Sell_Fort

- **Sell with Comfort**

Team

1. Avinash Ratnavel Maharaj
2. Srivishnu Alvakonda
3. Monika Tak

Title
Sell-Fort (Sell with Comfort!!)

Actors
Seller, Buyer, Admin

Project Summary
Designing an Object Oriented java *desktop* application using which sellers can find buyers and vice versa. Sellers will post their used products with a desirable cost, buyers will then have the ability to put a bid on the products. Based on the highest bid, the product will be sold to the respective buyer. Any user logged in to the application can either be a seller or a buyer.

1. List the Features that were implemented:

ID	**Requirement**	**Topic Area**	**User**	**Priority**
US – 01	User should be able to login using credentials	Authentication	All	Medium
US – 02	User should be able to register using a valid email address and password.	Authentication	All	Critical
US – 03	Seller should be able to post an item for sale	Posting an item	Seller	High
US - 04	Seller should be able to add description for the item	Posting an item	Seller	Medium

US – 05	Seller should be able to view his items added for sale	Posting an item	Seller	Medium
US – 06	Seller should be able to view his message inbox	Select winner of the bid	Seller	Medium
US -- 07	Seller should be able to select the winner of the bid	Select winner of the bid	Seller	High
US - 09	Seller should be able to notify the buyer who won the bid	Select the winner of the bid	Seller	medium
US – 10	Buyer should be able to view items by category	View Items	Buyer	High
US – 11	Buyer should be able to add an item to his wish list	Wish list creation	Buyer	High

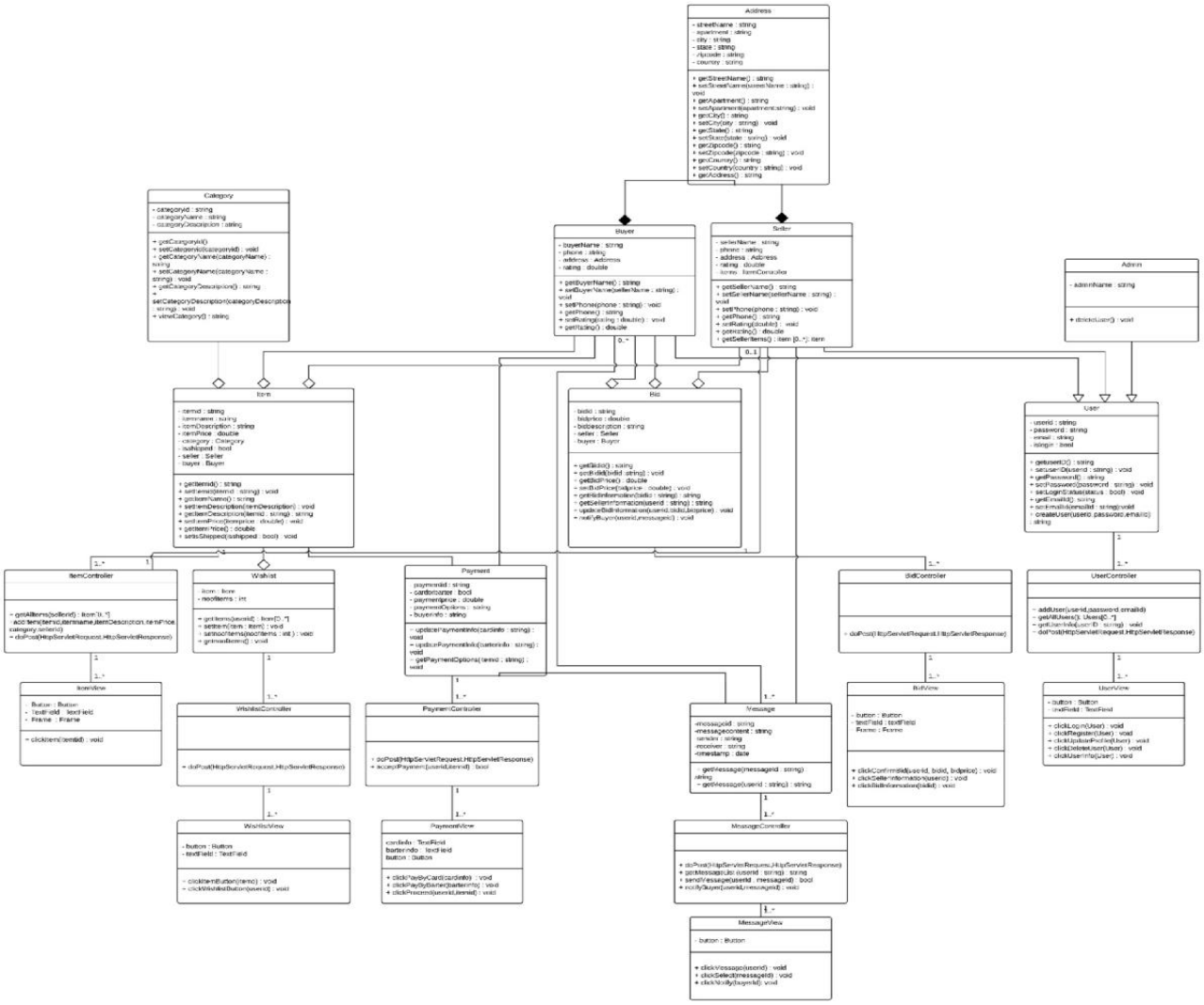
US – 12	Buyer should be able to view bid information	Posting a bid	Buyer	High
US - 13	Buyer should be able to view seller's information	Posting a bid	Buyer	High
US – 14	Buyer should be able to post a bid price for an item	Posting a bid	Buyer	High
US – 16	Buyer should be able to pay for the item	Payment	Buyer	High
US – 17	Buyer should be able to choose payment options	Payment	Buyer	High
US – 18	Seller should be able to accept payment by barter and send message	Payment	Seller	High

