EAST WEST UNIVERSITY

Department of Computer Science and Engineering

Project Report

Vehicle Fuel Management System

Course Code	CSE 110
Course Title	Object Oriented Programming
Section	11
Group No.	05

Submitted by

Al Wasy	(2025-1-60-423)
Md. Adnan Sami Chowdhuri	(2025-1-60-387)
Sanjida Alam Mithi	(2025-1-60-355)
Awandrila Sarker	(2025-1-60-365)
Sharmin Rahman	(2025-1-60-350)

Submitted to

Fouzia Risdin
Lecturer
Department of Computer Science and Engineering

Date: December 18, 2025

Contents

0.1 Project Overview and Objectives

This project, titled **Vehicle Fuel Management System**, is developed as part of the CSE110 (Object Oriented Programming) course requirements. The application is a console-based Java program that manages fuel levels for a fleet of vehicles of different types (Car, Bike, and Truck).

The primary objective is to demonstrate a solid understanding of core Object-Oriented Programming concepts while building a functional and practical system. The project fully satisfies all mandatory requirements of the course and includes a user-friendly text-based interface.

0.2 Features of the Application

The system provides the following functionalities through a menu-driven console interface:

- Display detailed information of all registered vehicles
- Refuel a specific vehicle by its ID
- Consume (use) fuel from a specific vehicle by its ID
- Graceful exit from the application

Three types of vehicles are supported, each with unique attributes:

- **Car**: Includes number of seats
- **Bike**: Includes type (e.g., Sport)
- **Truck**: Includes load capacity in tons

0.3 Fulfillment of Course Requirements

0.3.1 Class Structure (Minimum 4 Classes)

The project defines **six** well-structured classes:

- Vehicle (abstract base class)
- Car (subclass)
- Bike (subclass)
- Truck (subclass)
- FuelManagementSystem (manager class)
- Cseproject (main executable class)

All attributes are **private**, and public getters/setters are provided where necessary, ensuring proper **encapsulation**.

0.3.2 Inheritance and Polymorphism

- Vehicle is an **abstract class** with common attributes and an abstract `displayInfo()` method.
- Car, Bike, and Truck **extend** Vehicle and **override** `displayInfo()` to provide type-specific output.
- **Polymorphism** is demonstrated when storing all vehicles in an `ArrayList<Vehicle>` and calling `displayInfo()` — the correct overridden version is executed at runtime.

0.3.3 Collections

- An `ArrayList<Vehicle>` is used inside `FuelManagementSystem` to store and manage the fleet of vehicles polymorphically.
- Operations include adding vehicles and iterating through the list for display.

0.3.4 Exception Handling (More than 3 Cases)

The application handles multiple runtime exceptions:

- Invalid menu choice (handled via validation)
- Non-numeric input (`InputMismatchException`)
- Invalid vehicle ID (custom Exception: "Vehicle with ID ... not found.")
- Invalid fuel amount (≤ 0)
- Refueling beyond capacity ("Cannot refuel. Fuel capacity exceeded.")
- Consuming more fuel than available ("Not enough fuel.")

All exceptions are caught and display meaningful error messages to the user.

0.3.5 User Interface

A clean, interactive **text-based console menu** is implemented using Scanner, providing an intuitive user experience with prompts, success messages, and updated fuel levels after operations.

