PROJECT REPORT

VEHICLE FUEL MILEAGE TRACKER

Project Title: Vehicle Fuel Mileage Tracker

Name: Sapuram Deepthi

Student ID: GQT-S0 909

Institute: Global Quest Technologies

Duration: June 2025

TABLE OF CONTENTS:

CHAPTER 1

INTRODUCTION

1.1 Purpose of the Project

The purpose of the Vehicle Fuel Mileage Tracker project is to provide a simple yet effective tool for tracking and analysing a vehicle's fuel consumption, mileage, and related costs. This system is designed to help users—whether individuals or fleet operators—monitor their vehicle's performance, identify fuel usage trends, and improve overall efficiency.

By allowing users to input details such as fuel quantity, distance travelled, and fuel cost, the tracker calculates important metrics like mileage (km/l), cost per Kilometre, and total fuel expenses. These insights enable better financial planning, help detect performance issues early, and promote regular vehicle maintenance.

For businesses managing multiple vehicles, this tool offers centralized tracking and reporting, helping reduce fuel wastage and improve operational planning. It also supports environmentally responsible practices by encouraging users to maintain fuel-efficient driving habits and reduce unnecessary emissions.

1.2 Scope

The scope of the Vehicle Fuel Mileage Tracker project includes developing a user-friendly application that allows users to record and manage data related to fuel consumption and vehicle mileage. The system enables users to input details such as the amount of fuel filled, cost, date, and distance traveled. Based on this data, the application calculates key metrics like mileage (km/l), cost per kilometer, and total fuel expenses. It stores historical data, allowing users to track fuel efficiency trends over time and identify any irregularities that may indicate potential issues with the vehicle. The project is mainly targeted at individual vehicle owners and small business users who wish to monitor and optimize fuel usage. Additionally, the system can be extended to support multiple vehicles, making it useful for transport services and fleet management. Implemented using Java, the application will focus on desktop-based functionalities without complex dependencies, though it has the potential for future expansion into mobile or web platforms with features like user login, cloud storage, and GPS integration.

1.3 Target Audience

Individual Vehicle Owners:

- Want to track personal fuel consumption and expenses.
- Aim to improve fuel efficiency and reduce costs.
- Useful for daily commuters and frequent travelers.

Mobile Users:

- Prefer on-the-go data entry and mileage tracking.
- Would benefit from a future mobile-friendly version of the tracker.
- Seek instant access to fuel records and performance insights.

Fleet Managers and Business Owners:

- Need to monitor fuel usage across multiple vehicles.
- Aim to reduce operational costs and ensure timely maintenance.
- Useful for transport, delivery, and logistics companies.

Environmentally Conscious Users:

- Interested in reducing carbon emissions through efficient fuel use.
- Want to analyse driving patterns and support sustainable practices.

Students and Developers:

- Looking for a practical Java-based project for learning or academic submission.
- Want to implement real-world applications involving data tracking and analysis.

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 Hardware Requirements

The following hardware specifications are recommended for smooth development, testing, and execution of the Vehicle Fuel Mileage Tracker application:

- **Processor:** Intel Core i3 or above (or equivalent AMD processor)
- **RAM:** Minimum 4 GB (8 GB recommended for better performance)
- **Hard Disk:** At least 500 MB of free disk space for project files and application data
- **Display:** 1024x768 resolution or higher
- **Input Devices:** Standard keyboard and mouse
- **Optional:** Printer (for generating physical reports, if needed)

2.2 Software Requirements

The following software components are required for the development and execution of the Vehicle Fuel Mileage Tracker application:

- **Operating System:** Windows 10 or higher / Linux / macOS
- **Programming Language:** Java (JDK 8 or above)
- **IDE (Integrated Development Environment):** Eclipse / IntelliJ IDEA / NetBeans
- **Database (Optional for extended version):** MySQL / SQLite
- **Build Tools:** Apache Maven (if using dependency management)
- **Java Runtime Environment (JRE):** Version 8 or higher.

