ACADEMY

LAPIDANDO TALENTOS

CONHECENDO O ANGULAR ~~1.0~~

AULA 2 – TYPESCRIPT

Eder Franco @ FPF Tech

Agenda

1. Breve histórico
2. Porque utilizar TypeScript?
3. TypeScript vs JS simples
4. Download e configuração
5. Principais características:
    - Tipos básicos
    - Declaração de variáveis
    - Interfaces
    - Classes

Agenda

5. Principais características (continuação):
    - Funções
    - Tipos genéricos
    - Enums
    - Modularização
6. Utilizando arquivos de declaração de tipos
7. Referências

TypeScript

Breve histórico

- Criado em 2012 pela Microsoft;
- Anders Hejlsberg (principal arquiteto do C# e criador do Delphi e Turbo Pascal);
- Versão atual (jul/2020):
  - TypeScript 3.9

Microsoft

Porque utilizar TypeScript?

- Pleno suporte ao ES6;
- Sintaxe semelhante ao JavaScript;
- Potencializa e simplifica o uso da orientação a objetos;
- Adiciona recursos que ES6 puro não possui (annotations / decorators, generics, etc);
- Permite definir tipos de dados e interfaces, proporcionando um *lint* mais poderoso;

Download e Configuração

Download e Configuração

#comofaz?

- É necessário instalar o compilador do TypeScript:
  - Utilizando o Visual Studio (Windows);
  - Via NPM (para qualquer plataforma e IDE);
- Veja mais em:
  - http://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Download e Configuração

Mãos à obra!

- Instalar o Node.js:
  - https://nodejs.org/en/download/
- Instalação com node:
  - npm install –g typescript

Download e Configuração

Mãos à obra!

- Plugins para o Visual Studio Code:
  - Auto Import: steoates.autoimport
  - TSLint: eg2.tslint

TypeScript vs Javascript

TypeScript vs Javascript

Mãos à obra!

- Exemplo 1 (JS vanilla):
  - Criar uma página simples que solicite o nome e o sobrenome do usuário e escreva no documento HTML;

TypeScript vs Javascript

Mãos à obra!

- Exemplo 2 (ES6):
  - O mesmo exemplo anterior;

Principais características

Principais características

Compilação

- A maneira mais simples para compilar TypeScript em JavaScript é utilizando a linha de comando:
  - Criar um arquivo *index.ts*
  - Abrir o console e digitar:
    - tsc index.ts

Principais características

Compilação

- Exemplo:
  - Comando *tsc index.ts* e resultado:


```ts
let teste: string = "Olá, mundo!";
```

```js
var teste = "Olá, mundo!";
```

Principais características

Tipos básicos

- Boolean

Principais características

Tipos básicos

- Strings

strings.ts

```ts
let cor: string = "blue";
let nome_completo: string = `Eder Franco`;
let idade: number = 34;
let frase: string = `Ola! Meu nome é ${ nome_completo }.`;
```

strings.js

```js
var cor = "blue";
var nome_completo = "Eder Franco";
var idade = 34;
var frase = "Ola! Meu nome \u00E9 " + nome_completo + ".";
```

Principais características

Tipos básicos

- Number

TS number.ts

```ts
let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
```

JS number.js

```js
var decimal = 6;
var hex = 0xf00d;
var binary = 10;
var octal = 484;
```

Principais características

Tipos básicos

- Array

TS array.ts

```ts
let list: number[] = [1, 2, 3];
let list2: Array<number> = [1, 2, 3];
```

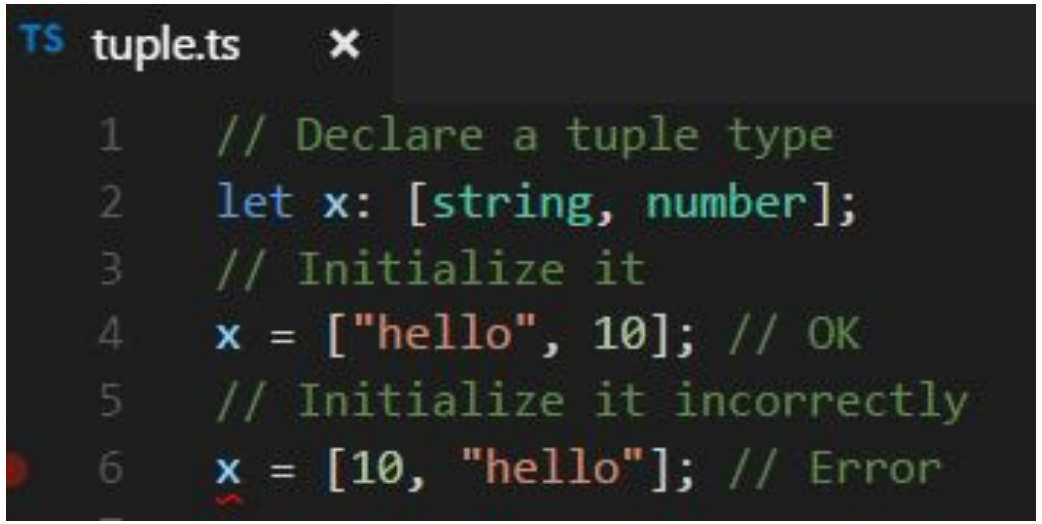
JS array.js

```js
var list = [1, 2, 3];
var list2 = [1, 2, 3];
```

Principais características

Tipos básicos

- Tuple
  - Permite declarar um array com tipos de dados diferentes.



Principais características

Tipos básicos

- Enum

TS enum.ts

```ts
enum Cor {
    Azul = 1,
    Vermelho = 2,
    Verde = 3
}

let chave_cor = Cor.Azul;
let nome_cor = Cor[1];

console.log(chave_cor); // 1
console.log(nome_cor); // Azul
```

JS enum.js

```js
var Cor;
(function (Cor) {
    Cor[Cor["Azul"] = 1] = "Azul";
    Cor[Cor["Vermelho"] = 2] = "Vermelho";
    Cor[Cor["Verde"] = 3] = "Verde";
})(Cor || (Cor = {}));
var chave_cor = Cor.Azul;
var nome_cor = Cor[1];
console.log(chave_cor); // 1
console.log(nome_cor); // Azul
```

Principais características

Tipos básicos

- Any
  - Para tipagem dinâmica;

any.ts

```ts
let qualquerCoisa: any = 4; // Numérico
qualquerCoisa = "Agora é uma string...";
qualquerCoisa = false; // Agora é um boleano
```

Principais características

Tipos básicos

- Any
  - Para tipagem dinâmica;

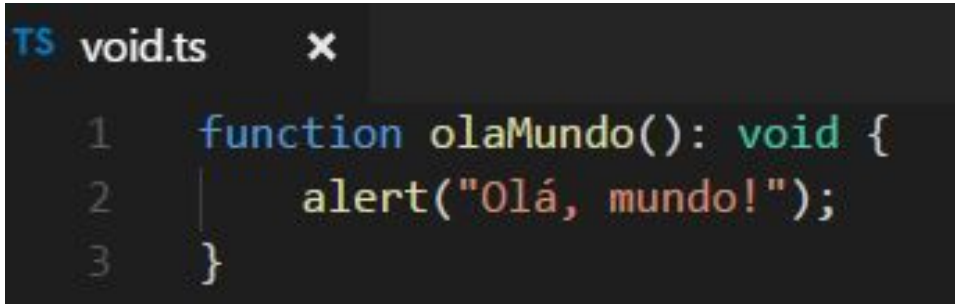
any.ts

```ts
let qualquerCoisa: any = 4; // Numérico
qualquerCoisa = "Agora é uma string...";
qualquerCoisa = false; // Agora é um boleano
```

Principais características

Tipos básicos

- Void
  - O oposto de any: não é nenhum tipo!



Principais características

Tipos básicos

- Outros tipos:
  - Null e Undefined (mesmos usos do JS);
  - Documentação:
    - https://www.typescriptlang.org/docs/handbook/basic-types.html

Principais características

Declaração de variáveis

- Segue o mesmo padrão do ES6.
  - Exemplos:

exemplos.ts

```ts
var numero: number = 0;

let mensagem: string = "Alguma coisa";

let objeto: Object = {
    chave1: null,
    chave2: null
};
```

Principais características

Interfaces

- Permite definir tipos customizados para variáveis;
  - Exemplo:

exemplos.ts

```ts
interface Pessoa {
    nome: string;
    idade: number;
}

let pessoa: Pessoa;
pessoa.nome = "Eder";
// Erro de checagem de tipo
pessoa.idade = "Franco";
```

Principais características

Interfaces

- Propriedades opcionais:
  - Exemplo:

exemplos.ts

```ts
interface Pessoa {
    nome: string;
    idade?: number;
}

let pessoa: Pessoa;
pessoa.nome = "Eder";
```

Principais características

Interfaces

- Readonly: propriedades que não podem ser modificadas após a criação do objeto.
  - Exemplo:

```
interface Coordenadas {
    readonly x: number;
    readonly y: number;
}

let ponto1: Coordenadas = { x: 10, y: 20 };
ponto1.x = 5; // error ao tentar mudar o valor
```

Principais características

Interfaces

- Tipos para funções
  - Exemplo:

```
interface Potencia {
    (numero: number, potencia: number): number;
}

let potencia: Potencia;
potencia = function (numero: number, potencia: number) {
    let resultado = Math.pow(numero, potencia);
    return resultado;
}

let resultado = potencia(2,4);
console.log(resultado); // 16
```

Principais características

Interfaces

- Classes implementando interfaces
  - Exemplo:

```
interface Pessoa {
    nome: string;
}

class Aluno implements Pessoa {
    nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }
}
```

Principais características

Interfaces

- Interfaces com herança
  - Exemplo:

```
interface Pessoa {
    nome: string;
}

interface Aluno extends Pessoa {
    registro: number;
}

let aluno = <Aluno>{}; // OU let aluno: Aluno;
aluno.nome = "José da Silva";
aluno.registro = 25;
```

Principais características

Classes

- Exemplo:

```
class Aluno {
    private nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }

    public getNome(): string {
        return this.nome;
    }
}

let aluno: Aluno = new Aluno("Pedro Oliveira");
console.log(aluno.getNome()); // Pedro Oliveira
```

Principais características

Classes

- Herança
  - Exemplo:

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

class Aluno extends Pessoa {
    registro: number;
    constructor(nome: string, registro: number) {
        super(nome);
        this.registro = registro;
    }
}
```

Principais características

Classes

- Modificadores de acesso (público)
  - Exemplo:

```
class Animal {
    public nome: string;
    public constructor(nome: string) { this.nome = nome; }
    public dizerNome() {
        alert(`Meu nome é ${this.nome}!`);
    }
}
```

Principais características

Classes

- Modificadores de acesso (privado)
  - Exemplo:

```
class Animal {
    private nome: string;
    public constructor(nome: string) { this.nome = nome; }
    public dizerNome() {
        alert(`Meu nome é ${this.nome}!`);
    }
}

new Animal("Floquinho").nome; // Erro - nome é privado
```

Principais características

Classes

- Modificadores de acesso (protegido)
  - Exemplo:

```
class Pesssoa {
    protected nome: string;
    constructor(nome: string) { this.nome = nome; }
}

class Funcionario extends Pesssoa {
    private setor: string;

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

    public apresentacao() {
        return `Olá! Meu nome é ${this.nome} e eu trabalho no setor ${this.setor}.`;
    }
}

let jose = new Funcionario("José", "Vendas");
console.log(jose.apresentacao());
console.log(jose.nome); // error
```

Principais características

Funções

- Podem ser definidas de maneira semelhante ao ES5, mas considerando os tipos de retorno:

```
// ES5
function soma(x, y) {
    return x + y;
}

// TypeScript
function somaTS(x: number, y: number): number {
    return x + y;
}
```

Principais características

Tipos Genéricos

- Exemplo:

```
class Funcionario {
    nome: string;
    data_nascimento: Date;
    matricula: number;
}

class Paginacao<T> {
    resultados: Array<T>;
    pagina: number = 1;
    limite?: number = 10;
}

let paginacao = new Paginacao<Funcionario>();
```

TypeScript vs Javascript

Mãos à obra!

- Exemplo 3 (TypeScript):
  - Vamos implementar o mesmo exemplo anterior, mas agora utilizando:
    - Orientação a Objetos;
    - TypeScript;

Principais características

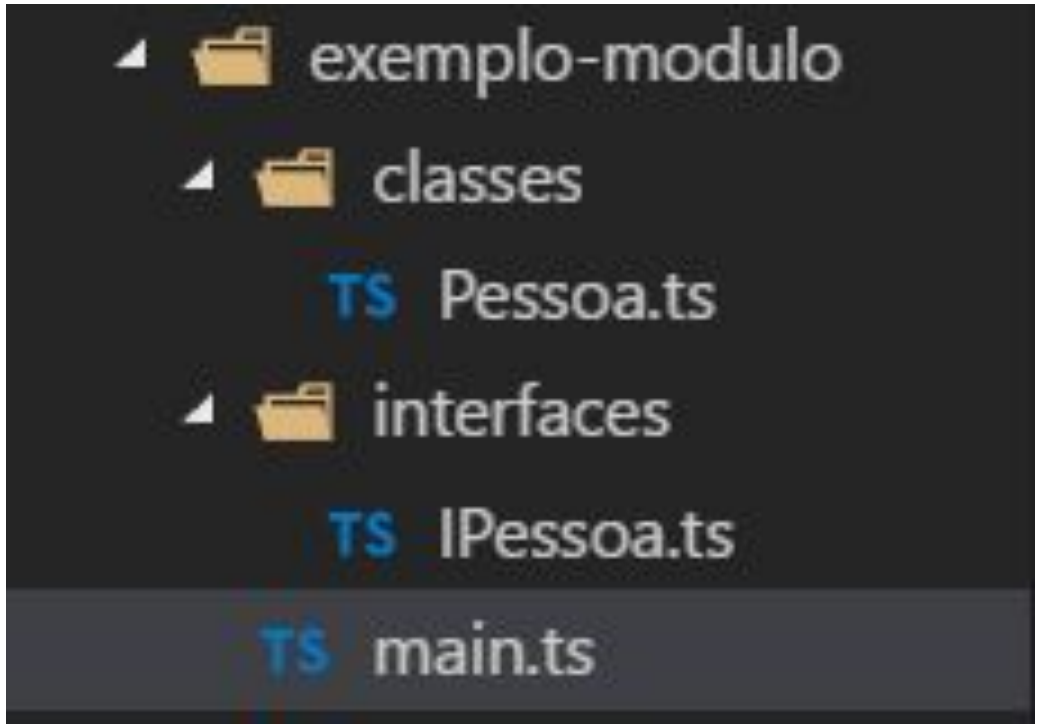
Modularização

- Typescript permite importar e exportar declarações (variáveis, classes, fuções, interfaces, etc) contidas em diferentes arquivos;
- Para isso, deve-se preceder as declarações com a palavra *export;*
- O elemento exportado pode ser utilizado em outro arquivo .ts utilizando o comando *import.*

Principais características

Modularização

- Exemplo:



Principais características

Modularização

- Exemplo:

IPessoa.ts

```ts
export interface IPessoa {
    nome: string;
    email: string;
    cpf?: string;
}
```

Pessoa.ts

```ts
import { IPessoa } from './../interfaces/IPessoa';

export class Pessoa implements IPessoa {
    nome: string;
    email: string;
}
```

main.ts

```ts
import { Pessoa } from "./classes/Pessoa";

let pessoa = new Pessoa();
pessoa.nome = "Eder";
pessoa.email = "Franco";
console.log(pessoa);
```

Principais características

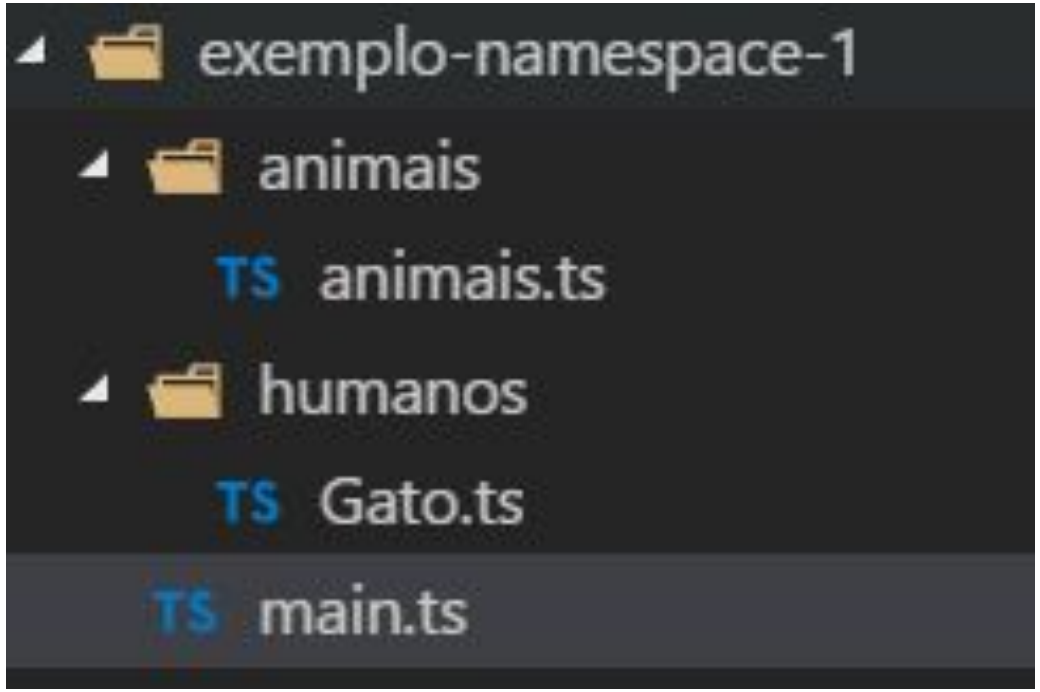
Namespaces

- Declarações podem ser agrupadas em *namespaces*;
- Funciona de maneira semelhante a *packages* e *namespaces* em outras linguagens de programação;
- Isso é útil para evitar colisão de nomes em diferentes contextos;

Principais características

Namespaces

- Exemplo:



Principais características

Modularização

- Exemplo:

animais.ts

```ts
export namespace Animais {
    export class Cachorro {
        latir(): void {
            console.log("Au! Au!");
        }
    }

    export class Gato {
        constructor(){
            this.miar();
        }

        miar(): void {
            console.log("Miau! Miau!");
        }
    }
}
```

Gato.ts

```ts
export class Gato  {
    constructor() {
        this.apresentacao();
    }

    apresentacao(): void {
        console.log("Olá! Eu sou um cara gato!");
    }
}
```

Principais características

Namespaces

- Exemplo:

main.ts

```ts
import { Gato } from './humanos/gato';
import { Animais } from "./animais/animais";

let felino = new Animais.Gato();
let humano_bonito = new Gato();
```

Principais características

Namespaces

- Exemplo:

main.ts

```ts
// Forma alternativa
import * as animais from "./animais/animais";
import { Gato } from './humanos/gato';

let felino = new animais.Animais.Gato();
let humano_bonito = new Gato();
```

TypeScript vs Javascript

Mãos à obra!

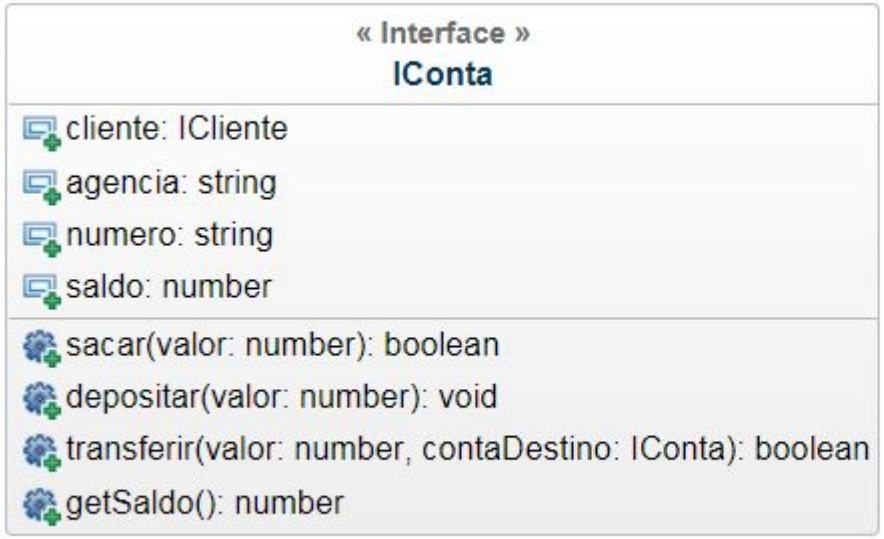
- Um bom e velho exercício de OO:
  - Crie um programa simples com Typescript que realize operações bancárias básicas em duas contas correntes;
  - Você deve utilizar o máximo possível dos elementos do TS apresentados nesta aula.

Mãos à obra!

- Um bom e velho exercício de OO:
  - Operações que devem ser realizadas:
    - Depósito, saque, transferência e verificação de saldo;
    - Ambas as contas do mesmo cliente;

Mãos à obra!

- Diagramas das Interfaces:



Referências

Referências

- TypeScript. **Documentation**. Disponível em https://www.typescriptlang.org/docs/home.html Acesso em 07/10/2017.
- TypeScript. **Modules.** Disponível em https://www.typescriptlang.org/docs/handbook/modules.html Acesso em: 07/10/2017.
- TypeScript. **Namespace and modules**. Disponível em https://www.typescriptlang.org/docs/handbook/namespaces-and-modules.html Acesso em: 07/10/2017.
- TypeScript. **TypeScript in 5 minutes**. Disponível em https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html Acesso em: 07/10/2017.

Eder Martins Franco

eder.franco@fpf.br
efranco23@gmail.com

facebook.com/efranco23
linkedin.com/in/efranco23
moodle.franco.eti.br