

**PIZZA ORDERING SYSTEM**

**Final Project**

**“Software Design Patterns” course**

**Group: SE-2226**

**Team members: Imanzhusup Saparbek,**

**Teaching Assistant: Arailym Bakenova**

**Astana, 2023**

**Project Overview**

### **Project Idea:**

The project simulates a basic pizza ordering system where users can select different types of pizzas, add extra toppings, choose a payment method, and place an order. It incorporates several design patterns (singleton, strategy, decorator, adapter, factory, and observer) and adheres to SOLID, DRY, and KISS principles.

### **Purpose of the Work:**

The primary purpose of the project is to demonstrate the implementation of various design patterns and coding principles in a Java application. It serves as an educational example to showcase how these patterns can be applied in a real-world scenario, such as a pizza ordering system.

### **Objectives of the Work:**

* **Design Patterns Implementation:**

Demonstrate the use of design patterns like singleton for the PizzaShop, strategy for payment methods, decorator for pizza toppings, adapter for ordering pizzas, factory for pizza creation, and observer for order notifications.

* **SOLID Principles:**

Follow SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion) to ensure a modular and maintainable codebase.

* **DRY (Don't Repeat Yourself):**

Avoid code duplication by creating reusable components and leveraging inheritance and composition where appropriate.

* **KISS (Keep It Simple, Stupid):**

Keep the codebase simple and straightforward, focusing on clarity and readability. Avoid unnecessary complexity.

* **User Interaction:**

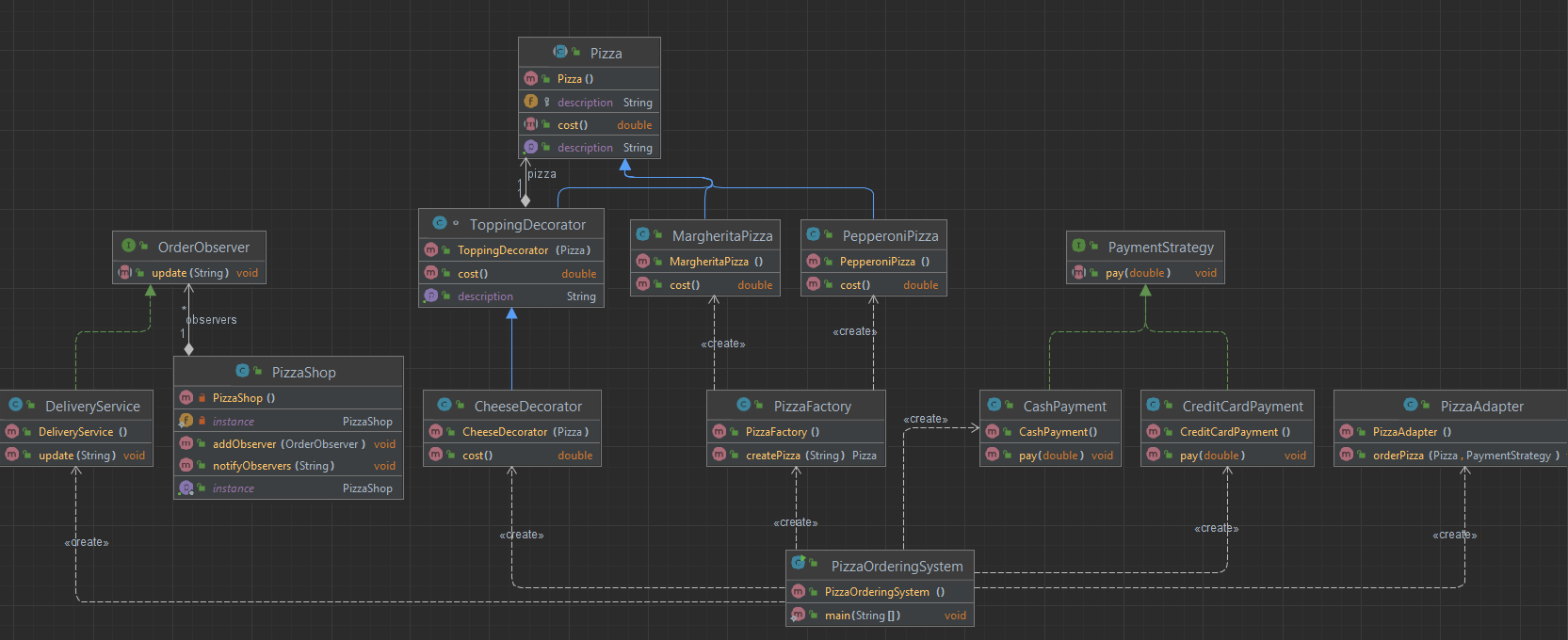
Provide a basic command-line interface for users to interact with the pizza ordering system.

