SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 18 | Módulo: Typescript e Orientação a Objetos

- ➔ Declaração de classes e criação de objetos
- ➔ Atributos e métodos: definição e uso

Abrindo editor de código

Vamos agora abrir o VSCode e criar os arquivos

- Com o programa aberto, clique em File > Open Folder... (Arquivo > Abrir Pasta...).
- Escolha um local para criar a sua pasta, crie uma nova pasta e dê o nome de **seunome_aula_18**. Depois dê dois clique nessa pasta criada e clique em **Selecionar pasta**. O VSCode reabrirá dentro dessa pasta que foi criada.
- Agora vamos criar o arquivo HTML:
- Dê o nome de **index.html**

Configurações iniciais

- Certifique que o NodeJS está instalado:
  - **node -v**
  - **npm -v**
- Se você usa Windows, caso não esteja instalado, baixe e instale:
  - **https://www.nodejs.tech/pt-br/download**
- Se você usa Linux, use o gerenciador de pacotes ou nvm se souber usar:
  - **sudo apt install nodejs npm**

Configurações iniciais

- No seu projeto, abra o terminal e vamos iniciar o projeto e instalar o pacote necessário:
  - **npm init -y** (criar o package.json, que controla as dependências do projeto.)
- Instale agora o compilador TypeScript como dev-dependência:
  - **npm i -D typescript**
- Gere agora o arquivo de configuração **tsconfig.json**:
  - **npx tsc --init**
- Crie uma pasta chamada **src** no diretório raiz do projeto (o mesmo diretório que está o index.html)

Configurações iniciais

- Quando o arquivo **tsconfig.json** for criado, vamos apagar todo o conteúdo e adicionar o que temos abaixo:

```
{
  "compilerOptions": {
    "target": "ES2017",
    "lib": ["DOM", "ES2017"],
    "module": "ES2015",
    "moduleResolution": "Node",
    "rootDir": "./src",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": ["src"]
}
```

Copie esse

tsconfig.json > ...

```
{
  "compilerOptions": {
    "target": "ES2017",          // versão do JS de saída (mais moderno e compatível)
    "lib": ["DOM", "ES2017"],    // bibliotecas incluídas (DOM = navegador, ES2017 = recursos modernos)
    "module": "ES2015",          // formato de módulo compatível com navegador (sem "exports")
    "moduleResolution": "Node",  // forma de resolver módulos (padrão do Node, funciona bem em geral)
    "rootDir": "./src",          // pasta onde ficam os arquivos .ts de origem
    "outDir": "./dist",          // pasta onde serão gerados os .js compilados
    "strict": true,              // ativa verificações mais rigorosas de tipagem
    "esModuleInterop": true,     // compatibilidade para importar libs JS antigas (ex: import express from "express")
    "skipLibCheck": true         // pula checagem de tipos em libs externas (compila mais rápido)
  },
  "include": ["src"]             // define quais pastas/arquivos entram na compilação
}

```

Explicação

Estrutura de Pastas + HTML base

- Crie o arquivo HTML e carregue o JS que será compilado.

index.html > ...

```html
<!DOCTYPE html>
<html lang="pt-BR">

<head>
    <meta charset="UTF-8" />
    <title>Aula 18</title>
</head>

<body>
    <h1>TypeScript + POO</h1>
    <p>Abra o console (F12) para ver a saída.</p>
    <script src="./dist/main.js"></script>
</body>

</html>
```

Estamos apontando para **./dist/main.js** *pois é o diretório onde o .js final ficará.*

“Assistir” mudanças (watch mode)

- Para compilar automaticamente ao salvar .ts, use o watch:
  - **npx tsc -w**
- Mantenha esse comando rodando enquanto edita TS.
  - Modifique a string da mensagem no arquivo **src/main.ts** e confirme que o dist/main.js é re-gerado.
- Atualize o navegador para ver mudanças.

Declaração de Classes e Criação de Objetos

- Na aula anterior, vimos os conceitos fundamentais da POO.
- Agora, vamos aprender como aplicar esses conceitos na prática com TypeScript.
- Focaremos em classes, objetos, atributos e métodos.
- Essa base é essencial para entender componentes e estados no React mais adiante.

O que é uma Classe

- Uma classe é um molde ou modelo usado para criar objetos.
- Ela define como o objeto será estruturado. Ou seja, quais informações ele terá (atributos) e o que poderá fazer (métodos).
- Em outras palavras:
  - classe = plano de construção
  - objeto = construção final
- No TypeScript, usamos a palavra-chave **class** para criar uma classe.

- **Analogia simples:**
  - A classe é a planta da casa;
  - O objeto é a casa construída a partir da planta.

