Parking Management System

Introduction

The **Parking Management System** is a robust and user-friendly desktop application designed to streamline vehicle parking operations efficiently in both public and private facilities. Developed using **Java Swing** for its graphical user interface and **MySQL** for backend data storage, the system addresses the growing challenges of managing parking slots with precision and simplicity. This application empowers administrators to monitor parking slot availability, manage vehicle entries and exits seamlessly, and maintain accurate records of parked vehicles. It provides a highly interactive and real-time interface, displaying parking slot occupancy through a **color-coded grid** for better visualization and management. The grid employs a clear color scheme: **green** for available slots and **red** for occupied slots, ensuring users can easily identify slot status immediately.

Key features of the system include the **Add Vehicle** functionality, which enables administrators to assign parking slots to vehicles based on their type, specifically distinguishing between **2-wheeler** and **4-wheeler** vehicles. Upon vehicle entry, the system automatically updates the slot status in the database, ensuring accurate and real-time monitoring. For vehicle exits, the **Remove Vehicle** feature allows administrators to quickly clear the slot details using the vehicle number, returning the slot to its "available" status. This feature simplifies slot management and minimizes errors.

The **View Parking Slots** option offers a clear and structured real-time overview of the parking area using the grid interface. By incorporating a visual approach, it allows administrators to efficiently track slot availability and occupancy. Additionally, the **View Vehicle Details** feature provides an organized table view containing essential vehicle information such as **Slot ID, Vehicle Number, Vehicle Type (2-wheeler or 4-wheeler), Contact Information**, and **Entry Time**. This comprehensive record ensures easy retrieval of vehicle details and enhances the overall management process.

In conclusion, the Parking Management System serves as an efficient and reliable solution for parking facility management. Its interactive design, real-time updates, and seamless integration of core functionalities make it an ideal tool for modern parking operations. By automating vehicle tracking, slot assignment, and data management, the system significantly reduces manual efforts, minimizes errors, and improves overall operational efficiency.

TableCreation.java

```
import java.sql.*;


public class TableCreation {
  private static final String URL = "jdbc:mysql://localhost:3306/parking_management";
  private static final String USER = "root";
  private static final String PASSWORD = "password"; // Update with your MySQL password


  public static void main(String[] args) {
    // Establish database connection
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD)) {
      // Create table if not exists with additional columns
      String createTableSQL = """
        CREATE TABLE IF NOT EXISTS parking_slots (
          slot_id INT PRIMARY KEY AUTO_INCREMENT,
          slot_number VARCHAR(10) NOT NULL,
          vehicle_type VARCHAR(20) NOT NULL,
          is_occupied BOOLEAN DEFAULT FALSE,
          vehicle_number VARCHAR(20),
          phone_number VARCHAR(15),
          entry_time DATETIME
        )
      """;
      try (Statement stmt = conn.createStatement()) {
        stmt.execute(createTableSQL);
        System.out.println("Table 'parking_slots' created successfully.");
      }

      // Add 50 slots (25 for 2-wheelers, 25 for 4-wheelers)
      for (int i = 1; i <= 50; i++) {
        String vehicleType = (i <= 25) ? "2 Wheeler" : "4 Wheeler";
        String slotNumber = "S" + i;
```

```
            // Prepare the insert statement with additional details
              String insertSlotSQL = "INSERT INTO parking_slots (slot_number, vehicle_type, vehicle_number,
phone_number, entry_time) VALUES (?, ?, ?, ?, ?)";
            try (PreparedStatement pst = conn.prepareStatement(insertSlotSQL)) {
               pst.setString(1, slotNumber);
               pst.setString(2, vehicleType);
               pst.setString(3, null); // No vehicle number initially
               pst.setString(4, null); // No phone number initially
               pst.setTimestamp(5, null); // No entry time initially
               pst.executeUpdate();
            }
          }
          System.out.println("50 parking slots (25 for 2-wheelers and 25 for 4-wheelers) added successfully.");
        } catch (SQLException e) {
          e.printStackTrace();
        }
      }
    }
```

