

Java Standard Edition (JSE)

11. O Pacote java.lang



Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com



Agenda

- O pacote `java.lang`;
- As classes `System`, `Runtime`, `Object`;
- Trabalhando com Casting;
- Sobrecarga de `Object.toString()`;
- Sobrecarga de `Object.equals()`;
- Classe `Wrapper(box)` e Autoboxing;
- A classe `java.lang.String`;
- Exercícios de fixação;



O pacote java.lang

- É comum utilizarmos as classes String e System;
- Mas...por que não precisamos fazer o `import` dessas classes?
- Resposta: Porque essas classes pertencem ao pacote java.lang;
- `java.lang` é o único pacote que é automaticamente importado pra você.



A classe System

- Possui vários métodos e atributos estáticos;
- Você já usou System.out? Pra quê?
- O atributo `out` é do tipo `PrintStream`, que pertence ao pacote `java.io`;
- A que classe pertence o método `println()`?



Atribuição simples

- Analise o código abaixo;
- Você já usou algo parecido?

```
1 package bean;  
2  
3 import java.io.PrintStream;  
4  
5 public class Arquivo {  
6     public static void main(String[] args) {  
7         PrintStream saida = System.out;  
8         saida.println("ola mundo!");  
9     }  
10 }
```



O atributo `System.in`

- A classe `System` possui o atributo `in`, utilizado para entrada de dados;
- O atributo `in` faz a captura byte a byte;
- `int i = System.in.read();`
- A linha acima exige blocos de try-catch, pois pode lançar uma exceção `IOException`;
- Falaremos mais a respeito de entrada de dados;



O método `System.exit(int i)`

- A classe `System` tem um método estático `exit(int i)`;
- O método `exit()` encerra a virtual machine;
- E devolve um código de erro para o Sistema Operacional;
- `System.exit(0)`;



A classe `java.lang.Runtime`

- A classe `Runtime` possui um método para fazer uma chamada ao sistema operacional e rodar algum programa:
 - `Runtime rt = Runtime.getRuntime();`
 - `Process p = rt.exec("dir");`
- Dependência do SO;
- Perca da portabilidade;
- Podemos substituir por uma tratativa mais genérica;



A classe Java.lang.Object

- Toda classe que criamos, é obrigada a herdar métodos e atributos de outra;
- Ou seja, toda classe que criamos em java tem, pelo menos, uma superclasse;
- Mas cadê a herança no código abaixo?

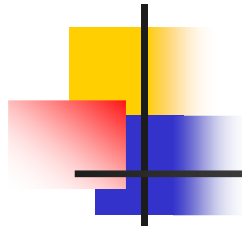
```
class MinhaClasse {  
  
}
```



A classe Java.lang.Object

- Quando não encontra a palavra **extends**, a JVM considera que você quer herdar da classe **java.lang.Object**;
- Object é a classe mãe de todas as outras;
- Você pode reescrever o código anterior:

```
class MinhaClasse extends Object {  
  
}
```



Trabalhando com casting

- O polimorfismo garante que possamos nos referir a qualquer objeto como **Object**;
- Um método que recebe um **Object** como argumento, pode, na verdade, receber qualquer “coisa”;
- Implementemos o código a seguir:



Armazenamento de Object

- Implementemos a classe Arquivo

```
1 package bean;
2
3 public class Arquivo {
4     1 private Object []arrayDeObjetos = new Object[10];
5     private int posicao = 0;
6
7     2 public void addObject(Object object){
8         this.arrayDeObjetos[this.posicao++]=object;
9     }
10
11     3 public Object getObject(int indice){
12         return this.arrayDeObjetos[indice];
13     }
14 }
```



Classe de teste

- O código abaixo compila?

```
package bean;

public class TestaArquivo {

    public static void main(String[] args) {
        1 Arquivo arquivo = new Arquivo();

        2 ContaBancaria c1 = new ContaBancaria();
        c1.depositar(500);
        arquivo.addObject(c1);

        3 ContaBancaria c2 = new ContaBancaria();
        c2.depositar(300);
        arquivo.addObject(c2);
    }
}
```



