Hostel Management System

Royal University of Bhutan

Gyalpozhing College of Information Technology

Bachelor of Computer Science - Year III, Semester II

CSF304: Design Patterns

Tutor: Ms Pema Wangmo

Submitted by: Group No - 4

Tapash Rai (12210034)

Rohit Gurung (12210028)

Pema Yangchen (12210025)

Table of Content

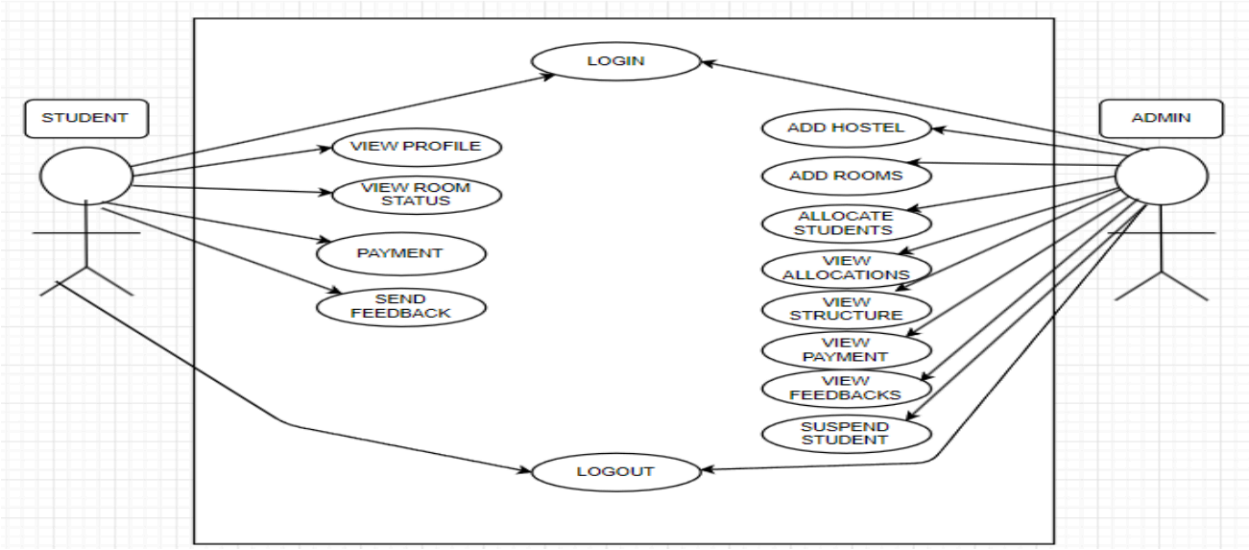
1. Brief Project Description

Aim: Develop a robust and efficient hostel management system that leverages design patterns to ensure scalability, maintainability, and ease of use.

The Hostel Management System is a comprehensive software solution designed to streamline the administration of student accommodations. In this system, the admin can create and manage hostels, allocate rooms to students based on year and gender, and suspend or reassign accommodations. Students can use the system to make payments, submit feedback, and access various services. This system ensures efficient room allocation, maintains clear communication channels between students and administrators, and enhances the overall management of hostel facilities.

2. Use Cases

Use case diagrams to describe what a system does from the standpoint of an external observer. The emphasis of use case diagrams is on what a system does rather than how it is further implemented



Actors

- Students
- Admin
- Hostel Management System.

Use Cases of Students.

- Apply for rooms.
- View their respective profiles.
- Send Feedback to the admin.
- Payment for their rooms,

Use Cases of Admin.

- Set up the hostel blocks.
- Set up the hostel rooms.
- Perform year-based gender room allocations for the students.
- View room allocations.
- View the hierarchical structure of the hostels with their respective rooms.
- View the payment status of the students.
- View feedback sent by the students
- Suspend a student.

3. Source Code

The GitLab link for our project Hostel Management System is :

https://github.com/tapashhrai/Hostel-Management

4. **Framework:** The framework is fixed and unalterable (non-changeable), while the implementation is adaptable and can be modified(changeable) according to the needs of the client or the preferences of the application developer. The framework provides the foundational structure and guidelines, ensuring consistency and reliability across different applications

Signup for students and admin implemented using abstract factory design-pattern.

Framework(Non-changeable functionalities) > UserFactory.java > ...

```java
// Abstract factory class for creating User objects
public abstract class UserFactory {
    /**
     * Abstract method to create a User object.
     * Subclasses of UserFactory must provide an implementation for this method.
     *
     * @param username The username of the user.
     * @param enrollmentNumber The enrollment number of the user.
     * @param email The email address of the user.
     * @param hashedPassword The hashed password of the user.
     * @param gender The gender of the user.
     * @param year The year associated with the user (e.g., enrollment year).
     * @return A User object.
     */
    public abstract User createUser(String username, String enrollmentNumber, String email, String hashedPassword, String gender, int year);
}
```

UserFactory abstract class:

- This is an abstract class meant to serve as a blueprint for creating User objects.
- It declares an abstract method createUser, which must be implemented by any concrete factories.
- The createUser method is intended to encapsulate the logic for creating different types of User objects.

```
Framework(Non-changeable functionalities) > User.java > ...
// Abstract class representing a generic User
public abstract class User {
    // Protected attributes to store user details
    protected String username;
    protected String enrollmentNumber;
    protected String email;
    protected String hashedPassword;

    //Constructor to initialize a User object.
    public User(String username, String enrollmentNumber, String email, String hashedPassword)
        this.username = username;
        this.enrollmentNumber = enrollmentNumber;
        this.email = email;
        this.hashedPassword = hashedPassword;
    }
}
```

User abstract class:

- An abstract class representing a generic user or an abstract product.
- It contains protected attributes for storing user details such as username, enrollment number, email, and hashedPassword.
- It cannot be instantiated directly but can be implemented by concrete products.

Creation of various objects of different types of hostel and rooms by the admin

```
Framework(Non-changeable functionalities) > FormFactory.java > ...
import javax.swing.JFrame;
// Interface for a factory that creates JFrame objects
public interface FormFactory {
    /**
     * Method to create a JFrame form.
     * Classes implementing this interface must provide an implementation for this method.
     *
     * @return A new instance of a JFrame.
     */
    JFrame createForm();
}
```

FormFactory interface:

- This is an interface meant to define an abstract factory for creating JFrame objects.
- It declares a single method, createForm, which is implemented by concrete factories (HostelFormFactory & RoomFormFactory) that implements the concrete product(AddHostelForm & AddRoomForm) to add the respective hostel with their rooms.

Payment of room fees by the students using proxy design-pattern

```
Framework(Non-changeable functionalities) > Payment.java > ...
// Interface for processing payments
public interface Payment {

    /**
     * Method to make a payment for a student.
     * Classes implementing this interface must provide an implementation for this method.
     *
     * @param studentEmail The email address of the student for whom the payment is being made.
     */
    void makePayment(String studentEmail);
}
```

Payment interface:

- An interface meant to define a contract for making payments.
- It declares a single method, makePayment, which is implemented by a proxy class to provide payment functionality.
- The makePayment method takes a studentEmail as a parameter, indicating the email of the student for whom the payment is being made.

Observer pattern to notify students and admins about successful or failed login attempts

```
Framework(Non-changeable functionalities) > LoginObserver.java > LoginObserver
// Interface for observing login events
public interface LoginObserver {
    void onLoginSuccess(String userType);
    /**
     * Method to handle successful login events.
     * Classes implementing this interface must provide an implementation for this method.
     *
     * @param userType The type of user that has successfully logged in (e.g., admin, student).
     */

    void onLoginFailure();
    /**
     * Method to handle login failure events.
     * Classes implementing this interface must provide an implementation for this method.
     */
}
```

LoginObserver interface:

- This is an interface meant to define a contract for observing login events.
- It declares two methods: onLoginSuccess and onLoginFailure.
- Concrete observers provide logic for what should happen when a login is successful (onLoginSuccess) or when it fails (onLoginFailure).

