PROJECT PART 5

Team: Meher Jain, Radhika Paryekar, Subir Kumar Padhee
Title: Hotel Management System
Project Summary: A Hotel Management System in the form of a stand-alone application. It will allow customers, hotel administrative staff and other employees to access the hotel services, administrative activities and employee services respectively.

1. List the features that were implemented (table with ID and title).

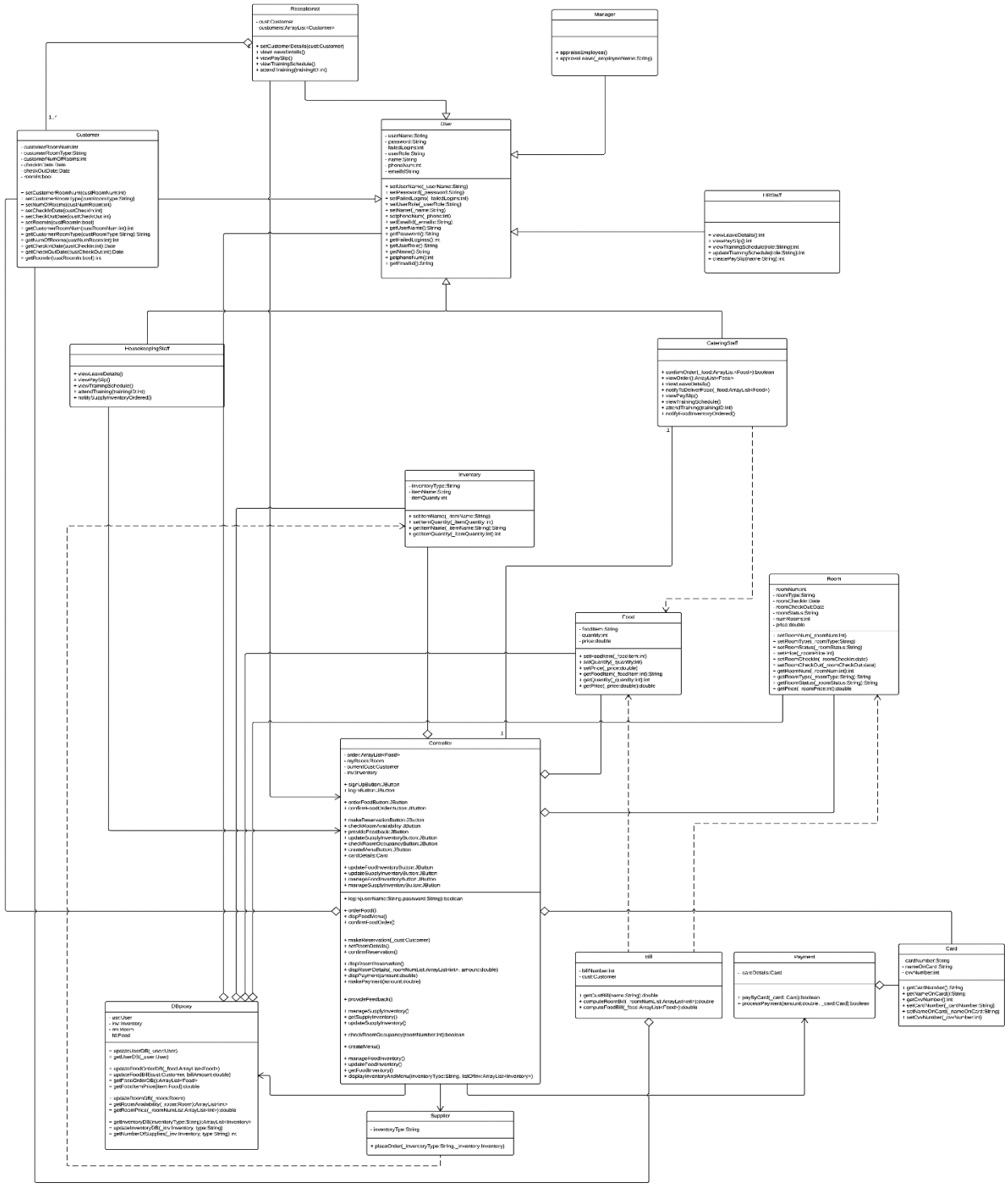
User Requirement		
ID	Description	User
US-01	As a new user, I should be able to sign-up in the system and create a role based profile	All User
US-02	As an existing user, I should be able to login to the system	All User
US-03	As a customer, we want to check room availability so that we can make booking	Customer
US-04	As a customer, I want to view food menu so that I can order food	Customer
US-05	As a customer, I want to view bill so that I can make payment	Customer
US-11	As a housekeeping staff, I want to view the supply inventory and update it	Housekeeping staff
US-12	As a catering staff, I want to view the food inventory and update it	Catering staff

