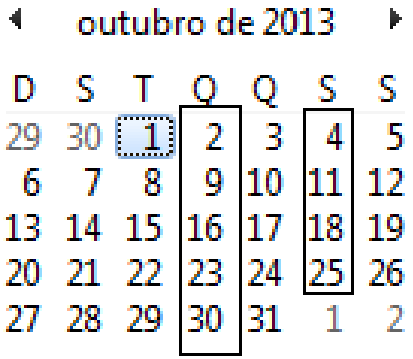
Linguagem de Programação Variáveis, Condição e Operadores Lógicos

Jose.wellington@uniceub.br

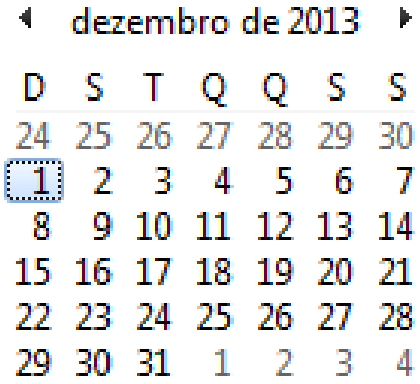
Calendário


agosto de 2013
D S T Q Q S S
28 29 30 31 1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31


setembro de 2013
D S T Q Q S S
25 26 27 28 29 30 31
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 1 2 3 4 5


outubro de 2013
D S T Q Q S S
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2


novembro de 2013
D S T Q Q S S
27 28 29 30 31 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30


dezembro de 2013
D S T Q Q S S
24 25 26 27 28 29 30
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 1 2 3 4

Agenda

- Palavras Reservadas
- Operações Lógicas
- IF e ELSE
- Operadores Aritméticos
- While – Repetição
- Do while – Repetição
- Exercício

Palavras Reservadas

Palavras reservadas

abstract	do	implements	private	throw
boolean	double	import	protected	throws
break	else	instanceof	public	transient
byte	extends	int	return	true
case	false	interface	short	try
catch	final	long	static	void
char	finally	native	super	volatile
class	float	new	switch	while
continue	for	null	synchronized	
default	if	package	this	

- Nota: atualmente as seguintes palavras reservadas não são utilizadas: cast, const, future, generic, goto, inner, operator, outer, rest, var.

Operações Lógicas

Operadores lógicos

- == -> igual a
- != -> diferente de
- > -> maior que
- < -> menor que
- >= -> maior ou igual a
- <= -> menor ou igual a

Fazendo comparações

Maior que: >
a > b -> retorna 'true' caso 'a' seja maior que 'b', e 'false' caso seja menor

Menor que: <
a < b -> retorna 'true' caso 'a' seja menor que 'b', e 'false' caso seja maior

Fazendo comparações

Maior ou igual a: >=

a >= b -> retorna 'true' caso 'a' seja maior ou igual à 'b', e 'false' caso seja menor

Menor ou igual a: <=

a <= b -> retorna 'true' caso 'a' seja menor ou igual à 'b', e 'false' caso seja maior

Fazendo comparações

Comparação de igualdade: ==
a == b -> retorna 'true' caso 'a' seja igual a b, e 'false' caso contrário

Comparação de negação: !=
a != b -> retorna 'true' caso 'a' seja diferente de b, e 'false' caso contrário

Operadores lógicos e de negação

Operadores lógicos

! -> não

&& -> operador lógico “e”

|| -> operador lógico “ou”

Operadores lógicos

O primeiro exemplo é representado por: (condicao_A **&&** condicao_B)

(condicao_A AND condicao_B)

Fazendo comparações

Exemplo é representado por: (condicao_A || condicao_B)

(condicao_A OR condicao_B)

Negando declarações: !

true = !false

false= !true

IF e ELSE

IF

- Em inglês quer dizer 'se'.

```
if ( condição ){
    caso a condição seja verdadeira
    esse bloco de código será executado
}
```

IF

```
public class Ifelse {
    public static void main(String[] args) {
        if (1 == 2){
            System.out.println("Você nunca lerá essa mensagem");
        }

        if (1 == 1){
            System.out.println("1 é igual a 1 ");
        }
    }
}
```

ELSE

Do inglês: senão

```
if ( condição ){
    caso a condição seja verdadeira
    esse bloco de código será executado
} else {
    caso a condição seja falsa
    esse bloco de código que será executado
}
```

ELSE

```
if( nota >= 7.0 ){
        System.out.println("Parabéns, você passou direto”);}
else {
        System.out.println("Não passou direto");}
```

Operadores Aritméticos

Operadores Aritméticos

Operador	*Uso*	*Descrição*
`+`	`op1 + op2`	Retorna a **soma** de `op1` e `op2`.
`-`	`op1 - op2`	Retorna a **subtração** de `op1` por `op2`.
`*`	`op1 * op2`	Retorna a **multiplicação** de `op1` por `op2`.
`/`	`op1 / op2`	Retorna a **divisão** de `op1` por `op2`.
`%`	`op1 % op2`	Retorna o **resto** da divisão de `op1` por `op2`.

```
if (count % 2 == 0)
```

Operadores Aritméticos

Operador	*Uso*	*Descrição*
`++`	`op++`	incrementa de 1.
`--`	`op--`	decrementa de 1.

While - Repetição

Demonstrações de repetição

- *Instruções de repetição permitem executar uma instrução várias vezes*
- *Muitas vezes eles são referidos como loop*
- *Como declarações condicionais, eles são controlados por expressões booleanas*
- *Algumas tipos de instruções de repetição em Java:*
  - while loop
  - do loop

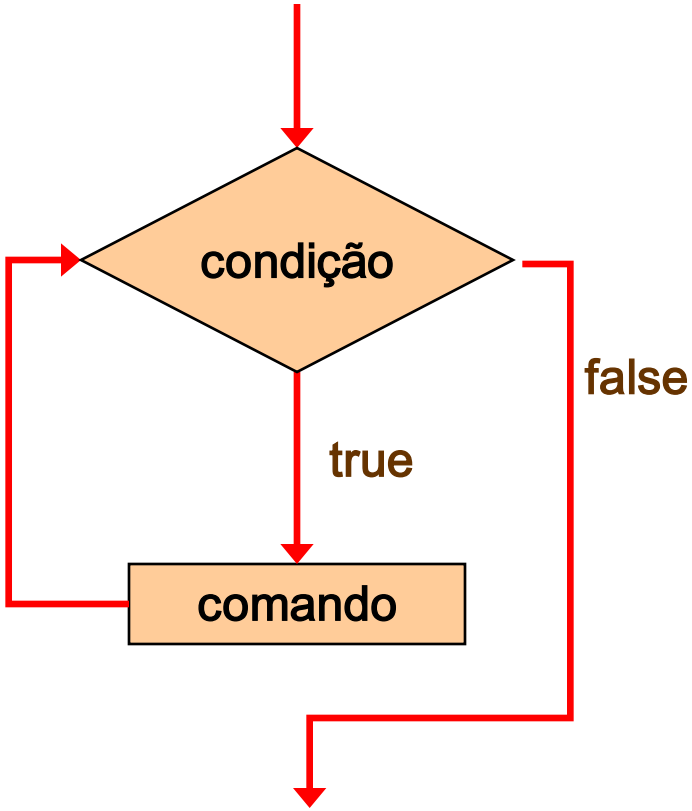
A instrução while

- syntax:

```
while ( condition ){
    statement;
}
```

- **Se a condição for verdadeira, a instrução é executada**
- **Em seguida, a condição é avaliada novamente, e se ainda é verdade, a instrução é executada novamente**
- **O comando é executado repetidamente até que a condição se torna falsa**

Lógica do while Loop



A instrução while

- Exemplo

```
int count = 1;
while (count <= 5){
    System.out.println (count);
    count++;
}
```

