CSIT 121 Project Report

Mikaeel Faraz Safdar - 8074689
Mohannad Alkurdi - 7868510
Devansh Popat - 8085298
Mohammed Salem - 7777139

Table Of Contents

Project Proposal

Application Details:

Our application is an Automobile Sales System. The system allows users to interact with a vehicle inventory, make purchases, and manage administrative tasks.

The program allows users to buy vehicles listed in the inventory from the showroom. It includes a user-friendly interface for administrators to manage the inventory and customers to browse available vehicles, make purchases, and view receipts. The application features a GUI based approach to enhance user interaction and simplify the overall experience.

Application Features:

User Authentication:

- The program consists of two types of users: administrators and customers.
- Users log in with their respective usernames and passwords to gain access to the system.

Administrator Functionality:

- Admins have privileged access to manage the vehicle inventory.
- Admins can add vehicles to the inventory, delete vehicles, and view the current inventory.
- When adding a vehicle, the system prompts for specific details such as ID, wheels, colour, engine, year, mileage, make, model, and price. Additional details are collected based on the vehicle type, such as the number of doors and interior colour for cars or storage box availability, type, and engine size for motorcycles.

Customer Functionality:

- Customers can log in to the system using their credentials.
- Once logged in, customers have access to a customer menu with several options.
- Customers can view the available inventory, including details such as vehicle ID, make, model, and price.
- Customers can purchase vehicles by selecting a vehicle from the inventory and generating a receipt. The system assigns a unique receipt ID for each purchase.
- The system keeps track of customer receipts, allowing customers to view their purchase history and receipts.

GUI-Based Interaction:

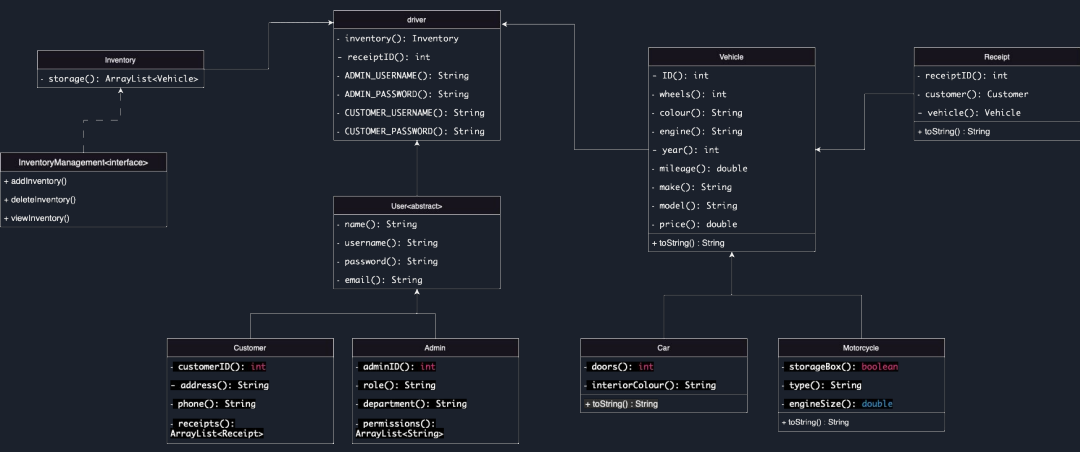
- The application utilizes dialog boxes and graphical interfaces to present information and interact with users.
- Dialog boxes are used to display menus, prompt for input, and show messages.
- The GUI approach enhances usability and simplifies user interaction.

Class Design

Our application consists of 9 classes including:

1. User - an abstract base class for different types of users in the application such as Customer and Admin
2. Customer - subclass of user class and represents a customer in the program.
3. Admin - subclass of user class and represents an administrator in the program.
4. Vehicle - represents a vehicle in the program.
5. Receipt - represents a receipt in the program.
6. Car - subclass of vehicle class and represents a car in the program.
7. Motorcycle - subclass of vehicle class and represents a motorcycle in the program.
8. Inventory - represents the inventory management system in the program and manages the vehicles available for sale.
9. Driver - runs the program, provides the main menu, login page, and menus for both admins and customers.

