Apostila de TypeScript Orientado a Objetos

1. Orientação a Objetos

O que é?

A Orientação a Objetos (OO) é um paradigma de programação que utiliza "objetos" - instâncias de "classes" - para criar modelos baseados no mundo real. Esses objetos possuem atributos (dados) e métodos (funções) que representam suas características e comportamentos.

Para que serve?

- **Modularidade**: Divide o código em partes menores e mais gerenciáveis.
- **Reutilização**: Facilita a reutilização de código através da herança e composição.
- **Flexibilidade**: Permite a fácil modificação e extensão do código existente.
- **Manutenção**: Torna o código mais fácil de entender, modificar e depurar.

Diferenças para Programação Estruturada

- **Estruturada**: Foco em funções e procedimentos.
- **OO**: Foco em objetos que combinam dados e comportamentos.

2. O que é uma Classe?

Uma classe é uma estrutura que define um conjunto de atributos e métodos que serão comuns a um determinado tipo de objeto.

Abstrair e Selecionar Classes de um Problema Real

1. **Identificar os objetos** no domínio do problema.
2. **Definir os atributos** e **comportamentos** desses objetos.
3. **Agrupar objetos semelhantes** em classes.

Exemplo: Em um sistema de biblioteca, podemos ter classes como `Livro`, `Autor`, e `Biblioteca`.

3. Classe: Atributos, Métodos e Relacionamento entre Classes

Atributos

Os atributos são as propriedades que descrevem os objetos.

Métodos

Os métodos são as funções que definem os comportamentos dos objetos.

Relacionamento entre Classes

- **Associação**: Relação de uso entre objetos.
- **Agregação**: Relação todo/parte onde as partes podem existir independentemente do todo.
- **Composição**: Relação todo/parte onde as partes não podem existir sem o todo.
- **Herança**: Especialização de classes (subclasses).

4. Atributos

Encapsulamento

Encapsulamento é a prática de esconder os detalhes internos de um objeto, expondo apenas o que é necessário através de métodos.

Tipos de Atributos com TypeScript

```
class Pessoa {
    private nome: string;
    public idade: number;
    protected endereco: string;
}
```

Atributos Nulos (Marcador com Sinal de Interrogação)

```
class Carro {
    public modelo?: string;

    constructor(modelo?: string) {
        this.modelo = modelo;
    }
}
```

5. Métodos

Definição

Métodos são funções definidas dentro de uma classe que descrevem os comportamentos dos objetos.

Exemplo em TypeScript

```
class Pessoa {
    private nome: string;

    constructor(nome: string) {
```

```
        this.nome = nome;
    }

    public saudar(): void {
        console.log(`Olá, meu nome é ${this.nome}`);
    }
}
```

6. Método Construtor

Definição

O construtor é um método especial que é chamado quando uma instância da classe é criada.

Exemplo em TypeScript

```
class Produto {
    public nome: string;
    public preco: number;

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

7. Instanciação de Objeto

Definição

A instanciação é o processo de criação de um objeto a partir de uma classe.

Exemplo em TypeScript

```
const produto = new Produto('Notebook', 2500);
console.log(produto.nome); // Output: Notebook
```

8. Método Estático

Definição

Métodos estáticos são aqueles que não precisam de um objeto para serem executados.

Exemplo em TypeScript

```
class Produto {
    public nome: string;
    public preco: number;

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

    cadastrar() {
        console.log(`O produto ${this.nome} foi cadastrado com sucesso!`);
    }

    static listar(stringpesquisa: string) {
        console.log(`Listando todos os produtos com ${stringpesquisa}`);
    }
}

let produto = new Produto('Notebook', 2500);
produto.cadastrar(); //Método não estático pois precisa de um objeto, no caso "prod
Produto.listar("Creme Dental"); //Método estático pois ele é evocado direto da clas
```

9. Herança de Classes

Definição

Herança é o mecanismo pelo qual uma classe herda atributos e métodos de outra classe.

Exemplo em TypeScript

```
class Animal {
    public nome: string;

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

    public mover(distancia: number): void {
        console.log(`${this.nome} moveu ${distancia} metros`);
    }
```

```
}

class Cachorro extends Animal {
    public latir(): void {
        console.log('Au au');
    }
}

const dog = new Cachorro('Rex');
dog.mover(5);
dog.latir();
```

10. Classes Abstratas

Definição

Classes abstratas não podem ser instanciadas diretamente e são usadas como base para outras classes.

Exemplo em TypeScript

```
abstract class Veiculo {
    abstract mover(): void;
}

class Carro extends Veiculo {
    mover(): void {
        console.log('O carro está se movendo');
    }
}
```

11. Novos Tipos de Dados a partir de Classes Criadas

Podemos usar classes como tipos de dados para atributos e retornos de métodos.

Exemplo em TypeScript

```
class Endereco {
    public rua: string;
    public numero: number;

    constructor(rua: string, numero: number) {
        this.rua = rua;
        this.numero = numero;
```

```
    }
}

class Pessoa {
    public nome: string;
    public endereco: Endereco;

    constructor(nome: string, endereco: Endereco) {
        this.nome = nome;
        this.endereco = endereco;
    }

    public obterEndereco(): Endereco {
        return this.endereco;
    }
}
```

12. Chamadas Assíncronas

Definição

Chamadas assíncronas permitem que o programa continue a execução enquanto espera por uma operação (como uma requisição de rede) ser completada.

Exemplo em TypeScript

```
class API {
    public async fetchData(): Promise<void> {
        const response = await fetch('https://api.example.com/data');
        const data = await response.json();
        console.log(data);
    }
}

const api = new API();
api.fetchData();
```

13. Interface com TypeScript

Definição

Interfaces definem a estrutura de um objeto, especificando os tipos de seus atributos e métodos.

Exemplo em TypeScript

```typescript
interface Pessoa {
    nome: string;
    idade: number;
    saudar(): void;
}

class Estudante implements Pessoa {
    public nome: string;
    public idade: number;

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

    public saudar(): void {
        console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);
    }
}

const estudante = new Estudante('João', 20);
estudante.saudar();
```