CA 3: Experiential Learning

Group Members:

Sr. No.	PRN	Name of Student	Mail id
1.	23070122098	Gunveer Singh	gunveer.singh.btech2023@sitpune.edu.in
2.	23070122102	Harsh Rajput	harsh.rajput.btech2023@sitpune.edu.in
3.	23070122122	Kushagra	kushagra.kushagra.btech2023@sitpune.edu.in
4.	23070122121	Kshitij Shah	kshitij.shah.btech2023@sitpune.edu.in

Problem Statement:

Vehicle Management System using inheritance and polymorphism. Manage different vehicle types like cars and trucks, allowing operations such as starting, stopping engines, and displaying specific vehicle information.

Brief Explanation:

This project is a simple C++ implementation of a vehicle management system that simulates the functionality of a **Garage**. It allows the user to store, start, stop, and display information for various types of vehicles, including **cars**, **bikes**, and **trucks**.

Key Components:

1. **Vehicle Class** (Abstract Base Class):
    - This class serves as the base class for all vehicle types, with common properties like brand and horsepower.
    - It declares pure virtual functions for actions like starting, stopping, displaying information, and calculating fuel efficiency, which must be overridden by derived classes.
2. **Car, Bike, and Truck Classes** (Derived from Vehicle):
    - These classes inherit from the **Vehicle** class and implement the virtual functions for specific vehicle types.
    - For example, Car has an additional property for numDoors, and each derived class provides its own implementation of fuelEfficiency and engine actions.
3. **Garage Class**:
    - This class stores a collection of vehicles using a polymorphic vector (vector<Vehicle*>), allowing the addition of different vehicle types.
    - It provides methods to start, stop, and display information for all vehicles in the garage.

4. **Main Function**:
    - The main function creates various vehicle objects (e.g., a **Toyota car**, a **Honda bike**, a **Volvo truck**, and a **BMW car**) and adds them to the **Garage**.
    - It demonstrates starting all engines, displaying vehicle information, and stopping the engines.

Functionality:

- Add vehicles to the garage.
- Start and stop the engines of all vehicles.
- Display detailed information (brand, horsepower, doors for cars, fuel efficiency) for each vehicle.

This project leverages object-oriented programming concepts such as **inheritance**, **polymorphism**, and **encapsulation** to manage different vehicle types efficiently.

Code snippets:

Vehicle.h

```
1
2   #include <iostream>
3   #include <string>
4   using  namespace std;
5   #ifndef VEHICLE_H// checks the number of function defined
6   #define VEHICLE_H// defines the actual body of the function
7
8
9   class Vehicle {
10  protected:
11      string brand;
12      int horsepower;
13  public:
14      Vehicle(const std::string& b, int hp) : brand(b), horsepower(hp) {}
15
16      // Pure virtual function for polymorphism
17      virtual void startEngine() const = 0;
18      virtual void stopEngine() const = 0;
19      virtual double fuelEfficiency() const = 0;
20
21      virtual void displayInfo() const {
22          cout << "Brand: " << brand << "\nHorsepower: " << horsepower << std::endl;
23      }
24  };
25
26  #endif
```

truck.h

```cpp
#include <iostream>
#include <string>
#ifndef TRUCK_H // checks the number of function defined
#define TRUCK_H // defines the actual body of the function
using  namespace std;
#include "vehicle.h"

class Truck : public Vehicle {
private:
    double maxLoadCapacity; //In tons
public:
    Truck(const string& b, int hp, double capacity) : Vehicle(b, hp), maxLoadCapacity(capacity) {}

    void startEngine() const override {
        cout << brand << " truck's engine started with " << horsepower << " HP." << endl;
    }

    void stopEngine() const override {
        cout << brand << " truck's engine stopped." << endl;
    }

    double fuelEfficiency() const override {
        return horsepower * 0.03; // Truck take more load than bike and cars
    }

    void displayInfo() const override {
        Vehicle::displayInfo();
        cout << "Max Load Capacity: " << maxLoadCapacity << " tons" << endl;
    }
};

#endif
```

