



LAB FINAL PROJECT

Programming Fundamentals

Instructor: Mr. Rizwan Rashid

Student Name:	Nashama Asim Muhammad Ahmad Harram Khan Tehreem Javed
----------------------	--

Registration No:	FA23-BDS-056 FA23-BDS-048 FA23-BDS-014 FA23-BDS-041
-------------------------	--

Department:	Data Science
--------------------	--------------

Submission Date:	28 th May 2024
-------------------------	---------------------------

COMSATS University Islamabad

Contents

Introduction.....	4
Task Distribution:.....	4
Features.....	5
1. User Authentication	6
2. Application Processing	6
3. Fee Management.....	6
4. Room Allocation.....	6
5. Notification System	6
6. Request Approval	6
7. Visitor Log Maintenance	6
8. Invoice Generation.....	6
9. File Handling	6
10. Entities	6
What makes our Management System Unique:	6
Use of Exceptional Handling:	6
Use of File Handling:	6
Components of Code:	7
Main Method:.....	7
'`main(String[] args)`:.....	7
File Creation Methods:.....	7
1. `createFiles()`:	7
2. `createRoomOccupancyFile()`:.....	7
Admin Portal Methods (Entity 1):	8
1. `adminPortal()`:	8
2. `adminLogIn()`:.....	8
3. `applications()`:	8
4. `reviewFeeSubmission()`:	8
5. `viewAllFeeSubmissions()`:	9
6. `changeFeeStatus()`:.....	9
7. `adminNotificationManagement()`:	9
8. `currentRoomResidents()`:.....	9

9. viewNotifications():	9
10. removeNotification(int lineNumber):	10
11. `visitorDetails()`	10
Hostelite Portal Methods (Entity 2):	10
1. `hostelitePortal()`	10
2. `hosteliteLogin(String userInHostelID)`	10
3. `displayApplication(String userInHostelID)`	11
4. hosteliteNotificationSubmission(String hostelID)	11
5. `studentFeeSubmission(String hostelID)`	11
6. `roomChangeOrVacant(String hostelID)`	11
7. `checkRoomOccupancy(String hostelID, int roomNo)`	11
8. `new_application()`	12
9. `isValidName(String str)`	12
10. `isValidNumericInput(String str)`	12
11. `idCreator()`	12
12. `generateInvoice()`	12
13. `hostelRules()`	13
Output Screenshots and Code Description:	13
Create a new file:	13
Main Menu	14
Admin Portal	16
View Application Code:	18
Viewing all applications:	18
Admin Notification Management	19
Admin Fee Review Portal:	20
Room Occupation	23
Visitor Log	24
Hostelite Portal	27
SignUp	29
Creating An Application:	29
Fee Invoice:	32
Hostel Rules	33

INTRODUCTION

This Java-based Hostel Management System streamlines hostel administration by automating key processes. It collects and validates applicants' personal details, assigns unique Hostel IDs, and allows room selection from available options, updating occupancy status dynamically. The system also manages fee submissions, enabling students to check their fee status and administrators to review and update fee records, enhancing transparency.

Additional features include managing application details, handling room change or vacation requests, generating administrative reports, and sending notifications. Overall, it maximizes operational efficiency and improves the experience for both administrators and residents.

TASK DISTRIBUTION:

Function	Member
main	nashama
createFiles	nashama
createRoomOccupancyFile	nashama
adminPortal	nashama
adminLogin	nashama
hostelitePortal	nashama
hosteliteLogin	nashama
visitorDetails	nashama
generateInvoice	nashama
new_application	nashama/ahmad
isValidName	ahmad
isValidNumericInput	ahmad
reviewFeeSubmission	ahmad

viewAllFeeSubmissions	ahmad
studentFeeSubmission	ahmad
changeFeeStatus	ahmad
adminNotificationManagement	Tehreem
viewRoomShiftRequests	Tehreem
currentRoomResidents	Tehreem
viewNoification	Tehreem
removeNotification	Tehreem
displayApplication	Tehreem
hosteliteNotificationSubmission	Tehreem
roomChangeOrVacant	harram
checkRoomOccupancy	harram
idCreator	harram
applications	harram
hostelRules	harram

FEATURES:

The Hostel Management System boasts a comprehensive suite of features designed to streamline hostel operations and enhance user experience. Some of the key features include:

1. **User Authentication:** Secure login for admins and residents to protect sensitive information.
2. **Application Processing:** Easy submission and tracking of hostel applications.
3. **Fee Management:** Automated calculation and detailed breakdown of various hostel fees, including accommodation, internet, laundry, and utilities.
4. **Room Allocation:** Efficient allocation of rooms based on availability and resident preferences.
5. **Notification System:** Real-time notifications for room shift requests, vacating notices, and general announcements.
6. **Request Approval:** Admin functionality to approve or deny room shift requests, with automatic updates to the resident's status.
7. **Visitor Log Maintenance:** Record-keeping of visitors for security and administrative purposes.
8. **Invoice Generation:** Automated generation of detailed invoices for residents, covering all applicable fees.
9. **File Handling:** Robust file handling to manage and store data securely, with efficient reading and writing operations.
10. **Entities:** The entities are hostelites and admin.

What makes our Management System Unique:

What makes this system unique is its focus on automation and user-friendliness. By automating routine tasks like fee calculations, room allocation, and notification management, the system significantly reduces administrative workload and minimizes human errors. This ensures a smooth and efficient operation, allowing administrators to focus on more critical tasks.

Use of Exceptional Handling:

Exceptional handling in this system includes comprehensive error handling to manage scenarios like file not found, input mismatches, and invalid entries. For instance, when reading from files, the system checks for potential `IOExceptions` and provides user-friendly error messages to help diagnose issues. It also incorporates mechanisms to ensure data consistency and integrity during read/write operations, such as ensuring proper file closing and synchronization when updating files.

