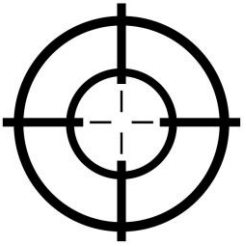


Paradigma de
programação orientada
à objetos, classes e
objetos

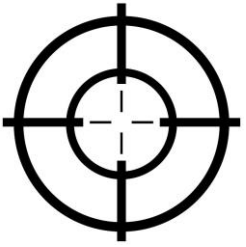
O que é um paradigma de programação?



Paradigmas de programação são uma das formas de classificar as linguagens de programação com base em seus recursos. As linguagens podem ser classificadas em vários paradigmas.

Contudo, é possível separar os paradigmas em dois grandes grupos. Eles são chamados de imperativo e declarativo. O paradigma de programação orientada à objetos faz parte do grupo imperativo.

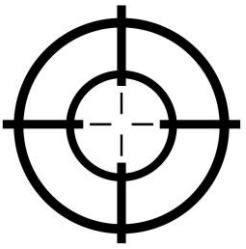
Grupo imperativo



O grupo imperativo possui como característica principal a necessidade do programador instruir a máquina como mudar seu estado e/ou comportamento.

Isto significa que cabe ao desenvolvedor definir as instruções que deverão ser executadas e a ordem dessas execuções. Geralmente aprende-se a programar em alguma linguagem que se encaixa no grupo imperativo.

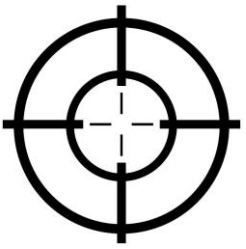
O que é programação orientada à objetos?



A programação orientada à objetos é um paradigma no qual os objetos do mundo real são vistos como entidades separadas com seu próprio estado, que é modificado apenas por procedimentos internos, chamados métodos.

A criação e organização de objetos acontece a partir de classes, que são definidas pelo desenvolvedor.

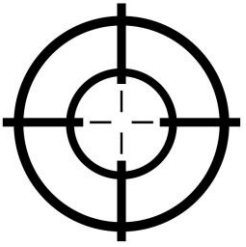
O que é programação orientada à objetos?



A programação orientada à objetos é uma metodologia, que é bioinspirada com base em objetos. Estes objetos constituem seus blocos de construção fundamentais, em oposição à programação procedural que é baseada em procedimentos.

Na programação orientada à objetos, estruturas de dados, ou objetos são definidos, cada um com suas próprias propriedades ou atributos. Cada objeto também pode conter seus próprios procedimentos ou métodos. O software é projetado usando objetos que interagem uns com os outros.

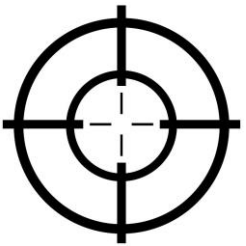
O problema do JavaScript...



JavaScript é uma das linguagens de programação mais difundidas já criadas. Cresceu e se tornou uma linguagem utilizada no desenvolvimento de aplicativos de front-end e back-end de todos os tamanhos.

Embora o tamanho, o escopo e a complexidade dos programas escritos em JavaScript tenham crescido exponencialmente, a capacidade da linguagem JavaScript de expressar os relacionamentos entre diferentes unidades de código não aumentou.

Erros mais comuns na programação JavaScript...



Os erros mais comuns durante a programação são provocados pelo uso equivocado de tipos. Por exemplo, quando uma variável de um determinado tipo é usada para armazenar um valor de um tipo diferente.

Este tipo de problema também ocorre durante a passagem de valor nos argumentos em uma função. Durante o desenvolvimento, a linguagem JavaScript não faz uma verificação do valor ou do tipo dos argumentos nas funções.

Qual é o objetivo do TypeScript?



O objetivo do TypeScript é ser um verificador de tipo estático para programas JavaScript, em outras palavras, uma ferramenta que garante verificação de tipo. Lembre-se, todo código JavaScript também é código TypeScript, inclusive, as linguagens compartilham sintaxe.

Além da verificação, TypeScript possui recursos que se sobrepõem as abordagens do JavaScript. Um dos principais recursos é a possibilidade de desenvolvimento com o paradigma de programação orientada à objetos.

Tipos primitivos

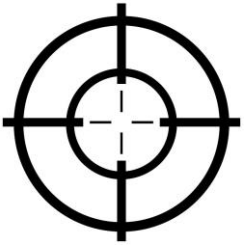


Tipos primitivos são tipos de dados básicos, fornecidos pela linguagem de programação. Em JavaScript, um tipo primitivo corresponde a um tipo de dado que não é, necessariamente, um objeto. Existem 7 tipos de dados primitivos: string, number, bigint, boolean, undefined, symbol e null.

```
let texto = 'string'  
let numero = 12.5  
let boleano = true
```

Os tipos string, number e boolean são comumente usados. Estes tipos são correspondentes no TypeScript. A partir dos tipos primitivos pode-se criar outros tipos, compostos e mais complexos, como objetos.

