



TIPOS DE DADOS E CONVERSÕES

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

- Tipos primitivos
- Classe `Math`
- Tipos 'ponto-flutuante'
- Internacionalização
- Constantes
- Referências vs. Valores
- Objetos vs. Tipos primitivos
- *Wrappers* classes
- *Boxing* e *unboxing*
- Conversões e *casting*
- Tipo `NULL`
- Desafio

TIPOS PRIMITIVOS

■ Tabela dos tipos primitivos em Java

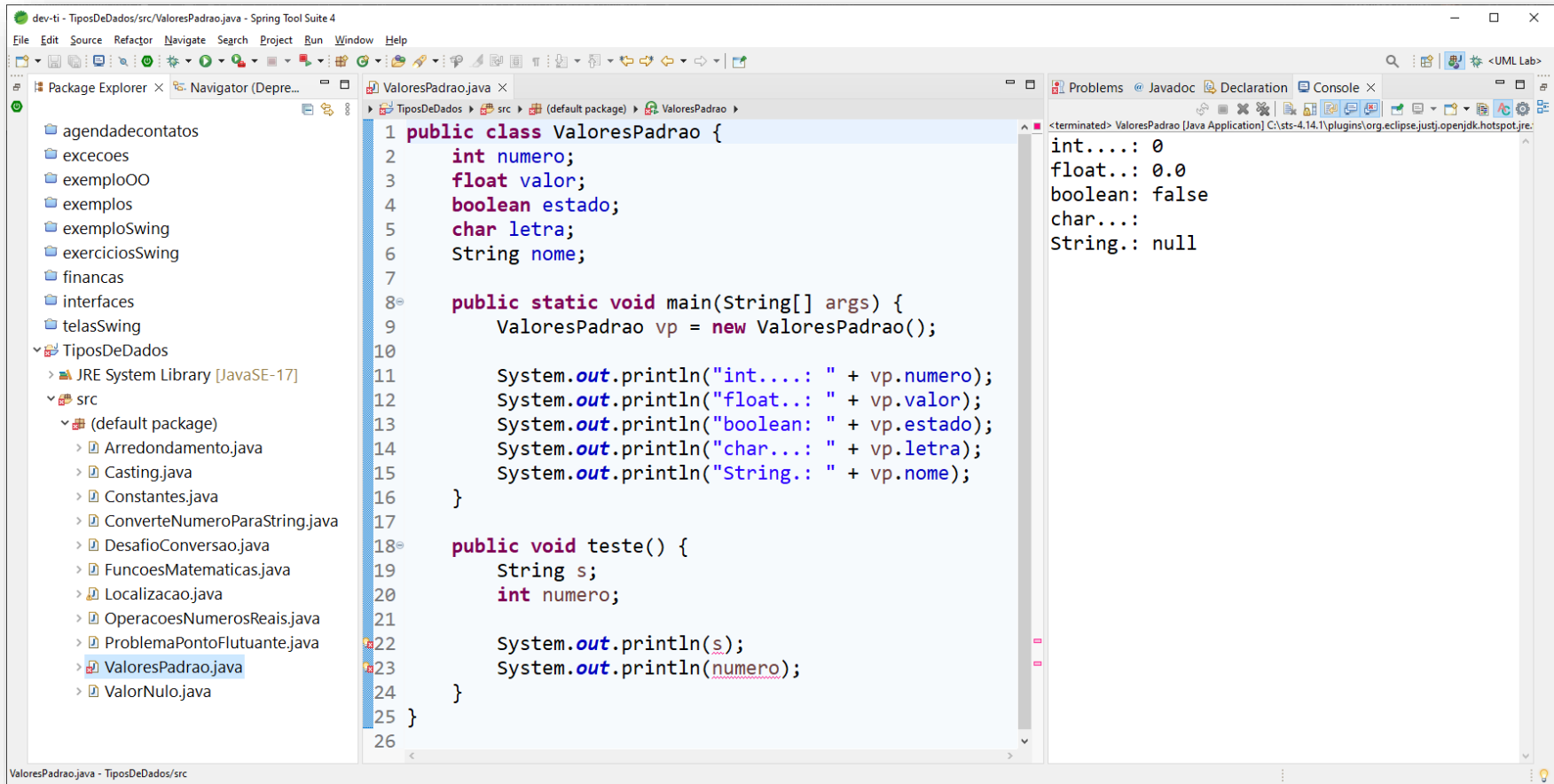
Descrição	Tipo	Tamanho	Valores	Valor padrão
tipos numéricos inteiros	byte	8 bits	-128 a 127	0
	short	16 bits	-32768 a 32767	0
	int	32 bits	-2147483648 a 2147483647	0
	long	64 bits	-9223372036854770000 a 9223372036854770000	0L
tipos numéricos com ponto flutuante	float	32 bits	-1,4024E-37 a 3,4028E+38	0.0f
	double	64 bits	-4,94E-307 a 1,79E+308	0.0
um caractere Unicode	char	16 bits	'\u0000' a '\uFFFF'	'\u0000'
valor verdade	boolean	1 bit	{false, true}	false

TIPOS PRIMITIVOS

- Valores padrão para atributos
 - Números inteiros: Valor 0
 - Números de ponto-flutuante: Valor 0.0
 - *Booleans*: false
 - Tipo `char`: Caractere código 0
 - Objetos, incluindo o objeto `String`: Valor `null`
- No caso de variáveis locais, se elas não forem inicializadas o compilador Java indicará um erro, pois não as inicializa automaticamente
 - Ou seja, para variáveis locais, a inicialização é obrigatória

TIPOS PRIMITIVOS

Exemplo de valores padrão



The screenshot shows an IDE window titled "dev-ti - TiposDeDados/src/ValoresPadrao.java - Spring Tool Suite 4". The left sidebar displays a project structure with folders like "agendadecontatos", "excecoes", "exemploOO", "exemplos", "exemploSwing", "exerciciosSwing", "financas", "interfaces", "telasSwing", and a package "TiposDeDados" containing a "src" folder with several Java files, including "ValoresPadrao.java". The main editor shows the code for "ValoresPadrao.java":

```
1 public class ValoresPadrao {
2     int numero;
3     float valor;
4     boolean estado;
5     char letra;
6     String nome;
7
8     public static void main(String[] args) {
9         ValoresPadrao vp = new ValoresPadrao();
10
11         System.out.println("int....: " + vp.numero);
12         System.out.println("float...: " + vp.valor);
13         System.out.println("boolean: " + vp.estado);
14         System.out.println("char...: " + vp.letra);
15         System.out.println("String.: " + vp.nome);
16     }
17
18     public void teste() {
19         String s;
20         int numero;
21
22         System.out.println(s);
23         System.out.println(numero);
24     }
25 }
26
```

The right sidebar shows the "Problems" tab with a message: "<terminated> ValoresPadrao [Java Application] C:\sts-4.14.1\plugins\org.eclipse.justj.openjdk.hotspot.jre:". Below this, the default values for primitive types are listed:

```
int....: 0
float...: 0.0
boolean: false
char...:
String.: null
```

TIPOS PRIMITIVOS

- Nomes de identificadores

- Não podem iniciar com dígito
- Podem começar com letra ou _ (sublinhado/*underscore*)
- Não podem ter espaço em branco
- Não usar acentos

- Utilize o padrão camelCase para nomes de variáveis



- Nomes de classes devem seguir o padrão PascalCase



CLASSE MATH

- A classe `java.lang.Math` do Java contém métodos que realizam operações numéricas básicas como exponenciais, logaritmos, trigonometria, raiz quadrada, etc

Método	Descrição
<code>Math.sqrt(x)</code>	Raiz quadrada de x
<code>Math.pow(x, y)</code>	Base x elevado à potência y
<code>Math.abs(x)</code>	Valor absoluto

```
modulo2 - FuncoesMatematicas.java

1 public class FuncoesMatematicas {
2     public static void main(String[] args) {
3         System.out.println("2^3 = " + Math.pow(2, 3));
4         System.out.println("Raiz quadrada de 25 = " + Math.sqrt(25));
5         System.out.println("Valor absoluto de -2 = " + Math.abs(-2));
6     }
7 }
```

Apoiadores:

TIPOS 'PONTO-FLUTUANTE'

- Os tipos `double` e `float` apresentam casas decimais, logo são também chamadas de números reais, ou de ponto flutuante
- É preciso ter atenção ao executar operações matemáticas com este tipo de dado para que não haja perda de precisão no resultado

```
modulo2 - OperacoesNumerosReais.java

1 public class OperacoesNumerosReais {
2
3     public static void main(String[] args) {
4         // Divisões                                Resultado
5         System.out.println("10/3 é igual a " + 10/3);        // 3
6         System.out.println("10./3 é igual a " + 10./3);      // 3.3333333333333335
7         System.out.println("10/3. é igual a " + 10/3.);      // 3.3333333333333335
8         System.out.println("10./3. é igual a " + 10./3.);    // 3.3333333333333335
9
10        double n1 = 10;
11        double n2 = 3;
12        double resultado = n1/n2;
13
14        System.out.println("Divisão de 'doubles': " + resultado); // 3.3333333333333335
15    }
16
17 }
```


