



## Java

---

### Operadores Específicos e Compactações Sintáticas

#### METODOLOGIA


- › Interprete o documento calmamente e com atenção.
- › Acompanhe a execução do exercício no seu computador.
- › Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- › Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- › Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

#### CONTEÚDO PROGRAMÁTICO

1. [Operador "and" e "or"](#)
2. [Operador resto da divisão](#)
3. [Operador negação](#)
4. [Operador diferente](#)
5. [Operador ternário](#)
6. [Utilização compacta de operadores](#)
  - 6.1. [Operador ++](#)
  - 6.2. [Operadores aritméticos](#)

## 1. Operador "and" e "or"

Estes dois operadores são bastante usados na programação. A sua utilização mais frequente é nas condições, pois permite-nos agregar condições. O `and` é o operador e, ao passo que o `or` será o operador ou.

 O símbolo do operador e é `&&`  
O símbolo do operador ou é `||`

Um exemplo disso:

```
if ((idade > 0) && (idade < 125)) {  
    1  
}
```

Este `if` interpretado em português lê-se da seguinte forma: "se idade maior que zero e idade menor que 125". Isto significa que se ambas as condições forem verdadeiras o código que se encontra na **área 1** será executado.

Para o operador `ou` exemplifica-se outra condição:

```
if ((idade > 60) || (numero_filhos > 5)) {  
    2  
}
```

Este último `if` contém um `ou`. É então lido da seguinte forma: "se idade maior que sessenta ou numero\_filhos maior que cinco". Caso uma ou outra seja verdadeira então o código na **área 2** será executado.

Agora mostram-se duas tabelas representando os casos possíveis dos valores de condições mediante os operadores. Estas chamam-se tabelas de

verdade.

Operador **E**:

<b>E (&amp;&amp;)</b>	<b>V</b>	<b>F</b>
V	V	F
F	F	F

Operador **Ou**:

<b>OU (  )</b>	<b>V</b>	<b>F</b>
V	V	V
F	V	F

## 2. Operador resto da divisão

Existe um operador que ainda não foi abordado mas que pode ser útil em algumas situações. Este operador permite guardar o resto da divisão por um determinado número. Utiliza-se, por exemplo, para determinar se um número é múltiplo de determinado número ou se um número é par. O símbolo deste operador é o `%`. A utilização é a mesma que para os outros operadores que já vimos.

Comecemos por ver um exemplo da utilização deste operador:

- › **Crie** a classe `RestoDivisao.java`
- › **Introduza** o seguinte **código**:

```
public class RestoDivisao {  
    public static void main(String[] args) {  
        int resto = 5 % 4;  
  
        System.out.println(resto);  
    }  
}
```

### › Compile e teste a classe

📄 O resto da divisão de 5 por 4 é 1. Confirme que é esse o resultado da execução do programa.

Um número par é um número que é divisível pelo número **2**. A definição de divisível subentende que o resto da divisão de um pelo outro resulta em zero. Algo possível de deduzir através deste mesmo operador.

### › Altere o código da classe para o seguinte:

```
import java.util.Scanner;  
public class RestoDivisao {  
    public static void main(String[] args) {  
  
        Scanner teclado = new Scanner(System.in);  
        System.out.println("Insira um numero");  
  
        int numero_lido = teclado.nextInt();  
  
        //obter o resto da divisão do numero por 2  
        int resto = numero_lido % 2;  
  
        if(resto == 0){  
            System.out.println("O numero que introduziu é par");  
        }  
        else {
```

```
        //se o numero não é par tem obrigatoriamente que ser impar
        System.out.println("O numero que introduziu é impar");
    }
}
```

› **Compile** e **execute** a classe

› **Teste** com vários valores

📄 Nesta alteração a divisão é feita entre o número introduzido pelo utilizador e o número **2**, e esse valor é guardado numa variável. É esse valor que depois é utilizado para verificar se o número é ou não par.

Vamos passar a um exemplo de múltiplos de outros valores:

› **Crie** uma nova classe de nome `Multiplos.java`, e **insira** o código:

```
import java.util.Scanner;

public class Multiplos {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        System.out.println("Insira um numero");

        int numero_lido = teclado.nextInt();
        int resto = numero_lido % 7;

        if(resto == 0){
            System.out.println("O numero é multiplo de 7");
        }
        else {
            System.out.println("O numero não é multiplo de 7");
        }
    }
}
```

```
}  
}
```

### 3. Operador negação

O operador negação serve para determinar o contrário de uma determinada expressão. Este tipo de construções é especialmente útil em condições `if`.

› **Altere** na classe `Condicao.java` a expressão de acordo com o código seguinte:

```
if (!(idade < 19)){
```

› **Visualize** o seu efeito

A alteração que acabámos de efetuar faz com que a condição funcione de forma inversa porque estamos a gerar o contrário da condição que lá estava previamente.

A alteração efetuada corresponde exatamente à instrução:

```
if ((idade >= 19)){
```

### 4. Operador diferente

Assim como temos o operador igual, que nos permite avaliar se determinada expressão é igual a outra, existe também o operador inverso, que nos permite comparar uma desigualdade. Isto é, avaliar se uma expressão é diferente da outra.

 O símbolo deste operador é o `!=`.

Segue-se uma ilustração da sua utilização:

```
if (variavel1 != variavel2){
```

## 5. Operador ternário

Este operador é a compactação a nível de sintaxe de um bloco `if` e `else` numa só linha. Permite-nos afetar imediatamente o resultado de uma expressão numa variável. Apesar disto, é desaconselhável o uso deste operador para condições complexas, pois poderá dificultar a leitura do código.

A utilização deste operador segue a seguinte regra:

```
variavel1 = (condição) ? [expressão se verdadeiro] : [expressão se falso] ;
```

Isto significa que o resultado que irá ficar na "variável1" é a "expressão se verdadeiro" ou a "expressão se falso", mediante a respetiva condição.

Passemos a visualizar um exemplo concreto deste operador.

› **Crie** uma classe de nome `OperadorTernario.java`

› **Insira** o seguinte **código**:

```
import java.util.Scanner;

public class OperadorTernario {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        System.out.println("Insira um numero");

        int numero_lido = teclado.nextInt();
        String acima = (numero_lido > 10) ? "maior que" : "menor ou igual a";

        System.out.println("O numero que escreveu é " + acima + " 10");
    }
}
```

› **Compile e teste** a classe

› **Experimente** com valores **acima** e **abaixo** de 10

📄 Repare que o valor guardado na variável `acima` depende da condição. Se a variável `numero_lido` for superior a 10 é guardado o texto "maior que", caso contrário será o texto "menor ou igual a". Este texto é posteriormente escrito para o ecrã.

## 6. Utilização compacta de operadores

### 6.1. Operador ++

No módulo anterior foi abordada uma variante de incremento do valor de uma variável que, no caso em questão, tinha o mesmo efeito que um incremento de forma normal. Exemplifica-se abaixo o mesmo:



```
saldo = saldo + 1;
```

No caso observado nos módulos anteriores, vimos que isto resultava exatamente no mesmo valor que:

```
saldo++;
```

No entanto, o que esta última instrução indica é que, primeiro é usado o valor do saldo e, após a instrução ser concluída, será incrementado este mesmo valor. Para se perceber melhor esta particularidade vamos passar a um exemplo.

› **Crie** uma classe de nome `OperadoresSintaxe.java`

› **Insira** o seguinte código:

```
import java.util.Scanner;

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

        Scanner teclado = new Scanner(System.in);
        System.out.println("Insira um numero");

        int numero_lido = teclado.nextInt();
        System.out.println("Numero lido - " + numero_lido);

        int numero_incrementado = numero_lido++;
        System.out.println("Numero Depois de incrementado - " +
            numero_lido + " | Numero com o valor do incremento - " +
            numero_incrementado);
    }
}
```

## › Compile e teste a classe

O valor do utilizador é lido e mostrado diretamente no ecrã:

```
int numero_lido = teclado.nextInt();  
System.out.println("Numero lido - " + numero_lido);
```


A seguir existem duas operações na mesma linha:

```
int numero_incrementado = numero_lido++;
```

Primeiro é guardado o valor da variável `numero_lido` na variável `numero_incrementado` e, de seguida, incrementado o valor da variável `numero_lido`. Ou seja, o compilador irá transformar a instrução anteriormente escrita pelas duas seguintes instruções:

```
int numero_incrementado = numero_lido;  
numero_lido = numero_lido + 1;
```

Pelo qual se conclui que o incremento é apenas feito a seguir à instrução em si.

 Em alternativa podemos usar o `++` antes da variável, que tem um efeito diferente. Primeiro é incrementada a variável e só depois devolvido o valor da mesma.

## › Troque a seguinte instrução:

```
int numero_incrementado = numero_lido++;
```

Por esta:

```
int numero_incrementado = ++numero_lido;
```

### › Execute e teste a classe

📄 Repare como desta vez os valores diferem do teste anterior. Isto acontece porque a forma como o compilador executa a instrução é diferente.

Analisando em detalhe esta situação e fazendo a mesma analogia, a instrução agora será transformada nas seguintes duas instruções:

```
numero_lido = numero_lido + 1;  
int numero_incrementado = numero_lido;
```

Disto se conclui que, neste caso, o incremento é feito primeiro e depois guardado o seu valor na variável `numero_incrementado`.

## 6.2. Operadores aritméticos

Assim como o incremento de uma unidade tem uma forma abreviada de ser escrito (ainda que com nuances específicas), também os outros operadores aritméticos que conhecemos têm. Operadores: `+`, `-`, `*`, `/` (soma, subtração, divisão e multiplicação respetivamente).

Para o operador `+` podemos utilizar o `+=`, isto quando desejamos somar um determinado valor a uma variável. O exemplo seguinte ilustra esta situação:

```
saldo = saldo + 500;
```

Pode ser escrito da seguinte forma:

```
saldo += 500;
```

Sendo que neste caso é exatamente igual e sem ter qualquer tipo de especificidades ou interpretações do compilador. A única diferença é a facilidade de escrita. Assim como foi somado um valor poderia ter sido somada outra variável.

```
saldo += ordenado;
```

Desta forma estaríamos a somar a variável de saldo o valor da variável `ordenado`.

De seguida visualiza-se uma tabela com os respetivos operadores para as outras operações aritméticas.

Operação	Operador	Exemplo
Soma	+=	variável += valor;
Subtração	-=	variável -= valor;
Multiplicação	*=	variável *= valor;
Divisão	/=	variável /= valor;