Use of File Handling:

File handling is a core aspect of the system, with structured methods to read from and write to files like `roomOccupancy.txt`, `request.txt`, and `notification.txt`. The system uses efficient file handling techniques to manage data, such as buffered reading and writing, which enhances performance and reliability. This approach ensures that all data is accurately recorded and easily retrievable, contributing to the system's overall robustness and effectiveness.

Hence, by integrating these features and handling mechanisms, the Hostel Management System provides a reliable, efficient, and user-friendly solution for managing hostel operations, ensuring a better experience for both administrators and residents.

COMPONENTS OF CODE:

Main Method:

`main(String[] args):`

- a) **Description**: The main method serves as the entry point for the program. It typically handles user interaction, displays a menu, and calls other methods based on user input.
- b) **Purpose**: To provide a user interface for interacting with the application, allowing users to view hostel management details.
- c) **Input**: args (String[]): Command-line arguments passed to the program (not used in this context).
- d) **Output**: Displays a menu to the user, processes user input to perform different operations, and prints relevant messages to the console

File Creation Methods:

We have used two methods to create files:

1. `createFiles()`:

- a) **Description**: This method is responsible for creating necessary files for the hostel management system, such as application records, room occupancy details, visitor logs, notifications, fee submissions, etc.
- b) **Purpose**: Ensures that all required files exist for storing essential data related to hostel operations.
- c) **Input**: None.
- d) **Output**: Creates files if they do not exist and logs a message indicating whether the file was created successfully or if it already exists.

2. `createRoomOccupancyFile()`:

- a) **Description**: This method creates and populates the room occupancy file if it is empty. It initializes room occupancy data with all rooms marked as vacant.
- b) **Purpose**: Ensures that the room occupancy file exists and is populated with initial data for tracking room allocations.
- c) **Input**: None.

- d) **Output:** Creates or populates the room occupancy file and logs appropriate messages to indicate the status of the operation.

Admin Portal Methods (Entity 1):

1. `adminPortal()`:

- a) **Description:** This method represents the portal for administrators to manage various aspects of hostel operations. It includes functionalities such as viewing applications, notifications, fee submissions, room occupancy, and visitor logs.
- b) **Purpose:** Provides administrators with a centralized interface to monitor and manage hostel-related activities efficiently.
- c) **Input:** None.
- d) **Output:** Displays a menu of options for administrators to choose from, executes the selected functionality, and handles user input validation.

2. `adminLogin()`:

- a) **Description:** This method handles the login process for administrators by validating the entered password against a predefined password.
- b) **Purpose:** Ensures secure access to the admin portal by authenticating administrator credentials.
- c) **Input:** None.
- d) **Output:** Prompts the administrator to enter the login password, validates it against the predefined password, and returns a flag indicating the login status.

3. `applications()`:

- a) **Description:** This method displays all submitted hostel accommodation applications along with their details.
- b) **Purpose:** Provides administrators with a comprehensive view of all accommodation applications received.
- c) **Input:** None.
- d) **Output:** Prints the details of all submitted applications (hostel ID, password, personal information, and current room number) to the console.

4. `reviewFeeSubmission()`

- a) **Description:** This method provides an interface for administrators to review fee submissions and perform related actions.
- b) **Purpose:** Allows administrators to view all fee submissions and change the fee status of a student.
- c) **Input:** User's choice to view fee submissions, change fee status, or return to the admin menu.

- d) **Output:** Displays options for administrators to choose from and executes the selected functionality, such as viewing fee submissions or changing fee status.

5. ``viewAllFeeSubmissions()``:

- a) **Description:** This method displays all fee submissions along with their corresponding hostel IDs and fee statuses.
- b) **Purpose:** Provides administrators with an overview of all fee submissions made by hostellites.
- c) **Input:** None.
- d) **Output:** Prints the hostel ID and fee status of all fee submissions to the console.

6. ``changeFeeStatus()``:

- a) **Description:** This method allows administrators to change the fee status (paid/unpaid) of a student.
- b) **Purpose:** Facilitates administrators in updating the fee status of hostellites as paid or unpaid.
- c) **Input:** Hostel ID of the student whose fee status needs to be changed and the new fee status.
- d) **Output:** Updates the fee status of the specified student in the fee file and notifies the administrator of the status update.

7. ``adminNotificationManagement()``:

- a) **Description:** This method provides administrators with an interface to manage notifications and requests submitted by hostellites.
- b) **Purpose:** Allows administrators to view, approve, or remove notifications and requests.
- c) **Input:** User's choice to view notifications, approve requests, or return to the admin portal.
- d) **Output:** Displays options for administrators to manage notifications and requests, executes the selected functionality, and handles user input accordingly.

8. ``currentRoomResidents()``:

- a) **Description:** This method displays the current residents of each room along with their hostel IDs.
- b) **Purpose:** Offers administrators a snapshot of room occupancy and current residents.
- c) **Input:** None.
- d) **Output:** Prints the room number and hostel IDs of current residents to the console.

9. `viewNotifications()`:

- a) **Description:** This method reads notifications from a file and displays them to the console. Each notification contains a hostel ID and a message.
- b) **Purpose:** To provide hostellites with a list of current notifications, allowing them to stay informed about relevant updates or announcements.
- c) **Input:** None.

- d) **Output:** Prints the notifications to the console. If there are no notifications, it prints a message stating that there are no notifications. If there is an error reading the file, it prints an error message.

10.removeNotification(int lineNumber):

- a) **Description:** This method removes a specific notification from the notification.txt file based on the given line number.
- b) **Purpose:** To allow users to delete a specific notification from the list, keeping the notification file up to date and manageable.
- c) **Input:** lineNumber (int): The line number of the notification to be removed.
- d) **Output:** Prints a message to the console indicating whether the notification was successfully removed or if there was no notification to remove at the specified line number. If there is an error reading or writing the file, it prints an error message.

11. `visitorDetails()`:

- a) **Description:** This method allows administrators to manage visitor logs by viewing previous visitor details or logging new visitor information.
- b) **Purpose:** Facilitates tracking of visitors for security and management purposes within the hostel.
- c) **Input:** Administrator's choice for viewing previous visitors or logging a new visitor, along with visitor details if logging a new visitor.
- d) **Output:** Displays options for viewing or logging visitors and performs the respective operations based on the administrator's choice.