Mediator pattern to send feedbacks by the students to the administrator

Framework(Non-changeable functionalities) > Mediator.java > ...

```java
// Interface for mediating feedback communication
public interface Mediator {
    /**
     * Method to send feedback.
     * @param email The email address to which the feedback will be sent.
     * @param feedback The feedback message to be sent.
     */
    void sendFeedback(String email, String feedback);
}
```

Mediator interface:

- This interface is meant to define a contract for sending feedback.
- It declares a single method, sendFeedback, which is implemented by a concrete FeedbackMediator to mediate feedback communication.
- The sendFeedback method takes two parameters: email (the recipient's email address) and feedback (the feedback message to be sent).

Composite pattern to represent hostel's hierarchical structure of rooms

Framework(Non-changeable functionalities) > HostelComponent.java > ...

```java
// Interface for hostel components that can be displayed
public interface HostelComponent {
    void display(StringBuilder builder, String indent);
    /**
     * Method to display the hostel component information.
     * @param builder The StringBuilder object to append the display information to.
     * @param indent The indentation string to format the display output.
     */
}
```

HostelComponent interface:

- An interface that defines a contract for displaying hostel-related information.
- It declares a single method, display, which is implemented by composite class(hostel) and leaf class (room) to represent a component of a hostel hierarchical structure.
- The display method takes two parameters: a StringBuilder object to append the display information to, and a String indent to format the display output properly

Strategy pattern for year wise gender based room allocation of the students.

Framework(Non-changeable functionalities) > RoomAllocationStrategy.java > ...

```java
// Interface for defining room allocation strategies
public interface RoomAllocationStrategy {
    void allocateRooms();
    // Method to allocate rooms according to a specific strategy.

}
```

RoomAllocationStrategy interface:

- An interface that defines a contract for room allocation strategies.
- It declares a single method, allocateRooms, which is implemented by year based gender allocation strategy class for allocating rooms.

State pattern to represent various states of the room (occupied or vacant)

Framework(Non-changeable functionalities) > RoomState.java > ...

```java
// Interface for defining the state of a room and handling state transitions
public interface RoomState {

    void handle(Room room);
    // Method to handle the current state of the room.\
    //@param room The Room object whose state is being handled.

    String getState();
    //Method to get the current state of the room as a string.
    //@return The current state of the room as a string.
}
```

RoomState interface:

- An interface that defines a contract for handling different states of a room.
- It declares two methods: handle and getState.
- The handle method is intended to contain the logic for transitioning the room to the next state or handling its current state.
- The getState method is intended to return the current state of the room as a string.

State pattern to represent various states of the students (undergraduate or suspended)

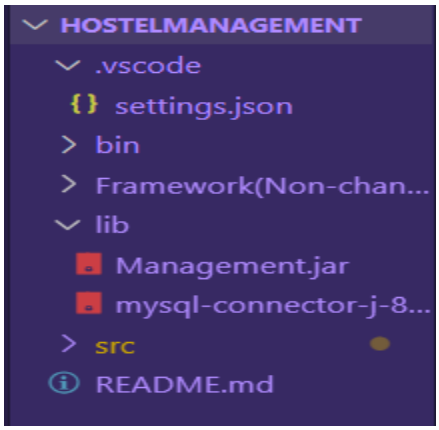
Framework(Non-changeable functionalities) > StudentState.java > ...

```java
// Interface for defining the state of a student and handling state transitions
public interface StudentState {
    void handleState(StudentContext context);
    //Method to handle the current state of the student.
    // @param context The StudentContext object that contains information about the student's environment and allows for state transitions.
}
```

- An interface that defines a contract for handling different states of a student.
- It declares a single method, handleState, which is implemented by active-state class and suspended-state class to represent a specific state of a student
- The handleState method takes a StudentContext object as a parameter, which represents the context or environment in which the student exists and is managed.

Conclusion

- All of the above-mentioned framework java files have been converted into a jar file using Eclipse IDE, a popular, open-source integrated development environment (IDE) used primarily for Java development.
- A JAR (Java ARchive) file is a platform-independent file format used for aggregating many Java class files or Java classes into a single compressed file.



- Saved all the framework files into a Management.jar file.
- Saved the Management.jar file into our hostel management project directory within the lib directory.
- All the subclasses imports, implements or extends this compressed framework jar file using the following command.

```java
import HostelManagement.*;
```

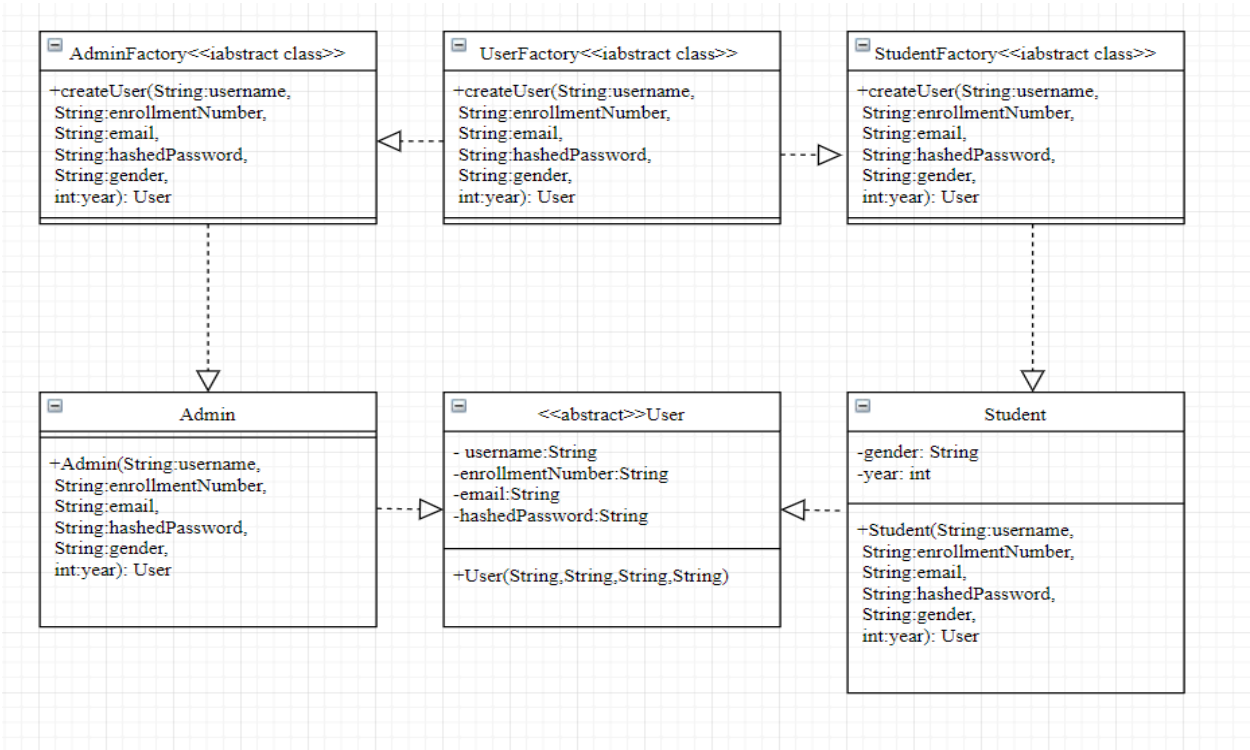
5. Class Diagrams.

The link for the class diagram is given below:

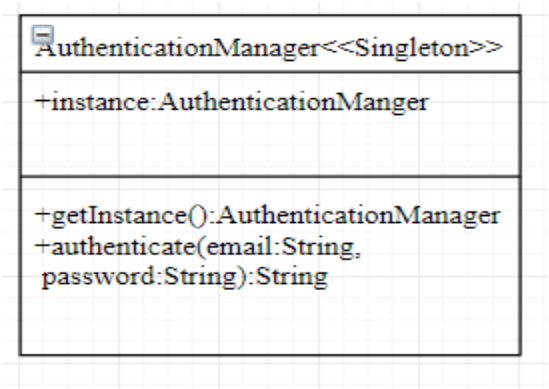
https://github.com/tapashhrai/Hostel-Management/blob/main/src/classdiagram.drawio

All of the following are the class diagrams for the respective design pattern as the joined class diagram has large width and height and cannot be inserted in the document. Please refer to the above link for the clubbed class diagrams.