CHAPTER 3

TECHNOLOGIES USED

3.1 Programming Languages

- **Java:**

  Java is the primary language used to develop the Vehicle Mileage Tracker. It handles user input, stores distance and fuel values, performs calculations (like mileage = distance / fuel), and displays output.

3.2 Development Environment

- **IDE Used:**
  - Eclipse IDE – Chosen for its user-friendly interface, code completion, syntax highlighting, and integrated Java compiler.
  - Alternatively, the project is also compatible with other IDEs like IntelliJ IDEA or NetBeans.
- **Java Version**:
  - Developed using Java SE 8 or above to ensure full support for Swing components and event handling.
- **Operating System:**
  - Primary development was carried out on Windows 10.
  - The application is platform-independent and can run on Linux, macOS, or other Windows versions.

- **Project Structure:**
    - Organized into packages (e.g., com.project) for better modularity and code management.
- **JDK Setup:**
    - Java Development Kit (JDK) installed and configured with environment variables to compile and run the application seamlessly.
- **Execution Support:**
    - Supports execution directly from the IDE or through a JAR file export for standalone use.

3.2 Libraries / Packages Used

java.util.ArrayList

- Purpose: To store multiple fuel entries dynamically.
- Role in Project:
    - Stores each FuelEntry object containing distance and fuel values.
    - Allows looping through all entries to display or compute average mileage.

java.util.Scanner

- Purpose: To read user input from the console.
- Role in Project:
    - Takes distance and fuel input from the user.
    - Reads the menu choice to perform the appropriate action.

java.lang.String (implicitly used)

- Purpose: Handles string formatting and text outputs.
- Role in Project:
    - Used in toString() method for displaying each entry clearly.
    - String formatting (e.g., "%.2f" for two decimal places) ensures neat output.

CHAPTER 4

PROJECT OVERVIEW

4.1 Problem Statement

Tracking fuel consumption and calculating vehicle mileage manually can be time-consuming and error-prone. Many individuals and vehicle owners do not maintain accurate records of their travel distance and fuel usage, which leads to poor fuel efficiency awareness and vehicle maintenance practices.

The problem is the **lack of a simple, user-friendly system** that allows users to **log fuel entries** and **automatically compute mileage** over time. Without such a system, users struggle to:

- Monitor vehicle efficiency
- Make informed fuel-related decisions
- Maintain consistent fuel usage records

This project aims to solve the above problem by developing a **console-based Java application** that allows users to:

- Enter the distance traveled and fuel consumed
- Store multiple entries
- View all past entries
- Calculate and display average mileage clearly

The solution is simple, efficient, and removes the need for manual calculations or paper logs.

4.2 Objectives

To develop a simple and user-friendly console-based Java application
– Allows users to easily interact with the system and track their vehicle mileage without a complex interface.

To record distance traveled and fuel consumed for each entry

– Enables users to maintain a history of trips and fuel usage.

To calculate mileage for each individual entry

– Automatically computes mileage using the formula: mileage = distance / fuel.

To display all fuel entries in a well-formatted and readable manner

– Each entry includes distance, fuel consumed, and calculated mileage, displayed with proper formatting.

To compute and display the average mileage based on all entries

– Helps users understand overall fuel efficiency across all trips.

To handle invalid or incorrect user input effectively

– Ensures that the system only accepts positive and valid numeric inputs for accurate results.

To maintain data temporarily using dynamic data structures (ArrayList)

– Allows flexible storage of multiple entries during the program's runtime.

To create a platform-independent application that runs on any operating system where Java is installed, without requiring complex setup or external libraries.

- **To enhance beginner-level understanding of logic building and condition checking**, particularly in turn-based games where user input and system-generated randomness are combined.
- **To build a foundation for more advanced games**, where this project can be extended to include additional features such as multiplayer support, animations, or board-based mechanics like in Ludo or Snake and Ladder.

4.3 Features

Console-Based User Interface

- Simple and clean interface for easy interaction via the command line.

