**Final Project**

**Object Oriented Programming**

A picture containing logo

Description automatically generated

Project Name: Hostel Management System

Student Name: Wallace Louis Tjang

Student ID:2602169705

Table of Contents

[Project Specification 3](#_Toc137840484)

[**Purpose of project and problem solved** 3](#_Toc137840485)

[**How to run the code** 3](#_Toc137840486)

[**How the app works** 4](#_Toc137840487)

[Solution Design 5](#_Toc137840488)

[**Each class and its workflow:** 5](#_Toc137840489)

[What was implemented and how it works 6](#_Toc137840490)

[**Main class** 6](#_Toc137840491)

[**Room class** 7](#_Toc137840492)

[**Student class** 7](#_Toc137840493)

[**AdminGUI class** 8](#_Toc137840494)

[**CreateAccountGUI class** 9](#_Toc137840495)

[**FilesHandler class** 10](#_Toc137840496)

[**StudentsGUI class** 10](#_Toc137840497)

[**StudentsLoginGUI class** 11](#_Toc137840498)

[**StudentsMessageGUI class** 12](#_Toc137840499)

[Evidence of Working Program 14](#_Toc137840500)

[**Lesson Learned** 19](#_Toc137840501)

# Project Specification

## **Purpose of project and problem solved**

The idea behind my project is related on how my friends who are students from out the city are living in Jakarta inside hostels. I was curious as to how the system would be in a Hostel. Hence, I created the Hostel Management which includes stuffs that are relatable.

The goal of my Java Hostel Management System written in Object-Oriented Programming (OOP) is to provide a complete and efficient solution for managing hostel-related operations and processes. It is intended to increase overall efficiency and organization inside a hostel facility by streamlining management chores, automating manual processes, and streamlining management tasks.

The system aims to solve several problems such as:

1. Reservation and Room Allocation

The system allows the hotel administrators to manage reservations and allocate rooms to students. It automates the process of checking room availability, assigning appropriate rooms based on preferences or criteria, and maintaining an organized record of bookings.

1. Students Management:

It provides a centralized database for storing guest information, including personal details, room details and any special requirements. This facilitates easy retrieval of guest data and enables efficient management of student-related activities throughout their stay.

1. Billing and Payment Processing

The system automates billing processes by allowing students to pay their fees with just a single click. It also can track payments by generating monthly reports for financial analysis and record-keeping.

1. Reporting and Analytics:

The system generates various reports and analytics to provide insights into hostel operations. These reports may include occupancy vacancies and special applications made by students.

## **How to run the code**

1. For the IDE, I used IntelliJ so you need to install IntelliJ and you will need to install the latest SDK from azul.com and JDK from gluonhq.com according to your operating system for the JavaFX application to run.
2. Open the folder HostelManagement inside the folder HostelManagement as a project in IntelliJ
3. Head on to File -> Project Structure and then select the JDK that you have installed as the SDK.
4. Head on to Global Libraries in the Project Structure page and then press the + button and press Java and then select your lib folder in Java SDK. (Example of path: JavaFX SDK (Version) -> java-sdk-version -> lib) then click OK.
5. Head on to SDKs in Project Structure page to set the JDK. Click on the + button then press Add JDK. If your downloaded the JDK from Zulu then select the zulu folder with format zulu(xx).(xx).(xx)-ca-jdk(version)-win\_x64 then click OK.
6. After you have finished the 2 previous steps, click Apply and OK
7. Head to File -> Settings -> Path Variables and click the + button to create a new variable. Find the lib folder in Java SDK folder. (Example of path: JavaFX SDK (Version) -> java-sdk-version -> lib) then click OK. Click Apply then OK.
8. Head on to File -> Project Structure -> Modules then select <Module Source> and then click the + button above the word Export then select the lib folder. Click Add Selected then Apply then OK.
9. Click the file FilesHandler.java in the src folder and then edit the following paths according to the paths of your own files (text in bold):