Definindo tipos no TypeScript...



Ao declarar uma variável usando `const`, `var` ou `let`, pode-se opcionalmente adicionar uma anotação de tipo para especificar explicitamente o tipo da variável.

Declaração da variável

Anotação do tipo

```
let texto: string = 'string'  
let numero: number = 12.5  
let boleano: boolean = true
```

A diagram illustrating TypeScript variable declarations. Three lines of code are shown: 'let texto: string = \'string\'', 'let numero: number = 12.5', and 'let boleano: boolean = true'. Brackets are used to group parts of the code. A bracket under 'let' is labeled 'Declaração da variável'. A bracket under ': string' is labeled 'Anotação do tipo'. A bracket under the entire first line is also labeled 'Anotação do tipo'. A bracket under the entire second line is also labeled 'Anotação do tipo'. A bracket under the entire third line is also labeled 'Anotação do tipo'.

Em TypeScript não utiliza-se anotações de tipo à esquerda, como `int x = 0;`; anotações de tipo, sempre irão após o nome da variável.

A definição de tipo não é obrigatória...



Sempre que possível, o compilador TypeScript tenta inferir automaticamente os tipos no código. Por exemplo, o tipo de uma variável pode ser inferido com base no seu valor.

```
let texto: string = 'string'  
let numero: number = 12.5  
let boleano: boolean = true
```

```
let texto = 'string'  
let numero = 12.5  
let boleano = true
```

Para a linguagem TypeScript, esses códigos são equivalentes.

Tipo indefinido...



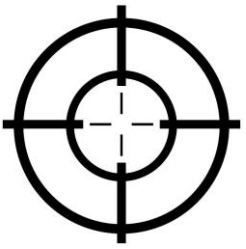
TypeScript também possui um tipo especial, “any”, que pode ser usado quando não se quiser atribuir um tipo explicitamente ou quando não se quiser colocar “certeza” no tipo da variável.

Quando não se especifica um tipo e o compilador TypeScript não pode inferir a partir do contexto, então o compilador normalmente assume como padrão any.

```
let cumprimento = (mensagem:any) =>{  
  console.log(`Bem-vindo: ${mensagem}`)  
}
```

```
let cumprimento = (mensagem) =>{  
  console.log(`Bem-vindo: ${mensagem}`)  
}
```

Definindo o tipo de retorno de uma função...



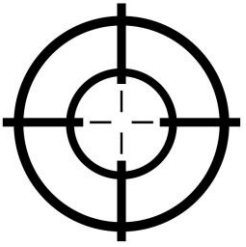
TypeScript permite a especificação dos tipos de valores de entrada e saída das funções.

```
let cumprimento = (mensagem): string => {  
    return `Bem-vindo: ${mensagem}`  
}
```

As anotações de tipo de retorno aparecem após a lista de parâmetros.

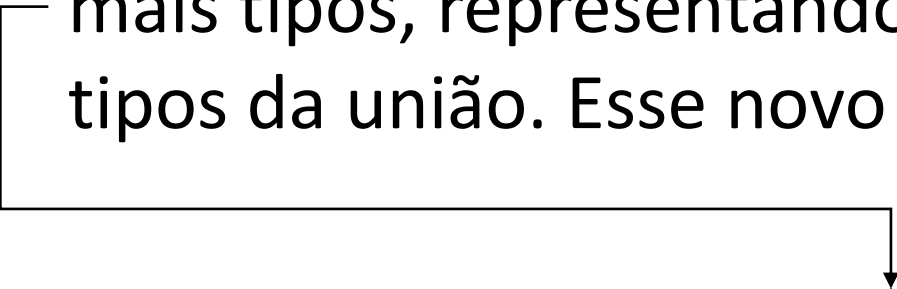
Assim como com variáveis, também ocorre a inferência do tipo de retorno, se ele não for definido explicitamente.

União de tipos



O sistema de tipos do TypeScript permite a combinação de tipos, conceito denominado de união de tipos.

A união de tipos cria um novo tipo, que é formado a partir de dois ou mais tipos, representando valores que podem ser de qualquer um dos tipos da união. Esse novo tipo é referido como união de membros.



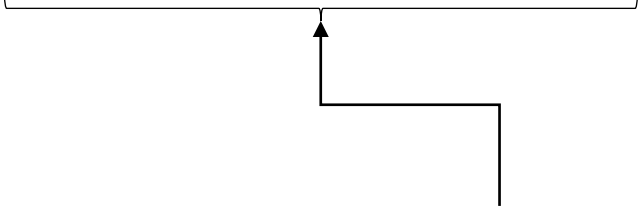
```
let mensagem = (informacao: string | number): string =>{  
    return `Informação: ${informacao}`  
}
```

Sobre a união de tipos...



