National University of Technology.

(NUTECH)

Name:	Muhammad Huzaifa Zaman Syed Maaz Shah Muhammad Ijlal Asif Mohammad Ali Shah Muhammad Usman
Registration No:	F24607096 F24607091 F24607072 F24607060 F24607094
Department:	Artificial Intelligence (Section-B)
Course:	Object Oriented Programming
Course Code:	CS-122
Semester:	Spring-2025
Project Name:	Parking Management System

Project Overview: The system is designed to manage vehicle entries and exits in a parking facility. It includes functionalities for admin login, recording vehicle entry details, processing vehicle exits with payment calculation, and searching for vehicle information. Data is stored in several text files (LoginInfo.txt, CarInfo.txt, ExitInfo.txt, parking_slots.txt, Payment.txt, Penalty.txt).

I. Core Modules and Class Descriptions:

1. **Admin.java**:
   - **Purpose**: Manages admin user credentials.
   - **Functionality**:
     - Stores admin usernames and passwords (up to 3 admins + 1 default) in static private variables.
     - Provides static methods (defaultAdmin, login1, login2, login3) to initialize these credentials.
     - The main method in this class serves as a one-time setup utility to populate LoginInfo.txt with predefined admin credentials, their associated shifts (X, A, B, C), and names.
   - **Data File Interaction**: Writes to LoginInfo.txt.
2. **LoginPage.java**:
   - **Purpose**: Provides the user login interface.
   - **Functionality**:
     - A Swing JFrame with fields for email (username) and password.
     - On "Login" click, it reads the entered credentials.
     - Determines the current shift (A, B, C) based on the system's current hour.
     - Reads LoginInfo.txt to validate the credentials against the stored usernames, passwords, and allowed shift. A default shift "X" (likely for a super admin) is also checked.
     - If authentication is successful, it disposes of the login window and opens AdminPanel.
   - **Data File Interaction**: Reads LoginInfo.txt.
3. **AdminPanel.java**:
   - **Purpose**: The main dashboard for administrators after successful login.
   - **Functionality**:
     - A Swing JFrame presenting three main options: "Enter Vehicle", "Exit Vehicle", and "Search".
     - Clicking these buttons navigates the user to the respective functional windows (Entry, Exit, Search).
   - **Data File Interaction**: None directly.
4. **VehicleEntry.java (contains Vehicle abstract class and subclasses Car, Bike, Jeep, Wagon, Truck; and Time interface)**:
   - **Vehicle (abstract class)**:
     - Base class for different vehicle types.
     - Attributes: DriverName, LicencePlate, ReceiptNo, SlotNo, Color, HourParked.
     - Abstract method: Type() to be implemented by subclasses. ✦ Getters for all attributes.
   - **Car, Bike, Jeep, Wagon, Truck (concrete classes)**:

      - Extend Vehicle.
      - Implement Type() to return the specific vehicle type string (e.g., "Car").
      - Implement the Time interface.
   - **Time (interface)**:
      - Defines Entry() and Exit() methods, which in the implementing vehicle classes, return the current timestamp. This seems somewhat redundant as the GUI classes (Entry.java, Exit.java) handle timestamp generation directly. ○ **VehicleEntry.java's main method**: This main method is rudimentary and seems like a placeholder or an incomplete attempt to directly write to CarInfo.txt. It does not align with the main application flow where Entry.java handles writing to CarInfo.txt. ○ **Data File Interaction**: The main method attempts to write to CarInfo.txt but is not used in the primary application flow.

5. **Entry.java**:
   - **Purpose**: Handles the process of a new vehicle entering the parking lot. ○ **Functionality**:
      - A Swing JFrame with input fields for: Driver Name, License Plate, Receipt No., Slot No., Color, Hours Parked (pre-defined duration).
      - A JComboBox for Vehicle Type (Car, Bike, Jeep, Wagon, Truck).
      - A JRadioButton for Handicap status.
      - An auto-populated, non-editable field for Entry Time.
      - **Slot Management**:
      - Checks parking_slots.txt for slot availability and type (e.g., General, Handicap) using checkSlotAvailability() and getSlotType().
      - Prevents entry if the slot is "Taken" or "INVALID".
      - Warns if a non-handicap user selects a handicap slot or viceversa, allowing override.
      - Checks CarInfo.txt via isSlotAlreadyUsed() to prevent re-using a slot number that has previous entry data (even if marked available in parking_slots.txt - this is a specific business rule).
      - Updates parking_slots.txt to mark the slot as "Taken" using updateSlotStatus().
      - **Admin Identification**: Determines the current admin's name based on the current time shift (A, B, C) by reading LoginInfo.txt. This logic seems to assign fixed names (Huzaifa, Maaz, Ijlal) based on shift rather than using the currently logged-in admin.
      - **Payment Calculation**:
      - Calculates totalPayment based on hoursParked (input by user) and a base rate retrieved via getBaseRateForVehicle().
      - Handicap vehicles have a fixed rate (Rs. 20 per hour).
      - **Data Recording**: Writes detailed vehicle entry information, including calculated payment, to CarInfo.txt.
      - Clears fields after successful entry.
   - **Data File Interaction**: Reads parking_slots.txt, LoginInfo.txt. Writes/Appends to CarInfo.txt. Modifies parking_slots.txt.
6. **Exit.java**:
   - **Purpose**: Manages the process of a vehicle exiting the parking lot and payment.