UML Diagram


driver
- inventory(): Inventory
- receiptID(): int
- ADMIN_USERNAME(): String
- ADMIN_PASSWORD(): String
- CUSTOMER_USERNAME(): String
- CUSTOMER_PASSWORD(): String
Inventory
- storage(): ArrayList<Vehicle>
InventoryManagement<interface>
+ addInventory()
+ deleteInventory()
+ viewInventory()
User<abstract>
- name(): String
- username(): String
- password(): String
- email(): String
Vehicle
- ID(): int
- wheels(): int
- colour(): String
- engine(): String
- year(): int
- mileage(): double
- make(): String
- model(): String
- price(): double
+ toString() : String
Receipt
- receiptID(): int
- customer(): Customer
- vehicle(): Vehicle
+ toString() : String
Customer
- customerID(): int
~ address(): String
- phone(): String
- receipts(): ArrayList<Receipt>
Admin
- adminID(): int
+ role(): String
- department(): String
- permissions(): ArrayList<String>
Car
- doors(): int
- interiorColour(): String
+ toString() : String
Motorcycle
- storageBox(): boolean
- type(): String
- engineSize(): double
+ toString() : String

User Class

```
public abstract class User {
    private String name;
    private String username;
    private String password;
    private String email;

    public User(String name, String username, String password, String email) {
        this.name = name;
        this.username = username;
        this.password = password;
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getEmail() {
        return email;
    }
}
```

User is an abstract class that represents a user in the program. It contains attributes that are common among different user types such as Admin and Customer.

Attributes of the class include name, username, password, and email, which helps authenticate the user.

Getter methods for each of the attributes are included.

Customer Class

```java
import java.util.ArrayList;

class Customer extends User {
    private int customerID;
    private String address;
    private String phone;
    private ArrayList<Receipt> receipts;

    public Customer(String name, String username, String password, String email, int customerID, String address, String phone) {
        super(name, username, password, email);
        this.customerID = customerID;
        this.address = address;
        this.phone = phone;
        this.receipts = new ArrayList<>();
    }

    public int getCustomerID() {
        return customerID;
    }

    public String getAddress() {
        return address;
    }

    public String getPhone() {
        return phone;
    }

    public ArrayList<Receipt> getReceipts() {
        return receipts;
    }

    public void addReceipt(Receipt receipt) {
        receipts.add(receipt);
    }
}
```

Customer is a subclass of the User class. It represents a specific type of User.

It extends the User class and adds additional attributes such as customerID, address, phone and features an ArrayList receipts.

It also features getters for the additional attributes and a method addReceipt which adds a receipt to the customer's list of receipts.

Admin Class

```java
import java.util.ArrayList;

class Admin extends User {
    private int adminID;
    private String role;
    private String department;
    private ArrayList<String> permissions;

    public Admin(String name, String username, String password, String email, int adminID, String role, String department) {
        super(name, username, password, email);
        this.adminID = adminID;
        this.role = role;
        this.department = department;
        this.permissions = new ArrayList<>();
    }

    public int getAdminID() {
        return adminID;
    }

    public String getRole() {
        return role;
    }

    public String getDepartment() {
        return department;
    }

    public ArrayList<String> getPermissions() {
        return permissions;
    }

    public void addPermission(String permission) {
        permissions.add(permission);
    }

    public void removePermission(String permission) {
        permissions.remove(permission);
    }
}
```

Admin is a subclass of the User class. It represents a specific type of User.

It extends the User class and adds additional attributes such as adminID, role, department and features an ArrayList permissions.

It also features getters for the additional attributes and methods addPermission which adds a permission to the admin's list of permissions and removePermission which removes a permission from the admin's list of permissions.