O compilador TypeScript só permitirá que se faça coisas com união de membros se isso for válido para todos os tipos usados na união.

```
let mensagem = (informacao: string | number): string =>{  
  | return `Informação: ${informacao.toUpperCase()}`  
}
```



Por exemplo, para a união dos tipos string e number, não pode-se usar métodos que estão disponíveis apenas em string.

Definindo tipos com objetos...



Pode-se definir tipos com objetos, que são usados como apelidos para dados complexos.

Lembre-se do objetivo do TypeScript. Criar um novo tipo não é obrigatório, mas pode ser conveniente.

```
type Telefone = {  
  ddd:string  
  numero:string  
}
```

```
let mensagem = (informacao: Telefone): string =>{  
  return `ddd: ${informacao.ddd} numero: ${informacao.numero}`  
}
```


Verificando tipos...

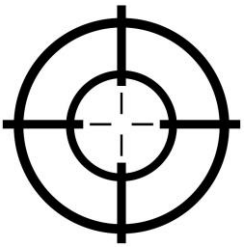


O interpretador JavaScript oferece suporte a um operador de tipo, o “typeof”. Este operador pode fornecer informações muito básicas sobre o tipo do valor de uma variável, em tempo de execução. Este operador também está disponível para TypeScript.

```
let mensagem = (informacao: string | number): string => {  
  if (typeof informacao === 'string') {  
    return `informação recebida: ${informacao.toUpperCase()}`  
  } else {  
    return `informação recebida: ${informacao}`  
  }  
}
```

O typeof compara os tipos pelo seu nome.

Como verificar igualdade de tipos primitivos?

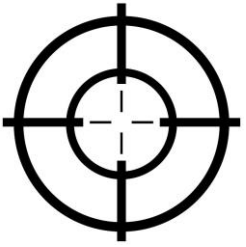


Importante! Em JavaScript existem dois tipos de operadores de igualdade, “==” e “===”.

O operador “==” compara a igualdade de valores, já o operador “===” compara a igualdade de tipo e valor.

O typeof consegue descobrir se uma variável possui algum dos tipos: “string”, “number”, “bigint”, “boolean”, “symbol”, “undefined”, “object”, “function”.

O que é um objeto?

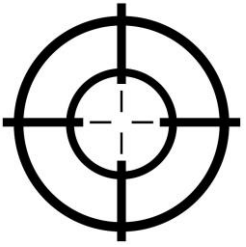


Um objeto é um elemento (instância) de uma classe. Objetos têm os comportamentos de sua classe. O objeto é o componente real dos programas, enquanto a classe especifica como as instâncias são criadas e como se comportam.

Um objeto pode ser uma variável, uma estrutura de dados, uma função ou um método e, como tal, é um valor na memória referenciado por um identificador.

Um objeto pode ser uma combinação de variáveis, funções e estruturas de dados.

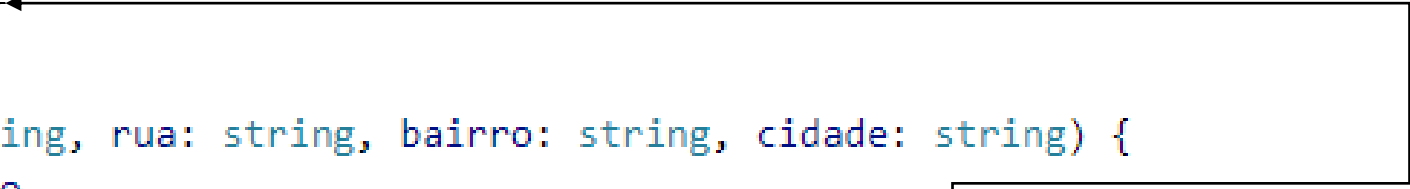
O que é uma classe?



Uma classe é uma forma de definir (declarar) um tipo especial de dado - tipo que não seja simples como um primitivo.

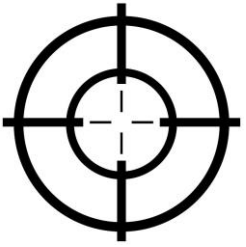
Como analogia: assim como um engenheiro desenha a planta baixa de um imóvel, a classe é a planta baixa do objeto.

```
class Endereco {  
    private numero: string  
    private rua: string  
    private bairro: string  
    private cidade: string  
    constructor(numero: string, rua: string, bairro: string, cidade: string) {  
        this.numero = numero  
        this.rua = rua  
        this.bairro = bairro  
        this.cidade = cidade  
    }  
}
```



Exemplo: Esta classe defini como será um objeto do tipo endereço. Quais dados ele irá possuir e qual o tipo desses dados.

O que é um atributo?



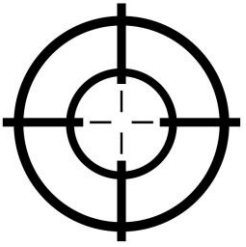
Importante! Os dados armazenados são chamados de atributos e eles não precisam ser do mesmo tipo.