Itens a ponderar

- O que aconteceu no slide anterior?
- Quantos objetos **Arquivo** foram criados?
- E quantos objetos **ContaBancaria**?
- Poderíamos passar outro tipo de objeto para o método **addObject()**?
- Se tivéssemos uma classe **Cliente**, a linha a seguir funcionaria?
 - `arquivo.addObject(new Cliente());`



Acesso aos objetos arquivados

- Como fazer para acessar os objetos?

```
package bean;
public class TestaArquivo {
    public static void main(String[] args) {
        Arquivo arquivo = new Arquivo();
        ContaBancaria c1 = new ContaBancaria();
        c1.depositar(500);
        arquivo.addObject(c1);
        ContaBancaria c2 = new ContaBancaria();
        c2.depositar(300);
        arquivo.addObject(c2);

        //Acessando os objetos ContaBancaria
        Object objetoConta = arquivo.getObject(0);
        objetoConta.sacar(100);
    }
}
```



Itens a ponderar

- O que aconteceu no slide anterior?
- Por que o código não compila?
- Um **Object** não tem o método **sacar()**;
- O método **sacar()** é de **ContaBancaria**;
- Já sei! Que tal mudarmos o tipo?

```
//Acessando os objetos ContaBancaria  
ContaBancaria objetoConta = arquivo.getObject(0);  
objetoConta.sacar(100);
```




Pensando melhor

- Shiii! E agora?
- A variável `objetoConta` tem o método `sacar()`, pois é do tipo `ContaBancaria`;
- Nós adicionamos as variáveis de `ContaBancaria` C1 e C2;
- Por que ocorre o erro `Type mismatch: can not convert Object to ContaBancaria`?
- Como resolver?



Conversão de tipos de referência

- Pelo polimorfismo, incluímos 2 objetos ContaBancaria em um array de Object;
- Temos certeza que o objeto no array é uma ContaBancaria;
- Precisamos avisar à JVM que os Objects armazenados são ContaBancaria
- A esse aviso de compatibilidade, chamamos Casting;



Casting de referências

```
package bean;
public class TestaArquivo {
    public static void main(String[] args) {
        Arquivo arquivo = new Arquivo();
        ContaBancaria c1 = new ContaBancaria();
        c1.depositar(500);
        arquivo.addObject(c1);
        ContaBancaria c2 = new ContaBancaria();
        c2.depositar(300);
        arquivo.addObject(c2);

        //Acessando os objetos ContaBancaria
        ContaBancaria objetoConta = (ContaBancaria)arquivo.getObject(0);
        objetoConta.sacar(100);
    }
}
```



Exercício 01

- Implemente a classe para armazenamento de objetos:

```
public class ArmazenaObjetos {  
    private Object[] arrayDeObjetos = new Object[10];  
    private int posicao; // precisamos inicializar?  
  
    public void addObject(Object object) {  
        this.arrayDeObjetos[posicao++] = object;  
    }  
  
    public Object getObject(int indice) {  
        return this.arrayDeObjetos[indice];  
    }  
}
```

Exercício 02

- Implemente a classe de testes

```
3 public class TestaArmazenamento {  
4     public static void main(String[] args) {  
5         ArmazenaObjetos caixaDeObjetos = new ArmazenaObjetos();  
6         ContaBancaria c1 = new ContaBancaria();  
7         c1.depositar(500);  
8         caixaDeObjetos.addObject(c1);  
9         //Acesso ao objeto  
10        ContaBancaria c2 = null;  
11        c2 = (ContaBancaria)caixaDeObjetos.getObject(0);  
12        c2.depositar(150);  
13        System.out.println("Saldo: "+c2.getSaldo());  
14    }  
15 }
```

Problems @ Javadoc Declaration Console

<terminated> TestaArmazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 13:50:12)

Saldo: 650.0

O método

java.lang.Object.toString()

- O método `String toString()` de `Object` retorna:
 - `pacote.nome_da_classe@nro_identidade`

```
26 public static void main(String[] args) {
27     ContaBancaria minhaConta = new ContaBancaria();
28     minhaConta.depositar(300);
29     1 System.out.println(minhaConta);
30 }
```