// Reading data of rooms

sc = new Scanner(new File("**C://Users//62819//Desktop//HostelManagement//HostelManagement//src//rooms.txt**"));

// Student Applications data

sc = new Scanner(new File("**C://Users//62819//Desktop//HostelManagement//HostelManagement//src//application.txt**"));

// Reading data from student file

sc = new Scanner(new File("**C://Users//62819//Desktop//HostelManagement//HostelManagement//src//students.txt**"));

// Add data to text file

FileWriter writer = new FileWriter("**C://Users//62819//Desktop//HostelManagement//HostelManagement//src//students.txt**", true);

// Writing data on file;

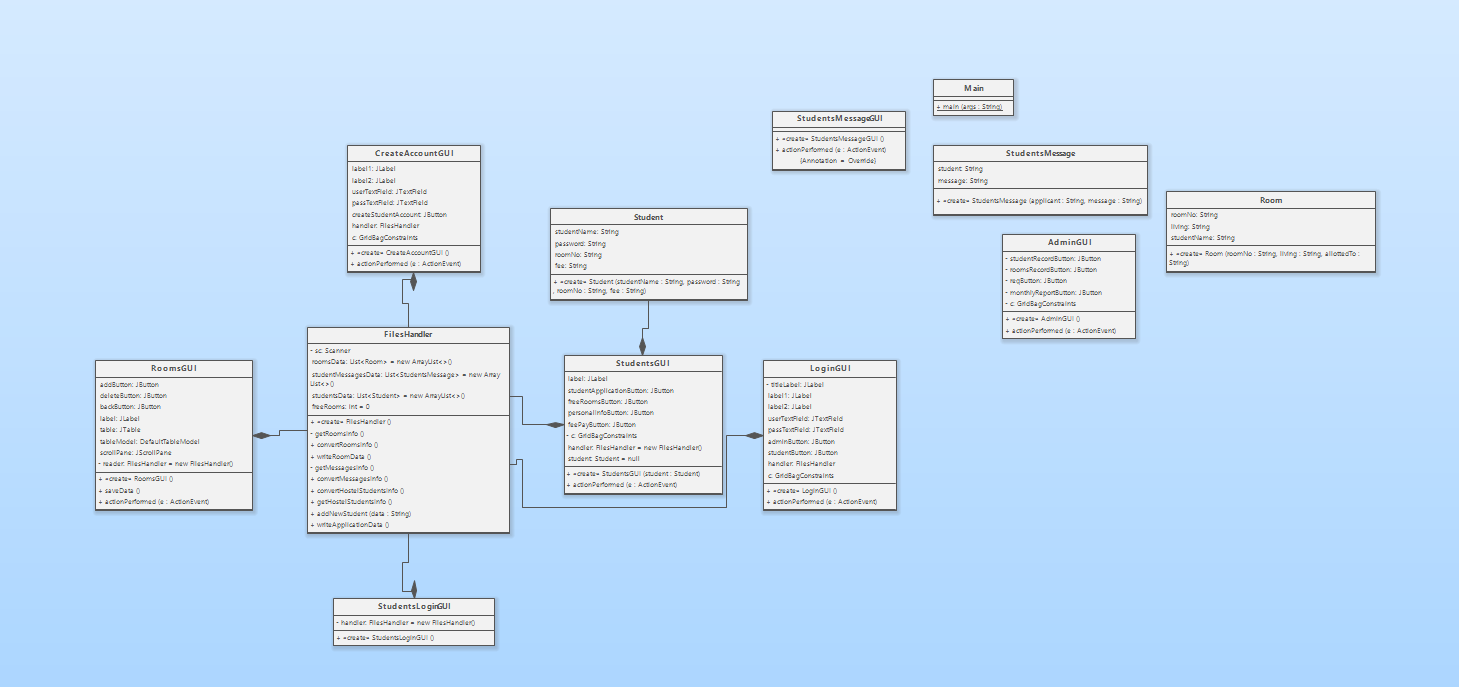
FileWriter writer = new FileWriter("**C://Users//62819//Desktop//HostelManagement//HostelManagement//src//application.txt**");

1. Run the code and the application will appear.

## **How the app works**

Starting off you will be on the login page. If you are a student, then create an account by pressing the Create Account button and type in the username and password. The admin details are default with username: admin and password: password. You would have to pay hostel fees before creating account, After creating the account student can login and access features such as making applications, checking available rooms, paying fees and checking personal record. In the admin page, admins can add/delete rooms, assign student’s rooms, check student’s reports, monthly report of the fees and look at applications created by students.