- **Functionality**:
    - ✦ A Swing JFrame with fields for Receipt Number and Payment Amount.
    - ✦ A JCheckBox for "Paid" confirmation.
    - ✦ **checkButtonClicked**:
    - ✦ Retrieves vehicle details from CarInfo.txt based on the entered Receipt Number.
    - ✦ Checks ExitInfo.txt to see if the vehicle has already exited using isVehicleAlreadyExited().
    - ✦ If found and not exited, it calculates currentPayment and currentHoursParked dynamically using methods from the Payment class (Payment.getHoursParked(), Payment.calculateVehiclePayment()), which use the recorded entry time and current time.
    - ✦ Displays vehicle details and payment due.
    - ✦ **updateButtonClicked**:
    - ✦ Validates the entered payment against the currentPayment.
    - ✦ Handles insufficient payment, exact payment, and overpayment (calculating change).
    - ✦ Requires the "Paid" checkbox to be selected.
    - ✦ Updates the corresponding slot in parking_slots.txt to "Available" using updateSlotStatus().
    - ✦ Records detailed exit information (including exit time, actual hours parked, payment details) into ExitInfo.txt.
    - ✦ Resets input fields.
    - ✦ **Internal PaymentCalculator class**: Contains logic to calculate payment and hours parked. This seems to be a fallback or an alternative implementation to the main PaymentCalculator in Payment.java. It can load rates from Payment.txt or use defaults.
- **Data File Interaction**: Reads CarInfo.txt, parking_slots.txt, Payment.txt (by internal PaymentCalculator). Writes/Appends to ExitInfo.txt. Modifies parking_slots.txt.

7. **Search.java**:
    - **Purpose**: Allows searching for vehicle records.
    - **Functionality**:
        - ✦ A Swing JFrame with a field to enter Receipt Number.
        - ✦ On "Search" click, it searches for the given receipt number in:
        - ✦ CarInfo.txt (for vehicles currently parked) using searchInCarInfo().
        - ✦ ExitInfo.txt (for vehicles that have already exited) using searchInExitInfo().
        - ✦ Displays the found information in a JTextPane. If not found, indicates so.
    - **Data File Interaction**: Reads CarInfo.txt and ExitInfo.txt.
8. **Payment.java (contains Penalty interface, PaymentMethod class, PaymentCalculator class)**:
    - **Penalty (interface)**: Defines methods for getting penalties for different vehicle types.

- **PaymentMethod (class)**: Implements Penalty. Can hold a baseAmount. The penalty methods calculate penalties as a percentage of an input amount or a fixed amount for handicap.
- **PaymentCalculator (class)**:
  - ✦ The primary logic for calculating parking fees.
  - ✦ Loads base vehicle rates from Payment.txt (format: VehicleType Rate).
  - ✦ Loads penalty rates from Penalty.txt (format: VehicleType PenaltyRate).
  - ✦ calculateHoursParked(): Calculates hours between entry and exit times.
  - ✦ calculatePayment(): Calculates payment based on vehicle type, hours parked (rounded up to the nearest hour), and an optional penalty.

o **Payment.java's main method**: A utility to create/overwrite Payment.txt and Penalty.txt with predefined rates and penalties. o Static helper methods calculateVehiclePayment() and getHoursParked() provide an interface for other classes (like Exit.java) to use the PaymentCalculator. o **Data File Interaction**: Writes to Payment.txt and Penalty.txt (via its main method). Reads Payment.txt and Penalty.txt (via PaymentCalculator).

II. Data Files Used: • **LoginInfo.txt**: Stores admin credentials.

- Format: email password shift admin_name (e.g., admin@gmail.com 1234 X Admin)

- **parking_slots.txt**: Manages the status of parking slots.
  - Format: Slot <slot_id> <SlotType> <Status> (e.g., Slot 1 General Available, Slot 10 Handicap Taken)
- **CarInfo.txt**: Logs details of vehicles currently parked.
  - Format: Multi-line entry per vehicle, including driver name, license plate, receipt no, slot no, color, hours parked (pre-defined), vehicle type, entry time, admin name, and total payment (calculated at entry).
- **ExitInfo.txt**: Logs details of vehicles that have exited.
  - Format: Multi-line entry per vehicle, including receipt no, driver name, license plate, slot no (marked available), vehicle type, entry time, exit time, actual hours parked, required payment, amount received, change, and payment status.
- **Payment.txt**: Stores base parking rates for different vehicle types. o Format: VehicleType Rate (e.g., Car 70.0)
- **Penalty.txt**: Stores penalty rates for different vehicle types.
  - Format: VehicleType PenaltyRate (e.g., Car 35.0)

III. Key Functionalities and Workflow:

1. **Setup (Manual/One-time)**:

    - Run Admin.java's main method to create LoginInfo.txt. ○ Run Payment.java's main method to create Payment.txt and Penalty.txt.
    - Manually create parking_slots.txt with initial slot configurations.
2. **Application Start**:
    - The application starts with LoginPage (any of the GUI classes' main methods can initiate it, but typically LoginPage.main() would be the entry point).
3. **Login**:
    - Admin enters credentials. ○ LoginPage validates against LoginInfo.txt and current shift.
    - On success, AdminPanel is displayed.
4. **Vehicle Entry (Entry.java)**:
    - Admin selects "Enter Vehicle" from AdminPanel.
    - Fills in vehicle and driver details, pre-defined parking duration, and selects slot.
    - System validates slot, calculates initial payment based on pre-defined hours.
    - CarInfo.txt is updated with entry details.
    - parking_slots.txt is updated (slot becomes "Taken").
5. **Vehicle Exit (Exit.java)**:
    - Admin selects "Exit Vehicle" from AdminPanel. ○ Enters Receipt Number. ○ System fetches details from CarInfo.txt. ○ Calculates actual hours parked (current time - entry time) and final payment due using PaymentCalculator.
    - Admin enters payment received and confirms. ○ ExitInfo.txt is updated with exit details.
    - parking_slots.txt is updated (slot becomes "Available").
6. **Search Vehicle (Search.java)**:
    - Admin selects "Search" from AdminPanel.
    - Enters Receipt Number. ○ System searches CarInfo.txt (parked) and ExitInfo.txt (exited) and displays results.