Vehicle Class

```
class Vehicle {
    private int ID;
    private int wheels;
    private String colour;
    private String engine;
    private int year;
    private double mileage;
    private String make;
    private String model;
    private double price;

    public Vehicle(int ID, int wheels, String colour, String engine, int year, double mileage, String make,
                   String model, double price) {
        this.ID = ID;
        this.wheels = wheels;
        this.colour = colour;
        this.engine = engine;
        this.year = year;
        this.mileage = mileage;
        this.make = make;
        this.model = model;
        this.price = price;
    }

    public int getID() {
        return ID;
    }

    public int getWheels() {
        return wheels;
    }

    public String getColour() {
        return colour;
    }

    public String getEngine() {
        return engine;
    }

    public int getYear() {
        return year;
    }

    public double getMileage() {
        return mileage;
    }

    public String getMake() {
        return make;
    }

    public String getModel() {
        return model;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return "Vehicle ID: " + ID + "\n" +
                "Wheels: " + wheels + "\n" +
                "Colour: " + colour + "\n" +
                "Engine: " + engine + "\n" +
                "Year: " + year + "\n" +
                "Mileage: " + mileage + "\n" +
                "Make: " + make + "\n" +
                "Model: " + model + "\n" +
                "Price: " + price;
    }
}
```

Vehicle represents a vehicle in the program. It contains the properties of a vehicle.

Attributes in the class include ID, wheels, colour, engine, year, mileage, make, model, price. These are common attributes that can be found in every vehicle whether it is a car or a motorcycle.

It also features getters for all the attributes and consists of a toString method which provides a string representation of the vehicle's information.

Receipt Class

```
class Receipt {
    private int receiptID;
    private Customer customer;
    private Vehicle vehicle;

    public Receipt(int receiptID, Customer customer, Vehicle vehicle) {
        this.receiptID = receiptID;
        this.customer = customer;
        this.vehicle = vehicle;
    }

    public int getReceiptID() {
        return receiptID;
    }

    public Customer getCustomer() {
        return customer;
    }

    public Vehicle getVehicle() {
        return vehicle;
    }

    @Override
    public String toString() {
        return "Receipt ID: " + receiptID + ", Customer: " + customer.getName() + ", Vehicle: " + vehicle.getMake() + " " + vehicle.getModel();
    }
}
```

The Receipt class represents a receipt for a vehicle purchase in the program.

Attributes in the class include receiptID, which is unique for each purchase, customer, which is of type Customer representing the customer who made the purchase, and vehicle, which is of type Vehicle representing the vehicle that was purchased.

It also features getters for all the attributes and consists of a toString method which provides a string representation of the receipt's information.

Car Class

```java
class Car extends Vehicle {
    private int doors;
    private String interiorColour;

    public Car(int ID, int wheels, String colour, String engine, int year, double mileage, String make,
               String model, double price, int doors, String interiorColour) {
        super(ID, wheels, colour, engine, year, mileage, make, model, price);
        this.doors = doors;
        this.interiorColour = interiorColour;
    }

    public int getDoors() {
        return doors;
    }

    public String getInteriorColour() {
        return interiorColour;
    }

    @Override
    public String toString() {
        return super.toString() + "\n" +
                "Doors: " + doors + "\n" +
                "Interior Colour: " + interiorColour;
    }
}
```

Car is a subclass of the Vehicle class. It represents a specific type of Vehicle.

It extends the vehicle class and adds additional attributes such as doors and interiorColour.

It also features getters for the additional attributes and a method toString which provides a string representation of the car's information.

Motorcycle Class

```
class Motorcycle extends Vehicle {
    private boolean storageBox;
    private String type;
    private double engineSize;

    public Motorcycle(int ID, int wheels, String colour, String engine, int year,
                      double mileage, String make, String model, double price,
                      boolean storageBox, String type, double engineSize) {
        super(ID, wheels, colour, engine, year, mileage, make, model, price);
        this.storageBox = storageBox;
        this.type = type;
        this.engineSize = engineSize;
    }

    public boolean hasStorageBox() {
        return storageBox;
    }

    public String getType() {
        return type;
    }

    public double getEngineSize() {
        return engineSize;
    }

    @Override
    public String toString() {
        return super.toString() + "\n" +
                "Storage Box: " + storageBox + "\n" +
                "Type: " + type + "\n" +
                "Engine Size: " + engineSize;
    }
}
```

Motorcycle is a subclass of the Vehicle class. It represents a specific type of Vehicle.