# Solution Design



## **Each class and its workflow:**

Main: It is a class that allows the main page of the system which is the Login GUI to load.

StudentsMessage: It is a class where the name of applicant along with the application made.

StudentsMessageGUI: It is class that allows the Applications to show on screen.

AdminGUI: It is a class that allows the buttons such as Display Students, Rooms Data, Display Applications and Manage Rooms to appear on screen.

Room: It is a class that set parameters of room number, occupancy and the student name.

FilesHandler: It is a class where are the files are read and written according to user’s actions.

StudentsGUI: It is a class that allows the buttons such as Create Applications, Available Rooms, Personal Record and Fee Payment to appear on screen.

CreateAccountGUI: It is a class where students can create account by inputting their preferred username and password.

StudentsLoginGUI: It is a class that displays the students data.

RoomsGUI: It is a class that displays the rooms data along with the Add Room, Delete Room, and Back button on screen.

LoginGUI: It is a class that creates the login page components such as username’s text field, password’s text field, student login button, admin login button and create account button (text).

Student: It is a class that set parameters such as StudentName, Password, room number, fees (paid/unpaid)

# What was implemented and how it works

## **Main class**

A picture containing text, screenshot, software, font

Description automatically generated

The main class is just a basic Java program that creates an instance of the LoginGUI class and makes it visible. The Main class is declared as a public class then the main method acts as the entry point of the program. Inside the main method, a new instance of the LoginGUI class is created and assigned to the variable loginGUI. The setVisible(true) method is called on the LoginGUI object, which makes the login page visible.

## **Room class**

A picture containing text, screenshot, font, software

Description automatically generated

This class represents room information and has three instance variables: roomNo, living, and studentName.

The Room class has a constructor that takes three parameters: roomNo, living, and allottedTo.

These parameters are used to initialize the instance variables of the class.

The roomNo variable represents the room number, which is a string. The living variable represents the living status of the room, indicating whether the room is available or unavailable.

The studentName variable represents the name of the student to whom the room is allotted. This is also a string.

## **Student class**

A picture containing text, screenshot, software, display

Description automatically generated

This class represents a student’s information and has four instance variables: studentName, password, roomNo, and fee.

The Student class has a constructor that takes four parameters: studentName, password, roomNo, and fee. These parameters are used to initialize the instance variables of the class.

The studentName variable represents the name of the student, which is a string.

The password variable represents the password of the student, which is also a string.

The roomNo variable represents the room number allotted to the student, which could is a string.

The fee variable represents the fee or payment information related to the student, which state whether fee is paid or unpaid.

## **AdminGUI class**

The AdminGUI class creates a GUI for the admin panel with various components such as buttons and labels. It imports necessary packages for GUI components and event handling. The class declares instance variables for buttons (studentRecordButton, roomsRecordButton, reqButton, and monthlyReportButton) and a grid bag constraints object (c). The class defines the constructor AdminGUI() that initializes the GUI components, sets up the layout, and configures the frame properties. The constructor creates a grid bag layout and a grid bag constraints object to position the components within the frame. It sets up the font and creates labels and buttons with their respective dimensions, alignments, and event listeners. The components are added to the frame using the grid bag constraints to specify their positions.

The class overrides the actionPerformed(ActionEvent e) method from the ActionListener interface to handle button clicks and perform appropriate actions based on the source of the event.