No desenvolvimento com o paradigma de programação orientada à objetos é comum chamar os atributos de campos ou membros de classe.

```
class Endereco {  
    public numero: number  
    public rua: string  
    public bairro: string  
    public cidade: string  
}
```

A diagram consisting of a vertical line on the left, a horizontal line at the top, and a horizontal line at the bottom. An arrow points from the top horizontal line to the word 'atributos' in the text above. Another arrow points from the bottom horizontal line to the opening curly brace of the class definition in the code block.

O que é um método?



Importante! Toda classe pode ter comportamentos, algo que ela faz. Isto é definidos por métodos.

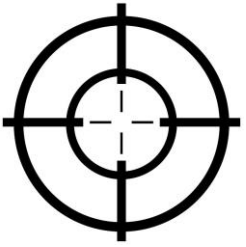
Métodos são funções definidas na classe, que todos os seus objetos terão.

Dentro do corpo de um método, o acesso a campos deve ser via palavra-chave “this”.

```
class Endereco {  
    public numero: number  
    public rua: string  
    public bairro: string  
    public cidade: string  
  
    public mostrarEndereco(){  
        return `Cidade: ${this.cidade}, bairro: ${this.bairro}, rua: ${this.rua}, número: ${this.numero}`  
    }  
}
```

A diagram illustrating the use of the 'this' keyword. A bracket on the left groups the four public fields (numero, rua, bairro, cidade) of the Endereco class. An arrow points from this bracket to the 'this' keyword in the return statement of the 'mostrarEndereco()' method. Another bracket on the right groups the four fields accessed in the return statement (this.cidade, this.bairro, this.rua, this.numero). An arrow points from the text 'Dentro do corpo de um método, o acesso a campos deve ser via palavra-chave “this”.' to this bracket.

Como criar objetos?



A partir de uma classe pode-se criar quantos objetos se desejar. Todos irão conter os mesmos atributos e comportamentos definidos na classe.

```
let endereco = new Endereco(123, 'Av. Paulista', 'Jardim Paulista', 'São Paulo')
let endereco2 = new Endereco(123, 'Av. 9 de Julho', 'Jardim Apolo', 'São José dos Campos')

console.log(endereco.mostrarEndereco())
console.log(endereco2.mostrarEndereco())
```

Importante! As variáveis que armazenam objetos são especiais, são chamadas de referências. São ponteiros implícitos.

Entendendo referências...

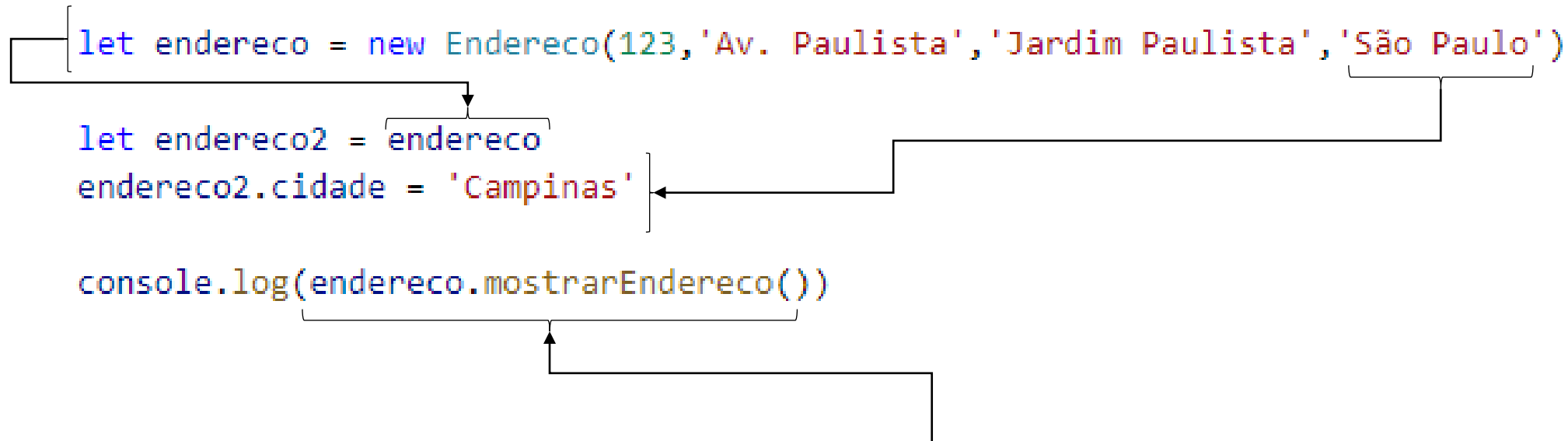


Uma referência se refere a uma instância de uma classe, um objeto. Ao contrário dos tipos primitivos que mantêm seus valores na memória onde a variável é alocada, as referências não mantêm o valor do objeto ao qual se referem.