HOSTELITE PORTAL METHODS (ENTITY 2):

1. `hostelitePortal()`:

- a) **Description:** This method represents the portal for hostellites (residents) to access various services provided by the hostel management system, such as viewing applications, fee submission status, room change requests, etc.
- b) **Purpose:** Provides hostellites with a user-friendly interface to manage their hostel-related activities and requests.
- c) **Input:** None.
- d) **Output:** Displays a menu of options for hostellites to choose from, executes the selected functionality, and handles user input validation.

2. `hosteliteLogin(String userInHostelID)`:

- a) **Description:** This method handles the login process for hostellites by validating the entered hostel ID and password against the records stored in the application file.
- b) **Purpose:** Ensures secure access to the hostelite portal by authenticating hostelite credentials.
- c) **Input:** Hostel ID entered by the hostelite.

- d) **Output:** Prompts the hostelite to enter their password, validates it against the stored password for the corresponding hostel ID, and returns a flag indicating the login status.

3. ``displayApplication(String userInHostelID)`` :

- a) **Description:** This method displays the application details of a hostelite based on their hostel ID.
- b) **Purpose:** Allows administrators to view the application details of a specific hostelite.
- c) **Input:** Hostel ID of the hostelite whose application details are to be displayed.
- d) **Output:** Prints the application details (hostel ID, password, personal information, and room number) of the specified hostelite to the console.

4. `hosteliteNotificationSubmission(String hostelID)`

- a) **Description:** This method allows a hostelite to submit a notification by entering a message. The message, along with the hostel ID, is appended to the 'notification.txt' file.
- b) **Purpose:** To provide a mechanism for hostelites to submit notifications that can be shared with others or stored for record-keeping.
- c) **Input:** hostelID (String): The ID of the hostelite submitting the notification.
- d) **Output:** Prompts the user to enter a message and appends the message along with the hostel ID to the 'notification.txt' file. It prints a confirmation message upon successful submission. If there is an error writing to the file, it prints an error message.

5. ``studentFeeSubmission(String hostelID)`` :

- a) **Description:** This method retrieves and displays the fee status of a hostelite based on their hostel ID.
- b) **Purpose:** Allows hostelites to check their fee submission status.
- c) **Input:** Hostel ID of the hostelite whose fee status is to be retrieved.
- d) **Output:** Prints the fee status (paid/unpaid) of the specified hostelite to the console.

6. ``roomChangeOrVacant(String hostelID)`` :

- a) **Description:** This method allows hostelites to request room changes or indicate their intention to vacate their current room.
- b) **Purpose:** Facilitates hostelites in submitting requests for room changes or indicating their intention to vacate their room.
- c) **Input:** Hostel ID of the hostelite initiating the request and their choice to shift rooms or vacate.
- d) **Output:** Processes the hostelite's choice and updates the request file accordingly.

7. ``checkRoomOccupancy(String hostelID, int roomNo)`` :

- a) **Description:** This method checks if a room is occupied by a specific hostelite based on the hostel ID and room number.
- b) **Purpose:** Verifies the occupancy status of a room to prevent double allocation or unauthorized room changes.
- c) **Input:** Hostel ID of the hostelite and the room number to be checked for occupancy.
- d) **Output:** Returns ``true`` if the room is occupied by the specified hostelite, otherwise returns ``false``.

8. `new_application()`

- a) **Description:** This method handles the process of creating a new application for hostel accommodation, collecting applicant details, assigning hostel ID, selecting a room, and saving the application data.
- b) **Purpose:** Enables new applicants to apply for hostel accommodation and manages their application details.
- c) **Input:** Applicant details such as name, date of birth, contact information, guardian details, etc.
- d) **Output:** Guides the applicant through the application process, assigns a unique hostel ID, selects a room based on availability, generates an invoice, and saves the application data.

9. `isValidName(String str)`

- a) **Description:** This method checks if the input string contains only alphabetic characters and spaces, ensuring it is a valid name.
- b) **Purpose:** Validates the format of names entered by users to ensure data integrity.
- c) **Input:** String to be validated.
- d) **Output:** Returns `true` if the string contains only alphabetic characters and spaces, indicating a valid name; otherwise, returns `false`.

10. `isValidNumericInput(String str)`:

- a) **Description:** This method checks if the input string contains only numeric characters and hyphens, ensuring it is a valid numeric input.
- b) **Purpose:** Validates the format of numeric inputs entered by users to ensure data integrity.
- c) **Input:** String to be validated.
- d) **Output:** Returns `true` if the string contains only numeric characters and hyphens, indicating a valid numeric input; otherwise, returns `false`.

11. `idCreator()`:

- a) **Description:** This method generates a unique hostel ID for new applicants by randomly generating an ID and checking if it already exists in the application records.
- b) **Purpose:** Ensures that each new applicant is assigned a unique hostel ID to identify their application and accommodation.
- c) **Input:** None.
- d) **Output:** Generates a unique hostel ID and returns it to be assigned to the new applicant.

