CSCI-5448 Project

Name: Jinpeng Miao
Title: Magazine Shopping System
GitHub Link: https://github.com/JinpengMiao/CSCI-5448_Project

Project Description: This project focuses on creating an online magazine shopping system that simulate a real-world shopping system allowing customers to purchase various types of magazines based on their tastes and keeping track of their customers and their corresponding order situation. A database will be created to store magazines and users' information. There are two main roles: customers and admin, who have different privileges. Admin can add or remove customers and add or remove magazines. Customers can search magazines directly and view magazines' detailed information including type, name, publishing date, price. Customers also can register to add magazines to their own shopping cart and purchase magazines. Additionally, once there are new magazines added into the system, all the customers will be notified.

Features that were implemented:

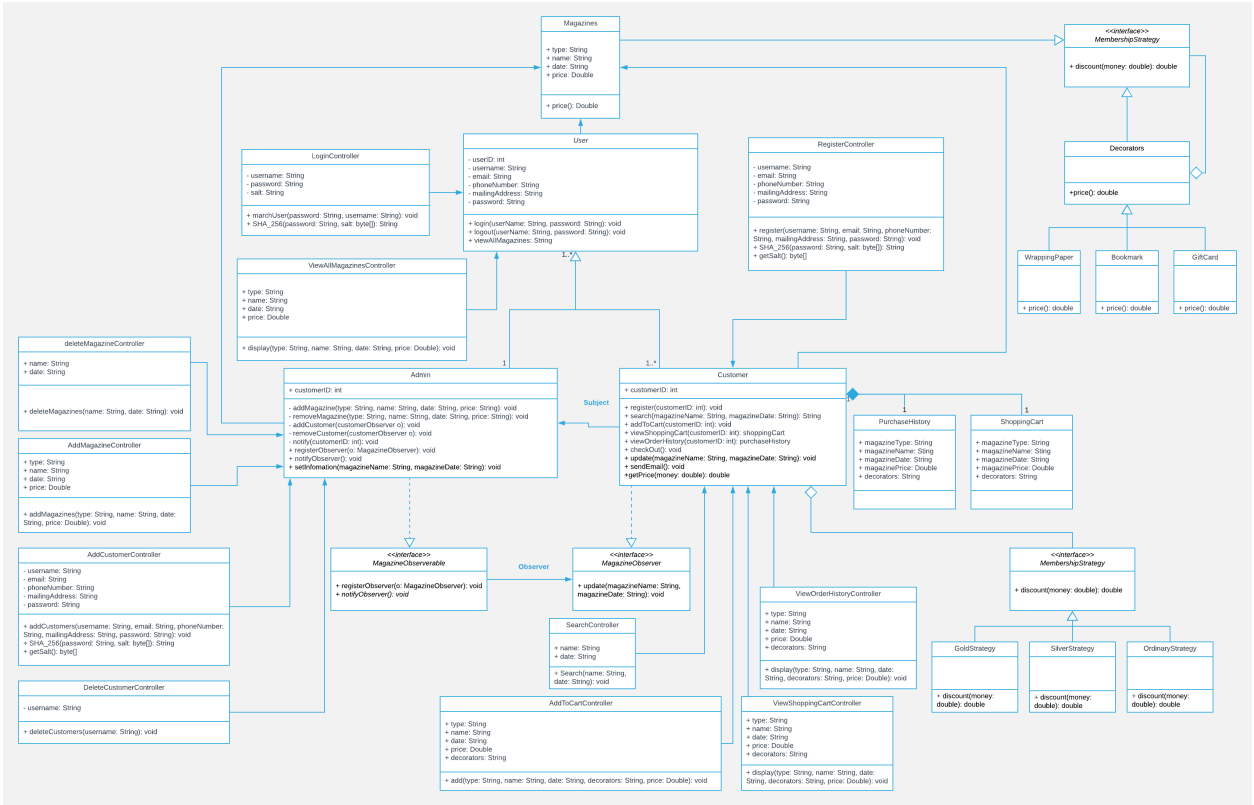
ID	Title
PR-01	Everybody can view magazines.
PR-02	Customers can sign up.
PR-03	Customers and admin can login.
PR-04	Customers and admin can logout.
PR-05	Customers and admin can search for magazines.
PR-06	Customers can add magazines to their own shopping cart.
PR-07	Customers can view their own shopping cart.
PR-08	Customers can add "decorators" (wrapping paper, bookmark, gift card) before they checkout
PR-09	Customers can check out.

PR-09	Customers can view their own order history.
PR-10	Admin can add or remove customers.
PR-11	Admin can add or remove magazines.
PR-12	Admin can notify all customers when new magazines added.
PR-13	Use “salt” to secure the login system.

Features that were not implemented:

1. I implement more than the requirements I proposed before. The only one I want to implement but have not implemented is that block the login if 5 failed attempts in less than 1 hour.

