

# Olhando as Classes de Perto

Prof. Marcelo Cohen

## 1. Revisão: Sobrecarga de construtores e métodos

- Sobrecarga (*overloading*) é quando cria-se dois métodos com o **mesmo nome**, mas com **parâmetros diferentes**

```
class Esfera
{
    private float x,y,z; // centro
    private float raio;

    // construtor
    public Esfera(float x1,float y1,float z1,float r)
    {
        x = x1;
        y = y1;
        z = z1;
        raio = r;
    }
    ...
}
```

- Como fazer sobrecarga ? Exemplo: sobrecarga de construtores

```
class Esfera
{
    private float x,y,z; // centro
    private float raio;
    ...

    // construtor
    public Esfera(float x1,float y1,float z1,float r)
    {
        x = x1;
        y = y1;
        z = z1;
        raio = r;
    }

    // outro construtor
    public Esfera(float r)
    {
        x = 0; y = 0; z = 0;
        raio = r;
    }
}
```

- Como usar ?

```
class TestaEsfera
{
    public static void main(String args[])
    {
        // Cria uma esfera com raio = 200 e centro
        // em (0,10,0)
        Esfera bola = new Esfera(0,10,0,100);


        // Cria uma esfera com raio = 20 e centro
        // em (0,0,0)
        Esfera bolinha = new Esfera(20);
        ...
    }
}
```

- Na verdade, qualquer método pode ser sobrecarregado, não apenas os construtores:

```
class Relogio
{
    private int hora,min,seg; // horário atual
    ...
    public void reset()
    {
        hora = 12;
        min = 0;
        seg = 0;
    }
    public void reset(int h, int m, int s)
    {
        hora = h;
        min = m;
        seg = s;
    }
    ...
}
```

- Para haver sobrecarga, o tipo de retorno tem que ser sempre o mesmo !

```
class Relogio
{
    private int hora,min,seg; // horário atual
    ...
    public int calculaAlgumaCoisa()
    {
        ...
    }
    public float calculaAlgumacoisa(int dado)
    {
    }
    ...
}
```



- Esta construção é inválida, pois um método está retornando um int e o outro, um float

## 2. Revisão: Variáveis de instância e de classe

- **Variáveis de instância**

- São as variáveis que representam o estado do objeto, ou seja, os seus atributos.

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    ...
}
```

- Só existem dentro de uma instância!

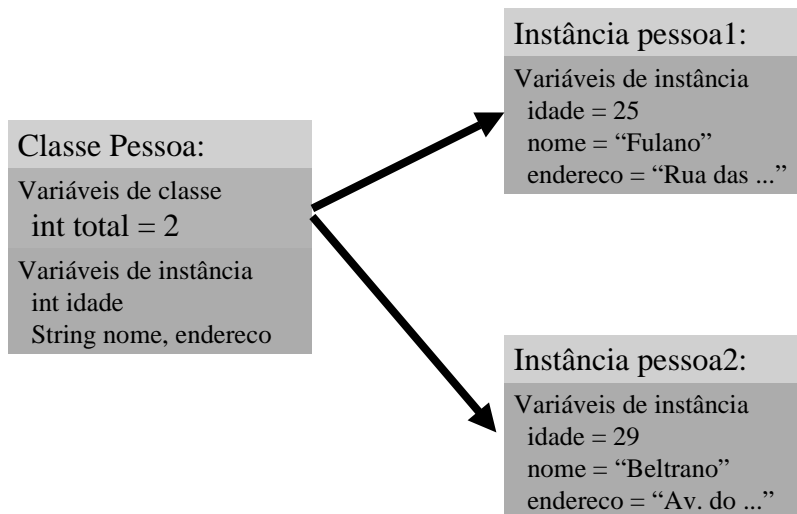
- **Variáveis de classe**

- São variáveis que servem para controlar informações em toda a classe, independente das instâncias.
- Declaradas através do modificador **static**

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    private static int total = 0;
    ...
}
```

- **Só vai existir uma variável *total*, não importando quantas instâncias forem criadas!**

- **Como funciona ???**



- **Serve para armazenar um valor, que é comum a todas as instâncias!**

- Exemplo: o construtor incrementa o total de pessoas

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    private static int total = 0;

    public Pessoa(int id, String n, String ender)
    {
        idade = id;
        nome = n;
        endereco = ender;
        total++; // incrementa o total de pessoas
    }
    ...
}
```

- Outra aplicação usual de variáveis de classe é a declaração de constantes:

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    private static int total = 0;
    // constante que armazena a idade a partir da qual
    // uma pessoa é considerada maior
    private final static int MAIOR = 21;
    ...
}
```

- Observe a utilização dos modificadores final e static
- Se quisermos declarar uma constante que possa ser utilizada por outras classes, utilizamos public:

```
public final static int MAIOR = 21;
```

### 3. Revisão: Métodos de instância e de classe

- **Métodos de instância**

- São métodos que só podem ser utilizados após a criação de um objeto, isto é, o que fazemos até agora.

