Inheritance, Overriding, and Abstract Class

Objective:

In this lab, you will learn about the concepts of inheritance, overriding, and abstract classes in Java. You will create a program that demonstrates these concepts through practical implementation.

Requirements:

1. Create a base class called `Vehicle` with the attributes `color`, `maxSpeed`, and `numWheels`. The class should include the following methods:
   - `displayDetails()` – Displays the vehicle's attributes.
   - `accelerate(int speedIncrease)` – Adjusts the vehicle's speed based on the parameter.
   - `brake(int speedDecrease)` – Adjusts the vehicle's speed when braking.
2. Create two subclasses `Car` and `Truck` that inherit from `Vehicle`. Each subclass should:
   - Add at least one unique attribute.
   - Override `accelerate()` and `brake()` methods to include specific messages for each type of vehicle.
3. Create an abstract class `ElectricVehicle` that extends the `Vehicle` class. The abstract class should:
   - Introduce an abstract method `chargeBattery(int percentage)` that accepts a parameter to indicate the charging percentage.
4. Create a subclass `ElectricCar` that extends `ElectricVehicle`. The class should:
   - Implement the `chargeBattery()` method.
5. Write a separate `TestVehicle` class containing the `main` method to demonstrate the classes and their functionalities.

Exercises:

1. Create an interface `FuelEfficient` with a method `calculateFuelEfficiency(double distance, double fuelConsumed)`. Implement this interface in the `Car` and `Truck` classes.
2. Create a subclass `HybridCar` that extends the `Car` class and implements both the `FuelEfficient` interface and the `chargeBattery()` method.
3. Create a subclass `Motorcycle` that extends the `Vehicle` class and overrides the `accelerate()` method to include parameter handling for speed adjustment.

Hint:

Here is a partial implementation to get you started. Complete the omitted parts as indicated in the comments:

```java
// Vehicle.java
public class Vehicle {
    private String color;
    private int maxSpeed;
    private int numWheels;
    private int currentSpeed;

    public Vehicle(String color, int maxSpeed, int numWheels) {
        this.color = color;
        this.maxSpeed = maxSpeed;
        this.numWheels = numWheels;
        this.currentSpeed = 0; // Initial speed is 0
    }

    public void displayDetails() {
        System.out.println("Color: " + color);
        System.out.println("Max Speed: " + maxSpeed + " km/h");
        System.out.println("Number of Wheels: " + numWheels);
        System.out.println("Current Speed: " + currentSpeed + " km/h");
    }

    public void accelerate(int speedIncrease) {
        // Update currentSpeed and ensure it doesn't exceed maxSpeed
        // TODO: Implement this method
    }

    public void brake(int speedDecrease) {
        // Decrease currentSpeed and ensure it doesn't go below 0
        // TODO: Implement this method
    }
}

// Car.java
public class Car extends Vehicle {
    private int numDoors;

    public Car(String color, int maxSpeed, int numWheels, int numDoors) {
        super(color, maxSpeed, numWheels);
        this.numDoors = numDoors;
    }

    @Override
    public void accelerate(int speedIncrease) {
        // TODO: Implement with a custom message for Car
    }
```

```
    @Override
    public void brake(int speedDecrease) {
        // TODO: Implement with a custom message for Car
    }
}

// ElectricVehicle.java
public abstract class ElectricVehicle extends Vehicle {
    public ElectricVehicle(String color, int maxSpeed, int numWheels) {
        super(color, maxSpeed, numWheels);
    }

    public abstract void chargeBattery(int percentage); // TODO: Implement in subcl
}

// TestVehicle.java
public class TestVehicle {
    public static void main(String[] args) {
        Vehicle car = new Car("Red", 200, 4, 4);
        car.displayDetails();
        car.accelerate(50);
        car.brake(20);

        // TODO: Add instances for Truck, ElectricCar, and demonstrate their functi
    }
}
```

Bonus Challenges:

- Enhance the `ElectricCar` class to track battery charge percentage and ensure it cannot exceed 100
- Use an array or a `List<Vehicle>` in the `main` method to showcase polymorphism by calling methods like `accelerate()` and `displayDetails()` on different types of vehicles.
- Add exception handling for invalid input values (e.g., negative speed increments or battery charge percentages).

Submission:

Submit your completed code files and a report explaining your implementation, along with screenshots of the program output.