2. List the Features were not Implemented from Part 2:

ID	Requirement	Topic Area	User	Priority
US – 19	Admin should be able to delete a user	Authorization	Admin	Medium
US-20	Admin should be able to verify a user	Authorization	Admin	Low
US-15	User should be able to rate another user	Profile	Seller, Buyer	Low
US-21	Admin should be able to change the role of a user	Authorization	Admin	Low
US-08	Seller should be able to view Buyer Information	Select the winner of the bid	Seller	Medium

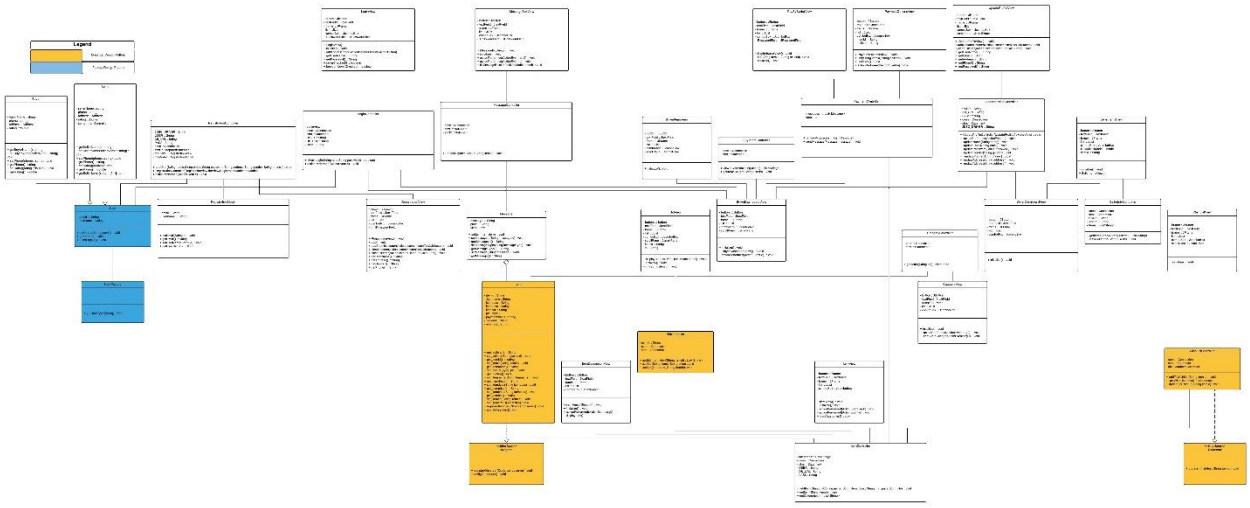
3. **Show your Part 2 class diagram and your class diagram. What changed? Why?**

Old Class Diagram:



Class diagram link: https://github.com/avra0601/CSCI-5448-OOAD-project/blob/master/Class%20Diagram.jpeg

New Class Diagram:



Class diagram link:

https://github.com/avra0601/CSCI-5448-OOAD-project/blob/master/Final%20Class%20Diagram%20-%20Page%201.jpeg

Why?

