BIRZEIT UNIVERSITY

Faculty of Engineering and Technology
Master of Software Engineering (SWEN)

SWEN6304: Software Design and Architecture
First Semester 2024/2025

Instructor: Dr. Yousef A. Hassouneh Masri 322

Design Pattern Project
Due date is 25/04/2025

Project Description

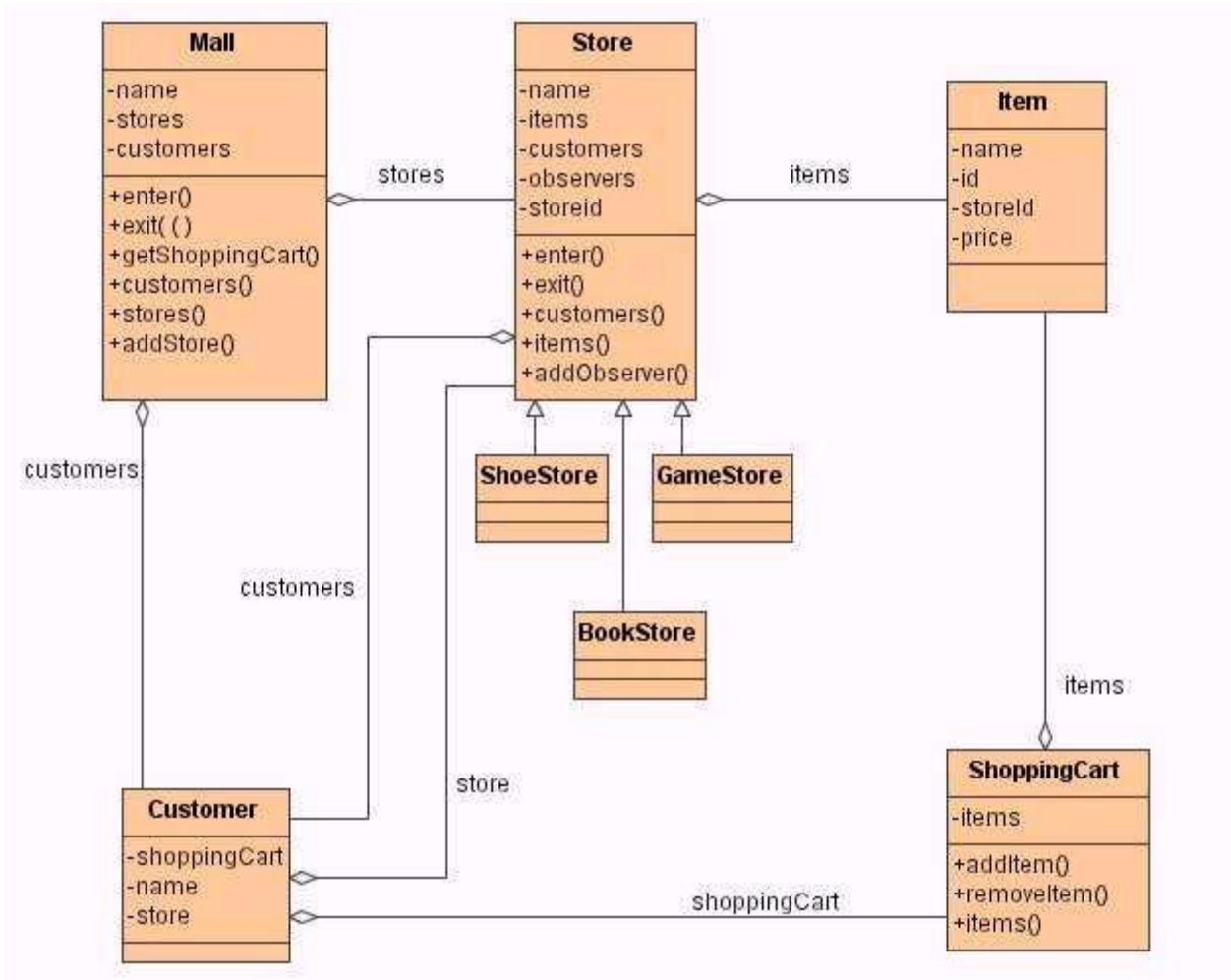
In this project, you will be implementing an online shopping mall in **Java**. The system allows users to browse, search, and purchase shopping items. You'll apply various design patterns to improve the software's architecture and maintainability. Here are some design patterns you can integrate into this project:

- Iterator, methods that return an Enumeration demonstrate the Iterator pattern.
- Abstract Factory or Factory Method (or both!) Create all stores using factories.
- Singleton allows only one instance of any factory type you create.
- Product Catalog (Factory Method Pattern): The product catalog will consist of various types of products, such as electronics, clothing, and books. The Factory Method pattern will be used to create different product instances while centralizing the creation logic.
- Shopping Cart (Observer and Memento Patterns): Users can add and remove items from their shopping carts. The Observer pattern will be employed to notify the cart when items are added or removed. The Memento pattern will enable users to save and restore the state of their shopping carts.
- Payment Processing (Strategy Pattern): Users can make payments using different methods (e.g., credit card, PayPal, Bitcoin). The Strategy pattern will be used to switch between different payment gateways easily.
- Order Management (Command Pattern): Users can place orders, modify them, and cancel them. The Command pattern will be applied to encapsulate these operations and manage the order lifecycle effectively.
- Discounts and Promotions (Decorator and Chain of Responsibility Patterns): Discounts and promotional offers can be added to products in the catalog. The Decorator pattern will be used to

dynamically add features (discounts) to products. The Chain of Responsibility pattern will apply multiple discounts in a specific sequence.