- **Se a condição de um loop while é falso, inicialmente, a declaração nunca é executado**
- **Portanto, o corpo de comando executará zero ou mais vezes**

Infinito Loops

- Se não, ele é chamado de um loop infinito, que será executado até que o usuário interrompe o programa
- Este é um erro comum lógica (semântica)
- Você deve sempre checar a lógica de um programa para garantir que seus loops terminará normalmente

Infinito Loops

- Exemplo

```
int count = 1;
while (count <= 25){
   System.out.println (count);
   count = count - 1;
}
```

- **Este loop vai continuar a executar até ser interrompido (Control-C) ou até que ocorra um erro de overflow**

Loops aninhados

- Semelhante ao IF alinhado, loops podem ser aninhado.
- Ou seja, o corpo de um loop pode conter um outro loop.
- Para cada iteração do loop, o loop interno itera completamente
- Seu projeto segundo curso envolve um loop while aninhada dentro de um loop

Loops aninhados

- Quantas vezes a string “Oi" ser impresso?

```
count1 = 1;
while (count1 <= 10){
    count2 = 1;
    while (count2 <= 20)    {
        System.out.println (“Oi");
        count2=count2+1;
    }
    count1=count1+1;
}
```

10 * 20 = 200

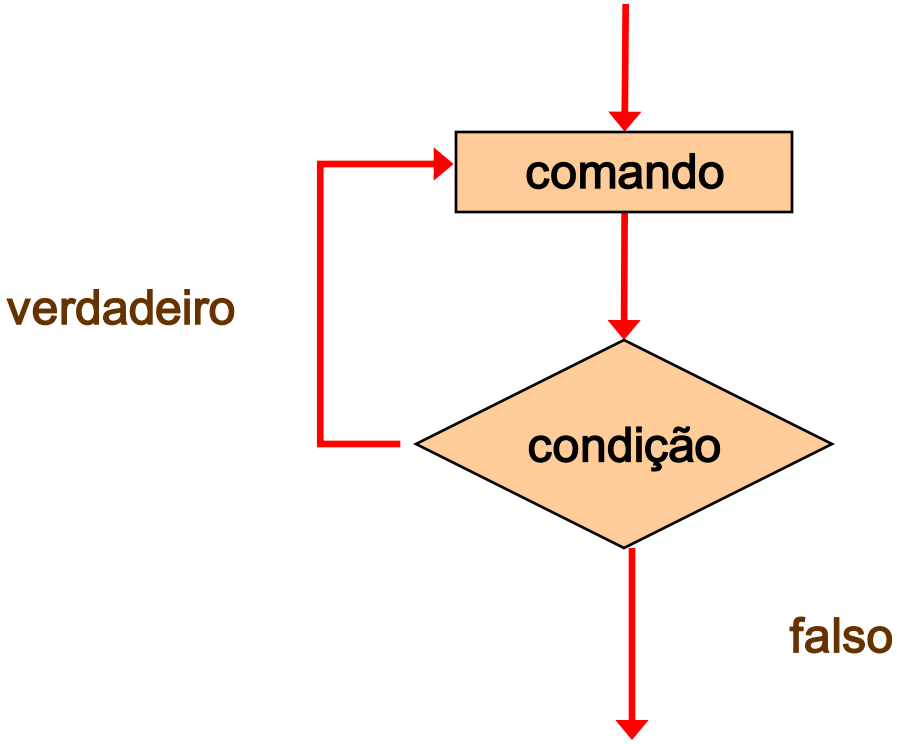
do-while

- A *do-while* (também chamada de `do` loop) tem a seguinte sintaxe:

```
do{
    comando;
}while ( condição )
```

- **A instrução é executada uma vez, inicialmente, e, em seguida, a condição é avaliada**
- **A declaração é executado repetidamente até que a condição se torna falsa**

