

Aula 7 Arrays, ArrayLists e Conversão de Tipos

Rafael Geraldeli Rossi

Arrays

- *Arrays* são estruturas de dados consistindo em itens de dados do mesmo tipo
- Os **elementos** de um *array* podem ser **tipos primitivos** ou **tipos por referência**
- *Arrays* tornam **conveniente** o **armazenamento** e o **processamento** de grupos de itens do mesmo tipo
- A criação de um *array* requer a determinação do número de elementos
- **Após criado, a capacidade do array não é alterada**



Declarando Arrays

- A **expressão de criação de um array** é dada por:
 - 1 Tipo de variável do *array* (primitivo ou referência)
 - 2 Colchetes (“[]”)
 - 3 Nome de referência do *array* (identificador)
 - 4 Sinal de “=”
 - 5 Palavra-chave `new`
 - 6 Tipo do *array* (novamente)
 - 7 Tamanho do *array* entre colchetes (“[]”)

```
// Criação de um array com capacidade de armazenar 5 inteiros
int[] array = new int[5];
```

- Observar que a declaração de um *array* utiliza a palavra-chave `new` → um *array* é um **objeto** ou um **tipo por referência**



Declarando Arrays

- Sendo um objeto, um *array* possui métodos e atributos

The screenshot shows a portion of a Java code editor. Line 18 contains the declaration of an integer array: `int[] testeArray = new int[12];`. A tooltip is displayed over the variable `testeArray`, listing its methods and fields. The methods listed are: length (int), clone () (Object), equals (Object obj) (boolean), getClass () (Class<?>), hashCode () (int), notify () (void), notifyAll () (void), toString () (String), wait () (void), wait (long timeout) (void), and wait (long timeout, int nanos) (void). The `length` field is highlighted with a blue border.

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17  
18         int[] testeArray = new int[12];  
19         testeArray.  
20     }  
21 }  
22 }  
23 }  
24 }
```

length	int
clone()	Object
equals(Object obj)	boolean
getClass()	Class<?>
hashCode()	int
notify()	void
notifyAll()	void
toString()	String
wait()	void
wait(long timeout)	void
wait(long timeout, int nanos)	void

- Na verdade, os métodos são herdados da classe Object (veremos isso) e não são muito úteis
- Porém, o campo **length** é muito útil → **retorna o tamanho do array**

Declarando Arrays

- Ao declarar um array de tipos primitivos, todos os elementos do array são inicializados com os valores padrão de inicialização dos tipos primitivos (assim como ocorre nos objetos)**

The screenshot shows a Java code editor and a debugger's variable viewer. The code is:

```
14 public class Teste {  
15       
16     public static void main(String[] args) {  
17           
18         int[] testeArray = new int[12];  
19           
20         System.out.println("...");  
21     }  
22       
23 }
```

The variable viewer shows the state of the variable `testeArray`:

Name	Type	Value
testeArray	int[]	#721 (length=12)
[0]	int	0
[1]	int	0
[2]	int	0
[3]	int	0
[4]	int	0
[5]	int	0
[6]	int	0
[7]	int	0
[8]	int	0
[9]	int	0
[10]	int	0
[11]	int	0

Declarando Arrays

- Ao declarar um array de tipos por referência, todos os elementos do array são inicializados com null

The screenshot shows an IDE interface with Java code and a variable inspector.

```
14 public class Teste {  
15       
16     public static void main(String[] args) {  
17           
18         Pessoa[] testeArray = new Pessoa[12];  
19           
20         System.out.println("");  
21     }  
22 }  
23 }
```

The variable inspector window is open, showing the state of variables:

Nome	Tipo	Valor
ESTÁTICO		
args	String[]	#71(length=0)
testeArray	Pessoa[]	#72(length=12)
[0]		null
[1]		null
[2]		null
[3]		null
[4]		null
[5]		null
[6]		null
[7]		null
[8]		null
[9]		null
[10]		null
[11]		null

Curiosidades sobre declaração de *Arrays*

- Um programa pode criar **vários arrays de um mesmo tipo em uma única declaração**

```
// Criação simultânea de dois arrays do mesmo tipo
int[] array1 = new int[5], array2 = new int[10];
```

- Quando somente um *array* é declarado em cada declaração, os colchetes podem ser colocados depois do tipo ou depois do nome da variável

```
String[] array1 = new String[5];
String array2[] = new String[10];
```



Declarando / Inicializando um *Array*

- Pode-se criar um *array* e inicializar seus elementos com um **inicializador de array**
- Um inicializador de *array* é uma **lista de elementos separada por vírgula colocada entre chaves**

```
int[] array1 = {1, 4, 5, 7, 9, 15};  
String array2[] = {"Rafael", "Programação", "Orientada", "à", "Objetos", "SI"};
```

- **OBSERVAÇÃO:** o tamanho do *array* corresponde ao número de elementos da lista inicializadora



Acessando Elementos de um Array

- Para inserir ou recuperar valores (acessar um elemento) de um *array* , é necessário informar o nome de referência do *array*, seguido de [] e o índice do elemento entre os colchetes
- Os **valores dos índices são sequenciais e não há lacunas** entre os valores dos índices
- O primeiro elemento de cada *array* tem **ÍNDICE ZERO**



Acessando Elementos de um Array

Visão Geral de um Array

```
int[] c = {1, 45, -30, 60, 80, 100, -15, 0};
```



c[0]	1
c[1]	45
c[2]	-30
c[3]	60
c[4]	80
c[5]	100
c[6]	-15
c[7]	0

Acessando Elementos de um Array

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         int[] array = {1, 5, 6, 10, 15, 67, 5};  
4         System.out.println("Listando os elementos do array");  
5         for(int i=0;i<array.length;i++){  
6             System.out.println((i+1) + "º elemento: " + array[i]);  
7         }  
8     }  
9 }  
10
```

Terminal Window (Saída):

```
run:  
Listando os elementos do array  
1º elemento: 1  
2º elemento: 5  
3º elemento: 6  
4º elemento: 10  
5º elemento: 15  
6º elemento: 67  
7º elemento: 5
```

Acessando Elementos de um Array

- **OBSERVAÇÃO:** ao lidar com arrays de tipo por referência, é preciso criar objetos para as posições do array

The screenshot shows a Java code editor and a debugger's variable inspection window.

Code:

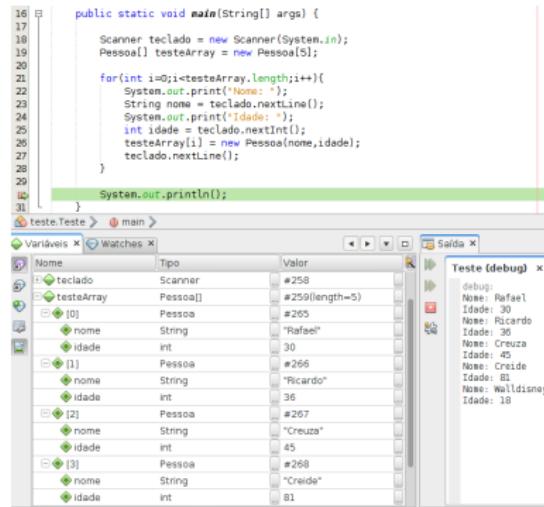
```
14  public class Teste {  
15    
16      public static void main(String[] args) {  
17          Pesssoa[] testeArray = new Pesssoa[5];  
18            
19          for(int i=0;i<testeArray.length;i++){  
20              testeArray[i] = new Pesssoa();  
21          }  
22            
23          System.out.println();  
24      }  
25        
26  }  
27 }
```

Variable Inspection Window:

Nome	Tipo	Valor
args	String[]	#73(length=0)
testeArray	Pesssoa[]	#74(length=5)
[0]	Pesssoa	#77
nome	String	--
idade	int	0
[1]	Pesssoa	#78
nome	String	--
idade	int	0
[2]	Pesssoa	#79
nome	String	--
idade	int	0
[3]	Pesssoa	#80
nome	String	--
idade	int	0

Acessando Elementos de um Array

- OBSERVAÇÃO:** ao lidar com arrays de tipo por referência, é preciso criar objetos para as posições do array



The screenshot shows an IDE interface with several windows:

- Code Editor:** Displays Java code for reading input from the console and creating a 5-element array of `Pessoal` objects.
- Variables Window:** Shows the state of variables during execution. It lists `teclado` (Scanner), `testeArray` (Pessoal[]), and individual elements `[0]` through `[4]`, each containing a `Pessoal` object with properties `nome` and `idade`.
- Output Window:** Labeled "Saída x", it shows the printed output of the program, listing five entries with names and ages: Rafael, Ricardo, Creusa, Crieide, and Walldisney.

```

16   public static void main(String[] args) {
17
18     Scanner teclado = new Scanner(System.in);
19     Pessoal[] testeArray = new Pessoal[5];
20
21     for(int i=0;i<testeArray.length;i++){
22       System.out.print("Nome: ");
23       String nome = teclado.nextLine();
24       System.out.print("Idade: ");
25       int idade = teclado.nextInt();
26       testeArray[i] = new Pessoal(nome,idade);
27       teclado.nextLine();
28     }
29
30     System.out.println();
31   }
  
```

Acessando Elementos de um Array

- Quando criado um *array* para armazenar objetos, ao acessar um *array* você estará acessando um objeto e portanto poderá acessar os membros desse objeto

The screenshot shows a Java code editor with the following code:

```
4 public class Programa {
5
6     public static void main(String[] args){
7
8         Pessoa[] pessoas = new Pessoa[4];
9
10    pessoas[0] = new Pessoa("Rafael", "0000000000");
11    pessoas[1] = new Pessoa("Abacata", "121564879878");
12    pessoas[2] = new Pessoa("Eita", "112456787");
13    pessoas[3] = new Pessoa("POO", "89789546123");
14
15    pessoas[2].]
16
17 }
```

The cursor is at the end of the line `pessoas[2].]`. A tooltip lists the methods available for the `Pessoa` object at that position:

- equals(Object obj) boolean
- getClass() Class<?>
- hashCode() int
- getId() int
- getName() String
- getQts_pessoas() int
- hashCode() int
- notify() void
- notifyAll() void
- printStatus() void
- setCpf(String cpf) void
- setId(int id) void
- setName(String nome) void
- setQts_pessoas(int qts_pessoas) void
- toString() String
- wait() void
- wait(long timeout) void

Acessando Elementos de um Array

- O Java não permite acessar posições inválidas de um array
- Por posições inválidas entende-se posições negativas ou posições além do tamanho do array
- A JVM verifica índices de array para assegura que eles são maiores que ou igual o comprimento do array e menores que 0
- Ao acessar uma posição inválida, o Java gera uma exceção do tipo `ArrayIndexOutOfBoundsException`



Copia de Arrays

- **OBSERVAÇÃO 1:** como arrays são objetos, `array1 = array2` só fará com que a variável `array1` aponte para o local em que a variável `array2` está apontando
- **OBSERVAÇÃO:** ao fazer `array1 = array2`, qualquer alteração em um elemento de `array2` também se refletirá no `array1`



Array no Método main

- Vocês repararam no argumento do método main????

```
14     public class Teste {  
15  
16     public static void main(String[] args) {  
17         System.out.println("Hello world!");  
18     }  
19 }  
20 }  
21 }  
22 }
```

- É um array!!!