12. `generateInvoice()`:

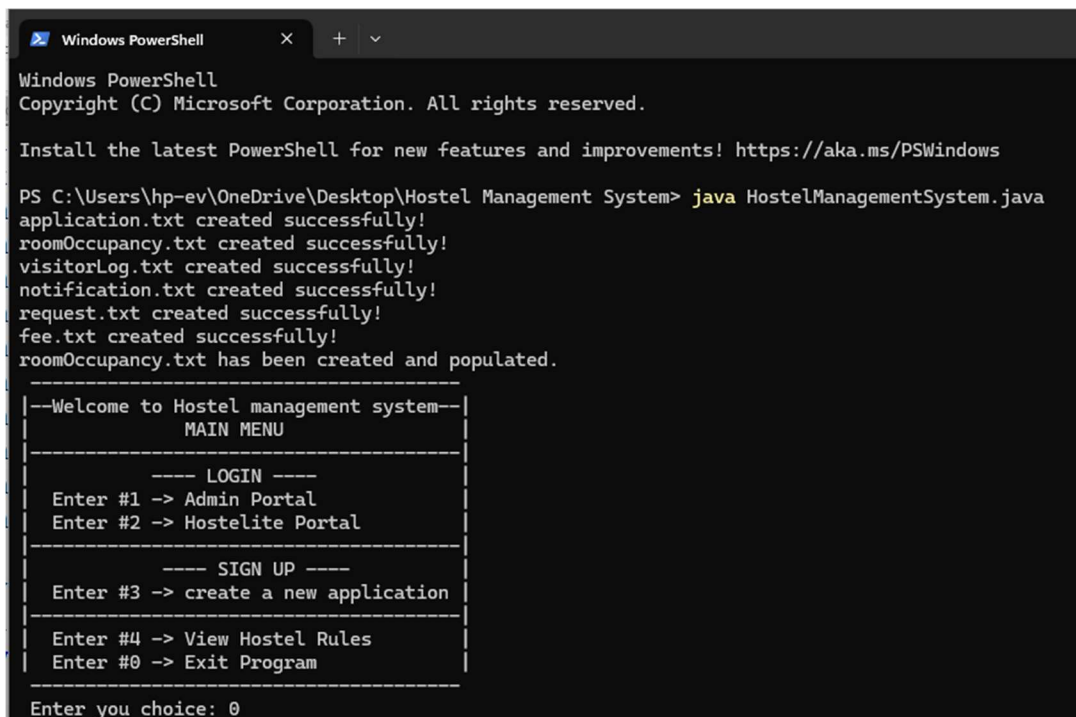
- a) **Description:** This method generates an invoice detailing various fees payable by hostellites.
- b) **Purpose:** Provides hostellites with a breakdown of accommodation and additional fees for payment.
- c) **Input:** None.
- d) **Output:** Prints an itemized invoice of accommodation fees, security deposit, and other charges to the console.

13. `hostelRules()`:

- a) **Description:** This method displays the rules and guidelines for residents of the hostel.
- b) **Purpose:** Informs residents about the expected behavior and regulations within the hostel premises.
- c) **Input:** None.
- d) **Output:** Prints out the hostel rules and guidelines to the console for residents to read and adhere to.

OUTPUT SCREENSHOTS AND CODE DESCRIPTION:

Create a new file:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\hp-ev\OneDrive\Desktop\Hostel Management System> java HostelManagementSystem.java
application.txt created successfully!
roomOccupancy.txt created successfully!
visitorLog.txt created successfully!
notification.txt created successfully!
request.txt created successfully!
fee.txt created successfully!
roomOccupancy.txt has been created and populated.

-----
--Welcome to Hostel management system--
-----
                MAIN MENU
-----
                ---- LOGIN ----
Enter #1 -> Admin Portal
Enter #2 -> Hostelite Portal
-----
                ---- SIGN UP ----
Enter #3 -> create a new application
-----
Enter #4 -> View Hostel Rules
Enter #0 -> Exit Program
-----
Enter you choice: 0
```

Show file already exists:

```

Enter you choice: 0
PS C:\Users\hp-ev\OneDrive\Desktop\Hostel Management System> java HostelManagementSystem.java
application.txt already exists.
roomOccupancy.txt already exists.
visitorLog.txt already exists.
notification.txt already exists.
request.txt already exists.
fee.txt already exists.
The file is not empty. No changes were made.

```

```

-----
|--Welcome to Hostel management system--|
|              MAIN MENU              |
|-----|
|              ---- LOGIN ----         |
| Enter #1 -> Admin Portal              |
| Enter #2 -> Hostelite Portal          |
|-----|
|              ---- SIGN UP ----       |
| Enter #3 -> create a new application  |
|-----|
| Enter #4 -> View Hostel Rules         |
| Enter #0 -> Exit Program              |
|-----|
Enter you choice:

```

Main Menu

In Main Menu there are total 3 options. Either to login as admin or hostelite, to creator new application in signup and to view rules.

Here is the code for main menu.

```

public static void main(String[] args) {
    createFiles();
    createRoomOccupancyFile();
    boolean exitProgram = false;
    Scanner input = new Scanner(System.in);

    while (!exitProgram) {
        System.out.println(" \033[32;1m----- " );
        System.out.println("|--Welcome to Hostel management system--|");
        System.out.println("|              MAIN MENU              |");
        System.out.println("|-----|");
        System.out.println("|              ---- LOGIN ----         |");
        System.out.println("| Enter #1 -> Admin Portal              |");
        System.out.println("| Enter #2 -> Hostelite Portal          |");
        System.out.println("|-----|");
        System.out.println("|              ---- SIGN UP ----       |");
        System.out.println("| Enter #3 -> create a new application  |");
        System.out.println("|-----|");
        System.out.println("| Enter #4 -> View Hostel Rules         |");
        System.out.println("| Enter #0 -> Exit Program              |");
        System.out.println("-----\033[0m");
    }
}

```

```

System.out.print(" Enter you choice: ");
String userInput1 = input.next();

if (userInput1.equals("1")) {
    adminPortal();
} else if (userInput1.equals("2")) {
    hostelitePortal();
} else if (userInput1.equals("3")) {
    new_application();
} else if (userInput1.equals("4")) {
    hostelRules();
} else if (userInput1.equals("0")) {
    exitProgram = true;
} else {
    System.out.println("Invalid choice.");
}
}
}

```

The user is not empty, no changes were made.