CRUDOperations.java

```
import java.sql.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.table.DefaultTableModel;

public class CRUDOperations {
    private static final String URL = "jdbc:mysql://localhost:3306/parking_management";
    private static final String USER = "root";
    private static final String PASSWORD = "password"; // Update with your MySQL password
```

```
    // Add a vehicle to a parking slot
    public static void addVehicle(int slotId, String vehicleNumber, String phoneNumber) {
        String addVehicleSQL = "UPDATE parking_slots SET vehicle_number = ?, phone_number = ?, entry_time = NOW(), is_occupied = TRUE WHERE slot_id = ?";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pst = conn.prepareStatement(addVehicleSQL)) {

            pst.setString(1, vehicleNumber);
            pst.setString(2, phoneNumber);
            pst.setInt(3, slotId);
            int rowsAffected = pst.executeUpdate();

            if (rowsAffected > 0) {
                System.out.println("Vehicle added to slot " + slotId);
            } else {
                System.out.println("Slot " + slotId + " is already occupied or does not exist.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Remove a vehicle from a parking slot
    public static void removeVehicle(int slotId) {
        String removeVehicleSQL = "UPDATE parking_slots SET vehicle_number = NULL, phone_number = NULL, entry_time = NULL, is_occupied = FALSE WHERE slot_id = ?";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pst = conn.prepareStatement(removeVehicleSQL)) {

            pst.setInt(1, slotId);
            int rowsAffected = pst.executeUpdate();
```

```
        if (rowsAffected > 0) {
            System.out.println("Vehicle removed from slot " + slotId);
        } else {
            System.out.println("Slot " + slotId + " is already empty or does not exist.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// View all parking slots (full/empty)
public static void viewAllSlots() {
    String viewSlotsSQL = "SELECT slot_id, slot_number, vehicle_type, is_occupied FROM parking_slots";
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(viewSlotsSQL)) {

        while (rs.next()) {
            int slotId = rs.getInt("slot_id");
            String slotNumber = rs.getString("slot_number");
            String vehicleType = rs.getString("vehicle_type");
            boolean isOccupied = rs.getBoolean("is_occupied");

            System.out.println("Slot ID: " + slotId + ", Slot Number: " + slotNumber +
                    ", Vehicle Type: " + vehicleType + ", Occupied: " + (isOccupied ? "Yes" : "No"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Get details of a specific parking slot by slot_id
public static void getSlotDetails(int slotId) {
```

```
        String getSlotSQL = "SELECT slot_id, slot_number, vehicle_type, vehicle_number, phone_number, entry_time, is_occupied FROM parking_slots WHERE slot_id = ?";
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
         PreparedStatement pst = conn.prepareStatement(getSlotSQL)) {

        pst.setInt(1, slotId);
        try (ResultSet rs = pst.executeQuery()) {
            if (rs.next()) {
                String slotNumber = rs.getString("slot_number");
                String vehicleType = rs.getString("vehicle_type");
                String vehicleNumber = rs.getString("vehicle_number");
                String phoneNumber = rs.getString("phone_number");
                Timestamp entryTime = rs.getTimestamp("entry_time");
                boolean isOccupied = rs.getBoolean("is_occupied");

                System.out.println("Slot ID: " + slotId + ", Slot Number: " + slotNumber +
                        ", Vehicle Type: " + vehicleType + ", Vehicle Number: " + vehicleNumber +
                        ", Phone Number: " + phoneNumber + ", Entry Time: " + entryTime +
                        ", Occupied: " + (isOccupied ? "Yes" : "No"));
            } else {
                System.out.println("Slot ID " + slotId + " not found.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Update a parking slot with vehicle details
public static void updateSlot(int slotId, String vehicleNumber, String phoneNumber) {
    String updateSlotSQL = "UPDATE parking_slots SET vehicle_number = ?, phone_number = ?, entry_time = NOW(), is_occupied = TRUE WHERE slot_id = ?";

    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
```

```
         PreparedStatement pst = conn.prepareStatement(updateSlotSQL)) {

        pst.setString(1, vehicleNumber);
        pst.setString(2, phoneNumber);
        pst.setInt(3, slotId);
        int rowsAffected = pst.executeUpdate();

        if (rowsAffected > 0) {
            System.out.println("Slot " + slotId + " updated with vehicle details.");
        } else {
            System.out.println("Slot " + slotId + " is already occupied or does not exist.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

App.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.*;
import java.util.concurrent.TimeUnit;
import javax.swing.border.Border;

public class App {
   private static final String URL = "jdbc:mysql://localhost:3306/parking_management";
   private static final String USER = "root";
   private static final String PASSWORD = "password"; // Update with your MySQL password

   private JFrame frame; // Frame for the application
   private JPanel slotsPanel; // Panel to hold the slot labels
```

```
private JTable vehicleTable; // Table to display vehicle details

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new App().initialize());
}

public void initialize() {
    // Create the frame
    frame = new JFrame("Parking Slot Management");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(800, 600); // Increased size for the new tab

    // Create the tabbed pane
    JTabbedPane tabbedPane = new JTabbedPane();

    // Create the "Add and Remove Vehicle" tab
    JPanel addRemovePanel = createAddRemovePanel();
    tabbedPane.addTab("Add/Remove Vehicle", addRemovePanel);

    // Create the "View Parking Slots" tab with colored labels
    JPanel viewPanel = createViewPanel();
    tabbedPane.addTab("View Parking Slots", viewPanel);

    // Create the "View Vehicle Details" tab with a table
    JPanel tableViewPanel = createTableViewPanel();
    tabbedPane.addTab("View Vehicle Details", tableViewPanel);

    // Add the tabbed pane to the frame
    frame.add(tabbedPane, BorderLayout.CENTER);
    frame.setVisible(true);
}

public JPanel createAddRemovePanel() {
```

```
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(7, 2, 10, 10)); // Added spacing between rows and columns

// Set background color for the panel
panel.setBackground(Color.decode("#f1f1f1"));

// Create labels with better font style and color
JLabel vehicleTypeLabel = new JLabel("Vehicle Type:");
vehicleTypeLabel.setFont(new Font("Arial", Font.PLAIN, 14));
vehicleTypeLabel.setForeground(Color.decode("#333333"));

JLabel vehicleNumberLabel = new JLabel("Vehicle Number:");
vehicleNumberLabel.setFont(new Font("Arial", Font.PLAIN, 14));
vehicleNumberLabel.setForeground(Color.decode("#333333"));

JLabel phoneNumberLabel = new JLabel("Phone Number:");
phoneNumberLabel.setFont(new Font("Arial", Font.PLAIN, 14));
phoneNumberLabel.setForeground(Color.decode("#333333"));

// Create JComboBox and JTextField with a modern look
    JComboBox<String> vehicleTypeComboBox = new JComboBox<>(new String[] { "2 Wheeler", "4 Wheeler" });
vehicleTypeComboBox.setFont(new Font("Arial", Font.PLAIN, 14));

JTextField vehicleNumberField = new JTextField();
vehicleNumberField.setFont(new Font("Arial", Font.PLAIN, 14));
vehicleNumberField.setBackground(Color.white);
vehicleNumberField.setForeground(Color.black);
vehicleNumberField.setBorder(BorderFactory.createLineBorder(Color.decode("#cccccc"), 1)); // Light border

JTextField phoneNumberField = new JTextField();
phoneNumberField.setFont(new Font("Arial", Font.PLAIN, 14));
phoneNumberField.setBackground(Color.white);
```

```
phoneNumberField.setForeground(Color.black);
phoneNumberField.setBorder(BorderFactory.createLineBorder(Color.decode("#cccccc"), 1));

// Create buttons with customized style
JButton addButton = new JButton("Add Vehicle");
addButton.setFont(new Font("Arial", Font.BOLD, 14));
addButton.setBackground(Color.decode("#4CAF50"));
addButton.setForeground(Color.white);
addButton.setFocusPainted(false);
addButton.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
addButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));