0.4 UML Class Diagram

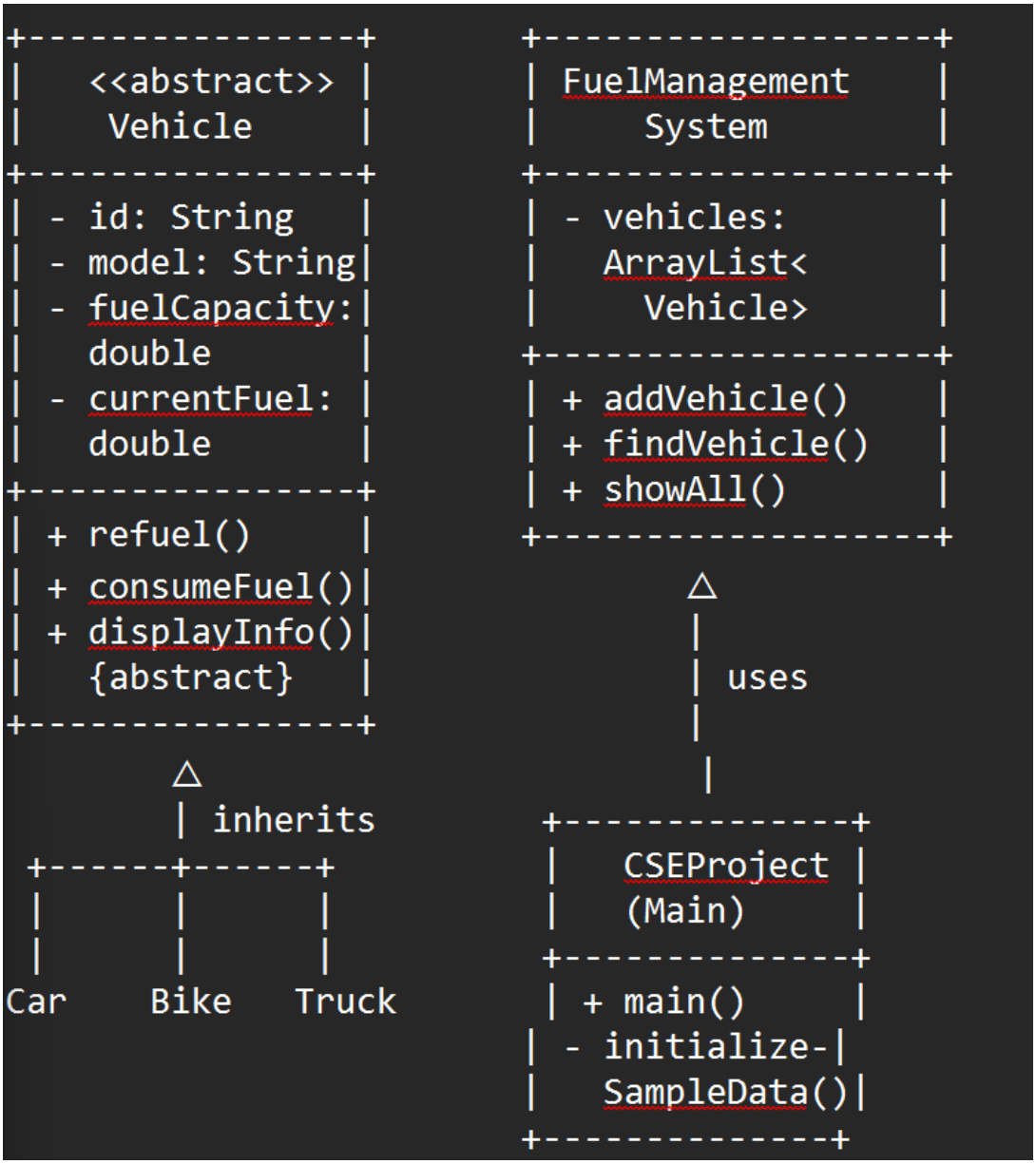

Figure 1: UML Class Diagram of Vehicle Fuel Management System

The diagram illustrates:

- Abstract Vehicle class with common attributes and methods
- Inheritance relationships to Car, Bike, and Truck
- Composition/association with FuelManagementSystem (uses ArrayList<Vehicle>)
- Main class initializing sample data

0.5 Complete Source Code

0.5.1 Package and Imports

```
package cseproject;

import java.util.ArrayList;
import java.util.Scanner;
import java.util.InputMismatchException;
```