IV. Strengths:

- **GUI Interface**: Provides a user-friendly way to interact with the system using Java Swing.
- **Modular Design**: Different functionalities are separated into different classes (e.g., Login, Entry, Exit, Payment).
- **Basic Functionality Covered**: Implements core features expected in a parking management system.
- **Shift Management**: Attempts to incorporate shift-based login.
- **Slot Management**: Includes features for checking slot availability and type.
- **Payment Calculation**: Dynamic payment calculation at exit based on actual duration.

V. Areas for Improvement and Concerns:

1. **Data Persistence**:

- **Flat Files**: Using text files is simple but not robust, scalable, or efficient for data querying, updates, and ensuring data integrity. Prone to corruption. ○ **String Parsing**: Relying heavily on string parsing (e.g., "Receipt No: ...") to extract data from files is fragile and error-prone if the format changes slightly.
- **No Database**: A relational database (like SQLite, MySQL, PostgreSQL) would be far more suitable.

2. **Security**:
    - **Plain Text Passwords**: Storing passwords in plain text in LoginInfo.txt is a major security vulnerability. Passwords should be hashed. ○ **No Input Sanitization**: Susceptible to issues if unexpected characters are entered, especially if file paths or commands were constructed from user input (though not apparent here, it's a general concern with direct file I/O).
3. **Code Duplication**:
    - The PaymentCalculator logic is present as a main class in Payment.java and also as an inner class within Exit.java. This should be consolidated. ○ Placeholder behavior logic (addPlaceholderBehavior, setupPlaceholder) is repeated in Entry.java, Exit.java, and Search.java. A utility class could handle this.
    - File I/O operations (reading/writing lines, updating slot status) are similar in Entry.java and Exit.java.
4. **Error Handling**:
    - Error handling is basic (e.g., JOptionPane for messages, System.err.println for file errors). More comprehensive error logging and recovery mechanisms could be implemented. ○ File not found errors are sometimes handled by printing to console, sometimes by showing a dialog. Consistency would be better.
5. **Design and Best Practices**:
    - **Static Misuse**: Admin.java uses static fields and methods extensively for what should ideally be instance-specific data if multiple admin profiles were managed more dynamically.
    - **Admin Identification in Entry.java**: The admin name logged in CarInfo.txt is hardcoded based on shift ("Huzaifa", "Maaz", "Ijlal") rather than identifying the actual logged-in user. The logged-in user's identity should be passed through the system.
    - **VehicleEntry.java main method**: The main method in VehicleEntry.java is confusing and doesn't fit the application's operational flow. ○ **Time interface in Vehicle subclasses**: The Entry() and Exit() methods in Car, Bike etc., which return current time, are not actually used by Entry.java or Exit.java for recording entry/exit times. These GUI classes generate timestamps themselves.
    - **Hardcoded Values**: Shift names, some rates (e.g., handicap in Entry.java before deferring to PaymentCalculator), and admin names are hardcoded.
    - **isSlotAlreadyUsed in Entry.java**: This method checks CarInfo.txt to see if a slot number has ever been used. This implies a slot, once used, can never be re-assigned even if the vehicle exits. This might be a specific business rule, but it means parking_slots.txt isn't the sole source of truth for availability for *new* entries.

   - **Concurrency**: Not designed for concurrent use. File access is not synchronized, which could lead to data corruption if multiple instances or threads tried to access files simultaneously.
6. **User Experience (UX)**:
   - Pre-defining "Hours Parked" at entry is unusual for a typical pay-on-exit system. Usually, time is calculated at exit. The system does calculate payment at exit based on actual duration, so the entry-time "Hours Parked" might be for estimation or a different parking model.
   - GUI layout is functional but could be improved with more modern look and feel or better component arrangement.
7. **Maintainability & Scalability**:
   - Reliance on specific text file formats makes changes difficult. o Adding new features or modifying existing ones would be cumbersome due to tight coupling with file structures and string parsing.