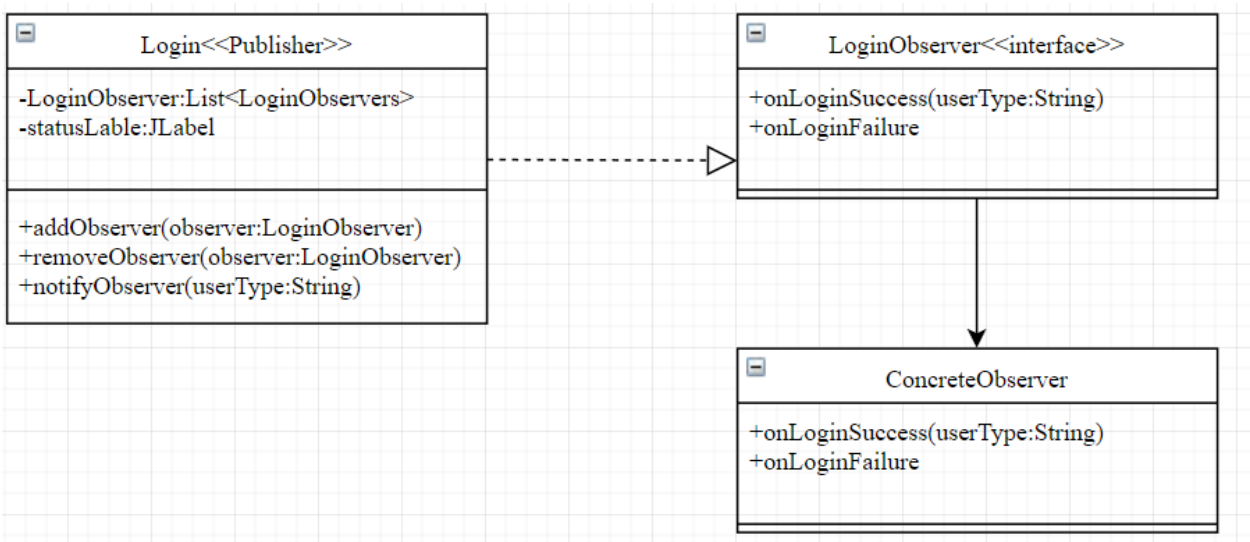
Sign up using an abstract factory for students and admin.



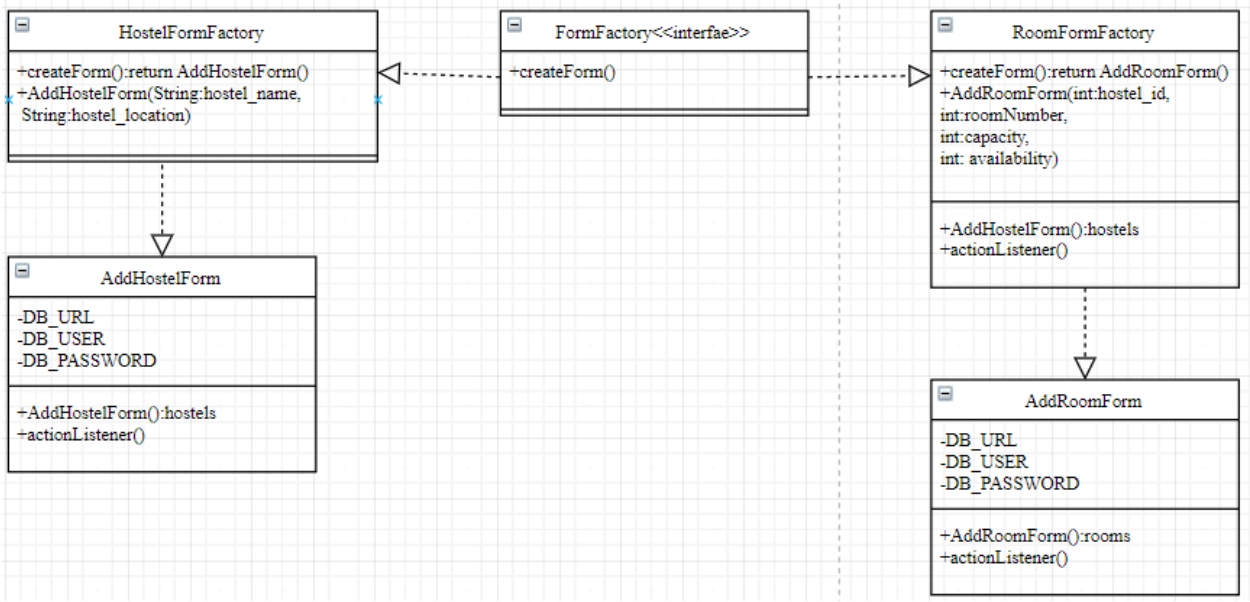
Login using Singleton for authentication



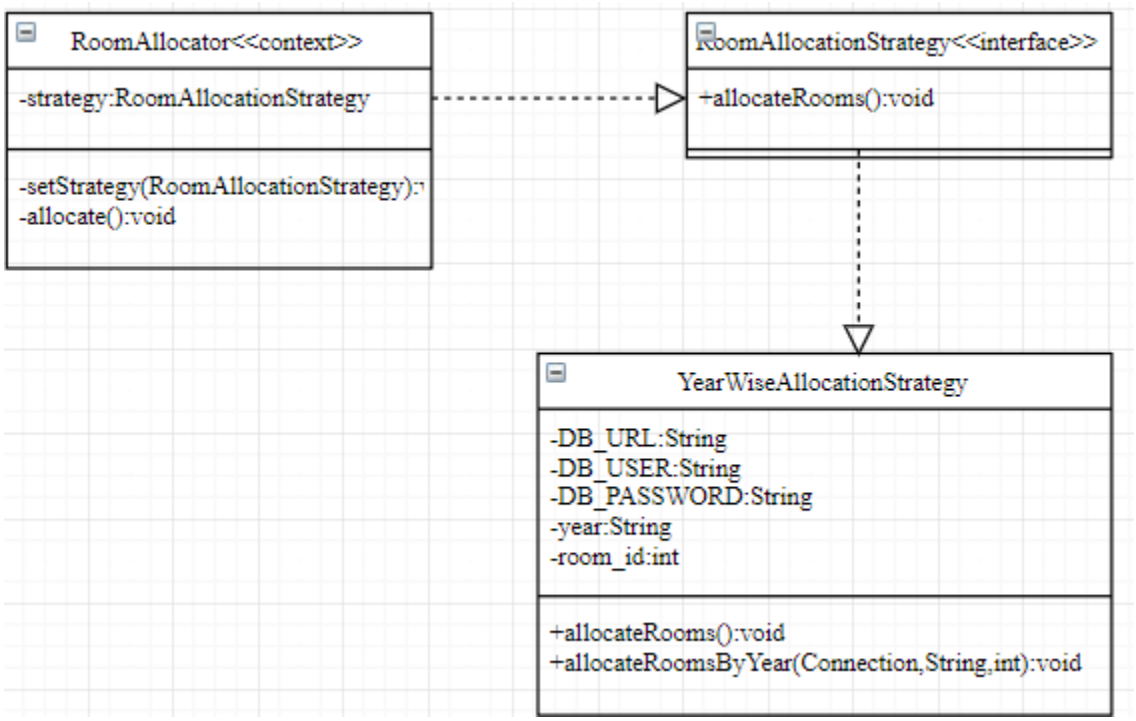
Observer pattern to notify students and admins about successful or failed login attempts.