Como analogia, uma referência é o "controle remoto" para um objeto e seus atributos. A referência não é o objeto em si, ela é o acesso a ele.

Uma referência é uma variável que permite a um programa acessar indiretamente um dado particular, como um outro valor de uma outra variável.

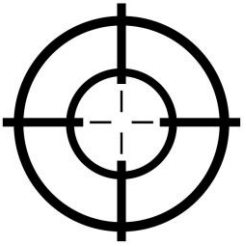
Entendendo referências...



Qual será a saída? Campinas ou São Paulo? Perceba a variável que está em uso, dentro do console.log.

No paradigma de programação orientada à objetos não existe cópia direta dos atributos de um objeto. Acontece cópia de referência, por padrão.

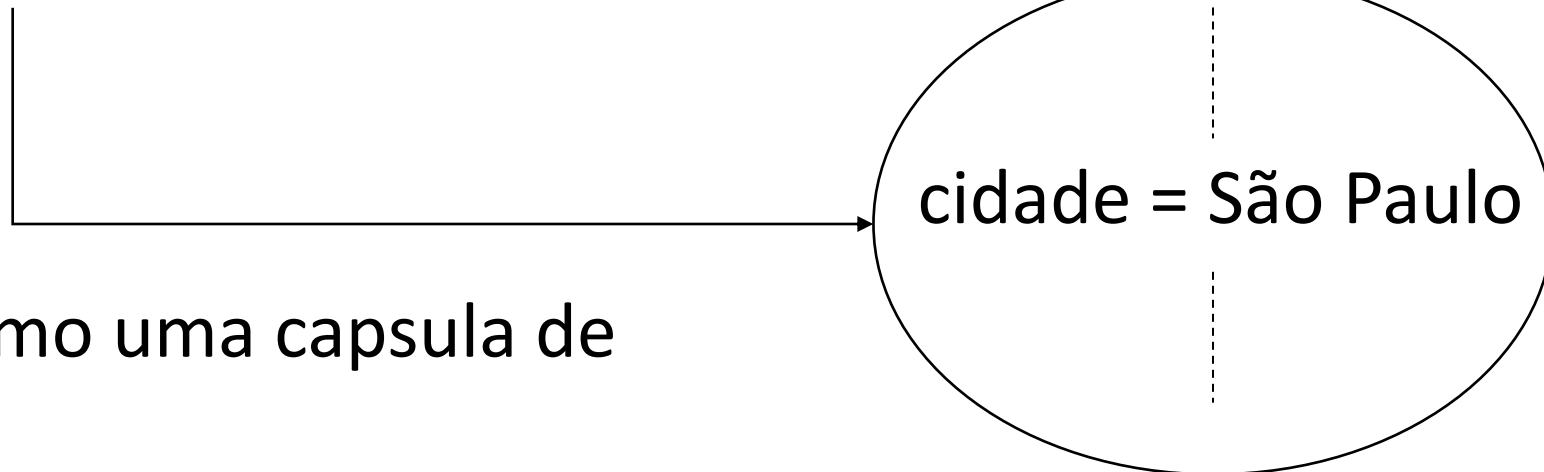
Entendendo referências...



Uma referência se refere a uma instância de uma classe - ou seja um objeto. Ao contrário dos tipos primitivos que mantêm seus valores na memória onde a variável é alocada, as referências não mantêm o valor do objeto ao qual se referem, ela apontam para o endereço do objeto na memória.

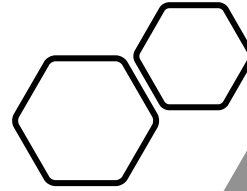
Os objetos ficam, armazenados, no heap de memória

```
let endereco = new Endereco(123, 'Av. Paulista', 'Jardim Paulista', 'São Paulo')
```