```

-----
|--Welcome to Hostel management system--|
|           MAIN MENU           |
|-----|
|           ---- LOGIN ----      |
| Enter #1 -> Admin Portal        |
| Enter #2 -> Hostelite Portal    |
|-----|
|           ---- SIGN UP ----    |
| Enter #3 -> create a new application |
|-----|
| Enter #4 -> View Hostel Rules    |
| Enter #0 -> Exit Program         |
|-----|
Enter you choice:

```

Admin Portal And Admin Login

Below is the code to check admin portal password. If password is correct admin portal is logged in

```
public static void adminPortal() {
    int loginStatus = adminLogIn();
    Scanner input = new Scanner(System.in);
    boolean exitProgram = false;
    if (loginStatus == 2) {
        while (!exitProgram) {
            System.out.println(" ----- ");
            System.out.println(" |          ---- ADMIN PORTAL ----          |");
            System.out.println(" | Enter #1 -> view all applications          |");
            System.out.println(" | Enter #2 -> see all notifications          |");
            System.out.println(" | Enter #3 -> Review Hotel Fee Submissions  |");
            System.out.println(" | Enter #4 -> Review all Room Occupation    |");
            System.out.println(" | Enter #5 -> Visitor Log and Details       |");
            System.out.println(" ----- ");
            System.out.println(" | Enter #0 -> Return to Main Menu           |");
            System.out.println(" ----- ");
            try {
                System.out.print("Choice: ");
                int userInput = input.nextInt();
                switch (userInput) {
                    case 1:
                        applications();
                        break;
                    case 2:
                        adminNotificationManagement();
                        break;
                    case 3:
                        reviewFeeSubmission();
                        break;
                    case 4:
                        currentRoomResidents();
                        break;
                    case 5:
                        visitorDetails();
                        break;
                    case 0:
                        exitProgram = true;
                        break;
                    default:
                        System.out.println("Invalid Input");
                        break;
                }
            }
        }
    }
}
```



```

        } catch (InputMismatchException e) {
            System.out.println("Invalid Input. Please enter a number.");
            input.next();
        }
    }
} else {
    System.out.println("You have already tried to login 5 times. Please try
again later.");
}
}
public static int adminLogIn(){
    int flag = 1;
    Scanner input = new Scanner(System.in);
    int count = 0;
    while (flag == 1) {
        System.out.print(" Enter the Login password: ");
        String userInput = input.nextLine();
        String password = "!@#$$%^&*()";
        count++;
        if (userInput.equals(password))
            flag = 2;
        else
            System.out.println("Invalid Password. Please Try again.");
        if (count == 5)
            flag = 3;
    }
    return flag;
}

```

```

Enter you choice: 1
Enter the Login password: !#$$%^&*()
Invalid Password. Please Try again.
Enter the Login password: !@#$$%^&*()
-----
|          ---- ADMIN PORTAL ----          |
| Enter #1 -> view all applications          |
| Enter #2 -> see all notifications          |
| Enter #3 -> Review Hotel Fee Submissions  |
| Enter #4 -> Review all Room Occupation    |
| Enter #5 -> Visitor Log and Details       |
|-----|
| Enter #0 -> Return to Main Menu           |
|-----|
Choice: |

```

View Application Code:

Below code prints all applications.

```
public static void applications(){
    int count = 1;
    try{
        File file = new File("application.txt");
        Scanner reader = new Scanner(file);
        while (reader.hasNext()) {
            System.out.println();
            System.out.println("Application "+(count++));
            System.out.println("-----");

            String[] application = reader.nextLine().split(",");
            System.out.println("Hostel ID "+application[0]);
            System.out.println("Password: "+application[1]);
            System.out.println("-----");
            System.out.println("Name           : " + application[2]);
            System.out.println("Date of birth   : " + application[3]);
            System.out.println("CNIC           : " + application[4]);
            System.out.println("Contact Number  : " + application[5]);
            System.out.println("Guardian's Name : " + application[6]);
            System.out.println("Guardian's phone no: " + application[7]);
            System.out.println("Current Room Number: " + application[8]);
            System.out.println("-----");
        }
        reader.close();
    } catch (IOException ex) {
        System.out.println("error");
    }
}
```

Viewing all applications:

```
Choice: 1

Application 1
-----
Hostel ID 534
Password: 12
-----
Name           : hm
Date of birth   : 11-11-2222
CNIC           : 11201-2222222-9
Contact Number  : 12
Guardian's Name : a
Guardian's phone no: 12
Current Room Number: 19
-----
```

Admin Notification Management

In admin notification management admin can see all the complaints or requests of the hostelite .He also can remove the notification.Admin can also view who wants to change his room and who will vacate his room in a month.

```
Choice: 2
```

```
-----  
|      --- Admin Notification Management ---      |  
| Enter #1 - View Room Shift and Vacating Requests |  
| Enter #2 - View Notifications/Complaints         |  
| Enter #3 - Remove Notification/complaints        |  
-----  
| Enter #0 - Return to Admin Portal.               |  
-----
```

```
Choice: 1
```

```
Room Shift and Vacating Requests:
```

```
Request 1: Hostelite with hostel ID 534 who lives in room No. 20 will vacate his room after one month  
Request 2: Student with hostel ID 104 who lives in room No. 20 wants to change his room to room No. 12  
Request 3: Student with hostel ID 104 who lives in room No. 20 wants to change his room to room No. 34  
Request 4: Student with hostel ID 61 who lives in room No. 18 wants to change his room to room No. 17  
Request 5: Student with hostel ID 104 who lives in room No. 20 wants to change his room to room No. 18  
Request 6: Student with hostel ID 971 who lives in room No. 18 wants to change his room to room No. 12
```

```
-----  
|      --- Admin Notification Management ---      |  
| Enter #1 - View Room Shift and Vacating Requests |  
| Enter #2 - View Notifications/Complaints         |  
| Enter #3 - Remove Notification/complaints        |  
-----
```

```
-----  
|      --- Admin Notification Management ---      |  
| Enter #1 - View Room Shift and Vacating Requests |  
| Enter #2 - View Notifications/Complaints         |  
| Enter #3 - Remove Notification/complaints        |  
-----  
| Enter #0 - Return to Admin Portal.               |  
-----
```

```
Choice: 2
```

```
There are no notifications at the moment.
```

```
-----  
|      --- Admin Notification Management ---      |  
| Enter #1 - View Room Shift and Vacating Requests |  
| Enter #2 - View Notifications/Complaints         |  
| Enter #3 - Remove Notification/complaints        |  
-----  
| Enter #0 - Return to Admin Portal.               |  
-----
```

```
Choice: 0
```

```
Exiting admin notification management
```

Admin Fee Review Portal:

In fee review portal, Admin can see all the applicants fee status and also can change the fee status too.