TIPOS 'PONTO-FLUTUANTE'

- Exemplos de arredondamento com as classes `DecimalFormat` e `String`

```
modulo2 - Arredondamento.java

1 import java.text.DecimalFormat;
2 import java.util.Locale;
3
4 public class Arredondamento {
5     public static void main(String[] args) {
6         // Arredondamento
7         final DecimalFormat df = new DecimalFormat("0.0");
8
9         double valor1 = 9.999;
10        double valor2 = 9.41;
11        double valor3 = 9.46;
12
13        Locale.setDefault(Locale.US);
14        System.out.println("9.999 arredondado para duas casas decimais: " + String.format("%.2f", valor1));
15        System.out.println("Arredondamento para baixo: " + df.format(valor2));
16        System.out.println("Arredondamento para cima: " + df.format(valor3));
17    }
18 }
```

TIPOS 'PONTO-FLUT

- 'Problema' com tipos ponto-flutuante

```
modulo2 - Localizacao.java
1 import java.text.DecimalFormat;
2 import java.util.Locale;
3
4 public class ProblemaPontoFlutuante {
5     public static void main(String[] args) {
6         // Problema com números em ponto flutuante
7         double a = 0.1;
8         double b = 0.2;
9         double c = a + b;
10
11         if (c != 0.3) {
12             System.out.println("Algo estranho está acontecendo: 0,1 + 0,2 = " + c);
13         }
14
15         // 1a. solução: Arredondamento 'manual'
16         c = Math.round((a + b) * 100) / 100.;
17
18         if (c == 0.3) {
19             System.out.println("Método 1, ok! 0,1 + 0,2 = " + c);
20         }
21
22         // 2a. solução: String.format()
23         String str = String.format(Locale.US, "%.2f", (a + b));
24         c = Double.valueOf(str);
25
26         if (c == 0.3) {
27             System.out.println("Método 2, ok! 0,1 + 0,2 = " + c);
28         }
29
30         // 3a. solução: DecimalFormat.format()
31         Locale.setDefault(Locale.US);
32         String res = new DecimalFormat("0.00").format(a + b);
33         c = Double.valueOf(str);
34
35         if (c == 0.3) {
36             System.out.println("Método 3, ok! 0,1 + 0,2 = " + c);
37         }
38     }
39 }
40
```

INTERNACIONALIZAÇÃO

- O recurso de 'Localização' (internacionalização) adapta o formato dos números reais (ponto ou vírgula flutuante) a formatos regionais

```
modulo2 - Localizacao.java

1 import java.text.NumberFormat;
2 import java.util.Locale;
3 import java.util.Scanner;
4
5 public class Localizacao {
6     public static void main(String[] args) {
7         // O Locale é preciso estar antes do Scanner
8         Locale.setDefault(Locale.US);
9         Scanner sc = new Scanner(System.in);
10
11         System.out.print("Digite um número real (com ponto decimal): ");
12         double valor = sc.nextDouble();
13
14         System.out.print(NumberFormat.getCurrencyInstance().format(valor));
15         System.out.printf(" || %.2f\n\n", valor);
16
17         // Padrão alemão
18         Locale.setDefault(Locale.GERMANY);
19         sc = new Scanner(System.in);
20
21         System.out.print("Digite um número real (com vírgula decimal): ");
22         valor = sc.nextDouble();
23
24         System.out.print(NumberFormat.getCurrencyInstance().format(valor));
25         System.out.printf(" || %.2f\n\n", valor);
26
27         // Padrão nacional
28         Locale.setDefault(new Locale("pt", "BR"));
29         sc = new Scanner(System.in);
30
31         System.out.print("Digite um número real (com vírgula decimal): ");
32         valor = sc.nextDouble();
33
34         NumberFormat nf = NumberFormat.getCurrencyInstance();
35         // nf.setGroupingUsed(false); // Agrupamento de dígitos
36         String resultado = nf.format(valor);
37
38         System.out.print(resultado);
39         System.out.printf(" || %.2f\n", valor);
40     }
41 }
```

CONSTANTES

- Constantes são valores fixos que não se modificam no decorrer da execução de um programa, podendo ser, por exemplo, dos tipos
 - Numérica
 - *String*
 - *Booleana* (lógica)
- Por convenção constantes são escritas em letras MAIÚSCULAS utilizando o padrão SCREAMING_SNAKE_CASE



CONSTANTES

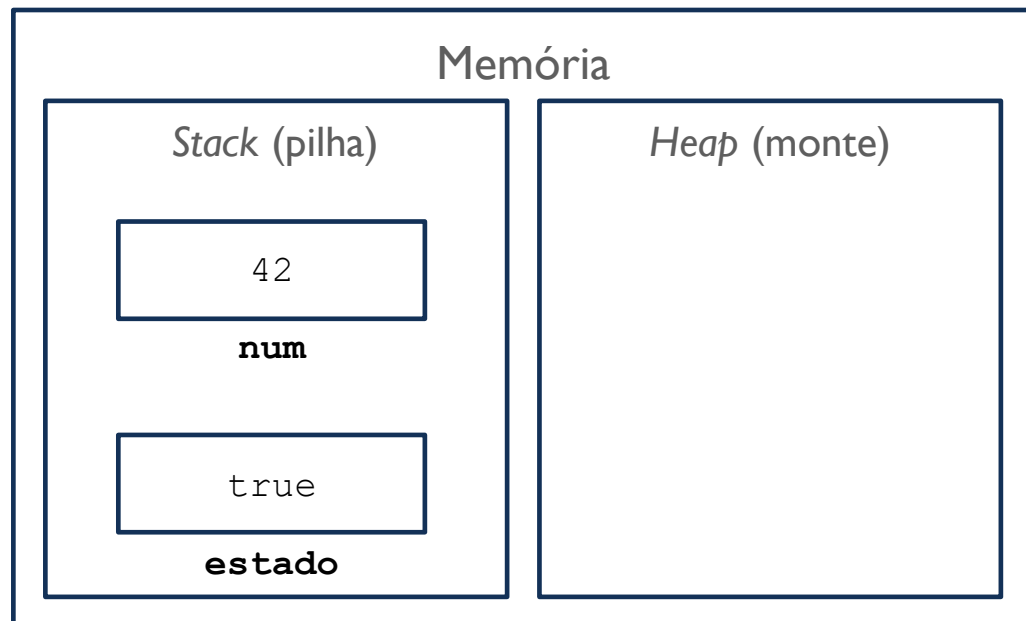
■ Exemplo

```
modulo2 - Constantes.java
1  public class Constantes {
2      // Floats literais exigem o sufixo 'f'
3      final static float NUMERO = 42.42f;
4
5      // Longs exigem o sufixo 'L'
6      final static long NUMERO_LONGO = 10_550_430_001L;
7
8      final static double PI = 3.14159;
9      final boolean STATUS = true;
10
11     public static void main(String[] args) {
12         System.out.println(Constantes.NUMERO);
13         System.out.println(Constantes.NUMERO_LONGO);
14         System.out.println(Constantes.PI);
15
16         System.out.println(new Constantes().STATUS);
17
18         // Definição de constantes locais
19         final String UNIVERSIDADE = "Unoesc";
20
21         // UNIVERSIDADE = "Unoutra"; // *** ERRO ***
22
23         System.out.println(UNIVERSIDADE);
24     }
25 }
```

REFERÊNCIAS VS. VALORES

- Variáveis dos tipos primitivos podem ser entendidas como caixas dentro na memória do computador, em uma área chamada de *stack* (pilha)

