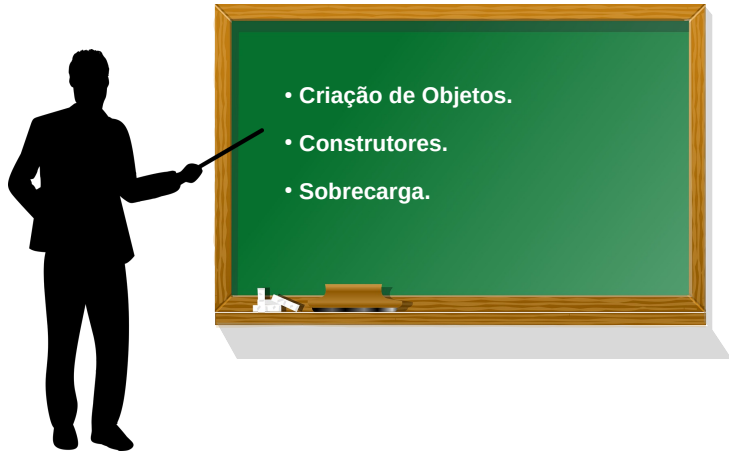


Criação de Objetos, Construtores e Sobrecarga

Luiz Henrique de Campos Merschmann
Departamento de Computação Aplicada
Universidade Federal de Lavras

luiz.hcm@ufla.br

Na Aula de Hoje



Criação de Objetos

- ▶ Para criar um objeto devemos usar a palavra-chave **new**.
- ▶ Vamos supor que temos uma classe **ContaBancaria** e desejamos criar um objeto a partir dessa classe. Como fazer?

```
ContaBancaria minhaConta;  
minhaConta = new ContaBancaria( );
```

- ▶ O que estamos fazendo na primeira linha? Criando um objeto?
 - ▶ Não!
 - ▶ E também não estamos declarando um objeto, mas sim uma variável que poderá referenciar um objeto.
- ▶ E na segunda linha, estamos criando um objeto?
 - ▶ Sim, pois quando utilizamos o comando **new**, o construtor da classe é chamado.
 - ▶ A atribuição (sinal =) faz com que a variável *minhaConta* referencie o objeto criado.

Criação de Objetos na Memória



Criação de Objetos na Memória

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //método para definir o nome
9     void setName(String umNome)
10    {
11        nome = umNome;
12    }
13
14    //método para recuperar o nome
15    String getNome()
16    {
17        return nome;
18    }
19 }
```

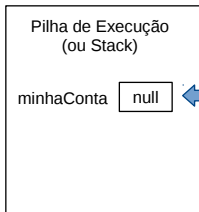
```
1 class ContaBancaria1Teste
2 {
3     public static void main(String[] args)
4     {
5         ContaBancaria1 minhaConta;
6         minhaConta = new ContaBancaria1();
7     }
8 }
```

Criação de Objetos na Memória

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //método para definir o nome
9     void setName(String umNome)
10    {
11        nome = umNome;
12    }
13
14    //método para recuperar o nome
15    String getName()
16    {
17        return nome;
18    }
19 }
```

```
1 class ContaBancaria1Teste
2 {
3     public static void main(String[] args)
4     {
5         ContaBancaria1 minhaConta;
6         minhaConta = new ContaBancaria1();
7     }
8 }
```

Reservando um espaço
da pilha de execução
para uma variável.



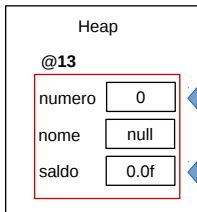
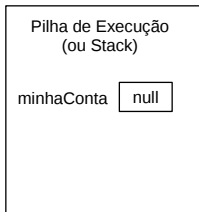
Em Java variáveis de uma classe
guardam o valor especial *null*, que
significa nenhum endereço.

Criação de Objetos na Memória

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //método para definir o nome
9     void setNome(String umNome)
10    {
11        nome = umNome;
12    }
13
14    //método para recuperar o nome
15    String getNome()
16    {
17        return nome;
18    }
19 }
```

```
1 class ContaBancaria1Teste
2 {
3     public static void main(String[] args)
4     {
5         ContaBancaria1 minhaConta;
6         minhaConta = new ContaBancaria1();
7     }
8 }
```

Além da pilha de execução, existe uma outra região de memória destinada ao programa denominada **Heap**.

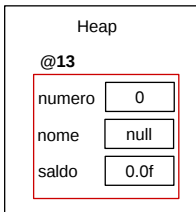
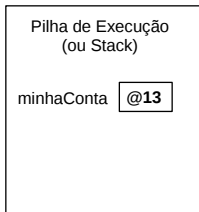



As variáveis de tipos primitivos em Java são inicializadas com valores padrões.

Criação de Objetos na Memória

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //método para definir o nome
9     void setNome(String umNome)
10    {
11        nome = umNome;
12    }
13
14    //método para recuperar o nome
15    String getNome()
16    {
17        return nome;
18    }
19 }
```