Add Fuel Entry

- Allows users to input distance traveled and fuel consumed for each trip.

View All Entries

- Displays all previously entered records in a clear, formatted output with:
    - Distance (in km)
    - Fuel used (in liters)
    - Mileage (in km/L)

Mileage Calculation for Each Entry

- Automatically calculates and displays mileage using the formula mileage = distance / fuel.

Average Mileage Calculation

- Computes and displays the average mileage based on all saved entries to help assess overall fuel efficiency.

Input Validation

- Ensures only valid, positive numbers are accepted for distance and fuel to avoid errors in calculation.

Dynamic Data Storage (ArrayList)

- Stores multiple entries during program runtime without a fixed limit.

Menu-Driven Options

- Repeatedly prompts the user with a menu until they choose to exit the program.

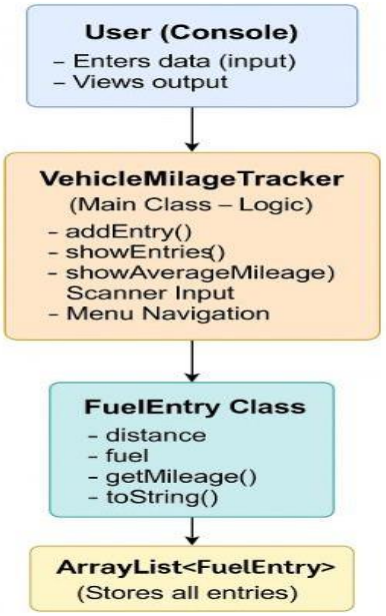
Error Handling for Invalid Choices

- Displays appropriate messages for invalid menu selections or incorrect data.

CHAPTER 5

SYSTEM DESIGN

5.1 Architecture Diagram:



The architecture of the Vehicle Mileage Tracker project is designed using a modular and object-oriented approach in Java. The system follows a simple flow that connects the user with the core logic and data handling components through a console interface. At the top level, the User interacts with the system via the command-line console, where they can input data such as the distance traveled and fuel consumed, view past entries, and check the average mileage.

The central component of the application is the `VehicleMilageTracker` class, which serves as the main controller**.** It handles user input, menu navigation, and calls the appropriate methods to perform operations like adding new entries, displaying all logs, or calculating the average mileage. It uses a `Scanner` object for input and manages all `FuelEntry` records within an `ArrayList`.

Each individual record is encapsulated in the `FuelEntry` class**,** which acts as a data model**.** This class stores two primary values: `distance` and `fuel`. It also provides a method to calculate mileage for each entry using the formula `mileage = distance / fuel`. The `toString()` method in this class is overridden to display the details in a clean and formatted way.

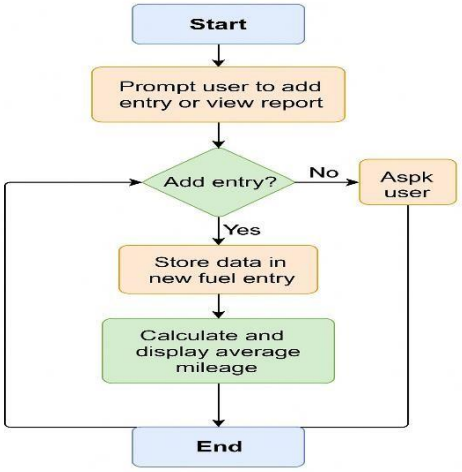
This layered structure improves the readability, maintainability, and scalability of the application and serves as a solid foundation for future upgrades like GUI integration, file storage, or multi-vehicle tracking.

5.2 Component Description

The **Vehicle Mileage Tracker** consists of four main components:

1. **User Interface (Console-Based):**
   Allows users to interact with the system through a simple menu-driven console where they can input data, view entries, and calculate average mileage.
2. **Main Logic (VehicleMilageTracker Class):**
   Controls program flow, handles user input, and performs core operations like adding entries, displaying data, and computing average mileage.
3. **Data Model (FuelEntry Class):**
   Represents individual fuel records, stores distance and fuel, and calculates mileage. It also formats data for display using the toString() method.
4. **Data Storage (ArrayList<FuelEntry>):**
   Dynamically stores all fuel entries during program execution. Acts as temporary in-memory storage.

