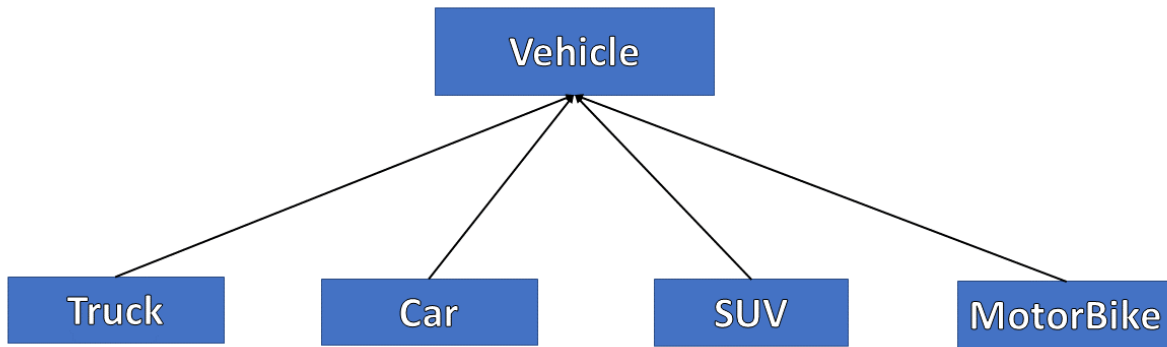


CMP_SC 3330 – Object-oriented Programming

Homework 4

Consider the following class diagram:



This assignment is a part of a vehicle showroom management system using Java Object-Oriented Programming principles. The assignment involves designing classes for different types of vehicles and implementing a singleton class for managing the vehicle information.

Class Definition:

Vehicle Class:

- Implement an **abstract** base class **Vehicle** with protected attributes/fields *brand(String)*, *make(String)*, *modelYear(long)*, *price(double)*, *color(VehicleColor)*, *fuelType(FuelType)*, *mileage(double)*, *mass(double)*, *cylinders(int)*, *gasTankCapacity(double)*, *startType(StartMechanism)*. The *color*, *fuelType*, and *startType* must be represented using an enum.
- Create subclasses of the **Vehicle** class called **Truck**, **Car**, **SUV**, and **MotorBike**, each representing a different type of vehicle. Make sure that all classes have a constructor, especially the subclasses, which must use the super keyword to initialize the attributes. Also, implement the respective copy constructors and the respective setter and getter methods.
- Implement copy constructors for each vehicle to prevent information leaks.
- Implement the **toString()** method that will display all the field information.
- The following abstract methods are provided for implementation and overriding:
 - public abstract double calculateMaintenanceCost(**double** distance); // calculates maintenance cost for a specific vehicle
 - public abstract double calculateFuelEfficiency(**double** distance, **double** fuelPrice); // calculates the engine efficiency
 - public abstract void startEngine(); // prints how the vehicle starts

FuelType Enum:

- The types are:
 - GASOLINE,

- DIESEL,
- ELECTRIC,
- HYBRID

StartMechanism Enum:

- The mechanisms are:
 - KEYSTART,
 - PUSHSTART,
 - KICKSTART,

VehicleColor Enum:

- The colors are:
 - BLACK,
 - RED,
 - BLUE,
 - BROWN,
 - WHITE,
 - YELLOW,
 - GRAY

Use the following table to implement the abstract methods for every class as described above:

Class	Method		
	calculateMaintenanceCost(double distance)	calculateFuelEfficiency(double distance, double fuelPrice)	startEngine();
Truck	$\text{maintenanceCost} = \text{distance} * \text{mass} * (\text{currentYear} - \text{modelYear}) * \text{cylinders} * 0.002$	$\text{fuelEfficiency} = \text{cylinders} * \text{gasTankCapacity} * \text{fuelPrice} / \text{distance} * 0.1$	KEYSTART
Car	$\text{maintenanceCost} = \text{distance} * \text{mass} * (\text{currentYear} - \text{modelYear}) * \text{cylinders} * 0.0005$	$\text{fuelEfficiency} = \text{cylinders} * \text{gasTankCapacity} * \text{fuelPrice} / \text{distance} * 0.003$	PUSHSTART
SUV	$\text{maintenanceCost} = \text{distance} * \text{mass} * (\text{currentYear} - \text{modelYear}) * \text{cylinders} * 0.001$	$\text{fuelEfficiency} = \text{cylinders} * \text{gasTankCapacity} * \text{fuelPrice} / \text{distance} * 0.05$	PUSHSTART
MotorBike	$\text{maintenanceCost} = \text{distance} * \text{mass} * (\text{currentYear} - \text{modelYear}) * \text{cylinders} * 0.0002$	$\text{fuelEfficiency} = \text{cylinders} * \text{gasTankCapacity} * \text{fuelPrice} / \text{distance} * 0.001$	KICKSTART

VehicleManager Class:

- Implement a class **VehicleManager** for managing the inventory of the shop.
- Similar to homework 3, the **VehicleManager** class should read the initial vehicle inventory from a CSV file (**vehicleList.csv**) during initialization, update existing items, add new items, remove items, and save the updated inventory back to the CSV file.

- The **VehicleManager** class should have a String field called *vehicleFilePath*, which is initialized to the relative path to the **vehicleList.csv** file, including the file name. However, make sure that the field does not leak any information and is closed for modification.
- An ArrayList <Vehicle> vehicleList which stores the list of all the vehicles as read from the file.

Program Requirements:

VehicleManager Methods: [Hint: in some of the methods, you may want to use instanceof or getClass()]

- `public boolean initializeStock():`
 - Reads the data from a CSV file located at *vehicleFilePath*. Initialize each of the Vehicle objects (**Hint:** Consider using the `split()` method for tokenization. Check the type of each object and instantiate the exact class. Store the objects into vehicleList).
 - Return **true** if the read file and initialization are successful.
 - Return **false** if cannot read/find the file.
- `public void displayAllCarInformation() :`
 - This will display the information, including maintenance cost, fuel efficiency, and how the vehicle starts, of all the cars present in the vehicleList.
 - If the vehicle is not found, then print an appropriate error message.
- `public void displayAllTruckInformation() :`
 - This will display the information, including maintenance cost, fuel efficiency, and how the vehicle starts, of all the trucks present in the vehicleList.
 - If the vehicle is not found, then print an appropriate error message.
- `public void displayAllSUVInformation() :`
 - This will display the information, including maintenance cost, fuel efficiency, and how the vehicle starts, of all the SUVs present in the vehicleList.
 - If the vehicle is not found, then print an appropriate error message.
- `public void displayAllMotorBikeInformation() :`
 - This will display the information, including maintenance cost, fuel efficiency, and how the vehicle starts, of all the motor bikes present in the vehicleList.
 - If the vehicle is not found, then print an appropriate error message.
- `public void displayVehicleInformation(Vehicle v) :`
 - This will display the vehicle information, including maintenance cost, fuel efficiency, and how the vehicle starts, of a **Vehicle** v which is present in the vehicleList.
 - If the vehicle is not found, then print an appropriate error message.
- `public void displayAllVehicleInformation():`
 - This will print the information, including maintenance cost, fuel efficiency, and how the vehicle starts, of all the vehicles in the vehicleList.
 - Print an appropriate message if the list is empty.
- `public boolean removeVehicle(Vehicle vehicle):`
 - Removes the given vehicle from the vehicleList.
 - Returns **true** if the removal is successful, **false** otherwise.
- `public boolean addVehicle(Vehicle vehicle):`
 - Adds the given vehicle into the vehicleList.
 - Returns **true** if the addition is successful, **false** otherwise.

