Design Pattern : Façade

Justification

Simplified interface: The Facade pattern covers for a complex system by presenting a user friendly interface. In our application, there will be different modules for registration of the course, calculation of fee, discount if any, and generation of bill. By applying the Facade pattern, it is possible to develop a unified interface for these operations, and as a result of that, they will be more convenient for users.

Higher maintainability: The complexity of the system is hidden behind the Facade, making it easier to maintain and modify.

Decoupling of Subsystems: The Facade pattern decouples subsystems, so that changes in one of them does not necessarily affect the others. This is relevant because it may require changes in the future due to such aspects as new financing packages, discounting techniques, and extra charges.

Modularity: Each of the subsystems like enrollment, billing, discount calculation and so on, can be implemented and tested separately which enhances the modularity and reduces the probability of occurrence of bugs.

Flexibility: With a Façade, it is also possible to extend this system with additional features in which these extensions do not affect severely the interface. For example, introducing new types of courses or financing packages can be managed within the existing Facade structure.

Extensibility: The pattern allows for future expansions such as changing discount rate without requiring changes to existing structures.

Design Pattern Usage

The *SystemFacade* is the administrative layer of the application, which provide an access to the diverse administrative functionalities in an easier manner. It acts as a facade by encapsulating complex interactions with subsystems, offering streamlined access to key tasks such as:

Login: Authenticating users' login details

Switching Content Panes: Switching between the different views in the application while keeping the GUI.

Enrollment: Handling new student enrollments into courses and subjects.

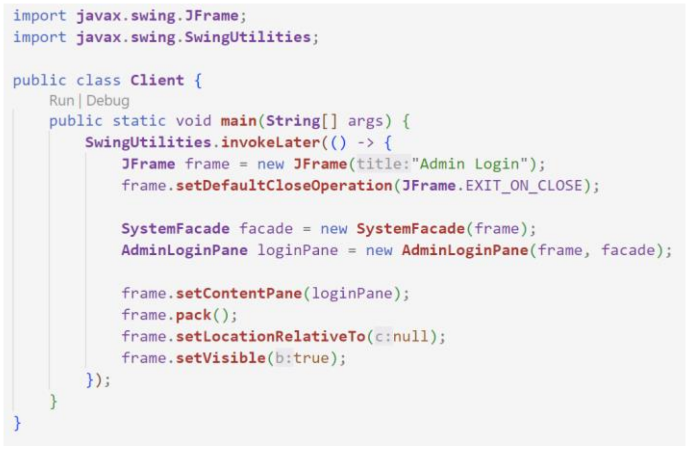
Student Enrollment Tracking: Providing mechanisms that enable supervision on the students enrollment status.

Accommodation Registration: Facilitating the registration process for student accommodation options.

Viewing Bills: Displaying detailed billing information of students

Modifying Discount Strategies: Allowing adjustments to fee structures and discount policies

These operations are hidden by the *SystemFacade*, providing a unified and coherent view of how the logical work associated with all of them can be easily maintained by administrators without delving into the specifics of each subsystem's implementation. On the client interface, it only interact with the façade.



```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class Client {
    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame(title:"Admin Login");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            SystemFacade facade = new SystemFacade(frame);
            AdminLoginPane loginPane = new AdminLoginPane(frame, facade);

            frame.setContentPane(loginPane);
            frame.pack();
            frame.setLocationRelativeTo(c:null);
            frame.setVisible(b:true);
        });
    }
}
```

Assumption made:

1. There is a database for storing all the user ID and password, and the program is only for admin use.

2. Students can register for up to three courses under the stack up program.

3. User run the program on *Client* class and the user will be able to access the whole GUI program.