A picture containing text, font, screenshot

Description automatically generated

If the "Display Students" button is clicked, it creates a new instance of the StudentsLoginGUI class, which displays student information.

A screen shot of a computer code

Description automatically generated with low confidence

If the "Rooms Data" button is clicked, it disposes of the current frame and creates a new instance of the RoomsGUI class, which displays information about rooms.

A screen shot of a computer code

Description automatically generated with low confidence

If the "Display Applications" button is clicked, it creates a new instance of the StudentsMessageGUI class, which displays student applications/

A picture containing text, font, screenshot, line

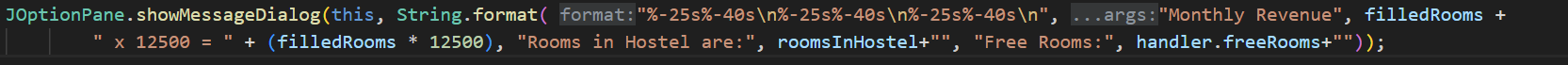
Description automatically generated

If the "Monthly Report" button is clicked, it creates an instance of the FilesHandler class, calculates the number of filled rooms and revenue, and displays the information using a message dialog.

A screen shot of a computer code

Description automatically generated with low confidence

The code below this message is how the monthly report is counted.



## **CreateAccountGUI class**

* The class extends JFrame to create a window for the GUI.
* The GUI layout is set using GridBagLayout, which allows components to be arranged in a grid with flexible positioning.
* The FilesHandler object is created to handle file operations and data management.
* Various GUI components such as labels, text fields, and buttons are initialized and configured.
* The actionPerformed method is implemented from the ActionListener interface, which handles the event when the "Create Account" button is clicked.
* Inside the actionPerformed method, the entered username and password are retrieved from the text fields.
* If either the username or password is empty, a message dialog is shown to prompt the user to enter both fields.
* If both fields are filled, a confirmation dialog is shown to ask the user to pay the hostel fees.
* If the user confirms the payment, a success message is shown, and the student's data is added to the text file using the addNewStudent method from the FilesHandler class.
* Finally, the GUI is closed, and a new instance of the LoginGUI class is created to go back to the login screen.

## **FilesHandler class**

* The class uses the Scanner class to read data from text files and store the data in lists (roomsData, studentMessagesData, and studentsData).
* The lists are used to store the data retrieved from the files.
* The class also defines three two-dimensional arrays (roomArray, studentMessagesArray, and studentsArray) to hold the data in a format suitable for displaying in GUI components like JTables.
* The constructor FilesHandler() is responsible for reading the data from the text files and populating the lists and arrays by calling various helper methods.
* The getRoomsInfo() method reads the data from the "rooms.txt" file and creates Room objects. It also counts the number of available rooms (freeRooms).
* The convertRoomsInfo() method converts the list of rooms (roomsData) into a two-dimensional array (roomArray).
* The writeRoomData() method writes the data from the roomsData list back to the "rooms.txt" file.
* The getMessagesInfo() method reads the student application data from the "application.txt" file and creates StudentsMessage objects.
* The convertMessagesInfo() method converts the list of student messages (studentMessagesData) into a two-dimensional array (studentMessagesArray).
* The convertHostelStudentsInfo() method converts the list of students (studentsData) into a two-dimensional array (studentsArray).
* The getHostelStudentsInfo() method reads the student data from the "students.txt" file and creates Student objects.
* The addNewStudent() method appends new student data to the "students.txt" file and updates the lists and arrays accordingly.
* The writeApplicationData() method writes the student application data from the studentMessagesData list back to the "application.txt" file.