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    ...

    public int retornaIdade()
    {
        return idade;
    }
}
```

- **Métodos de classe**

- São métodos que não dependem da existência de objetos para serem chamados. Por exemplo, podemos criar um método que retorna a quantidade de pessoas que já foram instanciadas:

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    private static int total = 0;
    ...

    public static int retornaTotal()
    {
        return total;
    }
}
```

- **Métodos de classe**

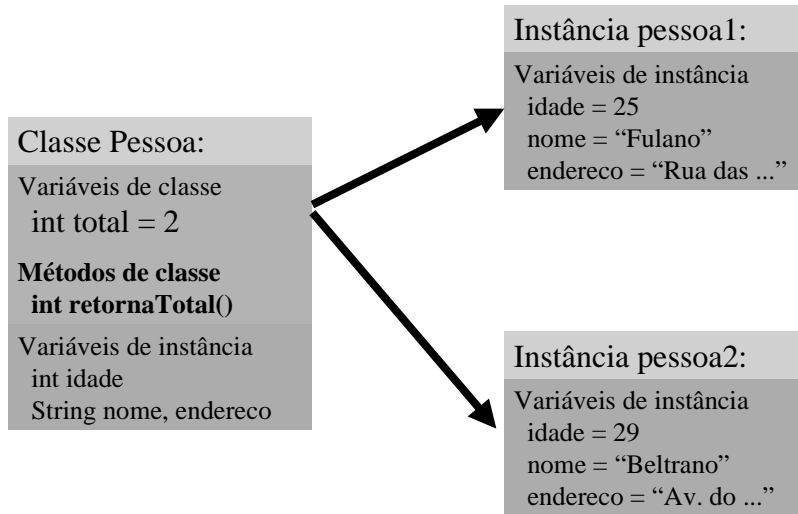
- **Obs: é possível retornar a variável total, pois ela é uma variável de classe**

```
class Pessoa
{
    private String nome, endereco;
    private int idade;
    private static int total = 0;
    ...

    public static int retornaTotal()
    {
        return total;
    }
}
```



- **Como funciona ???**



- **Como usar em outra classe (ou no main) ?**

```
class TestaPessoa
{
    public static void main(string args[])
    {
        Pessoa p1,p2;
        p1 = new Pessoa("Fulano","R. das ...",18);
        p2 = new Pessoa("Beltrano","Av. ...",21);
        // Agora, a variável de classe total
        // está armazenando o valor 2
        int total = Pessoa.retornaTotal();
        System.out.println(total);
    }
}
```

- Observe que usamos a sintaxe *Nome\_da\_classe.método*, ao invés de *nome\_da\_instância.método*

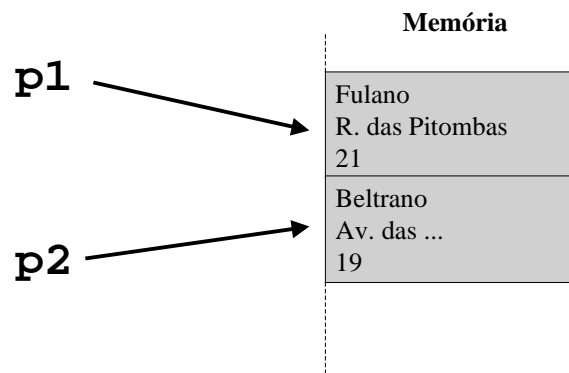
## 4. Conceito de “Referência”

- Considere o seguinte trecho de programa:

```
public static void main(String args[])
{
    Pessoa p1,p2; // duas pessoas
    p1 = new Pessoa("Fulano","R. das Pitombas",21);
    p2 = new Pessoa("Beltrano","Av. das ...",19);
}
```

- Como realmente estão sendo criados os dois objetos ?

```
public static void main(String args[])
{
    Pessoa p1,p2; // duas pessoas
    p1 = new Pessoa("Fulano","R. das Pitombas",21);
    p2 = new Pessoa("Beltrano","Av. das ...",19);
}
```

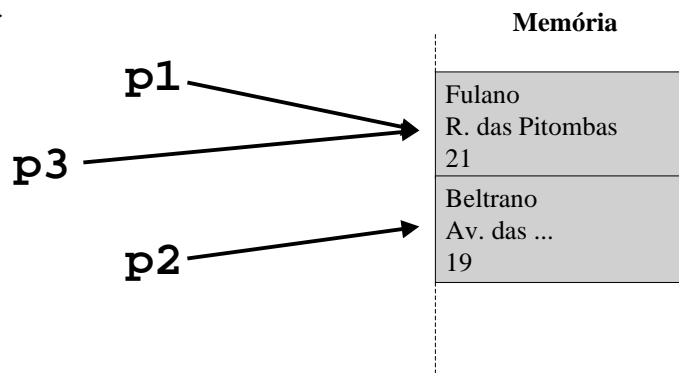


- p1 e p2 são referências às posições de memória ocupadas pelos objetos

- Então, se eu “atribuir” p1 ou p2 a outra variável, esta variável irá armazenar o quê ???

```
public static void main(String args[])
{
    Pessoa p1,p2,p3; // três pessoas ?
    p1 = new Pessoa("Fulano","R. das Pitombas",21);
    p2 = new Pessoa("Beltrano","Av. das ...",19);
    p3 = p1;
}
```

```
public static void main(String args[])
{
    Pessoa p1,p2,p3; // três pessoas ?
    p1 = new Pessoa("Fulano","R. das Pitombas",21);
    p2 = new Pessoa("Beltrano","Av. das ...",19);
    p3 = p1;
}
```

