



# TIPOS DE DADOS E CONVERSÕES

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

[herculano.debiasi@unoesc.edu.br](mailto:herculano.debiasi@unoesc.edu.br)



# TÓPICOS

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

# TIPOS PRIMITIVOS

## ■ Tabela dos tipos primitivos em Java

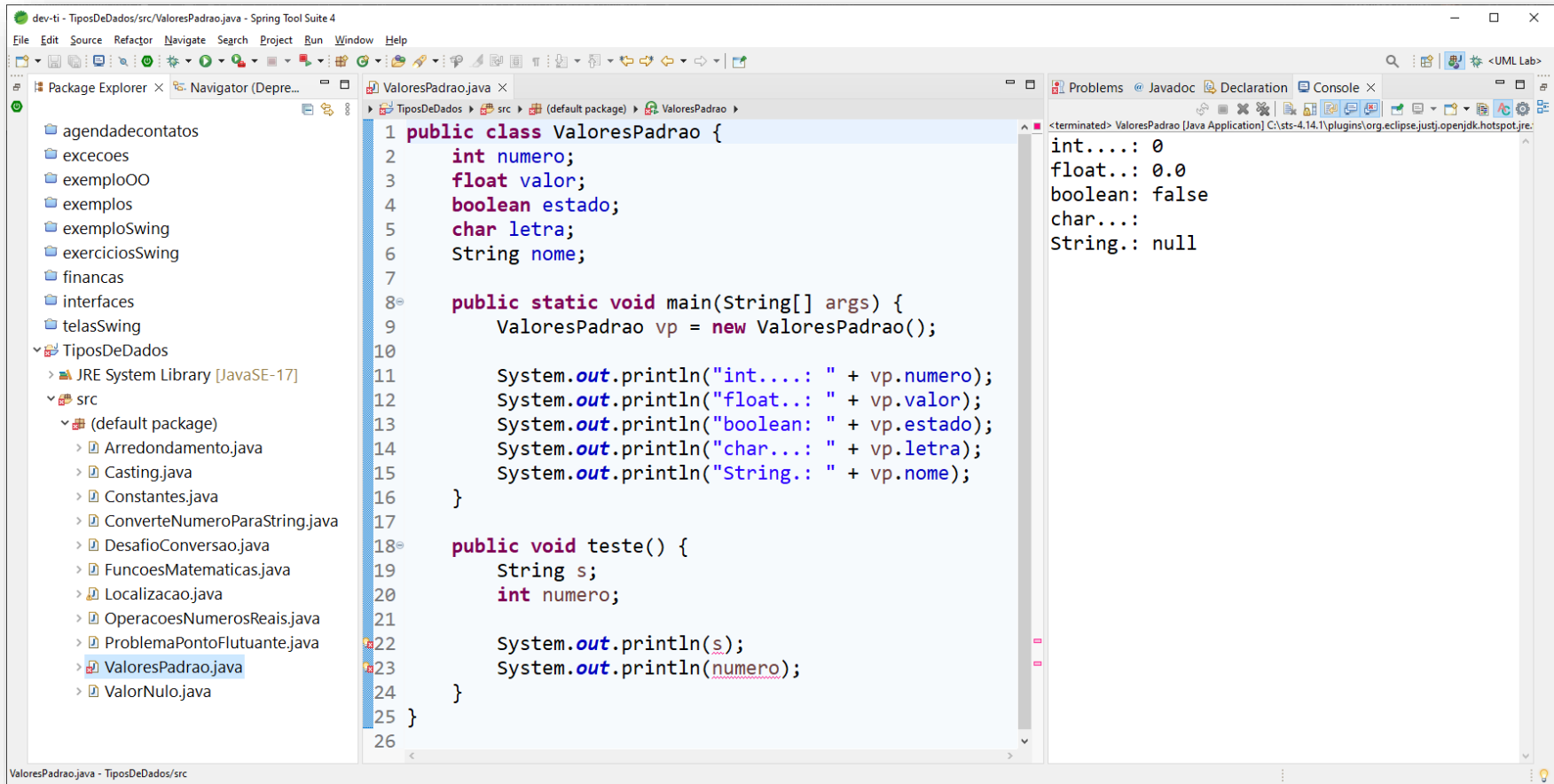
Descrição	Tipo	Tamanho	Valores	Valor padrão
tipos numéricos inteiros	<b>byte</b>	8 bits	-128 a 127	0
	<b>short</b>	16 bits	-32768 a 32767	0
	<b>int</b>	32 bits	-2147483648 a 2147483647	0
	<b>long</b>	64 bits	-9223372036854770000 a 9223372036854770000	0L
tipos numéricos com ponto flutuante	<b>float</b>	32 bits	-1,4024E-37 a 3,4028E+38	0.0f
	<b>double</b>	64 bits	-4,94E-307 a 1,79E+308	0.0
um caractere Unicode	<b>char</b>	16 bits	'\u0000' a '\uFFFF'	'\u0000'
valor verdade	<b>boolean</b>	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 displays the source code of "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 "Console" tab with the output of the program:

```
<terminated> ValoresPadrao [Java Application] C:\sts-4.14.1\plugins\org.eclipse.justj.openjdk.hotspot.jre:
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-FLUTUANTE'

- O recurso de 'Localização' procura adaptar o formato dos números reais (ponto ou vírgula flutuante) ao formatos regionais

