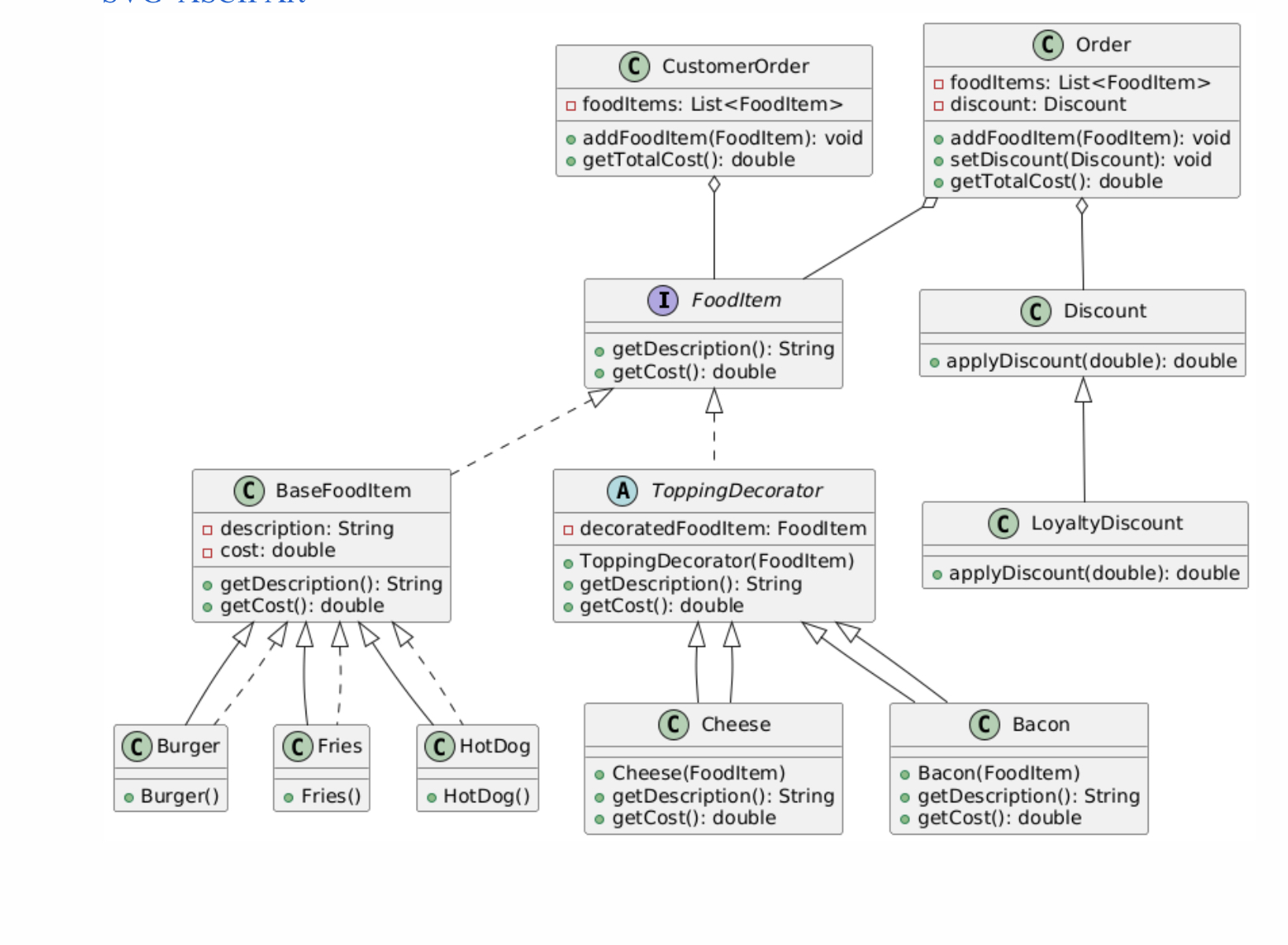
To accomplish this task, use the decorator design pattern to create a hierarchy of classes that represent the different food items and toppings. Each class should implement a common interface that allows the system to calculate the cost of the item.

