

Paradigmas de Programação



Aula 02

1. Palavras Reservadas
2. Operadores
3. Execução Condicional
4. Execução Iterativa
5. Palavras Reservadas Usadas
6. Links Úteis



Palavras reservadas

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert***</code>	<code>default</code>	<code>goto*</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum****</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp**</code>	<code>volatile</code>
<code>const*</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0



Operadores Multifunção

Operador Multifunção	Interpretação
++	Incrementa um valor de uma unidade
--	Decrementa um valor de uma unidade
+=	Incremento (de um valor específico)
-=	Decremento (de um valor específico)
*=	Multiplicação (por um valor específico)
/=	Divisão (por um valor específico)
&=	E de bits (de um valor específico)
=	Ou inclusivo de bits (de um valor específico)
^=	Ou exclusivo de bits (de um valor específico)
%=	resto da divisão inteira (por um valor específico)



- 
- A tabela seguinte mostra os exemplos e interpretações:

Operador Multifunção	Exemplo	Equivalente
++	X++, ++X	$x = x + 1$
--	X--, --X	$x = x - 1$
+=	X += y	$x = x + y$
-=	X -= y	$x = x - y$
*=	X *= y	$x = x * y$
/=	X /= y	$x = x / y$
&=	X &= y	$x = x \& y$
=	X = y	$x = x y$
^=	X ^= y	$x = x \wedge y$
%=	X %= y	$x = x \% y$



Repare que ++x e x++ ou --x e x-- não são iguais!

```
int x = 9;  
int y = 11;
```

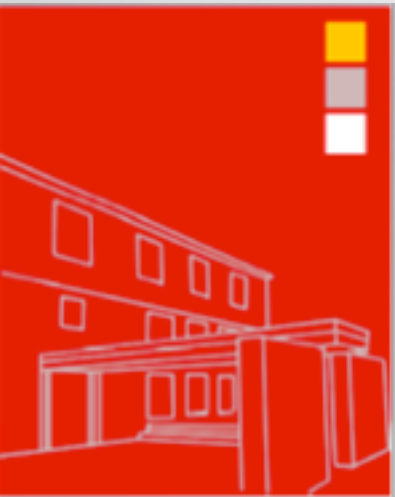
```
if(++x == 10)
```

/* the if condition is true since the value of x is incremented before the expression is evaluated*/

```
if(y-- == 10)
```

/* the if condition is false since the value of y is decremented after the expression is evaluated*/





Execução Condicional

- Como noutras linguagens de programação a linguagem Java fornece construções para a execução condicional, especificamente o if, switch e o operador condicional/ternário ?:



Construções Condicionais

```
if (<boolean-expression>)  
  <statement>...  
else  
  <statement>...
```

```
switch (<expression>) {  
  case <const-expression>:  
    <statements>...  
    break;  
  more-case-statement-break-groups...  
  default:  
    <statements>...}
```

```
(<boolean-expression>) ? <if boolean-expression is true>  
: <if boolean-expression is false>
```



- Uma <boolean-expression> é uma qualquer expressão que permita ter um valor booleano:

Expressões booleanas	Interpretação
$x < 3$	x é menor que 3
$x == y$	x é igual a y
$x \geq y$	x é maior ou igual a y
$x \neq 10$	x não é igual a 10
<variable>	a variável é verdade





- As expressões booleanas podem incluir um ou mais operadores de comparação:

Operador de Comparação	Interpretação
<	Menor que
<=	Menor ou igual que
>	Maior que
>=	Maior ou igual que
==	Igual
!=	Diferente



The if-then-else Statement

```
class IfElseDemo {  
    public static void main(String[] args) {  
  
        int testscore = 76;  
        char grade;  
  
        if (testscore >= 90) {  
            grade = 'A';  
        } else if (testscore >= 80) {  
            grade = 'B';  
        } else if (testscore >= 70) {  
            grade = 'C';  
        } else if (testscore >= 60) {  
            grade = 'D';  
        } else {  
            grade = 'F';  
        }  
        System.out.println("Grade = " + grade);  
    }  
}
```



