■ Roteiro de Ensino: Paradigma de Orientação a Objetos com TypeScript

Este roteiro foi desenvolvido para o ensino do paradigma de orientação a objetos (POO) utilizando a linguagem TypeScript. O material está dividido em quatro aulas práticas, com explicações teóricas, exemplos comentados e desafios.

■ Aula 1 – Introdução à POO e Classes

Objetivo: Compreender os pilares da POO e criar as primeiras classes.
Tópicos: O que é POO, classe e objeto, propriedades e métodos, construtores.

```
class Produto {
  constructor(
    public nome: string,
    public preco: number,
    public estoque: number
  ) {}

  exibirInfo() {
    console.log(`${this.nome} - R$${this.preco}`);
  }
}

const camisa = new Produto("Camisa Polo", 79.9, 50);
camisa.exibirInfo();
```

Desafio: Crie uma classe *Aluno* com propriedades *nome*, *idade* e *curso*. Adicione um método *apresentar()*.

■ Aula 2 – Encapsulamento e Modificadores de Acesso

Objetivo: Entender como proteger dados e controlar acesso a atributos.
Tópicos: public, private, protected, getters e setters.

```
class ContaBancaria {
  private saldo: number = 0;

  depositar(valor: number) {
    if (valor > 0) this.saldo += valor;
  }

  sacar(valor: number) {
    if (valor <= this.saldo) this.saldo -= valor;
  }

  getSaldo() {
    return this.saldo;
  }
```

```
}

const conta = new ContaBancaria();
conta.depositar(500);
conta.sacar(100);
console.log("Saldo atual:", conta.getSaldo());
```

Desafio: Implemente uma classe *Carro* com métodos *acelerar()* e *frear()*, mantendo a velocidade como atributo privado.

■ Aula 3 – Herança e Polimorfismo

Objetivo: Reutilizar e especializar comportamentos entre classes.
Tópicos: Classe base e derivada, extends, super, sobrescrita de métodos.

```
class Funcionario {
  constructor(public nome: string, public salario: number) {}

  calcularBonus() {
    return this.salario * 0.1;
  }
}

class Gerente extends Funcionario {
  calcularBonus() {
    return this.salario * 0.2 + 1000;
  }
}

const joao = new Funcionario("João", 3000);
const maria = new Gerente("Maria", 5000);

console.log(joao.calcularBonus());
console.log(maria.calcularBonus());
```

Desafio: Crie uma hierarquia *Veiculo* → *Moto* e *Carro*. Ambos devem sobrescrever o método *exibirTipo()*.

■ Aula 4 – Abstração e Interfaces

Objetivo: Trabalhar com contratos e classes abstratas.
Tópicos: Classes abstratas, interfaces, uso prático em sistemas.

```
interface Entregavel {
  entregar(): void;
}

class Pedido implements Entregavel {
  constructor(public numero: number, public endereco: string) {}

  entregar() {
    console.log(`Pedido ${this.numero} entregue em ${this.endereco}.`);
```

```
  }
}

const pedido = new Pedido(123, "Rua das Flores, 45");
pedido.entregar();
```

Desafio: Crie uma interface *Autenticavel* com o método *autenticar(senha: string): boolean* e implemente-a em uma classe *Usuario*.

■ Mini Projeto Integrador: Sistema de Delivery Simplificado

Implemente um pequeno sistema de delivery que inclua as classes **Cliente**, **Restaurante** e **Pedido**. Utilize os conceitos aprendidos: Herança: Cliente VIP Encapsulamento: controle de saldo e status do pedido Interface: *Entregavel* Simule o fluxo completo de um pedido no console.