Problems Javadoc Declaration Servers Progress Console

<terminated> ContaBancaria [Java Application] C:\Java\jre1.5.0_14\bin\javaw.exe (10/09/2010 16:44:37)

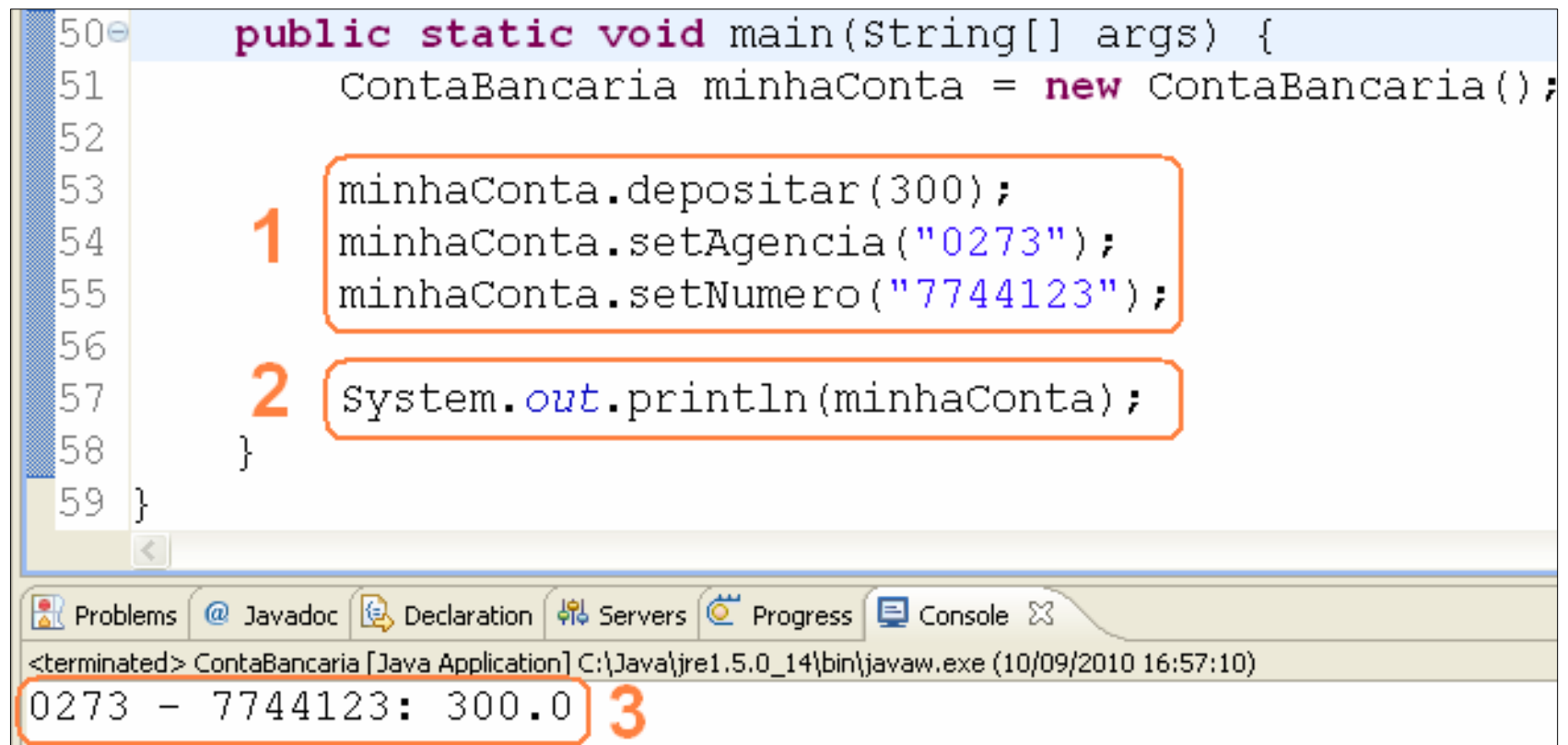
bean.ContaBancaria@82ba41 2

Sobrescrita do método toString()