Then, create a class that represents the customer's order. This class should have a list of food items and toppings, and it should be able to calculate the total cost of the order by iterating over the list and summing up the cost of each item.

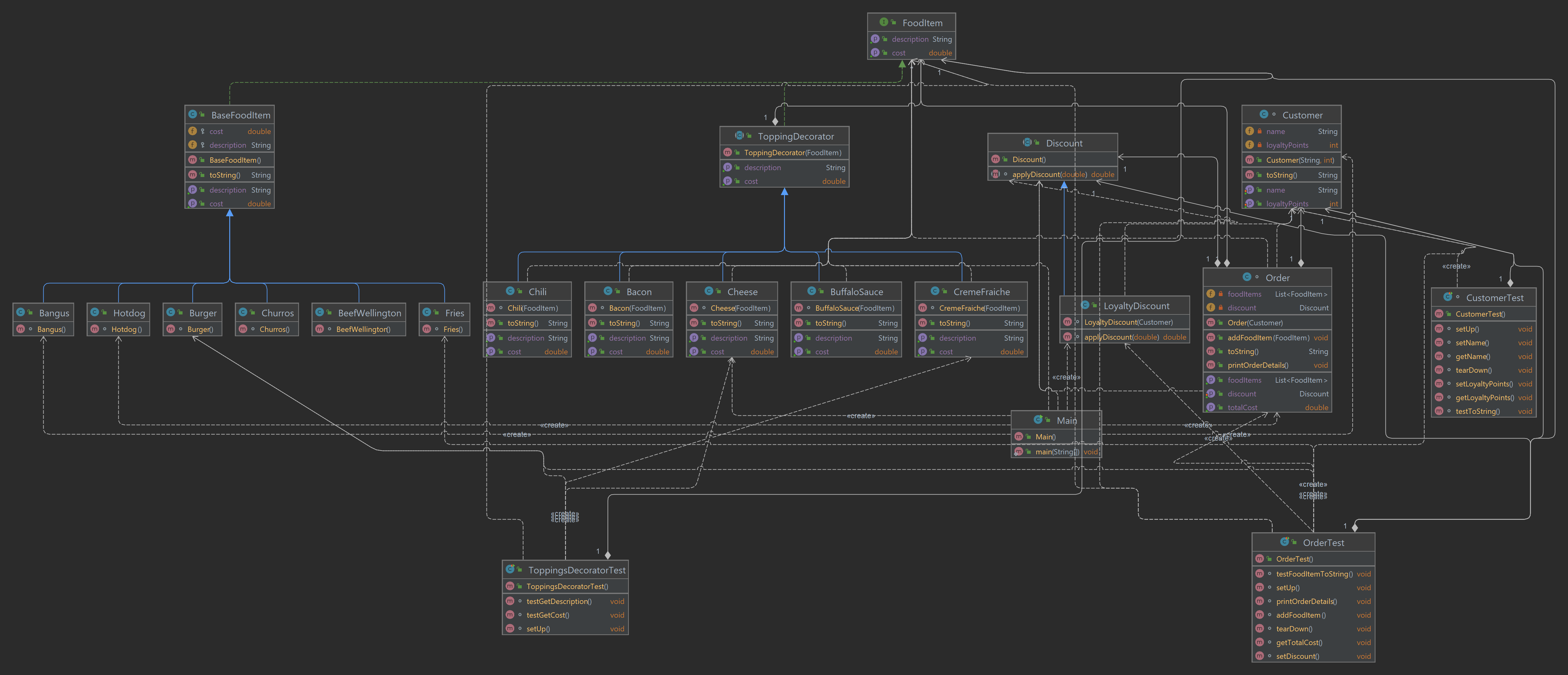
Finally, create a class that represents the customer's loyalty status. This class should have a method that applies a discount to the total order cost based on the customer's status.