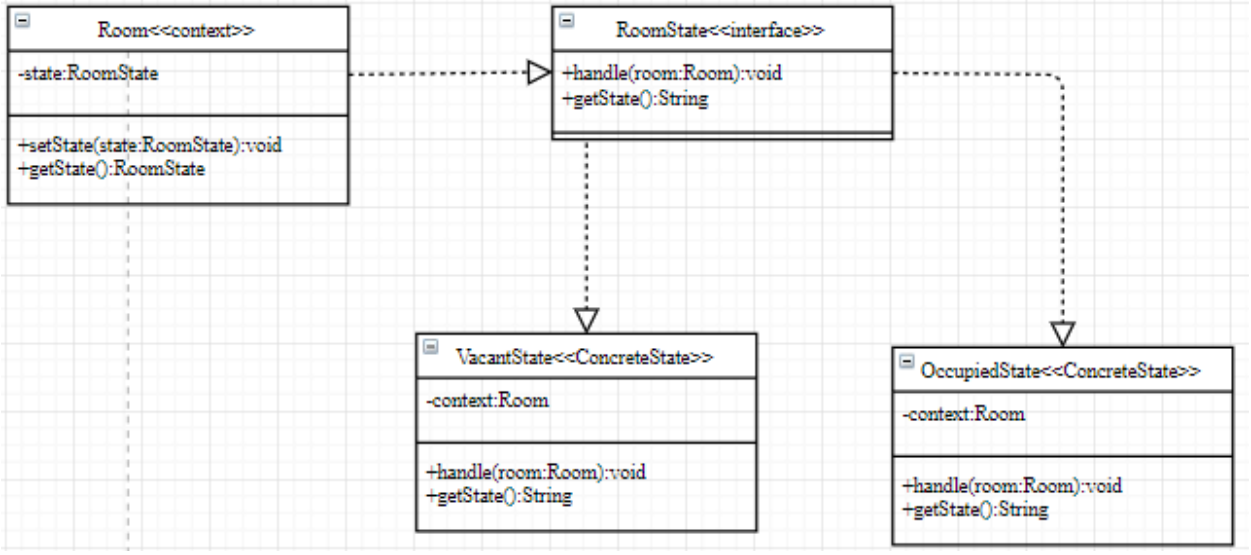
Abstract factory to create various objects of different types of hostel and rooms



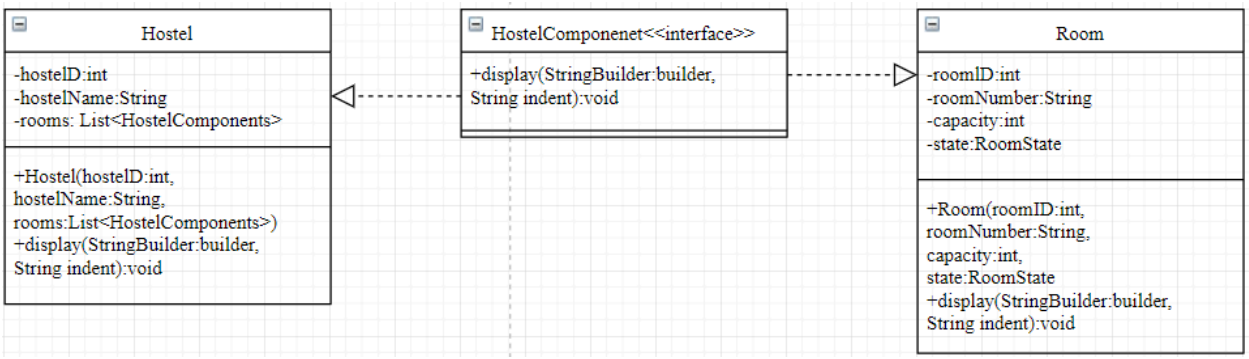
Strategy pattern for year wise gender based room allocation



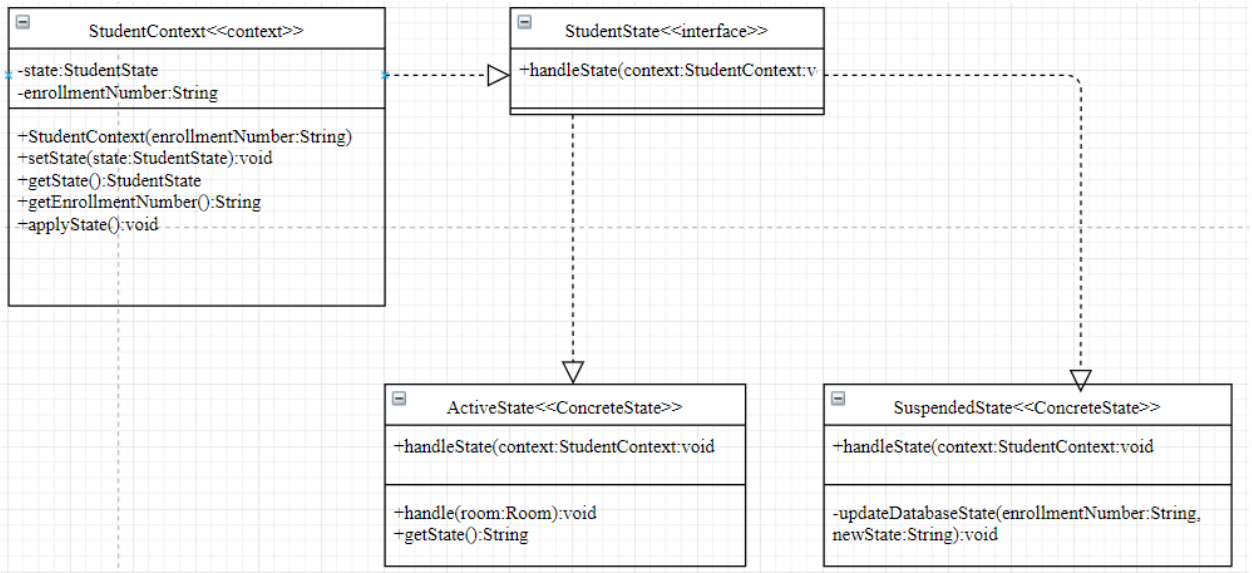
State pattern to view the state of the rooms.



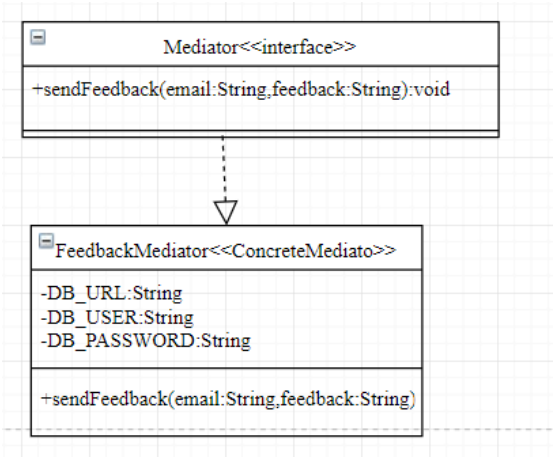
Composite pattern to view the hierarchical structure of the hostels.



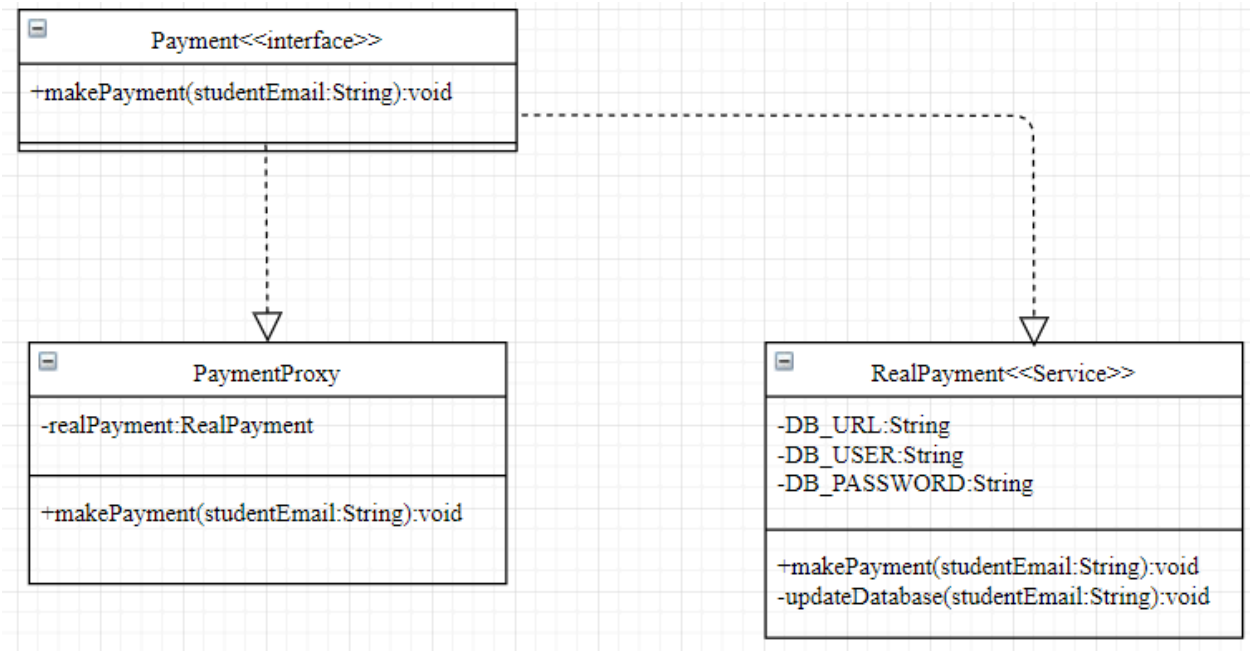
State pattern to view the state of the students.



