Questão 2

1. O que é herança em Programação Orientada a Objetos (POO) e qual seu principal objetivo?

Herança é um mecanismo da Programação Orientada a Objetos que permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse). O principal objetivo é **promover o reuso de código**, facilitando a **organização, manutenção e expansão** de sistemas, além de permitir a criação de hierarquias lógicas entre os objetos.

2. Qual é a diferença entre uma superclasse e uma subclasse? Dê um exemplo de cada.

A **superclasse** é a classe genérica que contém os atributos e comportamentos comuns. Já a **subclasse** é uma especialização da superclasse e pode herdar seus atributos e métodos, além de definir os seus próprios.

Exemplo:

Superclasse: `FuncionarioTI`
Subclasse: `Desenvolvedor` (especialização de FuncionarioTI)

3. O que significa a afirmação "todo Estudante é uma Pessoa, mas nem toda Pessoa é um Estudante"?

Significa que a **classe Estudante herda da classe Pessoa**, ou seja, **Estudante é uma especialização de Pessoa**. Um Estudante possui todas as características de uma Pessoa, mas também tem características específicas. Já uma Pessoa pode não ser um Estudante — pode ser um Professor, um Funcionário, etc.

4. Quais são as vantagens do uso da herança no desenvolvimento de software orientado a objetos?

As principais vantagens são:**Reutilização de código**: evita repetição de atributos e métodos, **Organização lógica**: estrutura hierárquica facilita o entendimento do sistema, **Facilidade de manutenção**: mudanças na superclasse refletem nas subclasses e **Extensibilidade**: permite ampliar funcionalidades criando subclasses específicas.

5. Qual o símbolo e a direção da seta usada para representar herança em diagramas de classes UML?

Usa-se uma **seta com linha sólida e um triângulo branco (vazio)** na ponta.
A seta **vai da subclasse para a superclasse**, indicando que a subclasse herda da superclasse.

6. Para que serve a palavra-chave `super` em Java? Cite dois contextos diferentes em que ela pode ser usada.

A palavra-chave `super` serve para fazer referência à superclasse a partir da subclasse. Dois contextos comuns:

Chamar o construtor da superclasse:
`super(nome, idade);` usado no construtor da subclasse.

Acessar métodos da superclasse (quando estão sobrescritos na subclasse):
`super.exibirDados();`

7. Por que é mais vantajoso centralizar atributos comuns na classe `Professor` ao invés de declará-los separadamente nas subclasses `ProfHorista` e `ProfDE`?

Porque ao centralizar os atributos comuns (como `nome`, `matrícula`, etc.) na superclasse `Professor`, **evitamos duplicação de código**, facilitamos a manutenção e garantimos **consistência** entre as subclasses. Isso também segue o princípio DRY (Don't Repeat Yourself), que é muito importante em engenharia de software.

8. Suponha que você esteja modelando um sistema de transporte. Como você organizaria uma hierarquia de classes para representar Transporte, TransporteTerrestre, TransporteAereo, Carro, Avião e Helicóptero? Qual o papel da herança nessa modelagem?

Eu criaria a seguinte hierarquia:

- `Transporte` (classe base)
    - `TransporteTerrestre`
        - `Carro`
    - `TransporteAereo`
        - `Avião`
        - `Helicóptero`

A herança é importante porque **permite definir os comportamentos e atributos comuns** na classe `Transporte` (como velocidade, capacidade, combustível), e depois **especializar** nas subclasses. Isso torna o sistema mais modular, reutilizável e fácil de entender.

9. O que acontece se, em uma subclasse, você não chamar explicitamente o construtor da superclasse usando `super()`? Em que situação isso pode gerar erro?

Se a superclasse tiver um **construtor padrão (sem parâmetros)**, o Java insere automaticamente `super()` na subclasse. Porém, se a superclasse **só tiver construtores com parâmetros**, e a subclasse **não chamar `super()` com os argumentos corretos**, ocorre um **erro de compilação**.

Exemplo de erro:

```
public class Pessoa {

    public Pessoa(String nome) { ... }

}

public class Estudante extends Pessoa {

    public Estudante() {

}

}
```