Exemplo de class em TypeScript

src > main.ts > ...

```ts
class Pessoa {
    nome: string; // atributo (característica)

    constructor(nome: string) { // método especial: construtor
        this.nome = nome; // 'this' se refere à própria instância
    }
}

const pessoa1 = new Pessoa("Ana"); // criação de um objeto
console.log(pessoa1.nome); // saída: Ana
```

Typescript e Orientação a Objetos

Anatomia de uma Classe

- Nome da classe: Começa com letra maiúscula.
- Atributos: Guardam dados do objeto.
- Construtor (constructor): Inicializa os valores quando criamos o objeto.
- Métodos: Executam ações.

src > main.ts > ...

```typescript
class Carro {
    // 1. Atributos
    marca: string;
    ano: number;

    // 2. Construtor
    constructor(marca: string, ano: number) {
        // 'this' referencia o próprio objeto
        this.marca = marca;
        this.ano = ano;
    }

    // 3. Método
    buzinar() {
        console.log("Bibip!");
    }
}

// Criando um objeto (instância)
const meuCarro = new Carro("Toyota", 2024);
meuCarro.buzinar();  // Saída: Bibip

```

Classe e Objeto - Entendendo a Relação

- Classe: Define o modelo.
- Objeto: É uma instância concreta desse modelo.
- Podemos criar quantos objetos quisermos a partir da mesma classe.
- Cada objeto tem seus próprios valores, mesmo que venham da mesma classe.

Classe e Objeto - Entendendo a Relação

src > main.ts > ...

```typescript
class Animal {
    tipo: string;
    constructor(tipo: string) {
        this.tipo = tipo;
    }
}

const a1 = new Animal("Cachorro");
const a2 = new Animal("Gato");

console.log(a1.tipo); // "Cachorro"
console.log(a2.tipo); // "Gato"

```

O que são Atributos

- Atributos são as características ou dados que descrevem um objeto.
- Cada objeto tem seus próprios valores desses atributos.
- Em TypeScript, os atributos são variáveis declaradas dentro da classe.
- Eles podem ter tipos definidos (string, number, boolean, etc.) e valores padrão.

- Analogia:
  - Pessoa: atributos: nome, idade, altura
  - Carro: atributos: marca, cor, ano

Typescript e Orientação a Objetos

Exemplo de uma classe com Atributos

src > main.ts > ...

```ts
class Pessoa {
    nome: string = "Desconhecido";
    idade: number = 0;
}

const p1 = new Pessoa();
p1.nome = "Maria";
p1.idade = 25;

console.log(`${p1.nome} tem ${p1.idade} anos.`);
// Saída: Maria tem 25 anos.
```

Como Acessar e Alterar Atributos

- Para acessar um atributo usamos: **nome_do_objeto.atributo**
- Para alterar um atributo usamos: **nome_do_objeto.atributo = novoValor**
- É o mesmo princípio de acessar propriedades em objetos comuns do JavaScript, conforme já vimos em aulas anteriores.

Exemplo de como Acessar e Alterar Atributos

src > main.ts > ...

```ts
class Conta {
    titular: string = "";
    saldo: number = 0;
}

const conta1 = new Conta();
conta1.titular = "Lucas";
conta1.saldo = 500;

console.log(conta1.saldo); // 500

conta1.saldo += 200;
console.log(conta1.saldo); // 700
```

Exercício 1

- Crie uma classe **Carro** com os atributos marca e ano. Atribua valores e exiba no console.

O Papel do constructor

- O método **constructor()** é chamado automaticamente quando criamos um novo objeto com new.
- Ele serve para inicializar os atributos do objeto.
- Podemos passar valores como parâmetros no momento da criação.

O Papel do constructor

src > main.ts > ...

```ts
class Livro {
    titulo: string;
    autor: string;

    constructor(titulo: string, autor: string) {
        this.titulo = titulo;
        this.autor = autor;
    }

    exibir() {
        console.log(`"${this.titulo}" - ${this.autor}`);
    }
}

const livro1 = new Livro("Dom Casmurro", "Machado de Assis");
livro1.exibir(); // "Dom Casmurro" - Machado de Assis

```

Exercício 2

- Crie uma classe **Filme** com os atributos titulo e ano, e um método exibir() que mostre as informações no console.

Entendendo Métodos

- Métodos são funções criadas dentro de uma classe.
- Representam ações ou comportamentos que o objeto pode executar.
- Eles acessam e modificam atributos usando a palavra-chave this.
- Exemplo no mundo real:
  - Classe Carro: atributos: cor, modelo
  - Métodos: ligar(), acelerar(), frear()

Exercício prático sobre Métodos

src > main.ts > ...

```ts
class Pessoa {
    nome: string;

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

    falar() {
        console.log(`${this.nome} está falando.`);
    }
}

const p1 = new Pessoa("João");
p1.falar(); // João está falando.
```

- Adicione o método **andar()** que exiba: **${this.nome} está andando.**

Como os Métodos Funcionam

- O **this** dentro de um método sempre se refere ao próprio objeto.
- Ele é usado para acessar os atributos da instância atual.
- Cada objeto usa o mesmo método, mas com valores próprios de atributos.