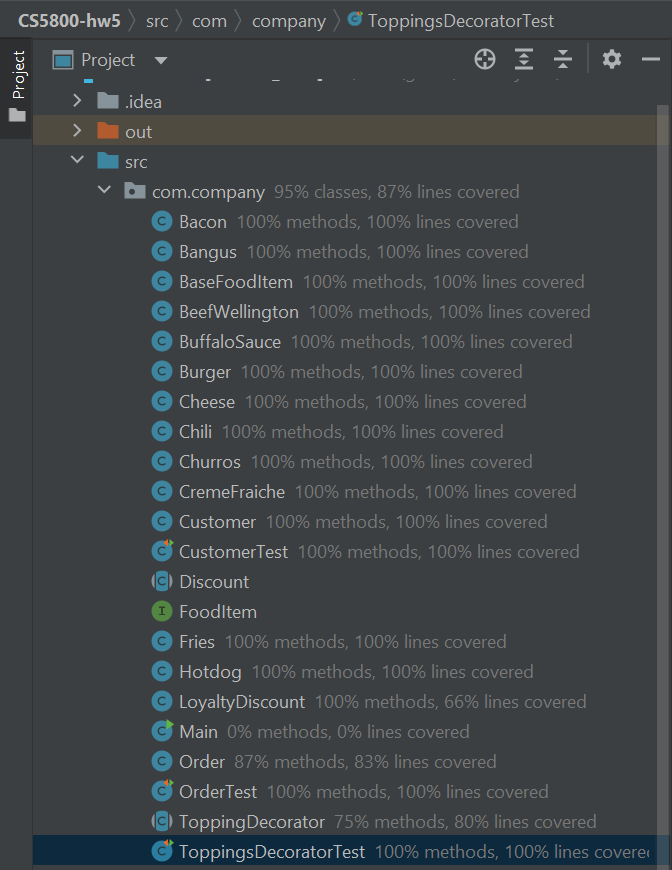
Implement this system using the decorator design pattern, and test it by creating some food items, adding toppings, creating an order, and applying a loyalty discount in the driver program.

Recall unit tests are required for all assignments at this point!

UML Diagram:

Final UML Diagram





OUTPUT:

A screenshot of a computer program

Description automatically generated

SOURCE CODE:

package com.company;  
  
public class Bacon extends ToppingDecorator{  
 Bacon(FoodItem decoratedFoodItem){  
 super(decoratedFoodItem,0.75);  
 }  
  
 @Override  
 public String getDescription(){  
 return decoratedFoodItem.getDescription() + ", Bacon";  
 }  
  
}

package com.company;  
  
*/\*\*  
 \* Fried fish item local to the philippines  
 \*/*public class Bangus extends BaseFoodItem {  
 Bangus(){  
 description = "Bangus";  
 cost = 18.00;  
 }  
}

package com.company;  
  
public class BaseFoodItem implements FoodItem{  
  
 protected String description;  
 protected double cost;  
  
 @Override  
 public String getDescription() {  
 return this.description;  
 }  
  
 @Override  
 public double getCost() {  
 return this.cost;  
 }  
  
 @Override  
 public String toString() {  
 return "BaseFoodItem{" +  
 "description='" + description + '\'' +  
 ", cost=" + cost +  
 '}';  
 }  
}

package com.company;  
  
*/\*\*  
 \* Fillet steak coated with pâté and duxelles, wrapped in shortcrust pastry, then baked  
 \*/*public class BeefWellington extends BaseFoodItem{  
 BeefWellington(){  
 description = "Beef Wellington";  
 cost = 24.99;  
 }  
}

package com.company;  
  
*/\*\*  
 \* Buffalo sauce duh  
 \*/*public class BuffaloSauce extends ToppingDecorator{  
  