JButton removeButton = new JButton("Remove Vehicle");
removeButton.setFont(new Font("Arial", Font.BOLD, 14));
removeButton.setBackground(Color.decode("#F44336"));
removeButton.setForeground(Color.white);
removeButton.setFocusPainted(false);
removeButton.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));
removeButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));

// Add action listener for "Add Vehicle"
addButton.addActionListener(e -> {
  // Input validation for empty fields
  if (vehicleNumberField.getText().isEmpty() || phoneNumberField.getText().isEmpty()) {
    JOptionPane.showMessageDialog(frame, "Please fill all fields.");
    return;
  }

  String vehicleType = (String) vehicleTypeComboBox.getSelectedItem();
  String vehicleNumber = vehicleNumberField.getText();
  String phoneNumber = phoneNumberField.getText();

  int availableSlotId = findAvailableSlot(vehicleType);
```

```
    if (availableSlotId != -1) {
        addVehicle(availableSlotId, vehicleNumber, phoneNumber);
        updateSlotsPanel(); // Update the slot labels after adding
        populateVehicleTable(vehicleTable); // Update the table after adding
    } else {
        JOptionPane.showMessageDialog(frame, "No available slot for " + vehicleType);
    }
});

// Add action listener for "Remove Vehicle"
removeButton.addActionListener(e -> {
    // Input validation for empty fields
    if (vehicleNumberField.getText().isEmpty()) {
        JOptionPane.showMessageDialog(frame, "Please enter a vehicle number to remove.");
        return;
    }

    String vehicleNumber = vehicleNumberField.getText();
    boolean removed = removeVehicle(vehicleNumber);
    if (removed) {
        updateSlotsPanel(); // Update the slot labels after removing
        populateVehicleTable(vehicleTable); // Update the table after removing
    } else {
        JOptionPane.showMessageDialog(frame, "Vehicle " + vehicleNumber + " Not Found.");
    }
});

// Add components to the panel
panel.add(vehicleTypeLabel);
panel.add(vehicleTypeComboBox);
panel.add(vehicleNumberLabel);
panel.add(vehicleNumberField);
panel.add(phoneNumberLabel);
```

```
    panel.add(phoneNumberField);
    panel.add(addButton);
    panel.add(removeButton);
    return panel;
}

// Create the panel to view the database records in a label-based view
public JPanel createViewPanel() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());

    // Create the panel to hold the labels for parking slots
    slotsPanel = new JPanel();
    slotsPanel.setLayout(new GridLayout(5, 5, 5, 5)); // 5x5 grid of labels
    updateSlotsPanel(); // Update the slot labels with current data
    JScrollPane scrollPane = new JScrollPane(slotsPanel);
    panel.add(scrollPane, BorderLayout.CENTER);
    return panel;
}

// Create the panel to view the vehicle details in a table format
public JPanel createTableViewPanel() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());

    // Create a table to display vehicle details
    vehicleTable = new JTable();
    JScrollPane scrollPane = new JScrollPane(vehicleTable);
    panel.add(scrollPane, BorderLayout.CENTER);

    // Fetch and display vehicle details
    populateVehicleTable(vehicleTable);
    return panel;
```

```
    }

    // Populate the table with vehicle details from the database
    public void populateVehicleTable(JTable vehicleTable) {
        String[] columnNames = { "Slot ID", "Slot Number", "Vehicle Type", "Vehicle Number", "Phone Number",
                "Entry Time" };
        DefaultTableModel model = new DefaultTableModel(columnNames, 0);
        vehicleTable.setModel(model);

        String sql = "SELECT * FROM parking_slots WHERE is_occupied = TRUE";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
                Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                int slotId = rs.getInt("slot_id");
                String slotNumber = rs.getString("slot_number");
                String vehicleType = rs.getString("vehicle_type");
                String vehicleNumber = rs.getString("vehicle_number");
                String phoneNumber = rs.getString("phone_number");
                Timestamp entryTime = rs.getTimestamp("entry_time");

                // Add row to the table model
                model.addRow(new Object[] { slotId, slotNumber, vehicleType, vehicleNumber, phoneNumber,
entryTime });
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Find the first available parking slot for the given vehicle type
    public int findAvailableSlot(String vehicleType) {
```

```
    String findSlotSQL = "SELECT slot_id FROM parking_slots WHERE vehicle_type = ? AND is_occupied = FALSE LIMIT 1";
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
         PreparedStatement pst = conn.prepareStatement(findSlotSQL)) {
        pst.setString(1, vehicleType);
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            return rs.getInt("slot_id");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1; // No available slot found
  }

  // Add a vehicle to a parking slot
  public void addVehicle(int slotId, String vehicleNumber, String phoneNumber) {
    if (!isVehicleExists(vehicleNumber)) {
        String addVehicleSQL = "UPDATE parking_slots SET vehicle_number = ?, phone_number = ?, entry_time = NOW(), is_occupied = TRUE WHERE slot_id = ?";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pst = conn.prepareStatement(addVehicleSQL)) {
            pst.setString(1, vehicleNumber);
            pst.setString(2, phoneNumber);
            pst.setInt(3, slotId);
            int rowsAffected = pst.executeUpdate();
            if (rowsAffected > 0) {
                JOptionPane.showMessageDialog(frame, "Vehicle added to slot " + slotId);
            } else {
                JOptionPane.showMessageDialog(frame, "Error adding vehicle to slot.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(frame, "Error: " + e.getMessage());
```

```
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Vehicle Number " + vehicleNumber + " Already Exists!...");
    }
}

public boolean removeVehicle(String vehicleNumber) {
    if (isVehicleExists(vehicleNumber)){
        String removeVehicleSQL = "UPDATE parking_slots SET vehicle_number = NULL, phone_number = NULL, entry_time = NULL, is_occupied = FALSE WHERE vehicle_number = ?";

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pst = conn.prepareStatement(removeVehicleSQL)) {

            getTotalAmount(vehicleNumber); // get the total parking amount.
            pst.setString(1, vehicleNumber);
            int rowsAffected = pst.executeUpdate();

            return rowsAffected > 0; // Returns true if the vehicle was removed successfully
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return false; // Vehicle not found or failed to remove
}

// Update the labels based on the parking slot data
public void updateSlotsPanel() {
    String printDatabaseSQL = "SELECT * FROM parking_slots";
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(printDatabaseSQL)) {
```

```
// Remove all previous labels from the panel
slotsPanel.removeAll();

// Create a panel for the 2-wheelers and 4-wheelers with borders
JPanel twoWheelerPanel = new JPanel();
JPanel fourWheelerPanel = new JPanel();

// Set layouts for the panels to have 5 columns and wrap after 5 slots
twoWheelerPanel.setLayout(new GridLayout(5, 5, 5, 5)); // 5x5 grid
fourWheelerPanel.setLayout(new GridLayout(5, 5, 5, 5)); // 5x5 grid

// Add a border between two-wheeler and four-wheeler sections
Border twoWheelerBorder = BorderFactory.createTitledBorder("2 Wheelers");
Border fourWheelerBorder = BorderFactory.createTitledBorder("4 Wheelers");
twoWheelerPanel.setBorder(twoWheelerBorder);
fourWheelerPanel.setBorder(fourWheelerBorder);

// Add the labels for slots based on the data
while (rs.next()) {
   int slotId = rs.getInt("slot_id");
   String vehicleNumber = rs.getString("vehicle_number");
   boolean isOccupied = rs.getBoolean("is_occupied");
   String slotNumber = rs.getString("slot_number");
   String vehicleType = rs.getString("vehicle_type");

   // Create a label for each slot with vehicle number or "Available"
   JLabel slotLabel = new JLabel(
         vehicleType.equals("2 Wheeler") ? (vehicleNumber != null ? vehicleNumber : "Available")
               : (vehicleNumber != null ? vehicleNumber : "Available"));
   slotLabel.setHorizontalAlignment(SwingConstants.CENTER);
   slotLabel.setVerticalAlignment(SwingConstants.CENTER);
   slotLabel.setPreferredSize(new Dimension(50, 50));
```