Final class diagram:

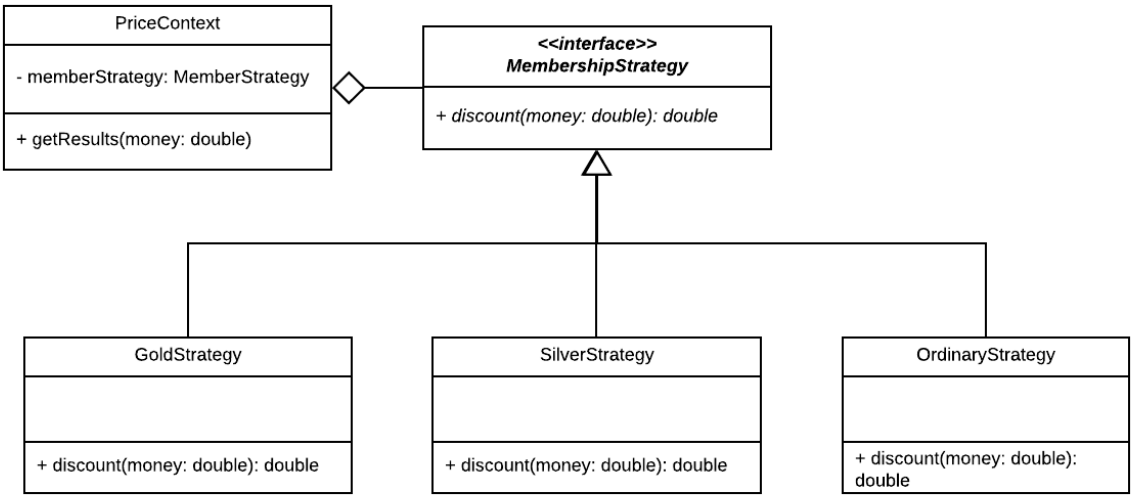


What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

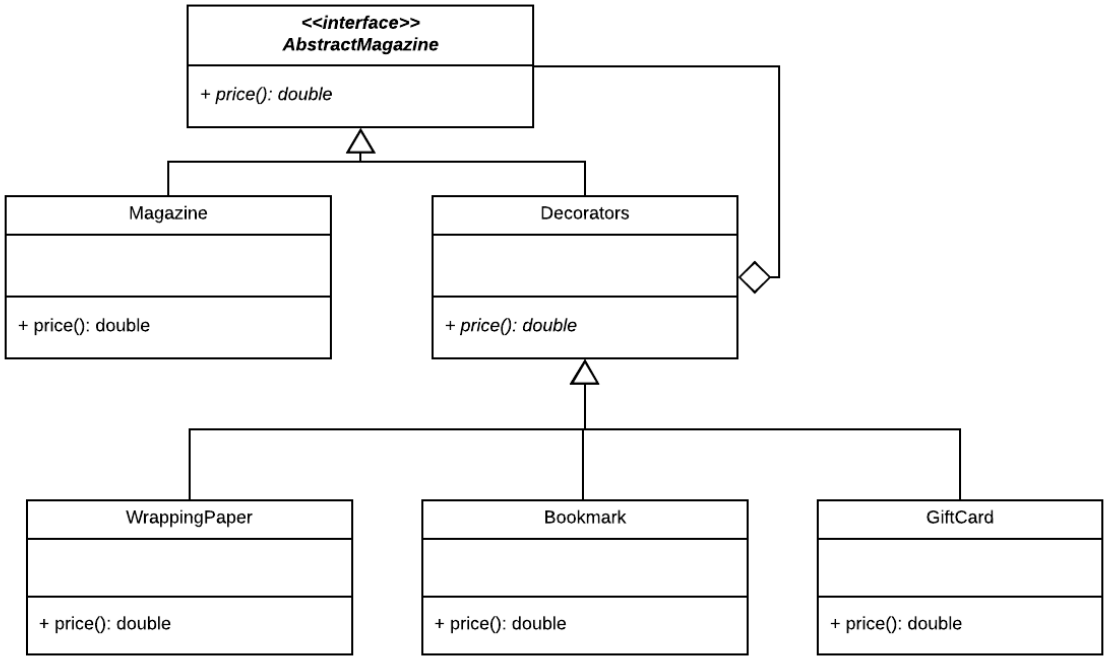
It does change a lot. Since I use MVC framework to complete this project. Previously, there is no controller classes added. Currently, I add all the controller classes here to make it more integrity and understandable. And I use more design patterns (Observer pattern Strategy & Factory pattern, Decorator pattern), all of which make this project more flexible and reusable. Additionally, I also add more classes here since this way can make the whole project more real-world and reasonable. Analyzing and designing the system from a subjective angel in advance is very important. Also, using design patterns reasonably is extremely helpful to implement the system.