Bike.h

```cpp
#include <iostream>
#include <string>
using  namespace std;
#ifndef BIKE_H// checks the number of function defined
#define BIKE_H// defines the actual body of the function

#include "vehicle.h"

class Bike : public Vehicle {
public:
    Bike(const string& b, int hp) : Vehicle(b, hp) {}

    void startEngine() const override {
        cout << brand << " bike's engine started with " << horsepower << " HP." << endl;
    }

    void stopEngine() const override {
        cout << brand << " bike's engine stopped." << endl;
    }

    double fuelEfficiency() const override {
        return horsepower * 0.1; // Different calculation for bikes
    }
};

#endif

```

Car.h

```cpp
#include <iostream>
#include <string>
using  namespace std;
#ifndef CAR_H// checks the number of function defined
#define CAR_H// defines the actual body of the function

#include "vehicle.h"

class Car : public Vehicle {
private:
    int numDoors;
public:
    Car(const string& b, int hp, int doors) : Vehicle(b, hp), numDoors(doors) {}

    void startEngine() const override {
        cout << brand << " car's engine started with " << horsepower << " HP." << endl;
    }

    void stopEngine() const override {
        cout << brand << " car's engine stopped." << endl;
    }

    double fuelEfficiency() const override {
        return horsepower * 0.05; // Simple calculation for fuelefficiency
    }

    void displayInfo() const override {
        Vehicle::displayInfo();
        cout << "Doors: " << numDoors << endl;
    }
};

#endif
```

Garage.h

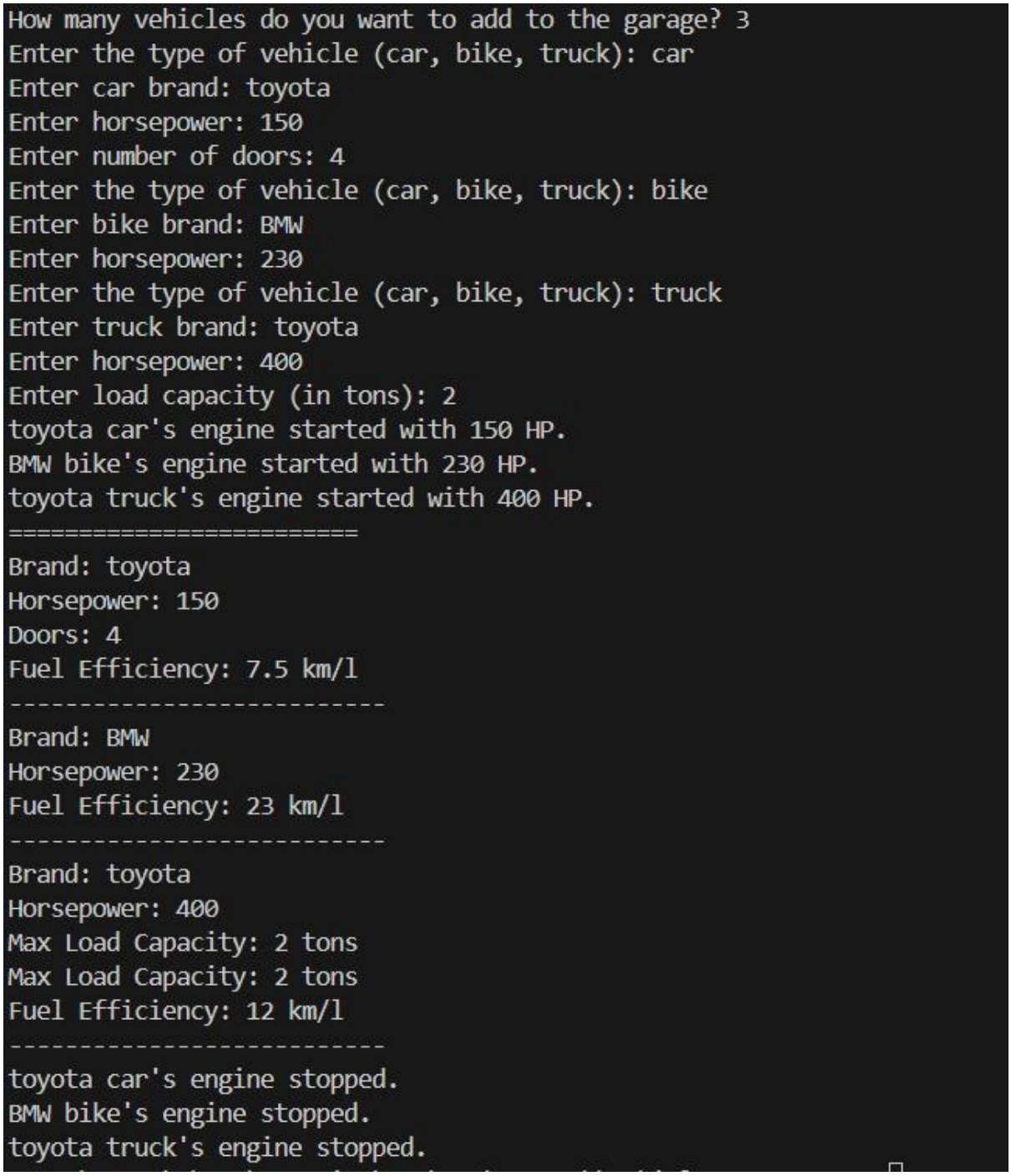
```
#ifndef GARAGE_H// checks the number of function defined
#define GARAGE_H// defines the actual body of the function
using  namespace std;

#include <iostream>
#include <string>
#include <vector>  // Needed for vector array
#include "vehicle.h"  // Assuming you have a vehicle base class with virtual methods

class Garage {
private:
    vector<Vehicle*> vehicles;  // Polymorphic storage of vehicles
public:
    // Add a vehicle to the garage
    void addVehicle(Vehicle* v) {
        vehicles.push_back(v);
    }

    // Start all engines in the garage
    void startAllEngines() const {
        for (const auto& v : vehicles) {
            v->startEngine();  // Assuming Vehicle has a virtual startEngine() method
        }
    }

    // Stop all engines in the garage
    void stopAllEngines() const {
        for (const auto& v : vehicles) {
            v->stopEngine();  // Assuming Vehicle has a virtual stopEngine() method
        }
    }

    // Display information of all vehicles
    void displayVehiclesInfo() const {
        for (const auto& v : vehicles) {
            v->displayInfo();  // Assuming Vehicle has a virtual displayInfo() method
            cout << "Fuel Efficiency: " << v->fuelEfficiency() << " km/l" << endl;  // Assuming fuelEfficiency() is a virtual method
            cout << "-------------------------" << endl;
        }
```

