Automobile Maintenance Library Creation

Team Members:

Alisha Tamboli

Shivkanya Dhupe

Gaurav Pandit

Yuvraj Agarkar

1. Overview

- Console-based application written in **C++**.
- Manages different types of vehicles and their corresponding maintenance tasks.
- Allows users to:
  - Add, view, and remove vehicles.
  - Add, view, and remove maintenance tasks.
  - Associate tasks with compatible vehicle types.
- Supports multiple vehicle types:
  - Electric Cars
  - Gas Cars
  - Diesel Cars
  - Hybrid Cars

---

2. System Architecture

2.1 Attributes of Class

1. Automobile (Base Class)

- string id — Unique identifier
- int odometerReading — Distance travelled
- int year — Year of manufacture
- string make — Vehicle brand
- string model — Vehicle model
- CarType carType — Enum: HatchBag, Sedan, SUV

2. ElectricCar (Derived from Automobile)

- float batteryCapacity — Maximum battery capacity
- float batteryLevel — Current battery level

3. GasCar (Derived from Automobile)

- float cylinderSize — Engine cylinder size
- GasType gasType — Enum: CNG, Premium, E85

4. DieselCar (Derived from Automobile)

- float tankSize — Fuel tank size
- bool coldStartAssist — Cold start assist flag
- EmissionStandard emissionStandard — Enum: BS6, Euro6, etc.

5. HybridCar (Multiple Inheritance: GasCar + DieselCar)

- bool isSwitchable — Whether it can switch between power sources

6. MaintenanceTask

- int id — Task ID
- string taskName — Name of the maintenance task
- string description — Description of the task
- vector<string> applicableVehicles — List of compatible vehicle types (as strings)

7. MaintenanceLibrary

- vector<Automobile*> vehicle — All vehicles
- vector<MaintenanceTask*> maintenanceTasks — All maintenance task templates
- unordered_map<string, vector<MaintenanceTask>> tasks — Map of vehicle ID → assigned tasks

2.2 Methods of Class

1. Automobile (Abstract Base Class)

- Constructors
    - Automobile()
    - Automobile(string id, string make, string model, int odometerReading, int year, CarType carType)

- Getters/Setters
    - string getId() const
    - void setId(string id_)
    - int getOdometerReading() const
    - void setOdometerReading(int odometerReading_)
    - int getYear() const
    - void setYear(int year_)
    - string getMake() const
    - void setMake(const string &make_)
    - string getModel() const
    - void setModel(const string &model_)
    - string getCarType() — Returns car type as string
- Pure Virtual
    - virtual string getType() = 0

2. ElectricCar
    - ElectricCar(...) — Constructor
    - float getBatteryCapacity() const
    - void setBatteryCapacity(float batteryCapacity_)
    - float getBatteryLevel() const
    - void setBatteryLevel(float batteryLevel_)
    - string getType() — Returns "Electric"
3. GasCar
    - GasCar(...) — Constructor
    - float getCylinderSize() const
    - void setCylinderSize(float cylinderSize_)
    - string getGasType() const
    - void setGasType(const GasType &gasType_)
    - string getType() — Returns "Gas"
4. DieselCar
    - DieselCar(...) — Constructor

- float getTankSize() const
- void setTankSize(float tankSize_)
- bool getColdStartAssist() const
- void setColdStartAssist(bool coldStartAssist_)
- string getEmissionStandard() const
- void setEmissionStandard(const EmissionStandard &emissionStandard_)
- string getType() — Returns "Diesel"

5. HybridCar
   - HybridCar(...) — Constructor
   - bool getIsSwitchable() const
   - void setIsSwitchable(bool isSwitchable_)
   - string getType() — Returns "Hybrid"
6. MaintenanceTask
   - MaintenanceTask(...) — Constructor
   - int getId() const
   - void setId(int id_)
   - string getTaskName() const
   - void setTaskName(const string &taskName_)
   - string getDescription() const
   - void setDescription(const string &description_)
   - vector<string> getApplicableVehicles() const
   - void setApplicableVehicles(const vector<string> &applicableVehicles_)
7. MaintenanceLibrary
   - Constructors
     - MaintenanceLibrary()
     - MaintenanceLibrary(const vector<Automobile*>&, const unordered_map<string, vector<MaintenanceTask>>&)
   - Vehicle Management
     - void addVehicle(Automobile* automobile)
     - bool removeVehicle(string id)
     - vector<Automobile*> getVehicle() const