## **StudentsGUI class**

* The StudentsGUI class creates a GUI for the student's page with various functionalities.
* It imports necessary packages for GUI components and file handling.
* The class defines a constructor StudentsGUI(Student student) that initializes the GUI and sets up its properties.
* The constructor configures the frame's properties, such as the default close operation, title, size, layout, and font.
* Components such as labels and buttons are created, with their properties set, including text, alignment, dimensions, and action listeners.
* The components are added to the frame using the GridBagLayout and GridBagConstraints for positioning and alignment.
* The constructor sets the frame to be visible.
* The class also implements the ActionListener interface, which allows it to handle button actions.
* The actionPerformed method is overridden to provide functionality for each button.
* If the Create Application Button is clicked, an input dialog is shown to get the application text, which is then saved to a file using the FilesHandler class. A message dialog is displayed to indicate that the application has been submitted.
* If the Available Rooms button is clicked, available rooms are retrieved from the FilesHandler class and displayed in a message dialog.
* If the Personal Record Button is clicked, the student's personal information is retrieved from the Student object and displayed in a message dialog.
* If the Fee Payment Button is clicked, the student's fee status is updated, and the student data is saved using the FilesHandler class. A message dialog is displayed to indicate that the fees have been paid.

## **StudentsLoginGUI class**

* The StudentsLoginGUI class creates a dialog box GUI for displaying student records.
* It imports necessary packages for GUI components and file handling.
* The class defines a constructor StudentsLoginGUI() that initializes the dialog box and sets up its properties.
* The constructor configures the dialog box's properties, such as the default close operation, title, size, layout, and location.
* A heading label is created and added to the dialog box's north region. The label displays the text "Students" and its font size is set to 16.
* An instance of the FilesHandler class is created to handle file operations.
* A JTable component is created using the student data retrieved from the FilesHandler. The table displays the columns "Name of student", "Password", "Room No", and "Fees". The JTable is then wrapped in a JScrollPane to enable scrolling functionality. The JScrollPane is added to the center of the dialog box using the BorderLayout.CENTER position. Finally, the dialog box is set to be visible.

A screen shot of a computer program

Description automatically generated with low confidence

## **StudentsMessageGUI class**

The StudentsMessageGUI class creates a dialog box GUI for displaying student applications. It imports necessary packages for GUI components and file handling. The class defines a constructor StudentsMessageGUI() that initializes the dialog box and sets up its properties. The constructor configures the font, sets the size, title, and layout of the dialog box. It creates a heading label to display "Student Applications" and sets its font, size, and alignment. The FilesHandler class is used to read student applications from a file and retrieve the data as a two-dimensional array. A JTable component is created using the data retrieved from the FilesHandler. The table displays the student name and their application. The table's font is set to "Consolas" with a size of 14. The table is added to the center of the dialog box using the BorderLayout.CENTER position. If there is an exception while reading the file (e.g., FileNotFoundException), an error message is displayed using a JOptionPane. A "Close" button is created and added to the bottom of the dialog box using the BorderLayout.PAGE\_END position. An action listener is added to the "Close" button to handle the event when it is clicked. When clicked, the dialog box is disposed of, closing it. The dialog box is set to be visible.

A screenshot of a computer program

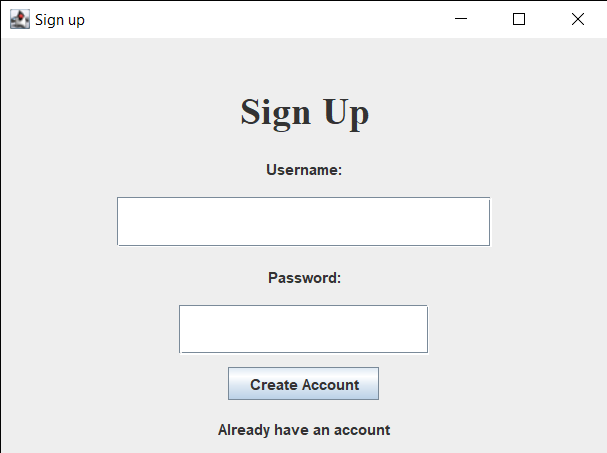
Description automatically generated with medium confidence

