OO – Orientação a Objetos

Métodos com retorno

- Um método **sempre tem** que **definir** o que **retorna**, nem que defina que **não** há **retorno** (**void**);

- Um **método** pode **retornar um valor** para o código que o **chamou**. No caso do nosso método **saca** , podemos **devolver** um valor **booleano** indicando se a operação foi **bem sucedida**.

```
1  class Conta {
2      // ... outros metodos e atributos ...
3
4      boolean saca(double valor) {
5          if (this.saldo < valor) {
6              return false;
7          }
8          else {
9              this.saldo = this.saldo - valor;
10             return true;
11         }
12     }
13 }
```

Métodos com retorno

- A palavra chave **return** indica que o método vai **terminar ali**, retornando tal informação. Exemplo de uso:

```
minhaConta.saldo = 1000;
boolean consegui = minhaConta.saca(2000);
if (consegui) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

- Ou eliminando variáveis temporárias:

```
minhaConta.saldo = 1000;
if (minhaConta.saca(2000)) {
    System.out.println("Consegui sacar");
} else {
    System.out.println("Não consegui sacar");
}
```

Criando dois Objetos

- O programa pode manter na **memória** não apenas **uma** conta, como **mais de uma**;

```
1  class TestaDuasContas {
2      public static void main(String[] args) {
3
4          Conta minhaConta;
5          minhaConta = new Conta();
6          minhaConta.saldo = 1000;
7
8          Conta meuSonho;
9          meuSonho = new Conta();
10         meuSonho.saldo = 1500000;
11     }
12 }
```

Referências de Objetos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Quando declaramos uma **variável** para **associar a um objeto**, na verdade, essa variável **não guarda o objeto**, e sim uma maneira de acessá-lo, chamada de **referência;**

- É por esse motivo que, diferente dos ***tipos primitivos*** como **int** e **long**, precisamos dar **new depois de declarada a variável**;

```
1 public static void main(String args[]) {
2     Conta c1;
3     c1 = new Conta();
4
5     Conta c2;
6     c2 = new Conta();
7 }
```

Referências de Objetos

```
1 public static void main(String args[]) {
2     Conta c1;
3     c1 = new Conta();
4
5     Conta c2;
6     c2 = new Conta();
7 }
```

- O correto aqui, é dizer que **c1** se **refere** a um objeto. **Não é correto** dizer que c1 **é um objeto**, pois c1 é uma **variável referência**;

- Basta lembrar que, em Java, **uma variável nunca é um objeto.**

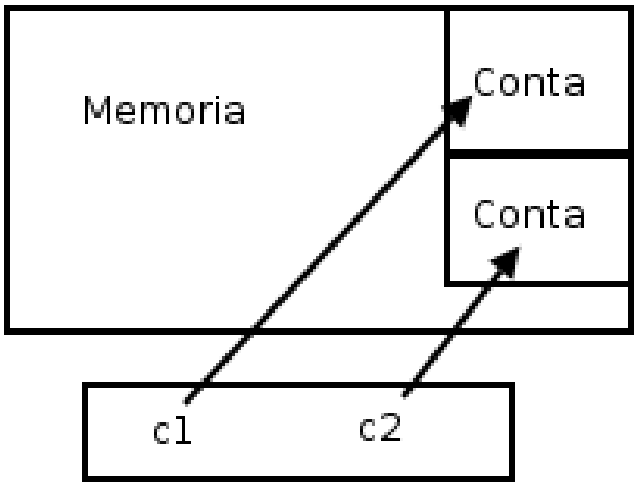
- **Por isso lembrem-se todo objeto** em **Java**, sem **exceção**, é **acessado** por uma **variável referência**.

Referências de Objetos

- Esse código nos deixa na **seguinte situação**;

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

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



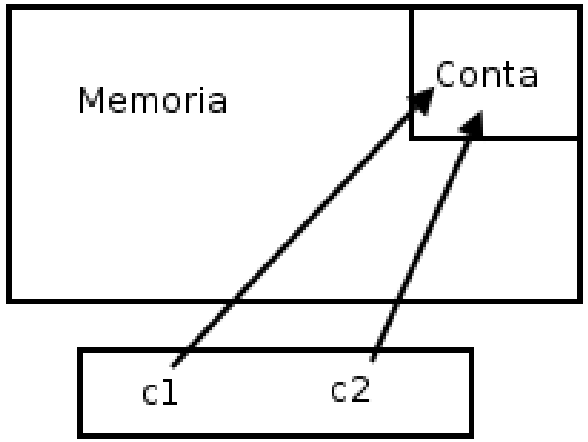
Exemplo

```
1  class TestaReferencias {
2      public static void main(String args[]) {
3          Conta c1 = new Conta();
4          c1.deposita(100);
5
6          Conta c2 = c1;  // linha importante!
7          c2.deposita(200);
8
9          System.out.println(c1.saldo);
10          System.out.println(c2.saldo);
11     }
12 }
```

- **Qual** é o **resultado** do **código** acima? O que **aparece** ao **rodar**?

Na memória, o que acontece nesse caso:

```
Conta c1 = new Conta();
Conta c2 = c1;
```



Parece mas não é

- Quando fizemos **c2 = c1**, c2 passa a fazer **referência** para o mesmo objeto que **c1 referencia** nesse **instante**;
- Então, nesse código em específico, quando utilizamos **c1** ou **c2** estamos nos **referindo** exatamente ao **mesmo objeto**;
- As duas **referências** são **distintas**, porém **apontam** para o **mesmo** objeto! Compará-las com "==" irá nos retornar **true**, pois o valor que elas carregam é o **mesmo**;

O que aconteçe?

```
1  public static void main(String args[]) {
2      Conta c1 = new Conta();
3      c1.dono = "Duke";
4      c1.saldo = 227;
5
6      Conta c2 = new Conta();
7      c2.dono = "Duke";
8      c2.saldo = 227;
9
10     if (c1 == c2) {
11         System.out.println("Contas iguais");
12     }
13 }
```

- O operador **==** compara o **conteúdo das variáveis**, mas essas variáveis **não guardam** o **objeto**, e sim o **endereço** em que ele se **encontra**.

- Quando se trata de **objetos**, pode ficar mais fácil pensar que o **==** compara se os **objetos** (**referências, na verdade**) são o **mesmo**, e **não** se **são iguais**.

Criando o método transfere

- Criaremos um método para **transferir** dinheiro entre **duas contas**;
- Podemos ficar tentados a criar um **método** que recebe **dois parâmetros**: conta1 e conta2 do tipo Conta.
- Mas cuidado: **assim estamos pensando de maneira procedural.**
- A ideia é que, quando chamarmos o método **transfere**, já teremos um **objeto** do tipo **Conta** (o this), portanto o método recebe apenas **um parâmetro do tipo Conta**;

Método Transfere

```
class Conta {

    // atributos e metodos...

    void transfere(Conta destino, double valor) {
        this.saldo = this.saldo - valor;
        destino.saldo = destino.saldo + valor;
    }
}
```

- Para **deixar** o **código** mais **robusto**, poderíamos **verificar** se a **conta** possui a **quantidade** a ser transferida **disponível**;

- Para ficar **ainda** mais **interessante**, você pode **chamar** os **métodos deposita** e **saca** já existentes para **fazer** essa **tarefa**;

Alterando Transfere

```
class Conta {

    // atributos e metodos...

 boolean transfere(Conta destino, double valor) {
    boolean retirou = this.saca(valor);
    if (retirou == false) {
        // não deu pra sacar!
        return false;
    }
    else {
        destino.deposita(valor);
        return true;
    }
 }
}
```

Continuando Atributos

- As variáveis do **tipo atributo**, diferentemente das variáveis **temporárias** (declaradas dentro de um **método**), **recebem** um **valor padrão**.

- No caso **numérico**, valem **0**, no caso de **boolean**, valem **false**.

```
1 class Conta {
2     int numero = 1234;

3     String dono = "Duke";
4     String cpf = "123.456.789-10";
5     double saldo = 1000;
6     double limite = 1000;
7 }
```

- Nesse **caso**, quando você **criar** uma **conta**, seus **atributos** já estão "**populados**" com esses **valores colocados**.

Adicionando Atributos

- **Imagine** que **começemos** a **aumentar** nossa classe **Conta** e adicionar **nome**, **sobrenome** e **cpf** do **cliente** dono da **conta**;

- Começaríamos a ter **muitos atributos**;

- Se pensarmos bem uma **Conta** não tem **nome**, nem **sobrenome** nem **CPF**, quem tem esses **atributos** é um **Cliente**;

Adicionando Atributos

- Então podemos **criar** uma nova **classe** e fazer uma **composição**;

- A Classe **Cliente** ficaria da seguinte **maneira**;

```
1 class Cliente {
2     String nome;
3     String sobrenome;
4     String cpf;
5 }
```

```
1 class Conta {
2     int numero;
3     double saldo;
4     double limite;
5     Cliente titular;
6     // ..
7 }
```

Adicionando Atributos

E dentro do `main` da classe de teste:

```
1 class Teste {
2     public static void main(String[] args) {
3         Conta minhaConta = new Conta();
4         Cliente c = new Cliente();
5         minhaConta.titular = c;
6         // ...
7     }
8 }
```

- Você **pode** realmente **navegar** sobre toda essa **estrutura** de **informação**, sempre usando o ponto:

```
Cliente clienteDaMinhaConta = minhaConta.titular;
clienteDaMinhaConta.nome = "Duke";
```

- Ou ainda, pode fazer isso de uma forma mais **direta** e até mais **elegante**:

```
minhaConta.titular.nome = "Duke";
```

Orientação a Objetos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Um **sistema OO** é um grande **conjunto** de **classes** que vão se **comunicar**, **delegando responsabilidades** para quem for mais **apto** a **realizar** determinada **tarefa**;

- A classe **Banco usa** a classe **Conta** que usa a classe **Cliente**, que usa a classe **Endereco**;

- Dizemos que **esses** objetos **colaboram**, **trocando mensagens** entre si. Por isso acabamos tendo **muitas classes** em nosso **sistema**, e elas **costumam** ter um **tamanho** relativamente **curto**.

Orientação a Objetos

- Mas, e se dentro do meu código eu não desse **new** em **Cliente** e tentasse acessá-lo **diretamente**?

```
class Teste {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();

        minhaConta.titular.nome = "Manoel";
        // ...
    }
}
```

- Quando damos **new** em um **objeto**, ele o **inicializa** com seus valores **default**, **0** para **números**, **false** para **boolean** e **null** para **referências**.

Orientação a Objetos

- **Null** é uma **palavra chave** em **java**, que indica uma **referência** para nenhum **objeto**.

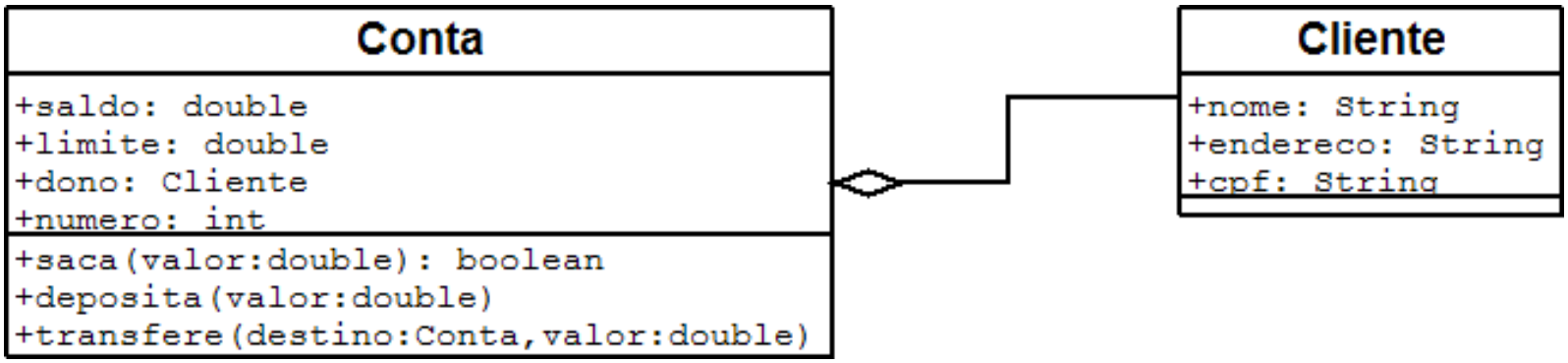
- Se, em algum caso, você tentar **acessar** um **atributo** ou **método** de alguém que está se **referenciando** para **null**, você receberá um **erro** durante a **execução** (**NullPointerException**).

- Percebemos que o **new** não traz um efeito **cascata**, a menos que você dê um **valor default**;

Orientação a Objetos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

```
1 class Conta {
2     int numero;
3     double saldo;
4     double limite;
5     Cliente titular = new Cliente();     // quando chamarem new Conta,
6                                          //havera um new Cliente para ele.
7 }
```



Orientação a Objetos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Além do **Banco** que estamos criando, vamos ver como ficariam certas **classes relacionadas** a uma fábrica de carros.

```
1  class Carro {
2      String cor;
3      String modelo;
4      double velocidadeAtual;
5      double velocidadeMaxima;
6
7      //liga o carro
8      void liga() {
9          System.out.println("O carro está ligado");
10     }
11
12     //acelera uma certa quantidade
13     void acelera(double quantidade) {
14         double velocidadeNova = this.velocidadeAtual + quantidade;
15         this.velocidadeAtual = velocidadeNova;
16     }
17
18     //devolve a marcha do carro
19     int pegaMarcha() {
20         if (this.velocidadeAtual < 0) {
21             return -1;
22         }
23         if (this.velocidadeAtual >= 0 && this.velocidadeAtual < 40) {
24             return 1;
25         }
26         if (this.velocidadeAtual >= 40 && this.velocidadeAtual < 80 {
27             return 2;
28         }
29         return 3;
30     }
31 }
```

Orientação a Objetos

- Vamos **testar** nosso **Carro** em um novo **programa**:

```
1  class TestaCarro {
2      public static void main(String[] args) {
3          Carro meuCarro;
4          meuCarro = new Carro();
5          meuCarro.cor = "Verde";
6          meuCarro.modelo = "Fusca";
7          meuCarro.velocidadeAtual = 0;
8          meuCarro.velocidadeMaxima = 80;
9
10         // liga o carro
11         meuCarro.liga();
12
13         // acelera o carro
14         meuCarro.acelera(20);
15         System.out.println(meuCarro.velocidadeAtual);
16     }
17 }
```

Orientação a Objetos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Nosso **carro** pode **conter também** um **Motor**:

```
1 class Motor {
2     int potencia;
3     String tipo;
4 }
```

```
1 class Carro {
2     String cor;
3     String modelo;
4     double velocidadeAtual;
5     double velocidadeMaxima;
6     Motor motor;
7
8     // ..
9 }
```