1. O que significa Programação Orientada a Objetos (POO)?

POO é uma **abordagem de programação** onde a gente tenta **modelar o mundo real** no código. A ideia central é organizar o software em **Objetos**, que são entidades que agrupam dados (características) e comportamento (ações). Isso deixa o código mais organizado, reutilizável e fácil de dar manutenção.

---

2. Quais são as vantagens de usar TypeScript em vez de JavaScript?

1. **Tipagem Estática e Segurança:** Essa é a maior vantagem! O TS permite definir os tipos de dados (number, string, etc.) e **encontra erros** (tipo tentar somar um texto com um número) **antes** mesmo do código rodar (em tempo de compilação). Isso evita muitos bugs em produção.
2. **Melhor *Developer Experience* (DX) e Legibilidade:** Com os tipos definidos, o VS Code consegue te dar um **autocompletar** muito mais inteligente (IntelliSense). Além disso, o código fica muito mais claro, pois você sabe exatamente o que esperar de cada função ou objeto.

---

3. Defina o que são: classe, objeto, atributo e método.

Conceito	Definição (Aplicações Práticas)
Classe	É o **molde** ou a planta de construção. Define a estrutura (o que o objeto vai ter).
Objeto	É a **instância** concreta da classe. É a "coisa" criada a partir do molde.
Atributo	São as **características** do objeto (os dados/variáveis). Ex: cor, nome.
Método	São as **ações** que o objeto pode realizar (as funções). Ex: salvar(), acelerar().

---

4. Para que serve o modificador de acesso private?

O private serve para indicar que um atributo ou método só pode ser acessado e alterado de **dentro** da própria classe.

Ele é importante para **Encapsulamento** porque **esconde a complexidade interna** e **protege os dados**. Ele garante que o estado do objeto só seja mudado através de métodos controlados (os **métodos públicos**), evitando alterações diretas e potencialmente inválidas vindas de fora.

---

5. O que é uma classe abstrata e em que situação ela deve ser usada?

Uma **classe abstrata** é uma classe que **não pode ser instanciada** (você nunca cria um objeto dela diretamente).

Ela serve como uma **base comum** ou **contrato** para outras classes herdarem. Você deve usá-la quando quiser que várias classes compartilhem uma estrutura e métodos básicos, mas obrigando que as classes filhas implementem a sua própria versão de certos métodos (os métodos abstratos).

---

6. O que é uma interface em TypeScript e qual a sua utilidade?

Uma **interface** em TypeScript é um **contrato de tipagem** puro. Ela não gera código JavaScript no final; ela existe **apenas em tempo de compilação** para:

1. **Descrever a forma** que um objeto, função ou classe deve ter.
2. **Garantir** que classes e objetos sigam um padrão específico, aumentando a **segurança de tipos** em toda a aplicação.

---

7. Explique a diferença entre herança e composição.

- **Herança (É UM):** É quando uma classe filha **herda** as características e métodos da classe pai. É a relação "**É UM**". Ex: Gerente É UM Funcionario. É bom para reuso de código, mas pode levar a acoplamento forte.
- **Composição (TEM UM):** É quando uma classe **contém** outras classes como propriedades para usar sua funcionalidade. É a relação "**TEM UM**". Ex: Um Carro TEM UM Motor. É a preferência moderna, pois oferece mais flexibilidade.

---

8. O que é polimorfismo?

Polimorfismo significa literalmente "**muitas formas**".

É a capacidade de diferentes objetos responderem ao **mesmo método** de maneiras diferentes.

- **Exemplo:** Se você tem uma função desenhar() em objetos diferentes, quando você chama quadrado.desenhar(), ele desenha um quadrado; quando chama circulo.desenhar(), ele desenha um círculo. A chamada (desenhar()) é a mesma, mas a ação interna é polimórfica (muda conforme o objeto).

---

9. O que são exceções? Como o TypeScript trata erros usando try / catch?

Exceções são erros que **interrompem** o fluxo normal da execução do programa (tipo dividir por zero ou tentar acessar uma variável null).

O bloco try / catch é o mecanismo que o TS (e JS) usam para **tratar** essas exceções de forma segura:

1. O código potencialmente problemático fica no bloco **try**.

2. Se uma exceção for lançada (usando throw), o fluxo normal é interrompido e o código pula imediatamente para o bloco **catch**, onde podemos lidar com o erro (ex: mostrar uma mensagem amigável ao usuário) sem deixar o programa quebrar.

---

10. Por que testes unitários são importantes no desenvolvimento orientado a objetos?

O principal motivo é: **Garantia de Qualidade e Confiança**.

Testes unitários garantem que **cada unidade** (cada classe, cada método) da sua POO funcione **isoladamente** e exatamente como foi projetado. Isso dá à equipe **confiança** total para fazer refatoração, otimizações ou adicionar novas funcionalidades, sabendo que, se os testes passarem, o código que já funcionava não foi quebrado (evitando **regressões**).