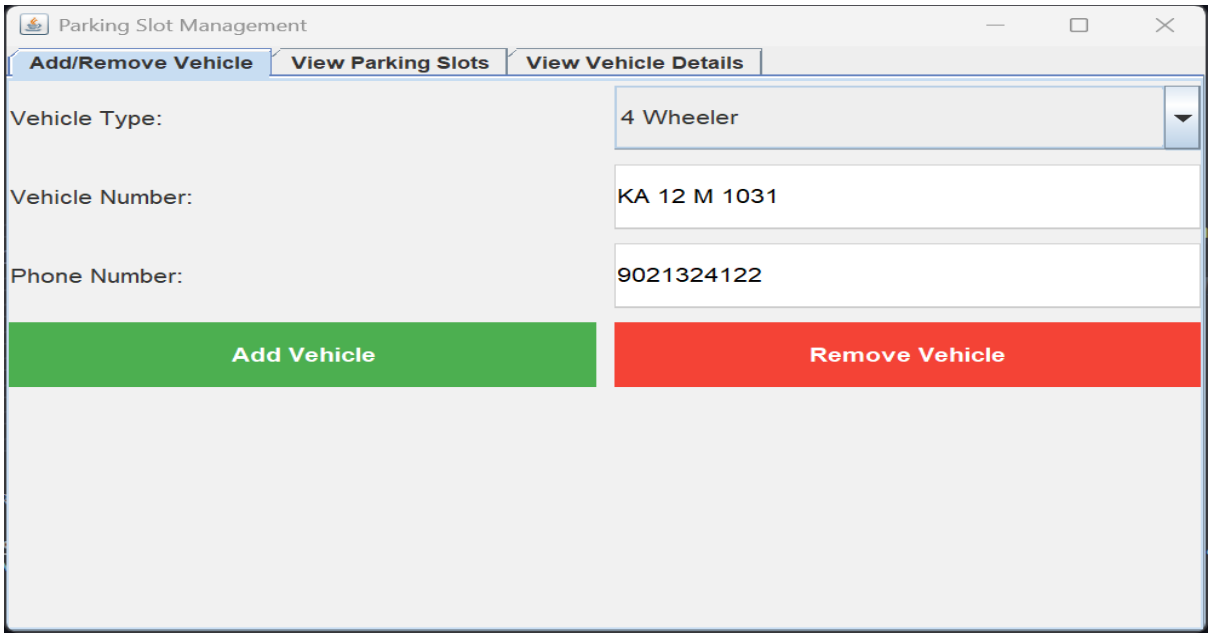
```
            // Set the label color based on availability
            if (isOccupied) {
                slotLabel.setBackground(Color.RED);
            } else {
                slotLabel.setBackground(Color.GREEN);
            }
            slotLabel.setOpaque(true);

            // Add the label to the appropriate panel based on vehicle type
            if (vehicleType.equals("2 Wheeler")) {
                twoWheelerPanel.add(slotLabel);
            } else if (vehicleType.equals("4 Wheeler")) {
                fourWheelerPanel.add(slotLabel);
            }
        }

        // Add both panels to the slotsPanel with a gap between them
        slotsPanel.setLayout(new GridLayout(2, 1, 5, 5));
        slotsPanel.add(twoWheelerPanel);
        slotsPanel.add(fourWheelerPanel);

        // Revalidate and repaint the panel to update the layout
        slotsPanel.revalidate();
        slotsPanel.repaint();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public boolean isVehicleExists(String vehicleNumber) {
    String checkVehicleSQL = "SELECT COUNT(*) FROM parking_slots WHERE vehicle_number = ?";
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
```

