SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 17 | Módulo: Typescript e Orientação a Objetos

- ➔ Conceitos fundamentais da Programação Orientada a Objetos (POO)
- ➔ Introdução à linguagem TypeScript e suas vantagens sobre JavaScript

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_17**. 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**

Programação Orientada a Objetos com TypeScript

- Hoje vamos explorar dois pilares para projetos modernos:
  - **POO (Programação Orientada a Objetos):** organizar e estruturar melhor o código.
  - **TypeScript:** JavaScript com “superpoderes” (tipagem, segurança, clareza).
- Relevância: usados em empresas, projetos grandes e frameworks como React.
- A meta dessa aula: entender conceitos e aplicar em exemplos práticos.

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 > ...

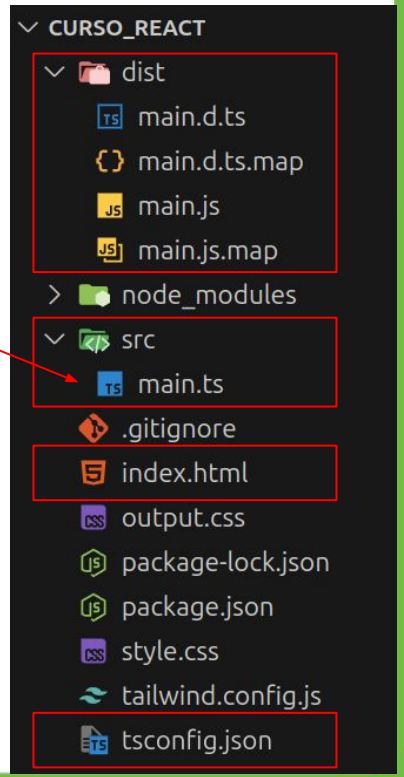
```html
<!DOCTYPE html>
<html lang="pt-BR">

<head>
    <meta charset="UTF-8" />
    <title>Aula 17</title>
    <link rel="stylesheet" href="style.css" />
</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.*

Primeiro arquivo TypeScript

- Crie o arquivo **main.ts** dentro da pasta src que criamos: **src/main.ts**
- Escreva um código simples (com tipagem) para validar o setup.

```
src > main.ts > ...
1   let mensagem: string = "Boas vindas à Aula 17!";
2   console.log(mensagem);
3
```



Depois de digitar, salve o arquivo e ele já está pronto para ser compilado.

Ao lado temos a estrutura base do projeto.

ATENÇÃO: A pasta **dist** será criada depois que compilarmos.

Compilar TS > JS (build)

- Agora vamos compilar para gerar **dist/main.js**
  - Rode o comando para compilar: **npx tsc**
- Verifique se **dist/main.js** foi criado/existe. Abra e veja que é JavaScript puro.
- Se der erro na hora de compilar, leia a mensagem do TypeScript.

Rodar no Navegador

- Abra o arquivo **index.html** no navegador ou use o Live Server do VS Code.
- Abra o console (F12) para visualizar a mensagem: “Boas vindas à Aula 17!”.
- Se não aparecer, verifique o caminho do <script> e recarregue a página.

“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.

O que é Programação Orientada a Objetos (POO)?

- É um paradigma de programação baseado em objetos.
- Objetos representam entidades do mundo real (ex.: carro, pessoa, produto).
- Cada objeto possui:
  - Atributos: características/dados (ex.: nome, cor, preço).
  - Métodos: comportamentos/ações (ex.: ligar(), andar(), calcularTotal()).
- Ajuda a:
  - Organizar melhor o código.
  - Reutilizar funcionalidades.
  - Tornar o projeto mais fácil de entender e manter.

Exemplo em TypeScript

src > main.ts > ...