It extends the vehicle class and adds additional attributes such as storageBox, type, and engineSize.

It also features getters for the additional attributes and a method toString which provides a string representation of the motorcycle's information.

Inventory Class

```java
import java.util.ArrayList;

class Inventory {
    private ArrayList<Vehicle> storage;

    public Inventory() {
        storage = new ArrayList<>();
    }

    public void addInventory(Vehicle vehicle) {
        storage.add(vehicle);
        System.out.println("Vehicle added to inventory.");
    }

    public boolean deleteInventory(Vehicle vehicle) {
        boolean removed = storage.remove(vehicle);
        if (removed) {
            System.out.println("Vehicle removed from inventory.");
        }
        return removed;
    }

    public void viewInventory() {
        System.out.println("==== Inventory ====");
        System.out.printf("%-12s %-8s %-10s %-10s %-6s %-8s %-12s %-10s %-8s %-15s %-8s %-12s\n",
                "Vehicle ID", "Wheels", "Colour", "Engine", "Year", "Mileage", "Make", "Model", "Price",
                "Type", "Interior Colour", "Engine Size");
        System.out.println("-------------------------------------------------------------------------------------------------------------------------------------");
        for (Vehicle vehicle : storage) {
            System.out.printf("%-12s %-8s %-10s %-10s %-6s %-8s %-12s %-10s %-8s",
                    vehicle.getID(), vehicle.getWheels(), vehicle.getColour(), vehicle.getEngine(),
                    vehicle.getYear(), vehicle.getMileage(), vehicle.getMake(), vehicle.getModel(),
                    vehicle.getPrice());
            if (vehicle instanceof Car) {
                Car car = (Car) vehicle;
                System.out.printf("%-15s %-8s\n", "Car", car.getInteriorColour());
            } else if (vehicle instanceof Motorcycle) {
                Motorcycle motorcycle = (Motorcycle) vehicle;
                System.out.printf("%-15s %-8s\n", "Motorcycle", motorcycle.getEngineSize());
            } else {
                System.out.println();
            }
            System.out.println("-------------------------------------------------------------------------------------------------------------------------------------");
        }
        System.out.println();
    }

    public Vehicle getInventory(int vehicleID) {
        for (Vehicle vehicle : storage) {
            if (vehicle.getID() == vehicleID) {
                return vehicle;
            }
        }
        return null;
    }
}
```

The inventory class manages the inventory of vehicles. It stores all vehicle objects along with providing methods to add, delete, view, and retrieve vehicles from the inventory.

It contains a storage ArrayList that holds the vehicle objects in the inventory.

Methods it consists of include :
addInventory which adds a vehicle to the inventory by adding it to the ArrayList.
deleteInventory which removes a vehicle from the inventory if it exists in the ArrayList and returns a success/failure message.
viewInventory which displays the details of each vehicle present in the ArrayList and distinguishes between car and motorcycle.
getInventory which searches the ArrayList for a matching vehicleID and returns it or displays an error message if not found.