- Shipping and Order Tracking (State and Observer Patterns): The system will manage the order's lifecycle, including order placement, payment processing, and shipping. The State pattern will be used to represent each order's different states. The Observer pattern will allow users to track their order status in real-time.
- Reviews and Ratings (Proxy Pattern): Users can leave product reviews and ratings. The Proxy pattern will control access to review data to ensure that only authorized users can view or modify reviews.
- User Notifications (Observer Pattern): Users will receive notifications about order updates, promotions, and other relevant information. The Observer pattern will be employed to manage these notifications effectively.

The class diagram which provides the framework for the shopping mall is as follows:



Here's a brief description of each class, note that required accessors and mutators are not listed in the above for the sake of brevity.

- Mall
  - The mall itself.
  - Notable Attributes:
    - name - the name of the mall
    - stores - a collection of stores of different types
    - customers - the customers currently in the mall
  - Possible Methods:
    - void enter(Customer c) - customer c enters the mall
    - void exit(Customer c) - customer c exits the mall
    - ShoppingCart getShoppingCart - returns an empty shopping cart

        - Enumeration customers() - returns an enumeration of the customers in the mall
        - Enumeration stores() - returns an enumeration of the stores in the mall
        - void addStore(Store s) - add a store to the mall

- Store
    - Abstract superclass for a store
    - Notable Attributes:
        - name - the name of the store
        - id - unique ID for the store
        - items - items available for sale in the store
        - customers - the customers currently in the store
        - observers - the observers of the store
    - Possible Methods:
        - abstract void enter(Customer c) - customer c enters the store
        - abstract void exit(Customer c) - customer c exits the store
        - Enumeration customers() - returns an enumeration of the customers in the store
        - Enumeration items() - returns an enumeration of the items available for sale in the store
        - void addObserver(Osberver o) - add an observer to the store
- BookStore
    - A possible subclass of Store
- ShoeStore
    - A possible subclass of Store
- GameStore
    - A possible subclass of Store
- Item
    - An item for sale in a store
    - Notable Attributes:
        - name - the name of the item
        - id - unique ID for the item
        - storeId - the ID of the store from which the item came
        - price - the price of the item
- Customer
    - A customer!
    - Notable Attributes:
        - name - the name of the customer
        - shoppingCart - the shopping cart being used by the customer
        - store - the store the customer is currently in

- ShoppingCart
    - A shopping cart for the customer
    - Notable Attributes:
        - items - items currently in the shopping cart
    - Possible Methods:
        - Enumeration items() - returns an enumeration of the items currently in the cart
        - void addItem(item) - add an item to the shopping cart
        - void removeItem(item) - remove an item from the shopping cart
        - void checkout() – make the payment and notify the customer.

Assume you have given the following implementation for Shopping Cart:

```
public class ShoppingCart {
    private List<Item> items = new ArrayList<>();

    public void addItem(Item item) {
        items.add(item);
        System.out.println("Item added: " + item.getName());
        sendNotification("Added " + item.getName());
    }

    public void checkout() {
        processCreditCardPayment();
        sendNotification("Order placed.");
    }

    private void sendNotification(String message) {
        System.out.println("Email to user: " + message);
    }

    private void processCreditCardPayment() {
        System.out.println("Paid with Credit Card");
    }
}
```

Questions

- Review the given code for ShoppingCart class, and list at least three problems that would occur if you tried to support additional payment methods or notification types.
- Refactor the given implementation using the Strategy pattern for payment methods and the Observer pattern for notifications. Implement at least two payment strategies and two observer types. Then, demonstrate that the cart can dynamically switch behaviors at runtime.
- Answer the following questions:

1. How did using design patterns improve the flexibility of your code?
2. Would this design make testing easier? Why or why not?

Project Requirements

Use the above class diagram to write a working version of the shopping mall.

Your project must have at least one customer, three stores and five items for sale at each store.

Implement the user interface any way you like, either textual or graphical.

Your project must implement the above design patterns:

Deliverables

- Well-documented source code (.java files), clearly **comment on each pattern used** in code:

```
// Strategy Pattern: This class allows switching payment methods dynamically
public class PaymentContext {
    ...
}
```

- Follow clean coding and SOLID principles
- Runnable JAR file, provide a readme text file, explain how to run your project from the console or terminal
- Implement unit tests for key pattern behavior.
- Short Report (max 8 pages) including:
    - Pattern Tracker Table example below:

Pattern	Where Used	Why Used	Classes Involved
Observer	Store, ShoppingCart	Notify on updates	Store, Customer, Cart
Strategy	Payment	Switch payment types	PaymentContext, PayPal, etc

- Updated UML Class Diagram with pattern annotations *that depict the use of the Design Patterns.*
- Submit a short video walkthrough (3-5 minutes) explaining your implementation and design decisions.