```
modulo2 - Localizacao.java

1 import java.util.Locale;
2 import java.util.Scanner;
3
4 public class Localizacao {
5     public static void main(String[] args) {
6         // O Locale é preciso estar antes do Scanner
7         Locale.setDefault(Locale.US);
8         Scanner sc = new Scanner(System.in);
9
10        System.out.print("Digite um número real (com ponto decimal): ");
11        double valor = sc.nextDouble();
12
13        System.out.printf("%.2f\n", valor);
14        //-----
15        Locale.setDefault(Locale.GERMANY);
16        sc = new Scanner(System.in);
17
18        System.out.print("Digite um número real (com vírgula decimal): ");
19        valor = sc.nextDouble();
20
21        System.out.printf("%.2f\n", valor);
22    }
23 }
```

# TIPOS 'PONTO-FLUTUANTE'

## ■ 'Problema' com tipos ponto-flutuante

```
modulo2 - ProblemaPontoFlutuante.java

1 public class ProblemaPontoFlutuante {
2     public static void main(String[] args) {
3         // Problema com números em ponto flutuante
4         double a = 0.1;
5         double b = 0.2;
6         double c = a + b;
7
8         if (c == 0.3) {
9             System.out.println("Acertou a conta!");
10        } else {
11            System.out.println("Epa, o que está acontecendo?!?!");
12            System.out.println(c);
13        }
14    }
15 }
```

# 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

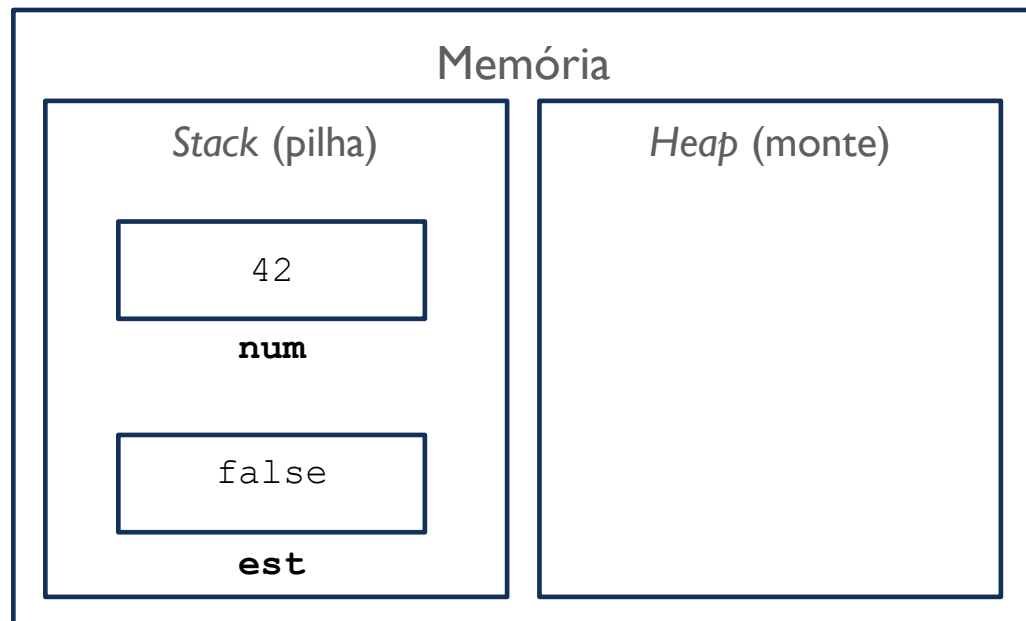
## ■ Exemplo

