
"CADA SONHO QUE VOCÊ DEIXA PARA TRÁS, É UM PEDAÇO DO SEU FUTURO QUE DEIXA DE EXISTIR."
- STEVE JOBS -


# Desenvolvimento de Aplicações Desktop

*Operadores Lógicos; Estruturas de Controle (Decisão)*

*Professor: Charles Leite*

# Fluxo de Controle

A ordem com que as instruções são executadas em um programa é chamada de FLUXO DE CONTROLE

Exceto quando especificado de outra forma, a execução de um programa acontece de forma linear

- As instruções são executadas uma após outra, em sequência, do início ao final de um programa

# Fluxo de Controle

## Em Java, uma aplicação:

- Inicia sua execução na primeira linha do método **`main`**
- Continua sua execução, passo a passo, com as próximas linhas do método **`main`**
- Finaliza sua execução na última linha do método **`main`**

# Fluxo de Controle

**Porém, quando um método é invocado, o fluxo de controle é alterado**

- O controle é desviado para o código definido pelo método
- Quando o método é finalizado, o controle retorna para o ponto onde o método foi chamado, continuando o processamento a partir desse ponto

**Dentro de um método, o fluxo de controle pode ser alterado por certos tipos de comandos**

- Comandos de decisão (condicionais)
- Comandos de repetição (*loops*)

# Fluxo de Controle

Esses comandos permitem decidir que instruções devem ser executadas

Cada decisão é tomada com base na avaliação de **EXPRESSÕES BOOLEANAS**

# Expressões Booleanas

- Expressões booleanas retornam, como resultado da sua avaliação, valores do tipo **`boolean`**
  - **`true`** e **`false`**

- Existem diversos grupos de operadores que, quando usados em expressões, geram valores booleanos após a avaliação dessas expressões
  - Operadores lógicos
  - Operadores relacionais
  - Operadores de igualdade

# Operadores Lógicos

- São aplicados sobre operandos booleanos em expressões, retornando, como resultado, um valor booleano

- Operadores lógicos:
  - `x && y` → Operador de conjunção – *AND (E)*
  - `x || y` → Operador de disjunção – *OR (OU)*
  - `!y` → Operador de negação – *NOT (NÃO)*

# Operadores Lógicos

- Numa expressão, para esses operadores, os operandos são avaliados na medida em que as condições são satisfeitas
  - Caso uma condição seja contrariada, dependendo do operador, a segunda condição pode não ser avaliada (avaliação em curto-circuito)
- Porém, existem situações nas quais os dois operandos precisam ser avaliados
  - Nesse caso, os seguintes operadores lógicos podem ser usados:
    - `x & y` → Operador lógico de conjunção – *AND*
    - `x | y` → Operador inclusivo lógico de disjunção – *OR*

# Operadores Lógicos

- Ordem de precedência (prioridade) e associatividade:

| Operador | Associação |
|---|---|
| ! | Da direita para a esquerda |
| ^ | Da esquerda para a direita |
| && | |
| \|\| | |

# Operadores Lógicos

- Tabela verdade (Matemática Discreta) para os operadores lógicos, considerando todas as combinações possíveis de valores **`true`** e **`false`**:

| a | b | !a | a && b | a \|\| b | a ^ b |
|---|---|---|---|---|---|
| true | true | | | | |
| true | false | | | | |
| false | true | | | | |
| false | false | | | | |

# Operadores Lógicos

- Exemplo:

```
{   ...
    boolean b, c;
    b = true;
    c = !b;
    c = !(true || b) && c;
    b = c || !(!b);
    ...
}
```

Parênteses são usados para evitar ambigüidades

Qual o valor de **b** neste ponto?

# Operadores Relacionais

- Usados em expressões booleanas, permitindo fazer comparações entre valores, retorna, como resultado, um valor booleano

- Operadores relacionais:
  - `x < y` → Menor que
  - `x <= y` → Menor que ou igual a
  - `x > y` → Maior que
  - `x >= y` → Maior que ou igual a

# Operadores Relacionais

Em suma, para expressões booleanas envolvendo operadores relacionais:

**x** e **y** são expressões de algum tipo numérico

As expressões resultantes são do tipo **boolean**, gerando **true** ou **false** como resultado da avaliação

# Operadores de Igualdade

- Operadores de igualdade testam se dois valores são iguais ou não

- Operadores de igualdade:
  - `x == y` → Operador de igualdade
  - `x != y` → Operador de diferença ou desigualdade

# Operadores de Igualdade

Em suma, para expressões booleanas envolvendo operadores de igualdade:

x e y são expressões do mesmo tipo, para qualquer tipo!