VI. Specific Recommendations:

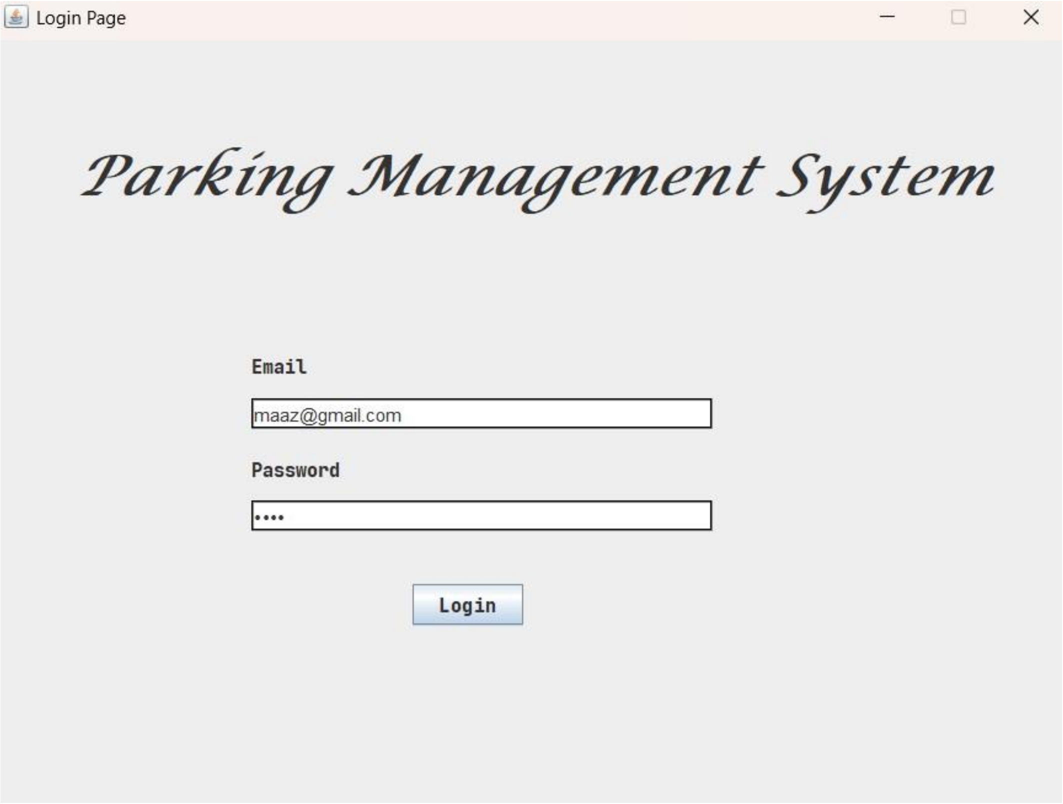
1. **Adopt a Database**: Migrate data storage to a relational database (e.g., SQLite for a simple desktop app). This would resolve many issues related to data integrity, querying, and parsing.
2. **Password Hashing**: Implement password hashing (e.g., using bcrypt or Argon2) instead of storing plain text passwords.
3. **Centralize Logic**:
   - Create a single, robust PaymentCalculator and use it consistently. o Develop utility classes for common tasks like placeholder text, file I/O, and date/time formatting.
4. **Improve Admin Tracking**: Pass the logged-in admin's identity (username or ID) from LoginPage through to AdminPanel and then to Entry/Exit panels to correctly log who performed actions.
5. **Refactor Vehicle hierarchy**: Re-evaluate the Time interface and its implementation in vehicle subclasses, as it seems unused.
6. **Configuration Files**: Instead of hardcoding rates or admin names directly in the code (like in Entry.java's admin name logic), consider using configuration files or deriving more information from the database.
7. **Refine Slot Logic**: Clarify the business rule for isSlotAlreadyUsed. If a slot is "Available" in parking_slots.txt, it should generally be usable.
8. **Object-Oriented Data Handling**: Instead of parsing strings from files repeatedly, read data into objects (e.g., a VehicleLog object) and operate on these objects.

VII. Conclusion:

The Parking Management System is a functional Java Swing application that covers the basic requirements of vehicle entry, exit, and search. It demonstrates an understanding of GUI

development and basic file I/O. However, it suffers from common pitfalls of using flat files for complex data storage, security oversights, and areas of code duplication.
Refactoring to use a database, implementing proper security practices, and centralizing business logic would significantly enhance the robustness, maintainability, and scalability of this application.