- Cada classe filha de Object pode sobrescrever o método da herdado;

```
public class ContaBancaria {  
    private String numero;  
    private String agencia;  
    private double saldo;  
  
    public String toString() {  
        return agencia + " - " + numero + ": " + saldo;  
    }  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```

Chamada ao método sobrescrito



The screenshot shows an IDE window with a Java file. The code defines a `main` method that creates a `ContaBancaria` object and calls its methods. The code is as follows:

```
50 public static void main(String[] args) {  
51     ContaBancaria minhaConta = new ContaBancaria();  
52  
53     1 minhaConta.depositar(300);  
54     1 minhaConta.setAgencia("0273");  
55     1 minhaConta.setNumero("7744123");  
56  
57     2 System.out.println(minhaConta);  
58 }  
59 }
```

The code is annotated with orange boxes and numbers:

- 1** (orange) highlights the calls to `depositar`, `setAgencia`, and `setNumero` on lines 53, 54, and 55.
- 2** (orange) highlights the call to `System.out.println` on line 57.

The IDE's console window at the bottom shows the output of the program:

```
<terminated> ContaBancaria [Java Application] C:\Java\jre1.5.0_14\bin\javaw.exe (10/09/2010 16:57:10)  
0273 - 7744123: 300.0
```

The output is annotated with an orange box and the number **3** (orange), indicating the result of the `println` call.



java.lang.Object.equals()

- Quando comparamos variáveis de referência com `==`, é verificado se as duas variáveis apontam para o mesmo objeto;
- Ou seja, `==` é usado para verificar se o duas variáveis de referência apontam para o mesmo endereço de memória;

Comparação com ==

```
58 public static void main(String[] args) {  
59     1 ContaBancaria c1 = new ContaBancaria(300);  
60     ContaBancaria c2 = new ContaBancaria(300);  
61  
62     2 if(c1 == c2){  
63         System.out.println("Objetos de referência iguais");  
64     } else{  
65         System.out.println("Objetos de referência diferentes");  
66     }  
67 }  
68 }
```

Problems Javadoc Declaration Servers Progress Console

<terminated> ContaBancaria [Java Application] C:\Java\jre1.5.0_14\bin\javaw.exe (10/09/2010 17:49:28)

Objetos de referência diferentes 3



Problemas de comparação

- E se fosse necessário comparar os conteúdos dos objetos, ao invés de suas referências?
- Quais atributos seriam comparados?
- O java não tem como fazer essa escolha sozinho;
- O método `Object.equals()` nos permite criar esses critérios de comparação;



Object.equals(Object)

- O método equals() herdado de Object compara o objeto que chega como parâmetro com a instância que o recebe;
- Implementação padrão de equals():

```
public boolean equals(Object object) {  
    1 return this == object;  
}
```



Sobrescrita de método

- Podemos sobrescrever o método `equals()`, herdado da classe `Object`, para atender à nossa realidade:

```
12 @Override
13 public boolean equals(Object object) {
14     1 ContaBancaria contaParametro = (ContaBancaria)object;
15
16     2 if(this.saldo == contaParametro.saldo){
17         return true;
18     }else{
19         return false;
20     }
21 }
```



Por que sobrescrever `equals`?

- Poderíamos criar outro método para implementar nossa comparação de saldos?
- Por que usar a sobrescrita de `equals()`?
- O método `equals()` é importante porque é muito usado por muitas bibliotecas java, através do polimorfismo;
- Veremos mais quando estudarmos o pacote `java.util`

Qual a diferença?