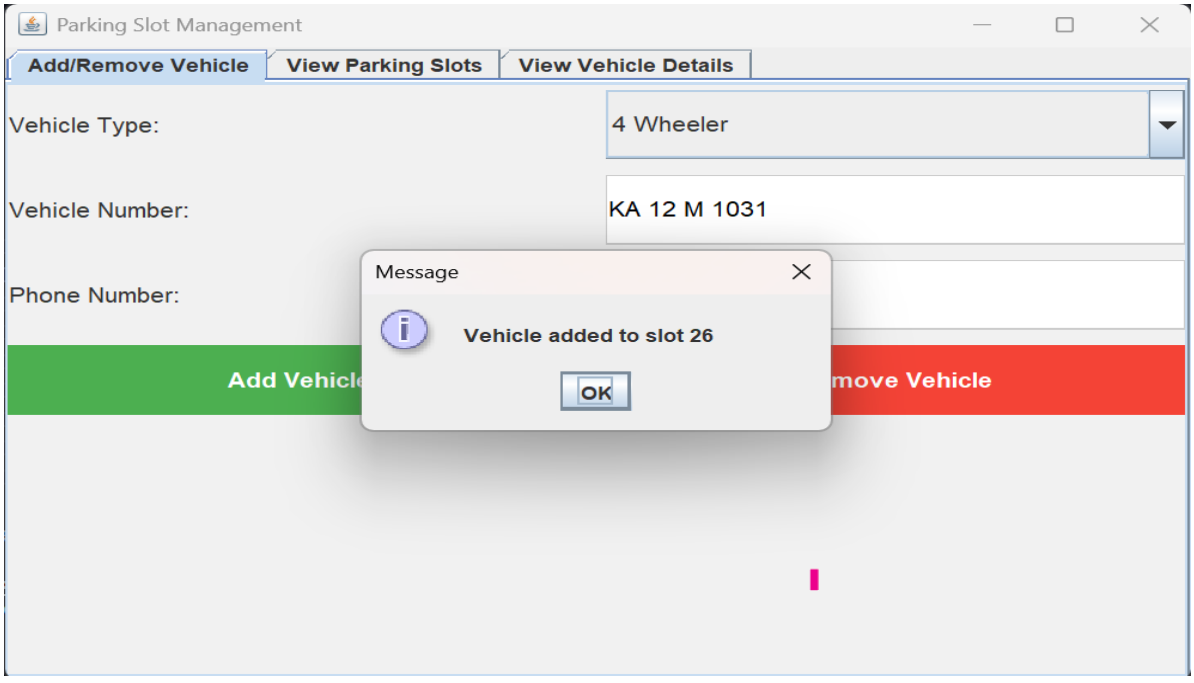
```
        PreparedStatement pst = conn.prepareStatement(checkVehicleSQL)) {
      pst.setString(1, vehicleNumber);
      ResultSet rs = pst.executeQuery();
      if (rs.next()) {
        int count = rs.getInt(1);
        return count > 0; // Returns true if the vehicle number exists
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return false; // Returns false if an error occurs or the vehicle is not found
}

public void getTotalAmount(String vehicleNumber) {
    Timestamp entryTime = getEntryTime(vehicleNumber);
    System.out.println(entryTime);

    // Define the hourly parking rate (₹50 per hour, for example)
    double hourlyRate = 50.0;

    // Get the current time
    Timestamp currentTime = new Timestamp(System.currentTimeMillis());

    // Calculate the duration between entry time and current time
    long durationInMillis = currentTime.getTime() - entryTime.getTime();

    // Convert the duration from milliseconds to hours
    long durationInHours = TimeUnit.MILLISECONDS.toHours(durationInMillis);

    // If the duration is not a full hour, round up to the next hour
    if (durationInMillis % 3600000 != 0) {
      durationInHours++;
    }
```

```
    // Calculate the total amount
    double totalAmount = durationInHours * hourlyRate;

    // Display the total amount
    JOptionPane.showMessageDialog(frame, "Total Parking Amount: ₹" + totalAmount);
}

// Function to extract entry time based on vehicle number
public Timestamp getEntryTime(String vehicleNumber) {
    // SQL query to fetch entry time for the given vehicle number
    String query = "SELECT entry_time FROM parking_slots WHERE vehicle_number = ?";

    // Declare a Timestamp variable to store the result
    Timestamp entryTime = null;

    // Establish the database connection and prepare the statement
    try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
         PreparedStatement stmt = conn.prepareStatement(query)) {

        // Set the vehicle number parameter
        stmt.setString(1, vehicleNumber);

        // Execute the query and get the result
        ResultSet rs = stmt.executeQuery();

        // If a record is found, retrieve the entry time
        if (rs.next()) {
            entryTime = rs.getTimestamp("entry_time");
        } else {
            // If no vehicle with that number is found, return null
            System.out.println("Vehicle not found!");
        }
```