```
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

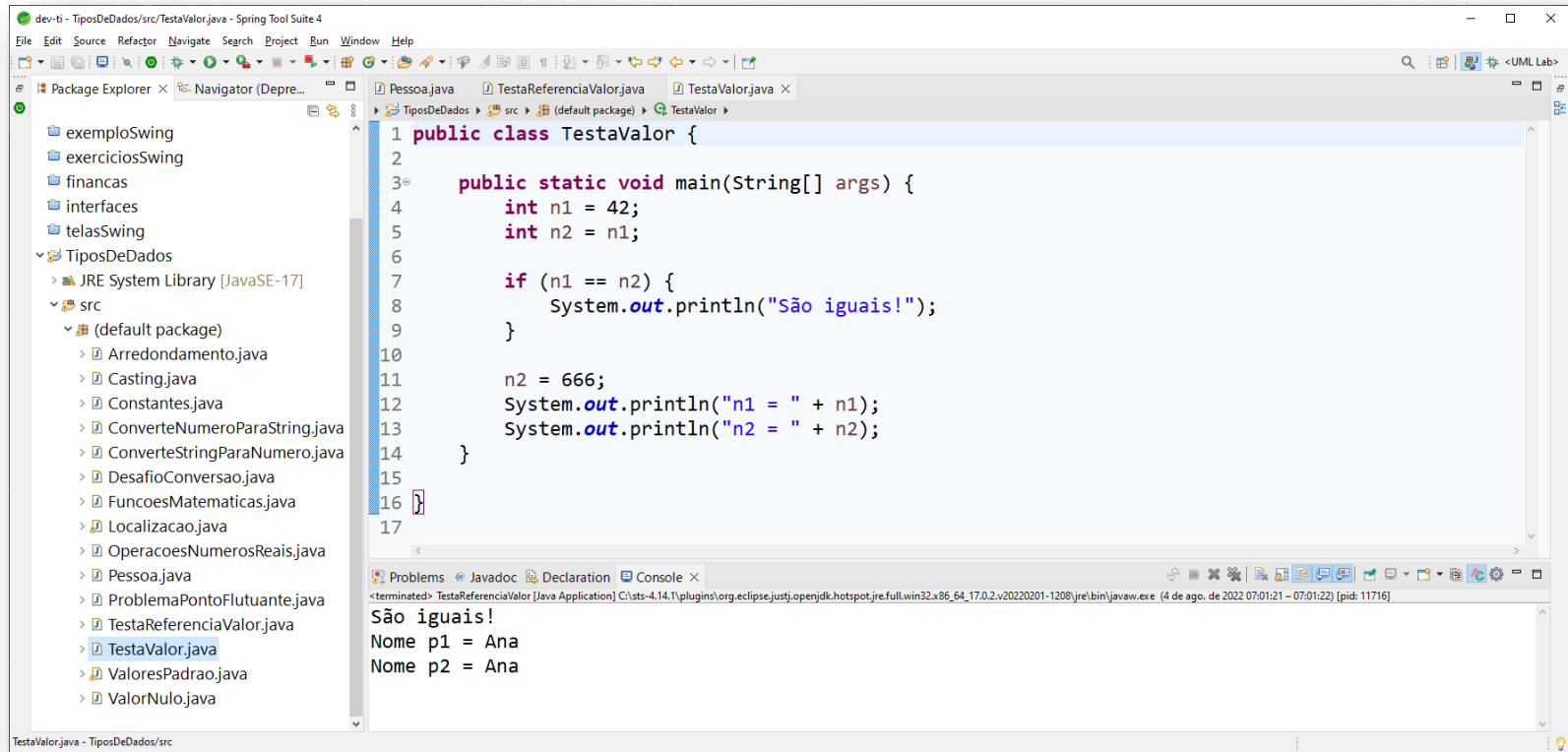
- Variáveis dos tipos primitivos podem ser entendidas como caixas dentro na memória do computador, em uma área chamada de *stack* (pilha)

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



# REFERÊNCIAS VS. VALORES

## Exemplo de atribuição de tipos primitivos



The screenshot shows an IDE window titled "dev-ti - TiposDeDados/src/TestaValor.java - Spring Tool Suite 4". The code in the editor is as follows:

```
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.println("São iguais!");
9         }
10
11         n2 = 666;
12         System.out.println("n1 = " + n1);
13         System.out.println("n2 = " + n2);
14     }
15 }
16
17
```

