



# LINGUAGEM DE PROGRAMAÇÃO JAVA

Cefet – Maracanã -RJ

BCC/TSI

Prof. Gustavo Guedes

E-mail: [gustavo.guedes@cefet-rj.br](mailto:gustavo.guedes@cefet-rj.br)



# Comentários

---

- Comentário de linha

- **Ex:** //variável responsável por definir um nome pa...

- Comentário de bloco

**/\*** Início do comentário

Tudo que está aqui dentro está comentado

**\*/** Fim do comentário



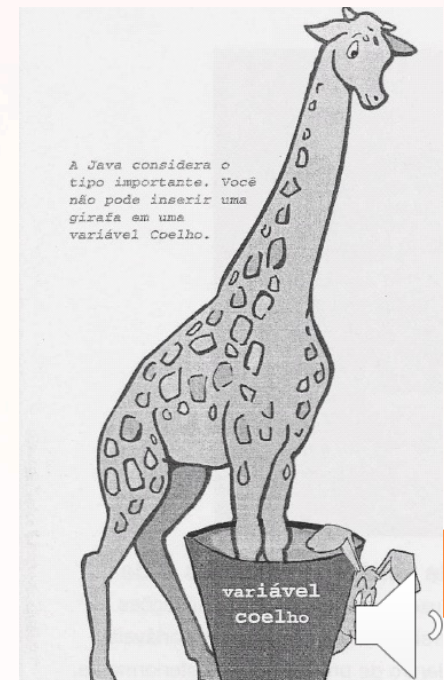
# Variáveis

---

- Existem dois tipos de variáveis: **primitivas** e de **referência**.
- O Java considera o tipo importante. Ele não permitirá que o programador faça algo bizarro e perigoso, como inserir a **referência** a um objeto girafa dentro de uma **variável** Coelho.

- O que aconteceria se mandássemos o suposto coelho saltar?

Coelho saltador = new Girafa(); //Não compila

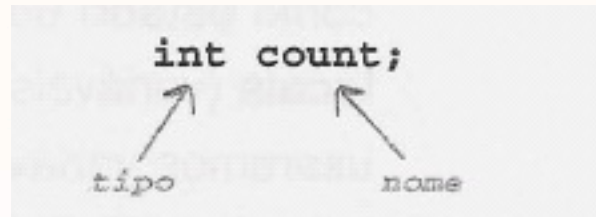


(Fonte: use a cabeça Java)

# Variáveis

---

- Os tipos de dados incorporados à tecnologia Java são classificados em duas categorias amplas – **variáveis de referência** de objeto e **variáveis primitivas**. As variáveis primitivas são valores simples e não objetos. As referências de objeto contêm nada mais que referências a objetos.
- As variáveis **DEVEM** ter um tipo.
- As variáveis **DEVEM** ter um nome.



```
int count;
```

tipo                      nome

(Fonte: use a cabeça Java)



# VISÃO GERAL

---

- Quando pensar em variáveis Java, pense em xícaras.
- Xícaras de café, xícaras de chá, xícaras grandes...  
Uma variável é apenas uma xícara, um contêiner.  
Ela contém algo.



# Declaração de classes

---

Sintaxe básica de uma classe Java:

```
[modificadores] class <nome_da_classe> {  
    [declaração de atributos]  
    [declaração de construtores] //veremos depois  
    [declaração de métodos]  
}
```

Exemplo:

```
public class Veiculo {  
    private double cargaMaxima;  
    public void setMaxLoad(double value){  
        cargaMaxima=value;  
    }  
}
```



# Declaração de atributos

---

Sintaxe básica da declaração de atributos:

[**modificadores**] <tipo> <nome> [= <valor\_inicial>];

Exemplo:

```
public class Veiculo {  
    private int x;  
    private float y = 1000.0F;  
    private String name = "Bates";  
}
```

O valor **private** declara que esse atributo pode ser acessado somente pelos métodos de objetos dessa classe.