 BuffaloSauce(FoodItem decoratedFoodItem){  
 super(decoratedFoodItem, 0.75);  
 }  
  
 @Override  
 public String getDescription() {  
 return decoratedFoodItem.getDescription() + ", Buffalo Sauce";  
 }  
  
}

package com.company;  
  
*/\*\*  
 \* Plain beef sirloin burger  
 \*/*public class Burger extends BaseFoodItem{  
 Burger(){  
 description = "Burger";  
 cost = 8.95;  
 }  
}

package com.company;  
  
public class Cheese extends ToppingDecorator{  
 Cheese(FoodItem decoratedFoodItem){  
 super(decoratedFoodItem,0.50);  
 }  
  
 @Override  
 public String getDescription() {  
 return decoratedFoodItem.getDescription() + ", Cheese";  
 }  
  
}

package com.company;  
  
*/\*\*  
 \* Specialty Chili made with Tri-tip Steak cooked for 5 hours  
 \*/*public class Chili extends ToppingDecorator{  
 Chili(FoodItem decoratedFoodItem){  
 super(decoratedFoodItem, 5.99);  
 }  
  
 @Override  
 public String getDescription() {  
 return decoratedFoodItem.getDescription() + ", Chili";  
 }  
  
}

package com.company;  
  
*/\*\*  
 \* 3 soft churros plate, choux pastry dough fried in hot oil, sprinkled with sugar  
 \*/*public class Churros extends BaseFoodItem{  
 Churros(){  
 description = "Churros Plate";  
 cost = 3.59;  
 }  
}

package com.company;  
  
*/\*\*  
 \* heavy cream thickened and slightly soured with buttermilk  
 \*/*public class CremeFraiche extends ToppingDecorator{  
 CremeFraiche(FoodItem decoratedFoodItem){  
 super(decoratedFoodItem,1.20);  
 }  
  
 @Override  
 public String getDescription() {  
 return decoratedFoodItem.getDescription() + ", Creme Fraiche";  
 }  
  
}

package com.company;  
  
// class representing a customer  
class Customer {  
 private String name;  
 private int loyaltyPoints;  
  
 public Customer(String name, int loyaltyPoints) {  
 setName(name);  
 setLoyaltyPoints(loyaltyPoints);  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getLoyaltyPoints() {  
 return loyaltyPoints;  
 }  
  
 public void setLoyaltyPoints(int loyaltyPoints) {  
 this.loyaltyPoints = loyaltyPoints;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 @Override  
 public String toString() {  
 return "Customer{" +  
 "name='" + name + '\'' +  
 ", loyaltyPoints=" + loyaltyPoints +  
 '}';  
 }  
}

package com.company;  
  
public abstract class Discount {  
 abstract double applyDiscount(double totalCost);  
}

package com.company;  
  
public interface FoodItem {  
  
 public String getDescription();  
 public double getCost();  
  
  
}

package com.company;  
  
*/\*\*  
 \* Crispy fries with a touch of salt  
 \*/*public class Fries extends BaseFoodItem{  
 Fries(){  
 description = "Fries";  
 cost = 2.99;  
 }  
}

package com.company;  
  
*/\*\*  
 \* Just a stupid hotdog  
 \*/*public class Hotdog extends BaseFoodItem{  
 Hotdog(){  
 description = "Hot dog";  
 cost = 1.35;  
 }  
}

package com.company;  
  
public class LoyaltyDiscount extends Discount{  
 private int loyaltyPoints = 0;  
  
 LoyaltyDiscount(Customer customer){  
 this.loyaltyPoints = customer.getLoyaltyPoints();  
 }  
  