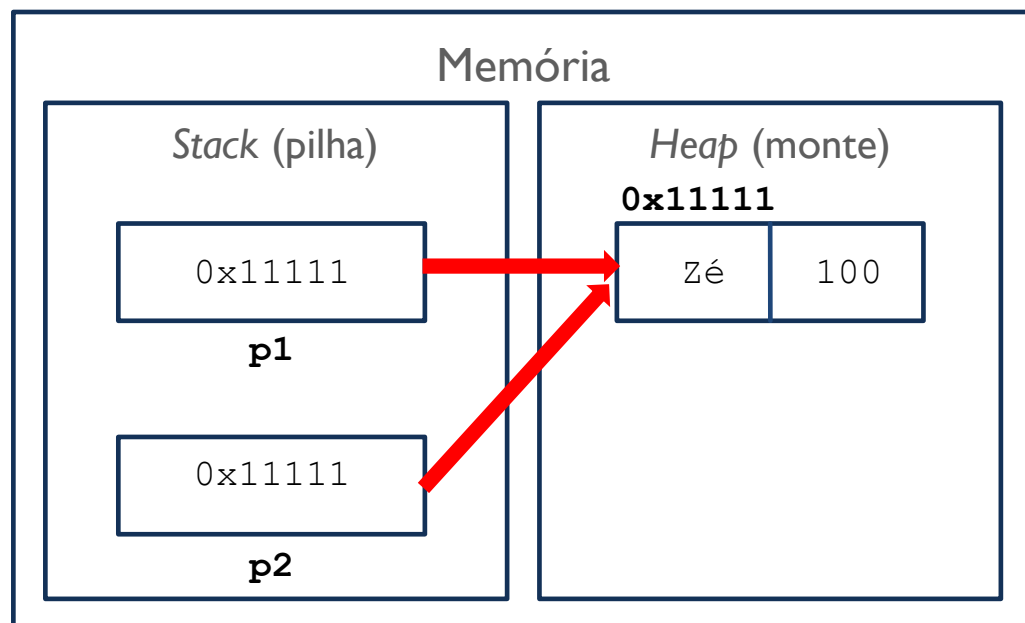
The console output at the bottom shows the following messages:

```
<terminated> TestaReferenciaValor [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 07:01:21 - 07:01:22) [pid: 11716]
São iguais!
Nome p1 = Ana
Nome p2 = Ana
```

# 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



The screenshot shows an IDE with two Java files: `Pessoa.java` and `TestaReferencia.java`. The `Pessoa` class has attributes `nome` and `idade`, and methods `getNome`, `setNome`, `getIdade`, and `setIdade`. The `TestaReferencia` class has a `main` method that creates two `Pessoa` objects, `p1` and `p2`, and checks if they are equal using `p1 == p2`. The console output shows that `p1` and `p2` are equal, indicating that the objects are the same reference.

```
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 getNome() {
12        return nome;
13    }
14
15    public void setNome(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 }
27
```

```
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!");
9         }
10
11         p2.setNome("Ana");
12         System.out.println("Nome p1 = " + p1.getNome());
13         System.out.println("Nome p2 = " + p2.getNome());
14     }
15 }
16
17
```

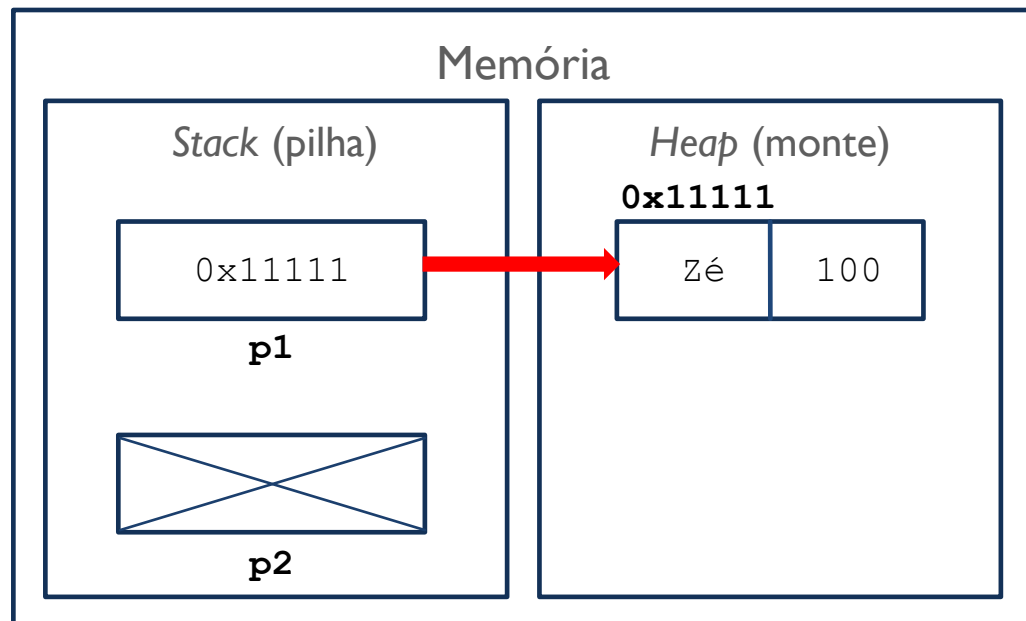
Console Output:

```
<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 (4 de ago. de 2022 07:14:23 - 07:14:23) [pid: 5856]
São iguais!
Nome p1 = Ana
Nome p2 = Ana
São iguais!
```