    - Automobile* searchVehicle(string vehicleId)
    - bool findVehicleWithId(string id)
- Task Management
    - void addTask(MaintenanceTask* t)
    - vector<MaintenanceTask*> getMaintenanceTasks()
    - MaintenanceTask* searchTask(int id)
    - bool findTaskWithId(int id)
    - void removeMaintenanceTask(MaintenanceTask* taskToRemove)
- Task Assignment
    - void addMaintenanceTask(string vehicleId, MaintenanceTask task)
    - void removeTask(string vehicleId, int taskId)
    - vector<MaintenanceTask> searchMaintenanceTask(string id)
    - MaintenanceTask* isValidTask(string id, int taskId)
    - unordered_map<string, vector<MaintenanceTask>> getTasks() const

---

3.Class Structure

The system employs an object-oriented design with the following main components:

- Automobile (Abstract Base Class):
    - Stores common vehicle attributes such as ID, make, model, odometer reading, year, and car type.
    - Declares a pure virtual method getType() to enforce type-specific behavior.
- Derived Classes:
    - ElectricCar: Adds battery-specific attributes.
    - GasCar: Adds attributes for cylinder size and gas type.
    - DieselCar: Adds attributes for tank size, cold start assist, and emission standards.
    - HybridCar: Inherits from both GasCar and DieselCar and adds a switchable engine type flag.
- MaintenanceTask:
    - Encapsulates details of maintenance tasks such as ID, name, description, and a list of compatible vehicle types.
- MaintenanceLibrary:

- Acts as the controller and data manager.
- Manages collections of vehicles and maintenance tasks.
- Provides CRUD operations for vehicles and tasks.
- Validates and assigns maintenance tasks to vehicles.

4.Data Structures & Storage

- Vehicle Store:
  std::vector<Automobile*>
  - Dynamic array of pointers to Automobile-derived instances.
  - Enables polymorphic storage and dynamic dispatch.
- Task Store:
  std::vector<MaintenanceTask*>
  - Stores task objects with metadata and applicable vehicle types.
- Assignment Map:
  std::unordered_map<std::string, std::vector<MaintenanceTask>>
  - Maps a vehicle's unique ID to a list of tasks.
  - Enables fast retrieval and update using hash-based lookup.

5.Object-Oriented Design Highlights

1.Inheritance

- Automobile is an abstract class with a pure virtual method getType().
- Specialized subclasses override behavior and add type-specific attributes.

2. Polymorphism

- Base class pointers are used to invoke virtual functions (e.g., getType()).
- Dynamic casting is used to identify concrete class types at runtime (e.g., dynamic_cast<ElectricCar*>).

3. Encapsulation

- All class members are private or protected.
- Getter/setter methods provide controlled access to internal state.

4. Reusability

- Shared logic encapsulated in the base class.
- Derived classes only override or extend where necessary.

---

6. User Guide

Getting Started

1. Clone or download the project.
2. Build the project using CMake.
    - Ensure that you have **MinGW** or another compatible C++ compiler installed.
    - Run cmake and mingw32-make to compile the code.

Running the System

1. After building, run the executable main.exe from the command line.
2. The system will simulate the car's operations based on the code in main.cpp.

---

7. Potential Improvements

- Persistence: Adding file-based or database storage to retain vehicle/task data across sessions.
- Error Handling: More comprehensive error handling and input validation.
- Testing: Incorporate unit testing for critical components.
- Memory Leak Prevention: Use of smart pointers (e.g., std::unique_ptr) instead of raw pointers to prevent memory leaks.

---

8. Conclusion

This project demonstrates a structured and well-implemented object-oriented approach to managing a vehicle maintenance system. The application can be effectively used in small auto shops or educational settings to manage vehicle records and scheduled maintenance tasks. With further enhancements, it could evolve into a fully-featured fleet maintenance solution.