A screen shot of a computer code

Description automatically generated with low confidence

# Evidence of Working Program

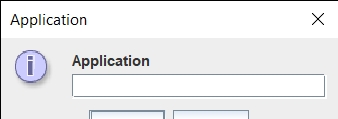
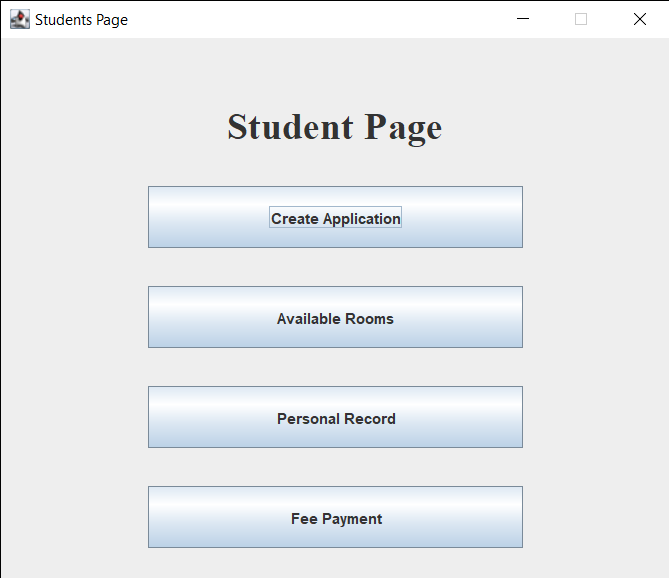
A screenshot of a computer login

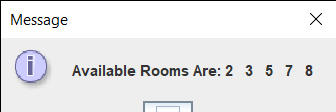
Description automatically generated with medium confidence

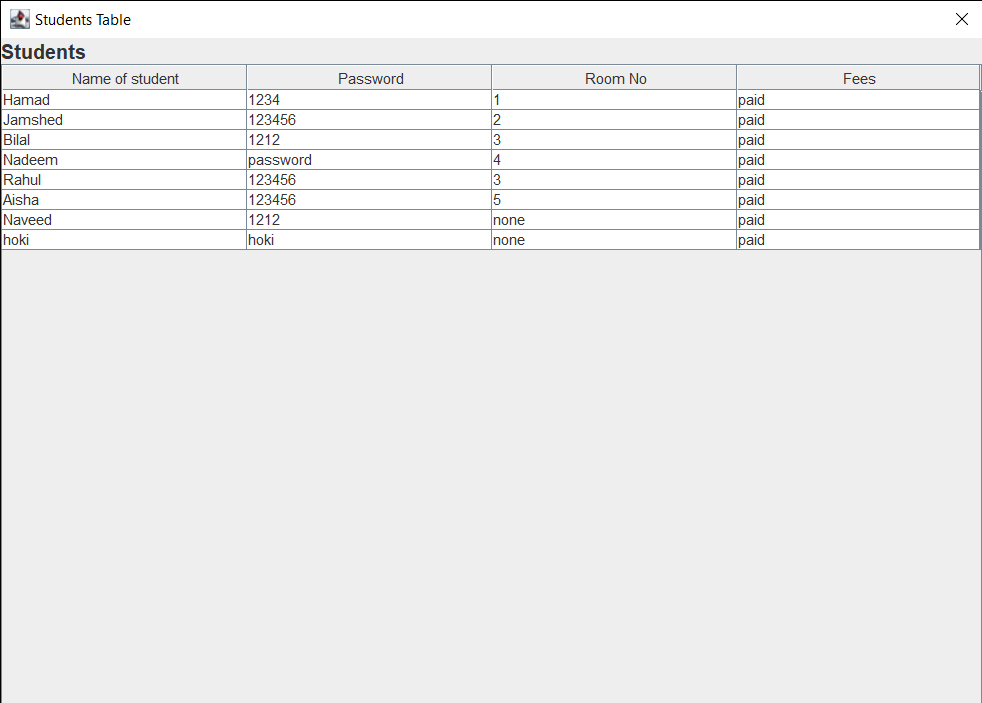
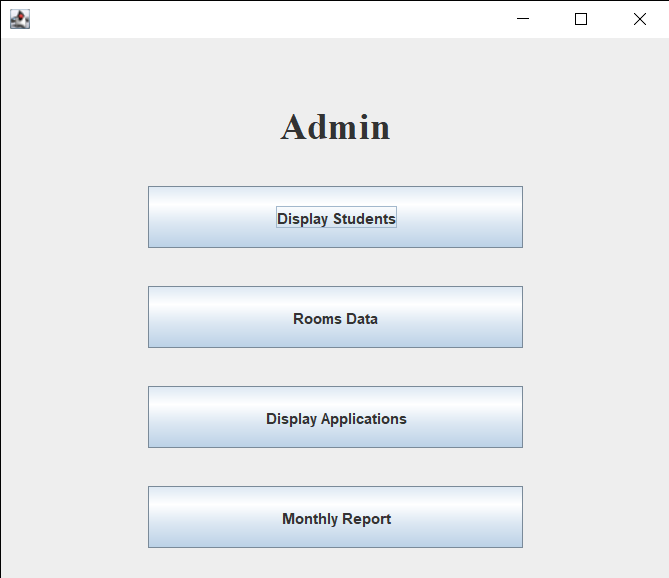
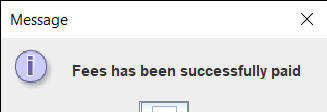
A screen shot of a computer