- `public boolean saveVehicleList():`
 - Saves the updated vehicleList back to the CSV file located at *vehicleFilePath*.
 - Overwrites the existing file with the updated data.
 - Returns **true** if the saving is successful, **false** otherwise (file does not exist, or file empty).
- `private boolean isVehicleType(Vehicle v, Class clazz):`
 - Checks if the given vehicle is a specific type of Vehicle subclass.
 - If the given vehicle object is the object type, then return **true**, otherwise return **false**.
 - Use **instanceof** or **getClass()** to count the number.
 - Call example: `isVehicleType(vehicleObj, Truck.class);`
- `public int getNumberOfVehiclesByType(Class clazz):`
 - Returns the number of objects in the vehicle list based on the object vehicle type
 - Use the `isVehicleType(Vehicle v, Class clazz)` method.
 - Call example: `getNumberOfVehiclesByType(SUV.class);`
- `public Vehicle getVehicleWithHighestMaintenanceCost(double distance):`
 - Calculate the maintenance cost for each vehicle in the vehicle list and return the vehicle with the **highest maintenance cost**.
 - If multiple vehicles have the same maintenance cost, **randomly** return one of the vehicles (Use the Random class for random selection).
- `public Vehicle getVehicleWithLowestMaintenanceCost(double distance):`
 - Calculate the maintenance cost for each vehicle in the vehicle list and return the vehicle with the lowest maintenance cost.
 - If multiple vehicles have the same maintenance cost, **randomly** return one of the vehicles (Use the Random class for random selection).
- `public ArrayList<Vehicle> getVehicleWithHighestFuelEfficiency(double distance, double fuelPrice):`
 - Calculate the fuel efficiencies for each vehicle in the vehicle list and return the vehicle with the highest fuel efficiency.
 - If multiple vehicles have the same highest fuel efficiency, return vehicles with the same highest fuel efficiency in an ArrayList.
- `public ArrayList<Vehicle> getVehicleWithLowestFuelEfficiency(double distance, double fuelPrice):`
 - Calculate the fuel efficiencies for each vehicle in the vehicle list and return the vehicle with the lowest fuel efficiency.
 - If multiple vehicles have the same lowest fuel efficiency, return vehicles with the same lowest fuel efficiency in an ArrayList.
- `public double getAverageFuelEfficiencyOfSUVs(double distance, double fuelPrice):`
 - Calculate the average/mean of the fuel efficiency of SUVs in the vehicle list.
 - Use the `isVehicleType(Vehicle v, Class clazz)` method.

- If no SUVs exist in the list return -1.0 as an error code that indicates there is no SUVs in the list to calculate the average fuel efficiency.

Main Class:

```
public class Main {
    public static void main(String[] args) {
        // Instantiate vehicleManager, perform operations based on the requirements.
        VehicleManager vehicleManager = new VehicleManager();

        // Read vehicle data from the vehcileList.csv file and initialize objects.
        // TODO

        // Display all vehicle information.
        // TODO

        // Display all car information.
        // TODO

        // Display all motorbike information.
        // TODO
    }
}
```

Submission Guidelines:

- Each team is required to create a GitHub repository [GroupName-CS3330-Hw4] for the project.
Eg: A-CS3330-Hw4
- Submit the link to the repository.
- The repository should include all the required Java files (Main.java, Vehicle.java, Truck.java, Car.java, SUV.java, MotorBike.java, VehicleColor.java, FuelType.java, and VehicleManager.java) and any other necessary files to run the program.
- Team members are expected to contribute equally to the project.
- Each team member should make meaningful contributions, and commit messages must be descriptive and related to the changes made. Your grades will be affected by your commits.
- The GitHub repository should demonstrate good version control practices, with commits logically organized and documenting the evolution of the code.
- Make sure to include a README.md file providing clear instructions on how to run the program, any dependencies, and a brief explanation of the project. Also, include here, a detailed account of the contributions made by each team member.
- Verify that the repository is accessible and properly organized, allowing anyone to clone and run the program without additional configuration.
- Your program must use the classes with described methods, given prototypes and signatures exactly. You are allowed to implement additional helper methods and classes.
- Late submission between 0hrs < late <= 24hrs will lose half of the grade. After 24 hours, submissions will receive a grade of 0 for the assignment.
- **Not following the submission guidelines will result in a penalty on your grades.**

Note:

- Ensure that your program handles cases where the file is not found or if there are any issues during file reading.

- Make use of the concepts you've learned, such as constructors, getter/setter methods, static fields/methods, and the toString() method.
- Test your program with different scenarios, including cases where the object is not found and the update is unsuccessful.