```
public static void reviewFeeSubmission() {
    Scanner input = new Scanner(System.in);
    boolean exit = false;
    while (!exit) {
        System.out.println("-----");
        System.out.println("|      ---- ADMIN FEE REVIEW PORTAL ----      |");
        System.out.println("| Enter #1 -> View All Fee Submissions          |");
        System.out.println("| Enter #2 -> Change Fee Status of a Student    |");
        System.out.println("-----");
        System.out.println("| Enter #0 -> Return to Admin Menu              |");
        System.out.println("-----");
        System.out.print("Choice: ");
        int choice = input.nextInt();
        switch (choice) {
            case 1:
                viewAllFeeSubmissions();
                break;
            case 2:
                changeFeeStatus();
                break;
            case 0:
                exit = true;
                break;
            default:
                System.out.println("Invalid Input. Please enter a number.");
                break;
        }
    }
}

public static void changeFeeStatus() {
    try {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter HostelID: ");
        String hostelID = input.nextLine();

        String newStatus = "";
        boolean validStatus = false;
        while (!validStatus) {
            System.out.print("Enter new FeeStatus (paid/unpaid): ");
            newStatus = input.nextLine().toLowerCase();
        }
    }
}
```

```

        if (newStatus.equals("paid") || newStatus.equals("unpaid")) {
            validStatus = true;
        } else {
            System.out.println("Invalid status. Please enter 'paid' or 'unpaid'.");
        }
    }

    File file = new File("fee.txt");
    String[] lines;

    // Read the file and count the number of lines
    int numOfLines = 0;
    try (Scanner reader = new Scanner(file)) {
        while (reader.hasNextLine()) {
            reader.nextLine();
            numOfLines++;
        }
    }

    lines = new String[numOfLines];

    // Read the file again and update the lines
    boolean found = false;
    try (Scanner reader = new Scanner(file)) {
        int index = 0;
        while (reader.hasNextLine()) {
            String line = reader.nextLine();
            String[] parts = line.split(",");
            if (parts[0].equals(hostelID)) {
                line = hostelID + "," + newStatus;
                found = true;
            }
            lines[index++] = line;
        }
    }

    // Write the modified lines back to the file
    if (found) {
        try (PrintWriter writer = new PrintWriter(new FileWriter(file))) {
            for (String line : lines) {
                writer.println(line);
            }
            System.out.println("Fee status updated successfully.");
        }
    } else {

```

```

        System.out.println("Hostel ID not found. Fee status not updated.");
    }
} catch (IOException e) {
    System.out.println("An error occurred while updating the fee file: " +
e.getMessage());
}
}

```

```

-----
|      ---- ADMIN FEE REVIEW PORTAL ----      |
| Enter #1 -> View All Fee Submissions          |
| Enter #2 -> Change Fee Status of a Student    |
|-----|
| Enter #0 -> Return to Admin Menu              |
|-----|
Choice: 1
HostelID, FeeStatus
-----
534,paid
104,unpaid
785,unpaid
21,unpaid
61,paid
971,unpaid
-----
|      ---- ADMIN FEE REVIEW PORTAL ----      |
| Enter #1 -> View All Fee Submissions          |
| Enter #2 -> Change Fee Status of a Student    |
|-----|
| Enter #0 -> Return to Admin Menu              |
|-----|

```

```

Choice: 2
Enter HostelID: 104
Enter new FeeStatus (paid/unpaid): paid
Fee status updated successfully.
-----
|      ---- ADMIN FEE REVIEW PORTAL ----      |
| Enter #1 -> View All Fee Submissions          |
| Enter #2 -> Change Fee Status of a Student    |
|-----|
| Enter #0 -> Return to Admin Menu              |
|-----|
Choice: 0
-----
|      ---- ADMIN PORTAL ----      |
| Enter #1 -> view all applications            |
| Enter #2 -> see all notifications            |
| Enter #3 -> Review Hotel Fee Submissions      |
| Enter #4 -> Review all Room Occupation        |
| Enter #5 -> Visitor Log and Details          |
|-----|
| Enter #0 -> Return to Main Menu              |
|-----|

```

Room Occupation

In room occupation function admin can see all the rooms vacancy. We have two beds in each room. Here is the code for it.

```

File file = new File("roomOccupancy.txt");
try {
    Scanner reader = new Scanner(file);
    while (reader.hasNextLine()) {
        String[] roomShiftDetails = reader.nextLine().split(",");
        System.out.println("ROOM NUMBER " + roomShiftDetails[0] + " : " +
roomShiftDetails[1] + " , " + roomShiftDetails[2] );
        count++;
    }
    reader.close();
} catch (IOException ex){
    System.out.println("Error in reading fee submission file" +
ex.getMessage());
}

```



```
Choice: 4
ROOM NUMBER 1 : Vacant , Vacant
ROOM NUMBER 2 : Vacant , Vacant
ROOM NUMBER 3 : Vacant , Vacant
ROOM NUMBER 4 : Vacant , Vacant
ROOM NUMBER 5 : Vacant , Vacant
ROOM NUMBER 6 : Vacant , Vacant
ROOM NUMBER 7 : Vacant , Vacant
ROOM NUMBER 8 : Vacant , Vacant
ROOM NUMBER 9 : Vacant , Vacants
ROOM NUMBER 10 : Vacant , Vacant
ROOM NUMBER 11 : Vacant , Vacant
ROOM NUMBER 12 : Vacant , Vacant
ROOM NUMBER 13 : Vacant , Vacant
ROOM NUMBER 14 : Vacant , Vacant
ROOM NUMBER 15 : Vacant , Vacant
ROOM NUMBER 16 : Vacant , Vacant
ROOM NUMBER 17 : Vacant , Vacant
ROOM NUMBER 18 : 61 , 971
ROOM NUMBER 19 : 785 , 534
ROOM NUMBER 20 : 21 , 104
```

Visitor Log

In visitor log admin can see the previous visitors and also can log a visitor by his name, date of visit in correct format, his phone number and his visiting id number. Here is the code for logging a visitor.