Driver Class

```java
public class driver {
    private static Inventory inventory = new Inventory();
    private static int receiptID = 1;
    private static final String ADMIN_USERNAME = "admin";
    private static final String ADMIN_PASSWORD = "admin123";
    private static final String CUSTOMER_USERNAME = "customer";
    private static final String CUSTOMER_PASSWORD = "customer123";

    public static void main(String[] args) {
        boolean exit = false;
        while (!exit) {
            int option = displayMainMenu();
            switch (option) {
                case 1:
                    adminLogin();
                    break;
                case 2:
                    customerLogin();
                    break;
                case 3:
                    exit = true;
                    System.out.println("Exiting the system...");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
                    break;
            }
        }
    }

    //Main Menu Function that presents 3 options -> Admin Login, Customer Login, Exit
    private static int displayMainMenu() {
        String[] options = {"Admin Login", "Customer Login", "Exit"};
        return JOptionPane.showOptionDialog(null, "==== Car Sales System ====", "Main Menu",
                JOptionPane.DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]) + 1;
    }


    //If user selects admin login, presents options for admin username and password and authenticates with the details saved.
    private static void adminLogin() {
        String username = JOptionPane.showInputDialog("Username:");
        String password = JOptionPane.showInputDialog("Password:");
        if (username.equals(ADMIN_USERNAME) && password.equals(ADMIN_PASSWORD)) {
            Admin admin = new Admin("Admin", username, password, "admin@example.com", 1, "Admin", "Sales");
            adminMenu(admin);
        } else {
            //Error thrown if not a match
            JOptionPane.showMessageDialog(null, "Invalid username or password. Please try again.");
        }
    }

    //If admin is authenticated, the admin menu is presented to them.
    private static void adminMenu(Admin admin) {
        boolean logout = false;
        while (!logout) {
            int option = displayAdminMenu();
            switch (option) {
                case 1:
                    addVehicleToInventory();
                    break;
                case 2:
                    deleteVehicleFromInventory();
                    break;
                case 3:
                    inventory.viewInventory();
                    break;
                case 4:
                    logout = true;
                    System.out.println("Logging out as admin...");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
                    break;
            }
        }
    }
```

The main method that handles the user interface for interacting with the system.

Contains a while loop that displays the main menu and prompts the user with admin or customer login options until the user opts to Exit.

The adminLogin method prompts the user to enter a username and password, if authenticated it calls the adminMenu method.

The adminMenu method displays the Admin Menu and prompts the user to choose between adding/removing vehicle from the inventory, view inventory, or log out.

Driver Class

```
//displays admin menu and takes input for what to do next
private static int displayAdminMenu() {
    String[] options = {"Add Vehicle to Inventory", "Delete Vehicle from Inventory", "View Inventory", "Logout"};
    return JOptionPane.showOptionDialog(null, "==== Admin Menu ====", "Admin Menu",
            JOptionPane.DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]) + 1;
}

//allows user to add a vehicle to the inventory list
private static void addVehicleToInventory() {
    String[] vehicleOptions = {"Car", "Motorcycle"};
    int vehicleTypeOption = JOptionPane.showOptionDialog(null, "Enter Vehicle Type:", "Add Vehicle to Inventory",
            JOptionPane.DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, vehicleOptions, vehicleOptions[0]);
    if (vehicleTypeOption == JOptionPane.CLOSED_OPTION) {
        JOptionPane.showMessageDialog(null, "Invalid vehicle type option. Vehicle not added to inventory.");
        return;
    }

    int vehicleID = 0;

    try {
        vehicleID = Integer.parseInt(JOptionPane.showInputDialog("Enter Vehicle ID:"));
        // Rest of the code
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Invalid input for Vehicle ID. Please enter a valid integer.");
    }

    int wheels = Integer.parseInt(JOptionPane.showInputDialog("Enter Wheels:"));
    String colour = JOptionPane.showInputDialog("Enter Colour:");
    String engine = JOptionPane.showInputDialog("Enter Engine:");
    int year = Integer.parseInt(JOptionPane.showInputDialog("Enter Year:"));
    double mileage = Double.parseDouble(JOptionPane.showInputDialog("Enter Mileage:"));
    String make = JOptionPane.showInputDialog("Enter Make:");
    String model = JOptionPane.showInputDialog("Enter Model:");
    double price = Double.parseDouble(JOptionPane.showInputDialog("Enter Price:"));

    Vehicle vehicle;
    if (vehicleTypeOption == 0) {
        int doors = Integer.parseInt(JOptionPane.showInputDialog("Enter Number of Doors:"));
        String interiorColour = JOptionPane.showInputDialog("Enter Interior Colour:");
        vehicle = new Car(vehicleID, wheels, colour, engine, year, mileage, make, model, price, doors, interiorColour);
    } else {
        boolean storageBox = Boolean.parseBoolean(JOptionPane.showInputDialog("Has Storage Box (true/false):"));
        String type = JOptionPane.showInputDialog("Enter Motorcycle Type:");
        double engineSize = Double.parseDouble(JOptionPane.showInputDialog("Enter Engine Size:"));
        vehicle = new Motorcycle(vehicleID, wheels, colour, engine, year, mileage, make, model, price, storageBox, type, engineSize);
    }

    // Add the vehicle to the inventory
    inventory.addInventory(vehicle);
    JOptionPane.showMessageDialog(null, "Vehicle added to inventory successfully!");
}

private static void deleteVehicleFromInventory() {
    int vehicleID = 0;
    try {
        vehicleID = Integer.parseInt(JOptionPane.showInputDialog("Enter Vehicle ID:"));
        // Rest of the code
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Invalid input for Vehicle ID. Please enter a valid integer.");
    }

    // Find the vehicle in the inventory
    Vehicle vehicle = inventory.getInventory(vehicleID);
    if (vehicle != null) {
        boolean removed = inventory.deleteInventory(vehicle);
        if (removed) {
            JOptionPane.showMessageDialog(null, "Vehicle deleted from inventory successfully!");
        }
    } else {
        JOptionPane.showMessageDialog(null, "Vehicle not found in inventory.");
    }
}

private static void customerLogin() {
    String username = JOptionPane.showInputDialog("Username:");
    String password = JOptionPane.showInputDialog("Password:");
    if (username.equals(CUSTOMER_USERNAME) && password.equals(CUSTOMER_PASSWORD)) {
        Customer customer = new Customer("Customer", username, password, "customer@example.com", 1, "Address", "Phone");
        customerMenu(customer);
    } else {
        JOptionPane.showMessageDialog(null, "Invalid username or password. Please try again.");
    }
}
```

The addVehicleToInventory method prompts the user to enter details about the vehicle and asks whether it is a car or motorcycle. Based on the input, it then asks for additional input specific to the selected type of vehicle and adds it to the inventory

The deleteVehicleFromInventory method prompts the user for the ID of the vehicle to be deleted. It uses getInventory to find the vehicle and removes it from the inventory using deleteInventory

The customerLogin method prompts the user to enter a username and password, if authenticated it calls the customerMenu method

Driver Class

```
    private static void customerMenu(Customer customer) {
        boolean logout = false;
        while (!logout) {
            int option = displayCustomerMenu();
            switch (option) {
                case 1:
                    inventory.viewInventory();
                    break;
                case 2:
                    purchaseVehicle(customer);
                    break;
                case 3:
                    viewReceipts(customer);
                    break;
                case 4:
                    logout = true;
                    System.out.println("Logging out as customer...");
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
                    break;
            }
        }
    }

    private static int displayCustomerMenu() {
        String[] options = {"View Inventory", "Purchase Vehicle", "View Receipts", "Logout"};
        return JOptionPane.showOptionDialog(null, "==== Customer Menu ====", "Customer Menu",
                JOptionPane.DEFAULT_OPTION, JOptionPane.PLAIN_MESSAGE, null, options, options[0]) + 1;
    }

    private static void purchaseVehicle(Customer customer) {

        int vehicleID = 0;

        try {
            vehicleID = Integer.parseInt(JOptionPane.showInputDialog("Enter Vehicle ID:"));
            // Rest of the code
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "Invalid input for Vehicle ID. Please enter a valid integer.");
        }

        // Find the vehicle in the inventory
        Vehicle vehicle = inventory.getInventory(vehicleID);
        if (vehicle != null) {
            // Generate a receipt ID
            int receiptID = generateReceiptID();
            // Create a new receipt
            Receipt receipt = new Receipt(receiptID, customer, vehicle);
            // Add the receipt to the customer's receipts
            customer.addReceipt(receipt);
            JOptionPane.showMessageDialog(null, "Vehicle purchased successfully!\nReceipt ID: " + receipt.getReceiptID());
        } else {
            JOptionPane.showMessageDialog(null, "Vehicle not found in inventory.");
        }
    }

    private static void viewReceipts(Customer customer) {
        StringBuilder message = new StringBuilder("==== Receipts ====\n");
        if (customer.getReceipts().isEmpty()) {
            message.append("No receipts found.");
        } else {
            for (Receipt receipt : customer.getReceipts()) {
                message.append(receipt).append("\n");
            }
        }
        JOptionPane.showMessageDialog(null, message.toString());
    }

    private static int generateReceiptID() {
        return receiptID++;
    }
}
```

The customerMenu method displays the customer menu and prompts the user to choose between viewInventory, purchaseVehicle, viewReceipts or log out.

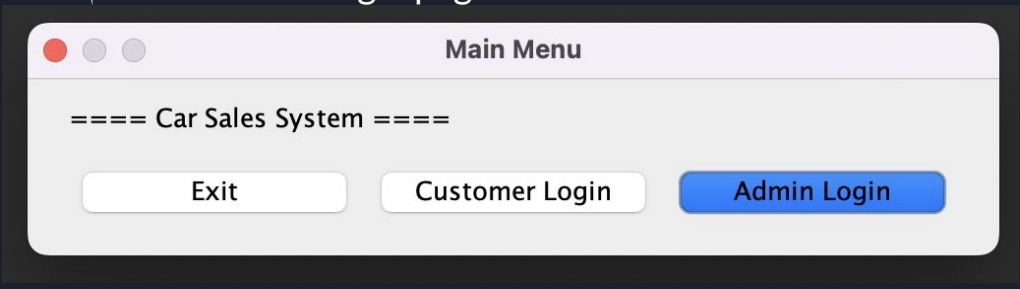
The purchaseVehicle method prompts the user for the ID of the vehicle they would like to purchase. It uses getInventory to retrieve the vehicle from the inventory class. Generates a receipt ID, creates a new receipt object, and adds it to the customer's receipts using addReceipt.

The viewReceipts method retrieves the receipts of the customer and displays them.

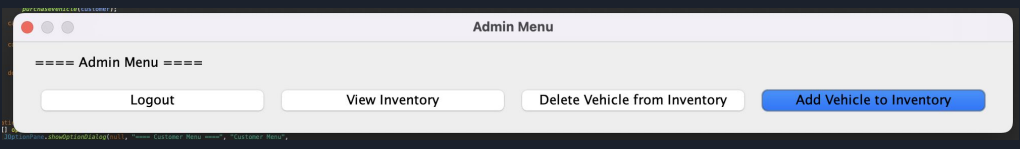
The generateReceiptID method generates a unique receipt ID for each purchase by incrementing the receiptID variable.

Program Running

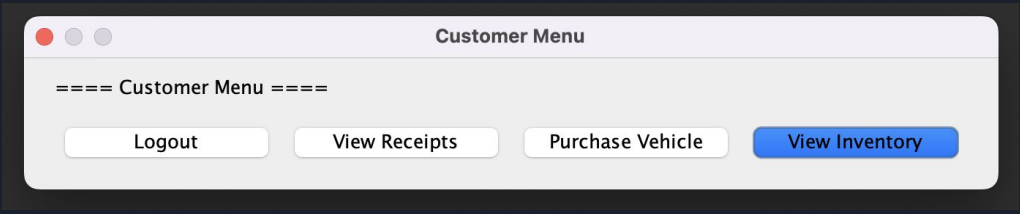
Initial login page



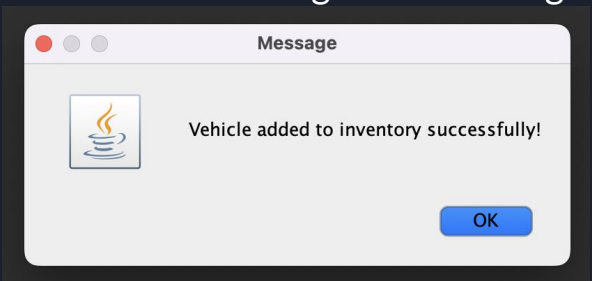
Upon successful login, this is what Admin sees



