SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 25 | Módulo: Typescript e Orientação a Objetos (continuação)

- ➔ Como utilizar interfaces em TypeScript
- ➔ Composição e agregação de objetos (relações entre objetos)

Abrindo editor de código

Vamos usar um editor na web para a aula de hoje

- Acesse:
  https://www.typescriptlang.org/play

➔ Nesse site, vamos aplicar nossos códigos e testar.

Abrindo editor de código

Caso queira usar o VSCode, siga os passos abaixo

- 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_25**. 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**

Typescript e Orientação a Objetos (continuação)

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)
- Dentro da pasta **src** criada, crie o arquivo **main.ts**

Typescript e Orientação a Objetos (continuação)

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

Typescript e Orientação a Objetos (continuação)

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.*

Typescript e Orientação a Objetos (continuação)

“Assistir” mudanças (watch mode)

- Para compilar automaticamente ao salvar .ts, use o watch:
  - **npx tsc -w**

Typescript e Orientação a Objetos (continuação)

Interfaces, Composição e Agregação

- Vamos retomar o estudo de Programação Orientada a Objetos com TypeScript, explorando como os objetos se relacionam entre si e como padronizar estruturas de código.
- O que veremos hoje?
  - **Interfaces:** Como definir um "modelo" para objetos e classes, garantindo que diferentes partes do sistema sigam a mesma estrutura.
  - **Composição:** Como construir objetos complexos a partir de outros objetos menores, criando relações do tipo “tem um” (exemplo: um carro tem um motor).
  - **Agregação:** Como criar relações mais flexíveis entre objetos, onde um pode existir sem o outro (exemplo: um time tem jogadores, mas os jogadores também existem fora do time).

Typescript e Orientação a Objetos (continuação)

Vamos relembrar...

Qual a diferença entre uma **classe** e um **objeto**?

- **Classe** é o molde ou modelo que define como algo deve ser.
  - **Exemplo:** A planta de uma casa (define como será construída).
- **Objeto** é uma instância real criada a partir desse molde.
  - **Exemplo:** A casa que foi construída usando a planta.

```
1   class Carro {
2       marca: string;
3       constructor(marca: string) {
4           this.marca = marca;
5       }
6   }
7
8   const meuCarro = new Carro("Fiat");
9
10  console.log(meuCarro);
11  console.log(meuCarro.marca);
```

Typescript e Orientação a Objetos (continuação)

O que é uma Interface?

- Uma interface é como um molde para objetos, que define quais propriedades e tipos de dados eles devem ter.
- Ela não guarda valores, apenas define como o objeto deve ser.
- Ajuda a manter o código padronizado e organizado.

Typescript e Orientação a Objetos (continuação)

- **interface Usuario** define o formato obrigatório do objeto.
- **const pessoa: Usuario** indica que o objeto deve seguir as regras da interface.
- Se faltar alguma propriedade (por exemplo, email), o TypeScript mostra um erro de compilação.

Exemplo

```typescript
// Definindo uma interface
interface Usuario {
  nome: string;
  idade: number;
  email: string;
}

// Criando um objeto seguindo a interface
const pessoa: Usuario = {
  nome: "Hygor",
  idade: 37,
  email: "hygorrasec@gmail.com"
};

console.log("Objeto completo:", pessoa);
console.log("Nome:", pessoa.nome);
console.log("Idade:", pessoa.idade);
console.log("Email:", pessoa.email);
```

Typescript e Orientação a Objetos (continuação)

Interface

- A interface serve justamente para garantir que o código mantenha o tipo correto, evitando erros futuros.

```typescript
// Definindo uma interface
interface Usuario {
  nome: string;
  idade: number;
  email: string;
}

// Criando um objeto seguindo a interface
const pessoa: Usuario = {
  nome: "Hygor",
  idade: "37",
  email: "hygorrasec@gmail.com"
};

console.log("Objeto completo:", pessoa);
console.log("Nome:", pessoa.nome);
console.log("Idade:", pessoa.idade);
console.log("Email:", pessoa.email);
```

Errors in code

Type 'string' is not assignable to type 'number'.

Typescript e Orientação a Objetos (continuação)

Interfaces em Classes

- Por que usar interfaces em classes?
  - Uma classe pode **“seguir um contrato”** definido por uma interface.
  - Isso significa que ela **deve conter** todos os atributos e métodos definidos na interface.
  - É uma forma de **garantir que diferentes classes tenham a mesma estrutura**.

Typescript e Orientação a Objetos (continuação)

```typescript
// Interface define o contrato
interface Animal {
  nome: string;
  fazerSom(): void;
}

// Classe que implementa a interface
class Cachorro implements Animal {
  constructor(public nome: string) {
    this.nome = nome;
  }

  fazerSom() {
    console.log(`${this.nome} diz: Au au!`);
  }
}

// Criando objeto (instância da classe)
const rex = new Cachorro("Titan");

// Exibindo no console
console.log("Nome do animal:", rex.nome);
rex.fazerSom();
```

```
[LOG]: "Nome do animal:",  "Titan"
[LOG]: "Titan diz: Au au!"
```

Typescript e Orientação a Objetos (continuação)

Interfaces em Classes

- A interface Animal define o que todo animal deve ter: um nome e um método fazerSom().
- A classe Cachorro implementa essa interface, ou seja, cumpre o contrato.
- Se você esquecer algum item da interface (ex: remover o método **fazerSom()**), o TypeScript mostrará um erro:
  - Class ‘Cachorro’ incorrectly implements interface ‘Animal’.
  - Property ‘fazerSom’ is missing in type ‘Cachorro’ but required in type ‘Animal’.

Typescript e Orientação a Objetos (continuação)

Exercício 1

- Crie uma nova classe chamada **Gato** que também implemente a interface Animal.
- No método fazerSom(), exibir:
  - console.log(`${this.nome} diz: Miau!`);
- Crie um objeto **const gato = new Gato("Gatinha")** e testar no console.

Typescript e Orientação a Objetos (continuação)

Vantagens do Uso de Interfaces

- As interfaces trazem vários benefícios que deixam o código mais seguro, padronizado e fácil de manter:
  - **Organização:** deixam claro o formato dos dados que o código deve seguir.
  - **Reutilização:** a mesma interface pode ser usada em várias partes do projeto.
  - **Padronização:** garante que todos os objetos e classes tenham a mesma estrutura.
  - **Menos erros:** o TypeScript alerta se algo estiver fora do padrão.
  - **Integração mais fácil:** muito usadas em APIs e componentes React para definir o formato de dados.

Typescript e Orientação a Objetos (continuação)

Vantagens do Uso de Interfaces

```typescript
// Interface que define o formato das props
interface BotaoProps {
  texto: string;
  onClick: () => void;
}

// Simulação de um componente React (sem JSX real)
function Botao(props: BotaoProps) {
  console.log("Renderizando botão com texto:", props.texto);
  props.onClick();
}

// Usando a interface corretamente
Botao({
  texto: "Clique aqui",
  onClick: () => console.log("Botão clicado!")
});
```

Typescript e Orientação a Objetos (continuação)

Vantagens do Uso de Interfaces

- A interface **BotaoProps** define o formato esperado para o parâmetro props.
- Se faltar algum campo (ex: não enviar onClick), o TypeScript avisa:
  - Property ‘onClick’ is missing in type ...
- Isso evita erros comuns em React, onde o componente espera uma prop e não a recebe corretamente.

➔ Interfaces garantem que diferentes partes do sistema “falem a mesma língua”.
➔ Em React, ajudam a padronizar props, evitar erros e facilitar manutenção.
➔ Em sistemas maiores, elas funcionam como pontes entre componentes e APIs.

Typescript e Orientação a Objetos (continuação)

Exercício 2

- Adicione um novo campo na interface:
  - cor?: string; // campo opcional
- Alterar a função Botao para exibir também a cor, caso exista:
  - console.log(“Cor:”, props.cor ?? “padrão”);
- Testar o código enviando:
  - Botao({ texto: “Enviar”, onClick: () => console.log(“Enviado!”), cor: “verde” });

Typescript e Orientação a Objetos (continuação)

Exercício 3

- Crie uma interface chamada **Produto** com os campos:
  - nome (string)
  - preco (number)
  - estoque (number)
- Depois, crie uma função **exibirProduto(produto: Produto)** que exibe as informações no console.

Typescript e Orientação a Objetos (continuação)

Composição de Objetos

- Composição é quando um objeto é formado por outros objetos.
- É uma relação do tipo “tem um”.
- Exemplo do mundo real:
  - Um carro tem um motor.
  - Um computador tem um processador.
- Em outras palavras:
  - Em vez de herdar algo, um objeto usa outro objeto como parte de si.

Typescript e Orientação a Objetos (continuação)

Composição de Objetos

```
1   class Motor {
2     constructor(public potencia: number) {
3       this.potencia = potencia;
4     }
5   }
6
7   class Carro {
8     constructor(public modelo: string, public motor: Motor) {
9       this.modelo = modelo;
10      this.motor = motor;
11    }
12  }
13
14  const motor = new Motor(85);
15  const carro = new Carro("Fiat", motor);
16  console.log(carro);
17  console.log(carro.modelo);
18  console.log(carro.motor.potencia)
```

Typescript e Orientação a Objetos (continuação)

Agregação de Objetos

- Parecida com composição, mas a existência é independente.
- Exemplo:
  - Um time tem jogadores, mas os jogadores podem existir sem o time.

Typescript e Orientação a Objetos (continuação)

Agregação de Objetos

```typescript
class Jogador {
  constructor(public nome: string) { this.nome = nome; }
}

class Time {
  constructor(public nome: string, public jogadores: Jogador[]) {
    this.nome = nome;
    this.jogadores = jogadores;
  }
}

const j1 = new Jogador("Ronaldo");
const j2 = new Jogador("Neymar");
const selecao = new Time("Brasil", [j1, j2]);

console.log(selecao.nome);
selecao.jogadores.forEach(jogador => {
  console.log(jogador.nome);
});

```

Typescript e Orientação a Objetos (continuação)

Agregação de Objetos

- A classe Time usa objeto(s) Jogador, mas não é dono deles.
- Mesmo que o time seja excluído, os jogadores continuam existindo.
- Diferente da composição, na agregação a dependência é fraca.

Typescript e Orientação a Objetos (continuação)

Interfaces + Relações

- Podemos usar interfaces junto com composição e agregação.
- Isso garante flexibilidade e organização.

Typescript e Orientação a Objetos (continuação)

Interfaces + Relações

```
1   interface Endereco {
2     rua: string;
3     cidade: string;
4   }
5
6   class Pessoa {
7     constructor(public nome: string, public endereco: Endereco) {}
8   }
9
10  const endereco = { rua: "Av. Brasil", cidade: "Maricá" };
11  const pessoa = new Pessoa("Hygor", endereco);
12
13  console.log(pessoa);
14  console.log(pessoa.endereco);
15  console.log(pessoa.nome);
16
```

Typescript e Orientação a Objetos (continuação)

Gabarito Exercício 1

```typescript
interface Animal {
  nome: string;
  fazerSom(): void;
}

class Gato implements Animal {
  constructor(public nome: string) {
    this.nome = nome;
  }

  fazerSom() {
    console.log(`${this.nome} diz: Miau!`);
  }
}

const gatinha = new Gato("Gatinha");
gatinha.fazerSom();
```

Typescript e Orientação a Objetos (continuação)

Gabarito Exercício 2

```typescript
interface BotaoProps {
  texto: string;
  onClick: () => void;
  cor?: string;  // opcional
}

function Botao(props: BotaoProps) {
  console.log("Renderizando botão com texto:", props.texto);
  props.onClick();
  console.log("Cor:", props.cor ?? "padrão");
}

Botao({
    texto: "Enviar",
    onClick: () => console.log("Enviado!"),
    cor: "verde"
});
```

Typescript e Orientação a Objetos (continuação)

Gabarito Exercício 3

```typescript
interface Produto {
  nome: string;
  preco: number;
  estoque: number;
}

function exibirProduto(produto: Produto) {
  console.log("Nome:", produto.nome);
  console.log("Preço: R$", produto.preco);
  console.log("Estoque disponível:", produto.estoque, "unidades");
}

const produto1: Produto = {
  nome: "Teclado Mecânico",
  preco: 350.99,
  estoque: 12
};

exibirProduto(produto1);
```

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

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