Mediator pattern to send feedback to the admin by the student.

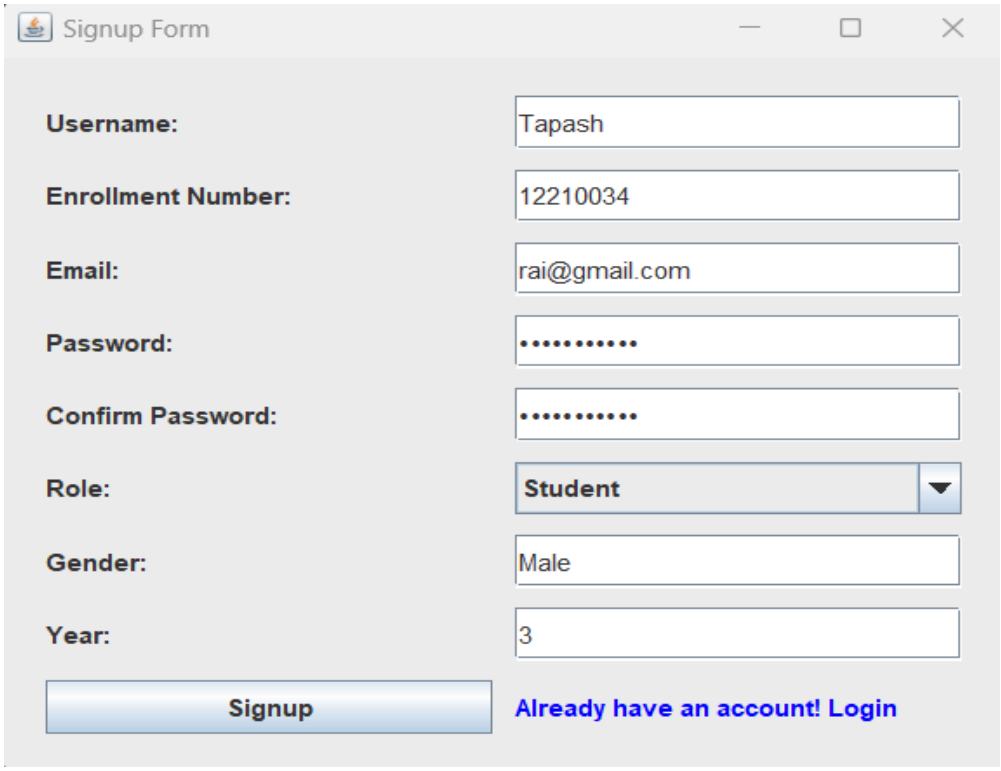


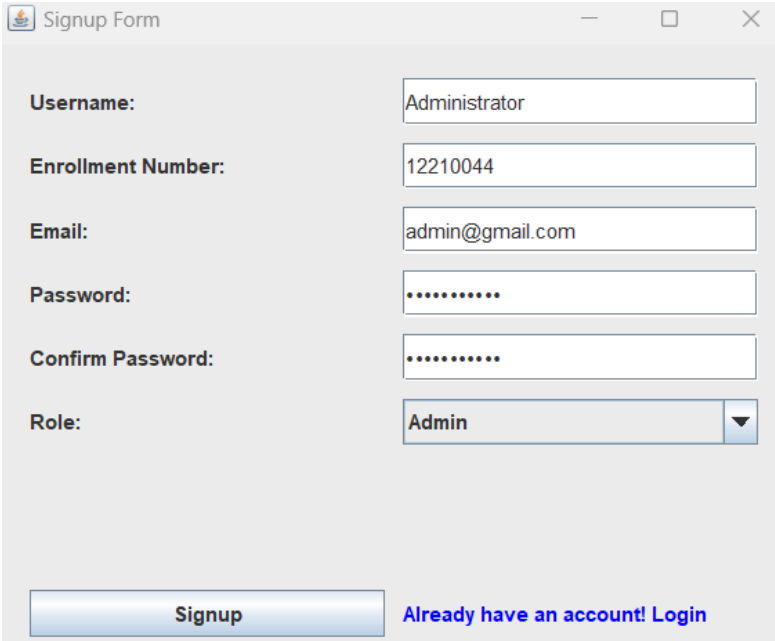
Proxy pattern for room payment by the students.



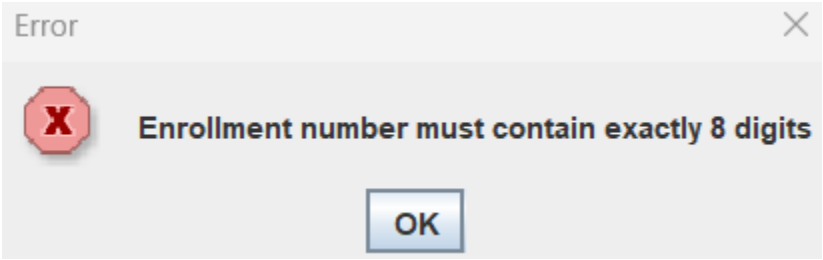
6. User Interface.

Our user interface starts with a signup page to sign up based on students or admin





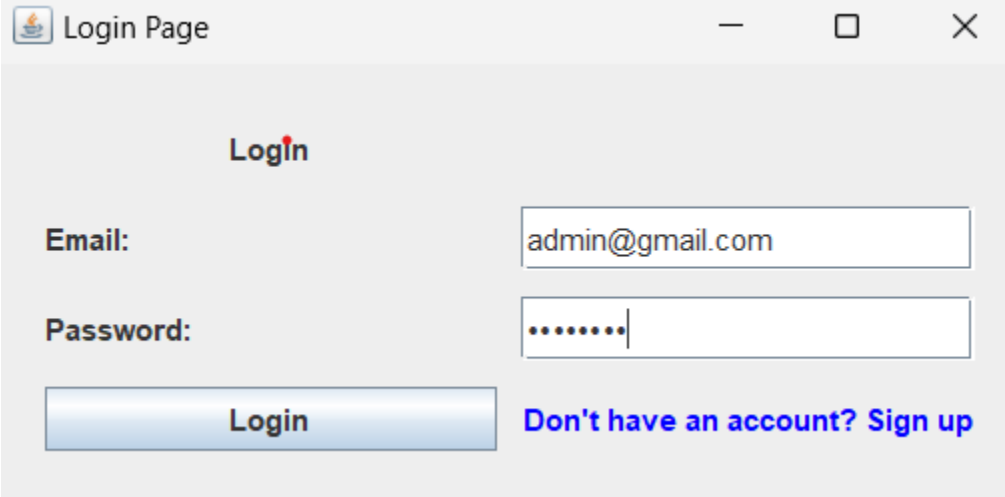
It has all the required form validations and it pops up to notify the user while signing up. The following snippet is an example of one of the form validations



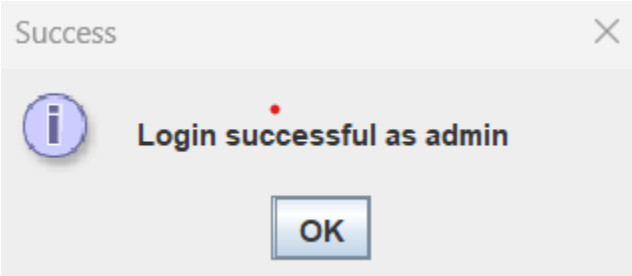
The form validation are as follows:

```
// Validate input
if (username.isEmpty() || enrollmentNumber.isEmpty() || email.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {
    JOptionPane.showMessageDialog(Signup.this, message:"Please fill in all fields", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}

if (!username.matches(regex:"[a-zA-Z]+")) {
    JOptionPane.showMessageDialog(Signup.this, message:"Username must contain only letters", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}

if (!enrollmentNumber.matches(regex:"\\d{8}")) {
    JOptionPane.showMessageDialog(Signup.this, message:"Enrollment number must contain exactly 8 digits", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}

if (!email.matches(regex:"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}")) {
    JOptionPane.showMessageDialog(Signup.this, message:"Invalid email format", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}

if (password.length() < 8) {
    JOptionPane.showMessageDialog(Signup.this, message:"Password must be at least 8 characters long", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}

if (!password.equals(confirmPassword)) {
    JOptionPane.showMessageDialog(Signup.this, message:"Password and confirm password do not match", title:"Error", JOptionPane.ERROR_MESSAGE);
    return;
}
```