Allow users to select pizza types (e.g., Margherita, Pepperoni), add extra toppings, choose payment methods (credit or cash), and place an order.

* **Observer Pattern for Notifications:**

Utilize the observer pattern to notify external services (e.g., DeliveryService) when an order is placed.

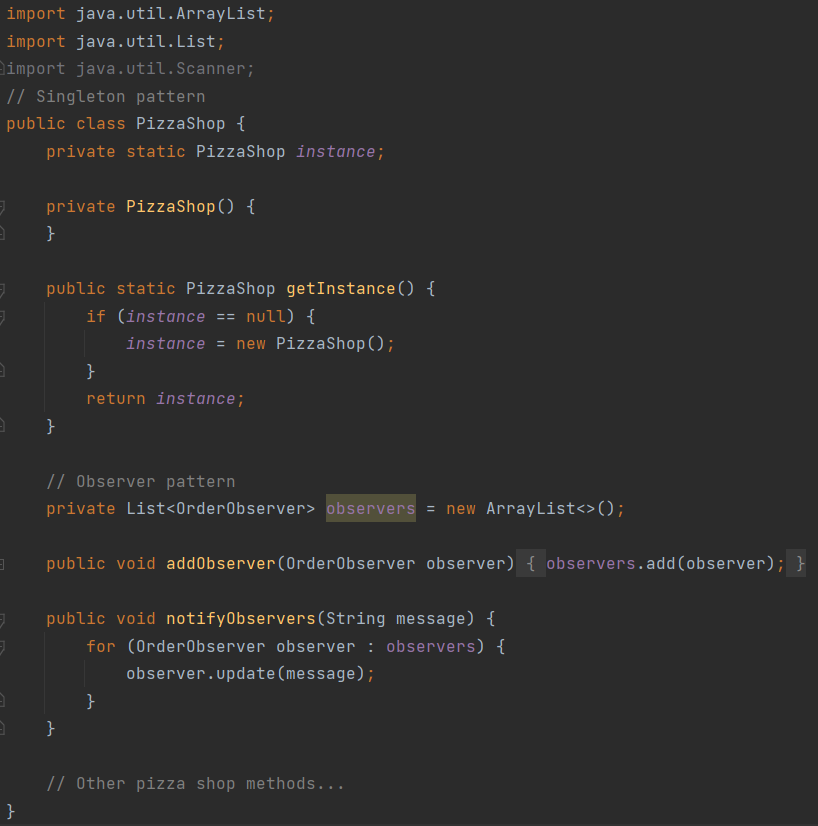
UML DIAGRAM

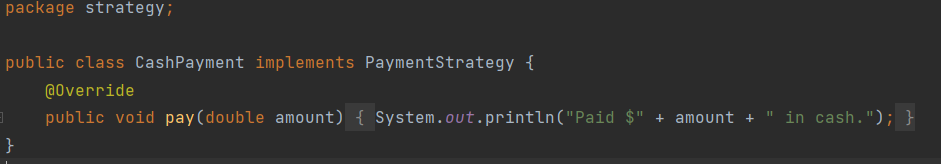


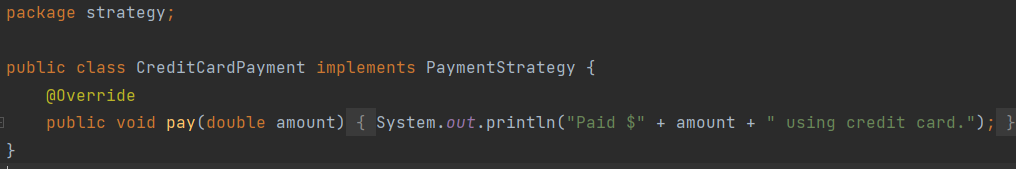
* **Singleton Pattern (PizzaShop):**

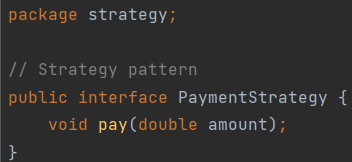
**Explanation:** The **PizzaShop** class follows the singleton pattern, ensuring that only one instance of the pizza shop exists throughout the program. This is achieved by making the constructor private and providing a static method (**getInstance()**) to retrieve the instance.