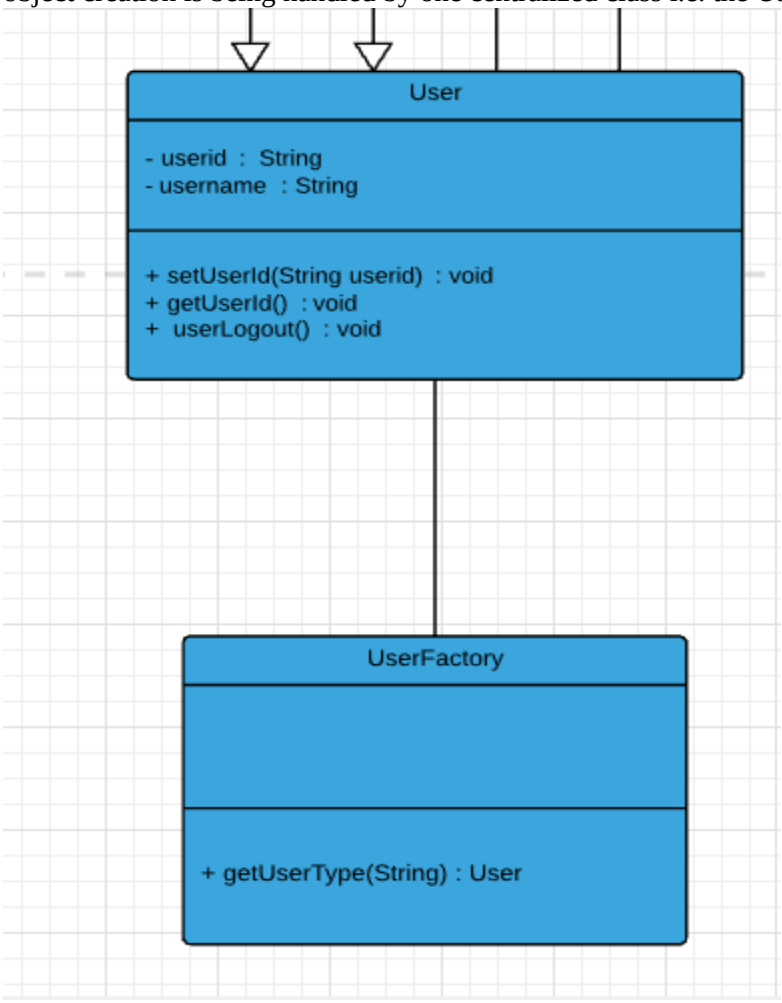
- We observed many changes,
    1. We have added few more classes to obtain the required implementation.
    2. We have removed few classes as we felt they are not required.
    3. We have changed the relationships between classes as we have progressed thought the implementation.
    4. Initially we were not sure of MVC implementation. So our class diagram included an empty Controller. But when implementing we add many methods in Controller class.
    5. We initially thought we will require a bid class but later we removed it
    6. While implementing, we came to know exactly what attributes we wanted to use. So we included more attributes.
    7. We have implemented Factory and Observer design patterns

4. Did you make use of any design patterns in the implementation of your final prototype?

Following are the design patterns we implemented:

Factory Method Design Pattern:

In our design, we have two types of users: Buyer, Seller. We introduced a UserFactory class which controls the instantiation of these users and refers to the newly created object using a common interface. For instance, in the future if we wish to extend the type of users and want to introduce a primeSeller/primeBuyer class, then the code modification will be easier because the object creation is being handled by one centralized class i.e. the UserFactory class.