- `int num = 42`
- `boolean estado = true`



REFERÊNCIAS VS. VALORES

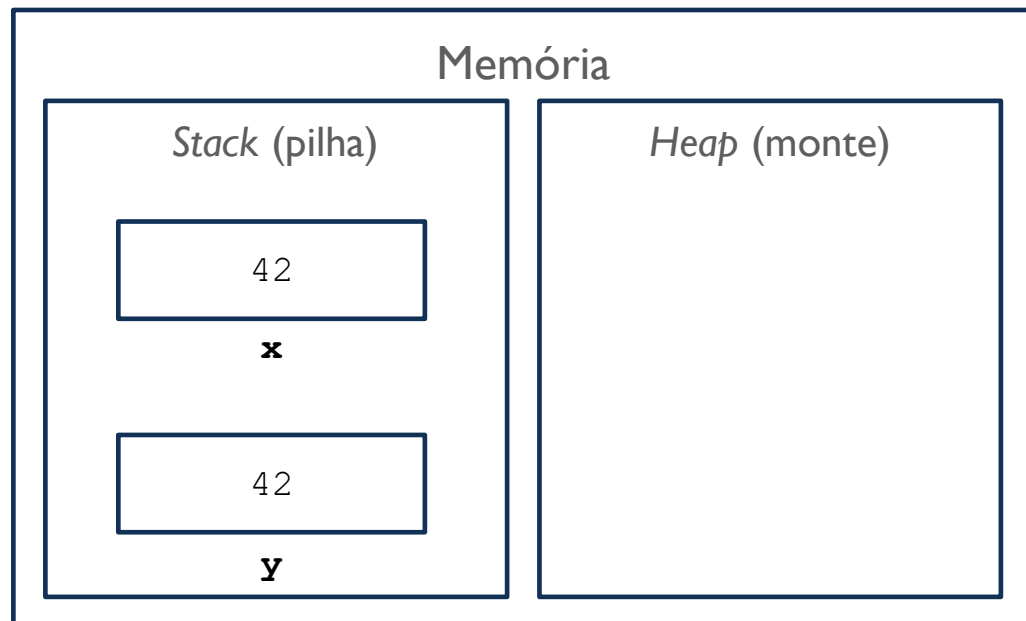
- Ao se fazer uma atribuição entre tipos primitivos, as 'caixas' permanecem com seus valores independentes

- `int x, y;`

- `x = 42;`

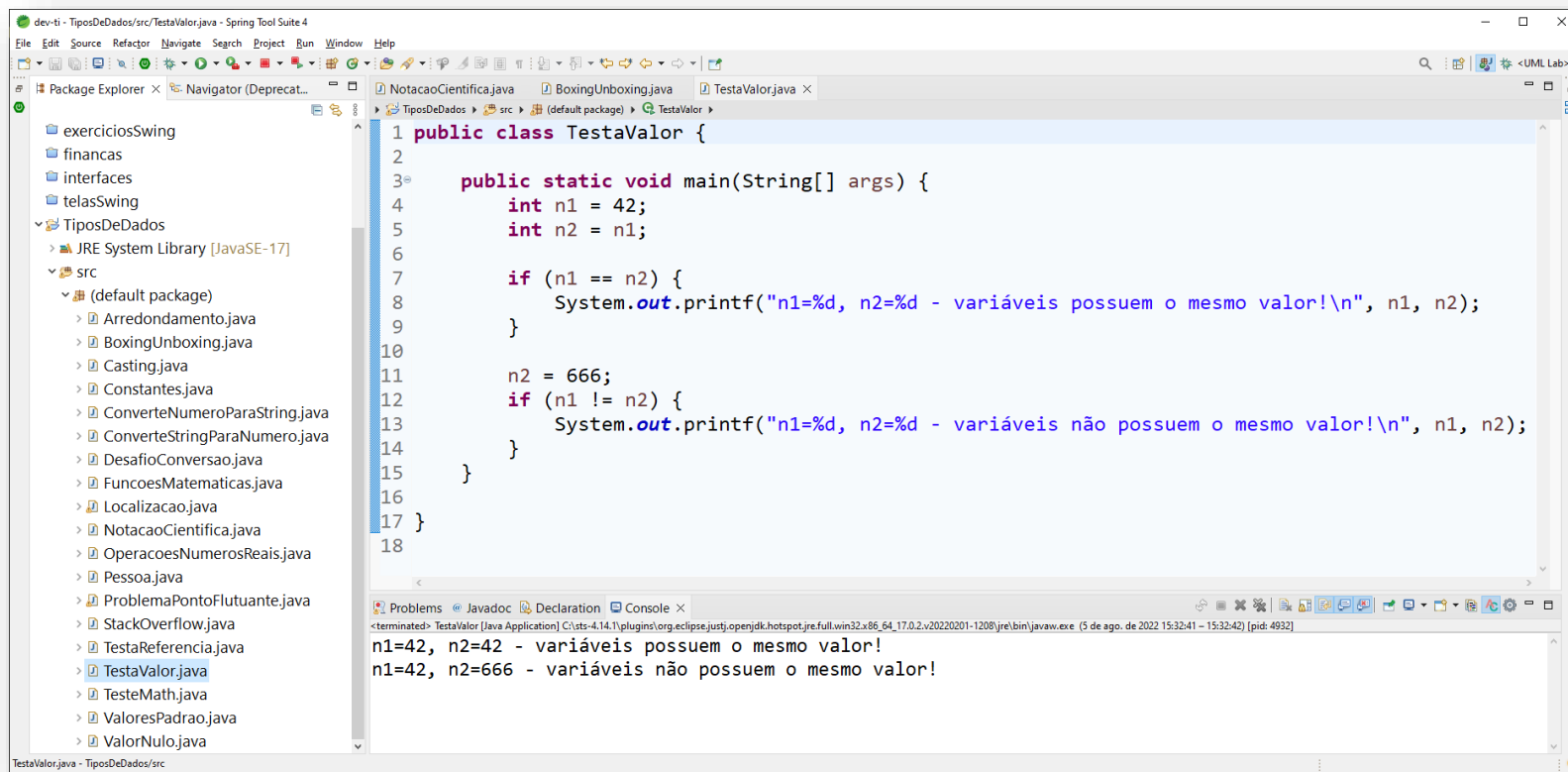
- `y = x;`

- `y` recebe uma cópia de `x`



REFERÊNCIAS VS. VALORES

Exemplo de atribuição de tipos primitivos



```
1 public class TestaValor {
2
3     public static void main(String[] args) {
4         int n1 = 42;
5         int n2 = n1;
6
7         if (n1 == n2) {
8             System.out.printf("n1=%d, n2=%d - variáveis possuem o mesmo valor!\n", n1, n2);
9         }
10
11         n2 = 666;
12         if (n1 != n2) {
13             System.out.printf("n1=%d, n2=%d - variáveis não possuem o mesmo valor!\n", n1, n2);
14         }
15     }
16 }
17
18 }
```

Problems | Javadoc | Declaration | Console

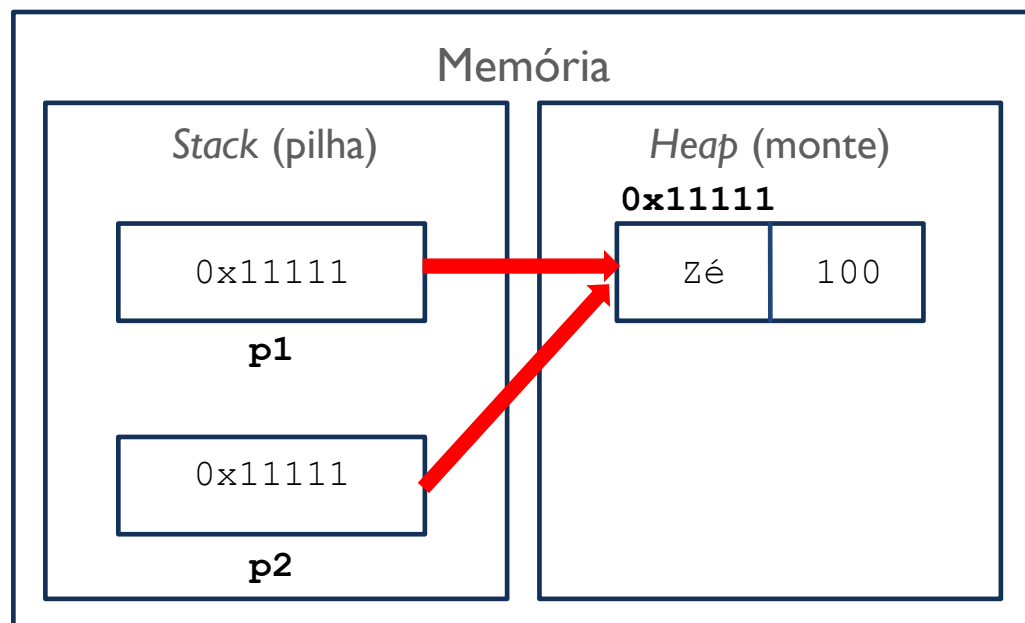
<terminated> TestaValor (Java Application) C:\sts-4.14.1\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (5 de ago. de 2022 15:32:41 - 15:32:42) [pid: 4932]

n1=42, n2=42 - variáveis possuem o mesmo valor!
n1=42, n2=666 - variáveis não possuem o mesmo valor!