```
if (choice == 1) {
    try (Scanner reader = new Scanner(file)) {
        while (reader.hasNext()) {
            String[] visitorDetailsList = reader.nextLine().split(",");
            System.out.println("-----");
            System.out.println("Visitor Number " + count);
            System.out.println("-----");
            System.out.println("Name : " + visitorDetailsList[0]);
            System.out.println("DateVisit : " + visitorDetailsList[1]);
            System.out.println("Phonenum: " + visitorDetailsList[2]);
            System.out.println("Visiting ID number : " +
visitorDetailsList[3]);

            System.out.println();
            count++;
        }
        System.out.println("Total Number of Visitors: " + (count - 1));
    } catch (IOException ex) {
        System.out.println("Error reading file: " + ex.getMessage());
    }
}
```



```

    } else if (choice == 2) {
        System.out.println();

        System.out.print("Enter Visitors Full Name: ");
        String name1 = input.nextLine();
        while (!isValidName(name1)) {
            System.out.print("Please Enter a Valid Full Name: ");
            name1 = input.nextLine();
        }

        System.out.print("Enter the date of your visit in DD-MM-YYYY
format: ");
        String bDate = input.nextLine();
        while (!isValidNumericInput(bDate) || bDate.length() != 10 ||
bDate.charAt(2) != '-' || bDate.charAt(5) != '-' || bDate.charAt(0) > '3' ||
bDate.charAt(1) > '9' || bDate.charAt(3) > '1' || (bDate.charAt(3) == '1' &&
bDate.charAt(4) > '2')) {
            System.out.println("Invalid input. Please enter date in DD-MM-
YYYY format: ");
            bDate = input.nextLine();
        }

        System.out.print("Contact Number (digits only): ");
        String phoneNumber = input.nextLine();
        while (!isValidNumericInput(phoneNumber)) {
            System.out.print("Invalid input. Please enter contact number
(digits only): ");
            phoneNumber = input.nextLine();
        }

        System.out.print("Visiting Student ID number: ");
        String idNumber = input.nextLine();
        while (!isValidNumericInput(idNumber)) {
            System.out.print("Invalid input. Please enter visiting ID
number (digits only): ");
            idNumber = input.nextLine();
        }

        try (FileWriter fileWriter = new FileWriter(file, true);
            PrintWriter writer = new PrintWriter(fileWriter)) {
            writer.println(name1 + "," + bDate + "," + phoneNumber + "," +
idNumber);

            System.out.println("Visitor logged successfully.");
        } catch (IOException ex) {

```

```

        System.out.println("Error writing to file: " +
ex.getMessage());
    }

    } else if (choice == 0) {
        flag = true;
        break;

    } else {
        System.out.println("Enter a Valid Number.");
    }
}
} catch (Exception ex) {
    System.out.println("Error: Enter a valid number." );
}
}

```

```

-----
|          ---- Visitor Log ----          |
-----
| Enter #1 -  View Previous Visitors      |
| Enter #2 -  Log Visitor                  |
| Enter #0 -  Return to Admin Portal      |
-----

```

Choice: 1

Visitor Number 1

```

Name           : ahmad
Date of Visit  : 11-01-2222
Phone number   : 12
Visiting ID number : 12
-----

```

Total Number of Visitors: 1

```

|          ---- Visitor Log ----          |
-----
| Enter #1 -  View Previous Visitors      |

```

```

-----
|          ---- Visitor Log ----          |
-----
| Enter #1 -  View Previous Visitors      |
| Enter #2 -   Log Visitor                 |
| Enter #0 -  Return to Admin Portal      |
-----
Choice: 2

Enter Visitors Full Name: ahmad
Enter the date of your visit in DD-MM-YYYY format: 12-01-2000
Contact Number (digits only): 1234
Visiting Student ID number: 12
Visitor logged successfully.

```

Hostelite Portal

In hostelite portal hostelite logs in by his hostel id and his password which is saved in application file. Hostelite can view his application, check his fee status, request to change or vacate his room, view hostel rules and also can file ny complaint or request.

Here is the code for fee status:

```

try {
    File file = new File("fee.txt");
    Scanner reader = new Scanner(file);
    String feeStatus = "";
    while (reader.hasNextLine()) {
        String line = reader.nextLine();
        String[] parts = line.split(",");
        if (parts[0].equals(hostelID)) {
            feeStatus = parts[1];
            break;
        }
    }
    reader.close();

    if (!feeStatus.isEmpty()) {
        System.out.println("Your fee status: " + feeStatus);
    } else {
        System.out.println("No fee record found for your hostel ID.");
    }
} catch (IOException e) {

```

```
        System.out.println("An error occurred while reading the fee file: " +
e.getMessage());
}
```

```
Enter you choice: 2
Enter your hostel ID: 104
Enter your password: 123
LoggedIn successfully
```

```
-----
|          ---- HOSTELITE PORTAL ----          |
| Enter #1 -> View Application                    |
| Enter #2 -> Fee Submission Status              |
| Enter #3 -> Request to change or vacant Room  |
| Enter #4 -> Hostel Rules and GuideLines       |
| Enter #5 -> Complaints/ Requests              |
|-----|
| Enter #0 -> Return to Main Menu                |
|-----|
Choice: 1
```

Viewing Application

```
Choice: 1
-----
Hostel ID       : 104
Password       : 123
Name           : a
Date of birth  : 12-01-2222
CNIC           : 11201-2222222-3
Contact Number : 1
Guardian's Name : w
Guardian's phone number: 2
Room Number    : 2
```

Fee Status

```
Choice: 2
Your fee status: paid
```

Request for changing or vacating room

```
Choice: 3
-----
| Enter #1 - Shift Room |
| Enter #2 - Vacant Room |
-----
NOTE: Please inform 1 month prior to vacate the room
-----
Choice: 1
Which room you are currently in:
2
Invalid current room number. Please try again.
Which room you are currently in:
20
Which room you want to change to:
12
Room change request submitted.
```

Complaint/Requests

```
Choice: 5
Enter your message: Door handle not working
Notification submitted successfully.
```

[SignUp](#)

Creating An Application:

Hostellite can create his application by entering some details in the correct format. If the details are not in correct format, he must put it in correct format. Here is the code for correct validation.

