

## Aula 4

# Introduções de Controle e Operadores

# Introdução

- Qualquer problema de computação pode ser resolvido executando uma série de ações em uma ordem específica
- Então, basicamente temos:
  - 1 Ações a executar
  - 2 Ordem em que as ações devem ser executada
- Instruções em um programa são executadas uma após a outra na ordem em que são escrita (**execução sequencial** ou **execução linear**)

# Introdução

- Várias instruções podem especificar a próxima instrução a executar → **instruções de controle**
- Especificar a ordem em que as instruções (ações) são executadas em um programa é chamado **controle de programa** ou **transferência de controle**

# Introdução

- Para especificar **quais conjuntos de instruções serão executadas**, qual a **ordem em que as instruções serão executadas** ou ainda fazer com que um **conjuntos de instrução sejam executados repetidamente**, pode-se basicamente utilizar:
  - **Instruções de Seleção**
  - **Instruções de Repetição**

# Instruções de Seleção

- **Java contém três tipos de instruções de seleção**
  - 1 **if**: realiza uma ação se uma condição for verdadeira ou pula a ação se a condição for falsa
  - 2 **if...else**: realiza uma ação se uma condição for verdadeira e realiza uma condição diferente se a ação for falsa
  - 3 **switch**: realiza uma de muitas ações diferentes, dependendo do valor de uma expressão
- Java também oferece um operador de seleção (`?:`) que é “semelhante” ao comando `if...else`

## A instrução de seleção única `if`

- A instrução `if` é uma **instrução de controle de uma única entrada e uma única saída**
- Se houver **uma única instrução a ser executada** caso a condição do `if` seja verdadeira, o comando `if` pode ser executado da seguinte forma:

```
14 public class Teste {  
15  
16  
17  
18  
19     public static void main(String[] args) {  
20         int idade = 70;  
21         if(idade > 65)  
22             System.out.println("O que que há velhinho?");  
23     }  
24 }
```

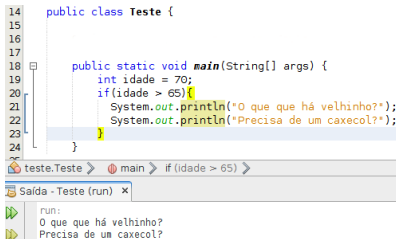
teste.Teste > main >

Saída - Teste (run) x

run:  
O que que há velhinho?  
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)

## A instrução de seleção única if

- Se houver **mais de uma instrução a ser executada** caso a condição do if seja verdadeira, todos os comando a serem executados devem estar contidos entre {...} (chamado de bloco)



```
14 public class Teste {  
15  
16  
17  
18     public static void main(String[] args) {  
19         int idade = 70;  
20         if(idade > 65){  
21             System.out.println("O que que há velhinho?");  
22             System.out.println("Precisa de um caxecol?");  
23         }  
24     }  
25 }
```

teste.Teste > main > if (idade > 65) >

Salida - Teste (run) x

run:  
O que que há velhinho?  
Precisa de um caxecol?

- OBSERVAÇÃO:** mesmo se houver uma única instrução a ser executada, pode-se colocá-la entre {...}

## A instrução de seleção dupla `if...else`

- A instrução `if` de seleção única realiza uma ação indicada somente quando a condição é verdadeira (`true`); caso contrário, a ação é pulada
- A instrução `if...else` permite **especificar uma ação a realizar quando a condição é verdadeira (`true`) e uma ação diferente quando a condição é falsa (`false`)**



## A instrução de seleção dupla if...else

- Caso haja **apenas uma instrução a ser executada** dentro do if ou else, pode-se programar o comando if...else da seguinte forma:

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int idade = 60;  
18         if(idade > 65)  
19             System.out.println("O que que há velhinho?");  
20         else  
21             System.out.println("Tá na flor da idade!");  
22     }  
23  
24 }  
25
```

teste.Teste > main >

Saída - Teste (run) x

run:  
Tá na flor da idade!

# A instrução de seleção dupla if...else

- Se houver **mais de uma instrução a ser executada** dentro do if ou else, as instruções devem estar contidas dentro de {...}

```
14 public class Teste {
15
16     public static void main(String[] args) {
17         int idade = 60;
18         if(idade > 65){
19             System.out.println("O que que há velhinho?");
20             System.out.println("Quer ajuda pra atravessar a rua?");
21         }else{
22             System.out.println("Tá na flor da idade!");
23             System.out.println("Acabou de sair da fralda.");
24         }
25     }
26 }
```

teste.Teste > main > if (idade > 65) else >

Saída - Teste (run) x

run:  
Tá na flor da idade!  
Acabou de sair da fralda.

## Instruções if...else aninhadas

- Pode-se utilizar instruções if...else **aninhadas**