Typescript e Orientação a Objetos

Exercício prático sobre Métodos

src > main.ts > ...

```ts
class Conta {
    titular: string;
    saldo: number;

    constructor(titular: string, saldo: number) {
        this.titular = titular;
        this.saldo = saldo;
    }

    depositar(valor: number) {
        this.saldo += valor;
        console.log(`${this.titular} depositou R$${valor}.`);
    }

    verSaldo() {
        console.log(`Saldo atual: R$${this.saldo}`);
    }
}

const c1 = new Conta("Maria", 100);
c1.depositar(50);
c1.verSaldo(); // Saldo atual: R$150
```

Crie o método **sacar(valor: number)** que diminui o saldo e mostre a nova quantia.

Métodos com Parâmetros e Retorno

- Métodos podem receber parâmetros (dados externos) e retornar valores.
- Isso os torna mais reutilizáveis e poderosos.
- O tipo de retorno pode ser especificado (ex: : number, : string, : void).

Métodos com Parâmetros e Retorno

src > main.ts > ...

```ts
class Calculadora {
    somar(a: number, b: number): number {
        return a + b;
    }

    exibirResultado(resultado: number): void {
        console.log(`O resultado é: ${resultado}`);
    }
}

const calc = new Calculadora();
const total = calc.somar(5, 3);
calc.exibirResultado(total); // O resultado é: 8
```

- Crie um método **multiplicar(a, b)** e exiba o resultado.

Métodos como Ações dos Objetos

- Métodos são úteis para modificar o estado (atributos) do objeto.
- Ajudam a manter o controle e encapsulamento dos dados.
- Exemplo: em vez de alterar o saldo diretamente, usamos o método depositar() para garantir segurança.

Métodos como Ações dos Objetos

src > main.ts > ...

```ts
class Lampada {
    ligada: boolean = false;

    ligar() {
        this.ligada = true;
        console.log("💡 A lâmpada está ligada.");
    }

    desligar() {
        this.ligada = false;
        console.log("💡 A lâmpada está desligada.");
    }
}

const lamp = new Lampada();
lamp.ligar(); // A lâmpada está ligada.
lamp.desligar(); // A lâmpada está desligada.
```

Exercício 3

➔ Vamos aplicar o que aprendemos criando uma classe mais completa.
➔ Ela terá atributos, construtor e múltiplos métodos.

- Crie uma classe **Aluno** com atributos: nome, nota1, nota2.
- Adicione o método **media()** que retorna a média.
- Adicione o método **situacao()** que exibe:
  - “Aprovado” se a média for maior ou igual a 7.
  - “Reprovado” se a média for menor que 7.

Gabarito - Exercício 1

src > main.ts > ...

```ts
class Carro {
    marca: string = "Desconhecido";
    ano: number = 0;
}

const c1 = new Carro();
c1.marca = "Fiat";
c1.ano = 2013;

console.log(`Marca: ${c1.marca} - Ano: ${c1.ano}`);
```

Gabarito - Exercício 2

src > main.ts > ...

```ts
class Filme {
    titulo: string;
    ano: number;

    constructor(titulo: string, ano: number) {
        this.titulo = titulo;
        this.ano = ano;
    }

    exibir() {
        console.log(`"${this.titulo}" - ${this.ano}`);
    }
}

const filme1 = new Filme("Titanic", 1997);
filme1.exibir(); // "Titanic" - 1997

```

Typescript e Orientação a Objetos

Gabarito - Exercício 3

src > main.ts > ...

```typescript
class Aluno {
    nome: string;
    nota1: number;
    nota2: number;

    constructor(nome: string, nota1: number, nota2: number) {
        this.nome = nome;
        this.nota1 = nota1;
        this.nota2 = nota2;
    }

    media(): number {
        return (this.nota1 + this.nota2) / 2;
    }

    situacao(): void {
        const media = this.media();
        if (media >= 7) {
            console.log(`${this.nome} está Aprovado.`)
        } else {
            console.log(`${this.nome} está Reprovado.`)
        }
    }
}

const aluno1 = new Aluno("Carlos", 8, 7);
aluno1.situacao(); // Carlos está Aprovado.
```

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

Professor: Hygor Rasec
https://www.linkedin.com/in/hygorrasec
https://github.com/hygorrasec