5.3 Flow Chart



The flowchart illustrates the step-by-step process of the Vehicle Mileage Tracker application. It begins with displaying a menu to the user, allowing them to choose between adding an entry, viewing entries, calculating average mileage, or exiting the program. Based on the user's choice, the corresponding function is executed. Input validations are performed before processing. The flow continues in a loop until the user decides to exit the application.

CHAPTER 6

IMPLEMENTATION

6.1 User Interface

The user interface is **text-based and menu-driven**, designed for simplicity and clarity. The interface runs in a continuous loop and presents the following options:

Menu Options:

1. **Add Fuel Entry** – Prompts the user to enter:
    - Distance traveled (in kilometers)
    - Fuel consumed (in liters)
2. **Show All Entries** – Displays all previously added records in a neat format.
3. **Show Average Mileage** – Calculates and displays the average mileage of all entries.
4. **Exit** – Terminates the program.

Features of the UI:

- Clear instructions for each input.
- Error messages for invalid inputs or menu choices.
- Visual separation of output using line breaks (\n) for better readability.
- Each entry is displayed with numbering, making it easy to track records.

User Benefit:
This UI ensures that the user can navigate through the system without any prior technical knowledge. It avoids GUI complexity, making it lightweight and accessible.

6.2 Core Logic

The core logic is built using **object-oriented programming (OOP)** concepts and clean control structures.

Key Components:

- **FuelEntry class**:
    - Represents a single fuel log.
    - Holds private variables: distance and fuel.
    - Provides getter methods and a getMileage() method.
    - Overrides toString() to display formatted output.
- **VehicleMilageTracker class**:
    - Contains the main() method and all logic for interacting with the user.
    - Maintains a dynamic list of entries using ArrayList<FuelEntry>.
    - Handles menu navigation and input/output operations.

Functional Flow:

- Accepts input via Scanner.
- Validates inputs (e.g., fuel > 0).
- Calculates mileage per entry.
- Calculates average mileage.

6.3 Mechanism

The internal mechanism defines how the system processes inputs, stores data, and delivers output.

Step-by-Step Workflow:

1. **Menu Display & User Input**
    - The system repeatedly shows the menu using a do-while loop until the user chooses to exit.
    - Input is accepted using Scanner and processed using a switch-case structure.
2. **Data Collection**
    - On choosing "Add Entry", the user is prompted to enter values.
    - Valid input is wrapped in a FuelEntry object and added to the ArrayList.

3. **Data Validation**
    - Ensures values like negative fuel or zero fuel are rejected with proper messages.
4. **Data Storage**
    - All entries are stored temporarily in memory (RAM) using ArrayList<FuelEntry>.
    - Entries are not persisted after program exit (as file/database is not used yet).
5. **Output Generation**
    - Entries are displayed using a custom toString() method.
    - Mileage values are formatted to two decimal places for clean output.
6. **Mileage Calculation**
    - For each entry: mileage = distance / fuel.
    - For average: sum of distances / sum of fuel.
7. **Exit**
    - When the user chooses 0, the program exits gracefully with a message.

CHAPTER 7

CODE EXPLANATION

7.1 Key Classes and Methods

FuelEntry Class

- Purpose: Represents a single record of distance and fuel data.
- Key Members:
  - private double distance – stores the distance traveled.
  - private double fuel – stores the amount of fuel consumed.
- Important Methods:
  - getDistance() – returns the distance value.
  - getFuel() – returns the fuel value.
  - getMileage() – calculates and returns mileage (distance / fuel).
  - toString() – overrides default output to return a formatted string showing distance, fuel, and mileage.

