Typescript

Construtores, inicialização de atributos e objetos x referências

Ely – ely.miranda@ifpi.edu.br

1

Relembrando...

- Analogia entre as linguagens

C	TypeScript
Struct	Classe
Variável	Objeto
Funções/Procedimentos	Métodos
Alocar memória	Instanciar (criar) objetos a partir de classes

2

Relembrando...

- Classes são modelos, objetos são classes em execução/memória (instanciadas):

Classe Pessoa	Objeto Pessoa
Nome: Texto; Data de Nascimento: Data; Altura: Número;	Nome: Cláudio; Data de Nascimento: 20/05/1978; Altura: 1.6

3

Relembrando...

- Uma classe está para um objeto, assim como:
  - Uma receita está para uma torta;
  - Uma planta está para uma casa.

Classe Pessoa	Objeto Pessoa
Nome: Texto; Data de Nascimento: Data; Altura: Número;	Nome: Cláudio; Data de Nascimento: 20/05/1978; Altura: 1.6

4

Relembrando...

- Para criarmos um objeto, devemos realizar uma instanciação;
- Um objeto é uma instância de uma classe, assim como um banco de dados é uma instância de um esquema.

5

Relembrando...

- Instanciar um objeto é o equivalente a:
  - alocar uma área de memória;
  - atribuirmos a uma variável o endereço dessa área.
- Dizemos que um variável é uma referência para um objeto.

6

Classes

- Em TypeScript são muito semelhantes às das linguagens Java e C#, onde temos atributos, construtores e métodos:

```
class Alo {
    nome: string;
    constructor(nome: string) {
        this.nome = nome;
    }
    dizerAlo(): void {
        console.log("Alô, " + this.nome);
    }
}
```


7

Classes

- Usam como conversão o mesmo padrão CamelCase do Java em suas definições;
- Um construtor usa a palavra reservada constructor;

8

Classes

- Acesso a atributos e métodos dentro da própria classe devem ser feitos usando this;
- Métodos têm as mesmas convenções de funções, sem o uso da palavra reservada function.

9

Classes

- Instanciando e uso de uma classe:

```
let alo = new Alo("Ely");

alo.dizerAlo();
```


10

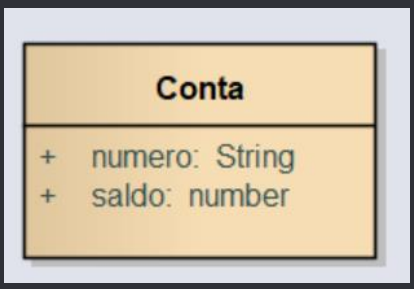
Classes

- Para simplificar a definição de atributos usamos modificadores de visibilidade ou a palavra readonly:

```
class Alo {
    constructor(public nome: string) { }

    dizerAlo() {
        console.log("Alô, " + this.nome);
    }
}
```


Estudo de caso: Sistema Bancário

- Classe com atributos em UML e em TypeScript:




```
class Conta {
    numero: string = "";
    saldo: number = 0;
}
```

Instanciação

- Para instanciar um objeto a palavra reservada new é utilizada;
- A instanciação retorna uma referência para o objeto criado:

```
let conta1 : Conta = new Conta();
```


13

Instanciação

- Objetos que não foram instanciados são nulos;
- Instanciamos um objeto a partir do nome de sua classe e usando um método especial chamado construtor:

```
let conta1 : Conta = new Conta();
```


14

Construtores

- É um “método especial” onde um objeto é inicializado;
- É invocado no momento da instanciação após o operador new.

15

Construtores

- Características sintáticas:
  - Possuem o mesmo nome da classe;
  - Não retornam valor;
  - Podem ter parâmetros.

16

Construtores

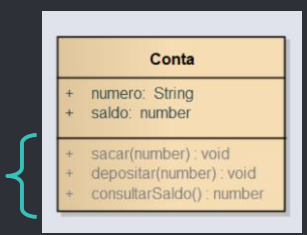
```
class Conta {
    numero: String;
    saldo: number;

    constructor(numero: String, saldo: number) {
        this.numero = numero;
        this.saldo = saldoInicial;
    }
}

let c1: Conta = new Conta("1", 1000);
let c2: Conta = new Conta(); //erro de compilação
```

17

17

Métodos

- Definem os comportamentos de uma classe, ou seja, o que ela faz;
- Todo método deve ter um tipo de retorno, mesmo que seja void.