Logica do do-while Loop



do-while

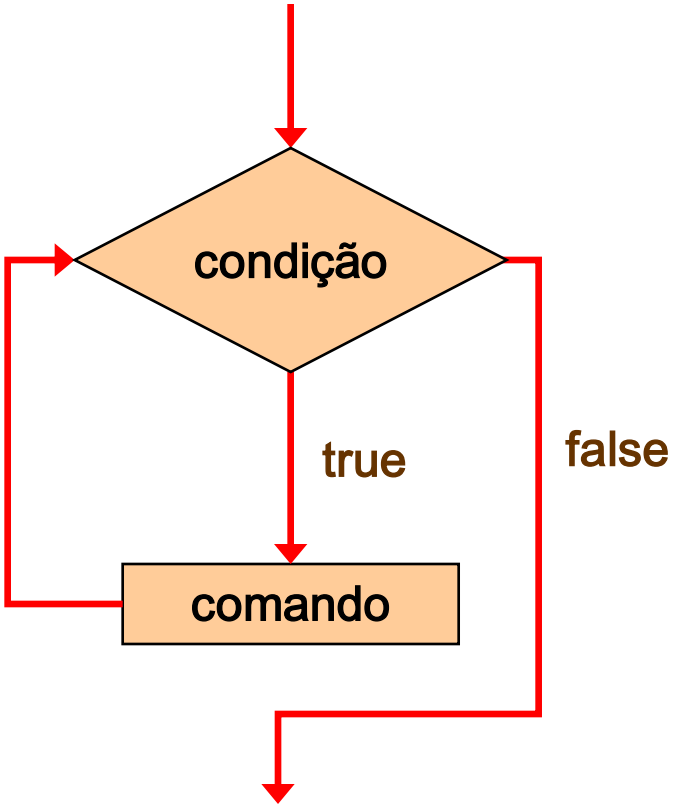
- Exemplo do `do` loop:

```
int count = 0;
do{
   count=cont+1;
   System.out.println (count);
} while (count < 5);
```

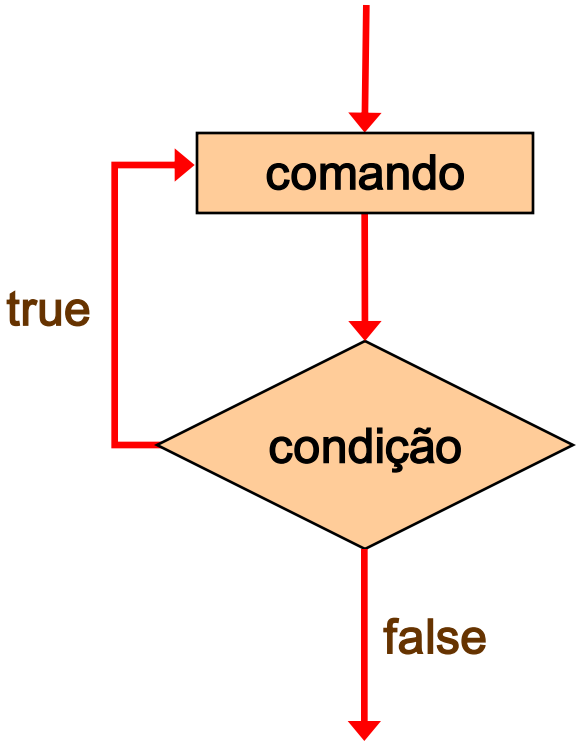
- **O corpo de um loop Do executa pelo menos uma vez**

Comparação

O while Loop



O do Loop



Exercício

Exercício 07

- Faça um programa que numere 1 a 100 e coloque o contador quando é par e quando é impar?

Exercício 07.01

- Definimos duas variáveis inteiras, a 'numero' que irá percorrer do número 1 até 1000.
- A variável 'soma' é inicializada em 0 e servirá para armazenar o valor de todos os números, de 1 até 1000.
- Isso acontece no código: soma = soma + numero;
  No final do loop deve imprimir o total da soma.

Agenda

- Palavras Reservadas
- Operações Lógicas
- IF e ELSE
- Operadores Aritméticos
- While – Repetição
- Do while – Repetição
- Exercício