 @Override  
 double applyDiscount(double totalCost) {  
 double discountPercentage;  
 if(loyaltyPoints >= 30000){  
 discountPercentage = 30.0; // 30% discount for 10,000 or more points  
 }  
 else if (loyaltyPoints >= 1000) {  
 discountPercentage = 20.0; // 20% discount for 1000 or more points  
 } else if (loyaltyPoints >= 500) {  
 discountPercentage = 15.0; // 15% discount for 500-999 points  
 } else if (loyaltyPoints >= 200) {  
 discountPercentage = 10.0; // 10% discount for 200-499 points  
 } else if (loyaltyPoints >= 100) {  
 discountPercentage = 5.0; // 5% discount for 100-199 points  
 } else {  
 discountPercentage = 0.0; // No discount for less than 100 points  
 }  
 return totalCost \* (1 - discountPercentage / 100);  
 }  
  
}

package com.company;  
  
import java.util.ArrayList;  
import java.util.List;  
  
// Class representing an order  
class Order {  
 private List<FoodItem> foodItems = new ArrayList<>();  
 private Discount discount;  
 private Customer customer;  
  
 public Order(Customer customer) {  
 this.customer = customer;  
 }  
  
 public void addFoodItem(FoodItem foodItem) {  
 foodItems.add(foodItem);  
 }  
  
 public void setDiscount(Discount discount) {  
 this.discount = discount;  
 }  
  
 public Discount getDiscount() {  
 return discount;  
 }  
  
 public List<FoodItem> getFoodItems() {  
 return foodItems;  
 }  
  
 public double getTotalCost() {  
 double totalCost = 0.0;  
 for (FoodItem item : foodItems) {  
 totalCost += item.getCost();  
 }  
 if (discount != null) {  
 totalCost = discount.applyDiscount(totalCost);  
 }  
 return totalCost;  
 }  
  
 public void printOrderDetails() {  
 System.*out*.println("Order for: " + customer.getName());  
 System.*out*.println("Loyalty points: " + customer.getLoyaltyPoints());  
 System.*out*.println("Total Cost: $" + getTotalCost());  
 }  
  
 @Override  
 public String toString() {  
 return "Order{" +  
 "foodItems=" + foodItems +  
 ", discount=" + discount +  
 ", customer=" + customer +  
 '}';  
 }  
}

package com.company;  
  
public abstract class ToppingDecorator implements FoodItem{  
 protected FoodItem decoratedFoodItem;  
 protected double toppingCost;  
  
 public ToppingDecorator(FoodItem decoratedFoodItem, double toppingCost) {  
 this.decoratedFoodItem = decoratedFoodItem; this.toppingCost = toppingCost;  
 }  
  
 @Override public String getDescription() {  
 return decoratedFoodItem.getDescription();  
 }  
  
 @Override public double getCost() {  
 return decoratedFoodItem.getCost() + toppingCost;  
 }  
 @Override public String toString() {  
 return getDescription() + ", total cost: " + getCost();  
 }  
  
  
}

TESTS:

package com.company;  
  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class CustomerTest {  
 private Customer customer1;  
  
 @BeforeEach  
 void setUp() {  
 customer1 = new Customer("John Doe", 10);  
 }  
  
 @AfterEach  
 void tearDown() {  
 // no specific cleanup needed for this test class  
 }  
  
  
 @Test  
 void setName() {  
 String name = "Jane Doe";  
 customer1.setName(name);  
 *assertEquals*(name, customer1.getName());  
 }  
  
 @Test  
 void getLoyaltyPoints() {  
 int expectedLoyaltyPoints = 10;  
 customer1.setLoyaltyPoints(expectedLoyaltyPoints);  
 *assertEquals*(expectedLoyaltyPoints, customer1.getLoyaltyPoints());  
 }  
  
 @Test  
 void setLoyaltyPoints() {  
 int newLoyaltyPoints = 20;  
 customer1.setLoyaltyPoints(newLoyaltyPoints);  
 *assertEquals*(newLoyaltyPoints, customer1.getLoyaltyPoints());  
 }  
  
