BULE HORA UNIVERSITY

**COLLEGE OF INFORMATICS**

**DEPARTEMENT OF SOFTWARE ENGINEERING**

**Documentation for: - Vehicle Management System**



**GROUP MEMBERS**

**NAME ID NO**

1. Nanbon Bekele………………………………………. RU0746/14
2. Kulani Yewalashet………………………………….... RU3661/14
3. Wakuma Terefe………………………………………. RU5216/13
4. Lewi Choramo……………………………………….. RU0210/14

**Introduction**

The Vehicle Management System (VMS) is designed to streamline the buying, selling, and management of vehicles. This system caters to various user roles, including Admin, Customer, and Owner, providing a comprehensive platform for vehicle registration, information retrieval, and management. The objective is to create a user-friendly application that leverages Object-Oriented Programming (OOP) principles and design patterns to ensure maintainability, scalability, and robustness.

**1. Vehicle Management System**

**1.1 Purpose**

The purpose of this document is to outline the design, functionality, and overall framework of the Vehicle Management System (VMS). This system is intended to facilitate the buying and selling of vehicles, including cars and bikes, by providing distinct and user-friendly interfaces for different user roles: admins, customers, and vehicle owners.

**1.2 Scope**

The VMS will support several key functionalities, including:

* Vehicle Registration: Allowing owners to register their vehicles for sale.
* Viewing Available Vehicles: Enabling customers to browse the inventory of vehicles.
* Purchasing Vehicles: Facilitating the transaction process for customers looking to buy vehicles.
* Administrative Functions: Allowing admins to manage the vehicle inventory by adding and removing vehicles as needed.

**2. System Overview**

**2.1 System Architecture**

The system is designed using Object-Oriented Programming (OOP) principles, which promotes modularity and reusability. The architecture consists of multiple classes, each with specific roles:

* **Admin**: Responsible for managing vehicle addition, removal, and inventory viewing.
* **Customer**: Handles the purchase of vehicles and the viewing of available options.
* **Owner**: Registers vehicles for sale in the system.
* **Vehicle**: An abstract class that represents common properties and behaviors of all vehicles.
* **Car and Bike**: Subclasses of the Vehicle class, representing specific types of vehicles with tailored attributes.
* **VehicleRegistry**: A Singleton class that manages the collection of vehicles, ensuring only one instance exists.
* **VehicleFactory**: A Factory class that abstracts the creation process of vehicle instances, promoting loose coupling.

**2.2 Technologies Used**

* Programming Language: Java
* Development Environment: Netbeans
* Version Control: Git for source code management, enabling collaborative development and version tracking.

**3. Design Quality**

**3.1 Principles and Patterns**

The design utilizes several key principles and patterns:

* OOP Principles: The system employs encapsulation (hiding data), inheritance (extending classes), and polymorphism (allowing methods to behave differently based on the object).
* Design Patterns:
  + Singleton Pattern: Used for the VehicleRegistry to ensure a single access point for vehicle management.
  + Factory Pattern: Utilized in the VehicleFactory to streamline object creation and enhance code reusability and maintainability.

**3.2 Logical Organization**

The VMS is organized into clearly defined classes, each with specific responsibilities, promoting clarity and ease of maintenance. The main application class, Main, serves as the entry point, orchestrating user interactions and managing the flow of the application.

**4. Implementation**

**4.1 Code Functionality**

The system provides distinct functionalities for each user role:

* Admin: Can add new vehicles, remove existing ones, and view the current inventory.
* Customer: Can browse available vehicle options and initiate the purchasing process.
* Owner: Can register their vehicles for sale, ensuring a streamlined experience.

**4.2 Code Correctness**

The implementation includes input validation and error handling to ensure proper functionality. The system checks user inputs for validity, preventing incorrect data entry, and provides feedback or error messages when necessary. Each class method is rigorously tested to confirm that it performs its intended function.

**4.3 Adherence to OOP Concepts**

The implementation strictly adheres to OOP concepts:

* **Encapsulation**: Vehicle properties are kept private and accessed through public methods, ensuring data integrity.
* **Inheritance**: The Car and Bike classes inherit from the Vehicle class, promoting code reuse and reducing redundancy.
* **Polymorphism**: The displayInfo method is overridden in subclasses to provide specific details about each vehicle type, allowing for tailored output.

**Adherence to Software Design Styles:**

* **Maintainability:**
  + The code is organized into separate classes with clear responsibilities, making it easier to maintain and update. Each class handles a specific part of the functionality, following the Single Responsibility Principle.
* **Scalability:**
  + The design allows for easy addition of new vehicle types or user roles by simply creating new classes. For instance, if you wanted to add a new type of vehicle, you would create a new class that extends Vehicle and modify the VehicleFactory to accommodate it.

**5. User Interface**

**5.1 Interaction Flow**

Users interact with the system through a console interface. Upon launching the application, users are prompted to choose their role (Admin, Customer, Owner). Depending on their selection, they can perform various actions, such as viewing available vehicles or managing inventory. The interaction flow is designed to be straightforward, guiding users through available options with clear instructions.

**5.2 User Experience**

The system prioritizes user experience by providing clear prompts and feedback messages. Invalid inputs are handled gracefully, with appropriate error messages guiding users on how to correct their entries. This ensures that users can navigate the system with ease and confidence.

**6. Future Work**

**6.1 Enhancements**

* Database Integration: Implement a relational database (e.g., MySQL, PostgreSQL) to store vehicle data persistently. This will allow for better data management and retrieval.
* Graphical User Interface: Develop a GUI to enhance user interaction and provide a more intuitive experience compared to the console interface.
* Additional Features: Consider implementing features such as vehicle search and filtering options, user authentication for secure access, and reporting functionalities for admins to analyze sales and inventory trends.

**Conclusion**

The Vehicle Management System is a well-structured application that effectively facilitates vehicle transactions among different user roles. By applying OOP principles and design patterns, the system is maintainable, scalable, and easy to extend. Future enhancements could include integrating a database for persistent storage and developing a graphical user interface (GUI) to improve user interaction.