

**Mohammad Usama**

**20K-0190**

**Design Defects And Refactoring**

**Sec 8-A**

**Assignment 04**

**Question 01**

Imagine the ordering procedure of a fast-food restaurant. Consumers can place separate orders for burgers, fries, drinks, and other culinary items in a variety of sizes and add-ons. Order processing, pricing calculations, and alerting the kitchen to prepare are all handled by the system. The following design patterns are going to be used in this scenario:

1. Factory Method: To produce various food products.

2. Decorator Pattern: This allows food items to be customized (size, extras, etc.).

3. Observer Pattern: To communicate any new orders to the kitchen.

4. Template Method Pattern: Designed to specify a typical order processing procedure.

(1) implement solution for this scenario using at least 4 different design patterns in java language.

1. Factory

public interface FoodItemFactory {

FoodItem createFoodItem(String category);

}

public class BurgerFactory implements FoodItemFactory {

@Override

public FoodItem createFoodItem(String category) {

return new Burger();

}

}

1. Decorator

public class CheeseAddition extends FoodItemDecorator {

public CheeseAddition(FoodItem foodItem) {

super(foodItem);

}

@Override

public String getDisplayName() {

return foodItem.getDisplayName() + " with Extra Cheese";

}

@Override

public double getPriceWithAdditions() {

return foodItem.getPriceWithAdditions() + 1.50; // Assuming extra cheese costs $1.50

}

}

1. Observer

public class BaconAddition extends FoodItemDecorator {

public BaconAddition(FoodItem foodItem) {

super(foodItem);

}

@Override

public String getDisplayName() {

return foodItem.getDisplayName() + " with Bacon";

}

@Override

public double getPriceWithAdditions() {

return foodItem.getPriceWithAdditions() + 2.00; // Assuming bacon costs $2.00

}

}

1. Template

public class SauceAddition extends FoodItemDecorator {

public SauceAddition(FoodItem foodItem) {

super(foodItem);

}

@Override

public String getDisplayName() {

return foodItem.getDisplayName() + " with Extra Sauce";

}

@Override

public double getPriceWithAdditions() {

return foodItem.getPriceWithAdditions() + 1.00; // Assuming extra sauce costs $1.00

}

}

**Document your solution and give a detailed textual descriptions to explain the usage of patterns and reasons that you have used these patterns**

**(1) Factory Method Pattern:**

The Factory Method Pattern simplifies the creation of various types of food items, such as burgers or fries. Each food item has its own unique preparation method. Instead of manually specifying the creation process for each item every time we order, this pattern provides a straightforward way to communicate our preferences to the system, which then handles the specifics accordingly.

**(2) Decorator Pattern:**

Incorporating the Decorator Pattern allows for the addition of extra ingredients or toppings to our food items, like adding extra cheese to a burger or extra sauce to fries. It facilitates customization without altering the core identity of the food. We can seamlessly include multiple extras, and the system gracefully manages all the modifications.

**(3) Observer Pattern:**

Consider the scenario of placing an order where the kitchen needs to be promptly notified to initiate food preparation. The Observer Pattern serves as a means of communication within the system, signaling, "We have a new order!" It fosters interaction between different components without necessitating extensive knowledge of one another's internal workings.

**(4) Template Method Pattern:**

When processing an order, there's a predefined sequence of steps to follow, such as adding extras, computing the total price, and informing the kitchen about the order. The Template Method Pattern streamlines these procedural steps, ensuring they occur in the correct sequence each time. Nevertheless, it also offers flexibility to adjust certain aspects as needed, akin to following a cooking recipe while retaining the ability to tailor elements to individual preferences.

**Compare advantages and disadvantages of each pattern that you have used.**

(1) Factory Method Pattern:

Advantages: Facilitates the seamless addition of new food item types without disrupting existing code. Enhances code organization by encapsulating the creation logic of each food item type within dedicated factory classes. Empowers different components of the program to create specific food item types tailored to their requirements.

Disadvantages: Introducing numerous factory classes for various food item types could potentially complicate the program's structure. Requires careful management to ensure each factory class is responsible for creating only one type of food item.

(2) Decorator Pattern:

Advantages: Enables the addition of extra elements to food items without altering their fundamental nature. Allows for the flexible inclusion of multiple extras in different combinations. Promotes code extensibility by facilitating changes without necessitating extensive

Disadvantages:

- If there are lots of ways to customize our food, we might end up with a lot of small, special classes, which could make things more complicated.

**Alternative Design Patterns and Architectural Improvements**

**Strategy Pattern:**

● Current Approach: Price calculation might be implemented within the Order class.

● Alternative: Use the Strategy pattern. Define separate price calculation strategies for different items (burgers, fries) or sizes (small, medium).