Características das Linguagens de Programação I

Conteúdo: Classes, atributos e métodos

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Classes, atributos e métodos

Classes em Java

- Sintaxe:

<qualificador> ***class*** *<nome_da_classe>{*
<declaração dos atributos da classe>
<declaração dos métodos da classe>
}

<qualificador>: O qualificador de acesso determinará a visibilidade da classe. Pode ser **public** (classe pública) ou **private** (classe privada).

<nome>: *nome que identifica a classe. Há um padrão entre os programadores* de sempre iniciarem os nomes de classes com letras maiúsculas.

Classes em Java

- Criando uma classe em Java

```
class Pessoa{
    . . .
}
```

- Criando objetos (instâncias) em Java

```
Pessoa p1 = new Pessoa();
Pessoa p2 = new Pessoa();
```

Declaração de atributos

- Sintaxe:

  <qualificador> <tipo_do_atributo> <nome_do_atributo>;

<qualificador>: O qualificador de acesso determinará a visibilidade do atributo. É opcional e, se não for informado, por padrão o atributo será protegido (protected).

<tipo_do_atributo>: é um tipo primitivo ou classe que define o atributo.

<nome_do_atributo>: nome que identifica o atributo. Há um padrão entre os programadores de sempre iniciarem os nomes de atributos com letras minúsculas.

Declaração de atributos: Exemplo

- Classes como atributos de uma classe

```
class Pessoa{
    int idade;
    String nome;
    Endereco end;
}
```

```
class Endereco{
    String rua;
    String bairro;
    int numero;
}
```

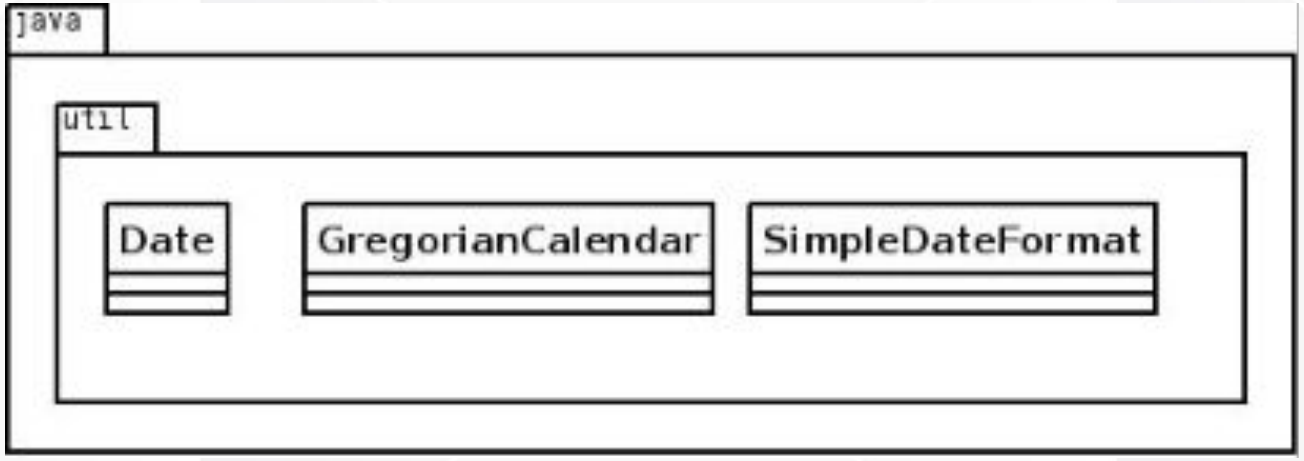
- Criando objetos (instâncias) em Java

```
Pessoa p1 = new Pessoa();
p1.nome="João";
p1.endereco = new Endereco();
```

Classes e pacotes em Java

- Os **pacotes** servem para agrupar classes de funcionalidades similares ou relacionadas.

- Por exemplo, no pacote **java.util** temos as classes Date, SimpleDateFormat e GregorianCalendar; todas elas trabalham com datas de formas diferentes.



Classes e pacotes em Java: Diretórios

```
package br.com.anhanguera.banco;

class Cliente {
  // ...
}
```

Se a classe **Cliente** está no pacote **banco**, ela deverá estar no diretório com o mesmo nome: banco. Se ela se localiza no pacote **br.com.anhanguera.banco**, significa que está no diretório **br/com/anhanguera/banco**.

Um pacote pode conter nenhum ou mais subpacotes e/ou classes dentro dele.

Classes e pacotes em Java: import

Para usar uma classe do mesmo pacote, basta fazer referência a ela simplesmente escrevendo o próprio nome da classe. Se quisermos que a classe Banco fique dentro do pacote **br.com.anhanguera.banco**, ela deve ser declarada assim:

```
package br.com.anhanguera.banco;

class Banco {
  String nome;
  Cliente clientes[];
}
```

Para a classe Cliente ficar no mesmo pacote, seguimos a mesma fórmula:

```
package br.com.anhanguera.banco;

class Cliente {
  String nome;
  String endereco;
}
```

Classes e pacotes em Java: Diretórios

Ao tentar utilizar a classe **Banco** (ou Cliente) em uma outra classe que esteja **fora** desse **pacote**, por exemplo, no pacote br.com.anhanguera.util:

```
package br.com.anhanguera.banco.util;

class TesteDoBanco {

  public static void main(String args[]) {
    br.com.anhanguera.banco.Banco meuBanco = new br.com.anhanguera.banco.Banco();
    meuBanco.nome = "Banco do Brasil";
    System.out.println(meuBanco.nome);
  }

}
```

O que acontece ao testar o programa?

Estrutura Projeto

Classes e pacotes em Java: import

Fazendo a classe e os atributos públicos:

```
package br.com.anhanguera.banco;

public class Banco {
    public String nome;
    public Cliente clientes[];
}
```

```
package br.com.anhanguera.banco;

public class Cliente {
    public String nome;
    public String endereco;
}
```

Existe uma maneira mais simples de se referenciar a classe Banco: basta importá-la do pacote **br.com.anhanguera.banco**:

Classes e pacotes em Java: Exemplo

```java
package exemplos;

public class Conta {
    int numero;
    String nomeTitular;
    double saldo;
}
```

```java
package pacote1;

class ClassePrivada {
    int atributo1;
}
```

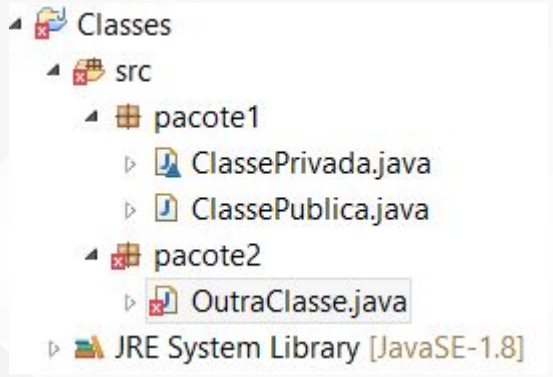
```java
package pacote1;

public class ClassePublica {
    int atributo1;
}
```

Classes e pacotes em Java: Exemplo

- Classes como atributos:

```
package pacote2;

import pacote1.ClassePrivada;
import pacote1.ClassePublica;


public class OutraClasse {
    ClassePublica objeto1;
    ClassePrivada objeto2;
}
```

Estrutura Projeto



- As linhas sublinhadas em vermelho indicam erro no Eclipse. Os erros ocorreram exatamente nas linhas em que tentamos utilizar a **classe privada** em um pacote externo ao pacote no qual ela foi criada

Declaração de métodos

- Sintaxe:

```
<qualificador> <tipo_de_retorno> <nome_do_metodo>
(<lista_de_argumentos>){
        <corpo_ do_ método>
}
```

<qualificador>: Mesmo conceito usado no caso de atributos.
<tipo_do_retorno>: é um tipo primitivo ou classe que define o retorno a ser dado pelo método.
<nome_do_método>: nome que identifica o método.
<corpo_do_método>: código que define o que o método faz.

Declaração de métodos: Exemplo

- Métodos de uma classe

```
class Pessoa{
 ...
    void aniversariar(){
        idade = idade + 1;
}
```

```
Pessoa p1 = new Pessoa();
p1.nome="João";
p1.aniversariar();
```

Declaração de métodos: Exemplo 1

- Classe Cachorro

```
class Cachorro{

  int tamanho;
  String nome;

  void latir(){
    if(tamanho > 60)
      System.out.println("Wooof, Wooof!");
    else if(tamanho > 14)
      System.out.println("Ruff!, Ruff!");
      else System.out.println("Yip!, Yip!");
  }
}
```

- Criar a Classe Testa_Cachorro

```
public class Testa_Cachorro {

  public static void main(String[] args) {
    Cachorro bob = new Cachorro();
    bob.tamanho = 70;

    Cachorro rex = new Cachorro();
    rex.tamanho = 8;

    Cachorro scooby = new Cachorro();
    scooby.tamanho = 35;

    bob.latir();
    rex.latir();
    scooby.latir();
  }
}
```

Declaração de métodos: Exemplo 2

- Criar um novo projeto Java
- Criar a classe **Conta**

```
public class Conta {

    //atributos
    int numero;
    String nomeTitular;
    double saldo;

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

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

O **this** representa a instância atual da classe em que está sendo utilizado (é uma referência ao próprio objeto)

Declaração de métodos: Exemplo 2 (cont.)

- Criar a classe **TestaConta**

```
public class TestaConta {

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

        // alterando os valores de minhaConta
        minhaConta.nomeTitular = "Giomar";
        minhaConta.saldo = 1000;
        System.out.println("Saldo: "+minhaConta.saldo);

        // saca 200 reais
        minhaConta.saca(200);
        System.out.println("Saldo: "+minhaConta.saldo);

        // deposita 500 reais
        minhaConta.deposita(500);
        System.out.println("Saldo: "+minhaConta.saldo);
    }
}
```

Problemas no código???

Declaração de métodos: Exemplo 2 (cont.)

- Modificando o método **saca** na classe **Conta**

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

- Modificando o programa principal **TestaConta**

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

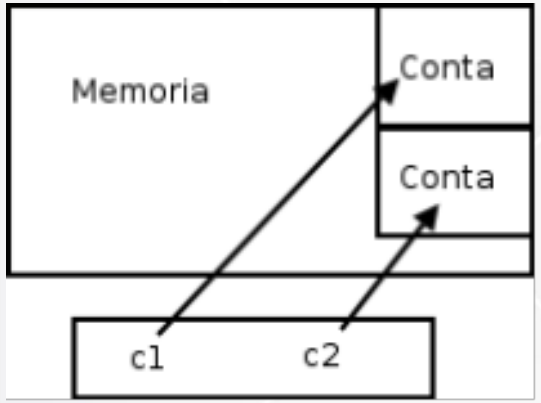
Objetos são acessados por referências

- 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**.

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

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

Internamente, **c1** e **c2** vão guardar um número que identifica em que **posição** da **memória** aquela Conta se encontra.

É parecido com um **ponteiro**, porém você **não pode manipulá-lo** como um número e nem utilizá-lo para aritmética, ela é tipada



Objetos são acessados por referências

- Outro exemplo.

```
class TestaReferencias {

   public static void main(String args[]) {
     Conta c1 = new Conta();
     c1.deposita(100);

     Conta c2 = c1;  // linha importante!
     c2.deposita(200);

     System.out.println(c1.saldo);
      System.out.println(c2.saldo);
   }
}
```

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

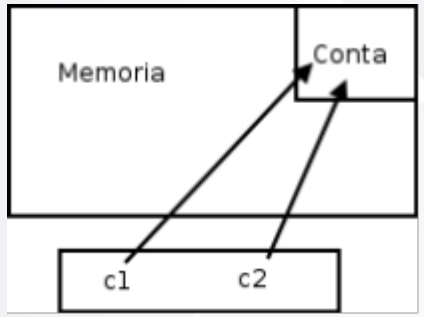
Objetos são acessados por referências

- Outro exemplo.