```
8 public boolean equals(ContaBancaria contaParametro) {1
9     if(this.saldo == contaParametro.saldo){
10         return true;
11     }else{
12         return false;
13     }
14 }
15
16 public boolean equals(Object object) {
17
18     3 ContaBancaria contaParametro = (ContaBancaria)object;
19
20     if(this.saldo == contaParametro.saldo){
21         return true;
22     }else{
23         return false;
24     }
25 }
```

Problemas na sobrescrita

- Que problemas podem surgir com a implementação abaixo?
- E se o parâmetro enviado não for uma Conta bancária?

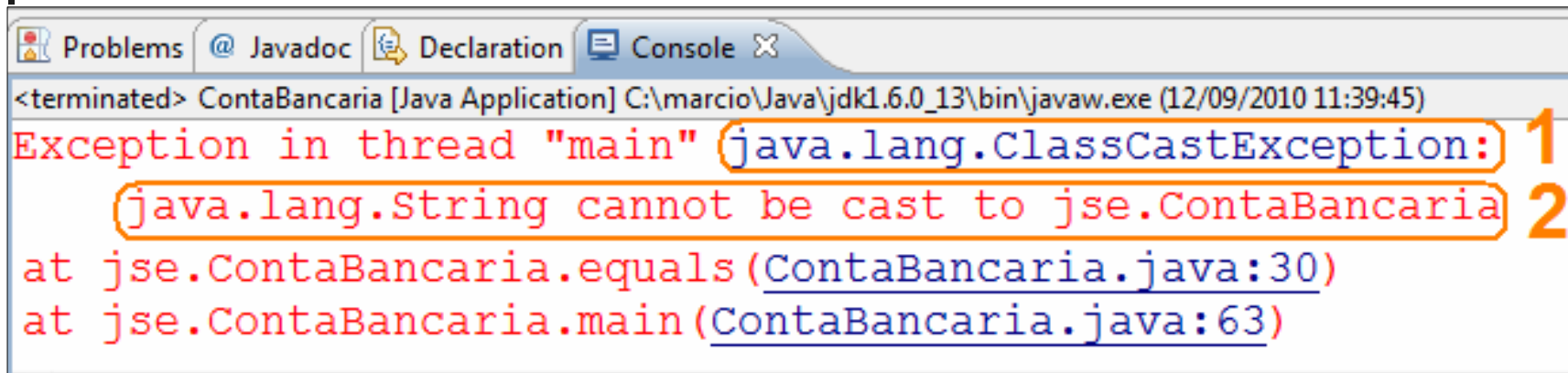
```
12 @Override
13 public boolean equals(Object object) {
14     1 ContaBancaria contaParametro = (ContaBancaria)object;
15
16     if(this.saldo == contaParametro.saldo){
17         return true;
18     }else{
19         2 return false;
20     }
21 }
```


Chamada ao método sobrescrito

- Qual o resultado da chamada abaixo?
- Há erro de compilação?
- Ocorre erro de execução? Qual?

```
public static void main(String[] args) {  
1  ContaBancaria c1 = new ContaBancaria(300);  
2  String testeSTR = "será?";  
3  System.out.println("resultado: "+c1.equals(testeSTR));  
}
```

Erro em tempo de execução



The screenshot shows a Java IDE console window with the following content:

```
<terminated> ContaBancaria [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 11:39:45)  
Exception in thread "main" java.lang.ClassCastException: 1  
    java.lang.String cannot be cast to jse.ContaBancaria 2  
at jse.ContaBancaria.equals(ContaBancaria.java:30)  
at jse.ContaBancaria.main(ContaBancaria.java:63)
```

The text is color-coded: red for error messages and blue for code paths. Two orange boxes highlight the exception type and the cast error message, with large orange numbers 1 and 2 to their right.

- Shiii!!! Ocorreu uma exceção. E agora?
- A exceção é Checked ou Unchecked?
- Como resolver o problema?
- Quais soluções podemos adotar?



Resolvendo o problema

- Vamos tratar a ClassCastException com blocos de try-catch?
- Vamos lançar a exceção com throws?
- Que tal evitarmos o erro?