```ts
// Representando um "Carro" como objeto
const carro = {
    marca: "Fiat", // atributo
    ano: 2022, // atributo
    ligar() {
        console.log("O carro foi ligado!") // método
    }
};

// Usando o objeto
console.log(carro.marca); // Fiat
carro.ligar(); // O carro foi ligado!
```

Esse é só um objeto literal. Mais à frente, veremos como criar **classes** para gerar muitos objetos do mesmo tipo.

Pilares da POO

- **Abstração**
  - Foca apenas nos detalhes essenciais, ignorando o que não importa.
  - Exemplo: um objeto "Carro" mostra ligar() mas não precisa mostrar como funciona o motor internamente.
- **Encapsulamento**
  - Esconder dados internos e controlar o acesso a eles.
  - Garante segurança e evita que partes externas quebrem o sistema.
- **Herança**
  - Reaproveitar código: uma classe filha herda atributos e métodos da classe pai.
  - Exemplo: "Cachorro" e "Gato" podem herdar de "Animal".
- **Polimorfismo**
  - Um mesmo método pode ter comportamentos diferentes dependendo do objeto.
  - Exemplo: o método falar() de um "Cachorro" retorna "Au au", de um "Gato" retorna "Miau".

Typescript e Orientação a Objetos

Exemplo

src > main.ts > ...

```ts
abstract class Animal {
    constructor(public nome: string) {}
    abstract falar(): void; // método abstrato
}

class Cachorro extends Animal {
    falar() { console.log("Au au!"); }
}

class Gato extends Animal {
    falar() { console.log("Miau!"); }
}

const animais: Animal[] = [new Cachorro("Rex"), new Gato("Luna")];
animais.forEach(a => a.falar()); // Au au! / Miau!

```

Explicação do Exemplo

- **abstract class Animal**
  - "abstract" significa que a classe não pode ser usada diretamente para criar objetos.
  - Ela serve como modelo para outras classes.
  - O método falar() está definido, mas não tem implementação, quem herdar dessa classe vai precisar implementar esse método.
  - *Veremos abstract em detalhe em uma próxima aula.*

```
abstract class Animal {
    constructor(public nome: string) {}
    abstract falar(): void; // método abstrato
}
```

Explicação do Exemplo

- **class Cachorro extends Animal / class Gato extends Animal**
  - Aqui temos herança: Cachorro e Gato herdam de Animal.
  - O extends significa "pega tudo que Animal tem e acrescenta a sua própria lógica".
  - Cada classe define sua versão própria do método falar().
  - *class e extends serão explicados melhor na próxima aula.*

```
class Cachorro extends Animal {
    falar() { console.log("Au au!"); }
}

class Gato extends Animal {
    falar() { console.log("Miau!"); }
}
```

Explicação do Exemplo

- **const animais: Animal[] = [...]**
  - Criamos uma lista (array) que só aceita objetos do tipo Animal.
  - Mesmo sendo Cachorro e Gato, eles podem ser tratados como Animal porque herdam dele.

```
const animais: Animal[] = [new Cachorro("Rex"), new Gato("Luna")];
```

Explicação do Exemplo

- **animais.forEach(a => a.falar());** (polimorfismo em ação)
  - Percorremos o array com forEach.
  - Para cada **a** (animal), chamamos falar().
  - Como cada classe tem sua própria implementação, o resultado é diferente:
    - Cachorro: "Au au!"
    - Gato: "Miau!"
  - Um mesmo método (falar) se comporta de formas diferentes dependendo do objeto.

```
animais.forEach(a => a.falar()); // Au au! / Miau!
```

Typescript e Orientação a Objetos

O que é TypeScript?

- TypeScript é uma linguagem criada pela Microsoft.
- É um superconjunto do JavaScript, ou seja: tudo que funciona em JS funciona em TS.
- Adiciona tipagem estática (definir tipos para variáveis, funções, parâmetros).
- Gera mais segurança e clareza no código.
- Antes de rodar no navegador, o TypeScript precisa ser compilado para JavaScript.

Typescript e Orientação a Objetos

O que é TypeScript?

- TypeScript é uma linguagem criada pela Microsoft.
- É um superconjunto do JavaScript, ou seja: tudo que funciona em JS funciona em TS.
- Adiciona tipagem estática (definir tipos para variáveis, funções, parâmetros).
- Gera mais segurança e clareza no código.
- Antes de rodar no navegador, o TypeScript precisa ser compilado para JavaScript.

src > main.ts > ...

```ts
// Exemplo em TypeScript
let idade: number = 20;
console.log(idade);

// Se tentarmos mudar para uma string, dá erro de compilação:
idade = "vinte"; // ❌ Erro: string não é número
```

Vantagens do TypeScript sobre JavaScript

- Tipagem estática: evita muitos erros antes mesmo de rodar o código.
- IntelliSense melhorado no VS Code: autocompletar mais inteligente.
- Organização e clareza: especialmente útil em projetos grandes com muitas pessoas.
- Compatível com JavaScript: você pode misturar JS e TS na transição.
- Integração com frameworks modernos (React, Angular, Vue).

Typescript e Orientação a Objetos

Exemplo prático de uma das vantagens do TypeScript sobre JavaScript

script.js > ...

```javascript
// JavaScript: pode causar erros inesperados
function somaJS(a, b) {
    return a + b;
}

console.log(somaJS(5, "10")); // Resultado: "510" (concatenação!)
```

src > main.ts > ...

```typescript
// TypeScript: garante segurança
function somaTS(a: number, b: number): number {
    return a + b;
}

console.log(somaTS(5, 10)); // Resultado: 15
// console.log(somaTS(5, "10")); // ❌ Erro: "string" não é "number"

```

Agora vamos praticar

- Crie um arquivo main.ts dentro da pasta src/.
- Escreva um código simples com tipagem.
- Compile para gerar main.js na pasta dist/.
- Visualize no navegador.

Typescript e Orientação a Objetos

Agora vamos praticar

src > main.ts > ...

```ts
// Declaração com tipagem
let mensagem: string = "Bem-vindos à Aula 17 de TypeScript!";
console.log(mensagem);

// Função tipada
function dobro(valor: number): number {
    return valor * 2;
}

console.log("Dobro de 5 é:", dobro(5));

```

let mensagem: string - variável só aceita texto.
function dobro(valor: number): number - função recebe e retorna apenas números.
Se tentarmos passar outro tipo, o TypeScript dá erro antes de rodar.

Checando o Entendimento

- Vimos os conceitos fundamentais da POO (objetos, atributos, métodos, pilares).
- Entendemos o que é TypeScript e por que ele é usado no lugar do JS puro.
- Vimos exemplos simples com tipagem e como ela evita erros.

Typescript e Orientação a Objetos

Desafio prático

- Crie uma variável chamada **aluno** que guarde o nome de um aluno (string).
- Crie uma função chamada **apresentarAluno** que recebe esse nome e imprima no console:
  - "Olá, eu sou [nome] e estou aprendendo TypeScript!".
- Teste chamando a função com uma string.
- Agora tente chamar passando um número (sem aspas), veja o erro do TypeScript.

ATENÇÃO! Quando uma função não retorna nada, usamos “void” no lugar do tipo.

```
function nomeDaFuncao(nome: string): void {
    //...
}
```

Typescript e Orientação a Objetos

Gabarito do desafio prático

src > main.ts > ...

```ts
// 1. Variável do tipo string
let aluno: string = "Hygor";

// 2. Função tipada
function apresentarAluno(nome: string): void {
    console.log(`Olá, eu sou ${nome} e estou aprendendo TypeScript!`);
}

// 3. Testando com string
apresentarAluno(aluno);
// Saída: Olá, eu sou Hygor e estou aprendendo TypeScript!

// 4. Testando com número
// apresentarAluno(123);
// ❌ Erro de compilação:
// Argument of type 'number' is not assignable to parameter of type 'string'.

```

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

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