REFERÊNCIAS VS. VALORES

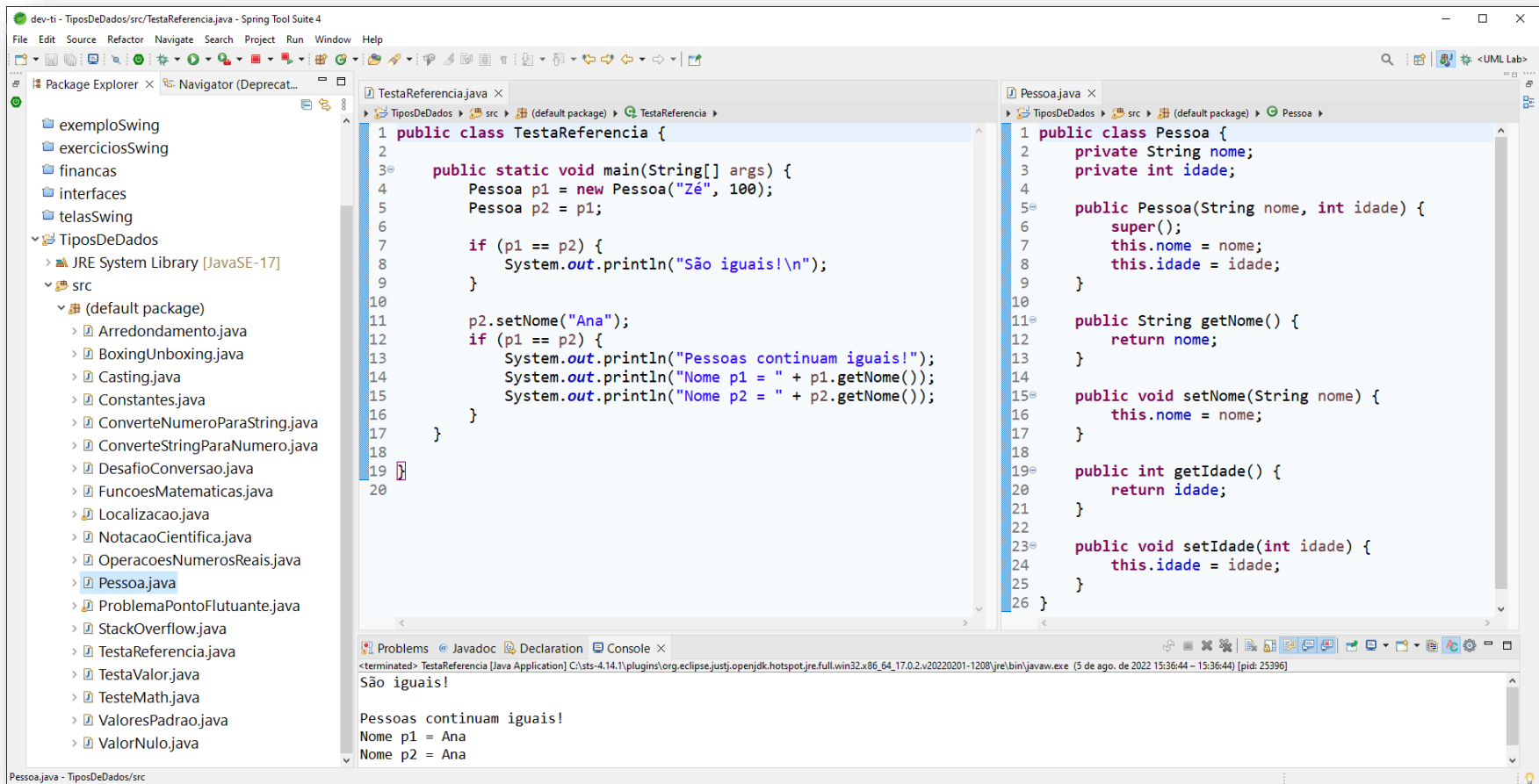
- Já objetos devem ser entendidos como variáveis ponteiros (localizadas no *stack*) que apontam para os objetos localizados no *heap* (monte)

- Pessoa p1, p2;
- p1 = new Pessoa("Zé", 100);
- p2 = p1;
- p2 passa a apontar para o mesmo objeto de p1



REFERÊNCIAS VS. VALORES

Exemplo com referências



```
dev-ti - TiposDeDados/src/TestaReferencia.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer Navigator (Deprecat...
exemploSwing
exerciciosSwing
financas
interfaces
telasSwing
TiposDeDados
  JRE System Library [JavaSE-17]
  src
    (default package)
      Arredondamento.java
      BoxingUnboxing.java
      Casting.java
      Constantes.java
      ConverteNumeroParaString.java
      ConverteStringParaNumero.java
      DesafioConversao.java
      FuncoesMatematicas.java
      Localizacao.java
      NotacaoCientifica.java
      OperacoesNumerosReais.java
      Pessoa.java
      ProblemaPontoFlutuante.java
      StackOverflow.java
      TestaReferencia.java
      TestaValor.java
      TesteMath.java
      ValoresPadrao.java
      ValorNulo.java

TestaReferencia.java
1 public class TestaReferencia {
2
3   public static void main(String[] args) {
4     Pessoa p1 = new Pessoa("Zé", 100);
5     Pessoa p2 = p1;
6
7     if (p1 == p2) {
8       System.out.println("São iguais!\n");
9     }
10
11     p2.setName("Ana");
12     if (p1 == p2) {
13       System.out.println("Pessoas continuam iguais!");
14       System.out.println("Nome p1 = " + p1.getName());
15       System.out.println("Nome p2 = " + p2.getName());
16     }
17   }
18 }
19
20

Pessoa.java
1 public class Pessoa {
2   private String nome;
3   private int idade;
4
5   public Pessoa(String nome, int idade) {
6     super();
7     this.nome = nome;
8     this.idade = idade;
9   }
10
11   public String getName() {
12     return nome;
13   }
14
15   public void setName(String nome) {
16     this.nome = nome;
17   }
18
19   public int getIdade() {
20     return idade;
21   }
22
23   public void setIdade(int idade) {
24     this.idade = idade;
25   }
26 }

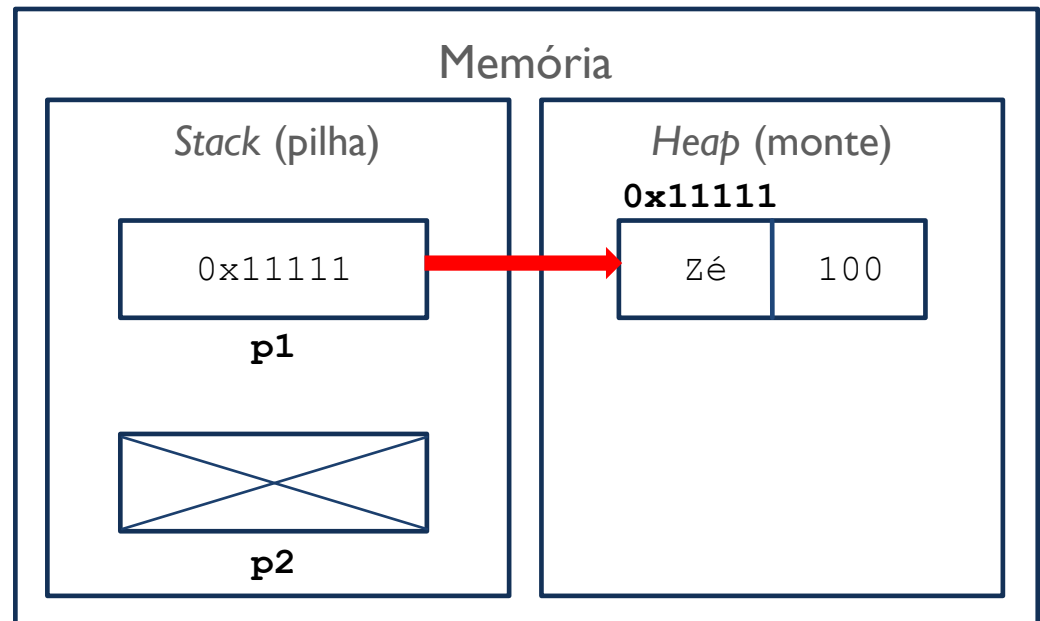
Problems Javadoc Declaration Console
<terminated> TestaReferencia [Java Application] C:\sts-4.14.1\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.2.v20220201-1208\jre\bin\javaw.exe (5 de ago. de 2022 15:36:44 - 15:36:44) [pid: 25396]
São iguais!

Pessoas continuam iguais!
Nome p1 = Ana
Nome p2 = Ana
```