 @Test  
 void getName() {  
 String expectedName = "John Doe";  
 *assertEquals*(expectedName, customer1.getName());  
 }  
  
 @Test  
 void testToString() {  
 // Test the toString method with different values  
 Customer customer2 = new Customer("Jane Doe", 10);  
 String expectedString = "Customer{" +  
 "name='" + "Jane Doe" + '\'' +  
 ", loyaltyPoints=" + 10 +  
 '}';  
 *assertEquals*(expectedString, customer2.toString());  
 }  
}

package com.company;  
  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.util.Comparator;  
  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class OrderTest {  
  
 private Customer customer1, customer2;  
 private FoodItem foodItem1, foodItem2, foodItem3, foodItem4, foodItem5, foodItem6;  
 private Discount discount1, discount2;  
  
 @BeforeEach  
 void setUp() {  
 customer1 = new Customer("John Doe", 10);  
 customer2 = new Customer("Joanna Mitchell", 500);  
 foodItem1 = new Burger();  
 foodItem2 = new Fries();  
 foodItem3 = new Bangus();  
 foodItem4 = new Hotdog();  
 foodItem5 = new BeefWellington();  
 foodItem6 = new Churros();  
  
 discount1 = new LoyaltyDiscount(customer1); // 20% discount  
 discount2 = new LoyaltyDiscount(customer1); // 3% discount  
 }  
  
 @AfterEach  
 void tearDown() {  
 // No specific cleanup needed for this test class  
 }  
  
 @Test  
 void addFoodItem() {  
 Order order = new Order(customer1);  
 order.addFoodItem(foodItem1);  
 order.addFoodItem(foodItem2);  
 *assertEquals*(2, order.getFoodItems().size());  
 }  
  
 @Test  
 void DecorateWithAllToppings(){  
 foodItem1 = new CremeFraiche(foodItem1);  
 foodItem1 = new Cheese(foodItem1);  
 foodItem1 = new BuffaloSauce(foodItem1);  
 foodItem1 = new Bacon(foodItem1);  
 foodItem1 = new Chili(foodItem1);  
 System.*out*.println(foodItem1);  
 }  
  
 @Test  
 void setDiscount() {  
 Order order = new Order(customer1);  
 Discount discount = new LoyaltyDiscount(customer1); // 20% discount  
 order.setDiscount(discount);  
 *assertEquals*(discount, order.getDiscount());  
 }  
  
 @Test  
 void getTotalCost() {  
 Order order = new Order(customer1);  
 FoodItem foodItem = new Burger();  
 order.addFoodItem(foodItem);  
 double expectedTotalCost = 8.95;  
 *assertEquals*(expectedTotalCost, order.getTotalCost(), 0.01); // allow for slight floating point error  
 }  
  
 @Test  
 void printOrderDetails() {  
 Order order = new Order(customer1);  
 FoodItem foodItem = new Burger();  
 order.addFoodItem(foodItem);  
// order.printOrderDetails();  
 System.*out*.println(order);  
  
 // Assert that the output matches the expected string  
 String expectedOutput = "Order{foodItems=[BaseFoodItem{description='Burger', cost=8.95}], discount=null, customer=Customer{name='John Doe', loyaltyPoints=10}}";  
 *assertEquals*(expectedOutput, order.toString());  
 }  
  
 @Test  
 void testFoodItemToString(){  
 FoodItem foodItem = new Burger();  
 String expectedOutput = "BaseFoodItem{description='Burger', cost=8.95}";  
 *assertEquals*(expectedOutput, foodItem.toString());  
 }  
 @Test  
 void applyDiscount() {  
 Order order = new Order(customer1);  
 FoodItem foodItem = new Burger();  
 order.addFoodItem(foodItem);  
 customer1.setLoyaltyPoints(1000);  
 Discount discount = new LoyaltyDiscount(customer1); // 20% discount  
 order.setDiscount(discount);  
 System.*out*.println(customer1);  
 double expectedTotalCost = 7.16; // after applying 20% discount  
 *assertEquals*(expectedTotalCost, order.getTotalCost(), 0.01); // allow for slight floating point error  
 }  
  