# REFERÊNCIAS VS. VALORES

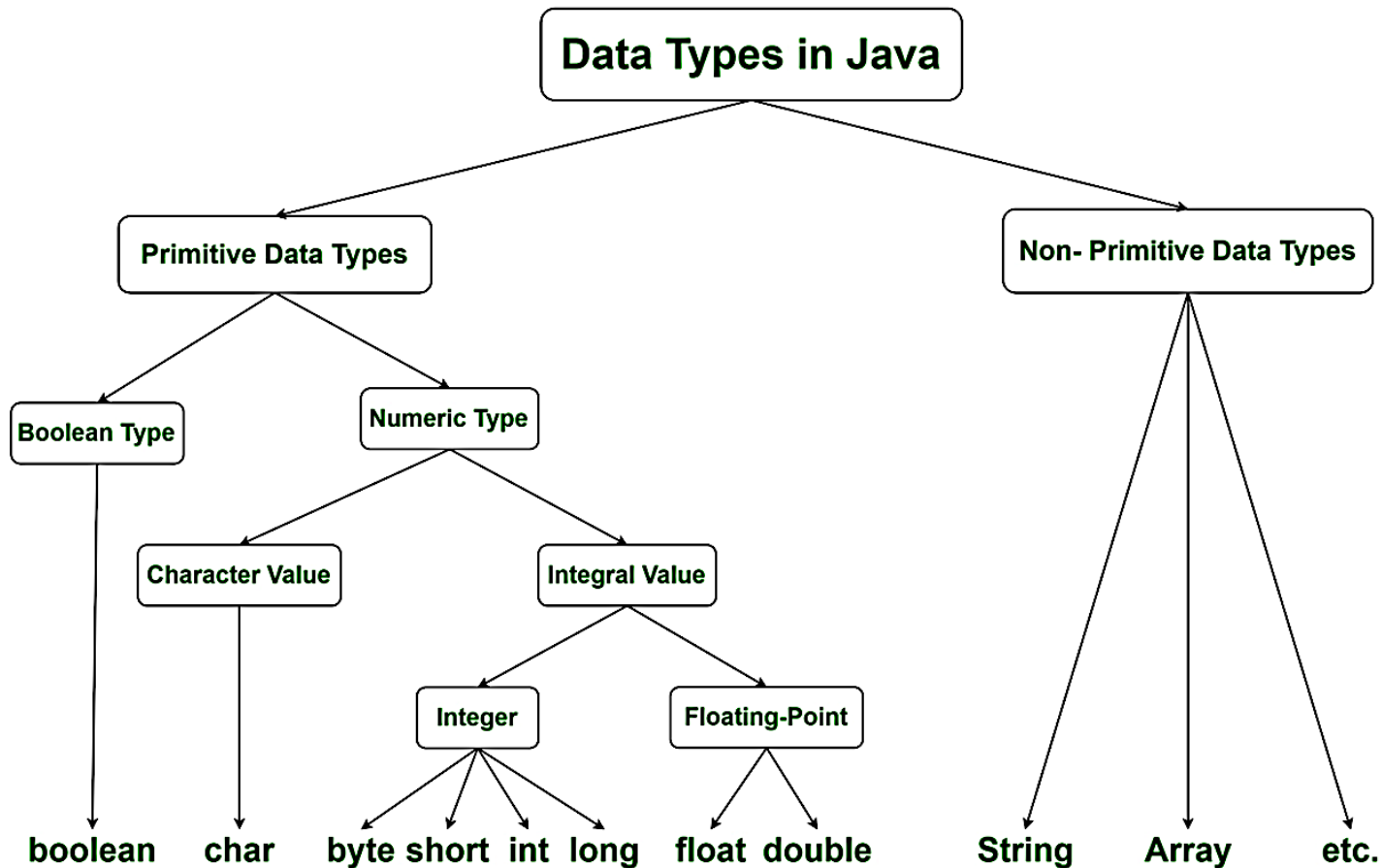
- Referências aceitam valores nulos, indicando que não estão apontando para nenhum objeto