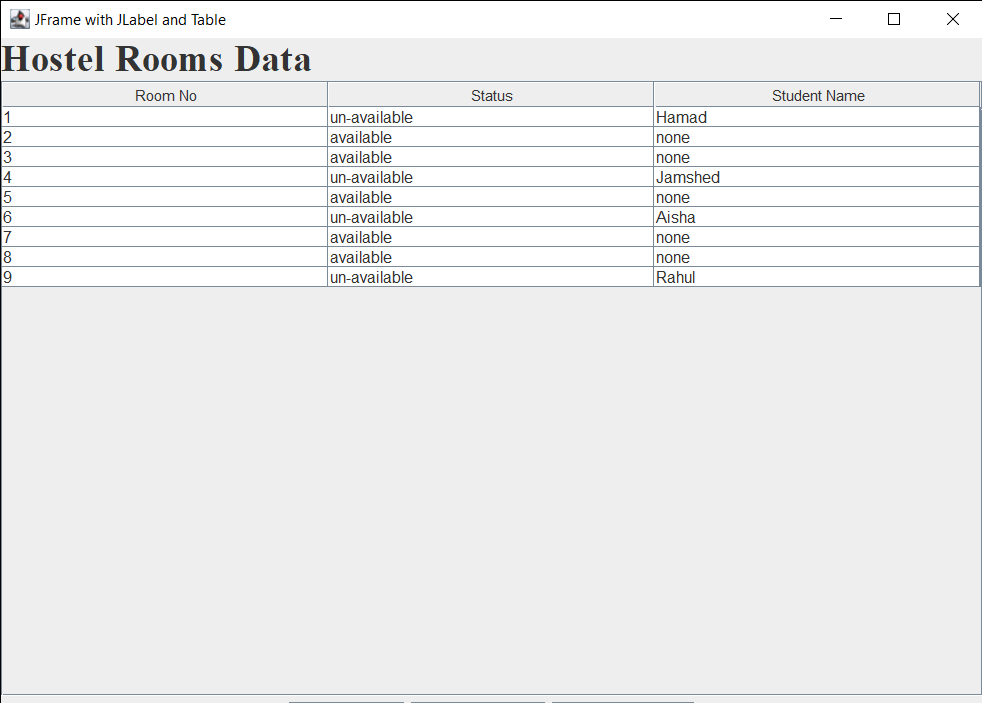
Description automatically generated with medium confidenceA screen shot of a computer