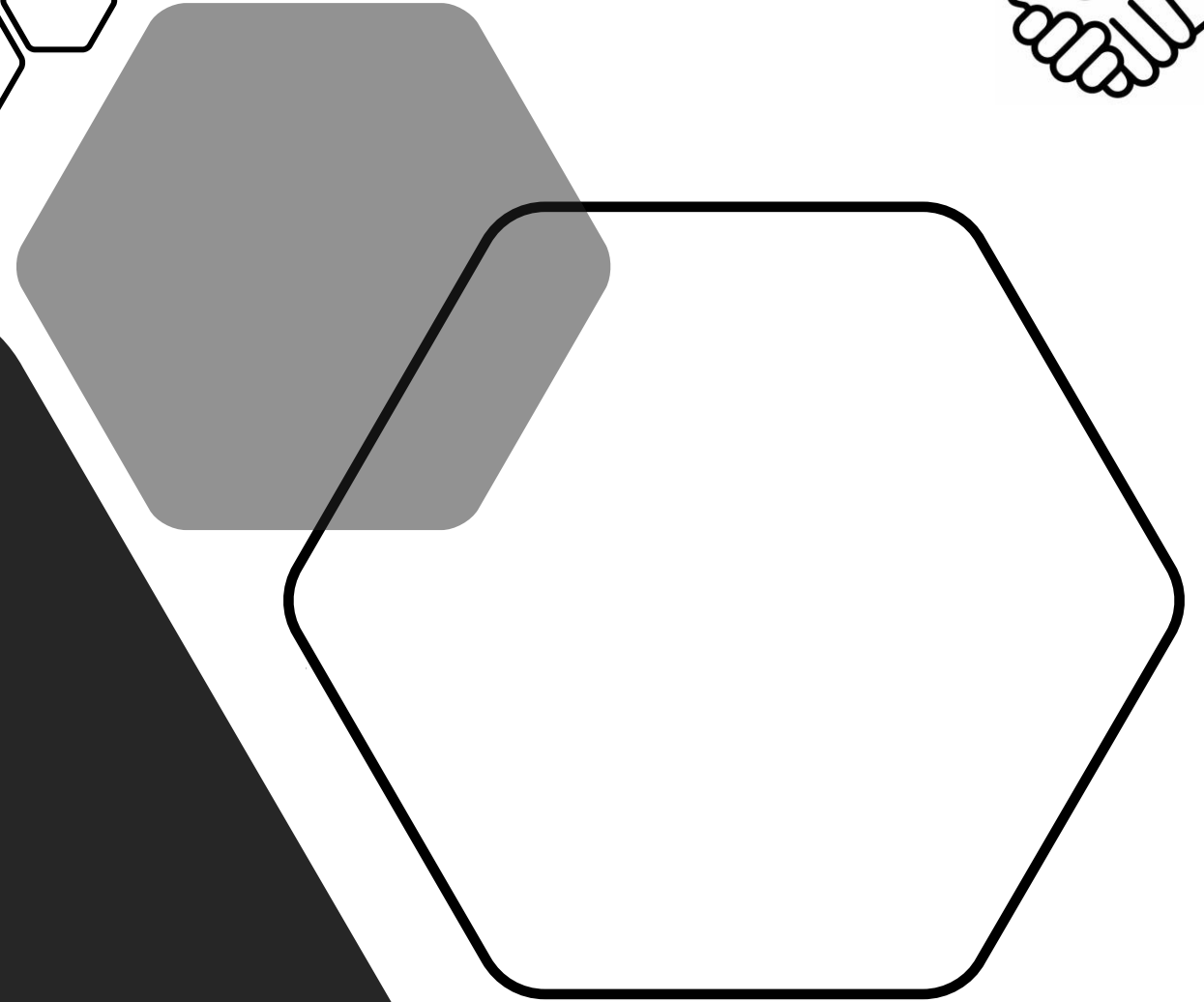


Um objeto na memória, é como uma capsula de valores.

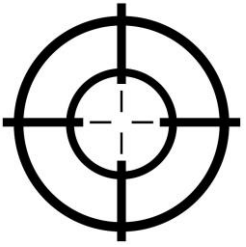
Heap de memória



- Para entender: um heap de memória é um local na memória onde a memória pode ser alocada em acesso aleatório. Ao contrário da pilha em que a memória é alocada e liberada em uma ordem muito definida, os elementos de dados individuais alocados no heap são normalmente liberados de forma assíncrona entre si.



Método construtor



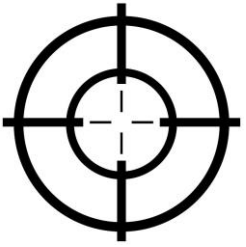
```
class Endereco {  
    private numero: string  
    private rua: string  
    private bairro: string  
    private cidade: string  
    constructor(numero: string, rua: string, bairro: string, cidade: string) {  
        this.numero = numero  
        this.rua = rua  
        this.bairro = bairro  
        this.cidade = cidade  
    }  
}
```

```
let endereco = new Endereco(123, 'Av. Paulista', 'Jardim Paulista', 'São Paulo')  
let endereco2 = new Endereco(123, 'Av. 9 de Julho', 'Jardim Apolo', 'São José dos Campos')
```

A diagram consisting of a horizontal line with a downward-pointing arrow at its right end. This arrow points from the closing curly brace of the 'Endereco' class definition to the 'new Endereco' constructor calls in the code block below, indicating that the class is used to create these objects.