```
14 public class Teste {
15
16     public static void main(String[] args) {
17         int idade = 65;
18         if(idade < 18){
19             System.out.println("Jovem");
20         }else{
21             if(idade < 60){
22                 System.out.println("Adulto");
23             }else{
24                 System.out.println("Idoso");
25             }
26         }
27     }
28 }
29 }
```

teste.Teste > main > idade >

Saída - Teste (run) x

run:  
Idoso

# Instruções if...else aninhadas

- Ou ainda...

```
16 public static void main(String[] args) {  
17     int idade = 15;  
18     if(idade < 10){  
19         System.out.println("Criança");  
20     }else if(idade < 18){  
21         System.out.println("Adolescente");  
22     }else if(idade < 60){  
23         System.out.println("Adulto");  
24     }else{  
25         System.out.println("Idoso");  
26     }  
27 }  
28  
29 }  
30
```

teste.Teste > main > idade >

Safda - Teste (run) x

run:  
Adolescente

## Operador Condicional (?:)

- O Java fornece o operador condicional (?:) que pode ser utilizado no lugar de uma instrução if...else
- Esse é o **único operador ternário do Java** (operador que recebe três operandos)
  - **O primeiro operando** (à esquerda do ?): é um teste lógico (avalia se a condição é verdadeira ou falsa)
  - **O segundo operando** (entre o ? e :): é o valor da expressão condicional se o teste lógico for verdadeiro
  - **O terceiro operando** (à direita do :): é o valor da expressão condicional se a expressão booleana for falsa

# Operador Condicional (?:)

```
1 import java.util.Scanner;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         double nota = 5.0;
8         String situacao = ((nota>=5)? "Aprovado": "Reprovado");
9         System.out.println(situacao);
10    }
11 }
12
13 }
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Aprovado

# Operador Condicional (?:)

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int idade = 15;  
18         System.out.println(idade>60?"veíhno":"novinho");  
19     }  
20  
21 }  
22
```

teste.Teste > main >

Saída - Teste (run) x

run:  
novinho

## A instrução de seleção múltipla switch

- A instrução de seleção múltipla switch realiza ações diferentes com base nos possíveis valores para uma **expressão integral constante**
- A instrução switch irá executar os comandos abaixo de um “caso” que foi atendido
- Por isso que é necessário utilizar o comando break para “brecar” a execução das instruções



# A instrução de seleção múltipla `switch`

- Após encontrar o comando `break`, o programa continuará executando instruções após a instrução `switch`
- Pode utilizar um caso `default` para disparar instruções caso nenhum caso anterior tenha sido atendido
- A instrução `switch` pode analisar caso de dado do tipo `char`, `byte`, `short`, `int` e da classe `String`

# A instrução de seleção múltipla switch

```
1 import java.util.Scanner;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         int diaDaSemana;
8         Scanner teclado = new Scanner(System.in);
9         System.out.print("Digite um dia da semana [1-7]: ");
10        diaDaSemana = teclado.nextInt();
11
12        switch(diaDaSemana){
13            case 1:
14                System.out.println("Domingo :)");
15                break;
16            case 2:
17                System.out.println("Segunda-feira :~");
18                break;
19            case 3:
20                System.out.println("Terça-feira :(");
21                break;
22            case 4:
23                System.out.println("Quarta-feira :/");
24                break;
25            case 5:
26                System.out.println("Quinta-feira :|");
27                break;
28            case 6:
29                System.out.println("Sexta-feira :)");
30                break;
31            case 7:
32                System.out.println("Sábado :D");
33                break;
34            default:
35                System.out.println("Qual parte do digite um número de 1 a 7 você não entendeu?");
36        }
37    }
38 }
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Digite um dia da semana [1-7]: 2  
Segunda-feira :~

```
10 public class Teste2 {
11
12     public static void main(String[] args){
13
14         String operacao = "-";
15
16         switch(operacao){
17             case "+":
18                 System.out.println("Soma");
19                 break;
20             case "-":
21                 System.out.println("Subtração");
22                 break;
23             case "*":
24                 System.out.println("Multiplicação");
25                 break;
26             case "/":
27                 System.out.println("Divisão");
28                 break;
29             default:
30                 System.out.println("Operador inválido");
31         }
32     }
33 }
34
```

teste.Teste2 > main >

Salda x

Console do Depurador x    Teste (run) x

run:  
Subtração

# Instruções de Repetição

- Uma instrução de repetição (ou *loop*) permite especificar que um programa deve **repetir uma ação enquanto alguma condição permanece verdadeira**
- O Java fornece três **instruções de repetição** (também chamadas **instruções de loop**)
  - 1 **while** ou **for**: realizam a ação (ou grupo de ações) no seu corpo zero ou mais vezes
  - 2 **do...while**: realiza uma ação (ou grupo de ações) no seu corpo uma ou mais vezes