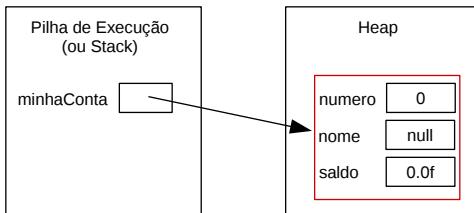
```
1 class ContaBancaria1Teste
2 {
3     public static void main(String[] args)
4     {
5         ContaBancaria1 minhaConta;
6         minhaConta = new ContaBancaria1();
7     }
8 }
```



Criação de Objetos na Memória

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //método para definir o nome
9     void setNome(String umNome)
10    {
11        nome = umNome;
12    }
13
14    //método para recuperar o nome
15    String getNome()
16    {
17        return nome;
18    }
19 }
```

```
1 class ContaBancaria1Teste
2 {
3     public static void main(String[] args)
4     {
5         ContaBancaria1 minhaConta;
6         minhaConta = new ContaBancaria1();
7     }
8 }
```



Variáveis e Objetos

- ▶ No exemplo anterior existe uma variável que guarda um endereço.
- ▶ Então essa variável não seria um ponteiro?
 - ▶ Certamente! Tecnicamente as variáveis de objetos em Java são ponteiros.
- ▶ Mas já ouvimos dizer que pelo fato de Java não ter ponteiros, ela é mais segura que C++. E agora?
 - ▶ Apesar de a variável ser um ponteiro, Java não permite que você faça as operações de ponteiro com a variável.
 - ▶ Isso ajuda a eliminar os principais riscos de uso de ponteiros.

Construtor

- ▶ Já aprendemos que quando o comando **new** é utilizado, além de se criar o objeto, o **construtor** do objeto é chamado.
 - ▶ O construtor é um método especial utilizado para inicializar os atributos de um objeto.
- ▶ O construtor é um método especial que:
 - ▶ **Não tem tipo de retorno**, pois retorna implicitamente a referência para o objeto.
 - ▶ Tem o **mesmo nome da classe**.
 - ▶ Assim como os demais métodos ele **pode ter parâmetros**.
 - ▶ E ele **nunca pode ser chamado diretamente** (chamado indiretamente pelo comando **new**).



Construtor

Vejamos um exemplo de um **construtor** para a classe **ContaBancaria1**.

```
1 class ContaBancaria1
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7
8     //construtor
9     ContaBancaria1(int umNumero, String umNome, float umSaldo)
10 {
11     numero = umNumero;
12     nome = umNome;
13     saldo = umSaldo;
14 }
15
16 // ...outros métodos
17 }
```

Os valores dos atributos são passados por parâmetro. Isso garante um estado inicial válido para o objeto.

Construtor Padrão

O que acontece se criarmos uma classe sem um método construtor? É possível criar objetos dessa classe?

- ▶ Sim, é possível criar objetos dessa classe.
- ▶ Isso porque Java cria (internamente) um construtor padrão (sem parâmetros).
- ▶ O construtor padrão inicializa os atributos dos objetos:
 - ▶ Com valores padrões para tipos primitivos (int, float ...).
 - ▶ Com *null* para tipos por referência (variáveis de objetos).

Construtor Padrão

- ▶ Se utilizássemos o construtor padrão da classe `ContaBancaria1`, seus atributos seriam inicializados com valores padrões:
 - ▶ número = 0
 - ▶ nome = *null*
 - ▶ saldo = 0.0f
- ▶ Mas uma conta bancária sem número é uma conta válida?
- ▶ Certamente não! Sem um número, um titular e um saldo não temos uma conta válida.
- ▶ Portanto, o construtor precisa garantir que o estado inicial dos objetos da classe seja um estado válido.

Construtor Padrão

Se existe o construtor padrão, para que criar um método construtor na minha classe?

- ▶ A classe é responsável por deixar o estado do objeto sempre válido.
- ▶ Usando apenas construtores padrões, poderíamos ter objetos com estados inválidos.

Se a classe tiver **apenas** um construtor com parâmetros, posso usar o construtor padrão do Java?

- ▶ Não, exceto se você tiver implementado um construtor igual ao construtor padrão do Java.
- ▶ Por que?
- ▶ Porque se Java permitisse isso seria impossível que o programador garantisse que os objetos tenham sempre um estado inicial válido.

Um Programa Java Completo