- `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

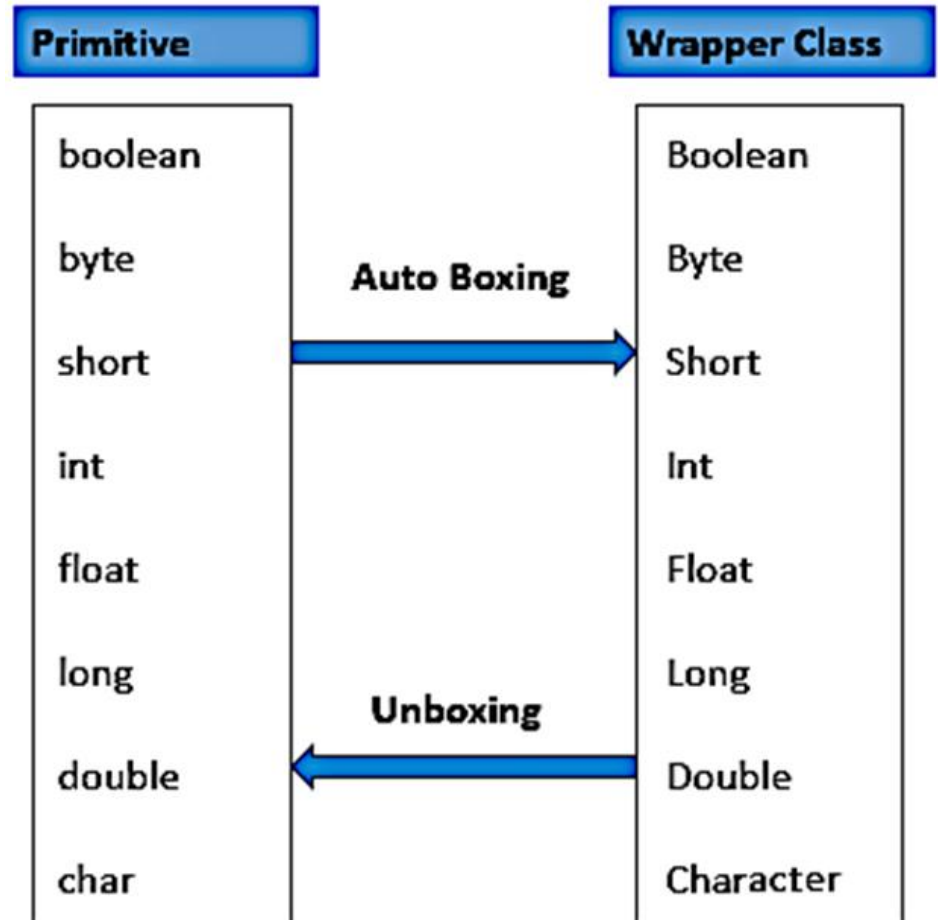


# BOXING, UNBOXING E WRAPPER CLASSES

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

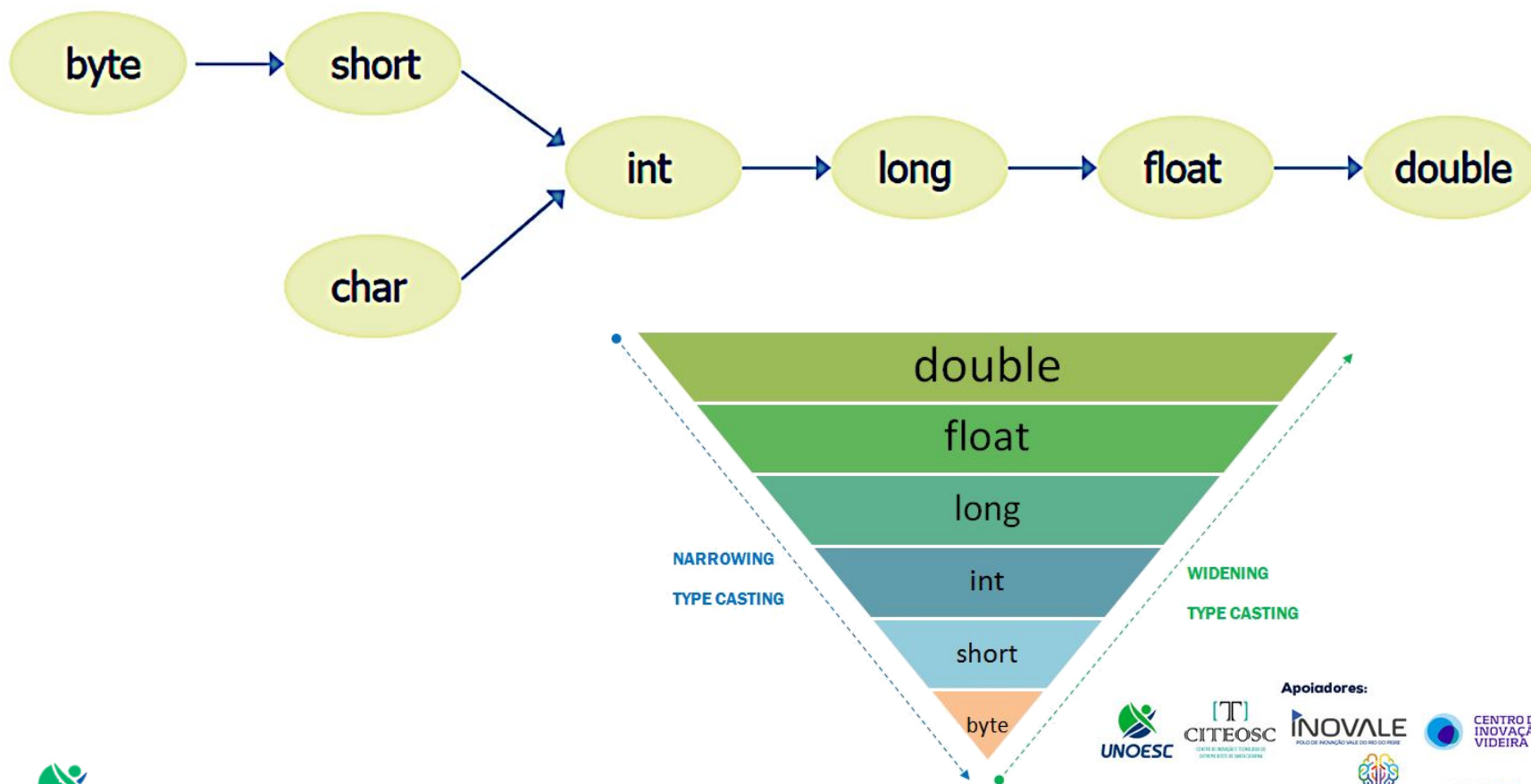
- Character
- Integer
- Float
- Double
- etc

- *Boxing* é a conversão de um tipo de dados primitivo em seu objeto equivalente (classes *wrapper*)
- *Unboxing* é o processo inverso



