Escola Estadual Professor João Anastácio
CURSO TÉCNICO EM INFORMÁTICA
AVALIAÇÃO DO 4º BIMESTRE DA DISCIPLINA
Programação Orientada a Objetos - Java

ALUNO (a):_____
PROFESSOR(a):
DATA: / /
VALOR: 07 pontos **TURMA: _____** **Nota: _____**

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

(Paulo Freire)

O objetivo é modelar um sistema básico de gestão de **Carros de Passeio**, explorando **Herança**, **Polimorfismo**, **Interfaces** e **Sobrecarga (Overloading)**.

1. Definição da Estrutura Base (Interface e Classe Abstrata)

1.1. Interface `Servicavel` (Contrato)

Crie uma interface chamada `Servicavel` que define o seguinte contrato:

- **Método 1:** `realizarManutencao()`: Não recebe parâmetros e retorna uma `String` indicando que a manutenção geral foi feita.
- **Método 2 (Sobrecarga):** `realizarManutencao(int meses)`: Recebe um inteiro (`meses`) e retorna uma `String` indicando que a manutenção foi programada para aquele número de meses.

1.2. Classe Abstrata `Veiculo` (Superclasse)

- **Tipo:** Declare esta classe como **abstrata**.
- **Atributos:**
  - `marca` (String) - `private`.
  - `modelo` (String) - `private`.
  - `ano` (int) - `private`.
  - `cor` (String) - `protected`.
- **Construtores (Sobrecarga):**
  - **Construtor 1:** Defina um construtor que inicialize **apenas** `marca`, `modelo` e `ano`. A `cor` deve ser inicializada como `"Indefinida"`.
  - **Construtor 2:** Defina um construtor que inicialize **todos** os quatro atributos.
- **Métodos:**
  - Implemente métodos **`getter`** para `marca`, `modelo`, `ano` e `cor`.
  - **Método Abstrato:** Declare um método abstrato chamado `calcularTarifaDiaria()` que retorna um `double`.

- **Método Concreto:** Crie o método `detalhesBase()` que retorna uma `String` formatada com `marca`, `modelo`, `ano` e `cor`.

2. Implementação da Subclasse e Polimorfismo

2.1. Subclasse Única: CarroPasseio

- Deve herdar de Veiculo e **implementar a interface Servicavel**.
- **Atributo Específico:** valorDiariaBase (double) - private.
- **Construtor:** Utilize super() para inicializar os atributos da superclasse. O construtor de CarroPasseio deve obrigar a passagem de valorDiariaBase e dos atributos base.
- **Sobrescrita (@Override):**
- Implemente calcularTarifaDiaria(): Retorna o valorDiariaBase acrescido de uma taxa de serviço fixa de 15.00 (Retorno: valorDiariaBase + 15.00).
- **Implementação da Interface (@Override):**
- Implemente os dois métodos realizarManutencao() da interface Servicavel.

3. Aplicação e Demonstração do Polimorfismo Dinâmico e Sobrecarga

3.1. Classe `AppVeiculos` (Classe Principal)

Crie uma classe principal que contenha:

- **Método Estático:** Crie um método **estático** chamado `processarVeiculos(List<Veiculo> lista)` que:
    - Itere sobre a lista de `Veiculos`.
    - Para cada veículo, imprima seus detalhes (`detalhesBase()`).
    - Calcule e imprima a tarifa (`calcularTarifaDiaria()`).
    - **Utilize o Cast Explícito:** Faça um *cast* do `Veiculo` atual para a interface `Servicavel` e chame **as duas versões** do método `realizarManutencao()` (uma sem parâmetro e outra com um valor de meses, por exemplo, 6). Imprima o resultado das duas chamadas.
- **Método `main`:**
    - Instancie, no mínimo, quatro `CarroPasseio`:
        - Dois usando o **Construtor 1** da Superclasse (com cor `"Indefinida"`).
        - Dois usando o **Construtor 2** da Superclasse (com cor definida).
    - Crie uma `List<Veiculo>` e adicione todos os objetos criados.
    - Chame o método estático `processarVeiculos()` para demonstrar o polimorfismo dinâmico.