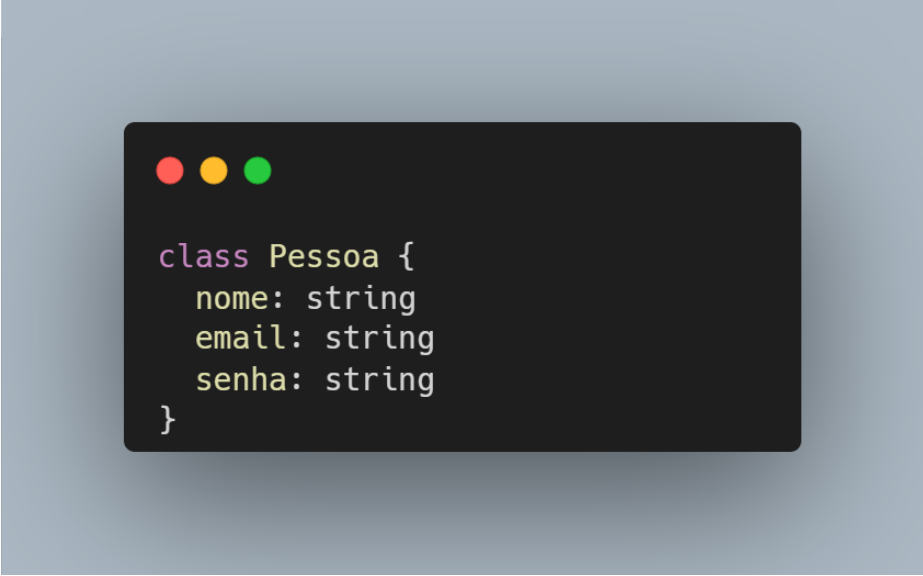


Classes no Typescript

Ao trabalhar com o Angular, você verá que praticamente todas as partes de um projeto Angular são classes e objetos. Por isso, é necessário ter, pelo menos, um conhecimento básico de orientação a objetos. Então, iremos ver como funcionam as classes no Typescript.

Para iniciarmos, vamos criar uma classe chamada Pessoa, com as propriedades **nome**, **email** e **senha**. A sintaxe do Typescript para criar essa classe, seria



```
class Pessoa {  
  nome: string  
  email: string  
  senha: string  
}
```

Nessa classe, definimos as propriedades que ela vai ter e seus tipos. Nesse caso, todos os tipos são strings. No Javascript, criar uma classe apenas assim é possível. No entanto, no Typescript, uma classe criada apenas como está no exemplo acima dará erro, pois as propriedades não são inicializadas, ou seja, não têm um valor definido, em nenhum momento da classe. Para resolvermos isso, podemos fazer de três maneiras: A primeira é criando a função construtora, a função que cria o objeto da classe. Para criarmos essa função, devemos usar um método reservado chamado **constructor()**. Esse método é especial para as classes e serve para criar o objeto em si. Esse método pode receber parâmetros para inserirmos os valores informados às propriedades.

```
class Pessoa {
  nome: string
  email: string
  senha: string

  constructor(nome: string, email: string, senha: string) {
    this.nome = nome
    this.email = email
    this.senha = senha
  }
}
```

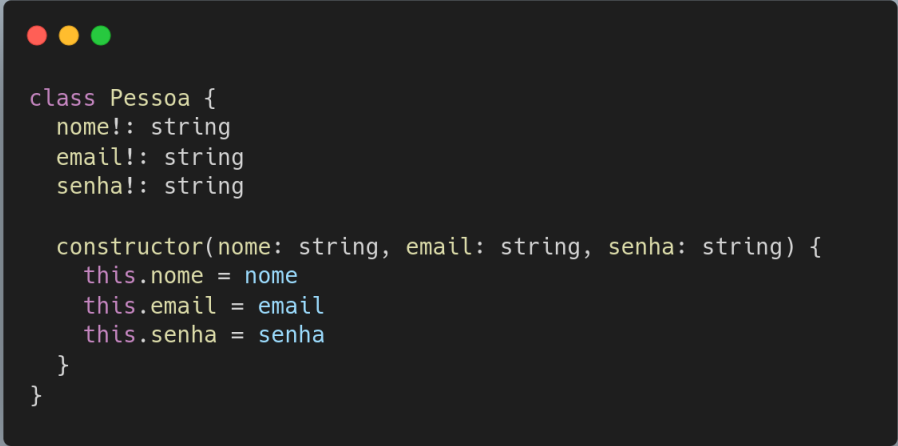
Dessa maneira, quando um objeto Pessoa for criado, o Typescript obrigará que os valores das propriedades sejam informados na sua criação.

A segunda maneira é já inicializando as propriedades com valores padrão. Assim, o compilador do Typescript também não acusará erro.

```
class Pessoa {
  nome: string = 'João'
  email: string = 'joão@mail.com'
  senha: string = 'minhasenha'
}
```

Com essa maneira, o Typescript também irá parar de acusar erro.