2. List the features were not implemented from Part 2 (table with ID and title).

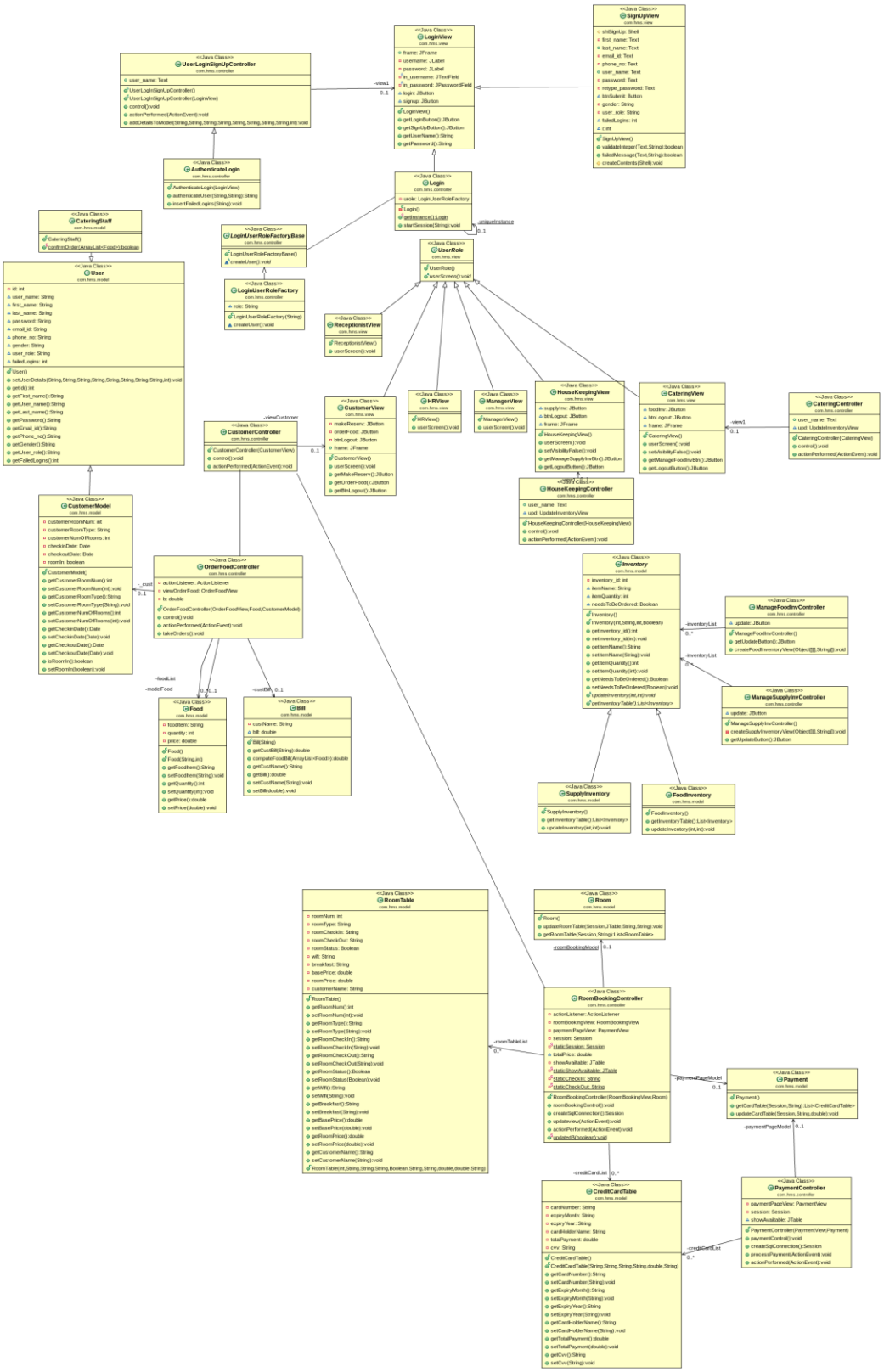
User Requirement		
ID	Description	User
US-06	As a customer, I want to give rating so that I can provide feedback to hotel	Customer
US-07	As a manager, I want to view employees attended training list and customer rating so that I can decide promotion and perks	Manager
US-08	As a manager, I want to access leave availability of all staff so that I can approve leave request	Manager
US-09	As an employee, I want to access my leave details so that I can take note of it.	All Staff
US-10	As a housekeeping staff, I want to check room occupancy so that I can clean the room	Housekeeping
US-13	As a HR, I want to check the attendance report of all staff, so that I can create pay-slip	HR
US-14	As a HR, I want to organize role based training workshop, so that new employees can undergo training	HR
US-15	As an Employee, I want to view my pay-slip so that I can print it.	All Staff
US-16	As an Employee, I want to view training schedule so that I can attend the training	All Staff