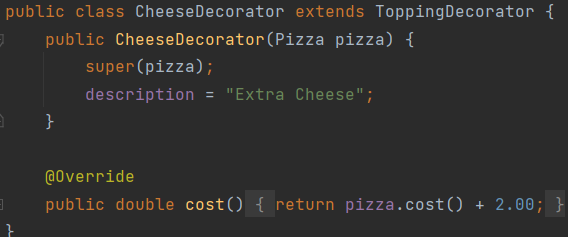
* **Screenshot:**

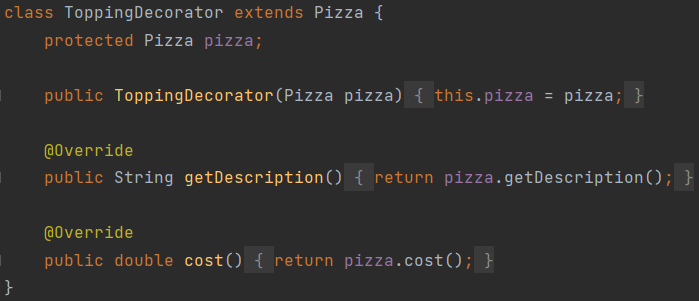


* **Strategy Pattern (PaymentStrategy, CreditCardPayment, CashPayment):**
* **Explanation:** The strategy pattern is used to represent different payment methods. The **PaymentStrategy** interface declares a common method, and concrete classes (**CreditCardPayment** and **CashPayment**) implement this interface. The pizza ordering system can dynamically choose the payment strategy at runtime.
* **Screenshot:** 

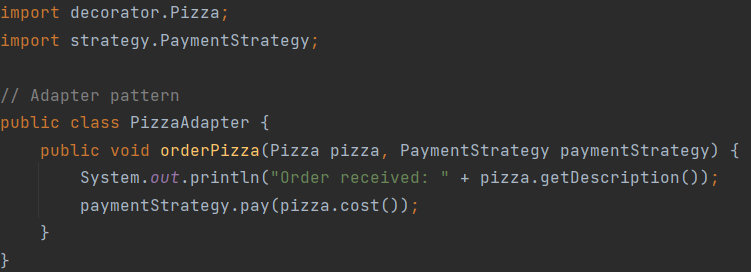




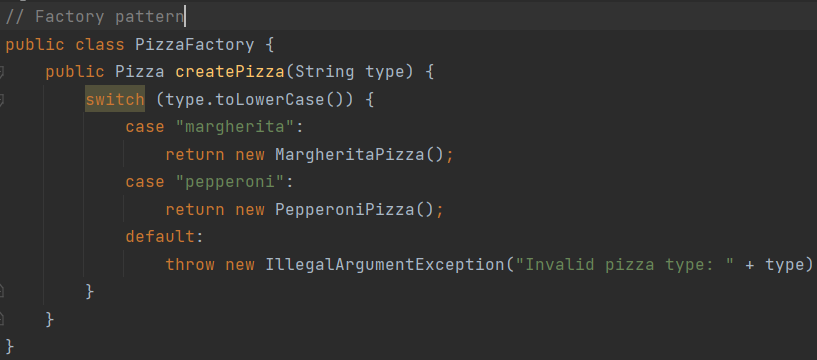
* **Decorator Pattern (Pizza, ToppingDecorator, CheeseDecorator):**
* **Explanation:** The decorator pattern is employed to add extra toppings to pizzas. The **Pizza** class is the base component, and **ToppingDecorator** and **CheeseDecorator** act as concrete decorators. Decorators can be stacked to add multiple toppings to a pizza.
* **Screenshot:** 



