Practice Assignment: Vehicle Information System

Objective

Create a set of Java classes to model different types of vehicles. The assignment is designed to practice key Object-Oriented Programming (OOP) concepts, including inheritance, method overriding, and function overloading.

You will be given the class names and a detailed breakdown of the required fields and methods for each one.

Class Requirements

1. `Vehicle` (Base Class)

This is the parent class from which all other vehicle types will inherit.

- **Member Fields:**
  - `String brand`
  - `int year`
  - `double fuelLeft`
  - `double fuelCapacity`
  - `double efficiency`
  - `double maxSpeed`
  - `double horsePower`
  - `int noOfPassengers`
- **Constructors:**
  - A constructor that initializes all of the member fields listed above in the same order.
- **Methods:**
  - **Getters and Setters** for all member fields. The getters should return the value of the field and the setters should update the value.
  - `double calculateRange()`: Calculates the total range of the vehicle by multiplying `fuelLeft` by `efficiency`.
  - **`void printInfo()`: (Overloaded)**

        - The first version should take no arguments and print the vehicle's `brand`, `year`, `maxSpeed`, and `horsePower`.
        - The second version should take a `String driverName` argument and print the vehicle's information along with the driver's name.

2. `Car` (Derived Class)

This class should inherit from the `Vehicle` class.

- **Member Fields:**
    - `String typeOfCar` (e.g., "Sedan", "SUV")
- **Constructors:**
    - A constructor that calls the base class constructor and initializes `typeOfCar`. The order of the fields of the constructor is mentioned in the java file.
- **Methods:**
    - **`void printInfo()`: (Overridden)** Overrides the base class method to display specific information for a car, including its `typeOfCar`, `noOfPassengers`, `fuelLeft`, and `efficiency`.
    - `void honk()`: Prints a message simulating a car honking sound.
    - **Getters and Setters** for `typeOfCar`.

3. `Bike` (Derived Class)

This class should also inherit from the `Vehicle` class.

- **Member Fields:**
    - `boolean isElectric`
- **Constructors:**
    - A constructor that calls the base class constructor and initializes `isElectric`. The order of the fields of the constructor is mentioned in the java file.
- **Methods:**
    - **`void printInfo()`: (Overridden)** Overrides the base class method to display specific information for a bike, including whether it's electric and its remaining fuel/charge and efficiency.
    - **`double calculateRange()`: (Overridden)** Overrides the base class method. If the bike is electric (`isElectric` is true), the range should be 1.5 times the standard calculation. Otherwise, it should use the standard `fuelLeft * efficiency` calculation.

   - `void honk()`: Prints a message simulating a bike ringing sound.
   - **Getters and Setters** for `isElectric`.

4. `Truck` (Derived Class)

This class should also inherit from the `Vehicle` class.

- **Member Fields:**
  - `double maximumWeightCapacity`
  - `double loadedWeight`
- **Constructors:**
  - A constructor that calls the base class constructor and initializes `maximumWeightCapacity` and `loadedWeight`. The order of the fields of the constructor is mentioned in the java file.
- **Methods:**
  - **`void printInfo()`: (Overridden)** Overrides the base class method to display specific information for a truck, including its weight capacities, `fuelLeft`, and `efficiency`.
  - `double calculateRange(double currentLoad)`: **(Special Method)** This method takes `currentLoad` as an argument and calculates the remaining range. The efficiency should be reduced by a factor proportional to the `currentLoad` relative to the `maximumWeightCapacity`. For example, `(1 - (currentLoad / maximumWeightCapacity) * 0.5)` multiplied by the base efficiency.
  - `void honk()`: Prints a message simulating a truck honking sound.
  - **Getters and Setters** for `maximumWeightCapacity` and `loadedWeight`.

How to Test Your Code

You can test your implementation using the provided `Main.java` file. This file acts as the **driver** for your program and contains a series of tests for each class.

1. **Save the Files:** Make sure all your Java class files (`Vehicle.java`, `Car.java`, `Bike.java`, `Truck.java`, and `Main.java`) are in the same directory.
2. **Compile:** Compile all the files together from your terminal: `javac *.java`
3. **Run:** Execute the main class: `java Main`

The `Main.java` file is fully commented to guide you. It creates instances of each class and calls their methods. Each test block includes a comment showing the **expected output**, so you can easily compare your results and confirm that your code is working correctly.