```
class TestaReferencias {

   public static void main(String args[]) {
     Conta c1 = new Conta();
     c1.deposita(100);

     Conta c2 = c1;  // linha importante!
     c2.deposita(200);

     System.out.println(c1.saldo);
      System.out.println(c2.saldo);
   }
}
```

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

  Quando fizemos c2 = c1, c2 passa a fazer referência para o mesmo objeto que c1 referencia nesse instante.



Objetos são acessados por referências

- O que acontece no código abaixo?

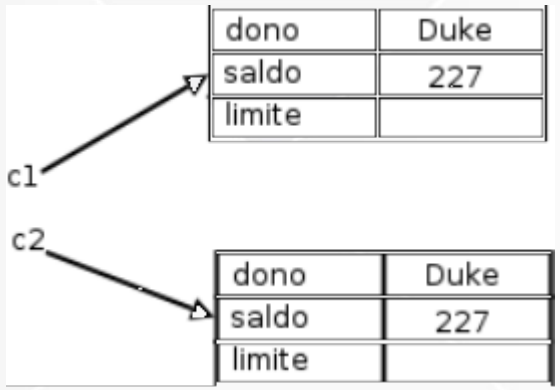
```
public static void main(String args[]) {

        Conta c1 = new Conta();
        c1.nomeTitular = "Duke";
        c1.saldo = 227;

        Conta c2 = new Conta();
        c2.nomeTitular = "Duke";
        c2.saldo = 227;

        if (c1 == c2) {
          System.out.println("Contas iguais");
        }
}
```

Objetos são acessados por referências

- O que acontece no código abaixo?

```
public static void main(String args[]) {

        Conta c1 = new Conta();
        c1.nomeTitular = "Duke";
        c1.saldo = 227;

        Conta c2 = new Conta();
        c2.nomeTitular = "Duke";
        c2.saldo = 227;

        if (c1 == c2) {
          System.out.println("Contas iguais");
        }
}
```

O operador **==** compara o conteúdo das **variáveis**, mas essas variáveis não guardam o objeto, e sim o **endereço** em que ele se encontra. Como em cada uma dessas variáveis guardamos duas contas criadas diferentemente, elas estão em espaços diferentes da memória, o que faz o teste no if valer **false**. As contas podem ser equivalentes no nosso critério de igualdade, porém elas **não são o mesmo objeto**.



Declaração de métodos: Exemplo 3

- Acrescentando o método **transfere** na classe **Conta**

- 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, a Conta destino (além do valor):

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

Declaração de métodos: Exemplo 3

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

```
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;
    }
  }
```

- Modifique o programa principal para testar o método transfere

Algumas considerações

- 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**.

- Você também pode dar valores **default**, como segue:

```
class Conta {
    int numero = 1234;
    String dono = "Giomar";
    String cpf = "123.456.789-10";
    double saldo = 1000;
    double limite = 1000;
 }
```

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

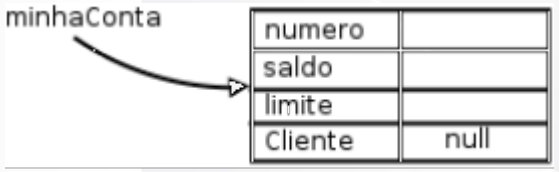
Algumas considerações

- Uma classe pode ter outras classes como atributos

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

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

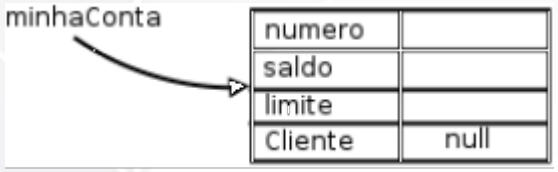
```
class Teste {

   public static void main(String[] args) {
      Conta minhaConta = new Conta();
      Cliente c = new Cliente();
      minhaConta.titular = c;
      // ...
   }
}
```



minhaConta tem uma referência ao mesmo Cliente que **c** se refere, e pode ser acessado através de **minhaConta.titular**.

Algumas considerações

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



- 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.**

- **Solução:**

```
class Conta {
  int numero;
  double saldo;
  double limite;
  Cliente titular = new Cliente();   // quando chamarem new Conta,
                                     //havera um new Cliente para ele.
}
```

Exercício

- Crie um **novo projeto Java** e depois crie a classe **Carro**

```
class Carro {
   String cor;
   String modelo;
   double velocidadeAtual;
   double velocidadeMaxima;

   //liga o carro
   void liga() {
     System.out.println("O carro está ligado");
   }

   //acelera uma certa quantidade
   void acelera(double quantidade) {
     double velocidadeNova = this.velocidadeAtual + quantidade;
     this.velocidadeAtual = velocidadeNova;
   }

   //devolve a marcha do carro
   int pegaMarcha() {
     if (this.velocidadeAtual < 0) {
       return -1;
     }
     if (this.velocidadeAtual >= 0 && this.velocidadeAtual < 40) {
       return 1;
     }
     if (this.velocidadeAtual >= 40 && this.velocidadeAtual < 80) {
       return 2;
     }
     return 3;
   }
 }
```

Exercício

- Adicione ao projeto a classe **TestaCarro**

```
class TestaCarro {
   public static void main(String[] args) {
     Carro meuCarro;
     meuCarro = new Carro();
     meuCarro.cor = "Verde";
     meuCarro.modelo = "BMW";
     meuCarro.velocidadeAtual = 0;
     meuCarro.velocidadeMaxima = 120;

     // liga o carro
     meuCarro.liga();

     // acelera o carro
     meuCarro.acelera(20);
     System.out.println("Velocidade Atual: "+meuCarro.velocidadeAtual);
   }
 }
```

Testar o método **pegarMarcha()**

Exercício

- Crie uma classe **Funcionario**. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário (double), a data de entrada no banco (String) e seu RG (String).

- Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método **recebeAumento** que aumenta o salario do funcionário de acordo com o parâmetro passado como argumento. Crie também um método **calculaGanhoAnual**, que não recebe parâmetro algum, devolvendo o valor do salário multiplicado por 12.

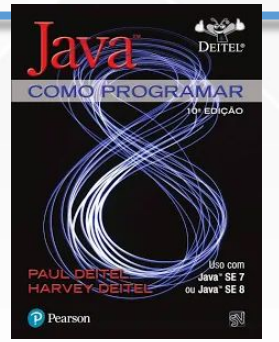
- Crie um método mostra(), que não recebe nem devolve parâmetro algum e simplesmente imprime todos os atributos do nosso funcionário.

- Crie uma classe **TestaFuncionario** para testar as funcionalidades implementadas

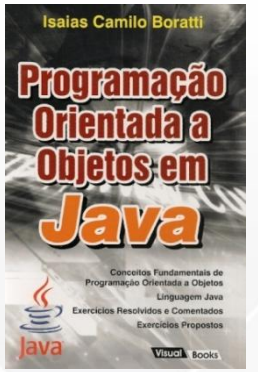
Referências

BIBLIOGRAFIA BÁSICA:

- DEITEL, Harvery M.. Java : como programar. 10ª ed. São Paulo: Pearson - Prentice Hall, 2017.



- BORATTI, Isaías Camilo. Programação Orientada a Objetos em Java : Conceitos Fundamentais de Programação Orientada a Objetos. 1ª ed. Florianópolis: VisualBooks, 2007.



- SIERRA, Kathy; BATES, Bert. Use a Cabeça! Java. 2ª ed. Rio de Janeiro: Alta Books, 2007.