POLIVALENTE	**Escola Estadual Professor João Anastácio** **CURSO TÉCNICO EM INFORMÁTICA** **PROGRAMAÇÃO ORIENTADA A OBJETOS** **ALUNO (a):**_____ **PROFESSOR(a):** **DATA: / /** **VALOR: 05 pontos TURMA: _____ Nota: _____**

"Educação não transforma o mundo. Educação muda as pessoas. Pessoas transformam o mundo."

(Paulo Freire)

Contexto

Uma empresa de gerenciamento de frotas deseja desenvolver um sistema para monitorar seus veículos. Eles precisam de um programa capaz de representar diferentes tipos de veículos, gerenciar seus estados (ligado/desligado) e registrar sua quilometragem.

O sistema deve ser construído utilizando Programação Orientada a Objetos (POO), mas **sem o uso de herança**. Cada tipo de veículo será uma classe independente.

Tarefa

Você deve criar um programa em Kotlin que atenda aos seguintes requisitos:

1. ****Criação de Classes:****
   * Crie uma classe chamada `Carro` para representar um carro de passeio.
   * Crie uma classe chamada `Caminhao` para representar um caminhão de carga.
   * Crie uma classe chamada `Moto` para representar uma motocicleta.

2. ****Propriedades das Classes:****
   * Todas as classes de veículo devem ter as seguintes propriedades:
     * `modelo`: `String` (ex: "Ford Fiesta", "Volvo FH16", "Honda Biz").
     * `placa`: `String` (ex: "ABC-1234").
     * `quilometragem`: `Double` (inicia em 0.0).
     * `ligado`: `Boolean` (inicia em `false`).

3. ****Métodos das Classes:****
   * Cada classe de veículo deve ter os seguintes métodos:
     * `ligar()`: Altera o estado do veículo para `ligado = true`. Se o veículo já estiver ligado, deve exibir uma mensagem informando que ele já está em funcionamento.
     * `desligar()`: Altera o estado do veículo para `ligado = false`. Se o veículo já estiver desligado, deve exibir uma mensagem informando que ele já está parado.
     * `percorrerDistancia(distancia: Double)`: Recebe a distância percorrida como parâmetro. Se o veículo estiver `ligado`, adiciona a distância à sua `quilometragem`. Caso o veículo esteja `desligado`, deve exibir uma mensagem de erro informando que não é possível percorrer a distância.

* `exibirStatus()`: Imprime no console o `modelo`, a `placa`, a `quilometragem` atual e o estado (`ligado` ou `desligado`) do veículo.

4. ****Implementação na Função `main()`:****

* Dentro da função `main`, instancie um objeto de cada classe (`Carro`, `Caminhao` e `Moto`).
* Realize uma série de chamadas aos métodos para cada objeto, simulando o uso dos veículos. Por exemplo:
    * Ligue o `Carro`.
    * Tente ligar o `Carro` novamente.
    * Faça o `Carro` percorrer 50.5 km.
    * Desligue o `Carro`.
    * Tente fazer o `Carro` percorrer 10.0 km (deve falhar).
    * Exiba o status final do `Carro`.
* Repita um conjunto de ações semelhantes para os objetos `Caminhao` e `Moto` para demonstrar o funcionamento de cada um.

Critérios de Avaliação

- Implementação de Classes e Propriedades:** As classes `Carro`, `Caminhao` e `Moto` foram criadas corretamente, com as propriedades especificadas.
- Encapsulamento:** As propriedades são acessadas e modificadas apenas pelos métodos da própria classe, garantindo o encapsulamento.
- Lógica dos Métodos:** A lógica dos métodos `ligar()`, `desligar()` e `percorrerDistancia()` está correta, lidando com os diferentes estados do veículo.
- Uso de Objetos:** A função `main` demonstra a correta criação de objetos e a chamada dos seus métodos.
- Organização e Clareza do Código:** O código é legível, bem formatado e segue as convenções de Kotlin.