```
public boolean equals(Object object) {  
    1  if(!(object instanceof ContaBancaria)){  
        return false;  
    }  
    2  ContaBancaria contaParametro = (ContaBancaria)object;  
    return this.saldo == contaParametro.saldo;  
}
```



Exercício 03

- Crie dois construtores para a classe ContaBancaria:
 - Um sem argumentos;
 - E outro recebendo o saldo inicial;

```
public ContaBancaria() {  
    1 //Por que criar este método?  
    //Pra que ele serve?  
}  
  
    2  
public ContaBancaria(double saldo) {  
    3 this.saldo = saldo;  
}
```



Exercício 04

- Sobrescreva os métodos `toString()` e `equals()` da classe `ContaBancaria`;

```
18 @Override
19 1 public String toString() {
20     return "Saldo: "+this.saldo;
21 }
22
23 @Override
24 2 public boolean equals(Object object) {
25     if (!(object instanceof ContaBancaria)) {
26         return false;
27     }
28     ContaBancaria contaParametro = (ContaBancaria) object;
29     return this.saldo == contaParametro.saldo;
30 }
```

Exercício 05

■ Teste os novos métodos:

```
55 public static void main(String[] args) {  
56     ContaBancaria c1 = new ContaBancaria(300);  
57     1 System.out.println("Conta c1 - "+c1);  
58     String testeSTR = "será?";  
59     System.out.println("TesteSTR: " + c1.equals(testeSTR));  
60  
61     2 ContaBancaria c2 = new ContaBancaria(300);  
62     System.out.println("Conta c2 - "+c2);  
63     System.out.println("TesteConta: " + c1.equals(c2));  
64 }
```

Problems @ Javadoc Declaration Console X
<terminated> ContaBancaria [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 12:42:06)

Conta c1 - Saldo: 300.0
TesteSTR: false 1.1

Conta c2 - Saldo: 300.0
TesteConta: true 2.1



Classes Wrapper

- Um problema comum de programação é:
 - Como converter uma String em um número e vice-versa?
- O que acontece quando executamos:
 - `System.out.println("Idade: "+18)?`
 - `E System.out.println("Peso: "+32.5)?`
- Para convertermos números para String, precisamos apenas concatená-los a uma String válida;



E como converter String em números?

- Já vimos que String não é um tipo primitivo;
- Com isso, precisamos da ajuda de outro tipo de referência;
- Nesse contexto surgem as classes Wrapper, que permitem que tipos primitivos sejam tratados como variáveis de referência;

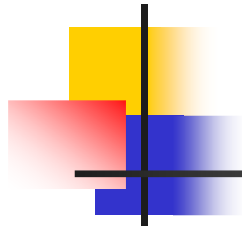


Convertendo em numéricos

- Cada tipo primitivo tem um wrapper adequado. Exemplo:
 - Integer(int), Float(float) e Double(double)
- Para conversão de String em Integer, usamos:

```
String s = "101";  
int i = Integer.parseInt(s);
```

- Float e Double possuem os métodos de conversão `parseFloat` e `parseDouble`



Empacotando tipos primitivos

- As classes de wrapper podem ser usadas, também, para embrulhar (wrapping) tipos primitivos como objetos;
- Qualquer classe wrapper é um Object;
- Com isso, podemos criar variáveis de referência a partir de tipos primitivos;



Armazenando tipos primitivos

- O que aconteceria se tentássemos guardar **inteiros** na nossa classe ArmazenaObjetos?

```
1 ArmazenaObjetos caixa = new ArmazenaObjetos();  
  int inteiroPrimitivo = 10;  
  caixa.addObject(inteiroPrimitivo);  
  //Acesso ao objeto  
2 int teste = (int)caixa.getObject(0);
```



Itens a ponderar

- Por que conseguimos armazenar um `int` ao invés de um `Object`?
- Que magia foi essa?
- A partir da versão 1.5, a JVM consegue fazer o wrapper e o unwrapper pra voce, de forma automática;
- A esse processo automático, chamamos Autoboxing;



O que aconteceu?

- O código:

1 `ArmazenaObjetos caixa = new ArmazenaObjetos();
int inteiroPrimitivo = 10;
caixa.addObject(inteiroPrimitivo);`

- Foi convertido para:

2 `ArmazenaObjetos caixa = new ArmazenaObjetos();
int inteiroPrimitivo = 10;
caixa.addObject(new Integer(inteiroPrimitivo));`



E como fazer o unwrapper?