# CASTING (MOLDAGEM)

---

- Casting significa atribuir um valor de um tipo a uma variável de outro tipo. Em alguns casos o casting é feito automaticamente. Por exemplo, um valor `int` sempre pode ser atribuído a uma variável `long`.
- Quando houver a possibilidade de perda de informações em uma atribuição, o compilador exige que você confirme a atribuição com um `casting` de tipo. Por exemplo, “espremendo” um valor `long` em uma variável `int`, como em:
  - `long bigValue = 99L;`
  - `int squashed = (int) (bigValue);` //utilizar o valor posterior ao casting entre parênteses é boa prática
- O tipo de destino desejado é colocado entre parênteses e usado como um prefixo para a expressão que deve ser modificada. Embora nem sempre isso seja necessário, é aconselhável colocar toda a expressão a ser convertida entre parênteses. Caso contrário, a precedência da operação de casting pode causar problemas.





# CASTING (MOLDAGEM)

---

- As variáveis podem ser promovidas automaticamente para um formato mais extenso (como por exemplo, de int para long) .
- `int k = 6;`
- `long bigval = k;` //6 é int, OK.
- `int smallval=99L;` //99L é long, ilegal.
- Não existe uma forma explícita de escrever um literal short e byte, portanto, o java faz o casting automático nesse caso.
- No caso de operadores binários, como o +, quando os dois operandos forem de tipos numéricos primitivos, o tipo de resultado será determinado pelo operando de maior tipo, ou int. Dessa forma, todas as operações binárias em tipos numéricos resultam pelo menos em int, e possivelmente um resultado maior se houver operandos float ou double na



# CASTING (MOLDAGEM)

---

- No caso de operadores binários, como o +, quando os dois operandos forem de tipos numéricos primitivos, o tipo de resultado será determinado pelo operando de maior tipo, ou int. Dessa forma, todas as operações binárias em tipos numéricos resultam pelo menos em int, e possivelmente um resultado maior se houver operandos float ou double na expressão.
- OBS: se for uma soma do tipo: `byte b = 15 + 17;` não dá problema porque os literais são verificados.



# CASTING (MOLDAGEM)

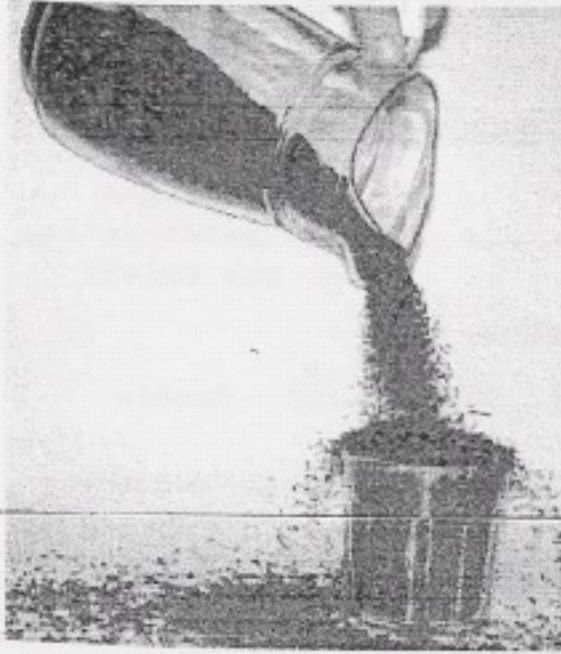
---

- Observe o código abaixo:
- `short a, b, c;`
- `a=1;`
- `b=2;`
- `c=a+b;`
- O código acima causa um erro porque ele promove cada short a um int antes de realizar a operação. Entretanto, se c for declarado como um int, ou se for realizado um casting de tipo:
- `c = (short) (a+b);`
- O código irá funcionar.



# CASTING (MOLDAGEM)

Certifique-se de que o valor cabe na variável.



Você não pode desejar uma quantidade grande em uma xícara pequena.

Bem, certo, você pode, mas vai perder uma parte. Você terá o que chamamos de *derramamento*. O compilador tentará ajudar a impedir isso se conseguir perceber que algo em seu código não caberá no contêiner (variável/xícara) que você está usando.

Por exemplo, você não pode despejar muitos inteiros em um contêiner de tamanho byte, como descrito a seguir:

```
int x = 24;  
byte b = x;  
//não funcionará!!
```

- Qual a razão de não funcionar? 24 é suficiente baixo para caber em b, certo? O que importa ao **compilador** é que houve a tentativa de inserir algo grande em um recipiente pequeno e há possibilidade de derramamento.
- Não espere que o **compilador** saiba qual o valor de X, **mesmo que você esteja vendo**.



