

Revisão: Java

Linguagem de Programação Orientada a Objetos II

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoaas

1

Principais Conceitos

- **Abstração**
 - Habilidade de ignorar aspectos não relevantes;
- **Objeto [lat. Objectum]**
 - É qualquer coisa, real ou abstrata, que possui um estado e operações para manipulá-lo;
- **Classe [lat. Classis]**
 - Definição estrutural de um conjunto de objetos que possuem características e comportamento em comum;

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoaas

2

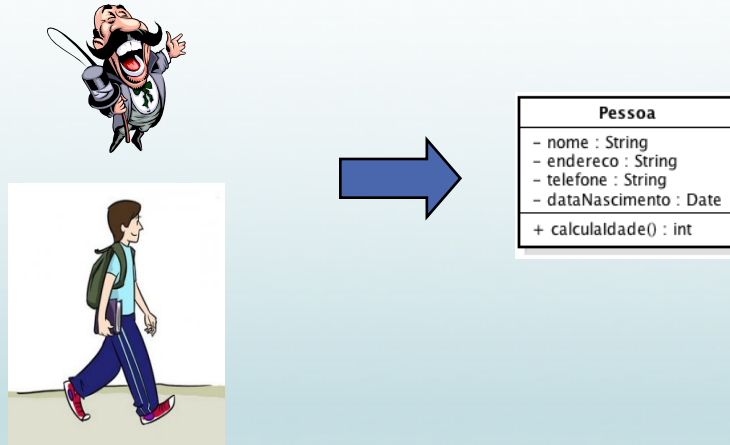
Principais Conceitos

- **Atributo [lat. *Attributum*]**
 - Qualidade que descreve uma característica de um objeto;
- **Operação [lat. *Operatio*]**
 - Serviço praticado pelos objetos e definido na classe;
- **Encapsulamento**
 - Ocultação de informações. São definidos limites para elementos que constituem a estrutura e comportamento (interfaces).

Abstração de Objetos

- **Consiste em definir os serviços e atributos aplicáveis a estes objetos dentro de um determinado contexto.**

Classe e Objetos

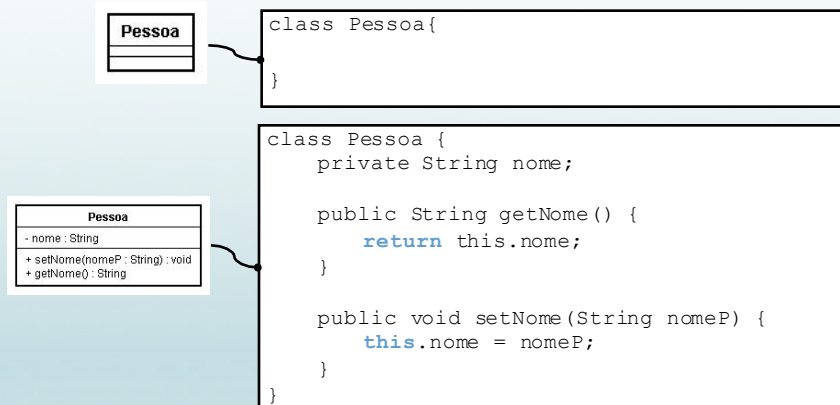


Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos I - IFRS/Canoas

5

Classes e Objetos em Java

- Em Java, a declaração de novas classes é feita através da construção class.



Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos I - IFRS/Canoas

6

Estrutura Geral de Classe Java

```
[package pacote;]
[import bibliotecas;]

[modificador] class NomeClasse {
    [modificador] [tipo] nomeAtributo;

    [modificador] [tipoRetorno] nomeMetodo ( [parametros] ) {
        // corpo do metodo
    }
}
```

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto 11 - IFRS/Canoas

7

Estrutura Geral de Classe Java

- A palavra-chave “**package**”:

```
package pacote;
public class Classe{
    ...
}
```
- A declaração “package” permite ao compilador conhecer em qual pacote as classes ou interfaces incluídas devem ser definidas.
- É a primeira declaração permitida no código fonte Java.

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto 11 - IFRS/Canoas

8

Estrutura Geral de Classe Java

- A palavra-chave “**import**”:

```
import java.util.ArrayList;
public class Classe{
    ArrayList lista;
    ...
}
```
- Organização dos Arquivos:
 - `java.util.Vector` = ... \java\util\Vector.java
 - `javax.swing.JFrame` = ... \javax\swing\JFrame.java
 - `javax.swing.*` = ... \javax\swing*.*
- A declaração “import” permite especificar quais classes de outros pacotes iremos ter visibilidade.
- Logo após a declaração “package” podem vir uma ou mais declarações “import”.

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto II - IFRS/Canoas

9

Atributos

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto II - IFRS/Canoas

10

Atributos

Java é uma linguagem fortemente tipada, isto é, toda a variável possui um tipo que não pode ser alterado

Variáveis são sempre declaradas dentro de blocos e podem ser declaradas em qualquer ponto do programa

Definição:

[modificador] [tipo] nomeAtributo;

Tipos podem ser primitivos ou referência a objetos.

Exemplo:

```
int idade;
```

Atributos

- São exemplos de atributos válidos:
 - AvgTemp count a4 \$test this_is_ok
- São exemplos de atributos inválidos:
 - 2cont high-temp not/ok

Comentários

Tipo	Notação	Exemplo
Comentário de linha	//	// Este é um comentário
Comentário multilinhas	/* */	/* Este comentário permite a inserção de várias linhas */
Comentário de documentação	/** */	/** Este comentário é utilizado para gerar a documentação dos sistemas. */

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto 11 - IFRS/Canoas

13

Atributos e atribuição

- A linha a seguir é a tradução de “idade deve valer agora quinze”.
idade = 15;
- O código a seguir declara novamente a variável idade com valor 15 e imprime seu valor na saída padrão através da chamada a System.out.println:
//declara a idade
int idade;
idade = 15;
// imprime a idade
System.out.println(idade);

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto 11 - IFRS/Canoas

14

Atributos e atribuição

- Pode-se utilizar o valor de uma variável para algum outro propósito, como alterar ou definir uma segunda variável:

```
//gera a idade no ano seguinte  
int idadeNoAnoQueVem;  
idadeNoAnoQueVem = idade + 1;
```

- Pode-se declarar uma variável a iniciando:

```
int idade = 15;
```

- Pode-se usar os operadores +, -, /, * e %:

```
int quatro = 2 + 2;  
int tres = 5 - 2;  
int oito = 4 * 2;  
int dezesesseis = 64 / 4;  
int um = 5 % 2;
```

Comandos de Controle de Fluxo

Comandos de Controle de Fluxo

- Comandos de Seleção
 - if-else
 - switch
- Comandos de Repetição
 - while
 - do-while
 - for
- Comandos de Desvio
 - break
 - continue

Comandos de Seleção

Comandos de Controle de Fluxo

Comando de Seleção - IF

```
if (condição) {
    bloco
} else if (condição) {
    bloco
} else {
    bloco
}
```

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canais

31

Comando de Seleção - IF

Exemplo:

```
if (x>10) a = 2;
else b = 5;
```

```
if ((y >= 10) && (x < 5))
    b = 5;
else {
    a = 5; b *=b;
}
```

32

Comando de Seleção - IF

- Exemplo:

```
int idade = 15;
if (idade < 18) {
    System.out.println("Não pode entrar");
}
else {
    System.out.println("Pode entrar");
}
```

Comando de Seleção - IF

- Exemplo:

```
int idade = 15;
boolean amigoDoDono = true;
if (idade < 18 & amigoDoDono == false) {
    System.out.println("Não pode entrar");
}
else {
    System.out.println("Pode entrar");
}
```

Comando de Seleção - IF

- Exemplo:

```
int idade = 15;
boolean amigoDoDono = true;
if (idade < 18 & !amigoDoDono) {
    System.out.println("Não pode entrar");
}
else {
    System.out.println("Pode entrar");
}
```

Operador && e ||

- Operadores && e || funcionam como seus operadores irmãos, porém eles funcionam da maneira mais rápida possível:
 - quando percebem que a resposta não mudará mais, eles param de verificar as outras condições booleanas.
 - Por isso, eles são chamados de operadores de curto circuito (short circuit operators).
- Exemplo:
 - if (true | algumaCoisa) { // ... }
 - O valor de algumaCoisa será analisado,
 - if (true || algumaCoisa) { // ... }
 - Neste caso o algumaCoisa não será analisado.

Operador Ternário (?)

- `expr1 ? expr2 : expr3`

- Exemplo:

```
razao = denom == 0 ? 0 : num / denom;
```

EQUIVALENTE:

```
if (denom==0)
    razao = 0;
else
    razao = num/denom;
```

Comando de Seleção - SWITCH

// A **[expressão]** deve resultar em um dos seguintes tipos:

// byte, short, int ou char.

```
switch ([expressão]) {
    case [constante 1] : [comando1]
        break;
    case [constante 2] : [comando 2]
        break;
    ...
    case [constante n] : [comando n]
        break;
    default : [comando]
}
```

Comando de Seleção - SWITCH

Exemplo

```
switch ( sex ) {  
  case 'm' :  
    name = "Male";  
    a = 1;  
    break;  
  case 'f' :  
    name = "Female"  
    break;  
}
```

Comandos de Repetição

Comandos de Controle de Fluxo

Comando de Repetição - WHILE

```
while (condicao) {  
    bloco de comandos  
}
```

Comando de Repetição - WHILE

Exemplo:

```
int idade = 15;  
while(idade < 18) {  
    System.out.println(idade);  
    idade = idade + 1;  
}  
  
int i = 0;  
while(i < 10) {  
    System.out.println(i);  
    i = i + 1;  
}
```

Comando de Repetição - WHILE

Exemplo:

```
while ( (cont > 0) || a < 3) {  
    a = n * 3;  
    cont += 4;  
}
```

Comando de Repetição - WHILE

Exemplo:

```
while ( (cont > 0) || a < 3)  
    cont ++;
```


Comandos de Repetição

DO-WHILE

```
do {  
    bloco de comandos  
} while ( condicao)
```

Comandos de Repetição

DO-WHILE

Exemplo:

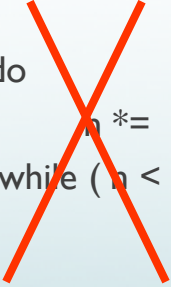
```
do {  
    j++;  
    n = j * 6;  
} while ( n < 0 );
```

Comandos de Repetição

DO-WHILE

Exemplo:

```
do  
    n *= 6;  
while ( n < 0 );
```



Comandos de Repetição – FOR

```
for (inicializacao; condicao; iteracao) {  
    bloco de comandos  
}
```

Comandos de Repetição - FOR

- Exemplo:

```
for (int i = 0; i < 10; i++) {
    n += i;
}
```

- Que é equivalente a:

```
int i = 0;
while(i < 10) {
    n += i;
    i++;
}
```

Comandos de Repetição - FOR

- Exemplo:

```
for (int i = 0; i < 10; i++) n += i;
```

Comandos de Repetição - FOR

Exemplo:

```
int a, b;  
b = 4;  
for (a = 1; a < b; a++) {  
    // comandos  
    b --;  
}
```

Comandos de Repetição - FOR

Exemplo

```
int a, b;  
for (a = 1, b=4; a < b; a++) {  
    // comandos  
}
```

Comandos de Repetição - FOR

Exemplo:

```
boolean achou= false;  
for (a = 1; !achou; a++) {  
    // comandos  
    if ( a == 10) achou = true;  
}
```

Pós e Pré incremento

- Quanto vale x e i?

```
int i = 5;  
int x = i++;
```

- E agora?

```
int i = 5;  
int x = ++i;
```

Comandos de Desvio

Comandos de Controle de Fluxo

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

55

Comandos de Desvio - BREAK

- Utilizado cancelar a iteração do loop;
- Desvia a execução para o próximo comando (primeiro comando após o loop).

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

56

Comandos de Desvio - BREAK

Exemplo: **em que condição o loop é interrompido?**

```
while (x > 10) {  
    a = x + 10;  
    if (a == 50 || (a > x))  
        break;  
}  
a = -30;  
...
```

Comandos de Desvio - CONTINUE

- Avança para a próxima iteração do loop sem executar o código subsequente.

Comandos de Desvio - CONTINUE

- Exemplo:

```
while (a != 10) {
    // comandos
    b++;
    if (b % 2 == 0)
        continue;
    a++;
}
```

Controlando Loops

- **BREAK**

```
for(int i = x; i < y; i++){
    if (i % 19 == 0){
        System.out.println("Achei um número divisível por 19 entre x e y");
        break;
    }
}
```

- **CONTINUE**

```
for(int i = 0; i < 100; i++){
    if(i > 50 && i < 60) {
        continue;
    }
    System.out.println(i);
}
```

- O código acima não vai imprimir alguns números. (Quais exatamente?)

Escopo das Variáveis

- Pode-se declarar variáveis a qualquer momento, mas vale após a declaração

```
//aqui a variável i não existe
int i = 5;
// a partir daqui ela existe
```

- E dentro do bloco:

```
//aqui a variável i não existe
int i = 5;
// a partir daqui ela existe
while (condição) {
    // o i ainda vale aqui
    int j = 7;
    // o j passa a existir
}
// aqui o j não existe mais, mas o i continua a valer
```

- O que acontece quando tenta acessar a variável fora do seu escopo?
Tente imprimir o j após o bloco.

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos 11 - IFRS/Canas

61

Escopo das Variáveis

- O que acontece com?

```
if (algumBooleano) {
    int i = 5;
}
else {
    int i = 10;
}
System.out.println(i);
```

- E com?

```
for (int i = 0; i < 10; i++) { System.out.println("olá!"); }
System.out.println(i);
```

- Corrija os códigos.

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos 11 - IFRS/Canas

62

Orientação a Objetos

Java

Criando um Tipo

- Pode-se considerar a entidade **Conta** como importante dentro de um sistema bancário
 - Para esse sistema, é importante:
 - número da conta,
 - tipo de conta,
 - nome do cliente,
 - saldo,
 - limite
 - E a conta poderia fazer:
 - sacar ou depositar uma quantidade x,
 - imprimir o nome do dono da conta,
 - devolver o saldo atual,
 - transferir uma quantidade x para outra conta,
 - retornar o tipo de conta (corrente, poupança, etc.)
- Mas dessa especificação, não se pode acessar saldo de ninguém, é preciso primeiro criar uma conta específica para depois solicitar algo dela.

Criando um Tipo

- Com isso, definimos a **classe** (especificação):

Conta
<ul style="list-style-type: none"> - numero : int - nome : String - saldo : double - limite : double
<ul style="list-style-type: none"> + saca(valor : double) : boolean + deposita(valor : double) : void

- A qual pode criar **objetos** reais (contas de verdade) que podem de armazenar valores:

contaDoRodrigo : Conta

```
numero = 12345
nome = Rodrigo
saldo = 1000.00
limite = 300.00
```

contaDoFulano : Conta

```
numero = 12346
nome = Fulano
saldo = 10000.00
limite = 2000.00
```

contaDoBeltrano : Conta

```
numero = 12347
nome = Beltrano
saldo = 3000.00
limite = 500.00
```

Definição de Classe

- Vem da taxonomia da biologia
 - Representa que todas as entidades de uma mesma classe compartilham uma série de atributos e comportamentos
 - Nem todas as entidades de uma classe são iguais, podem variar os valores de atributos e como realizam os comportamentos
- Exemplo:
 - Não se pode fazer com que um Carro ande, é preciso primeiro construir um carro específico (instanciar) para depois da construção solicitar que ele ande.
 - Uma receita de bolo não é comestível, é uma especificação. O que se come é o quando essa receita é executada criando o bolo a partir das instruções.
- Pergunta:
 - A planta de uma casa é uma classe ou um objeto? Por quê?

Um classe em Java

- Revendo a conta anterior:

```
class Conta {  
    int numero;  
    String nome;  
    double saldo;  
    double limite;  
  
    // ..  
}
```

- Por enquanto, declara-se o que toda conta deve ter – os atributos que toda conta que, quando criada, vai ter.
- Repare que as variáveis foram declaradas fora do main, **dentro do escopo da classe**, sendo assim chamada de variável de objeto, ou **atributo**.

Criação de Objetos como Atributos de Classe

Criação e Uso de Objetos

- Digamos que uma classe chamada **Programa.java** deve utilizar a conta criada
 - Para usar, é necessário criar (construir, instanciar) e, para isso, basta usar a palavra chave *new* com os parênteses (mais tarde falaremos disso)

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

- Esse código cria a conta, mas não é possível acessar depois disso, portanto precisamos de um objeto para a referenciar:

```
class Programa {
    public static void main(String[] args) {
        Conta minhaConta;
        minhaConta = new Conta();
        //E agora se pode utilizar as variáveis:
        minhaConta.nome = "Rodrigo";
        minhaConta.saldo = 1000.0;

        System.out.println("Saldo atual: " + minhaConta.saldo);
    }
}
```

Métodos

Métodos

- Métodos são utilizados para definir os comportamentos (funções) de uma classe – o que ela faz!

– Exemplo: como é realizado o saque?

```
class Conta {
    int numero;
    String nome;
    double saldo;
    double limite;

    void saca(double quantidade) {
        double novoSaldo = this.saldo - quantidade;
        this.saldo = novoSaldo;
    }
}
```

- Atente para:

- A palavra **void** diz que quando invocar o método, nada será retornado a quem pediu
- Quando alguém solicitar um saque, é necessário informar a quantidade como um argumento do método (ou parâmetro)
- Pode-se declarar dentro do método outras variáveis, como **novoSaldo** que só existe dentro do método
- A palavra **this** é usada para referenciar o atributo da classe.
- Neste caso, o saque pode estourar o valor limite fixado pelo banco.

Métodos

- Exemplo: como é realizado o depósito?

```
class Conta {
    int numero;
    String nome;
    double saldo;
    double limite;

    void saca(double quantidade) {
        double novoSaldo = this.saldo - quantidade;
        this.saldo = novoSaldo;
    }
    void deposita(double quantidade) {
        this.saldo += quantidade;
    }
}
```

Métodos

- Para mandar uma mensagem ao objeto e pedir que ele execute um método, usamos o ponto (**invocação** do método):

```
class SacaeDepositaTest {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        // alterando os valores de minhaConta
        minhaConta.nome = "Rodrigo";
        minhaConta.saldo = 1000;
        minhaConta.saca(200); // saca 200 reais
        minhaConta.deposita(500); // deposita 500 reais
        System.out.println(minhaConta.saldo);
    }
}
```

- Qual é o resultado da execução do código acima?

Métodos

- Um método sempre tem que retornar alguma coisa, nem que essa coisa seja nada (**void**).
- Observe os tipos de retorno:

Conta
- numero : int - nome : String - saldo : double - limite : double
+ saca(valor : double) : boolean + deposita(valor : double) : void

```
class SacaeDeposita {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta = new Conta();
        minhaConta.saldo = 1000;
        System.out.println(minhaConta.saca(200));
    }
}
```

```
class Conta {
    int numero;
    String nome;
    double saldo;
    double limite;

    boolean saca(double valor) {
        if (this.saldo < valor) {
            return false;
        } else {
            this.saldo = this.saldo - valor;
            return true;
        }
    }

    void deposita(double quantidade) {
        this.saldo += quantidade;
    }
}
```

Objetos e Referências

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto11 - IFRS/Canoas

75

Objetos e Referências

- 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:
- É correto dizer que **minhaConta** se refere a um objeto e não que é um objeto.
- Lembre-se, uma variável nunca é um objeto

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

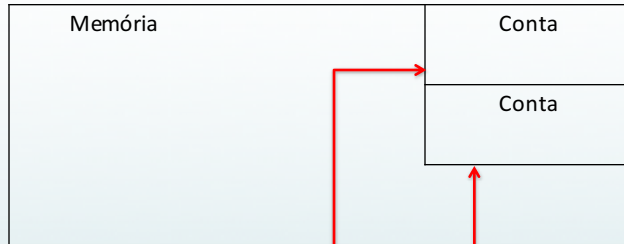
Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objeto11 - IFRS/Canoas

76

Objetos e Referências

Internamente, `minhaConta` e `minhaContaDosSonhos` vão guardar um número que identifica em que posição da memória aquela `Conta` se encontra.

Dessa maneira, ao utilizarmos o "." para navegar, o Java vai acessar a `Conta` que se encontra naquela posição de memória, e não uma outra.



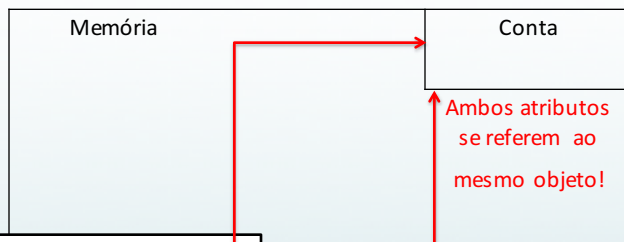
```
public static void main(String args[]) {
    Conta minhaConta;
    minhaConta = new Conta();
    minhaConta.deposita(100);

    Conta minhaContaDosSonhos;
    minhaContaDosSonhos = new Conta();
    minhaContaDosSonhos.deposita(1000000);
}
```

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

77

Objetos e Referências



```
class Programa {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();
        minhaConta.deposita(100);

        Conta minhaContaDosSonhos;
        minhaContaDosSonhos = minhaConta;
        minhaContaDosSonhos.deposita(1000000);

        System.out.println(minhaConta.saldo);
        System.out.println(minhaContaDosSonhos.saldo);
    }
}
```

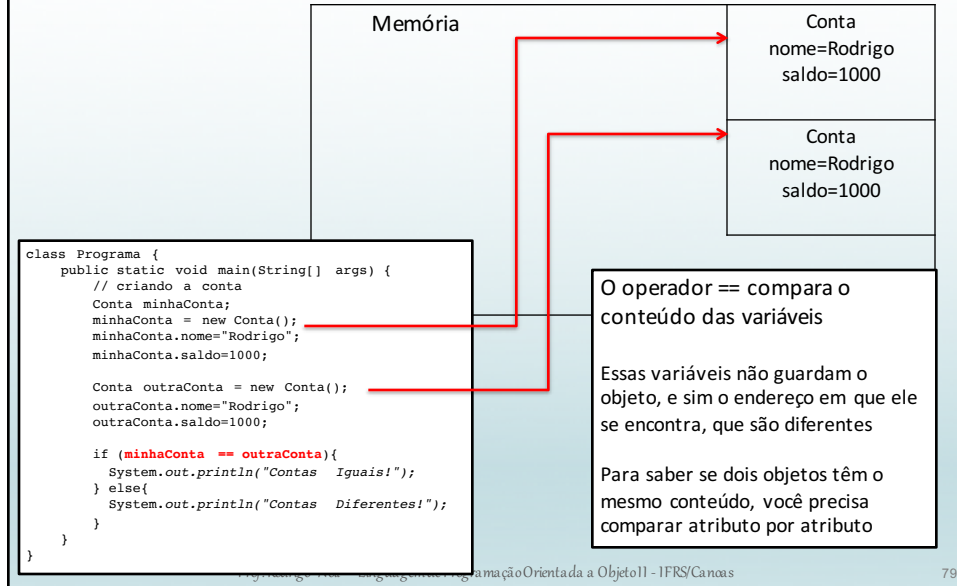
E qual o resultado?

100100

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

78

Objetos e Referências



Método transfere()

- Como realizar uma transferência?
 - Criando um método **transfere(Conta conta1, Conta conta2, double valor)?**
 - Essa é uma solução procedural.
 - Ao chamar o método **transfere**, já existe um objeto do tipo **Conta** (o **this**), portanto o método recebe apenas **um parâmetro do tipo Conta, a Conta destino** (além do valor)

Conta
- numero : int - nome : String - saldo : double - limite : double
+ saca(valor : double) : boolean + deposita(valor : double) : void + transfere(destino : Conta, valor : double) : boolean

```
public class Conta {
    int numero;
    String nome;
    double saldo;
    double limite;

    ...

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

Método transfere()

- Para realizar a transferência:

```
class Programa {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();
        minhaConta.nome="Rodrigo";
        minhaConta.saldo=1000;

        Conta outraConta = new Conta();
        outraConta.nome="Rodrigo";
        outraConta.saldo=1000;

        //minhaConta transfere para outraConta 100 reais
        minhaConta.transfere(outraConta, 100);
        System.out.println("Saldo da Minha Conta = "+minhaConta.saldo);
        System.out.println("Saldo da Outra Conta = "+outraConta.saldo);
    }
}
```

Saldo da Minha Conta = 900.0

Saldo da Outra Conta = 1100.0

Atributos de Classe

Atributos de Classes

- As variáveis do tipo atributo, diferentemente das variáveis temporárias (declaradas dentro de um método), recebem um valor padrão.
 - Valores numéricos recebem 0
 - Valores boolean recebem false
 - Objetos recebem null para as referências (**null** indica uma referência para nenhum objeto)
- Você também pode dar **valores default**, como segue:

```
public class Conta {
    int numero=1234;
    String nome="Rodrigo";
    double saldo=1000;
    double limite=1000;
}
```

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos I - IFRS/Canoas

83

Atributos de Classes

- Se começamos a aumentar nossa classe Conta e adicionar nome, sobrenome e cpf do cliente dono da conta, começaríamos a ter muitos atributos...
 - Se você pensar direito, uma Conta não tem nome, nem sobrenome nem cpf, quem tem esses atributos é um Cliente.

```
public class Conta{
    int numero;
    double saldo;
    double limite;
    Cliente titular;
}
```

```
public class Cliente{
    String nome;
    String sobrenome;
    String cpf;
}
```

Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos I - IFRS/Canoas

84

Atributos de Classes

- Um sistema orientado a objetos é um grande conjunto de classes que vai 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.

Atributos de Classes

```
class Programa {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta = new Conta();
        Cliente cliente = new Cliente();

        minhaConta.titular = cliente;
        minhaConta.titular.nome = "Rodrigo";
        cliente.nome = "Fulano";
        System.out.println(minhaConta.titular.nome);
    }
}
```

Altere o programa comentando a instanciação do cliente. O que acontece?

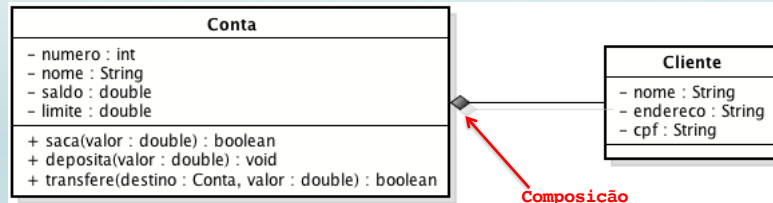
Se tentar acessar um atributo ou método de alguém que está null, você receberá um erro durante a execução (NullPointerException).

Atributos de Classes

- Para criar um cliente para cada nova Conta sem necessidade de instanciar esse cliente toda vez que criar uma conta, pode-se utilizar o código abaixo.
 - Ele significa **COMPOSIÇÃO**, isto é, para toda nova Conta você terá um novo Cliente. Apagando a conta, apaga o cliente.

```
public class Conta{
    int numero;
    double saldo;
    double limite;
    Cliente titular = new Cliente();
}
```

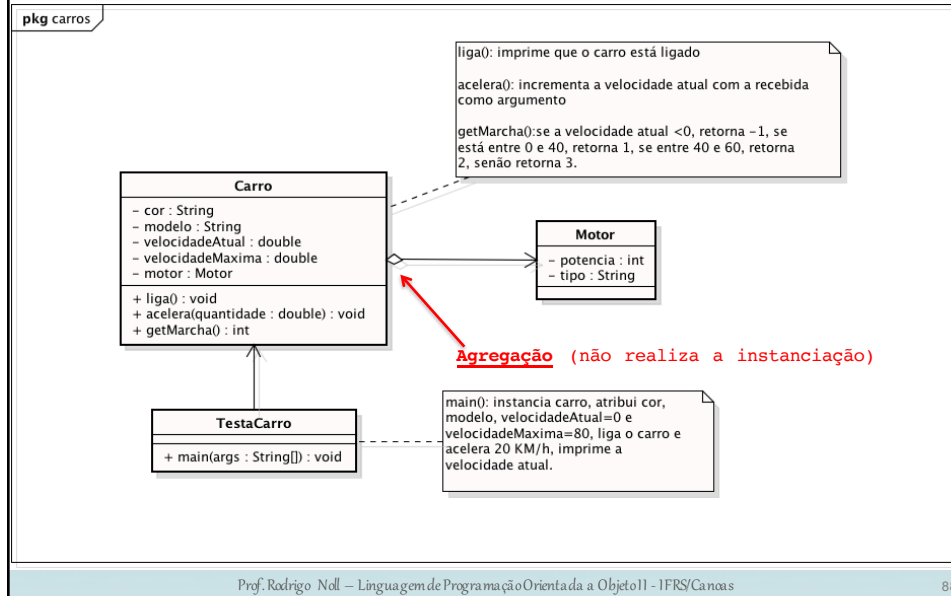
```
public class Cliente{
    String nome;
    String sobrenome;
    String cpf;
}
```



Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

87

Exercício



Prof. Rodrigo Noll – Linguagem de Programação Orientada a Objetos II - IFRS/Canoas

88