}

package com.company;  
  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class ToppingsDecoratorTest {  
  
 private FoodItem burger;  
  
 @BeforeEach  
 void setUp() {  
 burger = new Burger();  
 }  
  
 @Test  
 void testGetDescription() {  
 // Test that the description is calculated correctly for CremeFraiche  
 CremeFraiche cremeFraiche = new CremeFraiche(burger);  
 *assertEquals*("Burger, Creme Fraiche", cremeFraiche.getDescription());  
  
 // Test that the description is inherited from the decorated food item  
 Cheese withCheese = new Cheese(cremeFraiche);  
 *assertEquals*("Burger, Creme Fraiche, Cheese", withCheese.getDescription());  
 }  
  
 @Test  
 void testGetCost() {  
 CremeFraiche cremeFraiche = new CremeFraiche(burger);  
 // Test that the cost is calculated correctly for CremeFraiche  
 double expectedCost = cremeFraiche.getCost();  
 *assertEquals*(expectedCost, new CremeFraiche(burger).getCost(), 0.01);  
  
 // Test that the cost is inherited from the decorated food item  
 double expectedTotalCost = new Cheese(cremeFraiche).getCost();  
 *assertEquals*(expectedTotalCost, new Cheese(new CremeFraiche(burger)).getCost(), 0.01);  
 }  
  
 @Test  
 void testToString() {  
 // Test that the toString method returns the correct string for CremeFraiche  
 FoodItem burger = new Burger();  
 String expectedString = "Burger, Creme Fraiche, total cost: " + (burger.getCost() + 1.20);  
 *assertEquals*(expectedString, new CremeFraiche(burger).toString());  
 }  
}

MAIN:

package com.company;  
  
public class Main {  
  
 */\*\*  
 \* Implement this system using the decorator design pattern, and test it by creating some food items,  
 \* adding toppings, creating an order, and applying a loyalty discount in the driver program.  
 \** ***@param*** *args  
 \*/* public static void main(String[] args) {  
 Customer customer1 = new Customer("John", 105);  
 Customer customer2 = new Customer("Stan DARSH", 500);  
  
 Order order1 = new Order(customer1);  
 Order order2 = new Order(customer2);  
  
 FoodItem hotdog = new Hotdog();  
 FoodItem bangus = new Bangus();  
 FoodItem burger = new Burger();  
 FoodItem churros = new Churros();  
  
 // decorate hotdog  
 hotdog = new Cheese(hotdog);  
 hotdog = new Chili(hotdog);  
 hotdog = new Bacon(hotdog);  
  
 // decorate burger  
 burger = new Cheese(burger);  
 burger = new CremeFraiche(burger);  
  
 // decorate bangus  
 bangus = new BuffaloSauce(bangus);  
  
  
 // add fooditems to customer order  
 order1.addFoodItem(hotdog);  
 order1.addFoodItem(bangus);  
  
 order2.addFoodItem(burger);  
 order2.addFoodItem(churros);  
  
 // print order before applying loyalty discount using points  
 System.*out*.println("Customer and Orders: \n");  
 order1.printOrderDetails();  
 System.*out*.println();  
 order2.printOrderDetails();  
  
 // add points and discounts to order  
 Discount discount = new LoyaltyDiscount(customer1);  
 order1.setDiscount(discount);  
 Discount discount1 = new LoyaltyDiscount(customer2);  
 order2.setDiscount(discount1);  
  
 // print order after applying loyalty points discount  
 System.*out*.println("\nApplying loyalty discount... \nOrders after discount applied:");  
 order1.printOrderDetails();  
 System.*out*.println();  
 order2.printOrderDetails();  
 }  
}