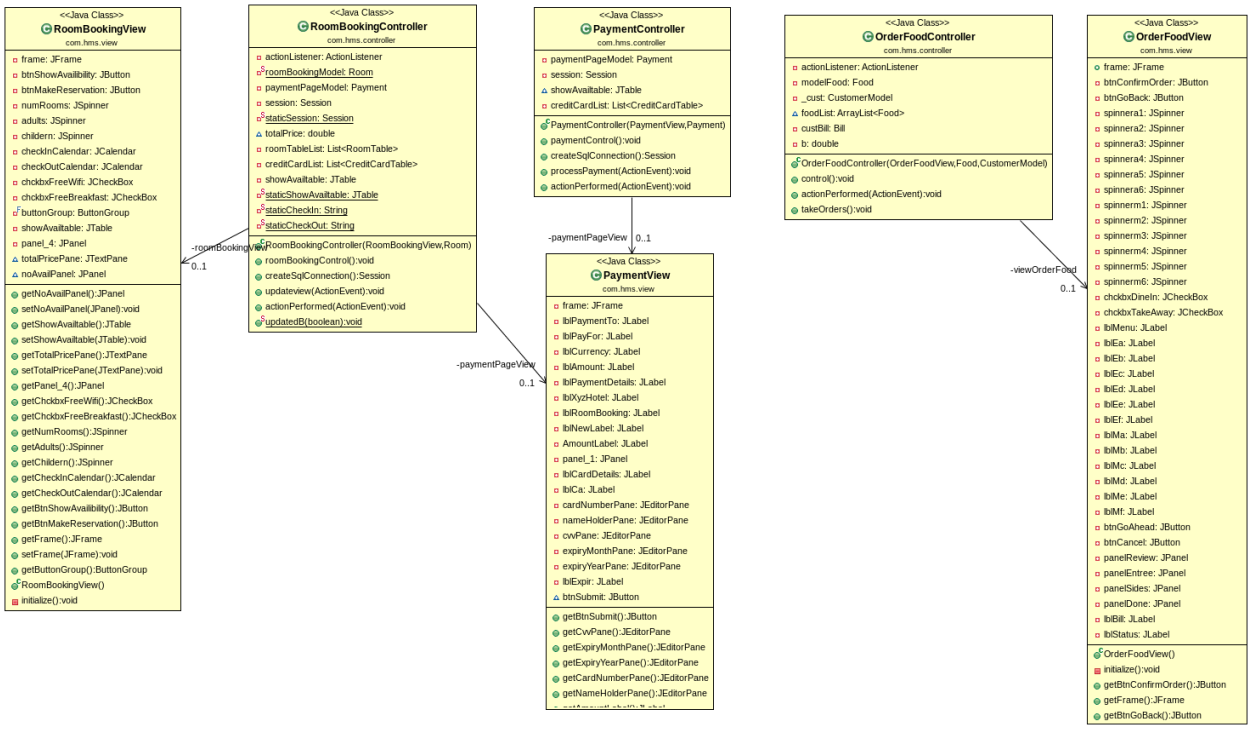
3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Part 2 Class Diagram:



Final Class Diagram:





Our final class diagram changed from the part 2 class diagram. In the part 2 class diagram we encountered the BLOB anti-pattern wherein the Controller class was assigned most of the responsibilities including the User Interface elements. We broke down this Controller class into different Controller classes and UI classes such that each class performed a specific task with high cohesion. We followed the Model View Controller (MVC) Architectural pattern to implement the same.

We introduced the Factory Method Design pattern to instantiate different users based on their roles. The Factory Method Design Pattern allowed us to encapsulate object (user) creation by putting the creation code in a separate Factory Class which has the Factory Method and decouple the object creation from the object behavior. This ensures that if new objects (users) are added to the application in the future only the object creation class has to be changed without changing the class which handles object behavior. All our user views inherit from an Abstract UserRole Class which requires all of them to implement the userScreen method. The LoginUserRoleFactory Class holds the user creation logic based on role which is taken from the user login process. It calls the userScreen method of the object to be created which completes the role based user login process. This ensures that when the application will be extended in the future, it follows the “Open for Extension, but Closed for Modification” Object Oriented Design Principle.

We made use of the Singleton Design Pattern to make sure only one instance of any user is instantiated in a running application.

For the Inventory class, we abstracted the Inventory class such that the FoodInventory class and SupplyInventory class inherit from it. This helped in code reuse and also in instantiating the Inventory object during run time.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

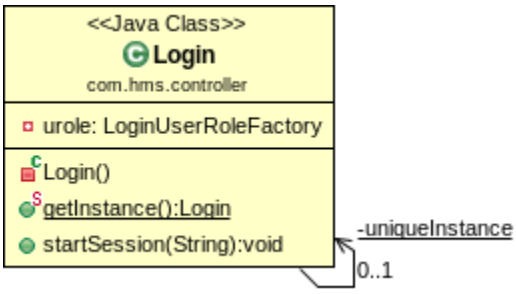
Yes, we made use of design patterns in the implementation of our final prototype. We implemented the following design patterns: Factory Method, Singleton and the Observer Pattern in- MVC.

We implemented the Singleton pattern in the Login class to create a single instance of a user, so that only one user can be logged into the system at a time.