The switch Statement

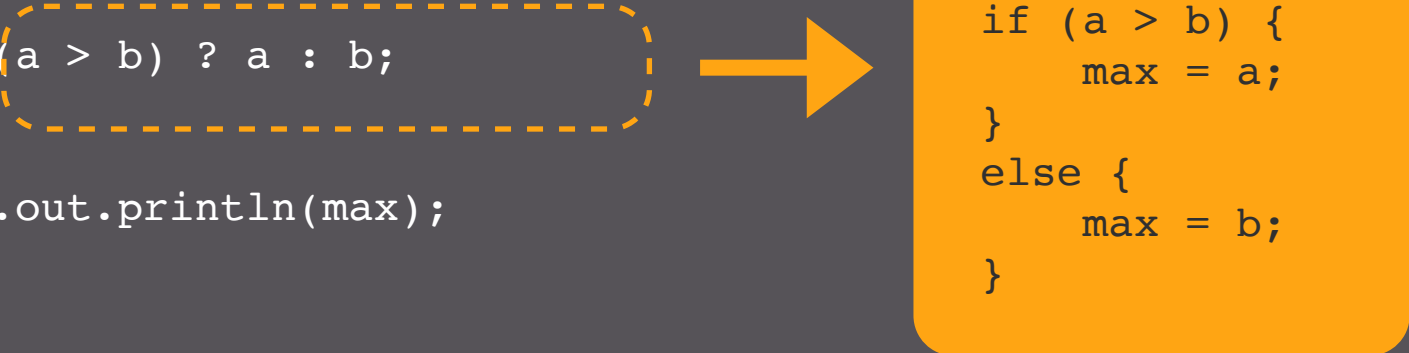
```
class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        switch (month) {
            case 1: System.out.println("January"); break;
            case 2: System.out.println("February"); break;
            case 3: System.out.println("March"); break;
            case 4: System.out.println("April"); break;
            case 5: System.out.println("May"); break;
            case 6: System.out.println("June"); break;
            case 7: System.out.println("July"); break;
            case 8: System.out.println("August"); break;
            case 9: System.out.println("September"); break;
            case 10: System.out.println("October"); break;
            case 11: System.out.println("November"); break;
            case 12: System.out.println("December"); break;
            default: System.out.println("Invalid month.");break;
        }
    }
}
```



The conditional operator

```
class ConditionalOpDemo {  
    public static void main(String[] args) {  
  
        int max,a=2, b=3;  
        max = (a > b) ? a : b;  
  
        System.out.println(max);  
    }  
}
```



```
if (a > b) {  
    max = a;  
}  
else {  
    max = b;  
}
```



Execução Iterativa

- A linguagem de programação Java fornece as estruturas de construção "while", "do-while", e "for" para a iteração de acções múltiplas vezes.



Construções Iterativas

```
while (<boolean-expression>) {  
    <statements>  
}
```

```
do {  
    <statements>  
}  
while (<boolean-expression>);
```

```
for (<init-stmts>...; <boolean-expression>; <exprs>...) {  
    <statements>  
}
```





The while Statement

```
class WhileDemo {  
    public static void main(String[] args){  
  
        int count = 1;  
  
        while (count < 11) {  
            System.out.println("Count is: " + count);  
            count++;  
        }  
    }  
}
```

A construção iterativa *while* executa, enquanto a expressão booleana é verdadeira, um conjunto de acções.

Deve ser usada quando não sabemos o número de vezes que vamos executar um conjunto de acções.



The do-while Statement

```
class DoWhileDemo {  
    public static void main(String[] args){  
  
        int count = 1;  
  
        do {  
            System.out.println("Count is: " + count);  
            count++;  
        } while (count <= 11);  
    }  
}
```

A construção iterativa *do-while* executa um conjunto de acções e uma expressão enquanto a expressão é verdadeira

Deve ser usada quando queremos executar um conjunto de acções pelo menos uma vez.

The for Statement

```
class ForDemo {  
    public static void main(String[] args){  
        for(int i=1; i<11; i++){  
            System.out.println("Count is: " + i); }  
    }  
}
```

Passos de execução para a construção iterativa *for*:

1. A área de inicialização é executada unicamente na primeira vez que o ciclo é executado
2. Avaliação da expressão booleana
 - i. Se verdadeira, execução do conjunto de acções
 - ii. Se falsa quebra a execução do ciclo
3. Executa o incremento
4. Voltar ao ponto 2

Deve ser usada quando queremos executar um determinado conjunto de acções um número específico de vezes



Palavras reservadas

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

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0