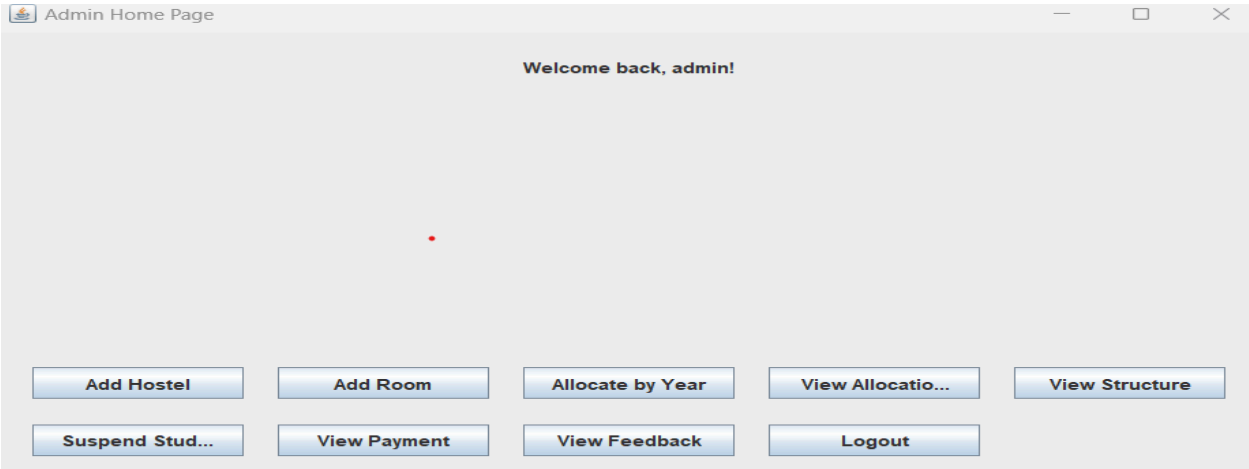
After signing up it redirects the user to the login page.



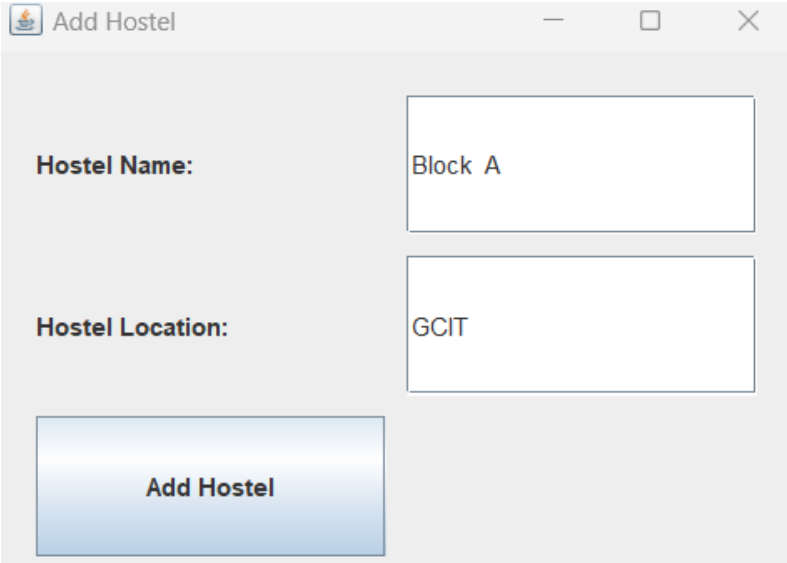
Upon successful or failed login credentials it screens the following pop up and redirects the user to their respective admin or student home page.

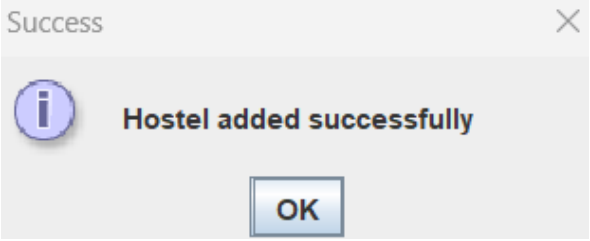


Admin Home page.

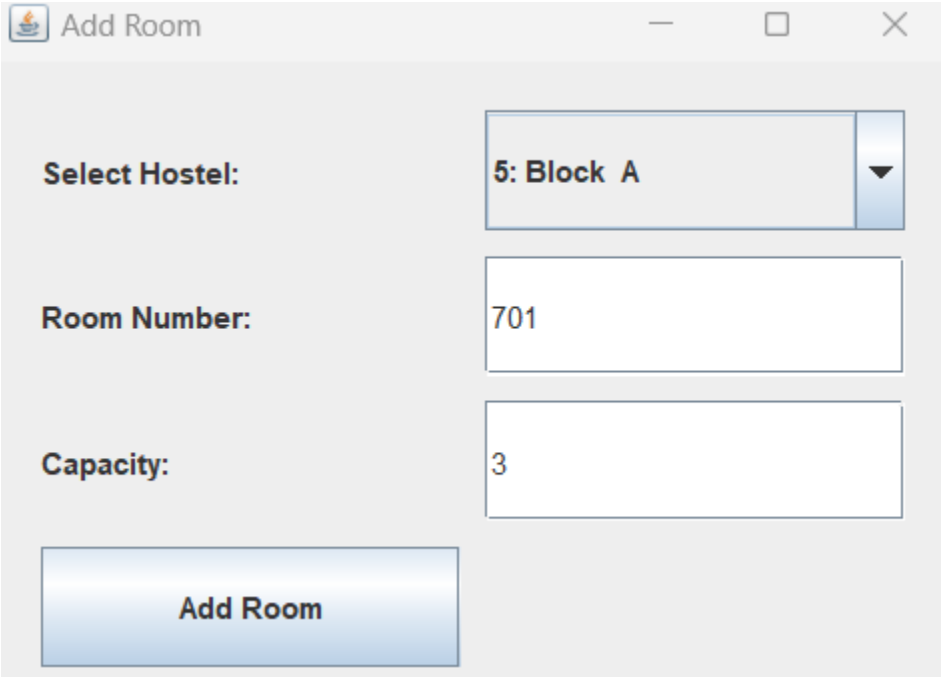


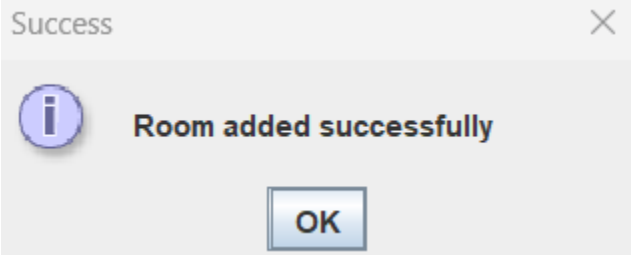
Upon clicking on Add Hostel it will add a hostel.



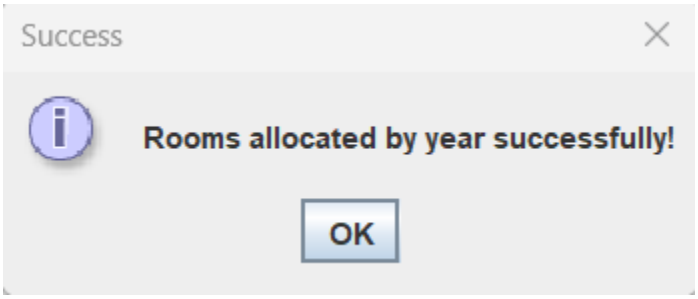


Clicking on Add Room lets us add rooms based on the hostel created.





Clicking on Allocate by Year will allocate year based gender room allocation for the students.