VehicleMilageTracker Class

- Purpose: Main class containing the program logic and user interaction.
- Key Components:
  - ArrayList<FuelEntry> entries – stores all the user fuel logs dynamically.
  - Scanner scanner – reads user input.
- Important Methods:
  - main() – contains the program loop and menu navigation.
  - addEntry() – takes input and adds a new FuelEntry to the list.
  - showEntries() – displays all entries from the list.
  - showAverageMileage() – calculates and displays the overall average mileage**.**

7.2 Input and Output

Input

- The program uses the Scanner class to take input from the user via the console.
- Required inputs:
  - **Distance traveled** in kilometers (double)
  - **Fuel consumed** in liters (double)
  - **Menu selection** (integer)

Output

- The program provides output in a well-formatted and readable form:
    - Success messages when entries are added.
    - Formatted mileage report for each entry.
    - Average mileage with two decimal precision.
    - Menu prompts and error messages.

7.3 Error Handling

Input Validation:

- Checks to ensure that:
    - Distance is not negative
    - Fuel is greater than zero
- Prevents invalid records from being added.

Invalid Menu Choice:

- Displays a friendly error message when a user enters a number outside the valid menu options.

Empty Entry Check:

- When displaying entries or calculating average mileage:
    - Checks if the entries list is empty.
    - If empty, it shows appropriate messages like:
        - "No entries found."
        - "No data to calculate average mileage."

CHAPTER 8

TESTING AND RESULTS

8.1 Test Plan

1. Functional Testing

The main goal of testing this project is to ensure that the **Vehicle Mileage Tracker** works as expected under normal and abnormal conditions. The testing objectives include:

1. **Ensure Correct Mileage Calculation for Valid Inputs**
    - Verify that when a user enters valid distance and fuel values, the system accurately computes the mileage using the formula:
      Mileage = Distance / Fuel
    - Test with multiple entries to ensure individual and average calculations are consistent and precise (up to two decimal places).
2. **Handle Invalid Inputs Gracefully**
    - Check how the system responds to invalid inputs such as:
        - Negative distance or fuel values
        - Zero fuel input (to avoid division by zero)
        - Non-numeric input (if attempted)
    - Ensure such entries are not accepted, and a clear, user-friendly message is displayed without crashing the program.
3. **Check Average Mileage Computation**
    - After multiple entries are added, ensure that the system correctly calculates the total distance and total fuel, and computes the average mileage:
      Average Mileage = Total Distance / Total Fuel
    - Validate against manually calculated values to confirm accuracy.
4. **Display Proper Messages for Empty or Invalid Data**
    - If the user tries to view entries or calculate average mileage without adding any data, the system should show informative messages like:
        - “No entries found.”
        - “No data to calculate average mileage.”
    - The application should never crash due to empty lists or invalid operations.

8.2 Test cases

TC ID	Test Case Description	Input	Expected Output	Actual Output	Result
TC01	Add a valid fuel entry	Distance: 120Fuel: 6	Entry added successfullyMileage: 20.00 km/L	Entry added successfully	Pass
TC02	Add entry with zero fuel	Distance: 90Fuel: 0	Error: Fuel must be greater than 0	Invalid input message	Pass
TC03	Add entry with negative distance	Distance: -50Fuel: 5	Error: Distance must be positive	Invalid input message	Pass
TC04	View all entries (after 1 valid entry)	Select menu option: 2	Displays the added entry formatted with mileage	Correctly displayed	Pass
TC05	View entries when none exist	No entries, select option: 2	Message: No entries found	No entries found	Pass
TC06	Calculate average mileage (multiple)	Entries: (120, 6), (90, 4.5)	Displays average: 20.00 km/L	Average Mileage: 20.00 km/L	Pass
TC07	Invalid menu choice	Option: 9	Error: Invalid choice message	Invalid choice	Pass
TC08	Exit program	Option: 0	Message: Exiting… and end program	Exiting...	Pass

8.3 Output Screenshots

Welcome Menu

```
--- Vehicle Fuel Mileage Tracker ---
1. Add fuel entry
2. Show all entries
3. Show average mileage
0. Exit
Choose an option:
```