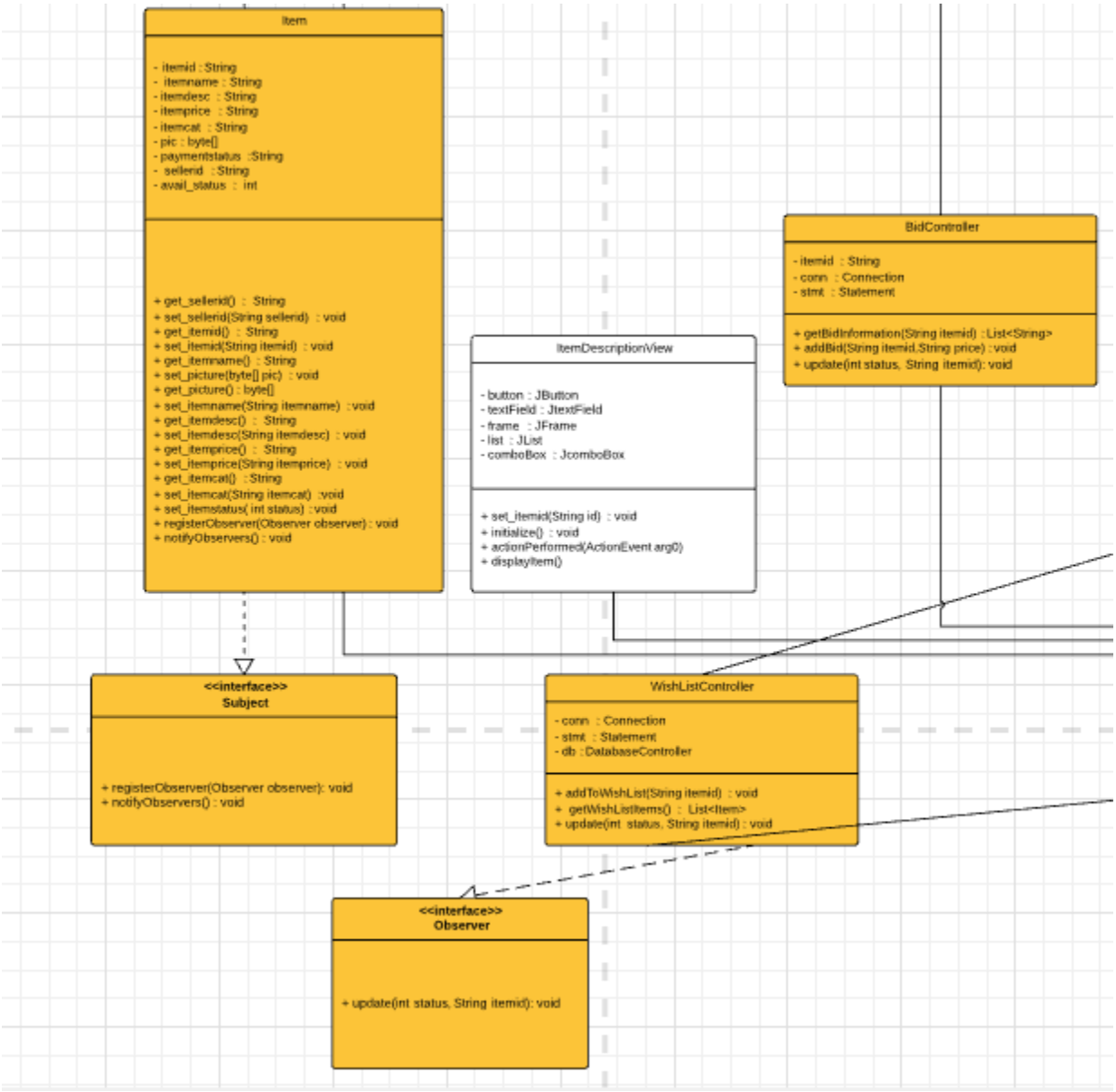


Observer Design Pattern:

In our design, we implemented Observer design pattern wherein we treated out Item class as the subject which was being observed and the WishListController class and the BidController class as our Observer classes which are observing item.

Whenever an item is sold, the status of the item changes from 1 → 0 (available to sold). In the set_status function of the Item class we call the notifyObservers() function which notifies the Observer classes. The WishController class when notified deletes the items from all the other users wish list which have been sold. The BidController class when notified deletes the items from all bid model which have been sold.

Subject ≪Interface≫
Observer ≪ Interface≫
WishListController implements
Observer
BidController implements Observer
Item implements Subject



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

Through the process of design an analysis , we were able to implement a more robust software.
> Building an initial prototype of the design and the conitnuosly refactoring it depending on the requiremnets , helped us in making a better design of our software.
> the UML diagrams helped us immensely while ddoing the actual implementation of the code. It helped us in understanding the data flow and intercations between the different modules of the system. It helped us visualize the data flow.
> The effort we put in making the initial class diagram really played to our benefit in the end while implementation . It made our code highly modular.
> The design pattern we implemented makes our code ready for smooth extension in the future.
> The process of design and analysis makes it easier to inceremently add fatures to the implementation without breaking anything in the existing code.