Clicking on View Allocation shows a table to view the student details, student room allocation and the state of the student.

Room Allocations

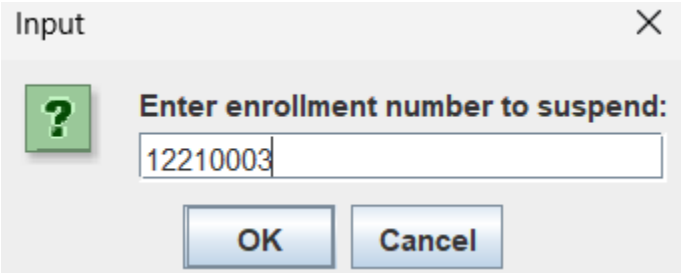
Student ID	Username	Gender	Hostel Name	Year	Room ID	Room Number	State
12210002	Dorji	Male	Block N	1st Year	1	101	suspended
12210003	Jigme	Male	Block N	1st Year	1	101	undergraduate
12210005	Tapash	Male	Block N	3rd Year	3	301	suspended
12210006	Rohit	Male	Block N	3rd Year	3	301	undergraduate
12210007	Tenzin	Male	Block N	4th Year	4	401	undergraduate
12210008	Dawa	Male	Block N	4th Year	4	401	undergraduate
12210003	PemaYangchen	Female	Block K	2nd Year	2	201	undergraduate
12210004	PemaWangmo	Female	Block K	2nd Year	2	201	undergraduate

All the Male students will be allocated to Block N and female students to Block K. All the 1st year students will be allocated to RoomID:1 and Room Number:101.All the 2nd year students will be allocated to RoomID:2 and Room Number:201. All the 3rd year students will be allocated to RoomID:3 and Room Number:301. All the 4tht year students will be allocated to RoomID:4 and Room Number:401.

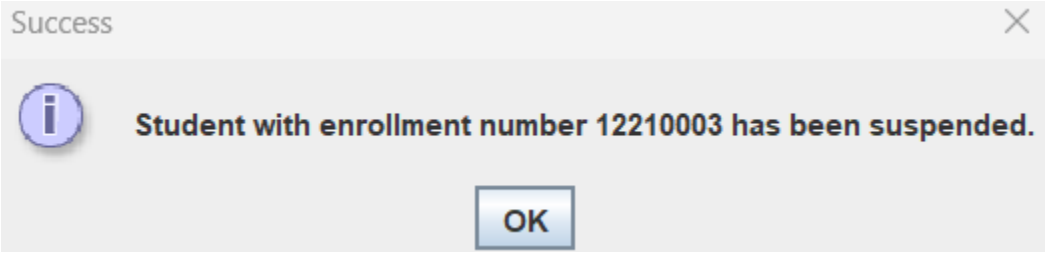
Clicking on View Structure will show a hierarchical structure of hostels with their respective rooms and the state of the rooms.

```
Hostel ID: 1, Hostel Name: Block N
  Room ID: 1, Room Number: 101, Capacity: 1, State: vacant
  Room ID: 3, Room Number: 301, Capacity: 0, State: occupied
  Room ID: 4, Room Number: 401, Capacity: 1, State: vacant
Hostel ID: 2, Hostel Name: Block K
  Room ID: 2, Room Number: 201, Capacity: 1, State: vacant
Hostel ID: 4, Hostel Name: Block O
  Room ID: 6, Room Number: 501, Capacity: 5, State: vacant
Hostel ID: 5, Hostel Name: Block  A
  Room ID: 7, Room Number: 701, Capacity: 3, State: vacant
```

Clicking on Suspend student will suspend a student by the admin based on their enrollment number. To view the state of the student you can click View Allocations.



\



Clicking on View Payment will show a table to view the status of the students who have paid their room fees.

Student Payments

Username	Enrollment Number	Email	Year	Payment
Dorji	12210002	dorji@gmail.com	1st Year	pending
Jigme	12210003	jigme@gmail.com	1st Year	pending
PemaYangchen	12210003	yangchen@gmail.com	2nd Year	pending
PemaWangmo	12210004	wangmo@gmail.com	2nd Year	pending
Tapash	12210005	trai@gmail.com	3rd Year	paid
Rohit	12210006	rohit@gmail.com	3rd Year	pending
Tenzin	12210007	tenzin@gmail.com	4th Year	pending
Dawa	12210008	dawa@gmail.com	4th Year	pending
sdsd	11111111	a@gmail.com	3rd Year	pending

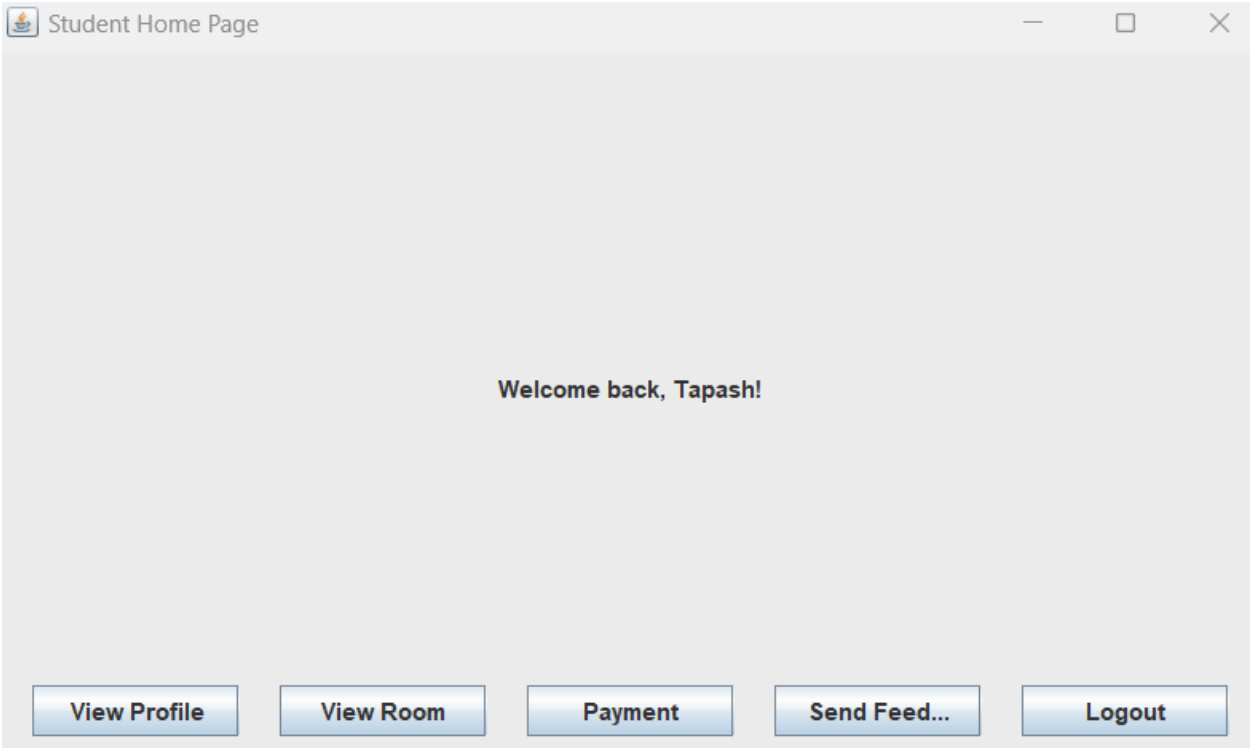
Clicking on View Feedback will show a table to view the feedback of the rooms sent by the students.

