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

R: Herança é a capacidade da subclasse (filha) herdar de uma superclasse(mãe) tanto métodos quando atributos, o principal objetivo é o reuso do código, e permitir que uma classe reutilize tanto métodos quanto atributos sem causar duplicação.

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

R: Superclasse é a classe "mãe" que fornece atributos e métodos que podem ser herdados e subclasse são as que herdam esses atributos e métodos e são denominadas "filhas". EX: uma subclasse seria Funcionário, enquanto duas possíveis subclasses seriam "gari" e "auxiliar administrativo"

3- Dada a seguinte hierarquia UML: Pessoa → Estudante, o que significa afirmar que "todo Estudante é uma Pessoa, mas nem toda Pessoa é um Estudante"?

R: A frase descreve o conceito de herança, pois a subclasse estudante herda tudo da classe pessoa, já a superclasse pessoa é mais genérica pode representar várias pessoas: professores, funcionários etc.

4- Quais são as vantagens do uso da herança no desenvolvimento de software orientado a objetos? Por que uma subclasse não consegue acessar diretamente atributos declarados como private na superclasse? Como isso pode ser resolvido?

R: São várias as vantagens no uso da herança, dentre elas temos: Reutilização do código, organização e lógica do código, facilidade de extensão e etc. Uma subclasse não consegue acessar diretamente atributos private da superclasse porque esses atributos são **restritos da** superclasse. Esse é um dos princípios da encapsulamento. Podemos resolver isso usando os getters e seterrs que permite ler ou modificar os atributos privados.

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

R: a herança é representada por uma seta com triângulo branco vazio apontando da subclasse para a superclasse**.**

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

R: É usada para acessar membros da classe mãe através da subclasse, também chama o construtor da superclasse. super(args): Inicializar atributos herdados. super.metodo() ou super.atributo: reaproveitar comportamentos da superclasse.

7- Um sistema define as classes Professor, ProfHorista e ProfDE. Por que é mais vantajoso centralizar atributos comuns na classe Professor ao invés de declará-los separadamente nas subclasses?

R: É mais vantajoso centralizar os atributos comuns na classe Professor porque assim evitamos repetição de código nas subclasses. Isso torna o sistema mais organizado, fácil de reutilizar**.** Se for preciso mudar algo, fazemos isso só na superclasse, **e** todas as subclasses já atualizaram

automaticamente. Além disso, fica mais claro que todos são tipos de professor e compartilham as mesmas características básicas.

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?

R: Eu organizaria Transporte como a classe principal, e depois criaria TransporteTerrestre e TransporteAereo como subclasses. Carro herdaria de TransporteTerrestre, e Avião e Helicóptero herdariam de TransporteAereo. A herança ajuda a evitar repetição de código e deixa o sistema mais organizado, porque coisas em comum ficam na classe Transporte, e os detalhes específicos ficam nas subclasses.

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?

R: Se você não usar o super() na subclasse, o Java tenta chamar o construtor padrão da superclasse automaticamente. Se a superclasse não tiver esse construtor sem parâmetros, vai dar erro e o programa não vai compilar. Então, quando a superclasse só tem construtor com parâmetros, você precisa chamar o super() passando os argumentos certos para evitar erro.