1. Você está desenvolvendo um sistema para uma escola de artes. Nessa escola, há dois tipos de alunos: AlunoDeMusica e AlunoDePintura. Todos os alunos possuem nome, idade e uma matrícula. Porém:AlunoDeMusica: Além das características básicas, eles têm um instrumento que tocam e precisam praticar esse instrumento.

AlunoDePintura: Além das características básicas, eles têm um estilo de pintura que preferem (por exemplo, "abstrato", "realismo", etc.) e precisam pintar quadros nesse estilo.Estruture as classes para esta situação. Estrutura Proposta:

- Classe Aluno (Classe Base):
  - Atributos:
    - String nome
    - int idade
    - String matricula
  - Construtor:
    - Aluno(String nome, int idade, String matricula)
  - Métodos:
    - void estudar(): Imprime uma mensagem genérica, como "O aluno está estudando."
    - String getNome(): Retorna o nome do aluno.
    - int getIdade(): Retorna a idade do aluno.
    - String getMatricula(): Retorna a matrícula do aluno.
- Classe AlunoDeMusica (Derivada de Aluno):
  - Atributos:
    - String instrumento
  - Construtor:
    - AlunoDeMusica(String nome, int idade, String matricula, String instrumento): Chama o construtor da superclasse e inicializa o atributo instrumento.
  - Métodos:
    - void praticar(): Imprime uma mensagem específica, como "O aluno está praticando [instrumento]."
    - String getInstrumento(): Retorna o instrumento que o aluno toca.
- Classe AlunoDePintura (Derivada de Aluno):
  - Atributos:
    - String estiloPintura
  - Construtor:
    - AlunoDePintura(String nome, int idade, String matricula, String estiloPintura): Chama o construtor da superclasse e inicializa o atributo estiloPintura.
  - Métodos:
    - void pintarQuadro(): Imprime uma mensagem específica, como "O aluno está pintando um quadro no estilo [estiloPintura]."
    - String getEstiloPintura(): Retorna o estilo de pintura do aluno.

Com esta estrutura proposta, a herança é utilizada para que ambos AlunoDeMusica e AlunoDePintura tenham atributos e métodos comuns de Aluno, mas também possuam suas características e comportamentos únicos.

2. Crie um sistema simples de gerenciamento para uma oficina de veículos. Existem dois tipos de veículos em nossa oficina: Carros e Motos. Todos os veículos possuem um modelo e uma quilometragem. No entanto, eles têm métodos diferentes para calcular o custo de serviço, que é baseado na quilometragem: Para Carros: o custo é *quilometragem x 0.05.*

Para Motos: o custo é *quilometragem x 0.03.*
Estruture as classes para esta situação e implemente um método para calcular o custo de serviço para cada tipo de veículo. Estruture o problema da seguinte forma:Classe Veiculo (Classe Base):

- Atributos:
  - String modelo
  - double quilometragem
- Construtor:
  - Veiculo(String modelo, double quilometragem)
- Métodos:
  - double calcularCustoServico(): Deve ser abstrato neste nível.
  - String getModelo(): Retorna o modelo do veículo.
  - double getQuilometragem(): Retorna a quilometragem do veículo.

Classe Carro (Derivada de Veiculo):

- Construtor:
  - Carro(String modelo, double quilometragem): Chama o construtor da superclasse.
- Métodos:
  - double calcularCustoServico(): Sobrescreve o método da classe Veiculo para calcular o custo baseado na quilometragem do carro.

Classe Moto (Derivada de Veiculo):

- Construtor:
  - Moto(String modelo, double quilometragem): Chama o construtor da superclasse.
- Métodos:
- double calcularCustoServico(): Sobrescreve o método da classe Veiculo para calcular o custo baseado na quilometragem da moto.

Desafio Adicional (para focar mais no polimorfismo): Após implementar as classes acima, crie uma lista de veículos (utilizando, por exemplo, ArrayList<Veiculo>) e percorra essa lista, chamando o método calcularCustoServico() para cada veículo. Observe como, graças ao polimorfismo, o método correto é chamado para cada tipo de veículo, mesmo que todos os itens da lista sejam do tipo Veiculo.

3. Uma empresa de tecnologia quer gerenciar seus funcionários de forma eficiente. Ela possui dois tipos principais de funcionários: Desenvolvedor e Gerente. Todos os funcionários possuem um nome, ID e salário base. Porém, há diferenças específicas entre eles:

- Desenvolvedor: Além dos atributos padrão, possui uma linguagem de programação principal.

- Gerente: Além dos atributos padrão, gerencia um departamento específico da empresa.

Seu Desafio:

Implemente a classe base Funcionario e suas classes derivadas Desenvolvedor e Gerente. Cada tipo de funcionário deve ter um método chamado apresentar() que retorna uma breve descrição sobre eles. Por exemplo, o desenvolvedor pode retornar algo como "Sou o desenvolvedor [nome], especialista em [linguagem].", enquanto o gerente pode retornar "Sou o gerente [nome] responsável pelo departamento de [departamento].".

Na classe Main, crie uma lista de funcionários e adicione exemplos de ambos os tipos. Em seguida, crie um método estático chamado listarFuncionarios() que aceita essa lista como parâmetro. Esse método deve iterar sobre a lista e imprimir a apresentação de cada funcionário. Utilize o conceito de polimorfismo para garantir que o método correto apresentar() seja chamado para cada tipo de funcionário. Dica: Utilize estruturas como ArrayList<Funcionario> para armazenar os funcionários e demonstrar o polimorfismo em ação.