Design pattern:

- **Strategy Pattern:**
  - **Classes:** MembershipStrategy, PriceContext, GoldStrategy, SilverStrategy, OrdinaryStrategy.
  - **How:** Customers have different membership levels: Gold membership, Silver membership, and Ordinary membership. Different discount strategies based on different membership levels which respectively are corresponding to different discounts (20% off, 10% off, and no discount). MembershipStrategy is implemented by GoldStrategy, SilverStrategy, and OrdinaryStrategy. PriceContext is a client to get the discounted price.
  - **Why:** I use strategy pattern here since different discounts are implemented by different algorithms. This design pattern makes algorithm independently from clients that use it. It is more flexible and reusable.
  - **Class Diagram:**

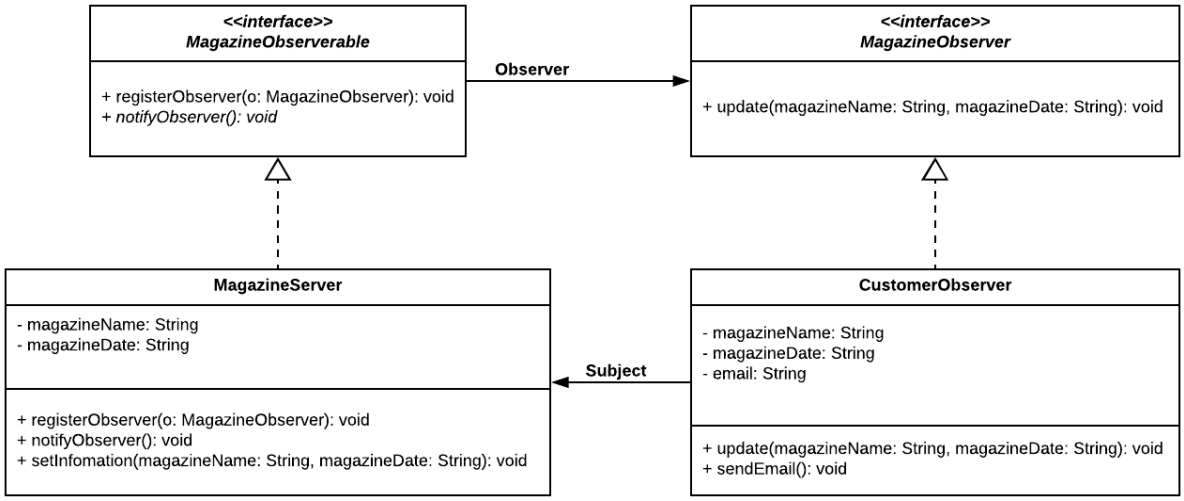


- **Decorator:**
  - **Classes:** AbstractMagazine, Magazine, Decorators, WrappingPaper, Bookmark, GiftCard
  - **How:** Customers can choose to add 3 “decorators” (wrapping paper, bookmark, and gift card) into their shopping carts. Each time a decorator is selected, the corresponding price will be added. AbstractMagazine is implemented by Magazine and Decorators. WrappingPaper, Bookmark, and GiftCard extend Decorator.
  - **Why:** These decorators are used to extend the functionality of magazine shopping system. It is beneficial for promotion and it is more convenient for customers to buy related product. Decorator pattern is used to add additional responsibilities dynamically to magazines.
  - **Class Diagram:**



- **Observer:**
  - **Classes:** AddMagazineController, MagazineObserverable, MagazineServer, MagazineObserver, CustomerObserver.
  - **How:** The admin has right to add new magazines. Whenever there is a new magazine, all the subscribed customers will be notified with emails. This would be beneficial for promotion. MagazineObserverable and MagazineObserver are implemented by MagazineServer and CustomerObserver, respectively.
  - **Why:** Magazine subscription is a one-to-many system feature, which is exactly in line with observer pattern, which is defined as that when one object changes state, all its dependents will be notified.

- **Class Diagram:**



What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

I have learned that paying more attention on designing the system before implementation is very necessary, which is not only good for the integrity of the system, but also beneficial for reusability, flexibility and beauty, especially for large systems. In this case, UML is far more important than what we can imagine, since UML can help us sort out ideas, which makes us implement the system quickly and successfully. And whoever wants to reuse the code, he can easily understand the architecture of the software, and quickly utilize it. More importantly, design patterns are really useful to implement complicated system, since as we all know, lots of code pieces are exactly same. Thus, reusability and flexibility make design patterns extremely important. However, if you don't understand the design pattern very deeply and use it hastily, it may cause many problems. Therefore, I need to understand the design pattern well first and practice a lot. Then in the future work, I will think about writing programs with more design patterns reasonably. Last but not lease, after implementation, refactoring is really helpful to perfect the code.