Enunciado do Problema

Uma empresa de logística está desenvolvendo um sistema para gerenciar sua frota de veículos. Você foi contratado para implementar a hierarquia de classes desse sistema, aplicando os conceitos de **interfaces**, **abstract classes**, **late binding** e **polymorphism**.

Requisitos

1. Crie uma **interface** chamada `Fuelable` que declare os seguintes métodos:
    - `double calculateFuelConsumption(double distance)` — retorna o combustível consumido (em litros) para percorrer uma determinada distância (em km).
    - `String getFuelType()` — retorna o tipo de combustível utilizado (ex.: `"Gasoline"`, `"Diesel"`, `"Electric"`).
2. Crie uma **abstract class** chamada `Vehicle` que implemente a interface `Fuelable`. Essa classe deve conter:
    - Os atributos `protected`: `String licensePlate`, `String brand`, `double averageSpeed`.
    - Um construtor que inicialize todos os atributos.
    - Um método **abstrato** `double calculateTripTime(double distance)` — retorna o tempo estimado (em horas) para percorrer uma distância.
    - Um método concreto `void displayInfo()` que imprima as informações básicas do veículo (placa, marca, velocidade média).
    - Um método concreto `double calculateFuelCost(double distance, double fuelPrice)` que utilize o método `calculateFuelConsumption` (a ser implementado pelas subclasses) para calcular o custo total de combustível de uma viagem.
3. Crie as seguintes **subclasses concretas** de `Vehicle`:
    - **`Truck`**: consumo de combustível calculado como `distance / 3.0` (3 km por litro). O tempo de viagem deve considerar que o caminhão perde 10% da velocidade média em vias urbanas — ou seja, `distance / (averageSpeed * 0.9)`. Tipo de combustível: `"Diesel"`.

- **`DeliveryVan`**: consumo calculado como `distance / 8.0`. O tempo de viagem é `distance / averageSpeed`. Tipo de combustível: `"Gasoline"`.
- **`ElectricScooter`**: consumo calculado como `distance / 40.0` (representando kWh por km). O tempo de viagem é `distance / averageSpeed`. Tipo de combustível: `"Electric"`.

4. Crie uma classe `FleetManager` com um método `main` que:
    - Instancie um array de `Vehicle` contendo pelo menos um objeto de cada subclasse.
    - Percorra o array chamando `displayInfo()`, `calculateTripTime(100)`, `calculateFuelCost(100, 5.50)` e `getFuelType()`.
    - Demonstre, por meio de comentários no código, onde ocorre o **late binding** (dynamic dispatch).
5. Crie uma **interface** adicional chamada `Trackable` com o método `String getCurrentLocation()`. Faça com que apenas `Truck` e `DeliveryVan` implementem essa interface. No `main`, utilize `instanceof` para verificar quais veículos são rastreáveis e chame `getCurrentLocation()` apenas nesses casos.

Observações

- O uso de `@Override` é obrigatório em todos os métodos sobrescritos.
- Não utilize `if/else` ou `switch` baseados no tipo concreto do veículo para calcular consumo ou tempo — o polimorfismo deve resolver isso naturalmente.
- Comente no código os pontos que evidenciam **late binding** e **polymorphism**.