# CONVERSÕES (CASTING)

- A conversão 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
- 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 números pode ser feita por meio dos métodos `parseXXX()` (`parseInt()`, `parseFloat()`, `parseDouble()`, etc) presentes nas classes *wrapper*

# 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

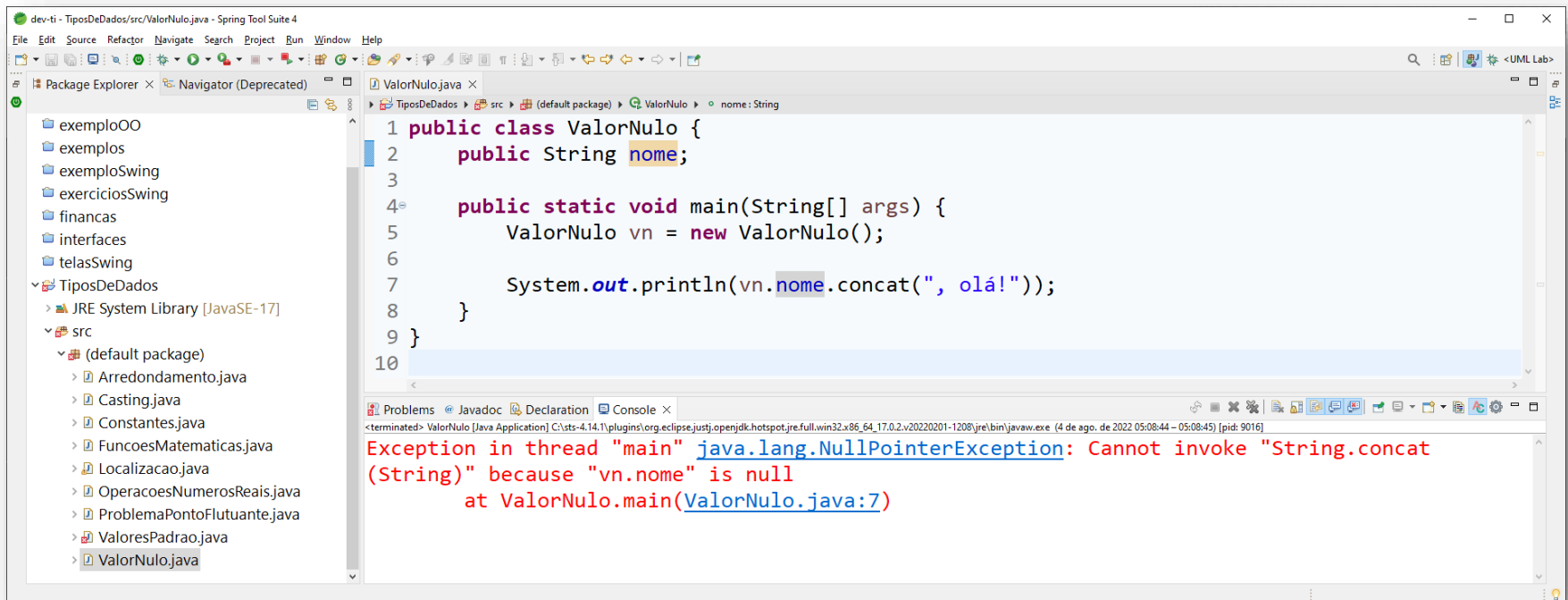
```
modulo2 - ConverteStringParaNumero.java

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
17 }
```