# A Instrução de Repetição while

- A instrução `while` executará uma ação ou uma série de ações desde que uma condição seja verdadeira
- A instrução `while` também é conhecida como **estrutura de repetição com teste lógico no início**

## A Instrução de Repetição while

- Caso haja **apenas uma instrução a ser executada** dentro da repetição, pode-se programar o comando while da seguinte forma:

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int contador = 0;  
18         while(contador < 10)  
19             contador++;  
20     }  
21  
22 }
```

## A Instrução de Repetição while

- Caso haja **mais de uma instrução a ser executada** dentro da repetição, pode-se programar o comando `while` da seguinte forma:

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int contador = 0;  
18         while(contador < 10){  
19             System.out.print(contador + " ");  
20             contador++;  
21         }  
22         System.out.println();  
23     }  
24  
25 }
```

teste.Teste > main >

Saída - Teste (run) x

run:  
0 1 2 3 4 5 6 7 8 9

## Instrução de Repetição do...while

- A instrução de repetição do...while é semelhante à instrução while
- Entretanto, na instrução do...while o teste de continuidade da repetição é feita depois de executar as instruções no corpo da estrutura de repetição → **o corpo sempre é executado ao menos uma vez**
- Essa instrução é conhecida como **while de ponta cabeça** ou **estrutura de repetição com teste no final**



## Instrução de Repetição do...while

- Caso haja **apenas uma instrução a ser executada** dentro da repetição, pode-se programar o comando do...while da seguinte forma:

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         int contador = 0;  
6         do  
7             System.out.println("Contador: " + contador++);  
8         while(contador <=10);  
9  
10    }  
11 }  
12
```

Salda x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Contador: 0  
Contador: 1  
Contador: 2  
Contador: 3  
Contador: 4  
Contador: 5  
Contador: 6  
Contador: 7  
Contador: 8  
Contador: 9  
Contador: 10

## Instrução de Repetição do...while

- Caso haja **mais de uma instrução a ser executada** dentro da repetição, pode-se programar o comando do...while da seguinte forma:

```
12 public class Teste2 {  
13  
14     public static void main(String[] args){  
15  
16         int contador = 0;  
17         do{  
18             System.out.println(contador);  
19             contador++;  
20         }while(contador <= 10);  
21     }  
22 }  
23  
24
```

teste.Teste2 > main > do ... while (contador <= 10)

Saída x

Console do Depurador x Teste (run) x

run:  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10

## A Instrução de Repetição for

- O Java também fornece a **instrução de repetição for**, que **especifica os detalhes da repetição controlada por contador em uma única linha de código**
- Portanto, na própria instrução já são declaradas a **variável de controle** e o **valor inicial desta variável**, o **teste lógico** e o **“passo”** (valor utilizado para modificar a variável de controle)
- Enquanto a condição de avaliação da variável contadora for verdadeira, o conteúdo da instrução for será executado
- Se a condição de continuação do *loop* for inicialmente false, o programa não executará o corpo da instrução for

## A Instrução de Repetição for

- No caso de haver uma única instrução a ser executada na repetição, a instrução for fica da seguinte forma:

```
14 public static void main(String[] args){  
15  
16     int numRepeticoes = 12;  
17  
18     for(int x=0;x<numRepeticoes;x++)  
19         System.out.print(x + " ");  
20  
21     System.out.println();  
22 }  
23  
24 }  
25
```

teste.Teste2 > main >

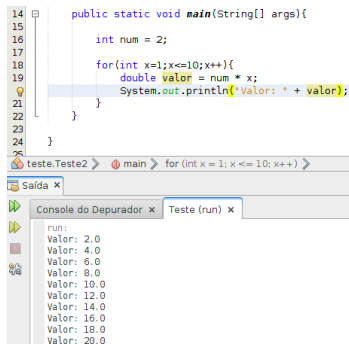
Saída x

Console do Depurador x Teste (run) x

run:  
0 1 2 3 4 5 6 7 8 9 10 11

## A Instrução de Repetição for

- No caso de haver várias instruções a serem executadas na repetição, a instrução for fica da seguinte forma:



```
14 public static void main(String[] args){
15
16     int num = 2;
17
18     for(int x=1;x<=10;x++){
19         double valor = num * x;
20         System.out.println("Valor: " + valor);
21     }
22 }
23
24 }
```

teste.Teste2 > main > for (int x = 1; x <= 10; x++) >

Salida x

Console do Depurador x    Teste (run) x

run:  
Valor: 2.0  
Valor: 4.0  
Valor: 6.0  
Valor: 8.0  
Valor: 10.0  
Valor: 12.0  
Valor: 14.0  
Valor: 16.0  
Valor: 18.0  
Valor: 20.0

## Funcionamento da Instrução for

- **OBSERVAÇÃO 1:** a inicialização, condição de continuação de *loop* e parte de incremento de uma variável na instrução `for` podem conter expressões aritméticas

```
for(int j=x; j<=4*x*y; j+=y/x)
```

- **OBSERVAÇÃO 2:** o incremento de uma instrução `for` também pode ser negativo (decremento)

```
for(int i=100; i>-100; i--)
```

# Instruções de Repetição Aninhadas

- Vale ressaltar que as instruções podem estar aninhadas, isto é, uma instrução de repetição pode estar contida dentro de outra instrução de repetição

```
1 import java.util.Scanner;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         System.out.println("Fazer tabuada de 1 até n");
8         System.out.print("Digite o valor de n: ");
9
10        Scanner teclado = new Scanner(System.in);
11        int n = teclado.nextInt();
12
13        for(int i=1;i<=n;i++){
14            System.out.print("Tabuada do " + i + " : ");
15            for(int j=1;j<=10;j++){
16                System.out.print(i + " x " + j + " = " + (i*j) + " ");
17            }
18            System.out.println("");
19        }
20    }
21 }
22 }
```

Saída X Resultados da Pesquisa X

Console do Depurador X Teste (run) X

run:  
Fazer tabuada de 1 até n  
Digite o valor de n: 5  
Tabuada do 1: 1 x 1 = 1; 1 x 2 = 2; 1 x 3 = 3; 1 x 4 = 4; 1 x 5 = 5; 1 x 6 = 6; 1 x 7 = 7; 1 x 8 = 8; 1 x 9 = 9; 1 x 10 = 10;  
Tabuada do 2: 2 x 1 = 2; 2 x 2 = 4; 2 x 3 = 6; 2 x 4 = 8; 2 x 5 = 10; 2 x 6 = 12; 2 x 7 = 14; 2 x 8 = 16; 2 x 9 = 18; 2 x 10 = 20;  
Tabuada do 3: 3 x 1 = 3; 3 x 2 = 6; 3 x 3 = 9; 3 x 4 = 12; 3 x 5 = 15; 3 x 6 = 18; 3 x 7 = 21; 3 x 8 = 24; 3 x 9 = 27; 3 x 10 = 30;  
Tabuada do 4: 4 x 1 = 4; 4 x 2 = 8; 4 x 3 = 12; 4 x 4 = 16; 4 x 5 = 20; 4 x 6 = 24; 4 x 7 = 28; 4 x 8 = 32; 4 x 9 = 36; 4 x 10 = 40;  
Tabuada do 5: 5 x 1 = 5; 5 x 2 = 10; 5 x 3 = 15; 5 x 4 = 20; 5 x 5 = 25; 5 x 6 = 30; 5 x 7 = 35; 5 x 8 = 40; 5 x 9 = 45; 5 x 10 = 50;

# Instruções de Repetição Aninhadas

```
1 import java.util.Scanner;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         Scanner teclado = new Scanner(System.in);
8         boolean sair = false;
9         int op;
10        while(sair == false){
11            System.out.println("Digite 1 para imprimir uma sequência de '*' ou outro valor para sair: ");
12            op = teclado.nextInt();
13            if(op == 1){
14                for(int i=0;i<30;i++){
15                    System.out.print("*");
16                }
17                System.out.println("");
18            }else{
19                sair = true;
20            }
21        }
22    }
23 }
24
25 >
```

Salda x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