GUI SCREENSHOTS

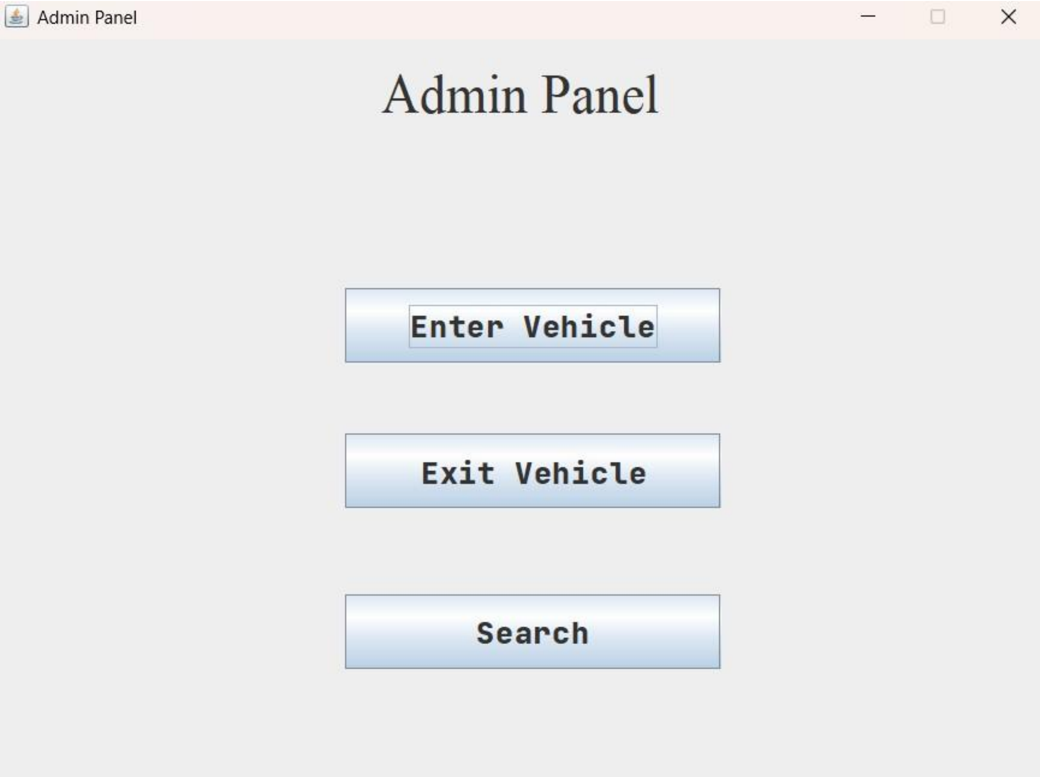


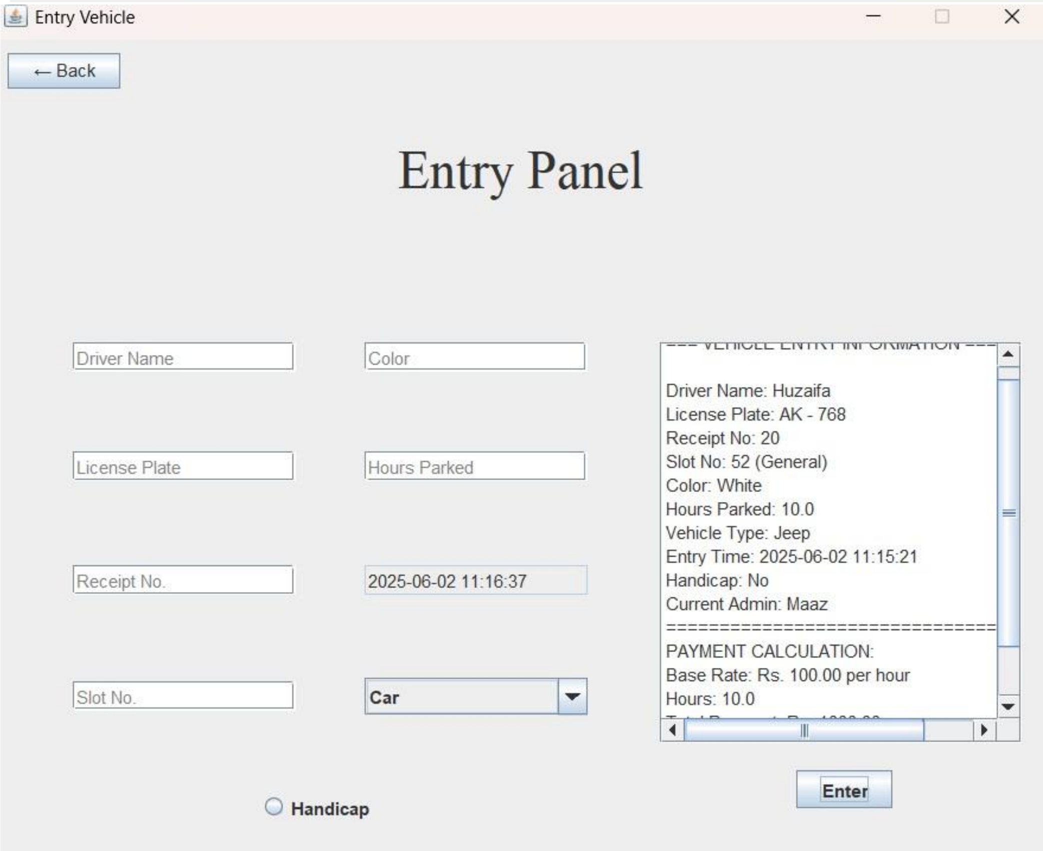
Email

maaz@gmail.com

Password

••••

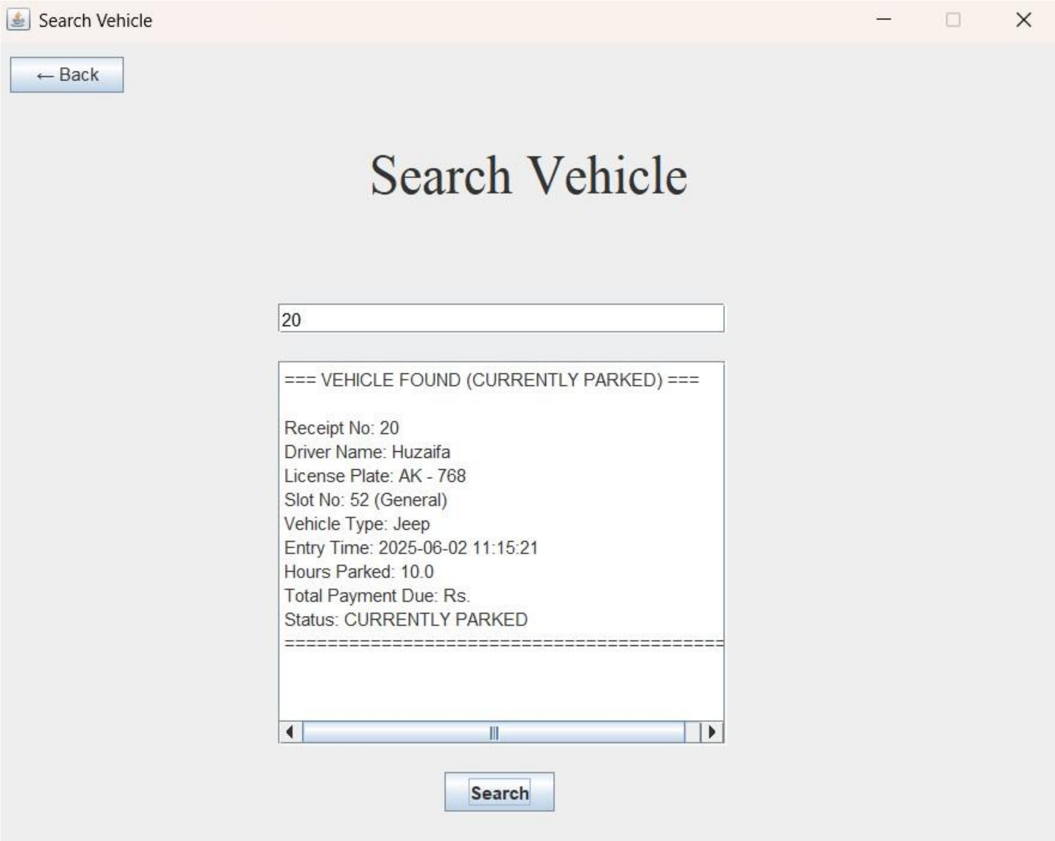




2025-06-02 11:16:37

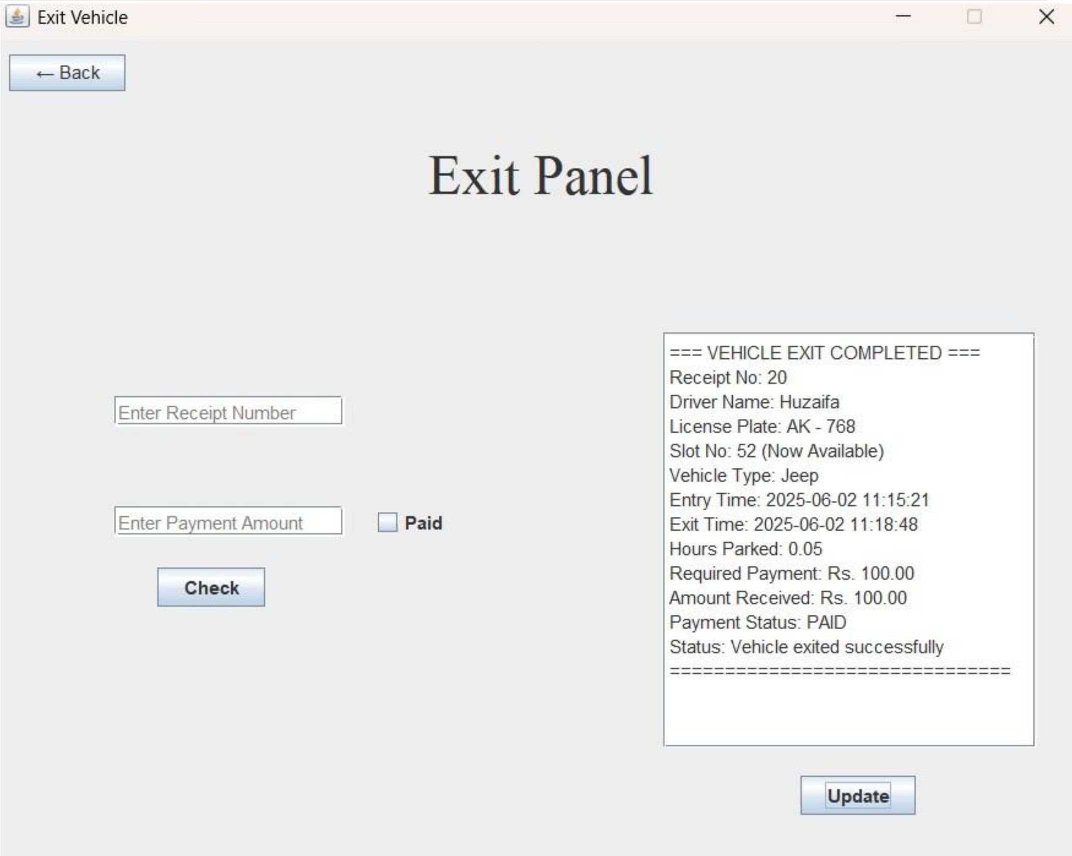
Car

```
=== VEHICLE ENTRY INFORMATION ===

Driver Name: Huzaifa
License Plate: AK - 768
Receipt No: 20
Slot No: 52 (General)
Color: White
Hours Parked: 10.0
Vehicle Type: Jeep
Entry Time: 2025-06-02 11:15:21
Handicap: No
Current Admin: Maaz
================================
PAYMENT CALCULATION:
Base Rate: Rs. 100.00 per hour
Hours: 10.0
Total Payment: Rs. 1000.00
```