```
ArmazenaObjetos caixa = new ArmazenaObjetos();  
int inteiroPrimitivo = 10;  
caixa.addObject(inteiroPrimitivo);  
//Acesso ao objeto  
Integer wrapper = (Integer)caixa.getObject(0);  
int teste = wrapper.intValue();  
System.out.println("valor: "+teste);
```

- As classes **Float** e **Double** possuem os métodos **floatValue** e **doubleValue**, respectivamente;

Usando o autoboxing

- O que acontece no código abaixo?

```
4 public static void main(String[] args) {  
5     Integer x = new Integer(10);  
6     int y = x;  
7     x = 2;  
8     System.out.println(x);  
9     System.out.println(y);  
10 }
```

1

2
10 2

java.lang.String – Exercício 06

- Implemente a comparação a seguir:

```
4 public static void main(String[] args) {  
5     1 String primeiro = "abc";  
6     String segundo = "abc";  
7  
8     2 if(primeiro == segundo){  
9         System.out.println("Referências para o mesmo objeto");  
10    }else{  
11        System.out.println("Referências para objetos diferentes");  
12    }  
13 }
```

Problems Javadoc Declaration Console

<terminated> TestaArmazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 15:20:01)

Referências para o mesmo objeto 3



Exercício 07

- Implemente outro código de comparação entre Strings;
- Sempre que solicitado, informe o valor **abc**

```
public static void main(String[] args) {  
1 String primeiro = JOptionPane.showInputDialog("Primeiro:");  
  String segundo = JOptionPane.showInputDialog("Segundo:");  
  if(primeiro == segundo){  
2      JOptionPane.showMessageDialog(null, "Referências para o mesmo objeto");  
  }else{  
      JOptionPane.showMessageDialog(null, "Referências para objetos diferentes");  
  }  
}
```

Exercício 07 - Resultado

The image shows three overlapping dialog boxes from a software application. The top two are 'Input' boxes, and the bottom one is a 'Message' box. Each box has a title bar with a close button (X) in the top right corner. The 'Input' boxes have a green question mark icon on the left and a text input field. The 'Message' box has a blue information icon (i) on the left and a message text area. All text input fields and the message text area are highlighted with an orange border.

Input

? Primeiro:

abc

OK Cancel

Input

? Segundo:

abc

OK Cancel

Message

i Referências para objetos diferentes

OK

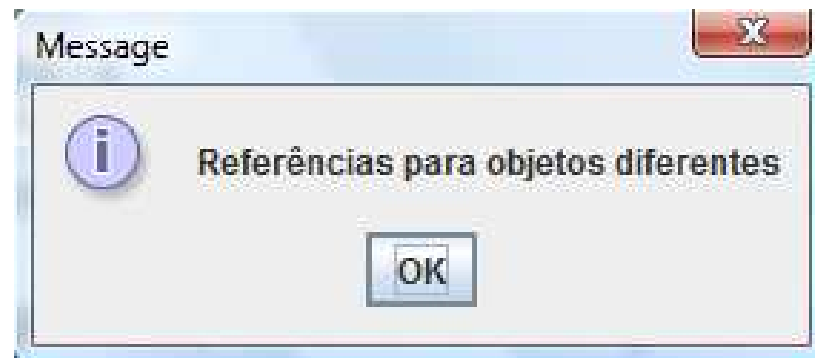


Itens a ponderar

- Por que dois tipos de comparação de String usando == tiveram resultados diferentes?
- O operador == verifica se as duas variáveis apontam para o mesmo objeto;
- Nos exercícios 6 e 7 temos as variáveis **primeiro** e **segundo**, do tipo String;

Itens a ponderar

- Quando criamos as duas Strings, cada uma está apontando para um endereço diferente, mesmo que tenham o mesmo conteúdo "abc";
- E foi o que aconteceu no exercício 07:





Economia de memória

- No Exercício 06, atribuímos o valor da Strings em tempo de desenvolvimento:

```
String primeiro = "abc";  
String segundo  = "abc";
```

- A fim de economizar espaço de memória, a JVM cria um buffer de Strings;
- Com isso, temos duas referências apontando para o mesmo objeto **abc**



Comparação do conteúdo de duas variáveis de String

- Vimos que **==** é usado para comparar se as variáveis apontam para o mesmo objeto;
- Para comparar o conteúdo de duas Strings, usamos o método **equals()**:

```
if(primeiro.equals(segundo)) {  
    System.out.println("Mesmo conteúdo");  
}else{  
    System.out.println("Conteúdos diferentes");  
}
```

O método String.split()

- Divide uma String em um array de Strings, de acordo com o critério;

```
1 //Criação da String
String mensagem = "Treinamento Java avançado";

2 //Criação do Array de Strings
String []arrayStr = mensagem.split(" ");

3 //Acesso aos dados do Array
for (String palavra : arrayStr) {
    System.out.println(palavra);
}
```

Javadoc Declaration Console

Armazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 22:38:54)

```
4 Treinamento
   Java
   avançado
```

Os métodos de String: toUpperCase e toLowerCase

- O que aconteceu com a variável base?

```
1 String mensagem = "Treinamento Java avançado";  
  //Criação da String em MAIUSCULO e minuscuro  
  String maiusculo = mensagem.toUpperCase();  
  String minuscuro = mensagem.toLowerCase();  
  
2 System.out.println("Mensagem: "+mensagem);  
  System.out.println("Maiusculo: "+maiusculo);  
  System.out.println("Minuscuro: "+minuscuro);  
  
3 Mensagem:  Treinamento Java avançado  
  Maiusculo: TREINAMENTO JAVA AVANÇADO  
  Minuscuro: treinamento java avançado
```




O método String.replace()

- Atualiza elementos de uma String, de acordo com seus parâmetros;

```
//Troca de elementos da String
1 String arroba = mensagem.replace("a", "@");
  System.out.println("Mensagem: "+mensagem);
  System.out.println("Arroba:    "+arroba);
```

javadoc Declaration Console

Armazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 23:07:

```
2 Mensagem: Treinamento Java avançado
  Arroba:   Trein@mento J@v@ @v@nç@do
```

Concatenação de métodos

- Podemos chamar dois métodos, no mesmo comando:

```
String mensagem = "Treinamento Java avançado";  
1 //Concatenacao de metodos  
String nova = mensagem.toUpperCase().replaceAll("A", "@");  
System.out.println("Mensagem: "+mensagem);  
System.out.println("Nova:      "+nova);
```

Javadoc Declaration Console

Armazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw.exe (12/09/2010 23:14:31)

```
2 Mensagem: Treinamento Java avançado  
Nova:      TREIN@MENTO J@V@ @V@NÇ@DO
```



Outros métodos de String

- `charAt(i)`, retorna o caractere existente na posição `i` da String;
- `length` retorna o número de caracteres;
- `substring` que recebe um `int` e devolve a SubString a partir da posição `int`;
- `indexOf` recebe uma String e devolve o índice em que aparece pela primeira vez na String principal;
- `isEmpty` devolve `true` para String vazia;

A classe java.lang.Math

- Possui uma série de métodos estatísticos

```
//Arredondamento
1 double d = 4.6;
  long i = Math.round(d);
  System.out.println(d + " => " + i);

//Tira o valor absoluto
2 int x = -4;
  int y = Math.abs(x);
  System.out.println(x + " => " + y);
```

Javadoc Declaration Console

Armazenamento [Java Application] C:\marcio\Java\jdk1.6.0_13\bin\javaw

```
3 4.6 => 5
   -4 => 4
```



Considerações

- As classes String e Math possuem uma grande variedade de métodos;
- Tenha o hábito de consultar a documentação do JAVA, a fim de entender e pesquisar métodos que facilitem suas atividades;



Exercícios

- Implemente uma classe Java para testar os métodos da classe String;
- Teste, também, os métodos da classe Math;



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

- SUN: www.java.sun.com

Fóruns e listas:

- Javaranch: www.javaranch.com
- GUJ: www.guj.com.br

Apostilas:

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br

Java Standard Edition (JSE)

11. O Pacote java.lang



Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com