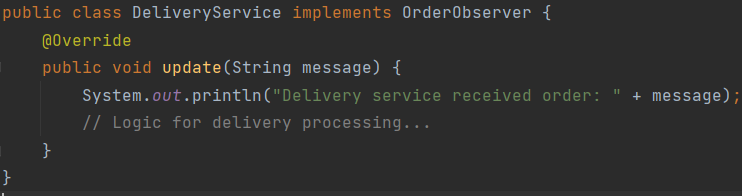
* **Adapter Pattern (PizzaAdapter):**
* **Explanation:** The adapter pattern is used to adapt the existing **PizzaAdapter** class to order pizzas using the **Pizza** and **PaymentStrategy** components. The **orderPizza** method adapts and calls the appropriate methods.
* **Screenshot:**

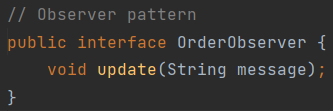


* **Factory Pattern (PizzaFactory):**
* **Explanation:** The factory pattern is implemented in the **PizzaFactory** class to create different types of pizzas based on user input. It centralizes the creation logic and provides a common interface for creating pizzas.
* **Screenshot:**

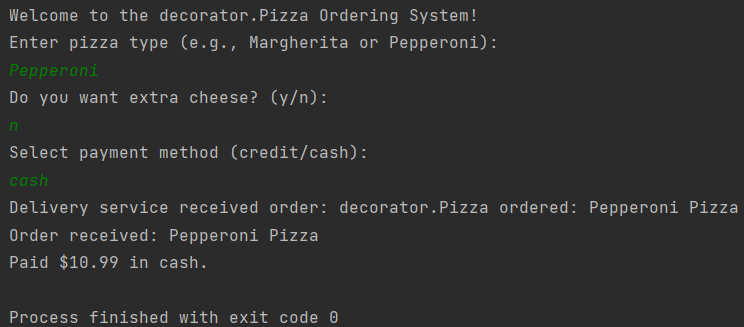


* **Observer Pattern (OrderObserver, DeliveryService):**
* **Explanation:** The observer pattern is used for order notifications. The **OrderObserver** interface declares the **update** method, and the **DeliveryService** class implements it. The pizza shop (**PizzaShop**) notifies observers (e.g., delivery service) when an order is placed.
* **Screenshot:**





* **User Interface (Console Interaction):**
* **Explanation:** The program provides a simple command-line interface for users to interact. Users can input the type of pizza, choose toppings, select a payment method, and place an order. The user's choices are processed to create a pizza and complete the order.
* **Screenshot:**



**Key Points of the Project:**

* **Design Patterns Used:**

Singleton Pattern: Ensured a single instance of the **PizzaShop**.

Strategy Pattern: Implemented payment strategies (**CreditCardPayment** and **CashPayment**).

Decorator Pattern: Added extra toppings to pizzas using decorators (**CheeseDecorator**).

Adapter Pattern: Adapted the **PizzaAdapter** class for ordering pizzas.

Factory Pattern: Centralized pizza creation logic in the **PizzaFactory**.

Observer Pattern: Notified external services, like **DeliveryService**, when an order is placed.

* **Project Outcomes:**

Successfully demonstrated the application of various design patterns in a simplified pizza ordering system.

Created a modular and extensible codebase by adhering to SOLID principles, DRY, and KISS.

* **Challenges Faced:**

Adapting the code for a user-friendly GUI was not addressed due to the nature of the console-based example.

Balancing simplicity with the demonstration of design patterns and principles posed a challenge.

* **Insights:**

Design patterns provide effective solutions to recurring design problems and enhance code maintainability and flexibility.

SOLID principles and best coding practices contribute to the creation of clean, readable, and modular code.

* **Future Improvements:**

Implementation of a graphical user interface (GUI) using JavaFX or Swing for a more interactive user experience.

Integration with a database to store and retrieve order information.

Expansion of pizza types, toppings, and payment methods for a more comprehensive system.

In conclusion, the project successfully showcased the integration of various design patterns in a pizza ordering system, following SOLID principles and best practices. The insights gained emphasize the importance of design patterns in creating scalable and maintainable software. Future improvements could focus on enhancing user interaction, extending functionality, and incorporating additional features for a more realistic pizza ordering experience.