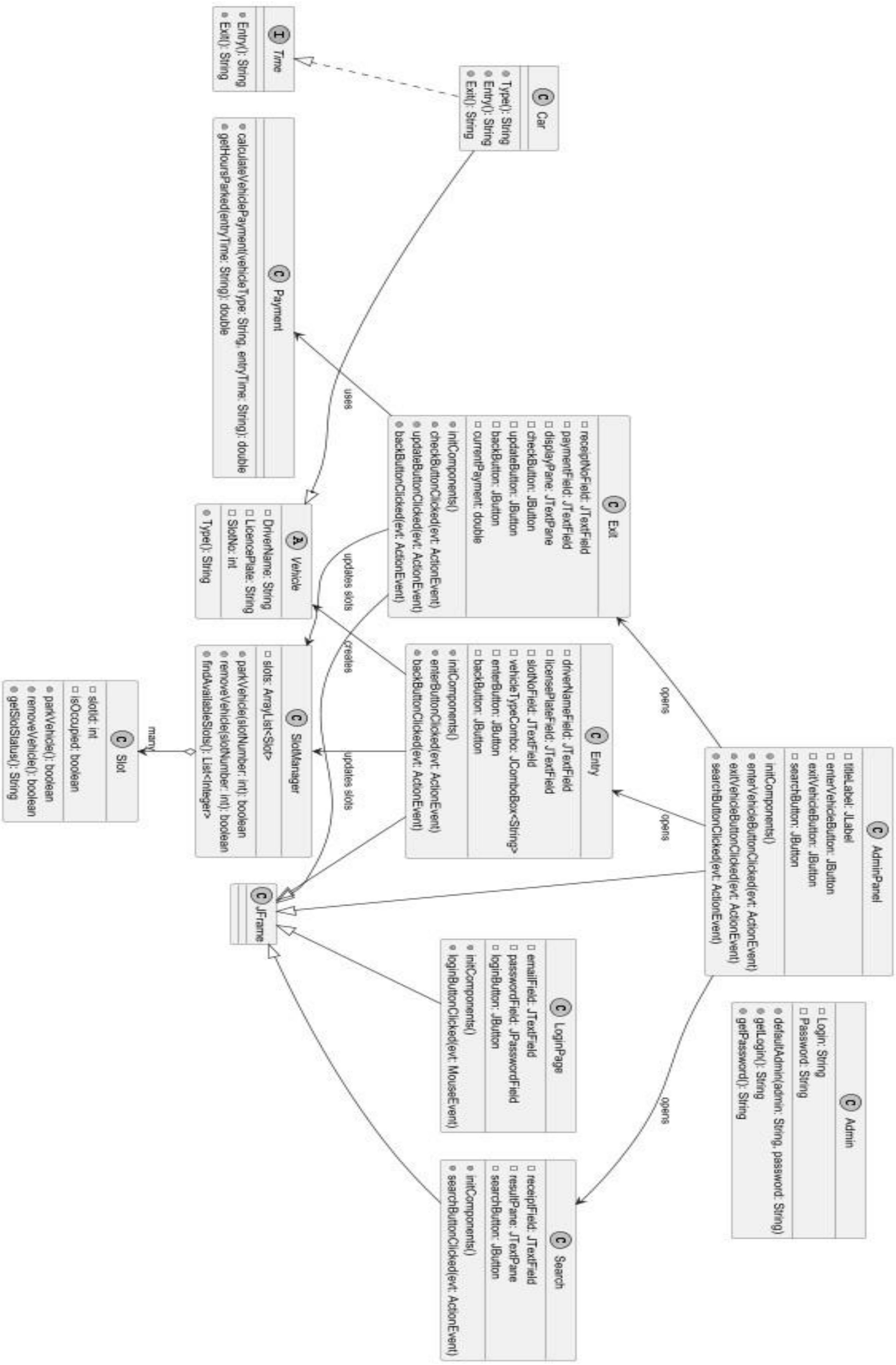
20

```
=== VEHICLE FOUND (CURRENTLY PARKED) ===

Receipt No: 20
Driver Name: Huzaifa
License Plate: AK - 768
Slot No: 52 (General)
Vehicle Type: Jeep
Entry Time: 2025-06-02 11:15:21
Hours Parked: 10.0
Total Payment Due: Rs.
Status: CURRENTLY PARKED
========================================
```



```
=== VEHICLE EXIT COMPLETED ===
Receipt No: 20
Driver Name: Huzaifa
License Plate: AK - 768
Slot No: 52 (Now Available)
Vehicle Type: Jeep
Entry Time: 2025-06-02 11:15:21
Exit Time: 2025-06-02 11:18:48
Hours Parked: 0.05
Required Payment: Rs. 100.00
Amount Received: Rs. 100.00
Payment Status: PAID
Status: Vehicle exited successfully
==============================
```

UML DIAGRAM:



I Time
Entry(): String
Exit(): String
C Car
Type(): String
Entry(): String
Exit(): String
C Payment
calculateVehiclePayment(vehicleType: String, entryTime: String): double
getHoursParked(entryTime: String): double
A Vehicle
DriverName: String
LicencePlate: String
SlotNo: int
Type(): String
C SlotManager
slots: ArrayList<Slot>
parkVehicle(slotNumber: int): boolean
removeVehicle(slotNumber: int): boolean
findAvailableSlots(): List<Integer>
C Slot
slotId: int
isOccupied: boolean
parkVehicle(): boolean
removeVehicle(): boolean
getSlotStatus(): String
many
C Exit
receiptNoField: JTextField
paymentField: JTextField
displayPane: JTextPane
checkButton: JButton
updateButton: JButton
backButton: JButton
currentPayment: double
initComponents()
checkButtonClicked(evt: ActionEvent)
updateButtonClicked(evt: ActionEvent)
backButtonClicked(evt: ActionEvent)
uses
updates slots
creates
C Entry
driverNameField: JTextField
licensePlateField: JTextField
slotNoField: JTextField
vehicleTypeCombo: JComboBox<String>
enterButton: JButton
backButton: JButton
initComponents()
enterButtonClicked(evt: ActionEvent)
backButtonClicked(evt: ActionEvent)
updates slots
C JFrame
C AdminPanel
titleLabel: JLabel
enterVehicleButton: JButton
exitVehicleButton: JButton
searchButton: JButton
initComponents()
enterVehicleButtonClicked(evt: ActionEvent)
exitVehicleButtonClicked(evt: ActionEvent)
searchButtonClicked(evt: ActionEvent)
opens
opens
opens
C LoginPage
emailField: JTextField
passwordField: JPasswordField
loginButton: JButton
initComponents()
loginButtonClicked(evt: MouseEvent)
C Admin
Login: String
Password: String
defaultAdmin(admin: String, password: String)
getLogin(): String
getPassword(): String
C Search
receiptField: JTextField
resultPane: JTextPane
searchButton: JButton
initComponents()
searchButtonClicked(evt: ActionEvent)