REFERÊNCIAS VS. VALORES

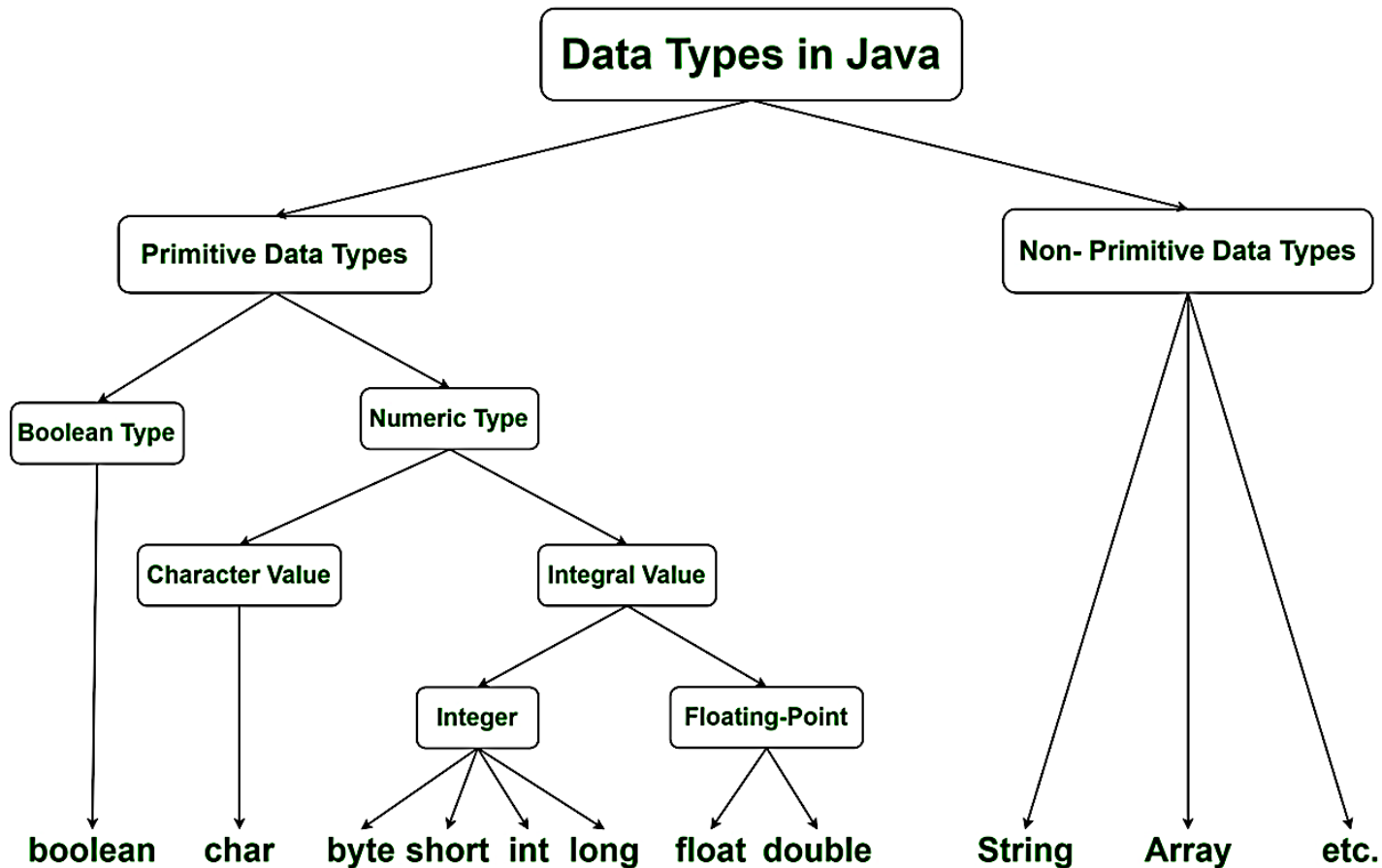
- Referências aceitam valores nulos, indicando que não estão apontando para nenhum objeto

- `Pessoa p1, p2;`
- `p1 = new Pessoa("Zé", 100);`
- `p2 = null;`



OBJETOS VS. TIPOS PRIMITIVOS

- Tipos primitivos são mais 'leves' e rápidos
- Objetos são mais 'ricos' em recursos, possuindo métodos e atributos



WRAPPER CLASSES

- *Wrappers* (embrulho, invólucro, envelope) são versões orientadas a objetos dos tipos primitivos

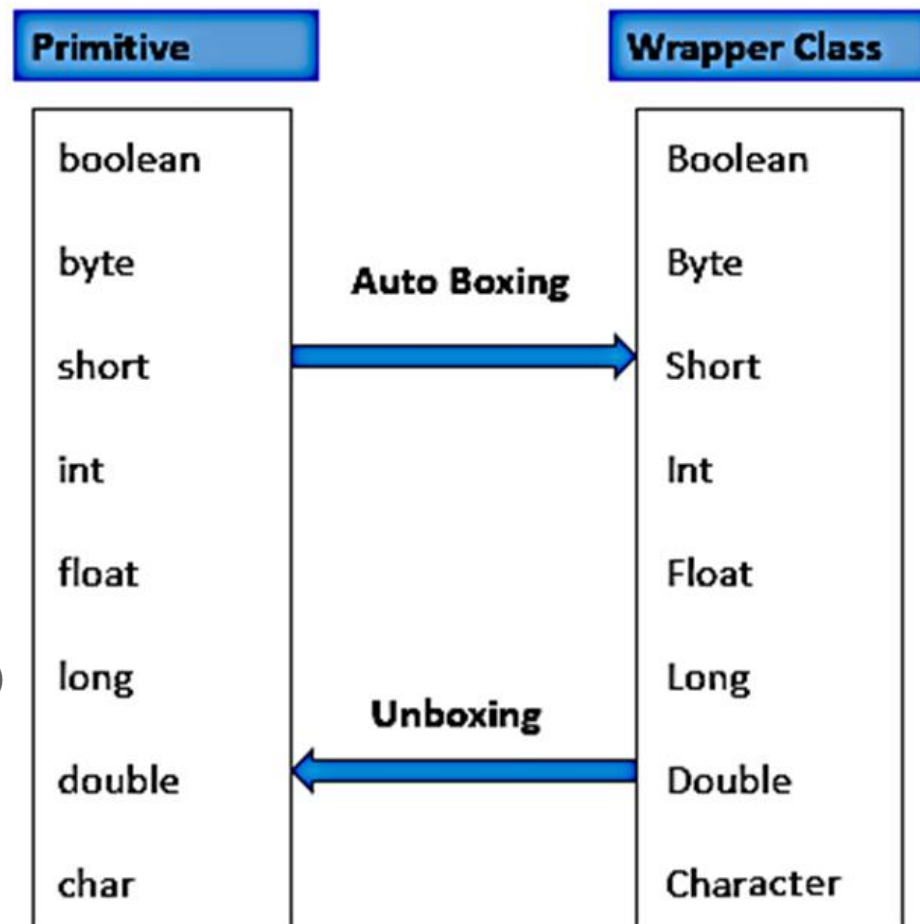
- Character
- Integer
- Float
- Double
- Etc

- **Vantagens**

- São objetos (atributos e métodos)
- Podem receber o valor nulo (`null`)
- Trabalhar com coleções

- **Desvantagem**

- Performance



WRAPPER CLASSES

■ Exemplo

```
modulo2 - Wrappers.java

1 public class Wrapper {
2     public static void main(String[] args) {
3         Boolean estado = null;
4         System.out.println(estado);
5
6         estado = Boolean.valueOf("true");
7         System.out.println(estado);
8
9         // Converte para String e então para maiúsculas
10        System.out.println(estado.toString().toUpperCase());
11        System.out.println();
12
13        //-----
14        Integer a = 1000;
15        Integer b = 1000;
16
17        // Sempre use equals(), nunca use == com wrappers
18        System.out.println(a.equals(b)); // true
19        System.out.println(a == b);     // false
20        System.out.println();
21
22        a = 100;
23        b = 100;
24
25        System.out.println(a.equals(b)); // true
26        System.out.println(a == b);     // true
27    }
28 }
```