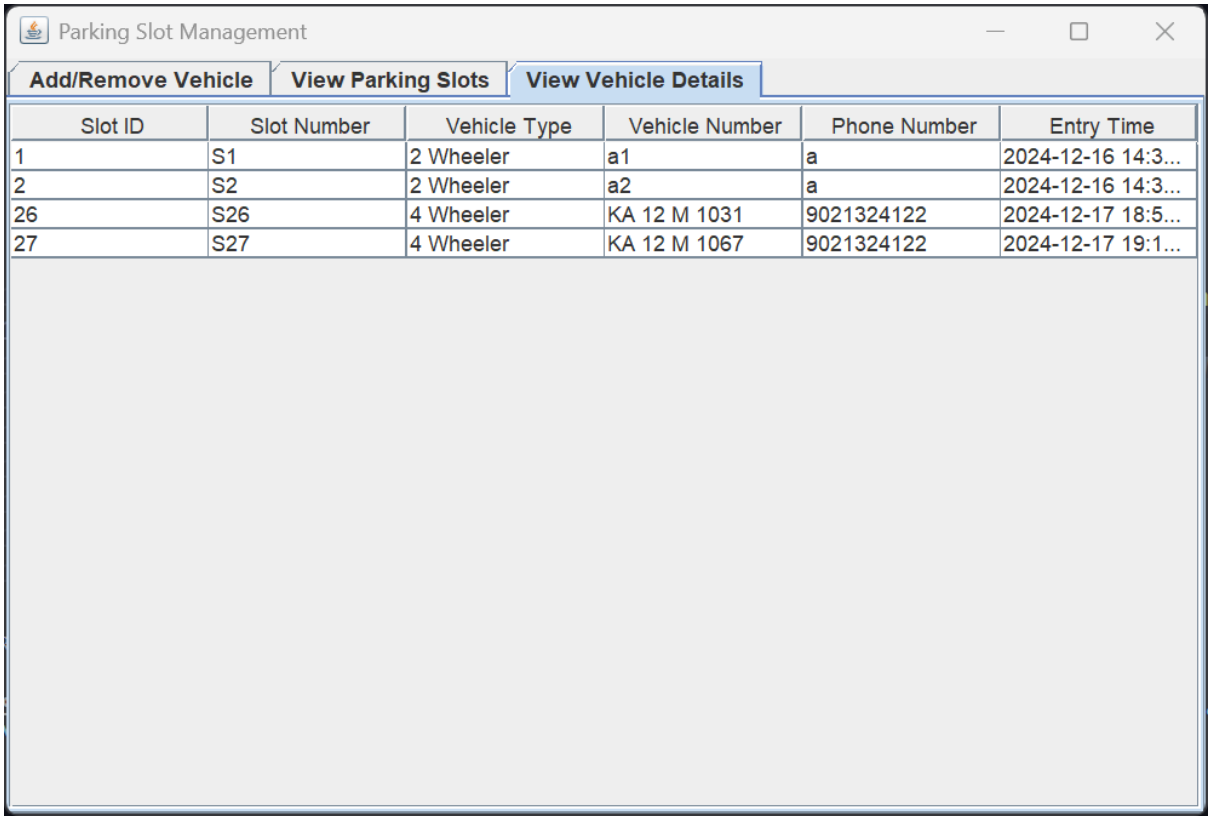
```
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return entryTime;
  }
}
```