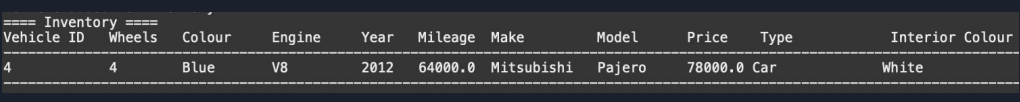
Upon successful login, this is what Customer sees



Success message when adding a vehicle to inventory



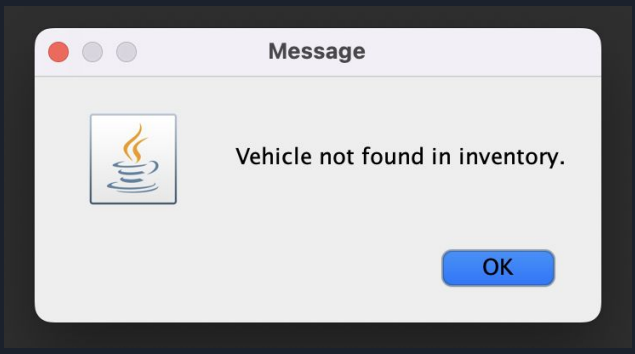
Inventory view accessible by both customer and admin.



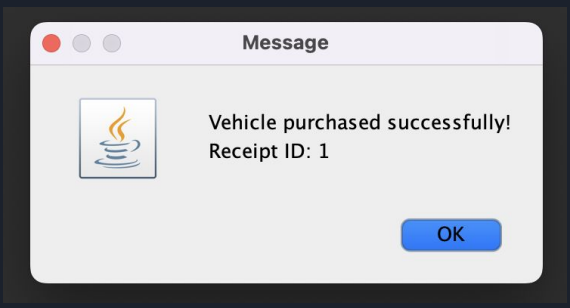
```
==== Inventory ====
Vehicle ID   Wheels   Colour     Engine     Year   Mileage  Make         Model      Price    Type            Interior Colour
-------------------------------------------------------------------------------------------------------------------------
4            4        Blue       V8         2012   64000.0  Mitsubishi   Pajero     78000.0  Car             White
-------------------------------------------------------------------------------------------------------------------------
```

Program Running

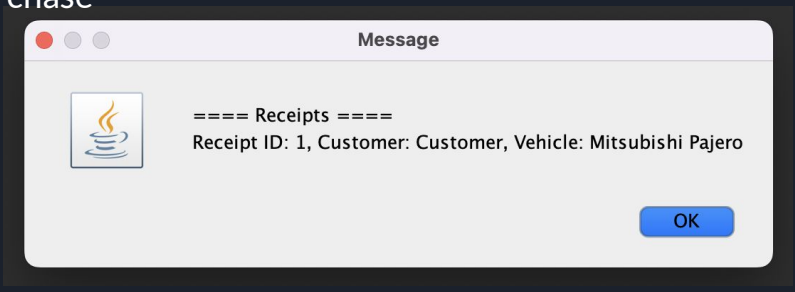
Error message when user attempts to purchase a vehicle not present in the inventory.



Success message when user purchases a vehicle present in the inventory



Receipts page accessible by user shows all unique receipts of purchases along with which customer made the purchase



Table

Classes:	Main,Vehicle,Inventory,Car,Motorcycle,User <abstract>,customer,admin,Receipt.
Inheritance:	Super class: Vehicle, Sub class: Car, motorcycle. Super class: User, Sub class: Customer, admin..
Exception class:	null
Use of files	null
interface	Inventory implements InventoryManagement
Out of the box	GUI - Graphic Interface

Distribution of Tasks

Mikaeel Faraz Safdar 8074689	Project Report, Inventory class
Mohannad Alkurdi 7868510	Vehicle, Motorcycle, Car, User , customer , admin, receipt classes and UML Diagram
Devansh Popat 8085298	Vehicle, Motorcycle, Car, User , customer , admin, receipt classes and UML Diagram
Mohammed Salem 7777139	Driver Class