Array no Método main

- Podemos passar valores como argumentos para esse método ao executar um programa Java

The screenshot shows a Java code editor and a terminal window. The code in the editor is:

```
14 public class Teste {  
15     public static void main(String[] args) {  
16         System.out.println("Hello " + args[0]);  
17     }  
18 }  
19  
20  
21 }  
22 }
```

The terminal window shows the following session:

```
Arquivo Editar Ver Pesquisar Terminal Ajuda  
rafael@gepic-rafael:~/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Códigos-Fonte  
/Aula7$ javac Teste.java  
rafael@gepic-rafael:~/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Códigos-Fonte  
/Aula7$ java Teste Rafael  
Hello Rafael  
rafael@gepic-rafael:~/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Códigos-Fonte  
/Aula7$
```

A red arrow points from the word "Saída" to the terminal output "Hello Rafael". Another red arrow points from the word "Entrada" to the command "java Teste Rafael".

Array no Método main

- Os valores dos parâmetros são separados por espaços

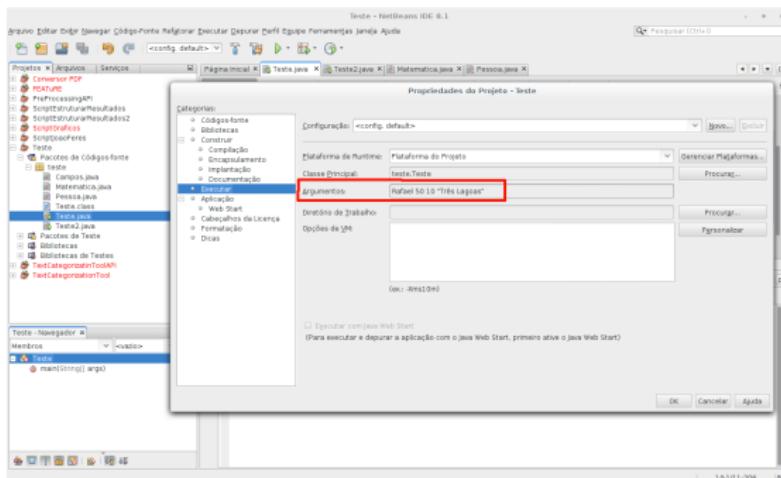
```
14 public class Teste {  
15     public static void main(String[] args) {  
16         for(int i=0;i<args.length;i++){  
17             System.out.println("Argumento " + i + " " + args[i]);  
18         }  
19     }  
20     Arquivo Editar Ver Pesquisar Terminal Ajuda  
21 }  
22 rafael@gepic-rafael:~/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Códigos-Fonte  
23 /Aula7$ java Teste Rafael 50 45 Daniel "Três Lagoas"  
24 Argumento i: Rafael  
Argumento i: 50  
Argumento i: 45  
Argumento i: Daniel  
Argumento i: Três Lagoas  
rafael@gepic-rafael:~/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Códigos-Fonte  
/Aula7$ █
```

- OBSERVAÇÃO:** todo o conteúdo entre aspas (mesmo que tenha espaço) é considerado como um único valor de argumento



Array no Método main

- Argumentos para o método main no Netbeans: Executar → Definir Configuração do Projeto → Executar → Argumentos



A instrução for aprimorada

- A **instrução for aprimorada** (também conhecida como **for each**) itera pelos elementos de um *array* **sem usar uma variável controladora** (contador)
- Isso evita a possibilidade de ultrapassar o limite do *array*

Sintaxe da instrução for aprimorada

```
for( tipo variável : nomeDoArray)  
    instrução;
```

- A variável deve ser declarada com o mesmo tipo do *array*
- A instrução for aprimorada iterar por valores sucessivos do *array* um a um



A instrução for aprimorada

- **OBSERVAÇÃO:** A variável não é um índice → a variável conterá o valor da uma determinada posição do array

The screenshot shows an IDE interface with two windows. The top window displays a Java code editor with the following content:

```
12 public class Teste {
13
14     public static void main(String[] args) {
15
16         int[] testeArray = {5, 6, 7, 11, 30};
17
18         for(int numero : testeArray){
19             System.out.println(numero);
20         }
21     }
22
23 }
```

The bottom window is a terminal-like interface titled "Console do Depurador" showing the output of the program:

```
run:
5
6
7
11
30
```

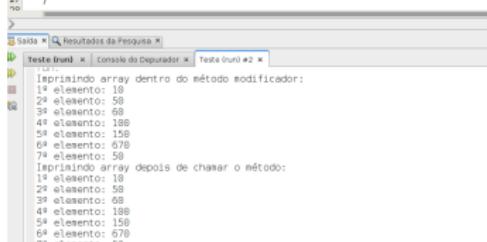
A instrução for aprimorada

```
12 public class Teste {  
13       
14     public static void main(String[] args) {  
15           
16         String[] testeArray = {"Eu", "Tu", "Ele", "Nós", "Vós", "Eles"};  
17           
18         for(String pronome : testeArray){  
19             System.out.println(pronome);  
20         }  
21     }  
22       
23 }  
24  
 teste.Teste > main > For (String pronome : testeArray) >  
Saída x  
Console do Depurador x Teste (run) x  
run:  
Eu  
Tu  
Ele  
Nós  
Vós  
Eles
```

Passando Array como Argumento de Método

- *Arrays podem ser passados como argumentos para métodos
→ passagem por referência*

```
1  public class Programa {
2
3      public static void main(String[] args){
4
5          int[] array = {1, 5, 6, 10, 15, 67, 5};
6
7          modificaArray(array);
8
9          System.out.println("Imprimindo array depois de chamar o método:");
10         for(int i=0;i<array.length;i++){
11             System.out.print(i+1" elemento: " + array[i]);
12         }
13     }
14
15     public static void modificaArray(int[] array){
16         for(int i=0;i<array.length;i++){
17             array[i] *= 10;
18         }
19
20         System.out.println("Imprimindo array dentro do método modificador:");
21         for(int i=0;i<array.length;i++){
22             System.out.print(i+1" elemento: " + array[i]);
23         }
24     }
25 }
26
27 }
```



Arrays como Retorno de Métodos

- Arrays podem ser usados como tipo de retorno de método

The screenshot shows an IDE interface with two tabs at the bottom: 'Saída' and 'Resultados da Pesquisa'. The 'Saída' tab contains the following text:

```
run:  
Imprimindo array retornado pelo método:  
1º elemento: 10  
2º elemento: 50  
3º elemento: 60  
4º elemento: 100  
5º elemento: 150  
6º elemento: 670  
7º elemento: 50
```

The Java code in the editor is:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         int[] array = {1, 5, 6, 10, 15, 67, 5};  
4         int[] arrayModificado = modificaArray(array);  
5         System.out.println("Imprimindo array retornado pelo método:");  
6         for(int i=0;i<array.length;i++){  
7             System.out.println((i+1) + "º elemento: " + arrayModificado(i));  
8         }  
9     }  
10    public static int[] modificaArray(int[] array){  
11        int[] novoArray = new int[array.length];  
12        for(int i=0;i<array.length;i++){  
13            novoArray[i] = array[i] * 10;  
14        }  
15        return novoArray;  
16    }  
17}
```

Arrays Multidimensionais

- Até o momento vimos *arrays* com um única dimensão → **array unidimensional ou vetor**
- Porém, podemos também declarar *arrays* com duas dimensões (**array bidimensional ou matriz**), ou ainda *arrays* com n dimensões (**array multidimensional**)
- Vale ressaltar que o Java não suporta *arrays* multidimensionais diretamente, mas permite especificar **arrays unidimensionais cujos elementos também são arrays unidimensionais** → alcançando assim o mesmo efeito



Declarando Arrays Multidimensionais

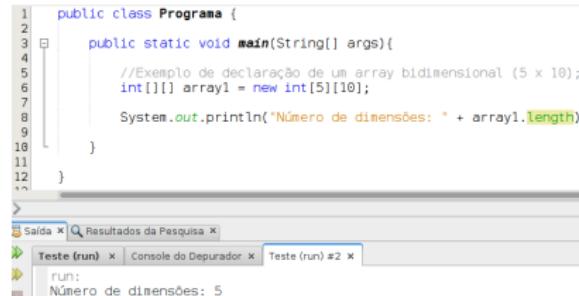
- A declaração de um *array* multidimensional é semelhante a declaração de *array* unidimensional, exceto pelo fato que **colchetes adicionais são utilizados representando as demais dimensões**

```
//Exemplo de declaração de um array bidimensional (5 x 10);  
int[][] array1 = new int[5][10];  
  
//Exemplo de declaração de um array tridimensional (5 x 8 x 10);  
String[][][] array2 = new String[5][8][10];
```



Declarando Arrays Multidimensionais

- **OBSERVAÇÃO 1:** em Java, a implementação de um *array* multidimensional nada mais são que **ponteiros para arrays** (todas as demais dimensões, exceto a última são ponteiros)
- **OBSERVAÇÃO 2:** o campo `length` de um *array* retorna o tamanho da primeira dimensão daquele *array*



A screenshot of an IDE interface. The code editor shows a Java program named 'Programa' with the following code:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         //Exemplo de declaração de um array bidimensional (5 x 10);  
4         int[][] array1 = new int[5][10];  
5         System.out.println("Número de dimensões: " + array1.length);  
6     }  
7 }  
8  
9  
10  
11  
12 }
```

The output window below the code editor shows the following results:

run:	Número de dimensões: 5
------	------------------------

Declarando Arrays Multidimensionais

- **OBSERVAÇÃO 3:** se um array multidimensional é um array de arrays, é possível retornar o tamanho de um array de um determinada dimensão

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         //Exemplo de declaração de um array bidimensional (5 x 10);
6         int[][] array1 = new int[5][10];
7
8         System.out.println("Número de dimensões da primeira dimensão: " + array1[1].length);
9     }
10
11 }
12 }
```

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code above.
- Toolbars:** Standard Java IDE toolbars for file operations, search, and navigation.
- Toolbox:** Shows icons for file, edit, run, and other development tasks.
- Bottom Bar:** Contains tabs for "Saída" (Output), "Resultados da Pesquisa" (Search Results), and three tabs under "Teste (run)": "Teste (run)", "Console do Depurador", and "Teste (run) #2".
- Output Window:** Shows the console output: "run:" followed by the printed value "Número de dimensões da primeira dimensão: 10".

Declarando Arrays Multidimensionais

- A primeira dimensão de um array deve ter tamanho fixo e as demais podem ter tamanho variado
- Quando uma dimensão puder ter tamanho variado, deve-se deixar o tamanho dessa dimensão vazio (“[]”) no momento da declaração do array



Declarando Arrays Multidimensionais

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor:

```
14 public class Teste {  
15     public static void main(String[] args) {  
16         // Criando uma matriz triangular  
17         int[][] teste = new int[10][];  
18  
19         for(int i=0;i<teste.length;i++){  
20             teste[i] = new int[i+1];  
21         }  
22  
23         for(int i=0;i<teste.length;i++){  
24             for(int j=0;j<teste[i].length;j++){  
25                 System.out.print(teste[i][j] + " ");  
26             }  
27             System.out.println();  
28         }  
29     }  
30 }  
31
```

Terminal (Saída):

```
teste.Teste > main > for (int i = 0; i < teste.length; i++) > for (int j = 0; j < teste[i].length; j++) > System.out.print(teste[i][j] + " ");  
Saída x  
Console do Depurador x Teste (run) x  
run:  
0  
0 0  
0 0 0  
0 0 0 0  
0 0 0 0 0  
0 0 0 0 0 0  
0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0
```

Ilustração de Arrays Multidimensionais

```
tipo[][] array = new tipo[n][m]
```

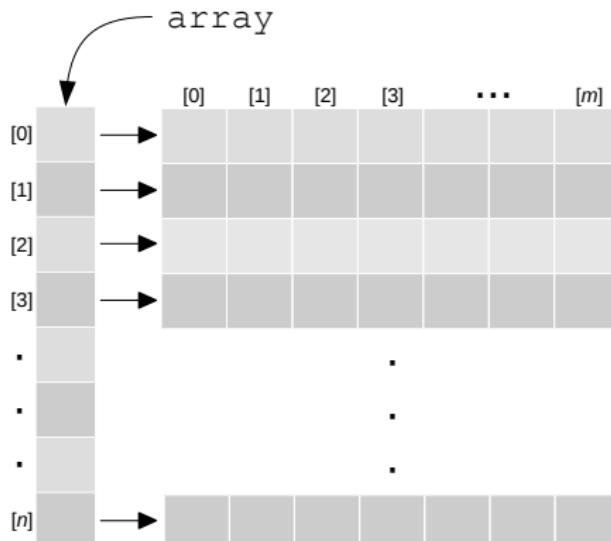


Ilustração de Arrays Multidimensionais

```
tipo[][] array = new tipo[n] []
```

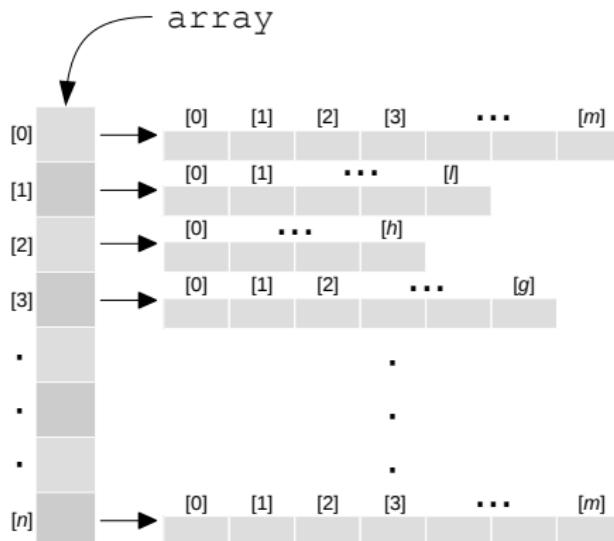
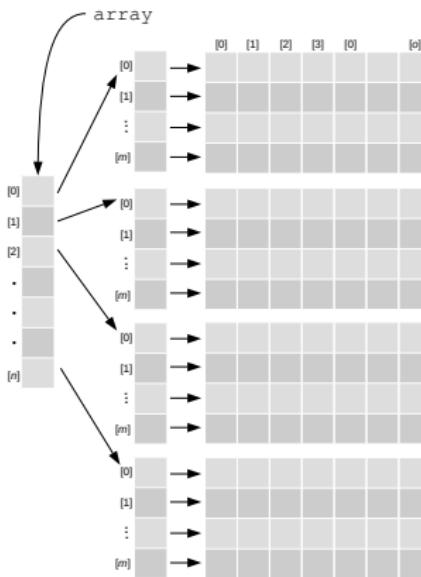


Ilustração de Arrays Multidimensionais

```
tipo[][] array = new tipo[n][m][o]
```



Declarando / Inicializando um *Array Multidimensional*

- Assim como os *arrays unidimensionais*, os *arrays multidimensionais* podem ser inicializados com *inicializadores de array* → **inicializadores de array aninhados**
- O número de listas irá determinar o número de linhas do *array* e o número de elementos dentro da lista irá determinar o número de dimensões do *array*

```
//Exemplo de inicialização de um array bidimensional de tamanho 3 × 2
int[][] array1 = {{1,2}, {3,4}, {4,6}};
```

```
//Exemplo de inicialização de um multidimensional de com tamanhos de dimensões variadas
int[][] array2 = {{1,2,4}, {5,6,7,8}, {9,12,13,15,17,20}};
```



Acessando Elementos de um *Array Multidimensional*

- Acesso de *arrays multidimensionais* é semelhante ao *arrays unidimensionais* → as posições nas dimensões devem ser especificadas entre colchetes

```
int[][] c = new int[3][3];
```



0 1 2

0	c[0][0]	c[0][1]	c[0][2]
1	c[1][0]	c[1][1]	c[1][2]
2	c[2][0]	c[2][1]	c[2][2]



Acessando Elementos de um *Array Multidimensional*

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays Java code for a `main` method. The code initializes a 5x4 matrix and prints its contents line by line.
- Output Area:** Shows the console output with the heading "run:" followed by the matrix data.
- Toolbars:** Includes standard Java IDE toolbars for file operations, search, and run.

```
17 public static void main(String[] args) {
18
19     int numLinhas = 5; int numColunas=4;
20
21     double[][] matriz = new double[numLinhas][numColunas];
22
23     for(int linha=0;linha<numLinhas;linha++){
24         for(int coluna=0;coluna<numColunas;coluna++){
25             matriz[linha][coluna] = linha * coluna;
26         }
27     }
28
29     for(int linha=0;linha<numLinhas;linha++){
30         for(int coluna=0;coluna<numColunas;coluna++){
31             System.out.print(matriz[linha][coluna] + "\t");
32         }
33         System.out.println();
34     }
35
36 }
```

teste.Teste > main > numLinhas >

Saída x

Console do Depurador x Teste (run) x

run:
0.0 0.0 0.0 0.0
0.0 1.0 2.0 3.0
0.0 2.0 4.0 6.0
0.0 3.0 6.0 9.0
0.0 4.0 8.0 12.0

A Instrução for Aprimorada

- No modo tradicional, precisamos de duas instruções `for` para percorrer um *array* bidimensional → um controlando a movimentação na linha e outro controlando a movimentação na coluna
- No caso de uma instrução `for` aprimorada, teremos um `for` para retornar o *array* de uma dimensão e um outro `for` para percorrer esse *array*

A Instrução for Aprimorada

The screenshot shows an IDE interface with a code editor and a run output window.

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         int[][] c = new int[3][3];
6
7         for(int[] a : c){
8             for(int b: a){
9                 System.out.print(b + "\t");
10            }
11            System.out.println("");
12        }
13    }
14
15 }
```

The run output window shows the following text:

Saída x Resultados da Pesquisa x

Teste (run) x Console do Depurador x Teste (run) #2 x

run:
0 0 0
0 0 0
0 0 0

Arrays Multidimensionais como Argumentos ou Retornos de Métodos

Funcionam da mesma forma que os arrays unidimensionais

```
1  public class Programa {
2
3      public static void main(String[] args) {
4          int[][] array = {{1,2,3}, {4,5,6}, {7,8,9}};
5          int[][] arrayModificado = modificaArray(array);
6
7          for(int i=0;i<arrayModificado.length;i++){
8              for(int j=0;j<arrayModificado[i].length;j++){
9                  System.out.print(arrayModificado[i][j] + "\t");
10             }
11             System.out.println("");
12         }
13     }
14
15     public static int[][] modificaArray(int[][] array){
16         // criado um arrayModificado com as mesmas dimensões do array original
17         int[][] arrayModificado = new int[array.length][];
18         for(int i=0;i<array.length;i++){
19             arrayModificado[i] = new int[array[i].length];
20
21             for(int l=0;l<array[i].length;l++){
22                 for(int j=0;j<array[i].length;j++){
23                     arrayModificado[i][l] = array[i][j] * 10;
24                 }
25             }
26         }
27         return arrayModificado;
28     }
29 }
```

10	20	30
40	50	60
70	80	90

Classe Arrays

- A classe **Arrays** (`java.util.Arrays`) evita reinventar a roda fornecendo métodos `static` para manipulações de *arrays* comuns
- Principais métodos da classe **Arrays**
 - `sort`: ordenar um *array*
 - `binarySearch`: para pesquisar em um *array* (verificar se um valor está contido em um *array*)
 - `equals`: comparar dois *arrays*
 - `fill`: inserir valores em um *array*



Método sort(...)

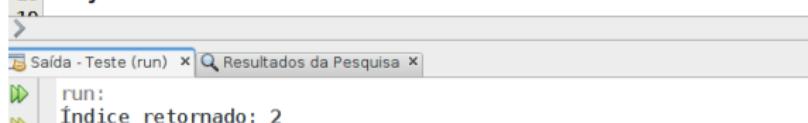
Ordenando valores com o método sort

```
15 public class Teste {  
16  
17     public static void main(String[] args) {  
18         // Criando uma matriz triangular  
19         int[] testeArray = {5, 10, 7, 9, 15, 35, 1, 40};  
20  
21         Arrays.sort(testeArray);  
22  
23         for(int i=0;i<testeArray.length;i++){  
24             System.out.print(testeArray[i] + " ");  
25         }  
26         System.out.println();  
27     }  
28 }  
29  
30 }  
31 }  
teste.Teste > main >  
Saída >  
Console do Depurador > Teste (run) >  
run:  
1 5 7 9 10 15 35 40
```

Método binarySearch(...)

Procurando valores com o método binarySearch

```
2 | import java.util.Arrays;
3 |
4 | public class TesteArray {
5 |
6 |     public static void main(String[] args) {
7 |
8 |         int[] array1 = {1, 10, 50, 75, 100};
9 |
10|         int key = 50;
11|
12|         int index = Arrays.binarySearch(array1, key);
13|
14|         System.out.println("Índice retornado: " + index);
15|
16|     }
17|
18| }
```



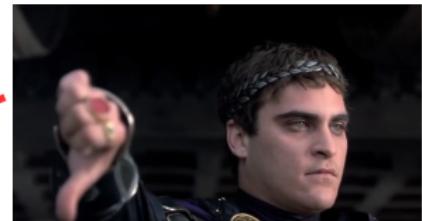
The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code for the `TesteArray` class.
- Output Area:** Shows the run configuration: "Saída - Teste (run)" and the results of the run: "run:" followed by "Índice retornado: 2".
- Bottom Bar:** Includes tabs for "Saída - Teste (run)" and "Resultados da Pesquisa", along with standard icons for file operations.

Método binarySearch(...)

OBSERVAÇÃO: vale ressaltar que o método a **busca binária** só tem garantia de funcionamento correto em um vetor ordenado

```
2  import java.util.Arrays;  
3  
4  public class TesteArray {  
5  
6      public static void main(String[] args) {  
7  
8          int[] array1 = {9, 8, 7, 6, 5, 4, 3, 2, 1};  
9  
10         int key = 2;  
11  
12         int index = Arrays.binarySearch(array1, key);  
13  
14         System.out.println("Índice retornado: " + index);  
15  
16     }  
17  
18 }
```



```
Saída - Teste (run) x Resultados da Pesquisa x  
run:  
Índice retornado: -1
```

Método binarySearch(...)

OBSERVAÇÃO: vale ressaltar que o método a **busca binária** só tem garantia de funcionamento correto em um vetor ordenado

```
2 import java.util.Arrays;
3
4 public class TesteArray {
5
6     public static void main(String[] args) {
7
8         int[] array1 = {9, 8, 7, 6, 5, 4, 3, 2, 1};
9
10        int key = 2;
11
12        Arrays.sort(array1);
13        int index = Arrays.binarySearch(array1, key);
14
15        System.out.println("Índice retornado: " + index);
16    }
17
18
19 }
```

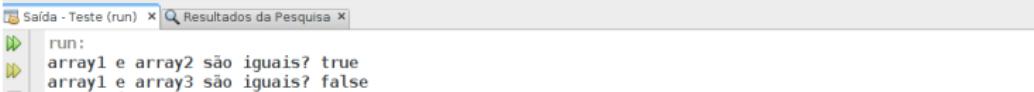
```
Salida - Teste (run) x Resultados da Pesquisa x
run:
Índice retornado: 1
```



Método `equals(...)`

Comparando arrays utilizando o método `equals`

```
2  import java.util.Arrays;
3
4  public class TesteArray {
5
6      public static void main(String[] args) {
7
8          int[] array1 = {1, 10, 50, 75, 100};
9          int[] array2 = {1, 10, 50, 75, 100};
10         int[] array3 = {10, 1, 75, 100, 50};
11
12         System.out.println("array1 e array2 são iguais? " + Arrays.equals(array1, array2));
13         System.out.println("array1 e array3 são iguais? " + Arrays.equals(array1, array3));
14     }
15 }
16
17 }
```



The screenshot shows the Java code running in an IDE. The output window displays the results of the `System.out.println` statements. The first line shows that `array1` and `array2` are equal, returning `true`. The second line shows that `array1` and `array3` are not equal, returning `false`.

```
Saída - Teste (run) x Resultados da Pesquisa x
run:
array1 e array2 são iguais? true
array1 e array3 são iguais? false
```

ArrayList

- A *API* do Java fornece várias estruturas de dados predefinidas, chamadas **coleções**, as quais são utilizadas para armazenar grupos de objetos relacionados
- Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados sem que seja necessário conhecer como os dados são armazenados → reduz o tempo de desenvolvimento de aplicativos



ArrayList

- Os *arrays* estudados até agora também armazenam sequências de objetos, porém, não podem alterar automaticamente seu tamanho
- **Duas consequências**
 - O programador deve prever o tamanho total de objetos a serem armazenados
 - Pode haver desperdício de memória



ArrayList

- A classe **ArrayList** (pacote `java.util`) fornece uma solução conveniente para **armazenar uma sequência de objetos**, na qual **o número de objetos a serem armazenados pode variar dinamicamente** ao longo do tempo
- A sintaxe de declaração de um **ArrayList** é:

```
ArrayList<TipoObjeto> nomeDaVariável = new  
ArrayList<Tipo>();
```

```
ArrayList<String> array = new ArrayList<String>();
```



ArrayList

- Um ArrayList não pode ser criado para armazenar tipos primitivos → **apenas objetos**
- Para criar um ArrayList para armazenar valores de tipos primitivos, tem-se que utilizar objetos correspondentes aos tipos primitivos

```
ArrayList<Integer> array = new ArrayList<Integer>();
```



Métodos

Métodos mais comum da classe ArrayList

Método	Descrição
add(Objeto)	Adiciona um objeto no final do ArrayList
add(indice, Objeto)	Adiciona um objeto na posição especificada pelo índice
clear()	Remove todos os elementos do ArrayList
contains(Objeto)	Retorna verdadeiro se o ArrayList contém o objeto especificado
indexOf(Objeto)	Retorna o índice da primeira ocorrência do objeto (-1 se não existir)
remove(Objeto)	Remove a primeira ocorrência do objeto especificado
remove(indice)	Remove o objeto no índice especificado
size()	Retorna o número de objetos armazenados no ArrayList



Métodos

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor:

```
19  ArrayList<String> testeArray = new ArrayList<String>();
20
21  String nome1 = new String("Professor");
22  testeArray.add(nome1);
23  String nome2 = new String("Rafael");
24  testeArray.add(nome2);
25  String nome3 = new String("Geraldeli");
26  testeArray.add(nome3);
27  String nome4 = new String("Rossi");
28  testeArray.add(nome4);
29
30  System.out.println("O tamanho do ArrayList é " + testeArray.size());
31  System.out.println("O segundo elemento do ArrayList é " + testeArray.get(1));
32
33  testeArray.remove(1);
34  System.out.println("O tamanho do ArrayList é " + testeArray.size());
35  System.out.println("O segundo elemento do ArrayList é " + testeArray.get(1));
```

Terminal (Saída):

```
run:
O tamanho do ArrayList é 4
O segundo elemento do ArrayList é Rafael
O tamanho do ArrayList é 3
O segundo elemento do ArrayList é Geraldeli
```

Métodos

The screenshot shows an IDE interface with the following components:

- Code Editor:** Displays Java code for an `ArrayList`. The code initializes an `ArrayList` named `testeArray`, adds three string elements ("Rafael", "Geraldeli", "Rossi") to it, performs a search for the string "Geraldeli", and then clears the list.
- Run Tab:** Shows the current run configuration: `teste.Teste > main`.
- Output Window:** Titled "Saída", it contains the execution results:

```
run:
O array contém a palavra Geraldeli
A palavra Geraldeli está na posição 1 do array
Tamanho do ArrayList: 0
```

ArrayList de ArrayList

- Assim como podemos ter *arrays* de *arrays* podemos ter *ArrayLists* de *ArrayLists*
- Como uma *ArrayList* também é um objeto, e o argumento para declaração de uma *ArrayList* é um objeto, basta passar uma *ArrayList* como argumento ao criar uma *ArrayList*

```
ArrayList<ArrayList<String>> array = new ArrayList<ArrayList<String>>();
```



ArrayList de ArrayList

The screenshot shows an IDE interface with the following components:

- Code Editor:** Displays Java code for creating nested ArrayLists and printing specific elements. The code uses nested ArrayList<String> declarations and System.out.println() statements.
- Toolbars:** Standard Java development toolbars for file operations, code navigation, and selection.
- Project Explorer:** Shows a project named "teste.Teste" with a "main" class selected.
- Output Window:** Titled "Saída", it contains the results of the program execution.
- Console:** Titled "Console do Depurador" and "Teste (run)", it shows the output of the run command.

```
public static void main(String[] args) {  
    ArrayList<ArrayList<String>> testeArray = new ArrayList<ArrayList<String>>();  
  
    ArrayList<String> testeArray1 = new ArrayList<String>();  
    testeArray1.add(new String("Rafael"));  
    testeArray1.add(new String("Geraldeli"));  
    testeArray1.add(new String("Rossi"));  
  
    testeArray.add(testeArray1);  
  
    ArrayList<String> testeArray2 = new ArrayList<String>();  
    testeArray2.add(new String("Sistemas"));  
    testeArray2.add(new String("de"));  
    testeArray2.add(new String("Informação"));  
  
    testeArray.add(testeArray2);  
  
    System.out.println("O terceiro elemento do primeiro elemento é : " + testeArray.get(0).get(2));  
    System.out.println("O primeiro elemento do segundo elemento é : " + testeArray.get(1).get(0));  
}
```

Console Output (Saída):

```
run:  
O terceiro elemento do primeiro elemento é : Rossi  
O primeiro elemento do segundo elemento é : Sistemas
```

Instrução for aprimorada

- Também podemos utilizar a instrução **for** aprimorada considerando um **ArrayList**

The screenshot shows an IDE interface with the following components:

- Code Editor:** Displays the following Java code:

```
1 import java.util.ArrayList;
2
3 public class Programa2 {
4
5     public static void main(String[] args){
6
7         ArrayList<String> array = new ArrayList<String>();
8
9         array.add("Programação");
10        array.add("Orientada");
11        array.add("Objetos");
12
13        for(String str : array){
14            System.out.println(str);
15        }
16    }
17
18 }
```

- Output Window:** Shows the console output:

```
run:
Programação
Orientada
Objetos
```
- Toolbars:** Includes tabs for "Saída", "Resultados da Pesquisa", "Teste (run)", "Console do Depurador", and "Teste (run) #2".
- Bottom Right:** Includes standard window control icons (minimize, maximize, close).

Instrução for aprimorada

- Também podemos utilizar a instrução **for** aprimorada considerando um **ArrayList**

```
1  import java.util.ArrayList;
2
3  public class Programa2 {
4
5      public static void main(String[] args){
6
7          ArrayList<Pessoa> array = new ArrayList<Pessoa>();
8
9          array.add(new Pessoa("Ricardão", "014.587.102-45"));
10         array.add(new Pessoa("Janete", "012.789.456-68"));
11         array.add(new Pessoa("Ratinho", "758.153.456-28"));
12
13         for(Pessoa pessoa : array){
14             System.out.println(pessoa.toString());
15         }
16     }
17 }
18
19 >
```

Salida x Resultados da Pesquisa x

Teste (run) x Console do Depurador x Teste (run) #2 x

run:

Nome: Ricardão CPF: 014.587.102-45

Nome: Janete CPF: 012.789.456-68

Nome: Ratinho CPF: 758.153.456-28

Iterator

- Podemos também utilizar um objeto da classe Iterator para percorrer um ArrayList
- Assim como na instrução for aprimorada, ao utilizar um Iterator, não é preciso especificar os índices
- A classe ArrayList contém um método iterator() que retorna um objeto do tipo Iterator
- Dois métodos básicos para a utilização do Iterator são:
 - **next()**: retorna um elemento da lista e move o ponteiro para o próximo elementos
 - **hasNext()**: verifica se existe um próximo elemento



Iterator

Exemplo de recuperação de elementos utilizando um Iterator

```
15 public class Teste {
16
17     public static void main(String[] args) {
18
19         ArrayList<String> testeArray1 = new ArrayList<String>();
20         testeArray1.add(new String("Rafael"));
21         testeArray1.add(new String("Geraldeli"));
22         testeArray1.add(new String("Rossi"));
23
24         Iterator<String> iterator = testeArray1.iterator();
25         while(iterator.hasNext()){
26             System.out.println(iterator.next());
27         }
28
29     }
30
31 }
32 }
```

teste.Teste ▶ main ▶ while (iterator.hasNext()) ▶

Salida □

Console do Depurador □ Teste (run) □

run:
Rafael
Geraldeli
Rossi .

Iterator

Como ficaria se quiséssemos remover um elemento utilizando a instrução for?

```
1 import java.util.ArrayList;
2 import java.util.Iterator;
3
4 public class Programa2 {
5
6     public static void main(String[] args){
7
8         ArrayList<Pessoa> array = new ArrayList<Pessoa>();
9
10        array.add(new Pessoa("Ricardão", "014.587.102-45"));
11        array.add(new Pessoa("Janete", "012.789.456-68"));
12        array.add(new Pessoa("Ratinho", "758.153.456-28"));
13        array.add(new Pessoa("Janete", "365.254.698-87"));
14        array.add(new Pessoa("Abelardo", "874.985.541-65"));
15
16        String removerNome = "Janete";
17
18        for(int i=0;i<array.size();i++){
19            if(array.get(i).getName().equals(removerNome)){
20                array.remove(i);
21                i--;
22            }
23        }
24
25        for(Pessoa pessoa : array){
26            System.out.println(pessoa.toString());
27        }
28    }
29
30}
31
32 }
```

Salida x Resultados da Pesquisa x

Teste (run) x Consola do Depurador x Teste (run) #2 x

run:
Note: Ricardão CPF: 014.587.102-45
Note: Ratinho CPF: 758.153.456-28
Note: Abelardo CPF: 874.985.541-65

Iterator

Exemplo da remoção de um elemento utilizando um Iterator

```
1 import java.util.ArrayList;
2 import java.util.Iterator;
3
4 public class Programa2 {
5
6     public static void main(String[] args){
7
8         ArrayList<Pessoa> array = new ArrayList<Pessoa>();
9
10        array.add(new Pessoa("Ricardão", "014.587.102-45"));
11        array.add(new Pessoa("Janete", "012.780.456-68"));
12        array.add(new Pessoa("Rotinho", "758.153.456-28"));
13        array.add(new Pessoa("Janete", "365.254.698-87"));
14        array.add(new Pessoa("Abelardo", "874.985.541-65"));
15
16        String removerNome = "Janete";
17
18        //Removendo a pessoa que tem o mesmo nome da String removerNome
19        Iterator<Pessoa> iterator = array.iterator();
20        while(iterator.hasNext()){
21            if(iterator.next().getNome().equals(removerNome)){
22                iterator.remove();
23            }
24        }
25
26        for(Pessoa pessoa : array){
27            System.out.println(pessoa.toString());
28        }
29    }
30}
31
32 }
```

The screenshot shows an IDE interface with two tabs at the bottom: 'Saída' (Output) and 'Resultados da Pesquisa x'. The 'Saída' tab contains the output of the program's execution:

```
run:
Nome: Ricardão  CPF: 014.587.102-45
Nome: Rotinho   CPF: 758.153.456-28
Nome: Abelardo  CPF: 874.985.541-65
```

Conversão de Tipos

- Em algumas situações, é necessário realizar a conversão de tipos, seja para realizar operações matemáticas, seja pelos tipos de dados que deverão ser utilizados no programa
- Um exemplo clássico da necessidade de conversão de tipos é a divisão de dois número inteiros

The screenshot shows a Java code editor and a terminal window. The code in the editor is:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         int idadeTotalPopulacao = 5000;  
4         int numPessoasPopulacao = 68;  
5  
6         double idadeMediaPopulacao = idadeTotalPopulacao / numPessoasPopulacao;  
7  
8         System.out.println("Idade Média da População: " + idadeMediaPopulacao);  
9     }  
10    }  
11  
12  
13  
14 }
```

The terminal window below shows the output of the program:

```
Teste (run) x | Console do Depurador x | Teste (run) #2 x  
run:  
Idade Média da População: 73.0
```

- Porém, o resultado correto é 73,529411765

Conversão de Tipos

- Isso acontece porque uma operação entre inteiros resulta em um inteiro, independente se esse resultado está sendo atribuído em outro tipo de variável
- Para gerar um double, ao menos um dos tipos envolvidos na divisão tem que ser double

The screenshot shows a Java code editor with the following code:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         int idadeTotalPopulacao = 5000;  
4         double numPessoasPopulacao = 68;  
5         double idadeMediaPopulacao = idadeTotalPopulacao / numPessoasPopulacao;  
6         System.out.println("Idade Média da População: " + idadeMediaPopulacao);  
7     }  
8 }
```

Below the code editor is a terminal window showing the output of the program:

```
Salida x Resultados da Pesquisa x  
Teste (run) x | Console do Depurador x | Teste (run) #2 x  
run:  
Idade Média da População: 73.52941176470588
```

Conversão de Tipos

- Outra forma é fazer uma conversão do tipo durante a operação → **casting**
- Para utilizar o *casting*, basta colocar o tipo desejado entre parênteses “()” na frente do identificador da variável que se deseja converter

The screenshot shows a Java code editor with the following code:

```
1 public class Programa {  
2     public static void main(String[] args){  
3         int idadeTotalPopulacao = 5000;  
4         int numPessoasPopulacao = 68;  
5  
6         double idadeMediaPopulacao = idadeTotalPopulacao / (double)numPessoasPopulacao;  
7  
8         System.out.println("Idade Média da População: " + idadeMediaPopulacao);  
9     }  
10    }  
11 }
```

Below the code editor is a tool bar with tabs: Salida, Resultados da Pesquisa, Teste (run), Consola do Depurador, and Teste (run) #2. The 'Teste (run)' tab is active. In the bottom status bar, it says 'run:' followed by the output: 'Idade Média da População: 73.52941176470588'.

Conversão de Tipos

- Em Java, o operador () é chamado de **operador unário de coerção**
- A utilização de um operador de coerção é chamado de **conversão explícita ou coerção de tipo**
- **O operador unário de coerção cria uma cópia temporária com o tipo especificado**



Conversão de Tipos

- **OBSERVAÇÃO 1:** nas expressões aritméticas os tipos precisam ser idênticos
- **OBSERVAÇÃO 2:** para assegurar que os operandos sejam do mesmo tipo, o Java realiza uma operação chamada de **promoção de tipos** (ou **conversão implícita**)
- **OBSERVAÇÃO 3:** a conversão será para o tipo de maior capacidade de armazenamento que está envolvido na operação



Conversão de Tipos

PARA / DE	byte	short	char	int	long	float	double
byte	---	<i>Impl.</i>	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
short	(byte)	---	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
char	(byte)	(short)	---		<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
int	(byte)	(short)	(char)	---		<i>Impl.</i>	<i>Impl.</i>
long	(byte)	(short)	(char)	(int)	---	<i>Impl.</i>	<i>Impl.</i>
float	(byte)	(short)	(char)	(int)	(long)	---	<i>Impl.</i>
double	(byte)	(short)	(char)	(int)	(long)	(float)	---



Geração de Números Aleatórios

- Números aleatórios são bastante utilizados em jogos ou simulações
- Inserem um fator “sorte” ou um fator “acaso” na formulação do problema
- Números aleatórios podem ser gerados via a instanciação da classe Random (pacote java.util) ou via o método static random da classe Math



Geração de Números Aleatórios

- Os objetos da classe `Random` podem produzir valores aleatórios `boolean`, `byte`, `float`, `double`, `int`, `long`, e gaussianos (distribuição normal)
- O método `random()` da classe `Math` pode produzir somente valores `double` no intervalo $0.0 \leq x < 1.0$, na qual x é o valor retornado pelo método `random`



Random

Exemplo de utilização da classe Random

```
14 public class Teste {
15
16     public static void main(String[] args) {
17
18         Random aleatorio = new Random();
19
20         System.out.println("Inteiro aleatório 1: " + aleatorio.nextInt());
21         System.out.println("Inteiro aleatório 2: " + aleatorio.nextInt());
22         System.out.println("Inteiro aleatório limitado em 5: " + aleatorio.nextInt(5));
23         System.out.println("Double aleatório 1: " + aleatorio.nextDouble());
24         System.out.println("Double aleatório 2: " + aleatorio.nextDouble());
25         System.out.println("Boolean aleatório 1: " + aleatorio.nextBoolean());
26         System.out.println("Boolean aleatório 2: " + aleatorio.nextBoolean());
27     }
28 }
```

teste.Teste > main >

Saída

Console do Depurador > Teste (run) >

```
run:
Inteiro aleatório 1: -115548456
Inteiro aleatório 2: -723955400
Inteiro aleatório limitado em 5: 4
Double aleatório 1: 0.6063452159973596
Double aleatório 2: 0.3090505681997092
Boolean aleatório 1: false
Boolean aleatório 2: true
```

Random

- **OBSERVAÇÃO:** é possível criar um objeto Random e definir uma semente, de forma que sempre serão gerados os mesmos números aleatórios

The screenshot shows an IDE interface with a code editor and a terminal window.

Code Editor:

```
14 public class Teste {
15
16     public static void main(String[] args) {
17
18         Random aleatorio = new Random();
19
20         System.out.println("Inteiro aleatório 1: " + aleatorio.nextInt());
21         System.out.println("Inteiro aleatório 2: " + aleatorio.nextInt());
22         System.out.println("Inteiro aleatório limitado em 5: " + aleatorio.nextInt(5));
23         System.out.println("Double aleatório 1: " + aleatorio.nextDouble());
24         System.out.println("Double aleatório 2: " + aleatorio.nextDouble());
25         System.out.println("Boolean aleatório 1: " + aleatorio.nextBoolean());
26         System.out.println("Boolean aleatório 2: " + aleatorio.nextBoolean());
27     }
28 }
```

Terminal (Saída):

```
Console do Depurador x Teste (run) x
run:
Inteiro aleatório 1: -1155484576
Inteiro aleatório 2: -723955400
Inteiro aleatório limitado em 5: 4
Double aleatório 1: 0.6063452159973596
Double aleatório 2: 0.309050568199702
Boolean aleatório 1: false
Boolean aleatório 2: true
```

Método random da classe Math

Exemplo de utilização do método random da classe Math

```
14  public class Teste {
15
16      public static void main(String[] args) {
17
18          Random aleatorio = new Random(0);
19
20          System.out.println("Double aleatório 1: " + Math.random());
21          System.out.println("Double aleatório 2: " + Math.random());
22          System.out.println("Double aleatório 3: " + Math.random());
23          System.out.println("Double aleatório 4: " + Math.random());
24      }
25
26
27  }
```

The screenshot shows an IDE interface with the following components:

- Editor Area:** Displays the Java code for the `Teste` class.
- Run/Debug Toolbar:** Shows icons for running, stopping, and refreshing the application.
- Saída (Output) Tab:** Contains two tabs: `Console do Depurador` and `Teste (run)`. The `Teste (run)` tab is active.
- Console Output:** Shows the results of the `System.out.println` statements from the code:

```
RUN:
Double aleatório 1: 0.649162604569316
Double aleatório 2: 0.2892136336071297
Double aleatório 3: 0.040004371769932545
Double aleatório 4: 0.9715305043376993
```

- Em Java, podemos utilizar objetos para formatar números, datas, ...
- Normalmente esses objetos contém um método **format(...)**, o qual irá receber o valor a ser formatado e gerará uma string corresponde ao valor formatado
- Para formatar números, vamos instanciar objetos da classe DecimalFormat e NumberFormat



DecimalFormat

- Classe DecimalFormat do pacote `java.text.DecimalFormat`

The screenshot shows an IDE interface with two tabs at the bottom: "teste.Teste >" and "Saída - Teste (run) >". The code in the editor is as follows:

```
14 public class Teste {
15
16     public static void main(String[] args) {
17
18         DecimalFormat format = new DecimalFormat();
19         format.setMaximumFractionDigits(2);
20         format.setMinimumFractionDigits(1);
21
22         double numero1 = 1582222;
23         System.out.println(format.format(numero1));
24
25         double numero2 = 1582222.7589;
26         System.out.println(format.format(numero2));
27     }
28
29
30 }
```

The "Saída - Teste (run)" tab shows the output:

```
run:
1.582.222,0
1.582.222,76
```

- OBSERVAÇÃO 1:** há um arredondamento de números ao utilizar esta classe
- OBSERVAÇÃO 2:** saída formatada no padrão do sistema

DecimalFormat

- Outras formas de uso do DecimalFormat

```
10 public class Teste2 {
11
12     public static void main(String[] args){
13
14         double numerador = 1000.0;
15         double denominador = 7.0;
16         double divisao = numerador/denominador;
17
18         DecimalFormat df = new DecimalFormat("0.#####");
19         String saida = df.format(divisao);
20         System.out.println("Sem formatação: " + divisao);
21         System.out.println("Com formatação: " + saida);
22     }
23
24 }
```

teste.Teste2 >

Saída x

Console do Depurador x Teste (run) x

run:
Sem formatação: 142.85714285714286
Com formatação: 142,8571

DecimalFormat

- Alguns exemplos de padrões que podem ser passados para o construtor da classe DecimalFormat

Número	Padrão	Saída
123456.789	###,###.###	123,456.789
123456.789	###.##	123456.79
123.78	000000.000	000123.780
12345.67	\$###,###.###	\$12,345.67

DecimalFormat

- Outras formas de uso do DecimalFormat

The screenshot shows an IDE interface with a code editor and a run output window.

```
14 public class Teste {
15
16     public static void main(String[] args) {
17
18         double x = 125878945.358258;
19
20         DecimalFormat df = new DecimalFormat("BLA###,###.###");
21         System.out.println(df.format(x));
22     }
23
24 }
25
```

The code defines a class named `Teste` with a `main` method. It creates a `DecimalFormat` object with the pattern "BLA###,###.###" and prints its formatted value. The run output window shows the result: `BLA125.878.945,358`.

DecimalFormat

- **OBSERVAÇÃO 1:** vale ressaltar que a vírgula (",") é o separador de cadas decimais e o ponto (".") é o separador de casas decimais da linguagem de programação
- **OBSERVAÇÃO 2:** entretanto, os símbolos de separação de casas são convertidos para a localidade do computador ao aplicar a formatação

DecimalFormat

```
2 import java.text.DecimalFormat;
3
4 public class TesteArray {
5
6     public static void main(String[] args) {
7
8         double numero = 45645456456.787897897;
9
10        DecimalFormat formatador = new DecimalFormat("###,###,##");
11
12        System.out.println(formatador.format(numero));
13    }
14
15
16 }
17
```

The screenshot shows an IDE interface with Java code in a editor window. A red arrow points from the circled line of code `formatador = new DecimalFormat("###,###,##");` to the circled output in the terminal window below. The terminal window shows the formatted number `45.645.456,456,79`.

NumberFormat

- A classe `NumberFormat` do pacote

`java.text.NumberFormat` permite aplicar a formatação de números, moedas e porcentagem correntes do sistema ou ainda especificar a língua e país da formatação

```
14  public class Teste {
15
16      public static void main(String[] args) {
17
18          double x = 125878945.350258;
19
20          NumberFormat formatter1 = NumberFormat.getNumberInstance();
21          String s = formatter1.format(x);
22          System.out.println("Formatação de número:" + s);
23
24          NumberFormat formatter2 = NumberFormat.getCurrencyInstance();
25          s = formatter2.format(x);
26          System.out.println("Formatação de moeda:" + s);
27
28          NumberFormat formatter3 = NumberFormat.getPercentInstance();
29          s = formatter3.format(x);
30          System.out.println("Formatação de porcentagem:" + s);
31
32      }
33
34  }
```

The screenshot shows an IDE interface with a code editor containing the provided Java code. Below the editor is a toolbar with icons for file operations. At the bottom, there are tabs for 'Saída' (Output), 'Console do Depurador' (Debugger Console), and 'Teste (run)' (Run Test). The 'Saída' tab is active, displaying the program's output:

```
run:
Formatação de número:125.878.945,35
Formatação de moeda:R$ 125.878.945,35
Formatação de porcentagem:12.587.894,53%
```

NumberFormat

- **OBSERVAÇÃO 1:** quando não se passa parâmetros para os métodos da classe NumberFormat, são consideradas as informações de formatação do sistema operacional
- **OBSERVAÇÃO 2:** é possível especificar uma formatação de outra língua para os métodos da classe NumberFormat → utilizar a classe Locale (`java.util.Locale`)



NumberFormat

```
15 public class Teste {  
16     □  
17     public static void main(String[] args) {  
18         □  
19         double x = 125878945.358258;  
20         □  
21         NumberFormat formatter2 = NumberFormat.getCurrencyInstance(Locale.GERMANY);  
22         String s = formatter2.format(x);  
23         System.out.println("Formatação de moeda alemã:" + s);  
24         □  
25         NumberFormat formatter3 = NumberFormat.getCurrencyInstance(Locale.KOREA);  
26         s = formatter3.format(x);  
27         System.out.println("Formatação de moeda japonesa:" + s);  
28         □  
29     }  
30     □  
31 }  
32 }
```

teste.Teste > main > formatter2 >

Saída x

Console do Depurador x Teste (run) x



run:
Formatação de moeda alemã:125.878.945,36 €
Formatação de moeda japonesa:₩125,878,945

Locale

- A biblioteca padrão do Java possui uma classe chamada Locale que contém formatações pré-definidas de números, datas, horas, etc., para várias regiões geográficas, políticas ou culturais
- Um objeto da classe Locale pode ser utilizado em conjunto com objetos formatadores para que possam retornar número formatados no padrão do local especificado
- O construtor mais básico da classe Locale contém um único parâmetro referente a língua e o segundo mais básico permite especificar a língua e o país
- A própria classe Locale contém uma série de campos estáticos contendo as strings correspondentes às línguas



Locale

```
1 | import java.text.DecimalFormat;
2 | import java.text.NumberFormat;
3 | import java.util.Locale;
4 |
5 | public class Programa {
6 |
7 |     public static void main(String[] args){
8 |
9 |         double valor = 2342323.4238492384;
10 |
11 |         Locale local = new Locale("pt");
12 |         NumberFormat formatador = DecimalFormat.getInstance(local);
13 |         System.out.println("Número formatado em pt: " + formatador.format(valor));
14 |
15 |         local = new Locale("en");
16 |         formatador = DecimalFormat.getInstance(local);
17 |         System.out.println("Número formatado em en: " + formatador.format(valor));
18 |
19 |     }
20 |
21 |
22 | }
```

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the provided Java code. The terminal window, titled 'Saída' (Output), displays the following text:

```
run:
Número formatado em pt: 2.342.323,424
Número formatado em en: 2,342,323.424
```

Locale

Criando um local com formatações adotadas no Brasil

```
1  import java.text.DecimalFormat;
2  import java.text.NumberFormat;
3  import java.util.Locale;
4
5  public class Programa {
6
7      public static void main(String[] args){
8
9          double valor = 2342323.4238492384;
10
11         Locale local = new Locale("pt","BR");
12         NumberFormat formatador = DecimalFormat.getCurrencyInstance();
13         System.out.println("Número formatado em pt: " + formatador.format(valor));
14
15         local = new Locale ("en", "US");
16         formatador = DecimalFormat.getCurrencyInstance(local);
17         System.out.println("Número formatado em en: " + formatador.format(valor));
18     }
19
20 }
21 }
```

The screenshot shows an IDE interface with a code editor and a run/test results panel. The code editor contains Java code demonstrating how to create a Locale for Brazil ('pt-BR') and another for the United States ('en-US'). It uses `DecimalFormat` to format a double value ('2342323.4238492384') as currency. The results panel shows two outputs: one for the Brazilian locale (formatting the number as 'R\$ 2.342.323,42') and one for the US locale (formatting it as '\$2,342,323.42').

```
Saída x Resultados da Pesquisa x
Teste (run) x Console do Depurador x Teste (run) #2 x
run:
Número formatado em pt: R$ 2.342.323,42
Número formatado em en: $2,342,323.42
```

Locale

OBSERVAÇÃO: pode-se alterar o Locale padrão da instancia da JVM que está executando seu programa por meio do método estático `setDeault(Locale locale)` da classe Locale

```
2 import java.text.DecimalFormat;
3 import java.util.Locale;
4
5 public class TesteArray {
6
7     public static void main(String[] args) {
8
9         Locale.setDefault(Locale.US);
10
11         double numero = 45645456456.787897897;
12
13         DecimalFormat formatador = new DecimalFormat("###,###.##");
14
15         System.out.println(formatador.format(numero));
16
17     }
18
19 }
```

The screenshot shows an IDE interface with the following details:
1. Top bar: Shows the file name "TesteArray.java" and the method "main".
2. Bottom bar: Shows the output window with the command "run:" and the result "45,645,456,456.79".
3. Status bar: Shows icons for file operations like save, close, and search.

Datas e Formatação de Datas

- Para se trabalhar com datas, pode-se utilizar o objeto Date (pacote `java.util.Date`) ou ainda trabalhar com bibliotecas para lidar com as datas, como a `joda-time` (<http://www.joda.org/joda-time/>)
- Como ainda não aprendemos a utilizar bibliotecas externas, vamos utilizar o objeto Date da biblioteca padrão do Java
- Esse objeto é capaz de armazenar não somente data como também as horas



Classe Date

- Para se criar um objeto da classe Date, pode-se fazer uso dos seguintes construtores:
 - Date()**: o objeto conterá a data e a hora em que ele foi criado
 - Date(int year, int month, int date)**: inicializa a data com o ano, mês e dia do mês (date) informados no construtor
 - Date(int year, int month, int date, int hrs, int min, int sec)**: inicializa o hoje contendo ano, mês, dia do mês, horas, minutos e segundos especificados no construtor
 - Date(long date)**: inicializa um objeto Date informando os milisegundos corridos a parte de 1 de Janeiro de 1970 as 00:00:00:GMT
 - Date(String s)**: converte uma *string* em uma data (desde que seja válida)



Classe Date

- **OBSERVAÇÃO 1:** ao passar o valor inteiro correspondendo ao ano, esse valor será acrescido no ano base que é 1900
- **OBSERVAÇÃO 2:** portanto, se quiser criar uma data com o ano de 2017, deve-se informar o valor 117 no parâmetro correspondente ao ano
- **OBSERVAÇÃO 3:** os inteiros representando os meses começam a partir do 0



Classe Date

```
2  import java.util.Date;
3
4  public class TesteArray {
5
6      public static void main(String[] args) {
7
8          Date data = new Date(118,3,28);
9
10         System.out.println(data);
11     }
12
13
14 }
```

Saída - Teste (run) x Resultados da Pesquisa x

run:
Sat Apr 28 00:00:00 AMT 2018

Formatação de Datas

- Para as datas serem apresentadas para um usuário em uma maneira formatada, isto é, sempre contendo uma mesma quantidade de dígitos para dia, mês e ano, pode-se utilizar um objeto para formatar essa data
- No caso, vamos utilizar a classe `SimpleDateFormat` (`java.text.SimpleDateFormat`) para tal finalidade
- Ao criar o objeto da classe `SimpleDateFormat`, deve-se especificar a “máscara da formatação” em seu construtor

```
SimpleDateFormat formatadorDataHora = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");  
SimpleDateFormat formatadorData = new SimpleDateFormat("MM/dd/yyyy");
```



Formatação de Datas

- Cada letra na formatação corresponde a um tipo (dia, mês, ano, hora, etc..)
- A quantidade de letras repetidas especifica a quantidade mínima de dígitos que deverá ser exibida ao apresentar o tipo
→ *mm* implica que o mês deverá ter no mínimo 2 dígitos
- A exceção se dá apenas para o campo mês
 - **Até dois Ms:** quantidade de dígitos em que o mês (em formato numérico) será apresentado
 - **3 Ms:** apresenta o mês em formato textual mas de forma abreviada
 - **4 ou mais Ms:** apresenta o mês por extenso



Formatação de Datas

- As principais letras utilizadas na formatação da data e hora são:
 - y**: ano
 - M**: mês
 - W**: semana no ano
 - w**: semana no mês
 - D**: dia no ano
 - E**: dia da semana em formato textual
 - H**: hora do dia variando de 0 a 23
 - k**: hora do dia variando de 1 a 24
 - K**: hora do dia variando de 0 a 11
 - h**: hora do dia variando de 1 a 12
 - m**: minutos (em uma determinada hora)
 - s**: segundos (em um determinado minuto)



Formatação de Datas

```
1 import java.text.SimpleDateFormat;
2 import java.util.Date;
3
4 public class Programa {
5
6     public static void main(String[] args){
7
8         Date data = new Date();
9         data.setDate(10);
10        data.setMonth(05);
11        data.setYear(117);
12
13        SimpleDateFormat formatadorDataHora = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
14        SimpleDateFormat formatadorData = new SimpleDateFormat("dd/M/yyyy");
15
16
17        System.out.println("Exibindo a data e a hora formatadas");
18        System.out.println(formatadorDataHora.format(data));
19
20
21        System.out.println("\nExibindo só a data formatada");
22        System.out.println(formatadorData.format(data));
23
24    }
25
26 }
```

The screenshot shows an IDE interface with the following components:

- Editor Area:** Displays the Java code for a `Programa` class.
- Output Area (Saída):** Shows the results of the program's execution. It includes two lines of text:
 - "Exibindo a data e a hora formatadas"
 - "06/10/2017 14:38:17"
- Output Area (Saída) - Second Line:** Shows another line of text:
 - "Exibindo só a data formatada"
 - "10/6/2017"
- Toolbar:** Includes tabs for "Saída", "Resultados da Pesquisa", "Teste (run)", "Console do Depurador", and "Teste (run) #2".
- Bottom Right Corner:** Contains three small circular icons with arrows, likely for navigating between windows or tabs.

Formatação de Datas

```
1 import java.text.SimpleDateFormat;
2 import java.util.Date;
3
4 public class Programa {
5
6     public static void main(String[] args){
7
8         Date data = new Date("10/05/1977");
9
10        SimpleDateFormat formatador = new SimpleDateFormat("ddd-MMMM-yyyy");
11        System.out.println("Exibindo a data: " + formatador.format(data));
12
13        formatador = new SimpleDateFormat("dd-MMM-yyyy");
14        System.out.println("Exibindo a data: " + formatador.format(data));
15    }
16
17
18 }
```

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code above.
- Output Area:** Shows the results of two runs:
 - Run 1: Exibindo a data: 005-Outubro-01977
 - Run 2: Exibindo a data: 05-out-1977
- Toolbars:** Standard Java development toolbar with icons for file operations, search, and run.
- Bottom Status Bar:** Shows the current run context: "Teste (run) #2".

Formatação de Datas

```
1  import java.text.SimpleDateFormat;
2  import java.util.Date;
3
4  public class Programa {
5
6      public static void main(String[] args){
7
8          Date data = new Date();
9          data.setMonth(05);
10         data.setYear(117);
11
12         SimpleDateFormat formatador = new SimpleDateFormat("D");
13         System.out.println("Hoje é o " + formatador.format(data) + "º dia do ano.");
14
15         formatador = new SimpleDateFormat("w");
16         System.out.println("Esta é a " + formatador.format(data) + "ª semana do ano.");
17
18         formatador = new SimpleDateFormat("W");
19         System.out.println("Esta é a " + formatador.format(data) + "ª semana do mês.");
20     }
21
22 }
23
24 }
```

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code above, which uses `SimpleDateFormat` to print the current date and day of the year, week number, and week of the year.
- Output Area:** Shows the results of the code execution:
 - Teste (run) #2
 - run:
 - Hoje é o 168º dia do ano.
 - Esta é a 24ª semana do ano.
 - Esta é a 3ª semana do mês.
- Toolbar:** Includes icons for file operations (New, Open, Save, Print), search, and help.

Datas e Formatação de Datas (2)

- Há algumas classes “novas” na biblioteca padrão do Java 8 que também nos permite trabalhar com datas e horas
- As classes mais comuns da biblioteca padrão do java são:
 - Instant
 - LocalDate
 - LocalTime
 - LocalDateTime
- Os objetos dessas classes são criados por meio de métodos estáticos das próprias classes



Instant

- No Java 8, a classe Instant é utilizada para representar o que antigamente era representado por um long contendo os milissegundos a partir de 00:00:00 do dia 01/01/1970
- Além disso, a classe Instant possui precisão de nanossegundos
- Para criar um objeto Instant contendo o instante de tempo atual, basta utilizar o método estático now() da classe Instant



Instant

- Pode-se utilizar o método `between(...)` para retornar um objeto `Duration` para armazenar a diferença entre dois períodos de tempo
 - Pode-se retornar essa diferença no formato de segundos, milissegundos, nanossegundos, etc., utilizando os respectivos métodos para tal
 - Ex: `toNanos()`, `toMillis()`, `toHours()`, `getSeconds()`;

Instant

```
15 public class TesteTempo {  
16     □  
17     public static void main(String[] args) {  
18         Instant instantInicio = Instant.now();  
19  
20         long soma = 0;  
21         for(int i=0;i<10000000000;i++){  
22             soma += Math.sqrt(i*i + (i - 1));  
23         }  
24         Instant instantFim = Instant.now();  
25  
26  
27         long segundos = Duration.between(instantInicio, instantFim).getSeconds();  
28  
29         System.out.println("Resultado da Soma: " + soma);  
30         System.out.println("Tempo para calcular a Soma: " + segundos);  
31  
32  
33  
34  
35     }  
36  
37 }
```

LocalDate

- A classe LocalDate armazena uma data no formato “humano”, i.e., dias, meses e ano
- A classe LocalDate possui métodos estáticos para a criação de objetos do tipo LocalDate
- Exs:
 - `LocalDate.now()` → retorna a data de hora (data do sistema)
 - `LocalDate.of(2000, 1, 15)` → cria a data do dia 15 de janeiro de 2000
- Pode-se utilizar o objeto Period para obter a diferença entre duas datas
 - Pode-se utilizar esse objeto para retornar a diferença em anos, meses e dias
 - Ex: `getYears()`, `getMonths()` e `getDays()`



LocalDate

```
1 import java.time.LocalDate;
2 import java.time.Period;
3
4
5 public class TesteTempo {
6
7     public static void main(String[] args) {
8
9         LocalDate hoje = LocalDate.now();
10        LocalDate nascimento = LocalDate.of(1985, 8, 5);
11
12        Period periodo = Period.between(nascimento, hoje);
13
14        System.out.println("Diferença em anos: " + periodo.getYears());
15        System.out.println("Diferença em meses: " + periodo.getMonths());
16        System.out.println("Diferença em dias: " + periodo.getDays());
17
18    }
19
20}
21
```

Saída - Teste (run) x Resultados da Pesquisa x ConvertPdfToTxt.java x

run:
Diferença em anos: 32
Diferença em meses: 7
Diferença em dias: 21

LocalDate

- A classe LocalDate provê diversos métodos para retornar informações sobre a data armazenada neste objeto

- **Exs:**

- `getDayOfMonth()` → retorna o dia do mês ([1 - 30 ou 31])
- `getDayOfWeek()` → retorna o dia da semana (DOMINGO, SEGUNDA, ... SÁBADO)
- `getDayOfYear()` → retorna o dia do ano ([1 - 365-366])
- `getMonthValue()` → retorna o mês do ano ([1 - 12])
- `getYear()` → retorna o ano



LocalDate

```
1  import java.time.LocalDate;
2
3  public class TesteTempo {
4
5      public static void main(String[] args) {
6
7          LocalDate hoje = LocalDate.now();
8
9          System.out.println("Dia do mês de hoje: " + hoje.getDayOfMonth());
10         System.out.println("Dia da semana: " + hoje.getDayOfWeek());
11         System.out.println("Dia do ano: " + hoje.getDayOfYear());
12         System.out.println("Mês atual: " + hoje.getMonthValue());
13         System.out.println("Ano atual: " + hoje.getYear());
14         System.out.println("Data: " + hoje.toString());
15     }
16
17 }
18 }
```

The screenshot shows an IDE interface with the code in the editor and its execution results in the terminal window below.

Terminal Output:

```
run:
Dia do mês de hoje: 26
Dia da semana: MONDAY
Dia do ano: 85
Mês atual: 3
Ano atual: 2018
Data: 2018-03-26
```

LocalTime

- A classe LocalTime serve para representar uma horário
- Seu uso é semelhante ao da classe LocalDate
- Exemplos de métodos para gerar um objeto LocalTime
 - now(): criar um horário com a hora corrente no sistema
 - of(int hour, int minute): cria um horário com as horas e minutos informados
 - of(int hour, int minute, int second): cria um horário com as horas, minutos e segundos informados
 - parse(CharSequence text): cria um horário de acordo com a *string* informada



LocalTime

```
1  import java.time.LocalTime;
2
3  public class TesteTempo {
4
5      public static void main(String[] args) {
6
7          LocalTime agora = LocalTime.now();
8          System.out.println(agora.toString());
9
10         LocalTime horario1 = LocalTime.of(20, 11, 55);
11         System.out.println(horario1.toString());
12
13         LocalTime horario2 = LocalTime.parse("10:55");
14         System.out.println(horario2.toString());
15     }
16
17 }
18 }
```

The screenshot shows an IDE interface with the following details:

- Code Area:** Displays the Java code for `TesteTempo` using `LocalTime`.
- Output Area:** Shows the execution results:
 - run: 16:49:58.035
 - 20:11:55
 - 10:55
- Toolbars:** Standard Java development toolbars for file operations.
- Bottom Status Bar:** Shows tabs for "Saída - Teste (run)", "Resultados da Pesquisa", and "ConvertPdfToTxt.java".

LocalDateTime

- Pode-se utilizar a classe LocalDateTime para armazenar uma data e uma hora no mesmo objeto
- Combinação de LocalDate e LocalTime

```
2 import java.time.LocalDateTime;
3
4 public class TesteTempo {
5
6     public static void main(String[] args) {
7
8         LocalDateTime agora = LocalDateTime.now();
9         LocalDateTime amistosoBrasil = LocalDateTime.of(2018, 3, 27, 14, 45, 0);
10
11         System.out.println("Agora: " + agora.toString());
12         System.out.println("Amistoso Brasil x Alemanha: " + amistosoBrasil.toString());
13
14     }
15 }
```

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the provided Java code. The terminal window titled 'Saída - Teste (run)' shows the output of the program: 'Agora: 2018-03-26T16:55:57.320' and 'Amistoso Brasil x Alemanha: 2018-03-27T14:45'. The IDE has tabs for 'Saída - Teste (run)', 'Resultados da Pesquisa', and 'ConvertPdfToTxt.java'.

LocalDateTime

- As classes LocalDate, LocalTime e LocalDateTime possuem métodos para adicionar dias, semanas, horas, minutos, etc.
- Para isso deve-se utilizar os métodos plus e minus

```
2 import java.time.LocalDateTime;
3
4 public class TesteTempo {
5
6     public static void main(String[] args) {
7
8         LocalDateTime agora = LocalDateTime.now();
9         System.out.println("Agora: " + agora.toString());
10        agora = agora.plusWeeks(2);
11        agora = agora.plusHours(5);
12        System.out.println("Agora Modificado (1): " + agora.toString());
13        agora = agora.minusMinutes(30);
14        agora = agora.minusYears(15);
15        System.out.println("Agora Modificado (2): " + agora.toString());
16    }
17 }
18 }
19 }
```

The screenshot shows an IDE interface with three tabs at the bottom: 'Saída - Teste (run)', 'Resultados da Pesquisa', and 'ConvertPdfToTxt.java'. The code in the editor is as shown above. The output window displays the following text:
Agora: 2018-03-26T17:03:11.759
Agora Modificado (1): 2018-04-09T22:03:11.759
Agora Modificado (2): 2003-04-09T21:33:11.759

DateTimeFormatter

- Pode-se utilizar também o objeto **DateTimeFomatter** para formatar datas
- Os valores de formatação são semelhantes ao do **SimpleDateFormat**
- Para se criar um objeto do tipo **DateTimeFormater**, pode-se utilizar os métodos (exemplos)
 - `ofPattern(''dd/MM/yyyy'')`
 - `ofLocalizedDateTime(FormatStyle.SHORT)`



DateTimeFormatter

```
2 import java.time.LocalDate;
3 import java.time.LocalDateTime;
4 import java.time.format.DateTimeFormatter;
5 import java.time.format.FormatStyle;
6
7 public class TesteTempo {
8
9     public static void main(String[] args) {
10
11         LocalDate hoje = LocalDate.now();
12         DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy");
13         System.out.println(hoje.format(formatador)); //08/04/2014
14
15         LocalDateTime agora = LocalDateTime.now();
16         DateTimeFormatter formatador2 = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT);
17         System.out.println(agora.format(formatador2));
18
19     }
20
21 }
22 }
```

Saída - Teste (run) x Resultados da Pesquisa x ConvertPdfToTxt.java x

run:
26/03/2018
26/03/18 17:20

Projeto Banco

- Vamos complementar o exercício realizado na última aula (Aula 6)
- O complemento se dará da seguinte forma:
 - Ao invés de uma única conta, termos um conjuntos de contas bancárias
 - Acrescentar um campo nroConta do tipo String na classe ContaBancaria
 - Fazer as adaptações necessários no Projeto Banco para considerar múltiplas contas (por exemplo, pedir o número da conta antes de fazer uma operação de depósito/saque)
 - Acrescentar a opção Transferência, a qual permitirá transferir dinheiro de uma conta para outra (desde que haja saldo da conta que está realizando a transferência)



Projeto Banco

- O complemento se dará da seguinte forma:
 - Para transferir, deve-se informar o nº da conta de origem, o nº da conta de destino e o valor (validar se as contas de origem e destino existem)
 - Cada operação bancária deverá ser armazenada e deverá aparecer no extrato
 - Armazenar a data da operação (data corrente no sistema)
 - Armazenar o tipo da operação (saque, depósito, transferência de, e transferência para)
 - Armazenar o valor da operação bancária
 - A data deverá ser formatada com dd/MM/yyyy e o valor da operação com o símbolo do Real (R\$) e duas casas decimais



Projeto Banco

- O complemento se dará da seguinte forma:
- Na área gerencial, vamos acrescentar as seguintes opções
 - 1 - Saldos por conta: lista as contas com o número, nome do titular e saldo de todas as contas do sistema
 - 2 - Saldo total dos clientes: apresentar o saldo total de todas as contas do banco



Material Complementar

- Curso de Java #14 - Vetores - Gustavo Guanabara

https://www.youtube.com/watch?v=KAS94-Lcboc&index=26&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR

- Arrays

<http://www.tiexpert.net/programacao/java/array.php>

- Explorando a Classe ArrayList no Java

<http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298>



Material Complementar

- Números Aleatórios em Java – A Classe `java.util.Random`

[http://www.devmedia.com.br/
numeros-aleatorios-em-java-a-classe-javau-util-random/26355](http://www.devmedia.com.br/numeros-aleatorios-em-java-a-classe-javau-util-random/26355)

- Conheça a nova API de datas do Java 8

<http://blog.caelum.com.br/conheca-a-nova-api-de-datas-do-java-8/>

- Como manipular datas com o Java 8

[https:
//www.devmedia.com.br/como-manipular-datas-com-o-javau-8/34135](https://www.devmedia.com.br/como-manipular-datas-com-o-javau-8/34135)

- Class `SimpleDateFormat`

[https:
//docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html](https://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html)



Material Complementar

- Class DecimalFormat

[https:](https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html)

[//docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html](https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html)

- Class NumberFormat

[https:](https://docs.oracle.com/javase/7/docs/api/java/text/NumberFormat.html)

[//docs.oracle.com/javase/7/docs/api/java/text/NumberFormat.html](https://docs.oracle.com/javase/7/docs/api/java/text/NumberFormat.html)

- Class Locale

<https://docs.oracle.com/javase/7/docs/api/java/util/Locale.html>

- DateTimeFormatter

[https://docs.oracle.com/javase/8/docs/api/java/time/format/
DateTimeFormatter.html](https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html)



Imagen do Dia



Programação Orientada a Objetos

<http://lives.ufms.br/moodle/>

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I



Deitel, P. and Deitel, H. (2010).

Java: How to Program.

How to program series. Pearson Prentice Hall, 8th edition.