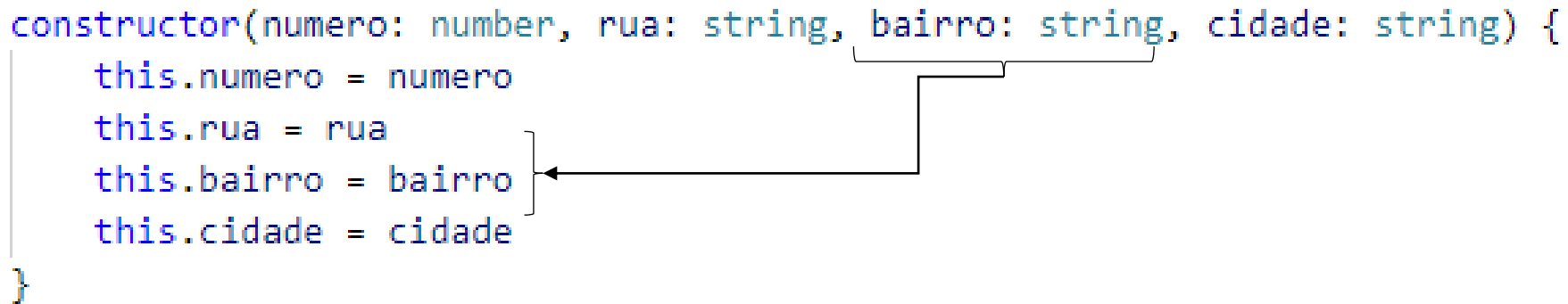
Toda classe possui um método construtor, que é utilizado para criar seus objetos.

O que é o “this”?



A palavra-chave “this” se refere ao próprio objeto (classe) atual em um método, que pode ser construtor ou não.

```
constructor(numero: number, rua: string, bairro: string, cidade: string) {  
    this.numero = numero  
    this.rua = rua  
    this.bairro = bairro  
    this.cidade = cidade  
}
```

A diagram with arrows illustrating the assignment of parameters to class attributes. A bracket groups the parameters 'bairro' and 'cidade' in the constructor signature. An arrow points from this bracket to a bracket grouping 'this.bairro' and 'this.cidade' in the body of the constructor. Another arrow points from the 'bairro' parameter to 'this.bairro'. A third arrow points from the 'cidade' parameter to 'this.cidade'.

O uso mais comum desta palavra-chave é eliminar a confusão entre atributos de classe e parâmetros de métodos com o mesmo nome.

Detalhe importante!

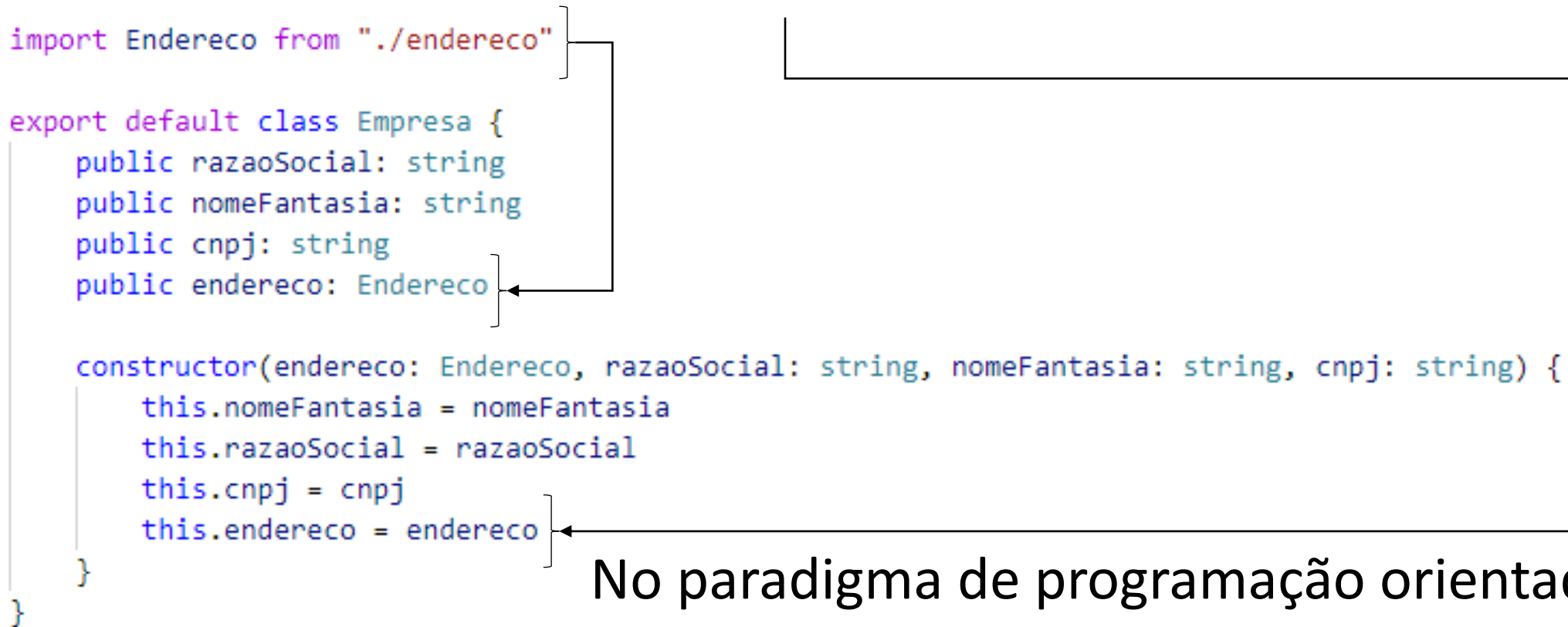


Os atributos de uma classe podem ser outras classes, ou seja, podem armazenar objetos de outras classes.

```
import Endereco from "../endereco"

export default class Empresa {
  public razaoSocial: string
  public nomeFantasia: string
  public cnpj: string
  public endereco: Endereco

  constructor(endereco: Endereco, razaoSocial: string, nomeFantasia: string, cnpj: string) {
    this.nomeFantasia = nomeFantasia
    this.razaoSocial = razaoSocial
    this.cnpj = cnpj
    this.endereco = endereco
  }
}
```