Listing 1: Package Declaration and Imports

0.5.2 Abstract Base Class: Vehicle

```
abstract class Vehicle {
    private String id;
    private String model;
    private double fuelCapacity;
    private double currentFuel;

    public Vehicle(String id, String model, double fuelCapacity,
       double currentFuel) {
        this.id = id;
        this.model = model;
        this.fuelCapacity = fuelCapacity;
        this.currentFuel = currentFuel;
    }

    public String getId() { return id; }
    public String getModel() { return model; }
    public double getFuelCapacity() { return fuelCapacity; }
    public double getCurrentFuel() { return currentFuel; }
    public void setCurrentFuel(double currentFuel) { this.
       currentFuel = currentFuel; }

    public abstract void displayInfo();

    public void refuel(double amount) throws Exception {
        if (amount <= 0) throw new Exception("Fuel amount must be
           greater than zero.");
        if (currentFuel + amount > fuelCapacity) throw new
           Exception("Cannot refuel. Fuel capacity exceeded.");
        currentFuel += amount;
    }

    public void consumeFuel(double amount) throws Exception {
        if (amount <= 0) throw new Exception("Fuel usage must be
           greater than zero.");
        if (amount > currentFuel) throw new Exception("Not enough
           fuel.");
```

```
31          currentFuel -= amount;
32      }
33  }
```

Listing 2: Vehicle Abstract Class

0.5.3 Subclass: Car

```
1   class Car extends Vehicle {
2       private int seats;
3
4       public Car(String id, String model, double fuelCapacity,
           double currentFuel, int seats) {
5           super(id, model, fuelCapacity, currentFuel);
6           this.seats = seats;
7       }
8
9       @Override
10      public void displayInfo() {
11          System.out.println("Car");
12          System.out.println("ID: " + getId());
13          System.out.println("Model: " + getModel());
14          System.out.println("Seats: " + seats);
15          System.out.println("Fuel Capacity: " + getFuelCapacity() +
                " L");
16          System.out.println("Current Fuel: " + getCurrentFuel() + "
               L");
17          System.out.println("");
18      }
19  }
```

Listing 3: Car Class

0.5.4 Subclass: Bike

```
1   class Bike extends Vehicle {
2       private String type;
3
4       public Bike(String id, String model, double fuelCapacity,
           double currentFuel, String type) {
5           super(id, model, fuelCapacity, currentFuel);
6           this.type = type;
7       }
8
9       @Override
10      public void displayInfo() {
11          System.out.println("Bike");
12          System.out.println("ID: " + getId());
13          System.out.println("Model: " + getModel());
14          System.out.println("Type: " + type);
```

```
15          System.out.println("Fuel Capacity: " + getFuelCapacity() +
                " L");
16          System.out.println("Current Fuel: " + getCurrentFuel() + "
               L");
17          System.out.println("");
18      }
19  }
```

Listing 4: Bike Class

0.5.5 Subclass: Truck

```
1   class Truck extends Vehicle {
2       private double loadCapacity;
3
4       public Truck(String id, String model, double fuelCapacity,
           double currentFuel, double loadCapacity) {
5           super(id, model, fuelCapacity, currentFuel);
6           this.loadCapacity = loadCapacity;
7       }
8
9       @Override
10      public void displayInfo() {
11          System.out.println("Truck");
12          System.out.println("ID: " + getId());
13          System.out.println("Model: " + getModel());
14          System.out.println("Load Capacity: " + loadCapacity + "
               tons");
15          System.out.println("Fuel Capacity: " + getFuelCapacity() +
                " L");
16          System.out.println("Current Fuel: " + getCurrentFuel() + "
               L");
17          System.out.println("");
18      }
19  }
```

Listing 5: Truck Class

0.5.6 FuelManagementSystem Class

```
1   class FuelManagementSystem {
2       private ArrayList<Vehicle> vehicles = new ArrayList<>();
3
4       public void addVehicle(Vehicle v) {
5           vehicles.add(v);
6       }
7
8       public Vehicle findVehicle(String id) throws Exception {
9           for (Vehicle v : vehicles) {
10              if (v.getId().equals(id)) return v;
```

```
        }
        throw new Exception("Vehicle with ID " + id + " not found");
    }

    public void showAll() {
        for (Vehicle v : vehicles) {
            v.displayInfo();
        }
    }
}
```