```
run:
Digite 1 para imprimir uma sequência de '*' ou outro valor para sair:
1
*****
Digite 1 para imprimir uma sequência de '*' ou outro valor para sair:
1
*****
Digite 1 para imprimir uma sequência de '*' ou outro valor para sair:
1
*****
Digite 1 para imprimir uma sequência de '*' ou outro valor para sair:
5
```



## Operadores de Atribuição Composta

- Os operadores de atribuição composta abreviam expressões de atribuição

### Operação de atribuição tradicional

`variável = variável [operador] expressão`

- Ex:** `a = a * 2;`

### Operação de atribuição composta

`variável operador= expressão`

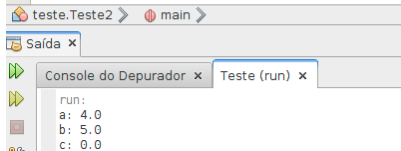
- Ex:** `a *= 2;`

# Operadores de Atribuição Composta

Operador de atribuição	Expressão de Exemplo	Expressão Correspondente
<code>+=</code>	<code>c += 7</code>	<code>c = c + 7</code>
<code>-=</code>	<code>d -= 7</code>	<code>d = d - 7</code>
<code>*=</code>	<code>e *= 5</code>	<code>e = e * 5</code>
<code>/=</code>	<code>f /= 3</code>	<code>f = f / 3</code>
<code>%=</code>	<code>g %= 9</code>	<code>g = g % 9</code>

# Operadores de Atribuição Composta

```
10 public class Teste2 {  
11  
12     public static void main(String[] args){  
13  
14         double a = 2;  
15         a *= 2;  
16         double b = 10;  
17         b /= 2;  
18         double c = 20;  
19         c %= 5;  
20         System.out.println("a: " + a);  
21         System.out.println("b: " + b);  
22         System.out.println("c: " + c);  
23     }  
24 }  
25 }
```



## Operadores de Incremento e Decremento

- O Java fornece **operadores unários** para **adicionar 1** ou para **subtrair 1** do valor de uma **variável numérica**:
  - **Operador de incremento** unário ++
  - **Operador de decremento** unário --

## Operadores de Incremento e Decremento

- Um operador de incremento ou de decremento que é colocado **antes de uma variável** é chamado de **operador de pré-incremento** ou **operador de pré-decremento**, respectivamente
- Um operador de incremento ou de decremento que é colocado **depois de uma variável** é chamado de **operador de pós-incremento** ou **operador de pós-decremento**, respectivamente

# Operadores de Incremento e Decremento

Operador	Nome do Operador	Expressão de Exemplo	Explicação
++	pré-incremento	++a	Incrementa <u>a</u> por 1 e então utiliza o novo valor de <u>a</u> na expressão em que <u>a</u> se encontra
++	pós-incremento	a++	Utiliza o valor atual de <u>a</u> na expressão em que <u>a</u> se encontra e então incrementa <u>a</u> por 1
--	pré-decremento	--b	Decrementa <u>b</u> por 1 e então utiliza o novo valor de <u>b</u> na expressão em que <u>b</u> reside
--	pós-decremento	b--	Utiliza o valor atual de <u>b</u> na expressão em que <u>b</u> se encontra e então decrementa <u>b</u> por 1

## Operadores de Incremento e Decremento

- Exemplo da utilização do operador de pós-incremento

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int contador = 0;  
18         System.out.println(contador++);  
19         System.out.println(contador);  
20     }  
21  
22 }  
23
```

teste.Teste > main >

Saída - Teste (run) x

run:  
0  
1

## Operadores de Incremento e Decremento

- Exemplo da utilização do operador de pré-incremento

```
14 public class Teste {  
15  
16     public static void main(String[] args) {  
17         int contador = 0;  
18         System.out.println(++contador);  
19         System.out.println(contador);  
20     }  
21  
22 }  
23
```

teste.Teste > main >

Saída - Teste (run) x

run:  
1  
1



## Instruções break e continue

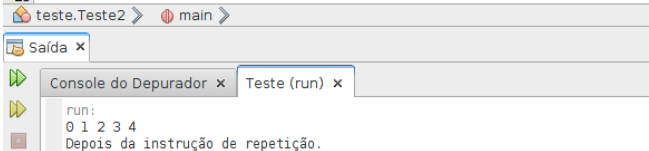
- Além das instruções de seleção e repetição, o Java oferece as instruções `break` e `continue`
- Já vimos o uso do `break` na instrução `switch` → interromper o fluxo de execução de instruções dentro de um bloco
- As instruções `break` e `continue` também **podem ser usadas dentro de instruções de repetição para alterar o fluxo de execução dessas instruções**

## Break

- A instrução `break`, quando executada em um `while`, `for`, `do...while` ou `switch`, **ocasiona a saída imediata dessas instruções**
- A execução continua com a primeira instrução depois da instrução de controle

# Break

```
12 public class Teste2 {  
13  
14     public static void main(String[] args){  
15         for(int cont=0;cont<10;cont++){  
16             if(cont==5){  
17                 break;  
18             }  
19             System.out.print(cont + " ");  
20         }  
21         System.out.println("\nDepois da instrução de repetição.");  
22     }  
23  
24 }  
25
```



## Continue

- A instrução **continue** quando executada em um **while**, **for** ou **do...while** **pula as instruções restantes no corpo do loop e prossegue com a próxima iteração do loop**
- Nas instruções **while** e **do...while**, o programa avalia o teste de continuação do *loop* imediatamente depois que a instrução **continue** é executada
- Em uma instrução **for**, a expressão incremento é executada e então o programa avalia o teste de continuação do *loop*

# Continue

```
12 public class Teste2 {
13
14     public static void main(String[] args){
15         for(int cont=0;cont<10;cont++){
16             if(cont==5){
17                 continue;
18             }
19             System.out.print(cont + " ");
20         }
21         System.out.println("\nDepois da instrução de repetição.");
22     }
23
24 }
25
```

teste.Teste2 > main > for (int cont = 0; cont < 10; cont++) > if (cont == 5) >

Salida x

Console do Depurador x Teste (run) x

run:  
0 1 2 3 4 6 7 8 9  
Depois da instrução de repetição.

## Substituição das Funções break e continue

- Vale ressaltar que as instruções break e continue são apenas uma **facilidade de programação**
- Seu uso pode ser substituído por instruções de seleção e manipulação das variáveis de controle

# Substituição das Funções break e continue

## Exemplo de substituição da função break

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         int contador = 0;
6         while(contador <= 10){
7             if(contador == 5){
8                 break;
9             }
10            System.out.println("Contador: " + contador);
11            contador++;
12        }
13    }
14 }
15
16
```

Salida x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Contador: 0  
Contador: 1  
Contador: 2  
Contador: 3  
Contador: 4

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         int contador = 0;
6         while(contador <= 10){
7             if(contador != 5){
8                 System.out.println("Contador: " + contador);
9             }else{
10                contador = 10;
11            }
12            contador++;
13        }
14    }
15 }
16
```

Salida x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Contador: 0  
Contador: 1  
Contador: 2  
Contador: 3  
Contador: 4

## Operadores Lógicos

- Considerando as instruções de seleção apresentadas até agora, todas irão determinar quais instruções serão realizadas de acordo com o resultado de um teste lógico
- Porém, pode-se utilizar uma composição de testes lógicos para determinar qual ação deverá ser executada
- Para considerar o resultados de vários testes lógicos, deve-se fazer uso dos **operadores lógicos**



## Operadores Lógicos

- As instruções if, if...else, while, do...while e for exigem uma condição para determinar como continuar o fluxo de um programa
- **Até agora só estudamos as condições simples**, expressas em termos dos operadores relacionais <, >, >= e dos operadores de igualdade == e !=
- Em um teste lógico considerando condições simples, apenas uma única condição é avaliada

# Operadores Lógicos

- Para avaliar mais de uma condição pode-se utilizar:
  - ① Instruções if ou if...else aninhadas ou sequências dessas instruções
  - ② **Operadores lógicos**
- Vale ressaltar que ao utilizar operadores lógicos, **condições complexas podem compor um único teste lógico** e portanto, o número de linhas de código é reduzido

# Operadores Lógicos

- **Os operadores lógicos do Java são:**
  - **&&**: E condicional
  - **||**: OU condicional
  - **&**: E lógico booleana
  - **|**: OU inclusivo lógico booleano
  - **^**: OU exclusivo lógico booleano
  - **!**: NÃO lógico

## Operador E condicional (&&)

- O operador E condicional && irá retornar o valor true se a expressão a direita e a expressão a esquerda forem ambas verdadeiras (true)
- O operador E condicional (&&) pode ser usado da seguinte forma:

```
if(sexo.equals("Feminino") && idade>=65);  
    contadorMulheresIdosas++;
```

- Exemplos de avaliação de expressões utilizando os operadores &&

Condição 1	Condição 2	Condição 1 && Condição 2
false	false	false
false	true	false
true	false	false
true	true	true

- OBSERVAÇÃO:** se a primeira expressão for falsa, o operador && não irá avaliar a segunda expressão

## Operador OU condicional (||)

- O operador || irá retornar o valor true se ao menos uma das expressões (à direita) ou à esquerda, forem verdadeiras
- O operador OU condicional (||) é utilizado da seguinte forma:

```
if((mediaFinal >= 7) || (recuperacao >= 7));  
    System.out.println("Passou");
```

