

Programação Orientada a Objetos

Construtores, inicialização de atributos, objetos x referências em Java, Garbage Collection

Ely – elydasilvamiranda@gmail.com

Relembrando...

- Analogia entre as linguagens

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

Relembrando...

- Classes são modelos, objetos são classes em execução/memória (instanciadas);
- Uma classe está para um objeto, assim como:
 - Uma receita está para uma torta;
 - Uma planta está para uma casa;

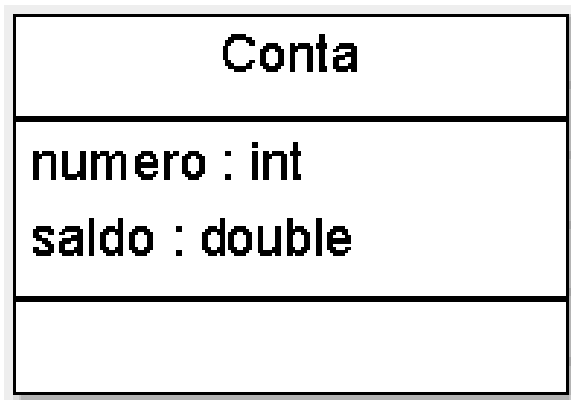
Classe Pessoa	Objeto Pessoa
Nome: Texto; Data de Nascimento: Data; Altura: Número;	Nome: Cláudio; Data de Nascimento: 20/05/1978; Altura: 1.6

Relembrando...

- Para criarmos um objeto, devemos realizar uma instanciação;
- Instanciar um objeto é o equivalente a:
 - alocar uma área de memória;
 - atribuímos a uma variável o endereço dessa área.
- Dizemos que um variável é uma **referência para um objeto**;

Estudo de caso: Sistema Bancário

- Classe em UML e Classe em Java (apenas os atributos:



```
class Conta {  
    int numero;  
    double saldo;  
}
```

Instanciação

- Para instanciar um objeto a palavra reservada **new** é utilizada que retorna uma referência para o objeto criado;
- Em Java, todos os objetos são criados dinamicamente;
- Objetos que não foram instanciados são nulos (null);
- Instanciamos um objeto a partir do nome de sua classe e usando um método especial chamado **construtor**

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

Construtores

- É um “método especial” onde um objeto é inicializado;
- É invocado no momento da instanciação após o operador **new**;
- Características sintáticas:
 - Possuem o mesmo nome da classe;
 - Não retornam valor;
 - Podem ter parâmetros.

Construtores

- Construtor comum:

```
[modificador] <nomeClasse> (<parâmetros>) {  
    <instruções>;  
}
```

– Ex:

```
Conta(String num, double saldoInicial) {  
    numero = num;  
    saldo = saldoInicial;  
}
```


Construtores

- Construtor padrão:
 - É público e sem argumentos;
 - Pode ser omitido;

```
Conta ( ) {  
    . . .  
}
```

Vários construtores

- Podem ser escritos vários construtores para uma classe;
- Isso se chama sobrecarga e será visto futuramente;

```
class Conta {  
    //atributos omitidos  
    Conta(String num) {  
        numero = num;  
    }  
    Conta(String num, double saldoInicial) {  
        numero = num;  
        saldo = saldoInicial  
    }  
}
```

Utilização de construtores

...

```
public static void main(String[] args) {  
    Conta c1 = new Conta("1123");  
    c1.saldo = 200;  
    Conta c2 = new Conta("1124", 200);  
    Conta c3 = new Conta();  
    c3.numero = "1125"  
    c3.saldo = 300;  
    ...  
}
```