# Declaração de condições

---

- As declarações if,else
- A sintaxe básica das declarações if,else é:
  - if (expressão booleana) {
  - instrução ou bloco.
  - }
- if (expressão booleana) {
- instrução ou bloco.
- } else {
- instrução ou bloco.
- }



# Declaração de condições

---

- Segue abaixo um exemplo completo:
- `int count;`
- `count = 4`
- `if (count < 0) {`
- `System.out.println("Erro: valor de COUNT é negativo.");`
- `} else if (count > 10){`
- `System.out.println("Erro: valor de COUNT é muito grande.");`
- `} else {`
- `System.out.println("Haverá" + count + " pessoas almoçando hoje.");`
- `}`
- O Java é diferente das linguagens C e C++ porque uma declaração "if" assume uma expressão booleana e não um valor numérico.



# Declaração de condições

---

- Segue abaixo um exemplo:
- Dessa forma, não podemos ter:
- `if (x) //x é inteiro`
- Devemos utilizar:
- `if (x != 0)`
- Toda parte **else** da declaração é opcional, e você pode omiti-la caso não exista uma ação a ser tomada quando a condição testada for falsa.
- Obs: também temos o switch/case que pode ser estudada paralelamente.



# Declaração de loop

---

- As declarações de **loop** permitem que você execute blocos de declaração repetidamente. O Java suporta três tipos de construções de **loop**: for, while e do while. Os loops for e while testam a condição do **loop** antes de executar o corpo do loop, enquanto que o **loop** do while verifica a condição do loop após executar o corpo do **loop**. Isso quer dizer que os loops for e while podem não executar o corpo do **loop**, enquanto que o loop do while executa o corpo do **loop** pelo menos uma vez.
- A sintaxe do **loop** for é:
- ```
for (init_expr; boolean testexpr; alter_expr) {  
    //instrução ou bloco;  
}
```





# Declaração de loop

---

- Exemplo:
  - `for (int i =0; i < 10; i++) {`
  - `System.out.println(“rodada ” + i);`
  - `}`
  - `System.out.println(“Finalmente!!”);`
- Como o compilador interpreta:
  - - criar uma variável `i` e configurar com 0.
  - - repetir enquanto `i` for menor que 10.
  - - no fim de cada iteração do loop, acrescentar uma unidade a `i`.



# Declaração de loop

---

- A sintaxe do loop while:
- while (boolean){
- //instrução ou bloco;
- }
  
- Exemplo:
- 1 int i = 0;
- 2
- 3 while (i<10) {
- 4   System.out.println("valor de i: " + i);
- 5   i++;
- 6 }
- 7 System.out.println("Finalmente!!");



# Declaração de loop

---

- loop do/while:
- do {
- //instrução ou bloco;
- } while (boolean test);
- Exemplo:
- 1 int i = 0;
- 2
- 3 do {
- 4   System.out.println("Já terminou?");
- 5 i++;
- 6 } while (i<10);
- 7 System.out.println("Finalmente!!");



# CONTROLE DE FLUXO ESPECIAL PARA LOOPS

---

- Cuidados com pré-incremento e pós-incremento:

- Exemplo 1:

- `int i = 5;`

- `int x = i++;`

- Exemplo 2:

- `int i = 5;`

- `int x = ++i;`

- Qual a diferença?



# CONTROLE DE FLUXO ESPECIAL PARA LOOPS

---

- `break;`
- Use a declaração `break` para sair prematuramente de declarações `switch` e de declarações de `loop`.
- `continue;`
- Use a declaração `continue` para saltar para o final do corpo do `loop` e retornar o controle para o início do `loop`.



# Escopo das variáveis

---

- O escopo da variável é o nome dado ao trecho de código onde ela existe e pode ser acessada.
- Quando abrimos um novo bloco com as chaves e declaramos uma variável ali dentro, significa que ela só vale até o fim daquele bloco.
- *//Aqui a variável x não existe.*
- `int x = 10;`
- *//Aqui x já existe.*
- `while (condição){`
- *//O x ainda vale aqui. O y ainda não existe.*
- `int y = 5;`
- *//O y passa a existir.*
- `}`
- *// O y não existe mais. O x continua valendo.*