ely.miranda@ifpi.edu.br

18

18

A classe Conta

```
class Conta {
    numero: String;
    saldo: number;
    //construtor omitido

    sacar(valor: number): void {
        this.saldo = this.saldo - valor;
    }
    depositar(valor: number): void {
        this.saldo = this.saldo + valor;
    }
    consultarSaldo(): number {
        return this.saldo;
    }
}
```


19

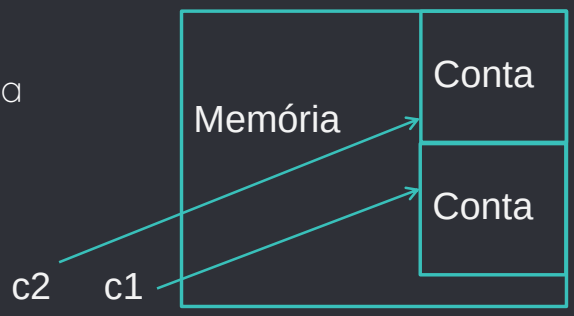
A classe conta

```
let c1 = new Conta("1", 1000);
c1.depositar(10000);
c1.sacar(2000);
console.log(`Novo saldo: ${c1.consultarSaldo()}`);
```


20

Referências de objetos

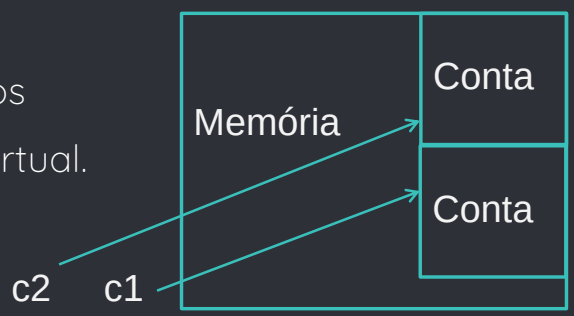
- Não é 100% correto dizer: "c1 é um objeto";
- Uma variável nunca é um objeto, mas uma referência para um objeto;
- Correto: "c1 é uma referência a um objeto do tipo Conta".



21

Referências de objetos

- Por praticidade diz-se: "c1 é um objeto Conta";
- c1 e c2 guardam um número interno que identifica a posição de memória dos seus respectivos objetos Conta na máquina virtual.

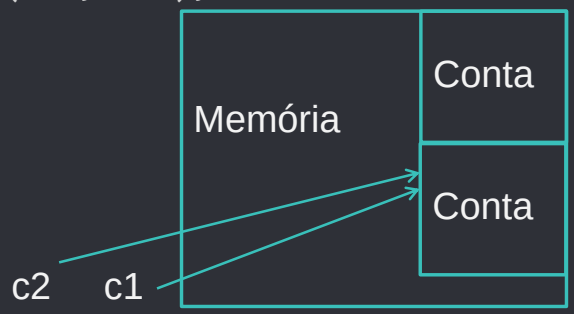


22

Referências de objetos

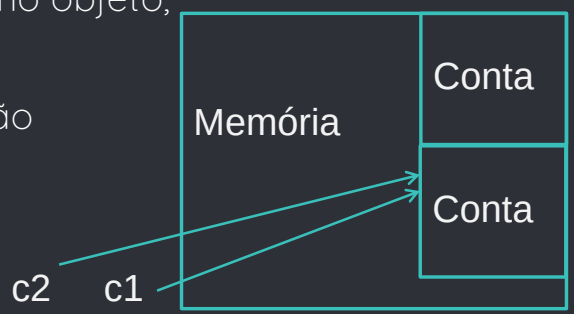
- E se c1 fosse atribuída a c2?

```
let c1 : Conta = new Conta("1",100);
let c2 : Conta = new Conta("2", 200);
c2 = c1;
```



Referências de objetos

- O objeto que era referenciado por c2 é perdido;
- c1 e c2 apontarão para o mesmo endereço em memória ou seja, apontarão para o mesmo objeto;
- Quaisquer alteração em c1 e/ou c2 resultará na alteração de ambos.



Referências de objetos

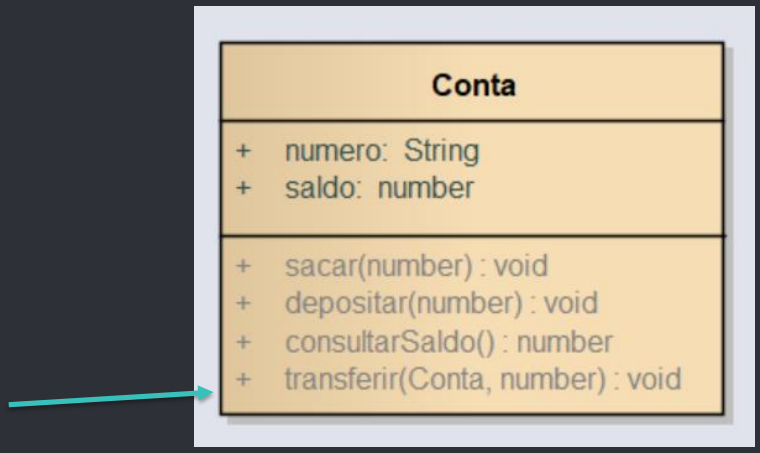
```
let c1 : Conta = new Conta("1",100);
let c2 : Conta = new Conta("2", 200);
c2 = c1;

console.log(c1.consultarSaldo());
console.log(c2.consultarSaldo());
```

Qual o valor de ambos os saldos?

Objetos como parâmetros

- Quando passamos objetos por parâmetro há passagem apenas do endereço de memória;
- Não há cópia de objetos;
- Alterando-se um objeto passado como parâmetro, é o mesmo que alterar o objeto original.

Objetos como parâmetros



27

Objetos como parâmetros

```
class Conta {

    //...
    transferir(contaDestino: Conta, valor: number): void {
        this.sacar(valor);
        contaDestino.depositar(valor);
    }
}
```


28

Objetos como parâmetros

```
let c1 : Conta = new Conta("1", 0);
let c2 : Conta = new Conta("2", 0);
c1.depositar(100);
c2.depositar(200);
c1.transferir(c2, 50);
console.log(c1.consultarSaldo());
console.log(c2.consultarSaldo());
```

Qual o valor de ambos os saldos?

29

Destruição de objetos

- Excluir objetos é uma tarefa onerosa ao programador;
- Alocar implica que o espaço seja desalocado;
- Há grande esforço para controlar desalocação semelhante ao que acontece com ponteiros em C.

30

Destruição de objetos

- As primeiras linguagens O.O. sempre tinham métodos "destrutores" de objetos;
- O programador pode falhar ou ser omisso em desalocar memória;
- Estouros de memória podem ocorrer por não haver mais espaço para alocação de objetos.

31

Garbage Collector

- Em TypeScript e em linguagens modernas não se exclui objetos explicitamente;
- Em um ambiente de uma máquina virtual é comum um serviço de "coleta de lixo" / *Garbage Collector* (GC).

32

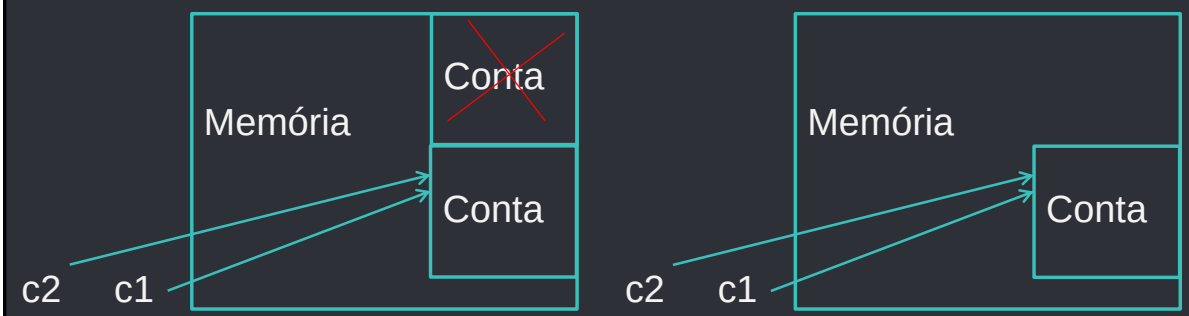
Destruição de objetos

- Frequentemente o GC percorre a memória e exclui objetos sem referência
- Objetos sem referências são objetos que não possuem mais variáveis apontando, como no caso abaixo:

```
let c1 : Conta = new Conta("1",100);
let c2 : Conta = new Conta("2", 200);
c2 = c1;
```


Garbage Collector

- O objeto referenciado por c1 é perdido;
- Futuramente o GC irá excluir esse objeto da memória automaticamente liberando espaço:



Garbage Collector

- Principais vantagens:
  - O programador não precisa se preocupar com desalocação de objetos;
  - Não há estouro de memória por objetos perdidos e não desalocados.

35

Typescript

Construtores, inicialização de atributos e objetos x referências

Ely – ely.miranda@ifpi.edu.br

36