A última maneira é fazendo com que as variáveis recebam o valor **undefined** por padrão, no entanto, dessa maneira, você precisará implementar o **constructor()** para inicializar as variáveis com os valores certos, como no exemplo abaixo



```
class Pessoa {
  nome!: string
  email!: string
  senha!: string

  constructor(nome: string, email: string, senha: string) {
    this.nome = nome
    this.email = email
    this.senha = senha
  }
}
```

colocando o ponto de exclamação (!) logo após o nome da propriedade e de sua tipagem, o Typescript infere o valor **undefined** à variável, não informando mais o erro. No entanto, será necessário que na instanciação do objeto os valores reais sejam informados, para que não haja problemas futuros na aplicação. Vale ressaltar que, independentemente da situação, sempre é bom implementar o **constructor()**

Modificadores de Acesso

Em outras linguagens fortemente tipadas que possuem o paradigma de orientação a objetos, elas possuem os **modificadores de acesso**. Esses modificadores de acesso informam se uma propriedade ou método são acessíveis ou não em qualquer momento da aplicação. Os modificadores de acesso permitem com que a segurança da aplicação aumente, impedindo que qualquer lugar da aplicação acesse informações sensíveis de alguma classe de qualquer maneira. No Typescript, temos três tipos de modificadores de acesso: **public** (padrão), **private** e **protected**

O modificador de acesso **public** faz com que qualquer propriedade ou método da classe possa ser acessado diretamente em qualquer momento da aplicação, como no exemplo abaixo

```
class Pessoa {  
  public nome: string  
  public email: string  
  public senha: string  
  
  constructor(nome: string, email: string, senha: string) {  
    this.nome = nome  
    this.email = email  
    this.senha = senha  
  }  
  
  falar(): void {  
    console.log(`Olá! Meu nome '${nome}'`)  
  }  
}  
  
let p = new Pessoa('Paulo', 'paulo@mail.com', '123456789')  
  
console.log('email: ', p.email)  
  
p.falar()
```

O modificador **public** é colocado por padrão em qualquer propriedade ou método mesmo que não seja explicitado no código.

O modificador de acesso **private** faz com que algum atributo ou método não possa ser acessado em qualquer lugar da aplicação.

```
class Pessoa {
  private nome: string
  private email: string
  private senha: string

  constructor(nome: string, email: string, senha: string) {
    this.nome = nome
    this.email = email
    this.senha = senha
  }

  private falar(): void {
    console.log(`Olá! Meu nome '${nome}'`)
  }
}

let p = new Pessoa('Paulo', 'paulo@mail.com', '123456789')

console.log('email: ', p.email) // erro

p.falar() // erro
```

“E como eu faria para acessar esses valores caso eu precisasse?” Para isso, usamos os métodos **getters** e **setters**. Os métodos getters retornam os valores das propriedades, enquanto os setters permitem vocês alterar o valor de uma propriedade. Fazer isso impede com que as propriedades fiquem protegidas, deixando as aplicações mais consistentes. Essa estratégia, na orientação a objetos, chama-se **encapsulamento**.

```
class Pessoa {
  private nome: string
  private email: string
  private senha: string

  constructor(nome: string, email: string, senha: string) {
    this.nome = nome
    this.email = email
    this.senha = senha
  }

  private falar(): void {
    console.log(`Olá! Meu nome '${nome}'`)
  }

  getNome(): string {
    return this.nome
  }

  setNome(nome: string): string {
    this.nome = nome
  }
}

let p = new Pessoa('Paulo', 'paulo@mail.com', '123456789')

console.log('nome: ', p.getNome()) // nome: Paulo

p.setNome('Pedro')

console.log('nome: ', p.getNome()) // nome: Pedro
```

O último modificador de acesso é o **protected**. O protected possui um comportamento semelhante ao do **private**, no entanto, quando uma propriedade de uma classe possui o modificador protected, as subclasses da superclasse possuem acesso direto à propriedade. Para acessar as propriedades com protected dentro de uma subclasse, utilize a palavra chave **super**.

```
class Pessoa {
  protected nome: string
  protected email: string
  protected senha: string

  constructor(
    nome: string,
    email: string,
    senha: string,
  ) {
    this.nome = nome
    this.email = email
    this.senha = senha
  }
}

let p = new Pessoa('Paulo', 'paulo@mail.com', '123456789')

console.log('nome: ' + p.nome) // erro

class PessoaB extends Pessoa {
  constructor() {
    console.log('nome: ', super.nome) // funciona
  }
}
```

Facilitando a instanciação de classes

Como vimos anteriormente, para instanciar uma classe com valores nas propriedades, precisamos declarar as propriedades no topo da classe, criar o construtor que recebe os valores que são passados às propriedades da classe. Isso, dependendo de quantas propriedades tem sua classe, pode se tornar altamente verboso.

```

class Pessoa {
  private nome: string
  private email: string
  private senha: string
  private username: string
  private telefone: string
  private idade: number
  private sexo: string

  constructor(
    nome: string,
    email: string,
    senha: string,
    username: string,
    telefone: string,
    idade: number,
    sexo: string
  ) {
    this.nome = nome
    this.email = email
    this.senha = senha
    this.username = username
    this.telefone = telefone
    this.idade = idade
    this.sexo = sexo
  }
}

```

Como pode ver, a classe fica altamente grande e verbosa e, com certeza, fica difícil de fazer manutenção neste código. Por isso, o Typescript possui uma sintaxe mais fácil para fazer essa parte da classe. Basta que, nos parâmetros do construtor, você coloque algum modificador de acesso (public, private ou protected), dessa maneira:

```

class Pessoa {

  constructor(
    public nome: string,
    public email: string,
    public senha: string,
    public username: string,
    public telefone: string,
    public idade: number,
    public sexo: string
  ) {}
}

```


Como pode ver, as propriedades não foram declaradas antes do construtor e os parâmetros possuem modificadores de acesso antes de seu nome. Assim, o Typescript interpreta cada parâmetro do construtor da classe como uma propriedade da classe, diminuindo a quantidade de linhas de código na classe.