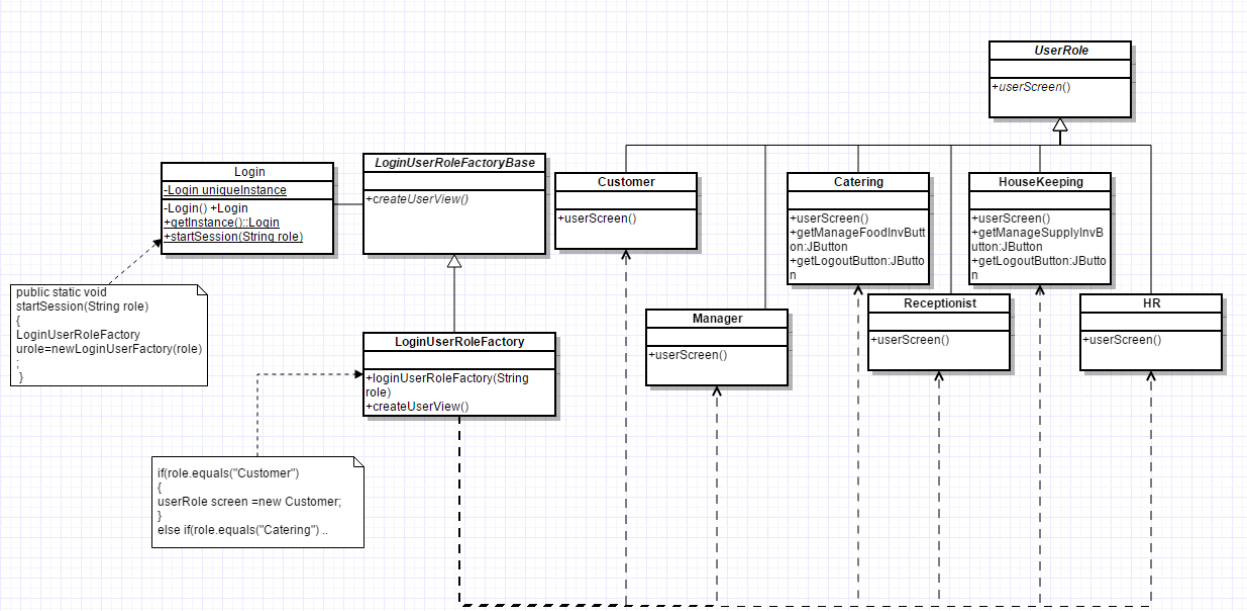
Following is the code snippet and class diagram for the singleton pattern in the Login Class:

```
public class Login extends LoginView
{
     private LoginUserRoleFactory urole;
     private static Login uniqueInstance=null;
     private Login()
     {

     }
     //looks to be incorrect. Should return an instance of the user-customer or whatever
     //What good is an instance of Login()?? -- ensures only one person logs in, fair enough?!
     public static Login getInstance()
     {
         if(uniqueInstance==null)
         {
             uniqueInstance=new Login();
         }
         else{
             JFrame frame = new JFrame();
             JOptionPane.showMessageDialog(frame, "An Instance of this user is already running");
         }
         return uniqueInstance;
     }
```



Next, we used the Factory Method Design pattern to instantiate different users based on their roles. All our user views inherit from an Abstract UserRole Class which requires all of them to implement the userScreen method. Each sub class of UserRole specifies the object it creates. The LoginUserRoleFactory Class holds the user creation logic based on role which is taken from the user login process. It calls the userScreen method of the object to be created which completes the role based user login process.



Lastly, we implemented our application using the Model View Controller architectural pattern. The controller holds all the handlers for the different UI Events, wherein it updates the model based on the code logic. Also, the controller notifies the UI whenever required. (The final class diagram shown in Question 3 showcases the MVC pattern).

5. 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?

The process of analysis and design helped us largely to implement good, maintainable and reusable code. Following the SDLC principle and use of design patterns helped us get a good picture of what we'll be working on and how to go about the implementation of the application. It helped us prevent smelly code by using object oriented principles and design patterns. It also taught us how to allocate responsibilities to classes following good design principles. During the implementation of the application we came across a few anti-patterns. We learned how to rectify these problems and found good design solutions. Further, we also learned that user acceptance testing and continuous refactoring is necessary to ascertain accurate application development and to have maintainable code. The complete process of creating, designing and implementing the system in its entirety has taught us good object oriented programming and designing skills, and enhanced us as programmers.