As expressões resultantes são do tipo **boolean**, gerando **true** ou **false** como resultado da avaliação

# Operadores de Igualdade

```
x == y
```

- Para avaliar a expressão acima:
  - Avalia-se `x`, primeiro, obtendo um valor
  - Avalia-se `y`, em seguida, obtendo outro valor
  - Compara-se os dois valores obtidos e retorna **`true`**, caso os dois sejam a mesma REFERÊNCIA para um OBJETO OU um mesmo ELEMENTO de um TIPO PRIMITIVO

# Operadores de Igualdade

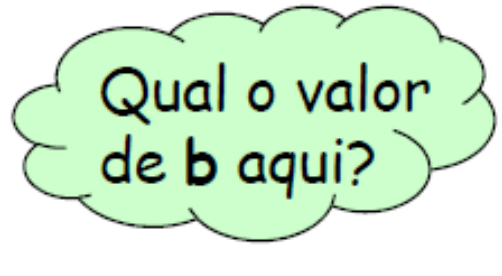
- Exemplo:

```
boolean b, c;

b = true || false;

c = true && b;

b = b == c;
```



# Operadores de Igualdade

- Exemplo:

```
boolean b;
Conta c, d;
c = new Conta();
d = null;
b = c == d;
d = new Conta();
b = c == d;
c = d;
b = c == d;
```

Qual o valor de **b** após cada uma das atribuições?

# Equivalência entre Objetos

- E caso queiramos saber se dois objetos são equivalentes, ou seja, se os valores de seus atributos são os mesmos?
- Não podemos, neste caso, usar o operador de igualdade. Por que?
- Solução:

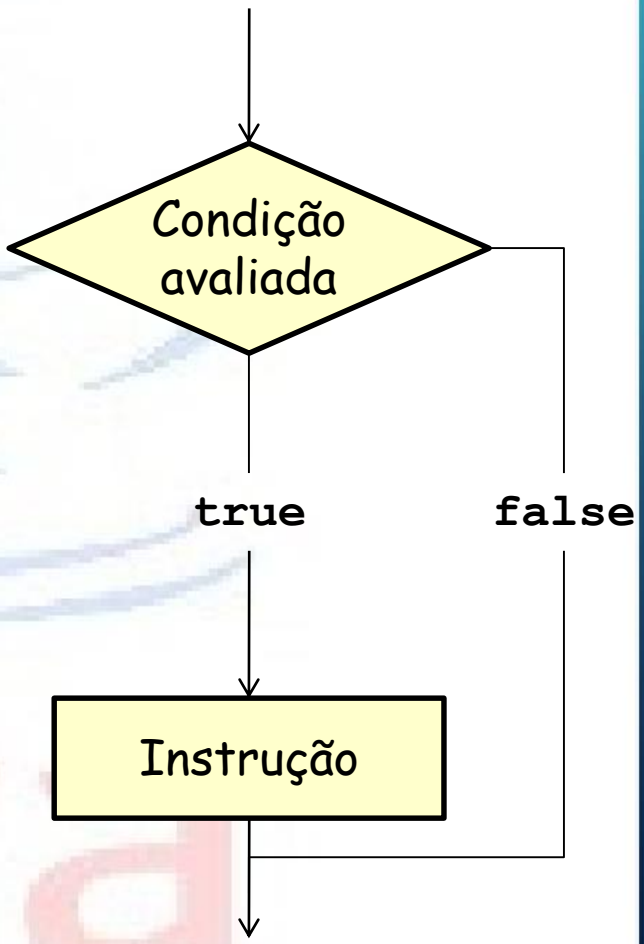
O método
**`equals()`**

# Comandos Condicionais

- Um COMANDO CONDICIONAL nos permite escolher qual será a próxima instrução a ser executada
- A execução de uma determinada instrução depende de uma condição
  - Resultado de uma expressão booleana
- Comandos condicionais presentes em Java:
  - **`if`**
  - **`if-else`**
  - **`switch`**

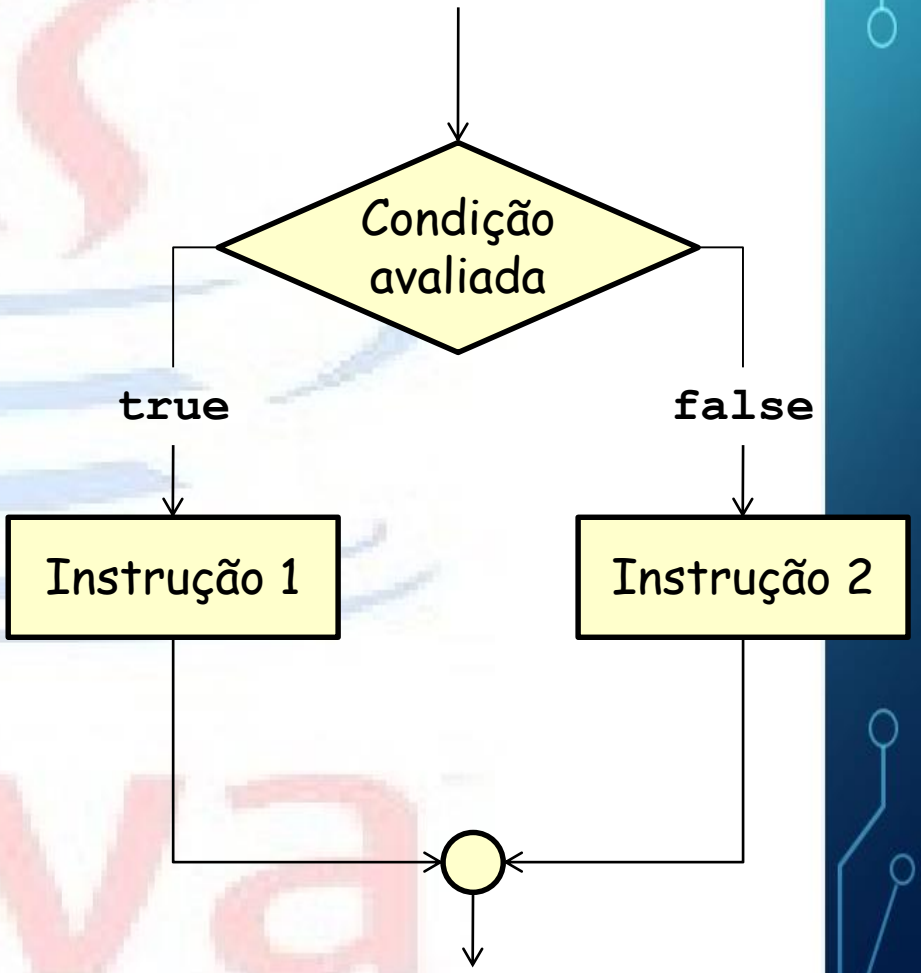
# O Comando `if`

- Consiste na palavra reservada **`if`**, seguida de uma expressão booleana (condição) e de uma ou mais instruções
  - A condição deve ser delimitada por parênteses e sua avaliação gera um valor **`true`** ou **`false`**
  - Se o valor da condição for **`true`**, a(s) instrução(ões) é(são) executada(s)
  - Caso contrário, a(s) instrução(ões) é(são) ignorada(s) e o processamento continua com a próxima instrução



# O Comando `if-else`

- Variação do comando **`if`**, com a adição da cláusula **`else`**, indicando que uma ou mais instruções devem ser executadas quando o valor da expressão booleana for **`false`**



# Comandos `if` e `if-else` com Blocos de Instruções

- Às vezes, várias instruções precisam ser executadas, após avaliação de uma expressão booleana, em um comando condicional
- Nesse caso, o bloco de instruções deve ser delimitado por chaves "{", "}"

```
if (expressaoBooleana) {
     instrucoes
}
```

```
if (expressaoBooleana) {
     instrucoes
} else {
     outrasInstrucoes
}
```

# Comandos **`if`** e **`if-else`** Aninhados

- Às vezes, a instrução executada, como resultado de um comando **`if`** (**`else`**), pode ser um outro comando **`if`**
  - Isto permite tomar outras decisões após determinar os resultados de uma decisão anterior

```
if (expressaoBooleana1) {
      instrucoes1
} else if (expressaoBooleana2) {
      instrucoes2
} else {
      instrucoes3
}
```

# O Comando **switch**

- Para executar um comando **switch**:
  - Avalia-se `expressão`
  - Executa-se os comandos **case** cujo **rótulo** seja igual ao valor resultante da expressão
  - Executa-se os comandos **default**, caso o valor resultante não seja igual a nenhum **rótulo**

```
switch (expressao) {
      case rotulo1:
            comandos1
            break;
      case rotulo2:
            comandos2
            break;
      ...
      default:
            comandos
}
```

# O Comando **switch**

- O tipo de `expressão` só pode ser **`char`**, **`byte`**, **`short`** ou **`int`**
- Os rótulos são constantes diferentes
- Existe, no máximo, uma cláusula **`default`** (que é opcional)
- Os tipos dos rótulos têm que ser o mesmo tipo de `expressão`

```
switch (expressao)
{
      case rotulo1:
            comandos1
            break;
      case rotulo2:
            comandos2
            break;
      ...
      default:
            comandos
}
```

# O Comando **switch**

- Vários rótulos podem estar associados ao mesmo comando
- Os comandos **break** são opcionais
  - Sem o **break**, a execução dos comandos de um rótulo continua nos comandos do próximo rótulo

```
switch (expressao) {
      case r1:
            comandos1
            break;
      case r2: case r3:
            comandos2
            break;
      ...
      default:
            comandos
}
```

# DÚVIDAS ...

# Referências

DEITEL, Harvey. **Java Como Programar.** *Capítulo 2, páginas 81-108.* 8ª edição. São Paulo: Pearson Prentice Hall, 2010.