Add Valid Fuel Entry

```
--- Vehicle Fuel Mileage Tracker ---
1. Add fuel entry
2. Show all entries
3. Show average mileage
0. Exit
Choose an option: 1

Enter distance traveled (km): 300
Enter fuel consumed (liters): 30

Entry added successfully.
```

Display All Entries

```
--- Vehicle Fuel Mileage Tracker ---
1. Add fuel entry
2. Show all entries
3. Show average mileage
0. Exit
Choose an option: 2

--- Fuel Entries ---
1.
Distance: 300.00 km
Fuel: 30.00 L
Mileage: 10.00 km/L

2.
Distance: 400.00 km
Fuel: 40.00 L
Mileage: 10.00 km/L
```

Calculate Average Mileage

```
--- Vehicle Fuel Mileage Tracker ---
1. Add fuel entry
2. Show all entries
3. Show average mileage
0. Exit
Choose an option: 3

Average Mileage: 10.00 km/L
```

Error on Zero Fuel Input

```
--- Vehicle Fuel Mileage Tracker ---
1. Add fuel entry
2. Show all entries
3. Show average mileage
0. Exit
Choose an option: 1

Enter distance traveled (km): 300
Enter fuel consumed (liters): 0

Invalid input. Distance and fuel must be positive.
```

CHAPTER 9

CONCLUSION

9.1 Summary of Work

The Vehicle Mileage Tracker is a simple console-based Java application developed to help users monitor their vehicle's fuel usage and efficiency. The program allows users to input the distance traveled and fuel consumed for each trip and then automatically calculates the mileage for each entry. It uses object-oriented programming principles and dynamic data structures (`ArrayList`) to manage multiple fuel records.

The system also provides features to view all entries, calculate and display the average mileage, and handle invalid inputs effectively. The entire application is built using core Java, with no external libraries or databases, making it lightweight and ideal for beginners to understand key Java concepts like classes, methods, conditionals, loops, and user input handling.

9.2 Limitations

While the application serves its purpose well, there are a few limitations:

- **No Data Persistence**
  All data is stored in memory (RAM) only. Once the program is closed, all entries are lost.
- **No Date/Time Tracking**
  Entries are not timestamped, so there's no way to track when a particular fuel entry was made.
- **No Support for Multiple Vehicles**
  The application assumes a single vehicle. There's no option to categorize entries by vehicle type or number.
- **No GUI or Visual Graphs**
  Being console-based, it lacks charts or visual aids that could help better understand mileage trends.
- **No Unit Conversion**
  The program uses fixed units (km and liters). Users cannot switch between miles/gallons or other systems.

9.3 Future Enhancements

To make the application more powerful, user-friendly, and applicable to real-world usage, the following enhancements can be considered:

1. Add File or Database Storage
    - Store entries permanently using text files, CSV, or databases (like SQLite or MySQL).
2. Include Date and Time Stamps
    - Automatically log the date and time with each fuel entry for better tracking and reporting.
3. Build a GUI Version (Using Swing or JavaFX)
    - Develop a graphical version of the application with forms, buttons, and charts.
4. Support for Multiple Vehicles
    - Allow users to manage multiple vehicles and view mileage per vehicle.
5. Generate Reports and Charts
    - Use libraries like JFreeChart or export data to Excel/PDF for visual representation and sharing.
6. Mobile App Integration
    - Develop an Android version for easy real-time tracking via mobile devices.
7. Unit Conversion Option
    - Provide options to switch between metric and imperial units for global usability.

CHAPTER 10

REFERENCES

1. **Oracle Java Documentation**
   https://docs.oracle.com/javase/8/docs/
   Official documentation for Java classes and methods used in the project, such as Scanner, ArrayList, and String formatting.
2. **Stack Overflow**
   https://stackoverflow.com/
   Helped in resolving small implementation doubts related to string formatting and logic validation.
3. **Eclipse IDE User Guide**
   https://www.eclipse.org/documentation/
   Used for IDE setup, console input/output management, and debugging assistance.