```
1 class ContaBancaria
2 {
3     //atributos
4     int numero; //numero da conta
5     String nome; //nome do titular da conta
6     float saldo; //saldo da conta;
7     /*A conta é construída com o nome e número
8     * informados e começa zerada.
9     */
10    ContaBancaria(int umNumero, String umNome)
11    {
12        numero = umNumero;
13        nome = umNome;
14        saldo = 0.0f;
15    }
16    //Retorna o numero da conta
17    int getNumero()
18    {
19        return numero;
20    }
21    //Retorna o nome do titular
22    String getNome()
23    {
24        return nome;
25    }
26    //Retorna o saldo da conta
27    float getSaldo()
28    {
29        return saldo;
30    }
```

Esta classe contém os atributos número, nome e saldo, além de um construtor que recebe o número e o nome do titular, e inicia o saldo com zero.

Existem métodos que nos permitem consultar o valor dos atributos número, nome e saldo.

Existem também métodos para depositar e sacar R\$ 50 da conta.



Repare também na utilização de comentários no código. Comentar o código é sempre uma **boa prática de programação!**

```
31 //Adiciona R$50 à conta
32 void depositar50()
33 {
34     saldo = saldo + 50;
35 }
36 //Retira R$50 da conta
37 void sacar50()
38 {
39     saldo = saldo - 50;
40 }
41 }
```


Um Programa Java Completo

Uma vez definida a classe *ContaBancaria*, para termos um programa mínimo, precisamos de uma **classe principal** que tenha o **método main**.

Nossa **classe principal** será denominada *ContaBancariaTeste* e, a partir dela, poderemos criar objetos da classe *ContaBancaria* e interagir com eles.



```
1  /** Classe principal que nos permite utilizar
2   * objetos da classe ContaBancaria
3   */
4  public class ContaBancariaTeste
5  {
6      public static void main(String[] args)
7      {
8          System.out.println("\nProtótipo de um Caixa Eletrônico");
9
10         ContaBancaria minhaConta = new ContaBancaria(1234, "Luiz Henrique");
11
12         System.out.printf("Conta nº %d de %s criada com
13         sucesso!\n", minhaConta.getNumero(), minhaConta.getNome());
14
15         System.out.println("Depositando dinheiro na conta...");
16         minhaConta.depositar50();
17         System.out.printf("Novo saldo da conta: R$ %.2f\n", minhaConta.getSaldo());
18     }
19 }
```

Os programas em Java começam com um método com essa assinatura.

O comando `System.out.println` serve para imprimir um texto na tela e mover o cursor para próxima linha.

Declaração da variável `minhaConta` que referencia um novo objeto da classe *ContaBancaria*.

Observem que estamos chamando quatro métodos da classe *ContaBancaria*

Sobrecarga de Métodos

Métodos com o **mesmo nome** podem ser declarados na mesma classe?

- ▶ Sim, desde que tenham diferentes conjuntos de parâmetros.
- ▶ Os conjuntos de parâmetros são determinados pelo **número**, **tipos** e **ordem** dos parâmetros.

Isso é o que chamamos de **sobrecarga de métodos**!

- ▶ Quando um método sobrecarregado é chamado, o compilador Java seleciona o método adequado examinando o **número**, os **tipos** e a **ordem** dos argumentos na chamada.

Sobrecarga de Métodos

Para que serve a sobrecarga de métodos?

- ▶ Ela é comumente utilizada para criar vários métodos com o **mesmo nome** que realizam as **mesmas tarefas** (ou tarefas **semelhantes**), mas sobre **conjuntos de parâmetros diferentes**.

Sobrecarga de Métodos

Posso criar dois métodos sobrecarregados com a mesma quantidade de parâmetros, dos mesmos tipos, mas com nomes diferentes?

- ▶ Exemplo: *depositar(float aluguel);*
depositar(float salario);
- ▶ O que pode ou não ser feito em termos de sobrecarga tem a ver com o que o compilador consegue identificar.
- ▶ No exemplo anterior, imagine a chamada:
minhaConta.depositar(2000.00);
- ▶ Como o compilador definiria qual dos métodos está sendo chamado?
- ▶ O compilador não consegue! Portanto, não podemos criar métodos sobrecarregados desse modo.

Sobrecarga de Métodos

E agora? Posso criar dois métodos sobrecarregados como os declarados a seguir?

- ▶ Exemplo: **void** *metodo1* (**int** *a*, **float** *b*);
 void *metodo1* (**float** *a*, **int** *b*);
- ▶ Sim, pois a ordem dos tipos de parâmetros é distinta!

Sobrecarga de Métodos

Eu posso criar dois métodos sobrecarregados com os mesmos parâmetros, mas com tipos de retorno diferentes?

- ▶ Exemplo: `int resgatar(String aplicacao);`
`float resgatar(String aplicacao);`
- ▶ Novamente, vamos pensar em um exemplo de chamada.
`int a = minhaConta.resgatar("poupanca");`
- ▶ Hummm... nesse caso parece que sim, pois o compilador sabe que a variável `a` é do tipo `int`.
- ▶ Mas e se eu não usar o valor retornado por uma chamada de método?
 - ▶ Por exemplo, eu poderia chamar o método da seguinte forma: `minhaConta.resgatar("poupanca");`
 - ▶ Nesse caso, o compilador não consegue definir qual método deverá ser chamado.
- ▶ Portanto, também não é possível declarar métodos sobrecarregados desse modo.

Sobrecarga de Construtores

Construtores também podem ser sobrecarregados...

- ▶ Assim como os métodos, posso ter mais de um construtor em uma classe.

Mas por que seriam necessários construtores diferentes?

- ▶ Dependendo do contexto, o programador pode dar mais de uma opção de inicialização dos objetos da classe.

Identidade dos Objetos

Considere o seguinte trecho de código:

```
ContaBancaria minhaConta = new ContaBancaria( );
```

```
ContaBancaria outraConta = new ContaBancaria( );
```

- ▶ Quantos objetos existem? Qual é a identidade de cada objeto?
 - ▶ Nós tendemos a pensar que a variável representa a identidade do objeto.

Considere agora esse outro trecho de código:

```
ContaBancaria minhaConta = new ContaBancaria( );
```

```
ContaBancaria outraConta = minhaConta;
```

- ▶ Quantos objetos existem? Qual é a identidade de cada objeto?
 - ▶ Esses exemplos mostram que a variável não pode ser a identidade do objeto. Veja que nesse último caso temos duas variáveis e apenas um objeto.

Identidade dos Objetos

Voltando ao trecho de código:

```
ContaBancaria minhaConta = new ContaBancaria( );
```

```
ContaBancaria outraConta = new ContaBancaria( );
```

- ▶ Os dois objetos são criados usando o mesmo construtor, portanto:
 - ▶ Eles têm os mesmos valores de atributos.
 - ▶ Possuem os mesmos métodos.
- ▶ Então o que define que eles são diferentes?
 - ▶ Já vimos que uma variável de objeto armazena o endereço do objeto na *Heap*.
 - ▶ Portanto, a **identidade do objeto** é dada pelo seu **endereço de memória**!

Comparando Objetos ou Variáveis?

Voltando ao nosso exemplo anterior:

```
ContaBancaria minhaConta = new ContaBancaria( );
```

```
ContaBancaria outraConta = new ContaBancaria( );
```

```
boolean saoIguais = (minhaConta == outraConta);
```

- ▶ Qual seria o valor da variável *saoIguais*? Por que?
 - ▶ O operador == está comparando variáveis (e não objetos)!
 - ▶ Nesse caso, está se comparando se as variáveis referenciam o mesmo objeto (mesmo endereço de memória).
 - ▶ Desse modo, ainda que os objetos sejam iguais (tenham o mesmo estado), a variável *saoIguais* recebe o valor *false* pelo fato de as variáveis referenciarem objetos distintos.

Comparando Objetos ou Variáveis?

Seguindo o mesmo raciocínio, analise o trecho de código a seguir:

```
ContaBancaria minhaConta = new ContaBancaria( );
```

```
ContaBancaria outraConta = minhaConta;
```

```
boolean saoIguais = (minhaConta == outraConta);
```

- ▶ Qual seria o valor da variável *saoIguais*? Por que?
 - ▶ Nesse caso o resultado é *true*, uma vez que as duas variáveis referenciam o mesmo objeto.

Comparando Objetos ou Variáveis?

Como fazer então para comparar se dois objetos são iguais?

- ▶ Poderíamos criar um método em uma classe para verificar se dois objetos são iguais?
- ▶ Como seria esse método?
 - ▶ Poderíamos criar em uma classe um método denominado *saoIguais* que receba por parâmetro um objeto da mesma classe.
 - ▶ O corpo desse método implementa a comparação entre os atributos do objeto em questão com aqueles do objeto passado por parâmetro.
- ▶ Como esse tipo de comparação é frequentemente necessário, a linguagem Java disponibiliza um método que implementa isso.

Perguntas?