Poly.h

```cpp
#include <iostream>
#include <string>
#include "car.h"
#include "bike.h"
#include "truck.h"
#include "garage.h"
using namespace std;

int main() {
    Garage myGarage;
    int numVehicles;

    cout << "How many vehicles do you want to add to the garage? ";
    cin >> numVehicles;

    for (int i = 0; i < numVehicles; ++i) {
        cout << "Enter the type of vehicle (car, bike, truck): ";
        string type;
        cin >> type;

        if (type == "car") {
            string brand;
            int horsepower, numDoors;
            cout << "Enter car brand: ";
            cin >> brand;
            cout << "Enter horsepower: ";
            cin >> horsepower;
            cout << "Enter number of doors: ";
            cin >> numDoors;

            Vehicle* car = new Car(brand, horsepower, numDoors);
            myGarage.addVehicle(car);
        }
        else if (type == "bike") {
            string brand;
            int horsepower;
            cout << "Enter bike brand: ";
            cin >> brand;
            cout << "Enter horsepower: ";
```

```
            cout << "Enter horsepower: ";
            cin >> horsepower;

            Vehicle* bike = new Bike(brand, horsepower);
            myGarage.addVehicle(bike);
        }
        else if (type == "truck") {
            string brand;
            int horsepower;
            double loadCapacity;
            cout << "Enter truck brand: ";
            cin >> brand;
            cout << "Enter horsepower: ";
            cin >> horsepower;
            cout << "Enter load capacity (in tons): ";
            cin >> loadCapacity;

            Vehicle* truck = new Truck(brand, horsepower, loadCapacity);
            myGarage.addVehicle(truck);
        }
        else {
            cout << "Invalid vehicle type. Please enter car, bike, or truck." << endl;
        }
    }

    // Start all engines
    myGarage.startAllEngines();
    cout << "========================" << endl;

    // Display vehicle information
    myGarage.displayVehiclesInfo();

    // Stop all engines
    myGarage.stopAllEngines();

    return 0;
```

Input/Output:



Github repository link:

[GitHub - ZoroO911/PP_Project]