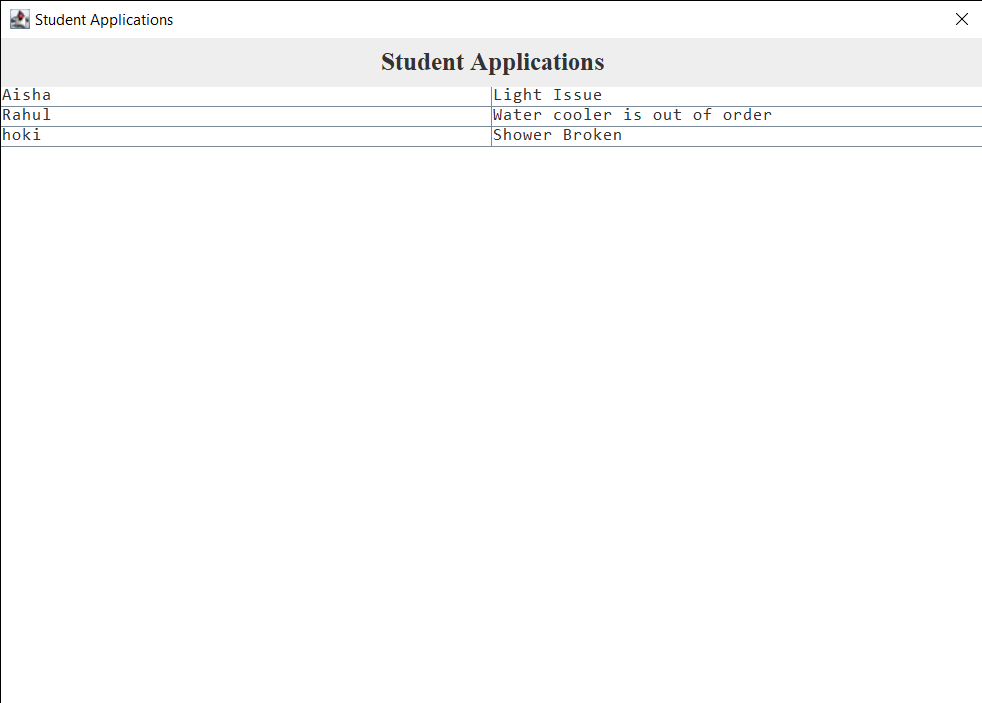
Description automatically generated with medium confidence

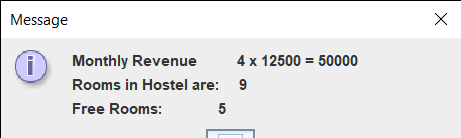
A screen shot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated







## **Lesson Learned**

Through this project in Java, ceating a hostel management system in Java can provide valuable lessons and insights.

* Understanding requirements: Developing a hostel management system requires a clear understanding of the requirements and functionality needed. It involves considering factors such as room allocation, student information and payment management. Gathering and analyzing requirements thoroughly is crucial for building an effective system.
* Object-oriented design: Java's object-oriented nature lends itself well to modeling the entities and relationships within a hostel management system. Designing classes and their interactions helps create a modular and maintainable codebase. Applying principles like encapsulation, inheritance, and polymorphism allows for more manageable and extensible code.
* File handling: Working with files is a common aspect of many software applications, including a hostel management system. Handling file operations, such as reading and writing data, requires understanding the appropriate Java APIs. Dealing with exceptions, file paths, and data serialization are essential skills in this context.
* Graphical User Interface (GUI) development: Building a user-friendly GUI enhances the usability of the system. Learning GUI frameworks in Java, such as Swing or JavaFX, enables the creation of visually appealing interfaces with interactive components like buttons, text fields, and tables. Designing intuitive layouts and handling user input are crucial aspects of GUI development.
* Exception handling: Dealing with exceptions is an integral part of robust software development. Hostel management systems may encounter exceptions related to file I/O, data validation, or external dependencies. Properly handling exceptions, logging errors, and providing meaningful error messages contribute to system reliability and maintainability.