WRAPPER CLASSES

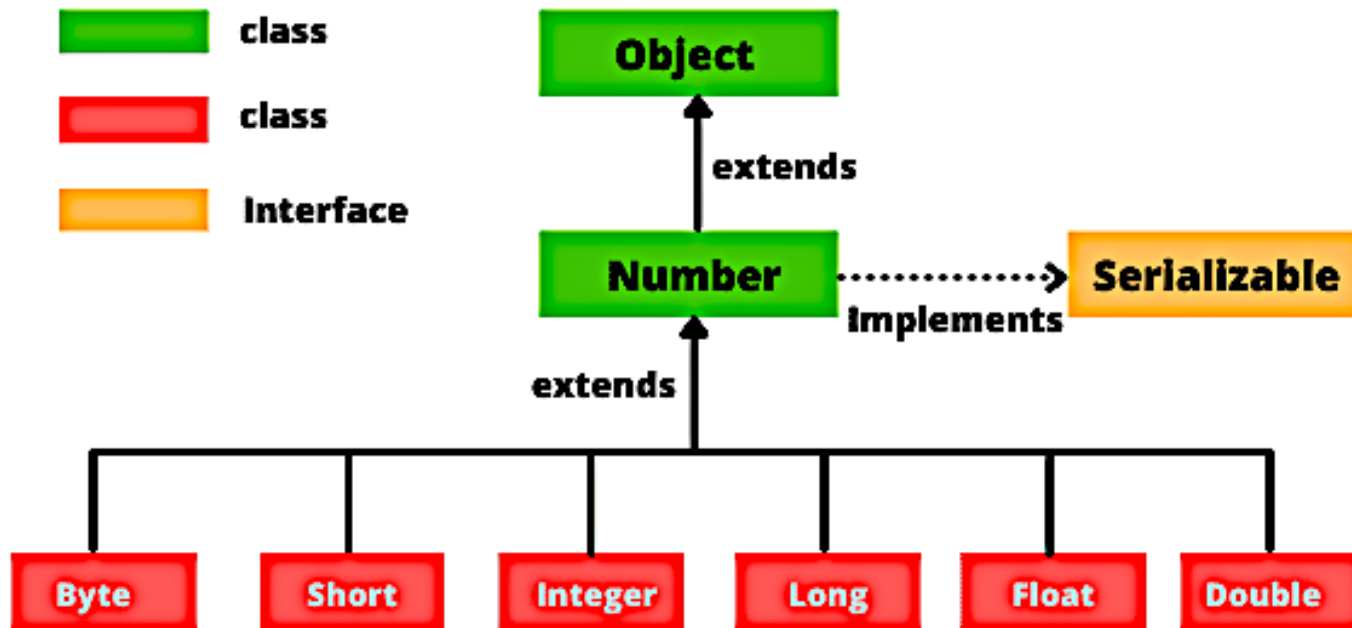
- Em Java, *arrays* podem ser de tipos primitivos ou de referências, coleções só trabalham com referências, mas o Java aplica *autoboxing* automaticamente ao se armazenar tipos primitivos em listas

```
modulo2 - Wrappers.java

1 import java.util.Arrays;
2 import java.util.List;
3
4 public class Wrappers {
5
6     public static void main(String[] args) {
7         int n1 = 1, n2 = 2, n3 = 3;
8         Integer iN1 = 1, iN2 = 2, iN3 = 3;
9
10        int[] numeros = {n1, n2, n3};
11        Integer[] iNumeros = {iN1, iN2, iN3};
12
13        System.out.println("Elementos de 'numeros': " + Arrays.toString(numeros));
14        System.out.println("Elementos de 'iNumeros': " + Arrays.toString(iNumeros));
15
16        System.out.println("Tipo do 1o. elemento 'numeros': int (não há métodos disponíveis)");
17        System.out.println("Tipo do 1o. elemento 'iNumeros': " + iNumeros[0].getClass());
18        //-----
19        List<Integer> lNumeros = Arrays.asList(n1, n2, n3);
20
21        System.out.println("-----");
22        System.out.println("Tipo do primeiro elemento: " + lNumeros.get(0).getClass());
23
24        for (Integer numero : lNumeros) {
25            System.out.print(numero + " ");
26        }
27
28        System.out.println();
29        lNumeros.forEach(System.out::println); // Outra forma, mais compacta, de imprimir
30    }
31
32 }
```

WRAPPER CLASSES

- A classe `Number` é uma superclasse mais genérica que pode representar qualquer *wrapper* numérico



WRAPPER CLASSES

- Com isso é possível criar uma lista de número que armazene números inteiros, longos e flutuantes de precisão simples e dupla

```
modulo2 - ConverteStringParaNumero.java

1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class ClasseNumber {
5
6      public static void main(String[] args) {
7          List<Number> numeros = new ArrayList<>();
8
9          numeros.add(10);
10         numeros.add(123_456_789L);
11         numeros.add(123f);
12         numeros.add(123_456.78);
13
14         for (Number numero : numeros) {
15             System.out.println("Número " + numero + " é do tipo " + numero.getClass());
16         }
17     }
18
19 }
```

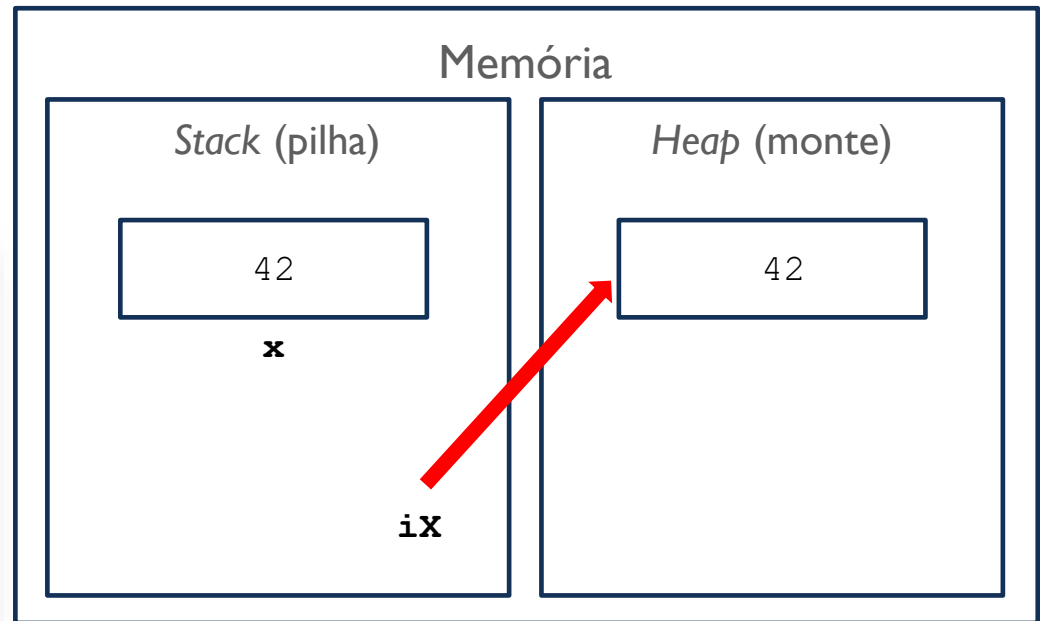
Apoiadores:

BOXING E UNBOXING

- **Boxing** (encaixotamento) é a conversão de um tipo de dados primitivo em seu objeto equivalente (classes *wrapper*)
- **Autoboxing**: O próprio Java já converte automaticamente o tipo primitivo em *wrapper* sempre que achar que isso é necessário

- `int x = 42;`
- `Integer iX = x;`

```
modulo2 - Boxing.java
1 public class Boxing {
2
3     public static void main(String[] args) {
4         int x = 42;
5         Integer iX = x;
6
7         System.out.println("Tipo primitivo.: " + x);
8         System.out.println("Objeto (wrapper): " + iX);
9     }
10
11 }
```