## Operador OU condicional (||)

- Exemplos de avaliação de expressões utilizando os operadores ||

Condição 1	Condição 2	Condição 1    Condição 2
false	false	false
false	true	true
true	false	true
true	true	true

- OBSERVAÇÃO:** se a primeira expressão for verdadeira, o operador || não irá avaliar a segunda expressão

## Operadores lógicos booleanos E (&) e OU Inclusivo (|)

- Os operadores lógicos booleanos E (&) e OU inclusivo (|) são idênticos aos operadores && e ||, exceto que os operadores & e | sempre avaliam seus dois operandos
- Isso é útil se o operando à direita do operador lógico booleano E ou do operador lógico booleano OU inclusive tiverem um “efeito colateral” requerido (ex: uma mudança no valor de uma variável)



## Operadores lógicos booleanos E (&) e OU Inclusivo (|)

- Exemplo com E lógico (&&):

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         int idade = 15;
6
7         if((idade >= 16) && (++idade <= 65)){
8             System.out.println("Tá aqui!");
9         }else{
10             System.out.println("Não tá aqui!");
11         }
12         System.out.println("Idade: " + idade);
13     }
14 }
15
16
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Não tá aqui!  
Idade: 15

## Operadores lógicos booleanos E (&) e OU Inclusivo (|)

- Exemplo com E inclusivo (&)

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         int idade = 15;  
6  
7         if((idade >= 16) & (++idade <= 65)){  
8             System.out.println("Tá aqui!");  
9         }else{  
10            System.out.println("Não tá aqui!");  
11        }  
12        System.out.println("Idade: " + idade);  
13    }  
14 }  
15  
16
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Não tá aqui!  
Idade: 16

## OU exclusivo lógico booleano (^)

- Uma condição simples que contém o operador OU exclusivo lógico booleano (^) é true se e somente se uma de suas condições for verdadeira e a outra for false
- Se ambas as condições forem verdadeiras ou ambas falsas, o operador OU exclusivo retornará false
- O operador OU exclusivo (^) pode ser utilizado da seguinte forma:

## OU exclusivo lógico booleano (^)

- Exemplos de avaliação de expressões utilizando o operador (^)

Condição 1	Condição 2	Condição 1 ^ Condição 2
false	false	false
false	true	true
true	false	true
true	true	false

## OU exclusivo lógico booleano (^)

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         double salario = 2500;  
6         boolean aposentado = true;  
7  
8         if((salario >= 2000) ^ (aposentado == true)){  
9             System.out.println("Verdadeiro");  
10        }else{  
11            System.out.println("False");  
12        }  
13    }  
14 }  
15  
16
```

Console do Depurador x Teste (run) x

run:  
False

## OU exclusivo lógico booleano (^)

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         double salario = 2500;
6         boolean aposentado = true;
7
8         if((salario >= 2000) ^ (aposentado == false)){
9             System.out.println("Verdadeiro");
10        }else{
11            System.out.println("Falso");
12        }
13    }
14 }
15
16
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Verdadeiro

## Operador de Negação Lógica (!)

- O operador ! (**NÃO lógico**, também chamado **negação lógico** ou **complemento lógico**) “inverte” o resultado de uma condição ou teste lógico
- Diferentemente dos operadores lógicos &&, ||, & e ^, que são operadores binários que combinam duas condições, o operador de negação lógica é um **operador unário** que tem apenas uma única condição como um único operando
- O operador lógico é colocado antes de uma condição

## Operador de Negação Lógica (!)

- Exemplos de avaliação de expressões utilizando o operador (!)

Condição	! Condição
false	true
true	false



## Operador de Negação Lógica (!)

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         String nome1 = "Rafael";  
6         String nome2 = "Ricardo";  
7  
8         if(!nome1.equals(nome2)){  
9             System.out.println("Os nomes são diferentes");  
10        }  
11    }  
12 }  
13  
14  
15
```

Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:  
Os nomes são diferentes

## Extra: Precedência e Associatividade de Operadores

- Quanto mais para cima, maior a precedência

Operadores	Associatividade	Tipo
++ --	Da direita para a esquerda	Unário pós-fixado
++ -- + - ! (tipo)	Da direita para a esquerda	Unário pré-fixado
* / %	Da esquerda para a direita	Multiplicativo
+ -	Da esquerda para a direita	Aditivo
< <= > >=	Da esquerda para a direita	Relacional
== !=	Da esquerda para a direita	Igualdade
&	Da esquerda para a direita	E lógico booleano
^	Da esquerda para a direita	OU lógico booleano exclusivo
	Da esquerda para a direita	OU lógico booleano inclusivo
&&	Da esquerda para a direita	E condicional
	Da esquerda para a direita	OU condicional
?:	Da direita para a esquerda	Ternário condicional
= += -= /= %=	Da direita para a esquerda	Atribuição