# 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 a Spring Tool Suite 4 IDE with a project named 'TiposDeDados'. The 'src' folder contains a file 'ValorNulo.java'. The code in the file is as follows:

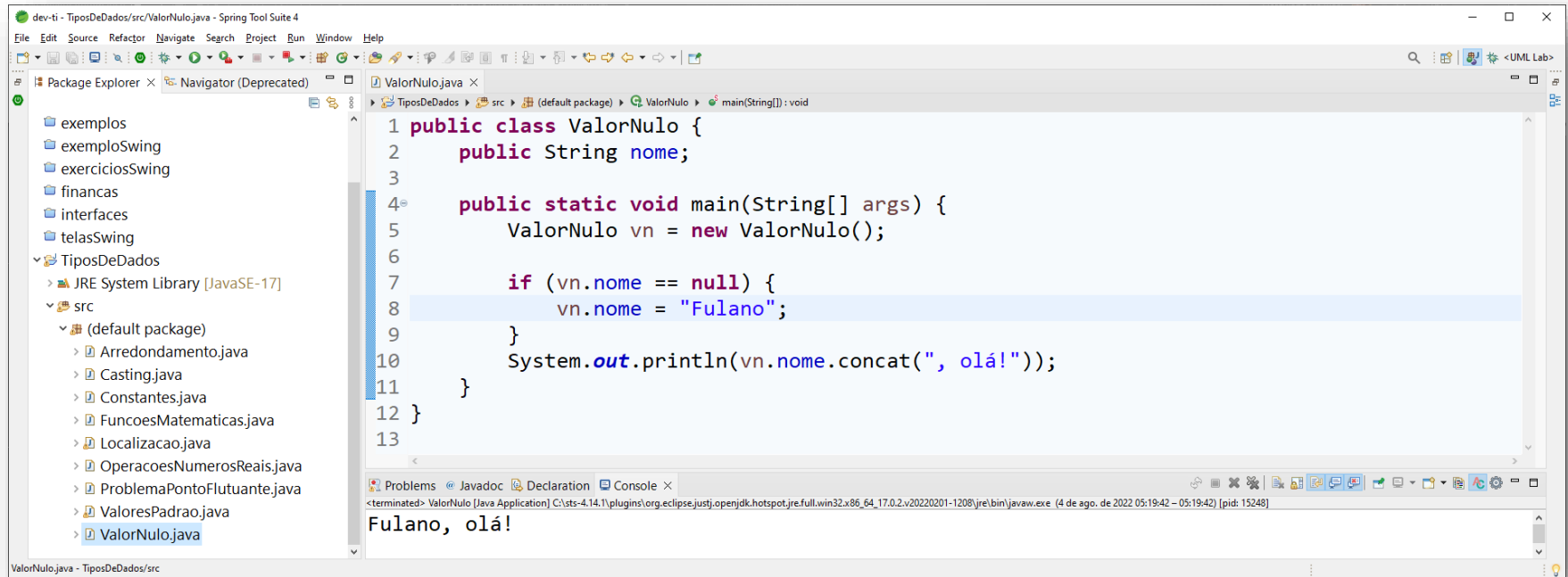
```
1 public class ValorNulo {
2     public String nome;
3
4     public static void main(String[] args) {
5         ValorNulo vn = new ValorNulo();
6
7         System.out.println(vn.nome.concat(", olá!"));
8     }
9 }
10
```

The console output 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)
```

# 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:

