SWENG304: Software Design and Architecture
Shopping Mall Java Project

Yahya AlAyaseh
Supervisor: Dr. Yousef Hassouneh

Second Semester 2024/2025

Project Overview

The Java Mall System is a command-line application designed to demonstrate mastery of object-oriented programming and software design using **12+ essential design patterns**. The system simulates a real-world shopping mall scenario where users can:

- Browse and enter stores
- View and add items to a cart
- Choose and switch payment methods
- Save/restore their cart state
- View and leave reviews
- Simulate order lifecycle transitions

The system is built with a **modular architecture**, **SOLID principles**, and is entirely coded in Java.

Requirement Fulfillment

Requirement	**Implementation Details**
One interactive customer	CLI prompts user for name to initialize session
Three stores	BookWorld, ShoeZone, GameSpot created via Factory + Singleton
At least five items per store	Each store is initialized with 5 unique items
Fully interactive CLI interface	Supports full flow: navigation, cart, checkout, reviews, order simulation
Implementation of the design patterns	Total of 12 patterns correctly implemented and demonstrated
Solve the requested questions on refactoring the code of the Shopping Cart	Code refactoring, and answer the requested questions
Create a UML diagram after refactoring the code.	Draw a class diagram that shows the relationships between classes and the changes made after implementing design patterns in the code.
Create Unit Tests to test the System.	All unit test files were created to cover possible scenarios of the system; most of the tests were created during development.

Design Pattern Tracker Table (with Explanation)

Each design pattern and its related class are included in an inner package named after the respective design pattern. For the Iterator and Singleton patterns, since they are used in multiple places, I have included comments to indicate where they are implemented.

Design Pattern	**Where Implemented**	**Purpose & Implementation Detail**
Singleton	`BookStoreFactory`, `ShoeStoreFactory`, etc.	Restricts each store factory to one instance via private constructor + static `getInstance()`
Factory Method	`StoreFactory`, `createStore()`	Enables polymorphic store creation without exposing instantiation logic
Abstract Factory	Book/Shoe/Game Store Factories	Groups store creation logic and encapsulates the families of related objects
Iterator	`Mall.customers()`, `Store.items()`	Provides external access to internal lists via `Enumeration`
Strategy	`PaymentStrategy`, `PayPalPayment`, `CreditCardPayment`	Enables runtime choice of payment strategy (`cart.setPaymentStrategy(...)`)
Observer	`CartObserver`, `EmailNotifier`, `ShoppingCart`	Observers are notified (e.g., email) when cart changes (add/remove/checkout) occur
Memento	`CartMemento`, `CartHistory`, `ShoppingCart`	Captures and restores internal cart state on user command (undo/restore cart)
Command	`OrderCommand`, `PlaceOrderCommand`, `CancelOrderCommand`	Encapsulates order requests and queues them in `OrderManager` to allow flexible execution
State	`OrderContext`, `NewState`, `PaidState`, etc.	Controls order behavior based on its state — transitions handled cleanly through polymorphism
Proxy	`ReviewServiceProxy` wraps `ReviewServiceImpl`	Validates access (auth check) before delegating review requests
Decorator	`DiscountDecorator`, `PercentageDiscount`, `FixedDiscount`	Wraps `ItemComponent` to modify price dynamically with discounts

Chain of Responsibility	`DiscountHandler`, `BlackFridayDiscount`, `CouponDiscount`	Chains discount rules to apply layered pricing logic

Running the Application

Requirements

- Java 17 or later

Compile & Run

All files are inside the `shoppingmall` package:

```
javac -d bin src/shoppingmall/**/*.java
java -cp bin shoppingmall.Main
```

CLI Functional Flow

- Prompt: Enter user name (Customer)
- List and enter stores (Factory, Singleton)
- View store inventory and add items (Iterator, Observer)
- Save and restore cart state (Memento)
- Choose and apply payment method (Strategy)
- Checkout with observer notification
- View or post reviews (Proxy)
- Apply discounts via Decorator + Chain of Responsibility
- Simulate full order lifecycle (State Pattern)

Directory Structure

```
shoppingmall/               # Core system logic and domain
 factories/               # Factory pattern implementations
 payment/                 # Strategy pattern - Payment systems
 observer/                # Observer pattern - Notification system
 command/                 # Command pattern - Order actions
 memento/                 # Memento pattern - Cart save/restore
 state/                   # State pattern - Order lifecycle
 proxy/                   # Proxy pattern - Review access
 discount/                # Decorator & CoR patterns - Discounts
 Main.java                # CLI entry point
test/
 CommandPatternTest.java
 MementoTest.java
 ObserverTest.java
 OrderStateTest.java
 PaymentStrategyTest.java
 ProxyTest.java
 ShoppingMallPatternsTestSuite.java
```

Other Files:

Mock Main Files: This folder contains several Main.java files that can be copied to replace the Main.java file. I wrote these files while building the system to test the scenarios.

Class Diagram Comparison

This UML class diagram provides a comprehensive overview of the differences between the initial code and the final code implementation. It highlights the structural changes and enhancements made throughout the development process, facilitating a better understanding of the evolution of the codebase.

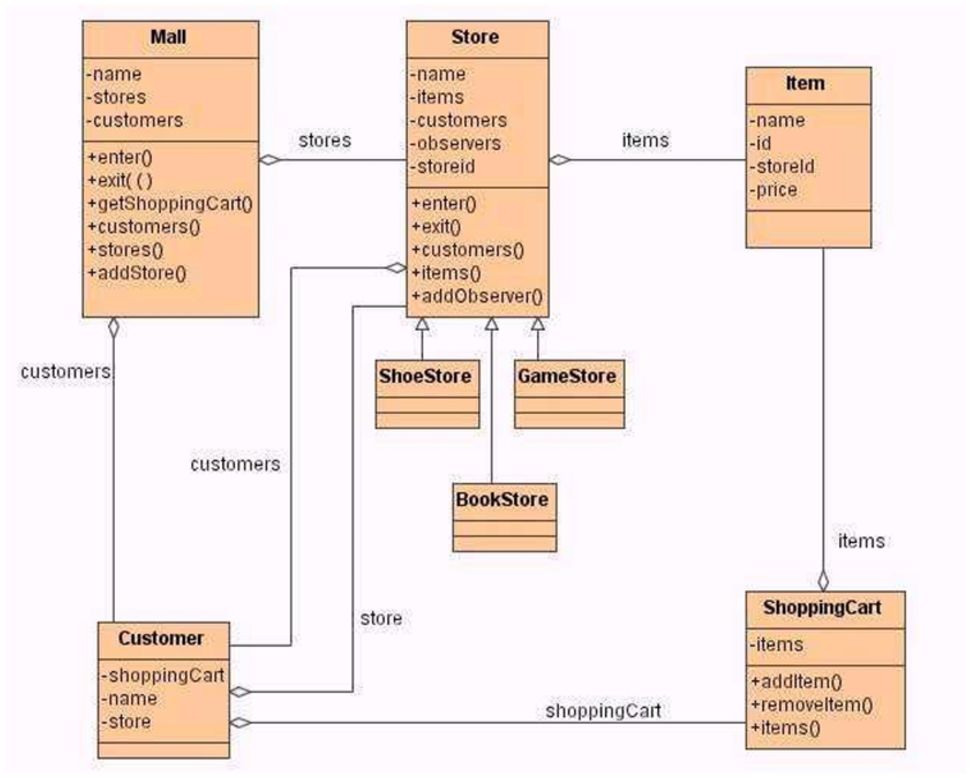

Figure 1: Class diagram before applying design patterns

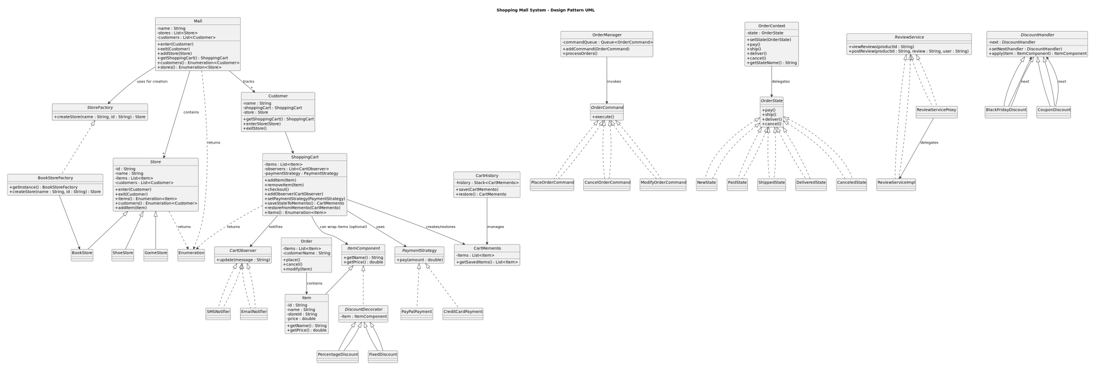

Figure 2: Class diagram after applying design patterns

View Diagram Image

(If image is not displaying properly, click the link above)

Conclusion

The *Java Mall System* project provided a rich opportunity to explore and apply core object-oriented design patterns in a real-world simulation. Through the development and refactoring process, the system evolved from a tightly coupled and limited design into a flexible, modular architecture guided by SOLID principles.

A total of **12 design patterns** were implemented, each chosen to solve a specific design concern. Patterns like **Strategy** and **Observer** were applied to make payment handling and notifications dynamic and easily extendable. Others, like **Command**, **State**, **Proxy**, and **Memento**, brought structure, reusability, and runtime behavior control to the system.

Refactoring efforts significantly improved the system's maintainability and testability. Components such as payment methods and notifiers are now interchangeable and independently testable. The introduction of unit tests further ensured that each part behaves correctly in isolation and in full integration.

This project not only strengthened my technical skills but also deepened my appreciation for clean architecture. It demonstrated how design patterns can transform code into a scalable and robust foundation, ready for future growth.

In summary, the Java Mall System is now a comprehensive example of well-structured, pattern-driven Java software. It reflects thoughtful design choices and provides a strong foundation for educational use, extension, or deployment.

These patterns were chosen and applied strategically to:

- Improve maintainability
- Allow runtime flexibility (strategy, state)
- Enable clear separation of concerns (command, factory)
- Provide scalable architecture (observer, decorator)

Let good design guide our code!