Listing 6: FuelManagementSystem Class

0.5.7 Main Class: Cseproject

```
public class Cseproject {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        FuelManagementSystem system = new FuelManagementSystem();

        system.addVehicle(new Car("C1", "Toyota", 50, 20, 5));
        system.addVehicle(new Bike("B1", "Yamaha", 15, 8, "Sport"));
        system.addVehicle(new Truck("T1", "Volvo", 150, 90, 12));

        while (true) {
            System.out.println("\n===== Vehicle Fuel Management System =====");
            System.out.println("1. Show All Vehicles");
            System.out.println("2. Refuel a Vehicle");
            System.out.println("3. Use Fuel");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");

            try {
                int choice = sc.nextInt();
                sc.nextLine();

                if (choice == 1) {
                    System.out.println("\n--- Vehicle List ---");
                    system.showAll();
                } else if (choice == 2) {
                    System.out.print("\nEnter Vehicle ID (C1 / B1 / T1): ");
                    String id = sc.nextLine();
                    Vehicle v = system.findVehicle(id);
                    System.out.print("Enter fuel amount to add (in liters): ");
                    double amt = sc.nextDouble();
                    v.refuel(amt);
```

```
                        System.out.println("Refuel␣successful.");
                        System.out.println("Updated␣Fuel:␣" + v.
                            getCurrentFuel() + "␣L");
                    } else if (choice == 3) {
                        System.out.print("\nEnter␣Vehicle␣ID␣(C1␣/␣B1␣
                            /␣T1):␣");
                        String id = sc.nextLine();
                        Vehicle v = system.findVehicle(id);
                        System.out.print("Enter␣fuel␣amount␣to␣use␣(in
                            ␣liters):␣");
                        double amt = sc.nextDouble();
                        v.consumeFuel(amt);
                        System.out.println("Fuel␣usage␣successful.");
                        System.out.println("Updated␣Fuel:␣" + v.
                            getCurrentFuel() + "␣L");
                    } else if (choice == 4) {
                        System.out.println("Exiting␣system␣...");
                        break;
                    } else {
                        System.out.println("Invalid␣choice.␣Try␣again.
                            ");
                    }
                } catch (InputMismatchException e) {
                    System.out.println("Please␣enter␣numbers␣only.");
                    sc.nextLine();
                } catch (Exception e) {
                    System.out.println(e.getMessage());
                }
            }
            sc.close();
        }
}
```

Listing 7: Main Class with Menu

0.6 Program Execution and Sample Output

This section illustrates the execution of the Vehicle Fuel Management System through sample console outputs.

```
===== Vehicle Fuel Management System =====
1. Show All Vehicles
2. Refuel a Vehicle
3. Use Fuel
4. Exit
Enter your choice:
```

Figure 2: Main Menu of the System

```
--- Vehicle List ---

Car
ID: C1
Model: Toyota
Seats: 5
Fuel Capacity: 50.0 L
Current Fuel: 20.0 L

Bike
ID: B1
Model: Yamaha
Type: Sport
Fuel Capacity: 15.0 L
Current Fuel: 8.0 L

Truck
ID: T1
Model: Volvo
Load Capacity: 12.0 tons
Fuel Capacity: 150.0 L
Current Fuel: 90.0 L
```

Figure 3: Displaying All Vehicles (Option 1 Selected)

```
Enter Vehicle ID (C1 / B1 / T1): C1
Enter fuel amount to add (in liters): 15

Refuel successful.
Updated Fuel: 35.0 L
```

Figure 4: Successful Refueling of a Vehicle (Option 2)

```
Enter Vehicle ID (C1 / B1 / T1): T1
Enter fuel amount to use (in liters): 30

Fuel usage successful.
Updated Fuel: 60.0 L
```

Figure 5: Successful Fuel Consumption (Option 3)

```
Enter your choice: 4
Exiting system ...
```

Figure 6: Exiting the Program (Option 4)

These outputs confirm that the program runs correctly, handles user inputs as expected, and demonstrates all required OOP features in action.

0.7 Conclusion

The **Vehicle Fuel Management System** successfully demonstrates all required Object-Oriented Programming concepts as outlined in the CSE 110 project guidelines. It features a clean class hierarchy with inheritance and polymorphism, proper encapsulation, effective use of collections, robust exception handling, and a functional console-based user interface.

This project not only meets but exceeds the minimum requirements, providing a solid foundation that can be further extended.