Student Feedback

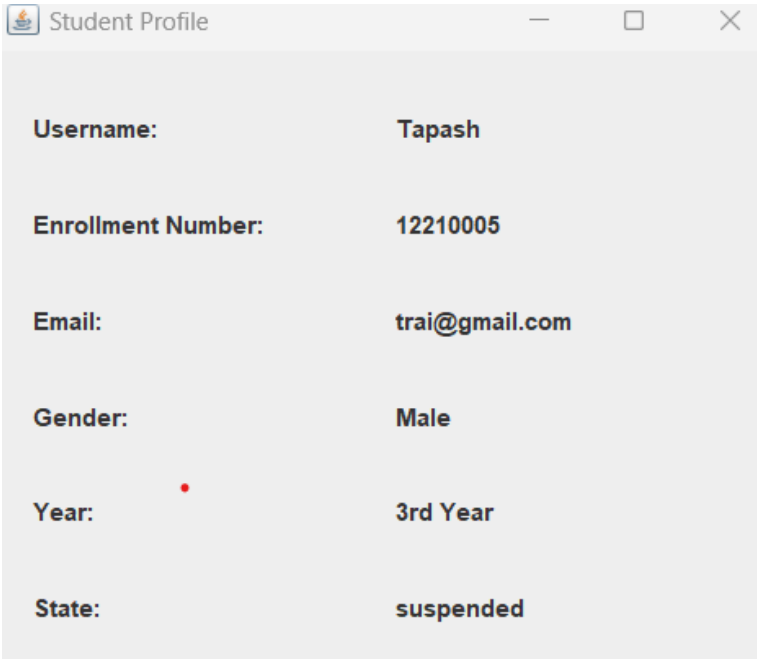
Enrollment Number	Username	Year	Hostel Name	Room Number	Feedback
12210002	Dorji	1st Year	Block N	101	no feedback
12210003	Jigme	1st Year	Block N	101	no feedback
12210005	Tapash	3rd Year	Block N	301	windows are broken
12210006	Rohit	3rd Year	Block N	301	no feedback
12210007	Tenzin	4th Year	Block N	401	no feedback
12210008	Dawa	4th Year	Block N	401	no feedback
11111111	sdsd	3rd Year	Block N	301	no feedback
12210003	PemaYangchen	2nd Year	Block K	201	no feedback
12210004	PemaWangmo	2nd Year	Block K	201	no feedback

Clicking on Logout will end the user session of the admin and redirect back to login page

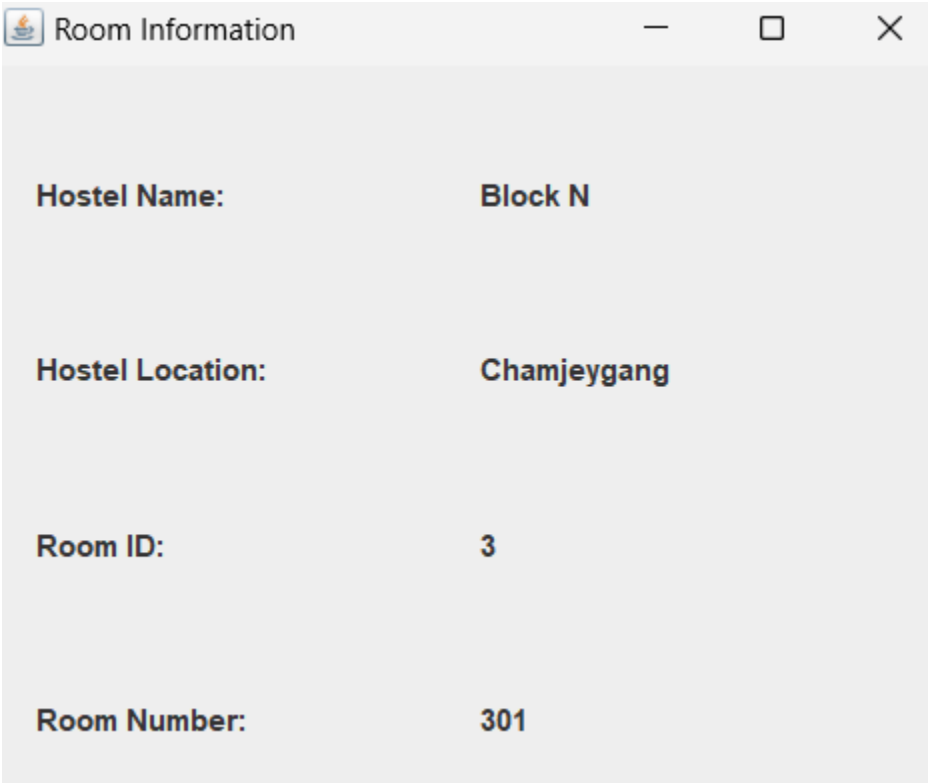
Student Home Page.



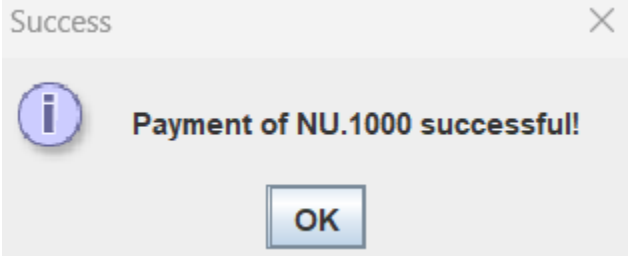
Clicking on the view profile will display the personal information of the students.



Clicking on the view room will display allocated room information of the students

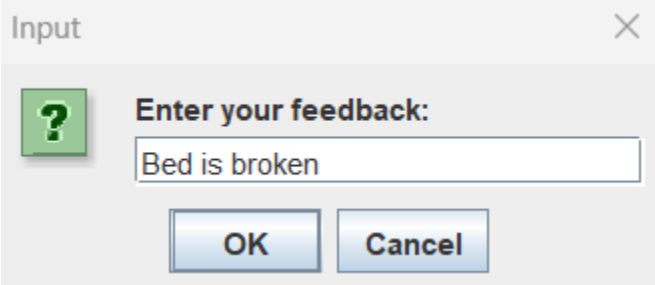


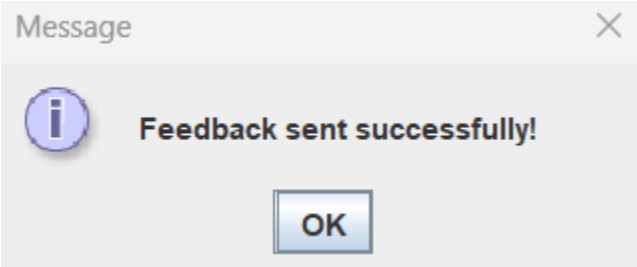
Clicking on payment will allow the student to pay the room fees.



The amount displayed is Nu.1000 because each semester an amount of Nu.250 is taken off from our monthly stipend so each student has to stay for a total of 4 months per semester (250x4=Nu.100)

Clicking on Send Feedback will send the feedback of the room to the admin by the student.





Clicking on Logout will end the user session of the student and redirect back to login page

7. Justification for all the design patterns used

1. **Singleton pattern** - Implement this pattern to ensure that only one instance of critical classes, such as the Authentication Manager exists to ensure only instance of the authentication exist. Authentication of only a single student or an administrator can be run. Multiple authentications cannot be possible.

2. **Observer pattern** - Observer pattern is implemented to notify students and admins about successful login attempts

3. **Proxy pattern** - Implement this pattern for the online payment method of room fees by the students

4. **Mediator pattern** - Implement this pattern to provide feedback and complaints about the hostel management by the students to the admin.

5. **Strategy pattern** - Implement this pattern for year wise gender based room allocation strategies to the students.

6. **Abstract Factory Method** - Implement factories to create various objects of different types of hostels and rooms. Create various objects of different types of users(students and admin).

7. **State Pattern** - Implement this pattern to represent various states of the rooms such as occupied, and vacant, and multiple states of the student such as under-graduated or suspended.

8. **Composite Pattern** - implement this tree-like hierarchy pattern to represent a hostel's hierarchical structure of rooms.

8. Challenges

Developing a hostel management system using design patterns involves several challenges:

1. **User Authentication and Authorization:** Securely managing different user roles and ensuring proper access controls to sensitive data and functionalities.

2. **Scalability:** Designing the system to handle increasing numbers of users.

3. **Maintainability and Extensibility:** Ensuring the system is easy to maintain and extend with new features as requirements evolve over time.

4. **User Interface Design:** Creating an intuitive and user-friendly interface that accommodates the needs of both students and administrators.

5. **Security:** Protecting sensitive user data to prevent unauthorized access and breaches.

9. Conclusion

The Hostel Management System is designed to streamline student accommodation administration. By leveraging design patterns like Singleton, Observer, Proxy, Mediator, Strategy, Factory Method, State, and Composite, the system achieves robustness, scalability, and flexibility. Despite development challenges, these patterns ensure maintainability and adaptability, resulting in a reliable, user-friendly platform that enhances hostel management for administrators and students.