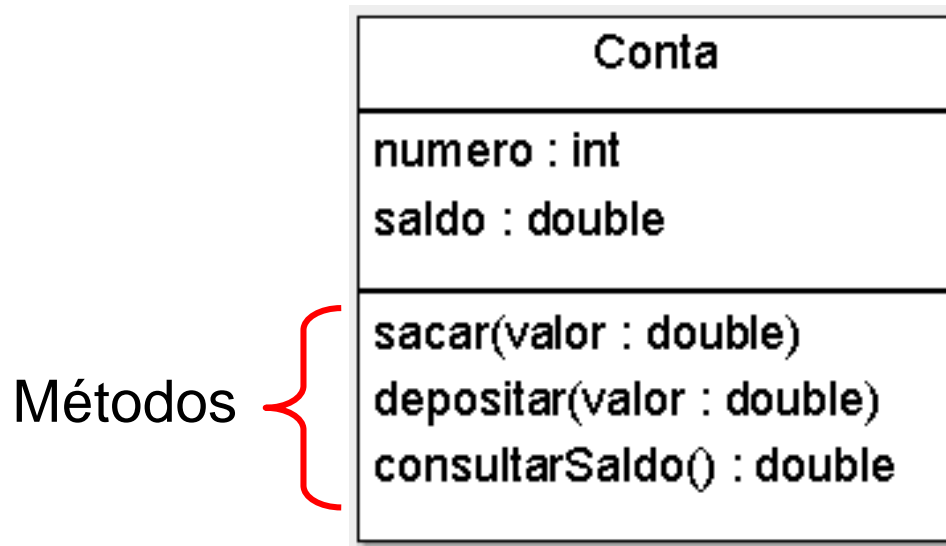
...

Inicialização de atributos

- Como visto anteriormente, variáveis locais devem ser inicializadas obrigatoriamente antes do uso;
- Variáveis locais são distintas dos atributos:
 - Variáveis locais estão declaradas dentro de um método;
 - Os atributos estão declarados no corpo da classe, fora de métodos;
- Atributos não inicializados explicitamente, são inicializados pela JVM:
 - Números recebem valores 0;
 - char ficam vazios;
 - Booleans com false;
 - Outros objetos ficam nulos;

Estudo de caso: Sistema Bancário

- Métodos da classe:
 - Relembrando: definem os comportamentos de uma classe, ou seja, o que ela faz;
 - Todo método deve ter um retorno, mesmo que seja void



Estudo de caso: Sistema Bancário

- A classe conta completa (por enquanto):

```
class Conta {  
    String numero;  
    double saldo;  
    //construtores omitidos  
  
    void sacar(double valor) {  
        saldo = saldo - valor;  
    }  
  
    void depositar(double valor) {  
        saldo = saldo + valor;  
    }  
  
    double consultarSaldo() {  
        return saldo;  
    }  
}
```

Estudo de caso: Sistema Bancário

- Usando a classe, atributos e métodos:

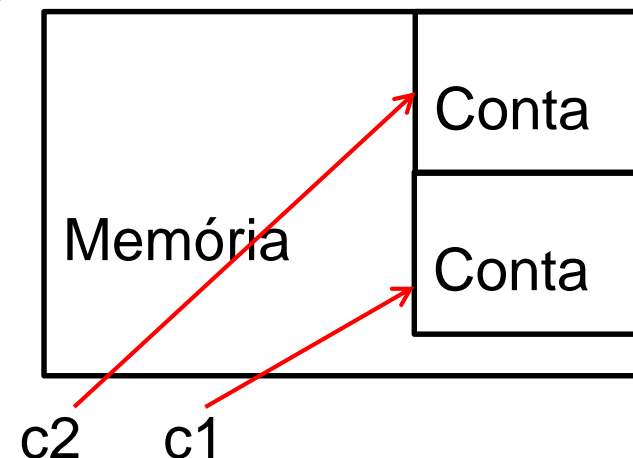
```
public class Programa {  
    public static void main(String[] args) {  
        Conta conta1 = new Conta();  
        conta1.depositar(1000);  
        conta1.sacar(2000);  
        System.out.println("Novo saldo:" +  
                           conta1.consultarSaldo());  
    }  
}
```

Várias instâncias

```
public class TestaDuasContas {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        minhaConta.depositar(1000);  
  
        Conta meuSonho = new Conta();  
        meuSonho.depositar(1500000);  
    }  
}
```


Referências

- 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”**;
 - Por praticidade diz-se: **“c1 é um objeto Conta”**.
 - Ex: c1 e c2 guardam um número que identifica a posição de memória dos seus respectivos objetos Conta



Referências

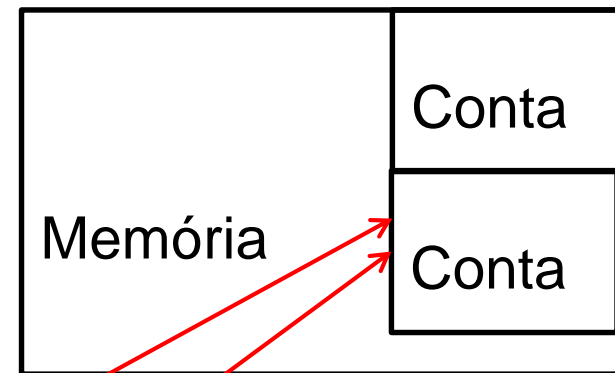
- E se c1 fosse atribuída a c2?

```
Conta c1 = new Conta();
```

```
Conta c2 = new Conta();
```

```
Conta c2 = c1;
```

- 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

```
public class TestaReferencias {  
    public static void main(String args[]) {  
        Conta c1 = new Conta();  
        c1.depositar(100);  
        Conta c2 = new Conta();  
        c2 = c1;  
        c2.depositar(200);  
  
        System.out.println(c1.consultarSaldo());  
        System.out.println(c2.consultarSaldo());  
    }  
}
```

Qual o valor de ambos os saldos?

Referência “this”

- Todo objeto possui uma referência implícita para si mesmo, chamada **this**.
- A referência **this** é válida dentro de todo o corpo de uma classe, até mesmo no construtor.
- Quando um parâmetro tiver o mesmo nome de um atributo, **this** é utilizada para referenciar o atributo.

...

```
void definirSaldoInicial(double saldo) {  
    this.saldo = saldo;  
}
```

...

Referência “this”

- Pode-se usar a palavra reservada `this` para chamar um outro construtor
- Isso deve ser feito na primeira linha do construtor
- Ex:

```
//em uma classe conta...
```

```
Conta() {  
    this("-1", 1.00);  
}
```

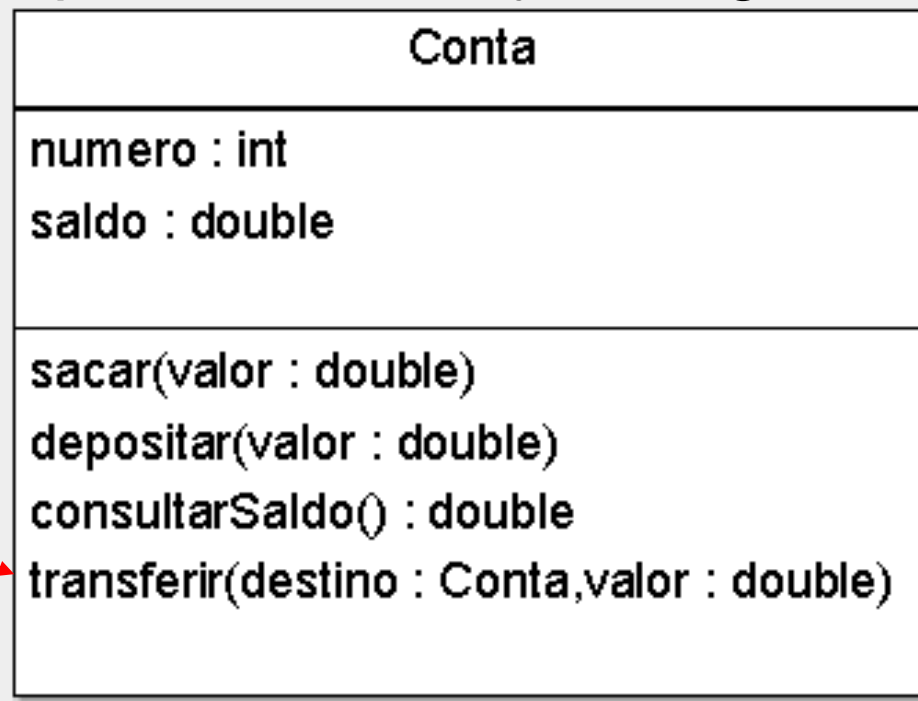
```
Conta(String numero, double saldo) {  
    this.numero = numero;  
    this.saldo = saldo;  
}
```

```
//...em outra classe
```

```
public static void main(String[] args) {  
    Conta c1 = new Conta();  
}
```

Objetos como parâmetros

- Para parâmetros que são objetos:
 - Há a 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

```
public class Conta {  
    // atributos, construtores e metodos omitidos...  
    void transferir(Conta destino, double valor) {  
        this.saldo = this.saldo - valor;  
        destino.saldo = destino.saldo + valor;  
    }  
}
```

Conta
numero : int saldo : double
sacar(valor : double) depositar(valor : double) consultarSaldo() : double transferir(destino : Conta, valor : double)

Objetos como parâmetros

```
public class TestaTransferencias {  
    public static void main(String args[]) {  
        Conta c1 = new Conta();  
        c1.depositar(100);  
        c2.depositar(200);  
        c1.transferir(c2, 50);  
        System.out.println(c1.consultarSaldo());  
        System.out.println(c2.consultarSaldo());  
    }  
}
```

Qual o valor de ambos os saldos?

Destruição de objetos

- Excluir objetos é uma tarefa onerosa ao programador:
 - Alocar exige que o espaço seja desalocado;
 - Grande esforço para controlar desalocação semelhante ao que acontece com ponteiros em C;
 - As primeiras linguagens O.O. sempre tinham métodos “destrutores” de objetos;
 - Em casos de falhas do programador, estouros de memória ocorrem por não haver mais espaço para alocação de objetos;
- Em Java não se exclui objetos explicitamente;

Garbage Collection

- Existe na JVM um serviço de “coleta de lixo” através do *Garbage Collector* (GC):
 - Frequentemente o GC percorre a memória e exclui objetos sem referência (sem variáveis apontando);
- Principais vantagens:
 - O programador não precisa se preocupar com desalocação de objetos;
 - Não há estouro de memória por objetos perdidos e não desalocados;
- O GC é chamado automaticamente, mas pode-se **sugerir** que o sistema o chame:
`System.gc();`

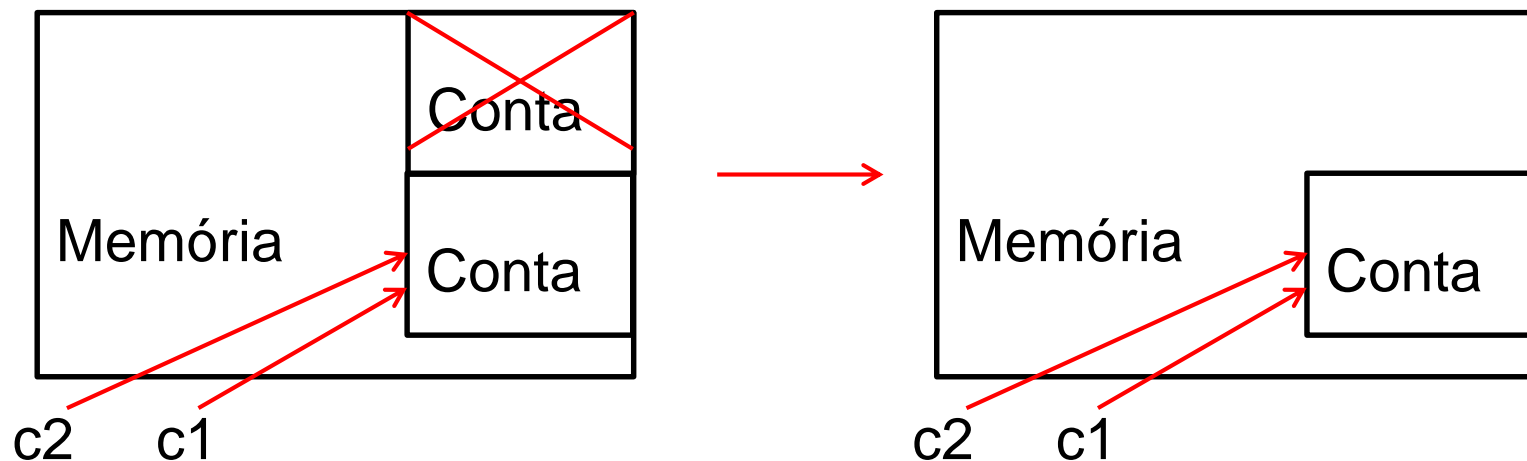
Garbage Collection

- Voltando à atribuição de c1 a c2:

```
Conta c1 = new Conta();
```

```
Conta c2 = c1;
```

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



Programação Orientada a Objetos

Construtores, inicialização de atributos, objetos x referências em Java, Garbage Collection

Ely – elydasilvamiranda@gmail.com