README.txt:

Parking Management System User Instructions

Overview

This Java-based Parking Management System allows administrators to manage vehicle entries, exits, payments, and searches in a parking facility using a Swing GUI. This guide explains how to operate the system.

Getting Started

1. **Run the Application**:
   - Double-click the executable JAR file (if provided) or run the program via your IDE (e.g., IntelliJ IDEA) by executing the `LoginPage` class.
   - The login window will appear.
2. **Login to the System**:
   - Enter the email and password based on the current time (24-hour format):
     - **Default Admin** (works any time):
     - Email: `admin@gmail.com`
     - Password: `1234` ✦ **Shift A** (00:00–07:59):
     - Email: `huzaifa@gmail.com`
     - Password: `7096` ✦ **Shift B** (08:00–15:59):
     - Email: `maaz@gmail.com`
     - Password: `7091` ✦ **Shift C** (16:00–23:59):
     - Email: `ijlal@gmail.com`
     - Password: `7072`
   - Click `Login`. If credentials are valid and match the shift, the Admin Panel opens. Otherwise, an error message appears.

Using the System

1. Admin Panel

The Admin Panel is the main interface with three options:

- **Enter Vehicle**: Opens the vehicle entry form.
- **Exit Vehicle**: Opens the vehicle exit and payment form.
- **Search**: Opens the vehicle search form.

2. Entering a Vehicle

1. From the Admin Panel, click `Enter Vehicle`.
2. Fill in the fields:
   - **Driver Name**: Enter the driver's name (e.g., John Doe).

- **License Plate**: Enter the vehicle’s license plate (e.g., ABC-123).
- **Receipt No.**: Enter a unique receipt number (e.g., R001). o **Slot No.**: Enter a slot number (1–100). Slots 1–10 are for handicapped vehicles.
- **Color**: Enter the vehicle’s color (e.g., Blue).
- **Hours Parked**: Enter estimated parking hours (e.g., 2.5).
- **Vehicle Type**: Select from the dropdown (Car, Bike, Jeep, Wagon, Truck). o **Handicap**: Check if the vehicle requires a handicapped slot. o **Entry Time**: Automatically set to the current time. 3. Click `Enter`:
- If all fields are valid and the slot is available, the entry is recorded, and the slot is marked as occupied. o A confirmation message shows the total payment based on vehicle type and hours.
- If the slot is taken or invalid, an error message appears.

4. Click `← Back` to return to the Admin Panel.

3. Exiting a Vehicle

1. From the Admin Panel, click `Exit Vehicle`.
2. Enter the **Receipt Number** and click `Check`:
   - If found, vehicle details (driver name, license plate, slot, vehicle type, entry time, hours parked, payment due) are displayed.
   - If not found or already exited, an error message appears.
3. Verify the **Payment Due** (e.g., Rs. 140 for a car parked 2 hours at Rs. 70/hour).
4. Enter the **Payment Amount** received from the driver.
5. Check the **Paid** checkbox to confirm payment. 6. Click `Update`:
   - If payment is sufficient, the exit is recorded, the slot is freed, and a confirmation message shows details (including change if overpaid). o If payment is insufficient or the checkbox is unchecked, an error message appears.

7. Click `← Back` to return to the Admin Panel.

4. Searching for a Vehicle

1. From the Admin Panel, click `Search`.
2. Enter the **Receipt Number** and click `Search`:
   - If the vehicle is currently parked, details (driver name, license plate, slot, vehicle type, entry time, hours parked, payment due) are displayed.
   - If the vehicle has exited, exit details (including exit time, payment received, change returned) are shown. o If not found, a “Not Found” message appears.
3. Click `← Back` to return to the Admin Panel.

Important Notes

- **Slot Management**: Slots 1–10 are reserved for handicapped vehicles. Ensure the slot is available before entering a vehicle.
- **Payment Rates**:
  - Car: Rs. 70/hour o
    Bike: Rs. 50/hour o
    Jeep: Rs. 100/hour o
    Wagon: Rs. 150/hour
  - Truck: Rs. 250/hour o
    Handicap: Rs.
  20/hour
- **Error Messages**: If you encounter errors (e.g., invalid slot, missing receipt), follow the prompts to correct the input.
- **Data Storage**: Vehicle entries are saved in `CarInfo.txt`, exits in `ExitInfo.txt`, and slot statuses in `parking_slots.txt`. Do not modify these files manually.

Troubleshooting

- **Login Fails**: Ensure the email, password, and current time match the shift. Use default admin credentials if unsure.
- **Slot Unavailable**: Check `parking_slots.txt` for available slots or free a slot via vehicle exit.
- **Receipt Not Found**: Verify the receipt number is correct and matches an existing entry.
- **Payment Issues**: Ensure the entered payment meets or exceeds the amount due and the `Paid` checkbox is selected.

Code uploaded by Muhammad Huzaifa Zaman, F24607096