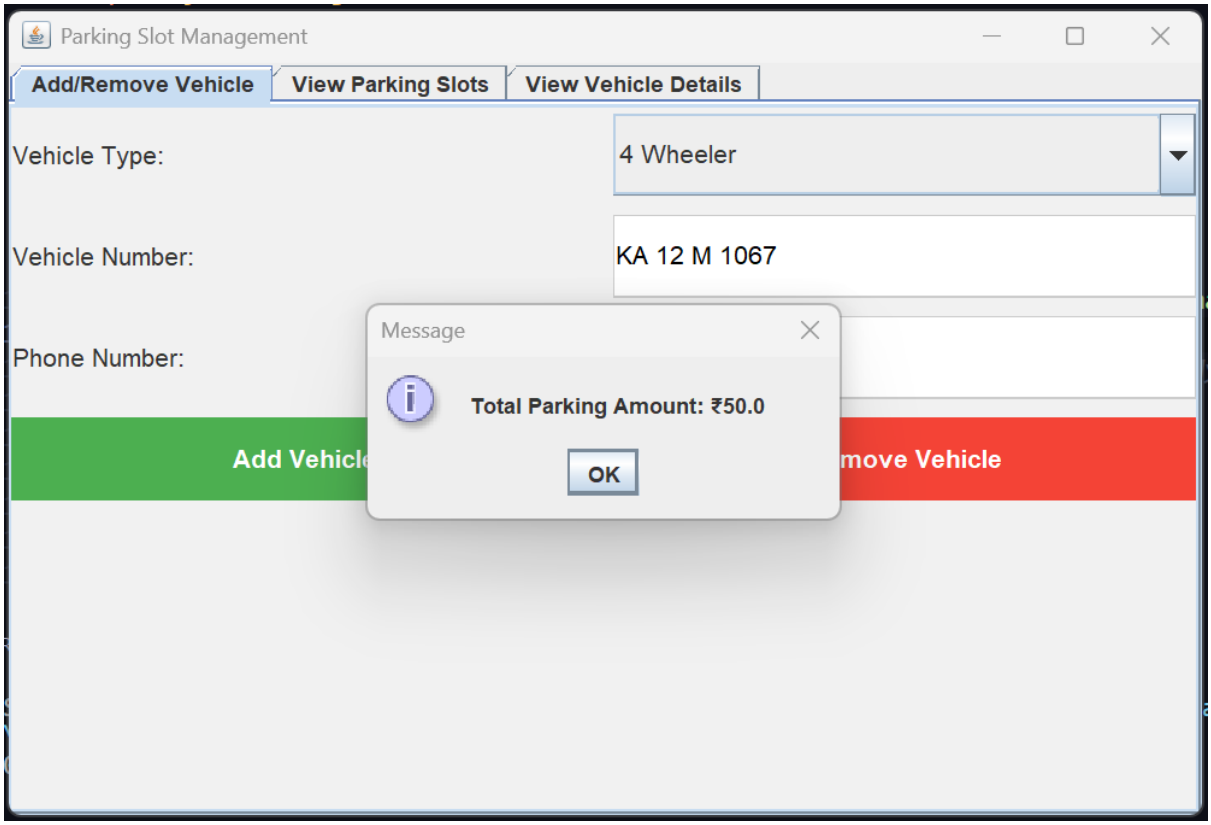
Output

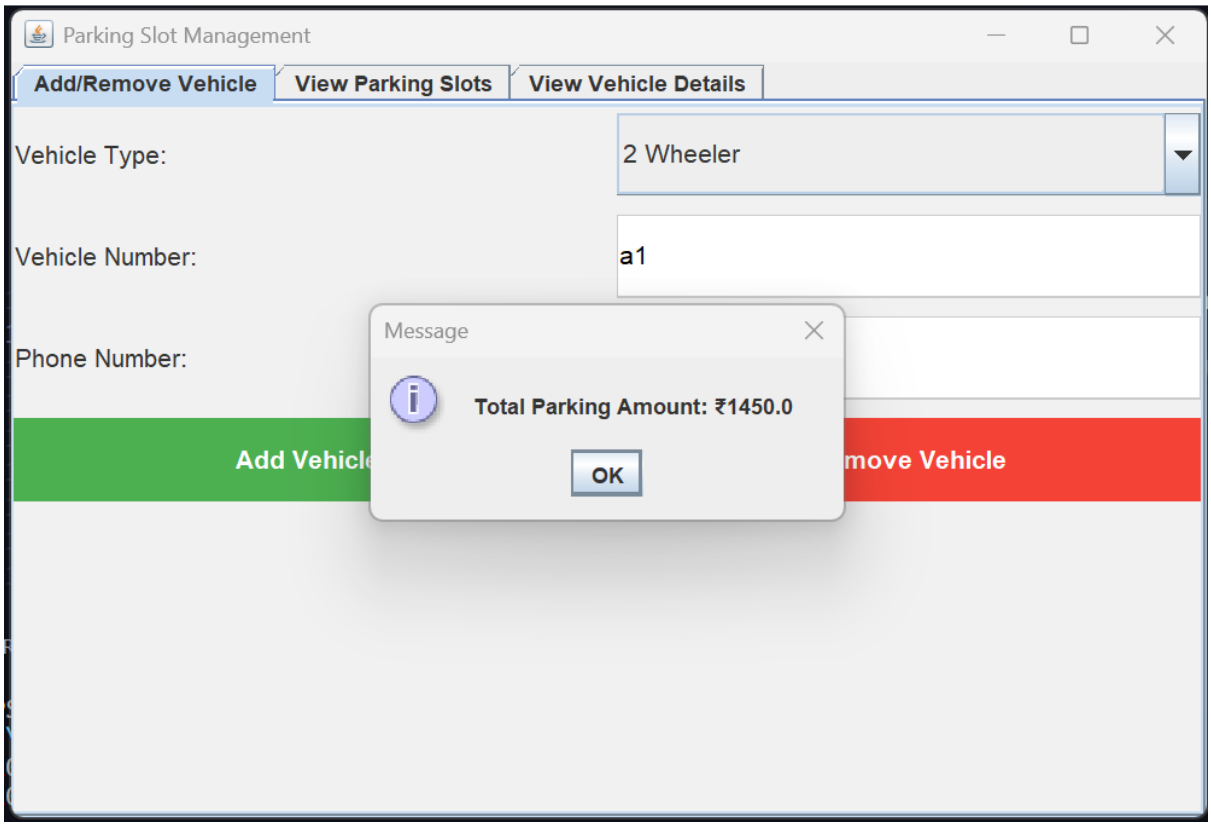




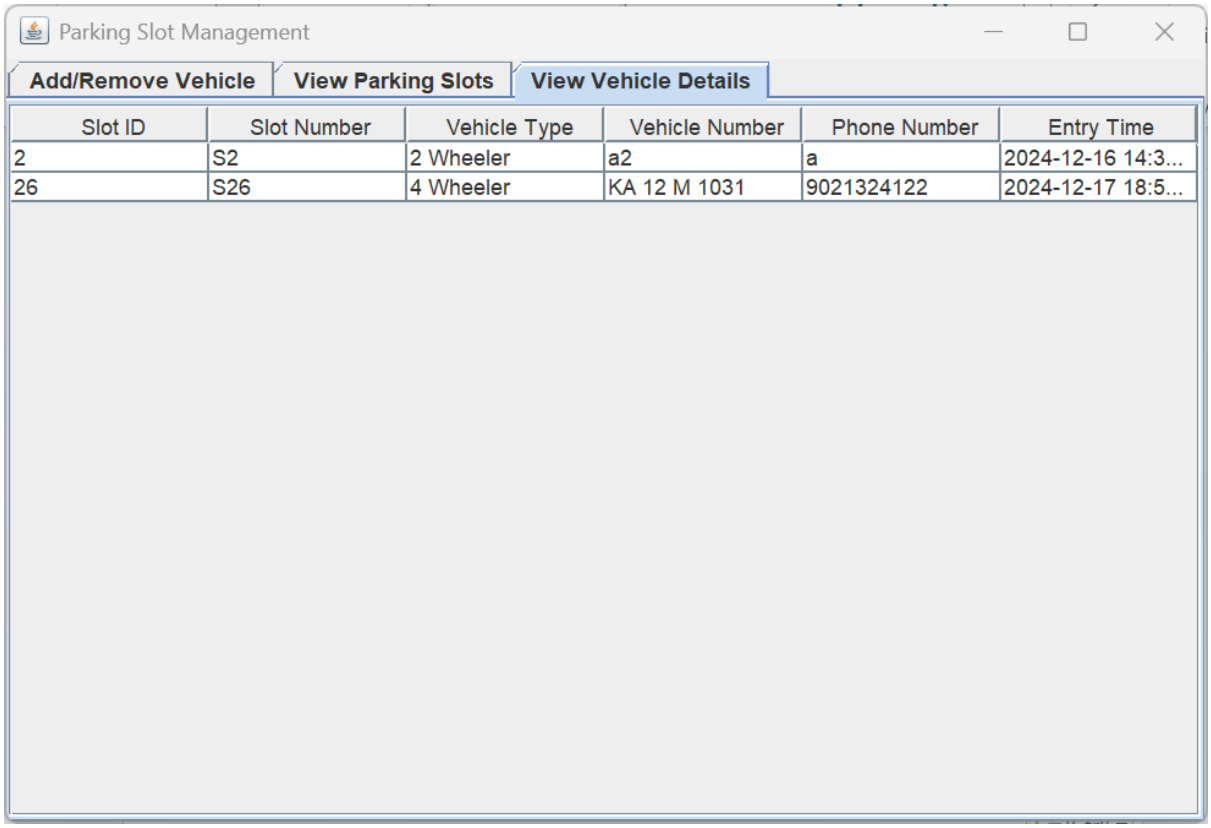

Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
2 Wheelers
a1
a2
Available
4 Wheelers
KA 12 M 1031


Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
Slot ID
Slot Number
Vehicle Type
Vehicle Number
Phone Number
Entry Time
1
S1
2 Wheeler
a1
a
2024-12-16 14:3...
2
S2
2 Wheeler
a2
a
2024-12-16 14:3...
26
S26
4 Wheeler
KA 12 M 1031
9021324122
2024-12-17 18:5...
27
S27
4 Wheeler
KA 12 M 1067
9021324122
2024-12-17 19:1...


Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
Vehicle Type:
4 Wheeler
Vehicle Number:
KA 12 M 1067
Phone Number:
Message
Total Parking Amount: ₹50.0
OK
Add Vehicle
move Vehicle


Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
Vehicle Type:
2 Wheeler
Vehicle Number:
a1
Phone Number:
Message
Total Parking Amount: ₹1450.0
OK
Add Vehicle
move Vehicle


Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
2 Wheelers
Available
a2
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
Available
4 Wheelers
KA 12 M 1031
Available
Available
Available
Available


Parking Slot Management
Add/Remove Vehicle
View Parking Slots
View Vehicle Details
Slot ID
Slot Number
Vehicle Type
Vehicle Number
Phone Number
Entry Time
2
S2
2 Wheeler
a2
a
2024-12-16 14:3...
26
S26
4 Wheeler
KA 12 M 1031
9021324122
2024-12-17 18:5...