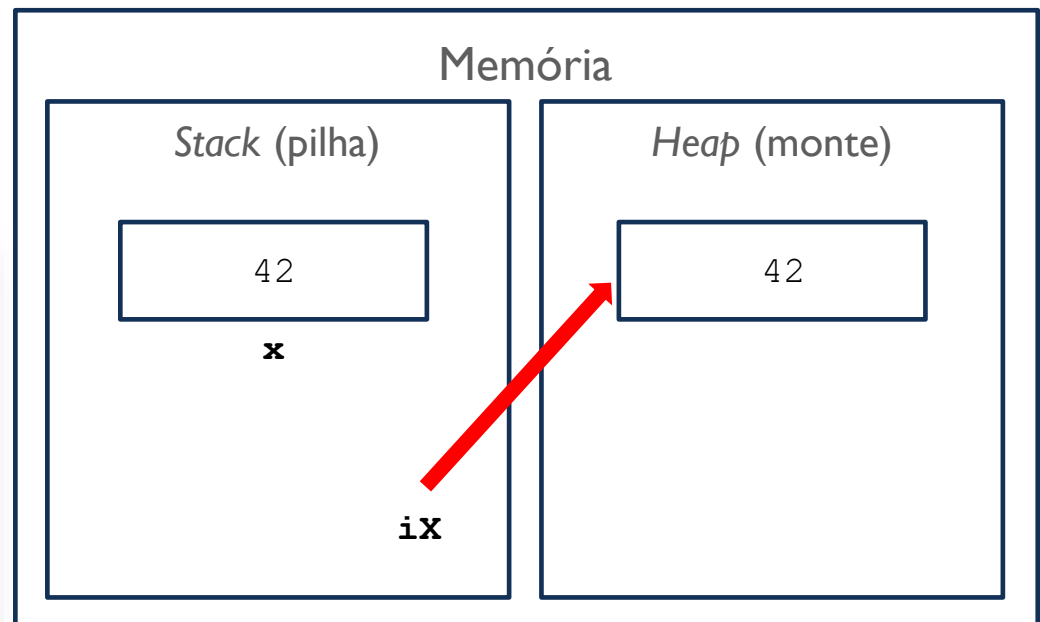


BOXING E UNBOXING

- *Unboxing* (desencaixotamento) é o processo inverso

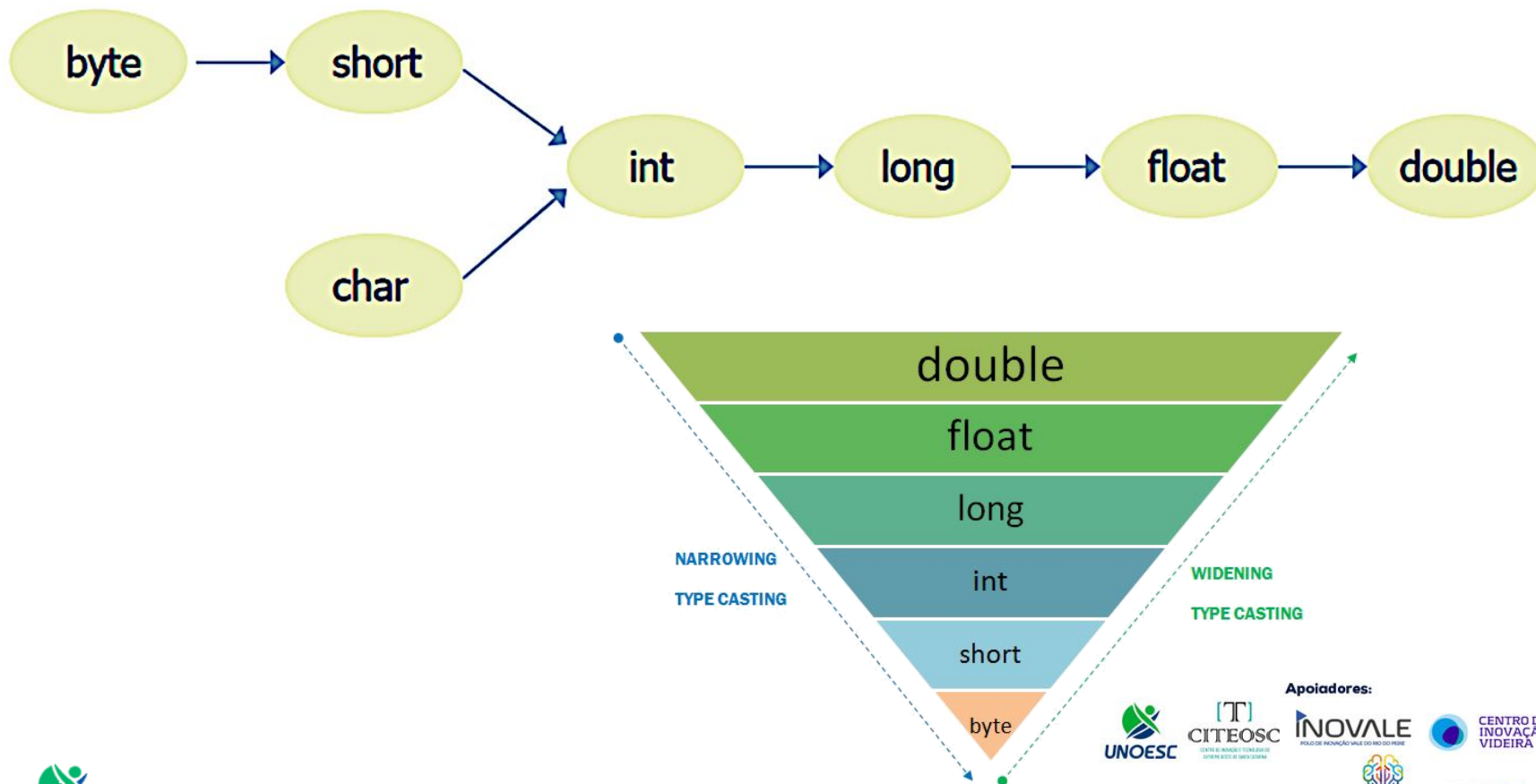
- `Integer iX = 42;`
- `int x = iX;`

```
modulo2 - Unboxing.java
1 public class Unboxing {
2
3     public static void main(String[] args) {
4         Integer iX = 42;
5         int x = iX;
6
7         System.out.println("Objeto (wrapper): " + iX);
8         System.out.println("Tipo primitivo..: " + x);
9     }
10
11 }
```



CONVERSÕES (CASTING)

- A ‘promoção’ entre tipos primitivos é sempre permitida quando ocorre de um tipo “menor” para um tipo “maior”, e, nestes casos, é feita de maneira implícita



CONVERSÕES (CASTING)

- A operação denominada *casting* é a conversão *explícita* de um tipo em outro
- Casting é usado quando o compilador não consegue determinar que o resultado de uma expressão deve ser de outro tipo ou então forçar determinadas conversões que não são permitidas por padrão pelo compilador porque pode haver perda de dados

CONVERSÕES (CASTING)

■ Exemplo

```
modulo2 - Wrapper.java

1  public class Casting {
2
3      public static void main(String[] args) {
4          // Casting em literais
5          System.out.println("Divisão sem casting: " + 10/3);
6          System.out.println("Divisão sem casting: " + (float) 10/3);
7          System.out.println();
8
9          // Casting em variáveis
10         int n1 = 10, n2 = 3;
11         double resultado1 = n1 / n2;
12         double resultado2 = (double) n1 / n2;
13         System.out.println("Sem casting: " + resultado1);
14         System.out.println("Com casting: " + resultado2);
15
16         // Casting de tipos 'maiores' para 'menores'
17         double a = 10.5;
18         // int b = a; // Erro
19         int b = (int) a;
20         System.out.printf("\nDe double (%.2f) para int (%d)", a, b);
21     }
22
23 }
```

Apoiadores:



CONVERSÕES (*CASTING*)

- A conversão de números para *strings* pode ser feita de várias formas, tais como
 - Método `toString()` do objeto ou das classes *wrapper*
 - Método `valueOf()` da classe `String`
- A conversão de *strings* para tipos primitivos pode ser feita por meio dos métodos `parseXXX()` (`parseInt()`, `parseFloat()`, `parseDouble()`, etc) presentes nas classes *wrapper*
- O método `valueOf()` das classes *wrapper* recebe uma *string* e devolve uma referência (objeto)

CONVERSÕES (CASTING)

- Exemplo de conversões de números para *strings*

```
modulo2 - ConverteNumeroParaString.java

1  public class ConverteNumeroParaString {
2
3      public static void main(String[] args) {
4          Integer num1 = 123;
5          System.out.println(num1.toString());
6
7          int num2 = 456;
8          System.out.println(String.valueOf(num2));
9
10         double num3 = 789.01;
11         System.out.println(Double.toString(num3));
12     }
13
14 }
```

Apoiadores:

CONVERSÕES (CASTING)

