
Java Starter
www.t2ti.com

Apresentação

O Curso Java Starter foi projetado com o objetivo de ajudar àquelas pessoas que têm uma base de lógica de programação e desejam entrar no mercado de trabalho sabendo Java,

A estrutura do curso é formada por módulos em PDF e por mini-cursos em vídeo. O aluno deve baixar esse material e estudá-lo. Deve realizar os exercícios propostos. Todas as dúvidas devem ser enviadas para a lista de discussão que está disponível para inscrição na página do Curso Java Starter no site www.t2ti.com. As dúvidas serão respondidas pelos instrutores Albert Eije, Cláudio de Barros e Miguel Kojiio, além dos demais participantes da lista.

Nosso objetivo é que após o estudo do Curso Java Starter o aluno não tenha dificuldades para acompanhar um curso avançado onde poderá aprender a desenvolver aplicativos para Web, utilizando tecnologias como Servlets e JSP e frameworks como Struts e JSF, além do desenvolvimento para dispositivos móveis.

Albert Eije trabalha com informática desde 1993. Durante esse período já trabalhou com várias linguagens de programação: Clipper, PHP, Delphi, C, Java, etc. Atualmente mantém o site www.alberteije.com.

Cláudio de Barros é Tecnólogo em Processamento de Dados.

Miguel Kojiio é bacharel em Sistemas de Informação, profissional certificado Java (SCJP 1.5).

O curso Java Starter surgiu da idéia dos três amigos que trabalham juntos em uma instituição financeira de grande porte.

Módulo 02

Tipos Primitivos, Operadores e Controle de Fluxo

Introdução

Se você está achando essa parte do curso muito teórica, não se preocupe com isso. Você vai usar quase tudo que está sendo visto aqui quando for criar uma aplicação Java, seja ela desktop, para web ou mobile.

Assista aos mini-cursos NetBeans e Eclipe para observar como instalar e utilizar essas IDEs. Você pode escolher qualquer umas das duas para desenvolver os seus projetos Java.

Tipos de Dados

Observe a tabela a seguir, sobre os tipos de dados. Esses tipos são conhecidos como Tipos de Dados Primitivos. Como podemos observar a linguagem Java oferece diversos tipos de dados com os quais podemos trabalhar. Há basicamente duas categorias em que se encaixam os tipos de dados: *tipos primitivos* e *tipos de referências*. Os tipos primitivos correspondem a dados mais simples ou escalares, enquanto os tipos de referências consistem em arrays, classes e interfaces. Estes serão vistos nos módulos subseqüentes.

Vamos a uma descrição curta sobre cada um dos tipos:

Tipo	Descrição
boolean	Pode ser contido em 1 bit, porém o seu tamanho não é precisamente definido. Assume os valores true ou false.
char	Caractere em notação Unicode de 16 bits. Serve para armazenar dados alfanuméricos. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535.
byte	Inteiro de 8 bits em notação de complemento de dois. Pode assumir valores entre -2^7=-128 e 2^7-1=127.
short	Inteiro de 16 bits em notação de complemento de dois. Os valores possíveis

	cobrem a faixa de -2^{-15}=-32.768 a 2^{15}-1=32.767
int	Inteiro de 32 bits em notação de complemento de dois. Pode assumir valores entre -2^{31}=2.147.483.648 e 2^{31}-1=2.147.483.647.
long	Inteiro de 64 bits em notação de complemento de dois. Pode assumir valores entre -2^{63} e 2^{63}-1.
float	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável por esse tipo é 1.40239846e-46 e o maior é 3.40282347e+38. 4 bytes de tamanho e 23 dígitos binários de precisão.
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável é 4.9406565841246544e-324 e o maior é 1.7976931348623157e+308. 8 bytes de tamanho e 52 dígitos binários de precisão.

Para entendermos o comportamento dos tipos acima vamos analisá-los em um programa. Observe atentamente o programa abaixo. Não passe adiante sem compreender cada linha desse programa. Ele está todo comentado. Essa classe está em anexo para que você a abra e teste sua execução no Netbeans ou no Eclipse.

```
public class TiposPrimitivos {

	public static void main(String[] args) {

		/*
		 Declaramos 8 variáveis. Exatamente o mesmo número
		 dos tipos primitivos do Java. Cada variável é de
		 um dos tipos.
		 */
		boolean bo;
		char c;
		byte b;
		short s;
		int i;
		long l;
		float f;
		double d;

		/*
		 Atribuímos o valor 65 à variável c, que é do tipo
		 char.

		 OBS Importante: O tipo char também é um inteiro!
		 No entanto é um inteiro que faz referência a tabela
		 Unicode, que contém 65535 simbolos.
		*/
		c = 65;
		System.out.println("============================");
		System.out.println("char");

		/*
```

```
 Note a diferença entre a impressão das duas linhas
 abaixo:
 */
System.out.println("============================");
System.out.println("valor de c como char = " + c);
System.out.printf("valor de c como numero = %d \n",(int)c);
System.out.println("----------------------------");

/*

 Saída da impressão:

 ============================
 char
 ============================
 valor de c como char = A
 valor de c como numero = 65
 ----------------------------

 Interessante notar que além do println você também
 pode usar o printf que aprendemos lá no C. Vale
 aqui tudo que aprendemos no Módulo 1 em relação
 à função printf.
 */


/*
 Agora vamos trabalhar com os outros tipos inteiros:
 byte, short e int.

 Veja que estamos trabalhando com o valor 10 e que
 atribuímos esse valor à variável b, que é do tipo
 byte. Como esse valor "cabe" em um short e em um int
 não há problema nenhum quando atribuímos o valor de b
 à variável s (short) e o valor de s à variável i (int)
 */

b = 10;
s = b;
i = s;
System.out.println("============================");
System.out.println("inteiros");
System.out.println("============================");
System.out.println("i = s = b = "+i);

/*

 Saída da impressão:

 ============================
 inteiros
 ============================
 i = s = b = 10
 */

/*
 Agora multiplicamos i por 100 e atribuimos o total
 dessa multiplicação ao próprio i.
 */
i *= 100;

System.out.println("novo valor de i = "+i);
```

```
/*

 Saída da impressão:

 novo valor de i = 1000
 */

/*
 Olhe para a linha de código abaixo com muita
 atenção. Eu tento colocar o valor de i em b, mas o
 Java não deixa. Preciso então fazer um casting
 (conversão) explícito. Antes da variável i eu insiro
 entre parênteses o tipo de dado para o qual eu
 quero convertê-la. Lembre-se que acima eu converti
 de byte para short e de short para int e isso não
 foi necessário. No entanto, essa conversão acima
 também foi um casting, só que foi um casting implícito.

 Quando eu realizo a conversão abaixo algo vai dar errado.

 Imagine que você pegasse 10 copos de suco de laranja
 e jogasse o coteúdo dos 10 copos em uma jarra. Tranquilo,
 sem problemas! Agora imagine você pegando o conteúdo da
 jarra e jogando em apenas um dos copos. Haverá uma
 perda considerável de suco!

 Depois da conversão de int para byte abaixo, onde atualmente
 o valor de i é 1000, o valor de b será -24! Veja mais abaixo
 porque isso acontece.
 */

b = (byte)i;

System.out.println("novo valor de b = "+b);

/*

 Saída da impressão:

 novo valor de b = -24
 */


/*
 O que ocorre é que no lugar de o Java lançar uma exceção
 ou converter para zero ele retira a parte binária do int
 e deixa apenas a do byte e apresenta o que restou.

 Observe o esquema abaixo:

 i = 1000 em binário --> 00000000 00000000 00000011 11101000
 b = 1000 em binário --> -------- -------- -------- 11101000


 Veja que boa parte do número é perdido. Se você pegar o que
 sobrou no b e fizer uma conversão de binário para decimal
 usando complemento de dois vai chegar ao número -24.

 Portanto, saiba que o Java vai deixar você fazer esse tipo
 de conversão porque ele acha que você, programador, sabe
 que poderá perder bits nessa conversão!
 */
```

```
/*
 Abaixo simplesmente inserimos o valor do int em um long
 que é o dobro de um int. Ou seja, sem problemas!
 */
l = i;

System.out.println("valor de l = "+l);
System.out.println("----------------------------");

/*

 Saída da impressão:

 valor de l = 1000
 ----------------------------
*/


/*
 Chegamos aos números com ponto flutuante (casas decimais).
 */
System.out.println("============================");
System.out.println("ponto flutuante");
System.out.println("============================");

d = 125.32;
System.out.println("valor de d = " + d);
d = 125.32d;
System.out.println("valor de d = " + d);
d = 125.32f;
System.out.println("valor de d = " + d);

/*
 no primeiro caso atribuimos 125.32 à variavel d.
 sem problemas. funciona legal, como podemos ver
 na Saída da impressão.

 Logo depois atribuimos 125.32d. Mas pra que serve
 esse "d" depois do valor? Indica que estou afirmando
 para o Java que a constante (125.32) é um double.

 Os dois casos acima são idênticos. Como assim? Para
 o Java, o tipo padrão de um literal com ponto
 flutuante é double. Portanto, não precisa colocar o
 "d" depois do literal se quiser que ela seja double.

 literal --> 125.32

 Mas o terceiro caso é interessante. Atribuimos à
 variável d o seguinte valor: 125.32f. Dessa vez estamos
 inserindo um float num double. problema nenhum, pois
 o float é 32 bit e o double é 64 bit. Mas, observe a Saída
 da impressão. Por que o número depois da casa decimal
 ficou tão quebrado? Investigue e comente na lista.


 Saída da impressão:

 ============================
 ponto flutuante
```

```
 ============================
 valor de d = 125.32
 valor de d = 125.32
 valor de d = 125.31999969482422
*/


f = (float)125.32;
System.out.println("valor de f = " + d);
f = 125.32f;
System.out.println("valor de f = " + d);
f = (float)125.32d;
System.out.println("valor de f = " + d);
System.out.println("----------------------------");

/*
 Mesma coisa que foi feita com o Double, mas dessa
 vez com o float.

 Logo no primeiro caso já é necessário fazer um casting.
 Lembre-se: o tipo padrão de um número literal em
 ponto flutuante é double.

 No segundo caso nada é preciso.

 No terceiro caso novamente a presença no casting
 já que estamos informando explicitamente que o literal
 é do tipo double.

 É bom lembrar que ao converter de double para float
 pode haver perda de bits.

 Pergunta: por que todos os valores saíram tão quebrados,
 em contraste com o double? Investigue e comente na lista.

 Saída da impressão:

 valor de f = 125.31999969482422
 valor de f = 125.31999969482422
 valor de f = 125.31999969482422
 ----------------------------
*/


/*
 Abordaremos agora o tipo booleano.
 */
System.out.println("============================");
System.out.println("booleano");
System.out.println("============================");

bo = true;
System.out.println("valor de bo = " + bo);

bo = (1 > 2);
System.out.println("valor de bo = " + bo);

bo = (f == d);
System.out.println("valor de bo = " + bo);
System.out.println("----------------------------");

/*
```

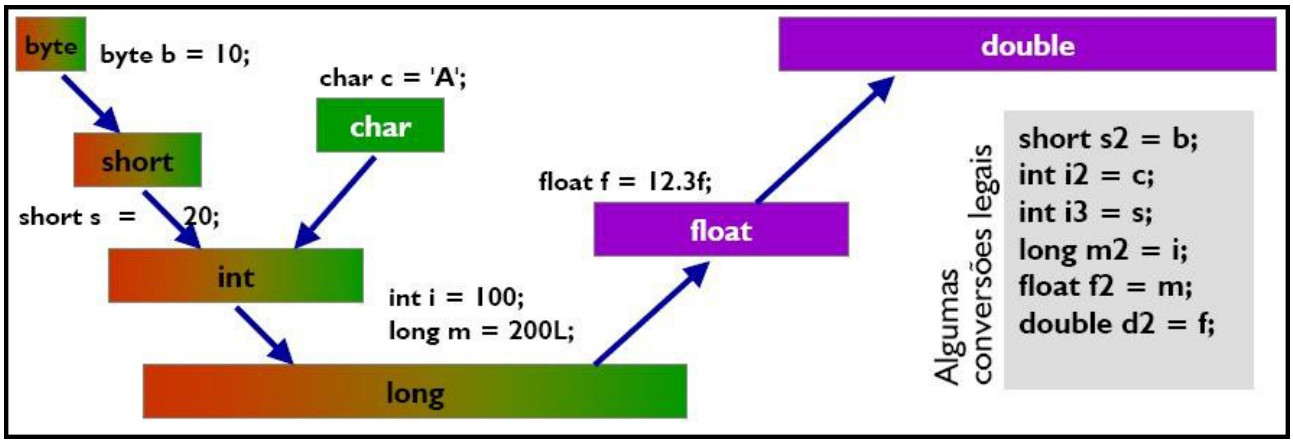
```
O valor true ou false pode ser atribuido sem problemas.

Se atribuirmos uma operação à variável. Essa operação
será avaliada e seu valor será armazenado na variável.
No caso de (1>2) sabemos que o resultado será false.

Depois testamos se f é igual a d. Observe que o operador
de teste de igualdade é o == (igual duas vezes), semelhante
à linguagem C. Falaremos sobre operadores mais a frente.


Saída da impressão:

============================
booleano
============================
valor de bo = true
valor de bo = false
valor de bo = true
----------------------------
*/

System.out.println("============================");
System.out.println("brincando com os tipos");
System.out.println("============================");

System.out.println("posso converter int para float?");
System.out.println("i antes da conversao = " + i);
System.out.println("f antes da conversao = " + f);
f = i;
System.out.println("i depois da conversao = " + i);
System.out.println("f depois da conversao = " + f);

System.out.println("--");

/*
 Em nossa primeira "brincadeira" com os tipos tentamos
 converter de int para float. Observe atentamente o
 resultado na saída de impressão.

 Saída de impressão:

 ============================
 brincando com os tipos
 ============================
 posso converter int para float?
 i antes da conversao = 1000
 f antes da conversao = 125.32
 i depois da conversao = 1000
 f depois da conversao = 1000.0
 --
*/

f = (float)d;

System.out.println("posso converter float para int?");
System.out.println("i antes da conversao = " + i);
System.out.println("f antes da conversao = " + f);
i = (int)f;
System.out.println("i depois da conversao = " + i);
System.out.println("f depois da conversao = " + f);
System.out.println("----------------------------");
```

```
        /*
         Depois de nossa primeira "brincadeira" o valor de
         f ficou igual ao valor de i. Por isso atribuímos o
         valor de d novamente à variável f.

         Agora o que queremos saber é se é possível atribuir
         o valor de um float a um inteiro. O contrário foi
         possível sem casting. Mas dessa vez o casting será
         necessário. Mas, além desse detalhe, outra coisa
         vai ocorrer. Pode ser desejável ou não. Comente na
         lista o que ocorreu nessa conversão de float para int
         e em que situação isso seria desejavel.


         Saída de impressão:

         posso converter float para int?
         i antes da conversao = 1000
         f antes da conversao = 125.32
         i depois da conversao = 125
         f depois da conversao = 125.32
         ----------------------------
         */
    }
}
```

Pelo que podemos observar no programa anterior, as conversões em Java ocorrem da seguinte maneira, sem perda de informação:

- Um **`byte`** pode ser convertido em um `short`, `int`, `long`, `float` ou `double`
- Um **`short`** pode ser convertido em um `int`, `long`, `float` ou `double`
- Um **`char`** pode ser convertido em um `int`, `long`, `float` ou `double`
- Um **`int`** pode ser convertido em um `long`, `float` ou `double`
- Um **`long`** pode ser convertido em um `float` ou `double`
- Um **`float`** pode ser convertido em um `double`

Obseve a imagem seguir:



Já as conversões explícitas, o casting, é permitido em todos os tipos (exceto o boolean), mas o programador deve estar ciente que poderá haver perda de bits.

Operadores

Os operadores em Java são muito parecidos com os da linguagem C. No entanto, existe alguns operadores novos. Analise cuidadosamente a tabela abaixo e faça diversos testes com esses operadores para se acostumar com eles. Durante o desenvolvimento de seus sistemas você utilizará, e muito, a maioria desses operadores.

Grupo = operadores organizados em grupos;
Operador = cada um dos operadores;
Precedência = ordem de precedência dos operadores;
A = associatividade do operador (esquerda ou direita);
Operando = operandos do operador;
Descrição = descrição da operação.

Alguns operadores relacionados abaixo não serão comentados agora porque são utilizados com objetos ou *tipos referências*, assunto que será estudado mais a frente. Eles estão com a cor da linha diferente. Os demais já podem ser implementados nos exercícios deste módulo.

É importante frisar que você deve praticar bastante o uso desses operadores. Crie uma classe com um método Main, similar a que fizemos no programa anterior e implemente diversas situações onde você possa analisar o uso de cada um dos

operadores.

Comente seus resultados e observações na lista de discussão. Além disso, se surgirem dúvidas, envie-as para a lista.

Grupo	Operador	Precedência	A	Operando	Descrição
Delimitador	.	1	E	objeto,membro	acesso a um membro de um objeto ou classe.
Delimitador	[ ]			vetor	acesso ao elemento de um vetor
	(argumentos)			método, lista de argumentos	acesso a elemento do vetor
aritmético	++ e --			variável	pós-incremento, pós-decremento
aritmético	++ e --	2	D	variável	pré-incremento, pré-decremento
aritmético	+ e -			variável	mais e menos unário
binário	~			inteiro	complemento de 1 (binário)
binário	!			booleana	Negação cujo resultado é lógico (true/false)
classe	new	3	D	classe, lista de argumentos	criação de objeto
conversão	(tipo)			tipo, qualquer coerção	conversão de tipo
aritmético	* / %	4	E	número, número	multiplicação, divisão e módulo (resto)
aritmético	+ -	5	E	número, número	adição e subtração
String	+ +=			String, qualquer	concatenação
binário	<<	6	E	inteiro, inteiro	desl. a esquerda
binário	>>			inteiro, inteiro	deslocamento a direita com sinal
binário	>>>			inteiro, inteiro	deslocamento a direita com zeros
condicional	< <=	7	E	número, número	menor que, menor ou igual a
condicional	> >=			número, número	maior que, maior ou igual a
condicional	instanceof			referência, tipo	verificação de tipo

condicional	==	8	E	primitivo,primitivo	igualdade de valores
condicional	!=			primitivo,primitivo	diferença de valores
condicional	==			referência, referência	igualdade, endereço dos objetos
condicional	!=			referência, referência	diferença, endereço de objetos
binário	&	9	E	inteiro, inteiro	E sobre bits
lógico	&			booleana, booleana	E booleano
binário	^	10	E	inteiro, inteiro	XOR sobre bits
Grupo	**Operador**	**Precedência**	**A**	**Operando**	**Descrição**
lógico	^			booleana, booleana	XOR (OU exclusivo) booleano
binário	|	11	E	inteiro, inteiro	OU sobre bits
lógico	|			booleana, booleana	OU booleano
lógico	&&	12	E	booleana, booleana	E condicional
lógico	||	13	E	booleana, booleana	OU condicional
condicional	?:	14	D	booleana, qualquer	operador ternário, condicional
atribuição	=	15	D	variável, qualquer	atribuição
	*= /= %=			variável, qualquer	atribuição com operação
	+= -= <<=				
	>>= >>>=				
	&= ^= |=				

Em se tratando de precedência é bom lembrar que os parênteses podem ser usados para modificá-la.

2 + 6 * 5 = 32
(2 + 6) * 5 = 40

Controle de Fluxo

O controle de fluxo é efetuado através do uso de **condicionais**, que são

estruturas que modificam o fluxo de execução normal do programa.

if-else

Sintaxe:

```
if (expressão booleana)
      instrução_simples;

if (expressão booleana) {
      instruções
}
```

```
if (expressão booleana) {
      instruções
} else if (expressão booleana) {
      instruções
} else {
      instruções
}
```

Quando existe apenas uma instrução após o if não precisamos abrir um bloco com as chaves. Se existirem mais instruções a abertura do bloco é necessária.

Se houver mais de uma condição usa-se o else como podemos observar na segunda coluna acima.

Veja o exemplo abaixo e comente o mesmo na lista de discussão.

```
if ( ano < 0) {
      System.out.println("Não é um ano!");
} else if ( (ano%4==0 && ano%100!=0) || (ano%400==0) ) {
      System.out.println("É bissexto!");
} else {
      System.out.println("Não é bissexto!");
}
```

while e do-while

Sintaxe:

```
while (expresão booleana)
{
      instruções;
}
```

```
do
{
      instruções;
} while (expresão booleana );
```

No caso do while as instruções serão executadas enquanto a expressão booleana for verdadeira.

O do-while executa a instrução pelo menos uma vez e continua executando enquanto a expressão booleana for verdadeira.

Observe os exemplos seguintes.

```
//exemplo com while
int x = 0;
while (x < 10) {
      System.out.println ("item " + x);
      x++;
}

//exemplo com do-while
int x = 0;
do {
      System.out.println ("item " + x);
      x++;
} while (x < 10);

//esse é um laço infinito. Pra que ele serve?
while ( true ) {
      if (obj.z == 0) {
            break;
      }
}
```

for

Sintaxe:

`for ( inicialização;`
`expressões booleanas;`
`passo da repetição )`
`  instrução_simples;`
	`for ( inicialização;`
	`expressões booleanas;`
	`passo da repetição )`
	`{`
	`  instruções;`
	`}`

O **for** pode conter apenas uma instrução no seu corpo. Neste caso não é necessário abrir um bloco. Isso é assim porque o for já implementa alguns comandos na sua assinatura, ou seja, no seu cabeçalho, como a inicialização da variável e o passo da repetição, ou seja, o incremento/decremento da variável.

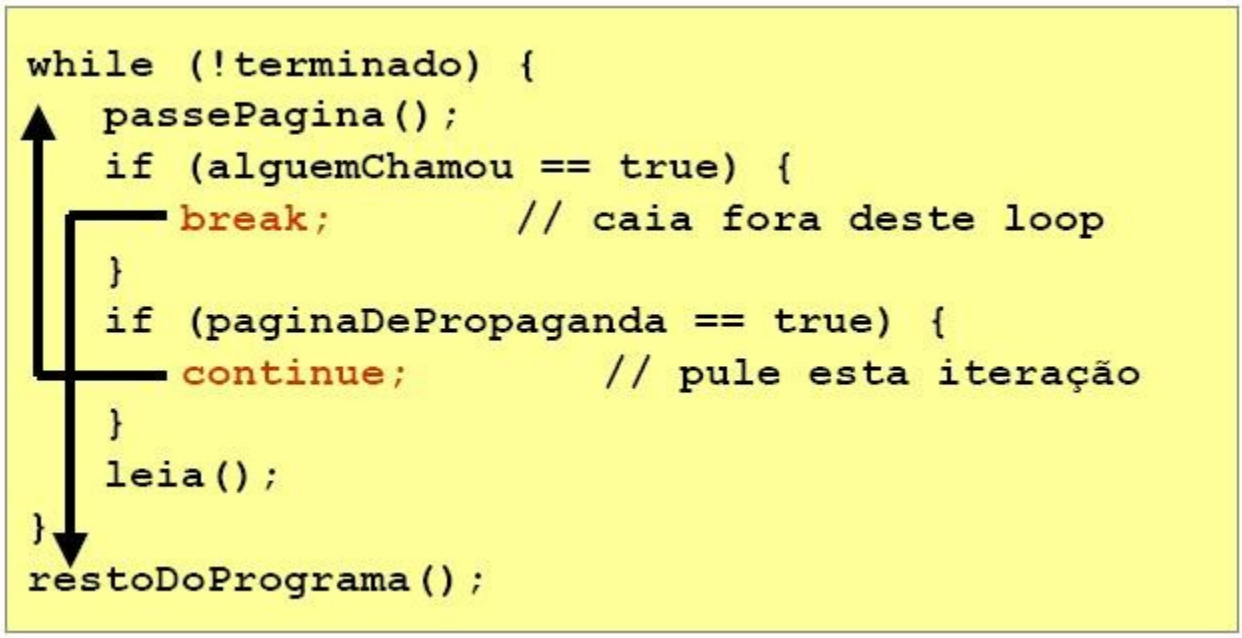
Veja os exemplos:

```
//esse tá bem fácil.
for ( int x = 0; x < 10; x++ ) {
      System.out.println ("item " + x);
}

//esse já não é tão simples só de olhar.
for ( int x = 0, int y = 25; x < 10 && (y % 2 == 0); x++, y = y - 1 ) {
      System.out.println (x + y);
}

//também podemos fazer um laço infinito com o for
for ( ; ; ) {
      if (obj.z == 0) {
            break;
      }
}
```

break e continue

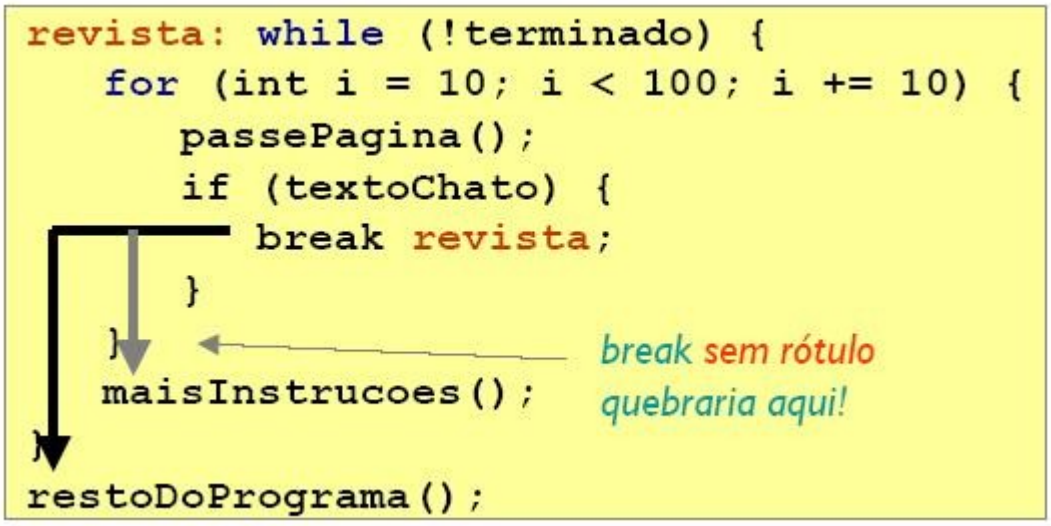
Você deve ter observado que quando fizemos os laços infinitos com o while e com o for utilizamos um comando break. Para que ele serve? Este comando serve para "quebrar" o laço, ou seja, ele fará com que o laço seja interrompido.

O comando continue também influi dentro de um laço. Mas ele não vai quebrar o laço. Ele interrompe aquela iteração do laço e reinicia o bloco com a próxima iteração.



Rótulos (break e continue)

E se tivermos com um laço dentro de outro e quisermos quebrar o laço mais externo? O break ou continue agem no laço mais interno. Existe uma maneira: rotular nossos laços. Observe a imagem a seguir:



switch (case)

Sintaxe:

```
switch(seletor_inteiro) {
      case valor_inteiro_1 :
            instruções;
            break;
      case valor_inteiro_2 :
            instruções;
            break;
      ...
      default:
            instruções;
}
```

```
switch(letra) {
      case 'A' :
            System.out.println("A");
            break;
      case 'B' :
            System.out.println("B");
            break;
      ...
      default:
            System.out.println("?");
}
```

Switch só trabalha com valores inteiros, incluindo o char.

Palavras Reservadas

Toda linguagem possui palavras reservadas e isso não seria diferente em Java. Uma versão mais recente do Java pode ter mais palavras reservadas que uma anterior. A seguir você pode ver uma imagem com boa parte das palavras reservadas em Java. Pesquise e descubra quais palavras reservadas não estão na lista abaixo. Comente na lista.

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

Tente utilizar algumas dessas palavras como nome de variáveis e veja como o compilador Java se comporta.

Você deve ter percebido que não utilizamos um tipo String neste módulo. O Java possui um tipo String, no entanto, String no Java não é um tipo primitivo. Falaremos sobre String em um módulo posterior.

Entrada de Dados

Como eu faço para ler algo que o usuário digita no prompt?

Observe o programa abaixo:

```
//importe a classe Scanner para utilizar a leitura pelo teclado
//Similar ao #include do C
import java.util.Scanner;

public class Leitura {

	public static void main(String[] args) {
		// crie a variável de leitura dos dados
		Scanner s = new Scanner(System.in);
		// use os métodos de leitura específicos do tipo desejado
		System.out.print("digite uma linha: ");
		String linha = s.nextLine(); // le a linha
		System.out.print("digite um numero: ");
		int i = s.nextInt(); // le um inteiro
		System.out.print("digite um numero: ");
		double d = s.nextDouble(); // le um ponto-flutuante
	}

}
```

Com o estudo do programa acima já dá para você solicitar que o usuário digite alguma coisa e utilizar o que o mesmo está digitando.

Vamos aos tão aguardados exercícios.