# 1º Quiz

Enzo fez o seguinte código:

```
public class TesteSalario{  
    public static void main(String[] args){  
        boolean foiPromovido = true;  
        if(foiPromovido){  
            double salario = 4200.0;  
        }else{  
            double salario = 3800.0;  
        }  
        System.out.println(salario);  
    }  
}
```

Qual será o resultado da compilação e execução do código?

- a) Será impresso 4200.0
- b) Será impresso 3800.0
- c) O código não compila pois o teste lógico dentro da instrução if - else não é válida
- d) O código não compila pois a variável salário não existe fora da estrutura if - else
- e) Nenhuma das anteriores

## 2º Quiz

Sobre a instrução `break`, é correto afirmar:

- a) Finaliza a execução do programa no momento em que é chamada.
- b) Interrompe a execução do laço mais interno que contém o comando `break` e continua executando os laços mais externos.
- c) Para a execução de todos os laços de repetição aninhados nos quais está inserido, mas não interrompe a execução do programa.
- d) Nenhuma das anteriores.

## 2º Quiz

Sobre a instrução break, é correto afirmar:

- a) Finaliza a execução do programa no momento em que é chamada.
- b) Interrompe a execução do laço mais interno que contém o comando break e continua executando os laços mais externos.**
- c) Para a execução de todos os laços de repetição aninhados nos quais está inserido, mas não interrompe a execução do programa.
- d) Nenhuma das anteriores.

## Exercício

- Vamos complementar o exercício realizado na última aula (Aula 3)
- O complemento se dará por meio da criação de um menu que conterá as opções:
  - Criar conta
  - Sacar
  - Depositar
  - Extrato
  - Sair

## Exercício

- **OBSERVAÇÃO 1:** só poderão ser realizadas operações de saque e depósito se uma conta for criada
- **OBSERVAÇÃO 2:** se o usuário já tiver criado uma conta e for selecionada a 1ª opção (Criar conta), a nova conta deverá substituir a conta anterior
- **OBSERVAÇÃO 3:** o menu deve ser apresentado a cada operação que o usuário fizer, exceto quando for acionada a 5ª opção (Sair)

## Material Complementar

- Curso de Java #09 - Estruturas Condicionais (Parte 1)

[https://www.youtube.com/watch?v=wW3eve4vTMc&list=PLHz\\_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=17](https://www.youtube.com/watch?v=wW3eve4vTMc&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=17)

- Curso de Java #10 - Estruturas Condicionais (Parte 2)

[https://www.youtube.com/watch?v=oNSrBld06qs&list=PLHz\\_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=19](https://www.youtube.com/watch?v=oNSrBld06qs&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=19)



## Material Complementar

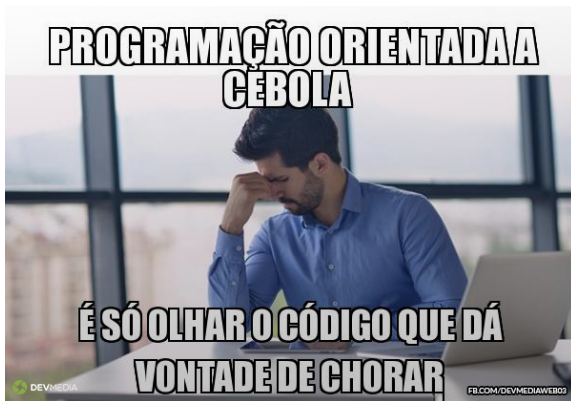
- Curso de Java #11 - Estruturas de Repetição (Parte 1)  
[https://www.youtube.com/watch?v=2fawKjR8d4c&list=PLHz\\_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=20](https://www.youtube.com/watch?v=2fawKjR8d4c&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=20)
- Curso de Java #12 - Estruturas de Repetição (Parte 2)  
[https://www.youtube.com/watch?v=ojLALwmvQIU&index=22&list=PLHz\\_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR](https://www.youtube.com/watch?v=ojLALwmvQIU&index=22&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR)

# Material Complementar

- Control Flow Statements

[https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/questions\\_flow.html](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/QandE/questions_flow.html)

## Imagem 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.