A diagram with two horizontal lines. The top line starts at the 'import Endereco from' statement and ends with an arrow pointing to the 'endereco: Endereco' attribute in the 'Empresa' class. The bottom line starts at the 'this.endereco = endereco' line in the constructor and ends with an arrow pointing to the 'endereco: Endereco' attribute in the constructor's parameter list.

No paradigma de programação orientada à objetos é possível criar componentes reutilizáveis!

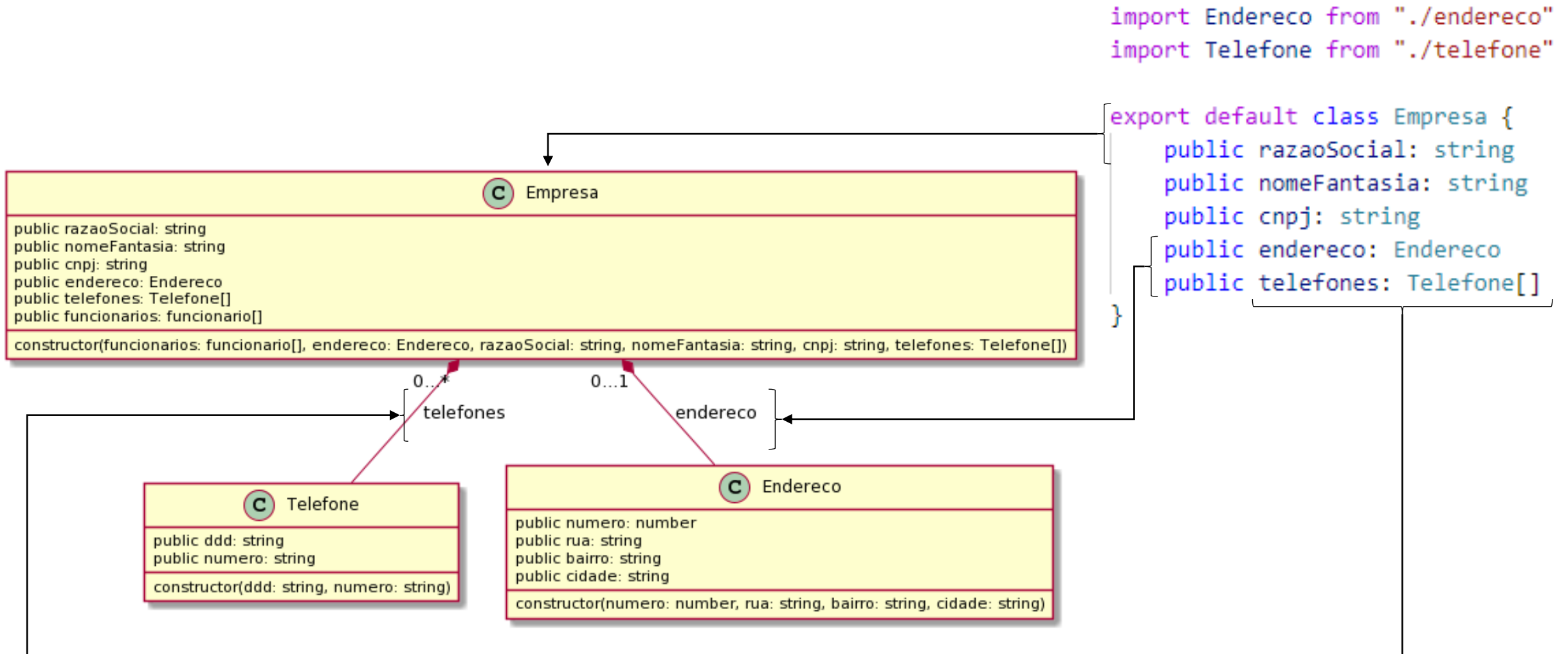
Representação em UML



Usar diagramas UML é uma outra forma de representar classes, seus métodos e atributos.

UML (Unified Modeling Language) é uma linguagem de modelagem usada por desenvolvedores de software. A UML pode ser usada para desenvolver diagramas e fornecer aos usuários (programadores) exemplos de modelagem expressivos prontos para uso.

Identificando as coisas...



Paradigma de
programação
orientada à objetos,
classes e objetos