Exercícios

Aprenda com quem também está aprendendo, veja e compartilhe as suas respostas no nosso Fórum:

Exercícios – Módulo 02 – Tipos Primitivos, Operadores e Controle de Fluxo

01 - Escreva um programa que imprima o resultado das expressões abaixo:
- 3 – 2 – 1 + 2 + 1 + 3
- 2 x 3 – 4 x 5
- 2 + 6 – 3 / 7 x 9
- 3 % 4 –8

02 - Escreva um programa que declare, inicialize e imprima as seguintes variáveis:
- Inteiro i de 32 bits com valor 1
- Inteiro j de 64 bits com valor 2
- Ponto-flutuante p de 32 bits com valor 20.0
- Ponto-flutuante q de 64 bits com valor 30.0
- Boolean b com valor verdadeiro
- Caracter c com valor 'k'

03 - Implemente um programa que recebe um número de 1 a 7 e imprime o dia da semana correspondente.

04 - Altere o programa do exercício anterior para ficar recebendo o número dentro de um laço enquanto o número for diferente de 0 (zero).

05 - Implemente um programa que recebe repetidamente um número de 1 a 12, enquanto esse número for diferente de 0 (zero), e imprime o mês correspondente. Quando o número estiver fora do intervalo permitido, a mensagem "mês inválido" deverá ser exibida.

06 - Escreva um laço while que execute 20 vezes, imprimindo o valor da variável x que inicialmente está com valor 10. Converta este laço para um do-while.

07 - Escreva um programa que imprima na tela a soma dos números ímpares entre 1 e 30 e a multiplicação dos números pares entre 1 e 30.

08 - Escreva um programa que percorra dois laços de 0 a 10, um interno ao outro, imprimindo os contadores, e quando estes forem iguais, o programa deve passar à próxima interação do laço mais externo, caso contrário, deve imprimir os valores dos contadores dos dois laços.

09 - Desenvolva um programa que solicita ao usuário um número inteiro de no máximo 4 dígitos. Inverta esse número.

10 – Desenvolva um programa que dado um número inteiro o programa informe se o mesmo é um número primo.

11 - Escreva um programa em Java que calcula e imprime na tela o salário proporcional de um funcionário que trabalhou apenas N dias num mês de 22 dias úteis. O número de dias trabalhados bem como o salário integral do funcionário devem ser lidos do teclado. O resultado deve ser um número ponto-flutuante.

12 - Escreva um programa em Java que leia repetidamente um número do teclado até que seja digitado o número zero (0) e determine se o número lido é perfeito ou não, imprimindo o resultado. Um número é dito perfeito quando é igual a soma dos seus divisores menores do que ele, por exemplo, 6 é perfeito, uma vez que 6 = 3 + 2 + 1.

13 – Faça um programa que contenha um menu com 4 opções:
1 – calcular o fatorial de um número dado
2 – calcular a raiz quadrada de 3 números dados
3 – imprimir a tabuada completa de 1 a 10
4 – sair do programa

14 - Escreva quatro instruções em Java diferentes para adicionar 1 a uma variável inteira x.

15 - Escreva instruções Java para realizar a seguinte tarefa: atribuir soma de **x** e **y** a **z** e incrementar **x** por 1 depois do calculo. Use somente uma instrução;

16 - Escreva instruções Java para realizar a seguinte tarefa: decrementar a variável **x** por 1, depois subtrair o resultado da variável **total** com somente uma instrução.

17 - Implemente um programa para calcular a área de um trapézio, onde:
h = altura
b = base menor
B = base maior
Área = (h.(b+B))/2

18 - Fulano aplicou R$ 100,00 com rendimento de 5% ao mês. Quantos meses serão necessários para o capital investido ultrapasse a R$ 200,00. Desenvolva um programa que realize essa operação.

19 – Faça um programa que imprima os quadrados dos números inteiros ímpares entre 15 e 35.

20 - Escreva um aplicativo que imprime as seguintes formas. Você pode utilizar instruções de saída que imprimem um único asterisco (*), um único caractere de espaço ou uma nova linha. Maximize a utilização de estruturas de repetição(com estruturas aninhadas **for**) e minimize o número de instruções de saída.

*	********	********	*
**	*******	*******	**
***	******	******	***
****	*****	*****	****
*****	****	****	*****
******	***	***	******
*******	**	**	*******
********	*	*	********