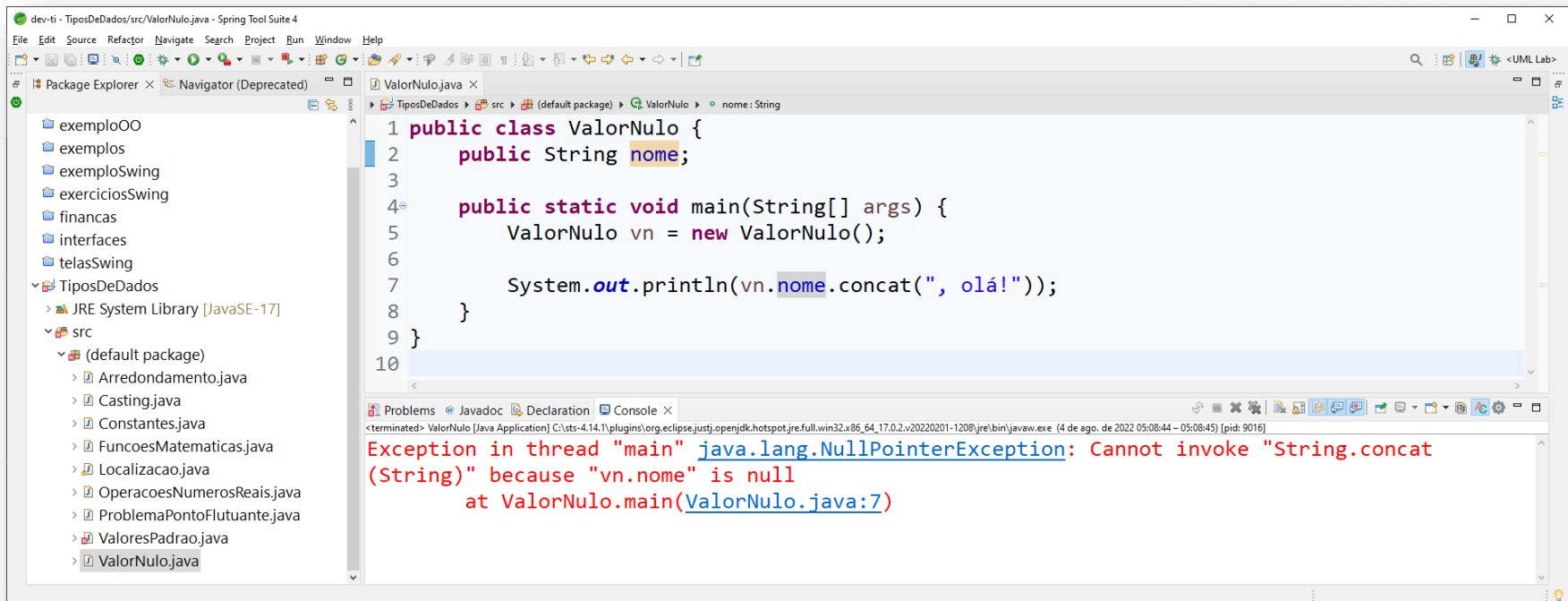
- Exemplo de conversões de *strings* para números

```
modulo2 - ConverteStringParaNumero.java

1  public class ConverteStringParaNumero {
2
3      public static void main(String[] args) {
4          String str1 = "123";
5          int num1 = Integer.parseInt(str1);
6          System.out.println(num1);
7
8          String str2 = "456.78f";
9          float num2 = Float.parseFloat(str2);
10         System.out.println(num2);
11
12         String str3 = "789.01";
13         double num3 = Double.parseDouble(str3);
14         System.out.println(num3);
15
16         Integer numero = Integer.valueOf("42");
17         System.out.println(numero);
18     }
19
20 }
```

TIPO NULL

- O valor `null` (nulo) indica que um objeto não aponta para lugar nenhum, ou seja, indica que uma referência não está apontando para nenhum objeto concreto
- A tentativa de invocar um método ou acessar um atributo a partir do valor `null` irá gerar o famoso (e muito comum) erro (exceção) chamado `NullPointerException`



The screenshot shows an IDE window titled "dev-ti - TiposDeDados/src/ValorNulo.java - Spring Tool Suite 4". The editor displays the following Java code:

```
1 public class ValorNulo {  
2     public String nome;  
3  
4     public static void main(String[] args) {  
5         ValorNulo vn = new ValorNulo();  
6  
7         System.out.println(vn.nome.concat(", olá!"));  
8     }  
9 }  
10
```

The variable `nome` is highlighted in yellow. The console at the bottom shows the following error message:

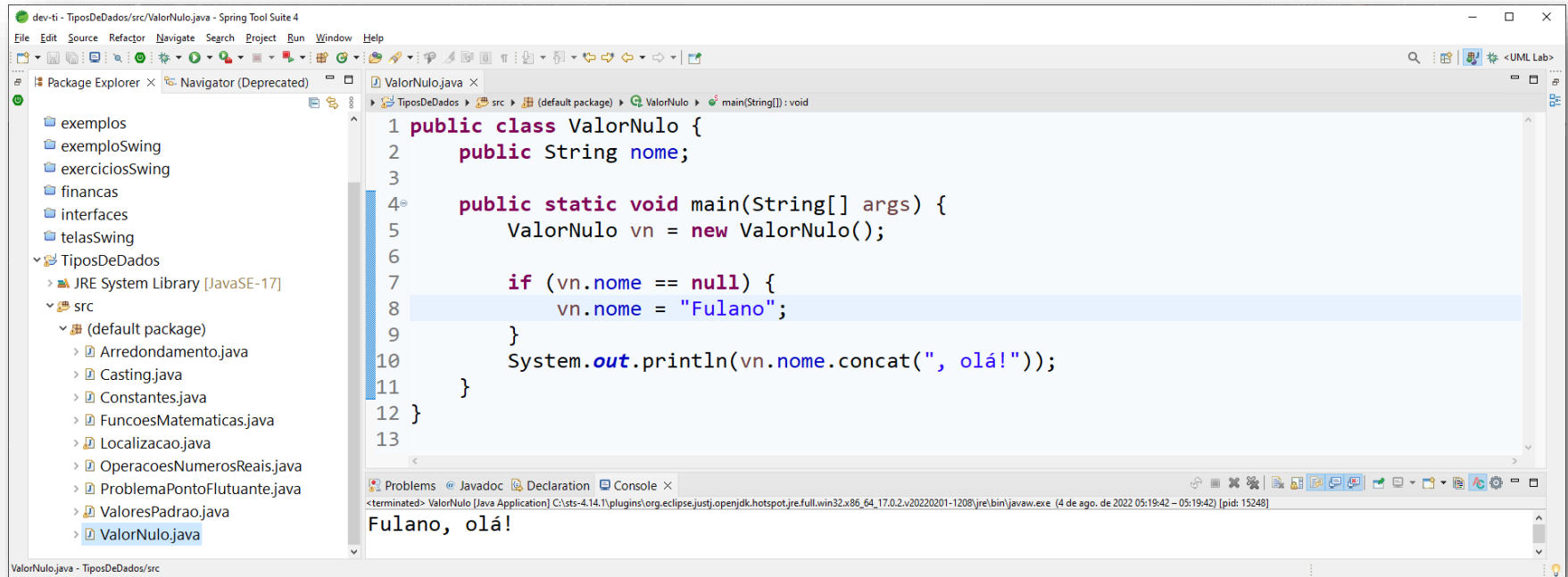
```
<terminated> ValorNulo [Java Application] C:\sts-4.14.1\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.2.v20220201-1208\jre\bin\javaw.exe (4 de ago. de 2022 05:08:44 - 05:08:45) [pid: 9016]  
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.concat (String)" because "vn.nome" is null  
    at ValorNulo.main(ValorNulo.java:7)
```

The left sidebar shows the Package Explorer with the following structure:

- exemploOO
- exemplos
- exemploSwing
- exerciciosSwing
- financas
- interfaces
- telasSwing
- TiposDeDados
 - JRE System Library [JavaSE-17]
 - src
 - (default package)
 - Arredondamento.java
 - Casting.java
 - Constantes.java
 - FuncoesMatematicas.java
 - Localizacao.java
 - OperacoesNumerosReais.java
 - ProblemaPontoFlutuante.java
 - ValoresPadrao.java
 - ValorNulo.java

TIPO NULL

■ O erro pode ser evitado através de uma simples verificação com a instrução `if()`



The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows a project named 'TiposDeDados' with a source folder 'src' containing several Java files, including 'ValorNulo.java' which is selected.
- Editor:** Displays the code for 'ValorNulo.java'. The code is as follows:

```
1 public class ValorNulo {
2     public String nome;
3
4     public static void main(String[] args) {
5         ValorNulo vn = new ValorNulo();
6
7         if (vn.nome == null) {
8             vn.nome = "Fulano";
9         }
10        System.out.println(vn.nome.concat(", olá!"));
11    }
12 }
13
```
- Console:** Shows the output of the program: 'Fulano, olá!'. The message is preceded by a 'terminated' status and the full path to the Java application.

DESAFIO

- A exceção `InputMismatchException` é gerada quando um número em ponto flutuante foi digitado incorretamente
- O Java utiliza um conceito chamado `Locale`, que é uma informação sobre qual país você se encontra, e é este `Locale` que dita as convenções de números, datas, horas, etc. que serão utilizados na máquina
- Se o computador está com o `Locale` setado para “en-US”, isso implica que utilizará as convenções de formatação dos Estados Unidos, ou seja, casas inteiras e casas decimais de um número em ponto flutuante separadas por ‘.’
- Caso o `Locale` esteja definido para “pt-BR”, um número em ponto flutuante utilizando ‘.’ como separador de casas decimais não é compreendido pelo Java
- O mesmo acontece ao tentar digitar um número com ‘,’ em uma máquina cujo `Locale` está setado para a língua inglesa
- Programe uma classe que, através de um `Scanner`, receba 3 *strings* correspondentes aos 3 últimos salários de um empregado e calcule a média deles
- O usuário poderá digitar o salário com ponto ou vírgula decimal
- Para resolver este problema, estude o método `replace` da classe `String` 😊

Apoiadores:

