

---

# Programação Orientada a Objetos

## Elementos básicos de Java

Profa.: Márcia Sampaio Lima

EST - UEA

---

Ref.: Slides Prof. Flávio José Mendes Coelho

---

# Conversões de Valores

---

**Ref.: Slides Prof. Flávio José Mendes Coelho**

---

# Conversão de Valores

- Qual será o valor de **x** após a atribuição?

```
double x;  
float y = 35.23f;  
int z = 10;  
x = y + z;
```

## Conversão de Valores

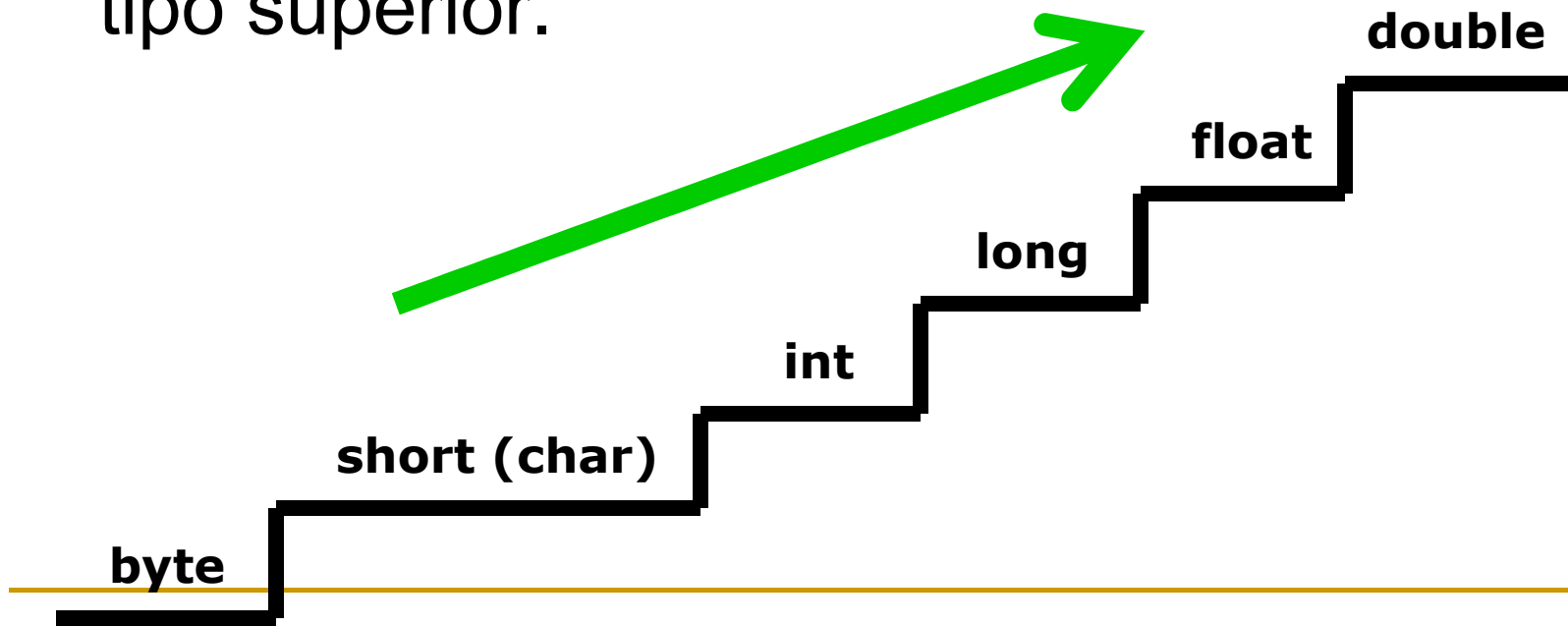
- Qual será o valor de **x** após a atribuição?

```
double x;  
float y = 35.23f;  
int z = 10;  
x = y + z;
```

- **45.23**

# Conversão de Valores

- **Solução:** promoção de tipos java.
- **Promoção:** conversão de um valor de um tipo para um valor correspondente em um tipo superior.



---

## Conversão de Valores

- Agora, qual será o valor de **x** após a atribuição?

```
double a = 10.0;  
float b = 35.23f;  
int c;  
c = a + b;
```

---

## Conversão de Valores

- Agora, qual será o valor de **x** após a atribuição?

```
double a = 10.0;  
float b = 35.23f;  
int c;  
c = a + b;
```

- **Erro: tipos incompatíveis!!**

---

# Conversão de Valores

- **Problema:** não há promoções para baixo!
- **Solução:** fazer um "rebaixamento" forçado chamado coerção explícita ou ***casting***.
- ***Casting***: conversão de um valor de um tipo para um valor correspondente em um tipo inferior.

`(tipo_destino) valor_tipo_origem`

- **tipo\_destino** é o tipo para o qual o **valor\_tipo\_origem** será convertido.
-



---

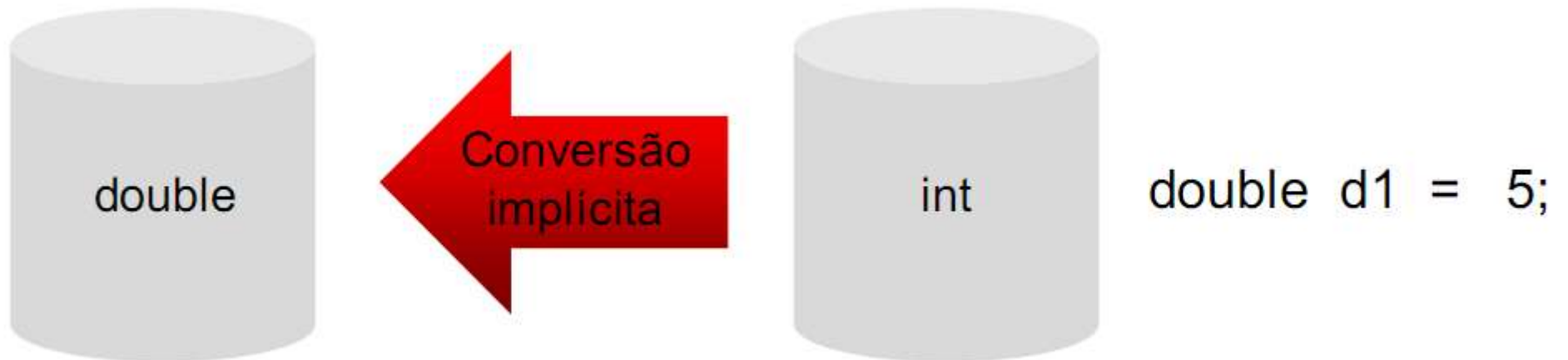
# Conversão de Valores

## Exemplos:

- `double a = 10.0;`  
`float b = 35.23f;`  
`int c = (int) (a + b);`
- `(byte) 345.66577f` **converterá** `345.66577f` para o valor 345.
- *Casting* se aplica a constantes numéricas, variáveis, expressões e resultados de chamadas de métodos.

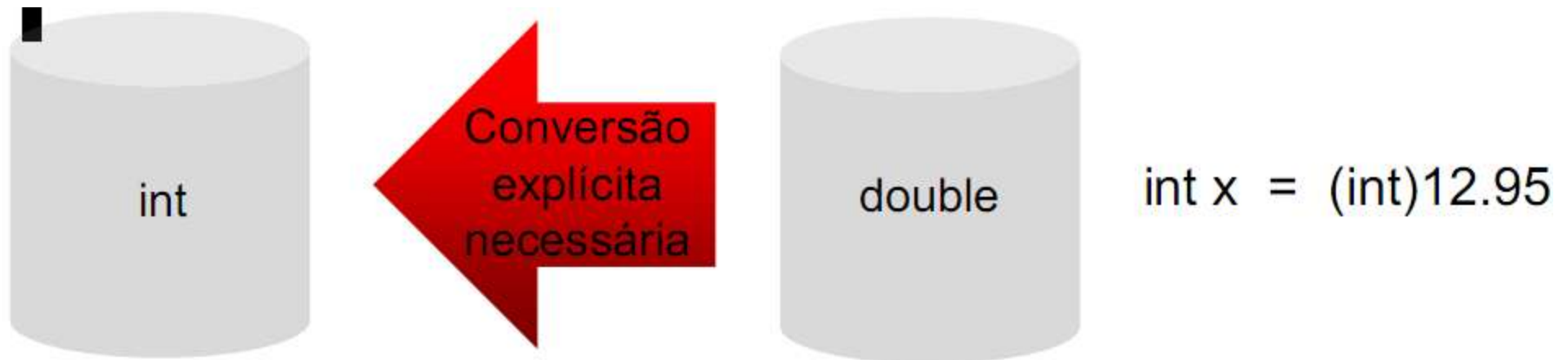
# Conversão de Valores

- Quando o Java está convertendo de um tipo primitivo menor para um tipo primitivo maior, a conversão é implícita.



# Conversão de Valores

- A conversão de um tipo primitivo maior para um tipo primitivo menor deve ser explícita por meio do casting de tipo.
- O Java **não** converterá implicitamente um tipo maior em um tipo menor devido à perda de precisão.

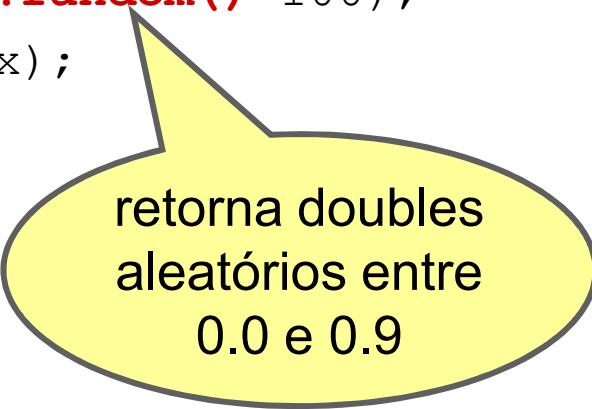


# Conversão de Valores

## Exemplos:

- Gerando números aleatórios de 1 a 100:

```
public class Execucao {  
    public static void main( String[] args ) {  
        int x;  
        for (int i = 0; i < 10; i++) {  
            x = (int) (1 + Math.random() * 100);  
            System.out.println(x);  
        }  
    }  
}
```



retorna doubles  
aleatórios entre  
0.0 e 0.9

# Strings Java

1. Uma String é um objeto que contém uma sequência de caracteres. Declarar e instanciar uma String é muito parecido com qualquer outra variável de objeto.
2. Uma **string** é uma sequência de caracteres delimitada por aspas.
  - Em Java, uma string é um objeto da classe **String**.
  - Formas de instanciación de strings:
    - `String palavras = "Ernesto Nazareth";`
    - `String palavras = new String( "Ernesto Nazareth" );`

---

# Strings Java

1. Uma **string** é uma seqüência de caracteres delimitada por aspas.

```
String s1 = new String("abc"); // new operator  
String s2 = "abc";           // String literals
```

# String - concatenação

- Concatene duas strings com os operadores + e +=.
- O símbolo + é usado para concatenar duas strings.
- O símbolo += é usado para concatenar duas strings e atribuí-las a si mesmas, tudo em uma operação.

```
String s1 = "This is a ";  
String s2 = "string";  
String s3 = s1 + s2;  
String s4 = "This is a " + s2;  
String s1 += s2;
```

# String - concatenação

- Qual será a saída de s1, s2, s3 e s4 e no final destas instruções?

```
String s1 = "This is a ";  
String s2 = "string";  
String s3 = s1 + s2;  
String s4 = "This is a " + s2;  
String s1 += s2;  
System.out.println("s1: " + s1);  
System.out.println("s2: " + s2);  
System.out.println("s3: " + s3);  
System.out.println("s4: " + s4);
```



---

# Seqüência de Escape

- As seqüência de escape em literais de string permitem ao usuário adicionar caracteres que seriam mal interpretados pelo compilador.
  - Por exemplo, para incluir aspas duplas em sua String, o compilador interpretaria as aspas duplas como o início ou o fim da String, em vez de incluí-las.
  - A seqüência de escape \" para incluir aspas duplas.
-

# Sequência de Escape

Sequência de Escape	Representação
\"	Aspas duplas
\'	Aspas simples
\\	Barra invertida
\t	Tabulação horizontal
\n	Alimentação de linha

---

# Comparação de String

- Existem métodos a serem usados ao comparar Strings.
  - **s1.compareTo(s2)**
    - ❑ Retorna um valor inteiro.
    - ❑ Se s1 for menor que s2, um  $\text{int} < 0$  será retornado.
    - ❑ Se s1 for igual a s2, 0 será retornado.
    - ❑ Se s1 for maior que s2, um  $\text{int} > 0$  será retornado.
-

---

# Comparação de String

- Existem métodos a serem usados ao comparar Strings.
  - **s1.equals(s2)**
    - ❑ Deve ser usado quando você só deseja saber se as duas strings são iguais.
    - ❑ Retorna um valor booleano.
    - ❑ Se true for retornado, s1 será igual a s2.
    - ❑ Se false for retornado, s1 não será igual a s2.
-

---

# String - tamanho

- Método Útil de String: `length()`
- **`s1.length()`**
  - Retorna o comprimento ou o número de caracteres em `s1` como um `int`.

```
String s1 = "This is a string.";
int n = s1.length();
//n is 17 because s1 has 17 characters
```

---

---

## Solicitando a Entrada do Usuário: Scanner

- A entrada do teclado usando um Scanner requer a seguinte instrução de importação:  
**java.util.Scanner**
  - A solicitação do usuário pode ser feita com um simples código que aparecerá na tela da console, onde o usuário pode inserir sua entrada.
-

# Solicitando a Entrada do Usuário: Scanner

## ■ Scanner

- ❑ Para ler a entrada que o usuário inseriu, use o Scanner.
- ❑ Para inicializar um Scanner, escreva:

```
Scanner in = new Scanner(System.in);
```

O scanner geralmente é chamado "in" que é a abreviatura de "input" (entrada)

System.in é usado pelo scanner para ler a entrada dos usuários da tela da console.

---

# Solicitando a Entrada do Usuário: Scanner

- O scanner facilita a leitura da entrada do usuário porque ela já tem métodos que executam esta tarefa bem.
- O método do Scanner `next()` lê a entrada do usuário como uma String e retorna essa String.
- Esta linha do código:
  - ❑ Cria uma nova string chamada INPUT.
  - ❑ Digitaliza a string que o usuário informou na console de saída usando o scanner chamado.
  - ❑ Define a entrada igual à string que foi lida pelo scanner.

```
String input = in.next();
```

---



---

# Solicitando a Entrada do Usuário: Scanner

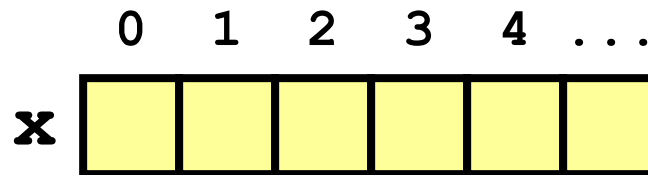
- Método `nextInt()` do Scanner
  - ❑ O método do Scanner `nextInt()` lê a entrada do usuário como um inteiro e retorna esse inteiro.
  - ❑ Esta linha do código cria um novo inteiro chamado `resposta`.
  - ❑ **`int resposta = in.nextInt();`**



# Arrays Java

- Um **array** é um agregado de variáveis todas do mesmo tipo de dado, acessadas por meio de um mesmo nome e diferenciadas por um índice.

- Exemplo:



```
double[] x = new double [10];  
x[0] = 34.65;  
x[9] = 1125.788;  
x[2] = x[0] + x[9];  
System.out.println( x[2] );
```

1160.438

---

# Arrays Java

- No exemplo

```
double[] x = new double [10];
```

x é um array que vai da posição 0 até a 9.

- Cada célula do array (x[0], x[1], ..., x[9]) somente pode armazenar valores do tipo double.

---

# Arrays Java

- De forma geral,

```
Tipo[] nome_array = new Tipo[tamanho];
```

onde:

- *nome\_array*: é o nome do array.
- *Tipo*: pode ser um tipo primitivo Java ou uma classe Java.
- *tamanho*: é a quantidade fixa de células do array. Se o tamanho é  $k$ , o array vai da célula 0 até a célula  $k-1$ .

# Arrays Java

- Instanciando e inicializando um array:

```
String[] bichanos = { "Garfield", "Frajola",  
    "Felix", "Tom", "Branquinho", "Oggy" };
```

**Ou...**

```
String[] bichanos = new String[6];  
bichanos[0] = "Garfield";  
bichanos[1] = "Frajola";  
bichanos[2] = "Felix";  
bichanos[3] = "Tom";  
bichanos[4] = "Branquinho";  
bichanos[5] = "Oggy";
```

---

# Arrays Java

- Mostrando todos os nomes contidos no array **bichanos**:

```
String[] bichanos = { "Garfield", "Frajola",  
    "Felix", "Tom", "Branquinho", "Oggy" };
```

```
for (int i = 0; i < bichanos.length; i++) {  
    System.out.println( bichanos[i] );  
}
```

- `bichanos.length` **armazena o tamanho do array.**

---

# Arrays Java

- O valor do índice de uma posição de um array pode ser:
  - ❑ uma constante numérica inteira (número inteiro);
  - ❑ uma variável inteira;
  - ❑ uma expressão que resulte em um número inteiro;
  - ❑ um atributo inteiro de um objeto;
  - ❑ ou uma chamada a um método que retorne um valor inteiro.

---

# Arrays Java

## ■ Exemplos de uso dos índices:

```
bichanos[3] = "Bechano";  
bichanos[j] = "Chanin";  
bichanos[inicio*3 - fim*2] = nome[j];  
bichanos[pessoa.idade] = "Gatuno";  
bichanos[frutas.obterQtd()] =  
    "Vampirento";
```



---

# Arrays Java

- Todo array Java é um tipo de objeto.
- Assim como ocorre com os atributos de tipos primitivos, as células de um array são inicializadas automaticamente no momento de sua alocação (new).

# Arrays Java

- Instanciando e inicializando um array de objetos:

```
Gato[] bichanosMesmo =  
    { new Gato(), new Gato(), new Gato() };
```

**Ou...**

```
Gato[] bichanosMesmo = new Gato[3];  
bichanosMesmo[0] = new Gato();  
bichanosMesmo[1] = new Gato();  
bichanosMesmo[2] = new Gato();
```

# Arrays Java

## ■ Ou com mais detalhes...

```
Gato[] bichanos = new Gato[3];  
Gato p = new Gato(), k = new Gato(), t =  
    new Gato();  
p.nome = "Meau";  
k.nome = "Chanim";  
t.nome = "Batatinha";  
bichanos[0] = p;  
bichanos[1] = k;  
bichanos[2] = t;
```

---

# Arrays Java

- Mostrando os nomes dos gatos do array **bichanos**:

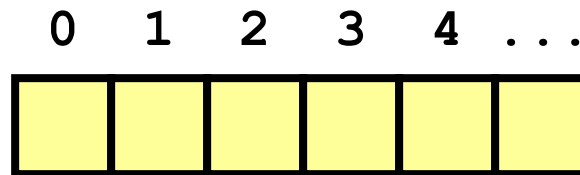
```
for (int i = 0; i < bichanos.length; i++) {  
    Gato g = bichanos[i];  
    System.out.println( g.nome );  
}
```

**Ou...**

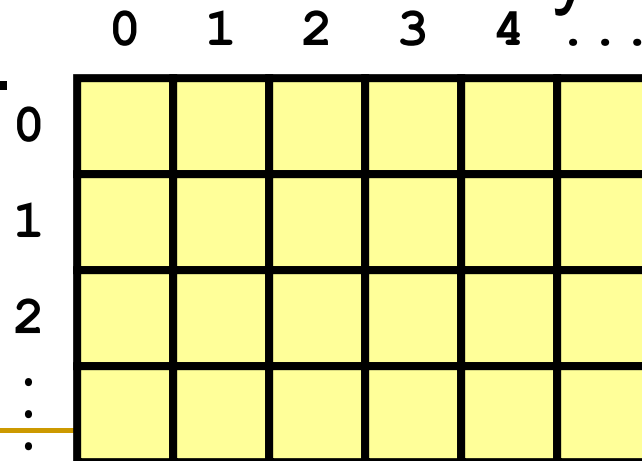
```
for (int i = 0; i < bichanos.length; i++) {  
    System.out.println( bichanos[i].nome );  
}
```

# Arrays Java

- Até agora mostramos arrays unidimensionais (vetores)...

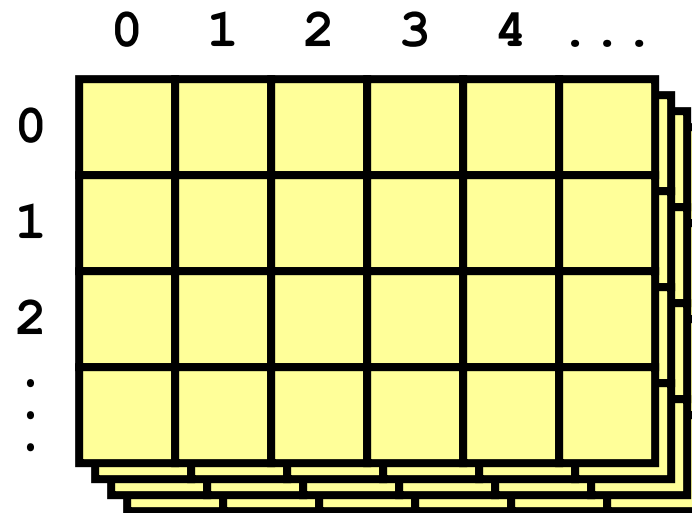


- Mas há também os arrays bidimensionais (matrizes)...



# Arrays Java

- E os arrays tridimensionais (cubos)... mas são menos usados.



- Mais à frente, voltaremos a falar sobre arrays bidimensionais.

---

## Bibliografia

- Fowler, Martin. UML Essencial. 2o. ed. – Porto Alegre: Bookman, 2000.
- Deitel, Harvey. Java - Como Programar. 6a. ed. Porto Alegre: Bookman, 2006.
- Eckel, Bruce. Thinking in Java. 3a. edição. Prentice-Hall, 2001.