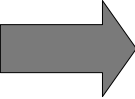


- Lembrando que p1 e p2 são referências, ao atribuir seu valor a outro objeto, este objeto passa a apontar para o mesmo conteúdo

## 5. Algumas classes úteis...

- **Wrappers**

- Para cada um dos tipos de dados básicos existe um tipo “empacotador” correspondente. São eles:

<b>Tipo primitivo</b>		<b>Classe correspondente</b>
boolean		Boolean
char		Character
int		Integer
long		Long
float		Float
double		Double

- Vantagem adicional: os *wrappers* oferecem um conjunto rico de métodos, para realizar as mais diversas tarefas.

### Exemplos de utilização da classe **Integer**:

- Como converter um inteiro para String ???

```
- String valor_s = Integer.toString(220);
```

- Como converter um inteiro para uma String binária ?

```
- String valor_bin = Integer.toBinaryString(220);
```

- Como converter uma String para inteiro ?

```
- int valor = Integer.parseInt("220");
```

- **Consulte a documentação !!!**

- **Sabemos que strings são implementadas através da classe String, mas como fazer outras coisas com elas ???**
  - Descobrir o tamanho
  - Obter uma parte da string
  - Procurar algo na string
  - Substituir caracteres
  - etc.
- A resposta é óbvia - através de métodos específicos, definidos pela classe String

```
String s1 = "As uvas estão verdes";  
String s2 = "Mad dog";  
String s3 = "Mad cat";  
...
```

**Métodos:**

```
s1.startsWith("As") => true  
s1.startsWith("As uvas") => true  
s2.equals(s3) => false  
s2.compareTo(s3) => > 0  
s3.compareTo(s2) => < 0  
s2 = s2.toLowerCase() => "mad dog"
```

```
String s1 = "As uvas estão verdes";  
String s2 = "Mad dog";  
String s3 = "Mad cat";  
...
```

**Métodos:**

```
int i; char c;  
i = s1.indexOf('e'); => 8  
i = s1.lastIndexOf('e'); => 18  
i = s1.indexOf('e',10); => 15  
i = s1.indexOf("uvas"); => 3  
c = s1.charAt(1); => 's'
```

```
String s1 = "As uvas estão verdes";  
String s2 = "Mad dog";  
String s3 = "Mad cat";  
...
```

**Métodos:**

```
String s4 = "hamburger".substring(4);  
=> "urger"  
String s5 = "hamburger".substring(4,8);  
=> "urge"  
String s5 = (new Integer(2)).toString();  
=> "2"  
String s6 = Integer.toString(2); => "2"
```

- Outra classe útil: **Character**

```
String s1 = "As uvas estão verdes";
String s2 = "Mad dog";
String s3 = "Mad cat";
...
char c = 'a';
Character.isLetter(c) => true
Character.isWhitespace(c) => false
c = Character.toLowerCase(s1.charAt(0))
=> 'a'
```

## 6. Pausa para Exercícios

- Crie uma classe para armazenar uma string. A classe deverá ter métodos para calcular o total de vogais, de consoantes e de espaços em branco. Crie também um programa que permita a digitação de uma string e chame esses métodos.

```
class MinhaString
{
    private String str;

    // construtor
    public MinhaString(String nova)
    {
        str = nova;
    }
}
```

### 3. Exercícios

```
class MinhaString
{
    private String str;

    // construtor
    public MinhaString(String nova)
    {
        str = nova;
    }

    public int contaEspacos()
    {
        int cont,total=0;
        char letra;
        for(cont=0;cont<str.length();cont++)
        {
            letra = str.charAt(cont); // pega a letra na posição
            if(letra == ' ') total++;
        }
        return total;
    }
}
```

### 3. Exercícios

...

```
public int contaVogais()
{
    int cont,total=0;
    char letra;
    for(cont=0;cont<str.length();cont++)
    {
        letra = str.charAt(cont); // pega a letra na posição
        letra = Character.toLowerCase(letra);
        if(letra == 'a' || letra == 'e' || letra == 'i' || letra == 'o'
           || letra == 'u') total++;
    }
    return total;
}

public int contaConsoantes()
{
    return str.length() - contaVogais() - contaEspacos();
}

} // fim da classe
```



### 3. Exercícios

- Crie uma classe Java capaz de armazenar um valor entre 0 e 100. A classe deve oferecer um método que retorne uma string contendo o valor por extenso. Ex: 23 => “vinte e três”. Faça um programa que exemplifique o uso.
- Faça um programa que crie um array de 10 posições, sendo que cada posição deverá conter um string arbitrário na forma dia/mês/ano (exemplo: “10/03/00”). Analise cada elemento do array e imprima as datas no formato “10 de março de 2000” (**não fazer até vermos arrays**).?

