📘 Atividade Final – TypeScript e Orientação a Objetos

Objetivo da Atividade

Avaliar sua compreensão dos principais conceitos de **TypeScript**, **Programação Orientada a Objetos (POO)** e **testes unitários**, utilizando tanto respostas teóricas quanto código prático.

Sempre que o exercício solicitar código, você deverá enviá-lo em arquivos separados, que podem ser compactados em `.zip` ou `.rar`.

✔️ Instruções Gerais

- Responda às perguntas teóricas em um único arquivo `.pdf`, `.docx` ou `.txt`.
- Exercícios que envolvem código TypeScript devem ser enviados em arquivos separados, ex.: `exercicio1.ts`, `carro.ts`, `testePessoa.ts` etc.
- Todos os arquivos podem ser enviados em um único `.zip` ou `.rar`.
- Seja claro e objetivo nas respostas.
- Não é necessário incluir critérios de nota.

📚 PARTE 1 — Questões Teóricas

Responda no arquivo teórico da atividade.

1. Explique com suas próprias palavras o que significa **Programação Orientada a Objetos (POO)**.

2. Quais são as vantagens de usar **TypeScript** em vez de JavaScript? Cite pelo menos duas.
3. Defina o que são: **classe**, **objeto**, **atributo** e **método**.
4. Para que serve o modificador de acesso `private`? Por que ele é importante para **encapsulamento**?
5. O que é uma **classe abstrata** e em que situação ela deve ser usada?
6. O que é uma **interface** em TypeScript e qual a sua utilidade?
7. Explique a diferença entre **herança** e **composição**.
8. O que é **polimorfismo**? Dê um exemplo simples em palavras.
9. O que são **exceções**? Como o TypeScript trata erros usando `try` / `catch`?
10. Por que **testes unitários** são importantes no desenvolvimento orientado a objetos? Cite no mínimo 1 motivo.

💻 PARTE 2 — Exercícios Práticos (TypeScript)

Os exercícios abaixo devem ser entregues em arquivos `.ts` separados. Você pode enviar tudo dentro de um `.zip` ou `.rar`.

Inclua comentários no código explicando trechos importantes.

🔹 Exercício 1 — Criação básica de classe

Crie um arquivo `exercicio1.ts` contendo:

- Uma classe `Pessoa` com:
  - atributos: `nome` (`string`) e `idade` (`number`)
  - um método `apresentar()` que retorna a frase: "Olá! Meu nome é ___ e eu tenho ___ anos."
- Crie dois objetos da classe e chame o método `apresentar()`.

🔹 Exercício 2 — Construtor e encapsulamento

Crie um arquivo `exercicio2.ts` contendo:

- Uma classe `ContaBancaria` com:
  - atributo privado `saldo` (`number`)
  - métodos públicos:
    - `depositar(valor: number)`
    - `sacar(valor: number)`
    - `verSaldo()`
- Faça tratamento para impedir saldo negativo (retornando mensagem de erro).
- Crie uma conta, faça depósitos/saques e exiba o saldo.

🔹 Exercício 3 — Herança

Crie:

- Uma classe base `Animal` com método `emitirSom()`
- Duas subclasses:
  - `Cachorro` (retorna "Au au!")
  - `Gato` (retorna "Miau!")
- Crie um arquivo `exercicio3.ts` criando objetos das subclasses e chamando seus sons.

🔹 Exercício 4 — Classe Abstrata

Crie:

- Uma classe abstrata `Veiculo` com método abstrato `mover()`
- Classes concretas:
  - `Carro` (mostra "O carro está se movendo…")
  - `Bicicleta` (mostra "A bicicleta está se movendo…")
- Crie um arquivo `exercicio4.ts` demonstrando seu funcionamento.

🔹 Exercício 5 — Interface

Crie um arquivo `exercicio5.ts` contendo:

- Uma interface `FormaGeometrica` com método `calcularArea(): number`
- Classes `Quadrado` e `Circulo` implementando a interface.
- Crie objetos das classes e exiba as áreas calculadas.

🔹 Exercício 6 — Composição

Crie um arquivo `exercicio6.ts` com:

- Uma classe `Endereco`
- Uma classe `Cliente` contendo um objeto do tipo `Endereco`
- Mostre que um cliente possui um endereço, exibindo as informações no console.

🔹 Exercício 7 — Polimorfismo

Crie um arquivo `exercicio7.ts`:

- Utilize as classes de animais criadas anteriormente.
- Crie uma função `emitirSomDoAnimal(animal: Animal)` que chame `animal.emitirSom()`
- Teste passando diferentes subclasses.

🔹 Exercício 8 — Tratamento de Erros

Crie um arquivo `exercicio8.ts`:

- Uma função `dividir(a: number, b: number)` que:
  - lança um erro personalizado se `b === 0`
  - utiliza `try` / `catch` ao chamá-la

🔹 Exercício 9 — Teste Unitário Simples

Crie um arquivo `exercicio9.ts` simulando um teste simples (sem necessidade de framework):

- Crie uma função `somar(a, b)`
- Crie uma função `testeSomar()` que:

- chama a função `somar`
- verifica se o retorno está correto usando um `if`
- exibe "Teste passou" ou "Teste falhou"

🔹 Exercício 10 — Teste de método com objeto

Crie um arquivo `exercicio10.ts`:

- Crie uma classe `Calculadora` com método `multiplicar(a, b)`
- Escreva um teste simples nos mesmos moldes do exercício anterior para validar esse método.

✔️ ENTREGA

Envie tudo como preferir:

- 📄 Arquivo único com as respostas teóricas: `atividade_teorica.pdf`, `.docx` ou `.txt`
- 📄 Arquivos separados `.ts` com os exercícios práticos (por exemplo: `exercicio1.ts`, `exercicio2.ts`, ...)
- 📦 Tudo pode ser compactado em um único `.zip` ou `.rar`