7. Orientação a Objetos em TypeScript

TypeScript é uma linguagem que suporta os conceitos fundamentais de orientação a objetos como classes, interfaces, herança, e polimorfismo. Vamos detalhar como esses conceitos são implementados e utilizados em TypeScript para ajudar a organizar e estruturar melhor seu código.

Classes em TypeScript

1. **Definindo e Instanciando Classes**
    - **Classes são usadas como moldes para criar objetos. Aqui está um exemplo de como definir e instanciar uma classe.**

```
class Pessoa {
    nome: string;
    idade: number;

    constructor(nome: string, idade: number) {
        this.nome = nome;
        this.idade = idade;
    }

    descrever(): string {
        return `Nome: ${this.nome}, Idade: ${this.idade}`;
    }
}

// Instanciando a classe Pessoa
let pessoa1 = new Pessoa("Maria", 30);
console.log(pessoa1.descrever());
```

2. **Herança e Instanciação**
    - **Demonstração de como uma classe pode herdar de outra e como instanciar a classe derivada.**

```
class Empregado extends Pessoa {
    salario: number;

    constructor(nome: string, idade: number, salario: number) {
        super(nome, idade); // Chama o construtor da classe base
        this.salario = salario;
    }

    descrever(): string {
        return `${super.descrever()}, Salário: R$${this.salario}`;
    }
}

// Instanciando a classe Empregado
let empregado1 = new Empregado("João", 45, 5000);
console.log(empregado1.descrever());
```

Encapsulamento e Modificadores de Acesso

- **Classes com Encapsulamento**
    - **Exemplo de como encapsular propriedades e permitir acesso controlado através de métodos.**

```
class Conta {
    private saldo: number;

    constructor(saldoInicial: number) {
        this.saldo = saldoInicial;
    }

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

    obterSaldo(): number {
        return this.saldo;
    }

    protected calcularJuros(taxa: number): void {
        this.saldo += this.saldo * taxa;
    }
}

// Instanciando e usando a classe Conta
let minhaConta = new Conta(1000);
minhaConta.depositar(500);
console.log(`Saldo atual: R$${minhaConta.obterSaldo()}`);
```

Polimorfismo

- **Polimorfismo com Métodos Sobrescritos**
    - **Como classes derivadas podem sobrescrever métodos de suas classes base.**

```
class ContaPoupanca extends Conta {
    calcularJuros(): void {
        super.calcularJuros(0.02); // Aplica uma taxa de juros específica
    }
}

// Instanciando e usando a classe ContaPoupanca
let poupanca = new ContaPoupanca(2000);
poupanca.calcularJuros();
console.log(`Saldo com juros: R$${poupanca.obterSaldo()}`);
```

Exercícios para Praticar

1. **Criando e Instanciando Classes:**
   - Defina uma classe `Carro` com propriedades para `marca`, `modelo`, e `ano`. Adicione um método que imprime uma descrição do carro.
2. **Explorando Herança:**
   - Crie uma classe `CarroEletrico` que herda de `Carro` e adiciona uma propriedade para a capacidade da bateria.
3. **Uso de Modificadores de Acesso:**
   - Modifique a classe `Conta` para prevenir acesso direto ao saldo, fornecendo métodos para depositar e obter o saldo atual.
4. **Implementando Polimorfismo:**
   - Crie uma classe `ContaCorrente` que sobrescreve o método `calcularJuros` com uma taxa específica para contas correntes.
5. **Classes com Interfaces:**
   - Implemente uma interface `Motorizavel` com um método `ligarMotor`. Crie classes `Barco` e `Motocicleta` que implementam esta interface.