```
public static boolean isValidName(String str) {
    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        if (!Character.isLetter(c) && c != ' ') {
            return false;
        }
    }
}
```

```

    }
    return true;
}
public static boolean isValidNumericInput(String str) {
    for (int i =0; i<str.length(); i++) {
        char c = str.charAt(i);
        if (!Character.isDigit(c) && c!='-' ) {
            return false;
        }
    }
    return true;
}
}

```

```

|-----|
|          ---- SIGN UP ----          |
|  Enter #3 -> create a new application |
|-----|
|  Enter #4 -> View Hostel Rules        |
|  Enter #0 -> Exit Program              |
|-----|

Enter you choice: 3

-----

To create a new application please enter the following information.
-----

Please enter your full name: ahmad
Enter your date of birth in DD-MM-YYYY format: 10-11-2000
Enter your CNIC number in DDDDD-DDDDDDD-D format: 11201-9383828-9
Contact Number (digits only): 1234
Guardian's Name: falak
Guardian's Contact Number (digits only): 123

-----

Your Hostel ID is: 497
NOTE: Don't forget your hostel ID. Kindly note it down somewhere.
-----

Create a new password to log in: 123
-----

```

Choosing room no. after creating application

Room Number 1

Left Bed: Available.

Right Bed: Available.

Room Number 2

Left Bed: Available.

Right Bed: Available.

Room Number 3

Left Bed: Available.

Right Bed: Available.

Room Number 4

Left Bed: Available.

Right Bed: Available.

Room Number 17

Left Bed: Available.

Right Bed: Available.

Room Number 18

Left Bed: Unavailable.

Right Bed: Unavailable.

Room Number 19

Left Bed: Unavailable.

Right Bed: Unavailable.

Room Number 20

Left Bed: Unavailable.

Right Bed: Unavailable.

Fee Invoice:

Fee invoice is generated to the user. If he enters paid it gets stored in fee file with his hostel id otherwise if he enters unpaid, a message is displayed to submit fee within 5 days.
Here is its code.

```
generateInvoice();
System.out.println();

// Prompt the user if they have paid the fee or not
String feeStatus = " ";
boolean validInput = false;
while (!validInput) {
    System.out.print("What is your fee status (Paid/Unpaid): ");
    feeStatus = input.nextLine().toLowerCase();
    if (feeStatus.equals("paid") || feeStatus.equals("unpaid")) {
        validInput = true;
    } else {
        System.out.println("Invalid input. Please enter 'paid' or 'unpaid'.");
    }
    if (feeStatus.equals("unpaid")) {
        System.out.println("Please make sure to submit your fee within 5 days to avoid penalties.");
    }
}

try {
    // Save fee payment status along with hostel ID in the fee file
    try (PrintWriter writer = new PrintWriter(new FileWriter("fee.txt", true))) {
        writer.println(hostelID + "," + feeStatus);
        System.out.println("Fee payment status saved successfully.");
    } catch (IOException e) {
        System.out.println("An error occurred while saving fee payment status: " + e.getMessage());
    }
}
```



```
Enter the room number you want to occupy: 19
Invalid or unavailable room number. Please enter a valid room number: 17
```

Hostel Fee Invoice:

```
-----
Accommodation Fee      : PKR 25000
Hostel Security Fee    : PKR 5000
Internet Fee           : PKR 2000
Damage Deposit         : PKR 3000
Lockers Fee            : PKR 1500
Breakfast Fee          : PKR 6000
Laundry Fee            : PKR 4000
Utilities Fee          : PKR 3000
Activity Fee           : PKR 2000
-----
Total                  : PKR 51500
```

```
What is your fee status (Paid/Unpaid): unpaid
Please make sure to submit your fee within 5 days to avoid penalties.
Fee payment status saved successfully.
Your account has been created
```

Hostel Rules

This function displays all the hostel rules to the hostelite and also in main menu too.

```
| Enter #4 -> View Hostel Rules |
| Enter #0 -> Exit Program      |
|-----|
Enter you choice: 4
HOSTEL RULES:
|-----|
| 1. Quiet hours: Respect designated quiet hours to ensure everyone can rest peacefully. |
| 2. No smoking: Most hostels have no-smoking policies inside the building for the comfort and safety of all guests. |
| 3. Clean up after yourself: Keep common areas tidy and clean up after using the kitchen or any shared facilities. |
| 4. Respect others' privacy: Avoid disturbing other guests and respect their personal space and belongings. |
| 5. No drugs or illegal activities: Hostels typically have strict policies against drug use and any illegal activities on the premises. |
| 7. Lockers: Use provided lockers to secure your valuables and always lock your dorm room when you leave. |
| 8. Guests: Inform the hostel staff if you plan to have guests over, and ensure they adhere to hostel rules. |
| 9. Use of facilities: Be mindful of other guests when using shared facilities like bathrooms, showers, and the kitchen. |
| 10. Respect the staff: Treat hostel staff with kindness and respect their authority in enforcing hostel rules. |
| 11. No pets: Most hostels do not allow pets, so make sure to leave your funny friends at home. |
| 12. Socialize responsibly: Be considerate of others and avoid disturbing them during late-night socializing. |
| 13. Be mindful of energy and water usage: Conserve resources by turning off lights, fans, and taps when not in use. |
|-----|
```