Java e sua Sintaxe

Prof. Omero Francisco Bertol.

(omero@utfpr.edu.br)

Março/ 2011

Estrutura Mínima de um Programa

Um programa java (seja uma aplicação ou um *applet*) pode ser composto de um ou mais arquivos-fonte, denominados unidades de compilação, que podem utilizar:

- uma declaração de pacote (*package*);
- uma ou mais diretivas de importação (*import*);
- uma ou mais declarações de classes (*class*);
- uma ou mais declarações de interfaces (*interface*).

A declaração de um pacote (*package*) se refere à definição de um conjunto de classes que pertencem a esse pacote, enquanto as diretivas de importação (*import*) permitem indicar quais pocotes de classes serão utilizadas.

A declaração de classes (*class*) e interfaces (*interface*) permite a definição de novos tipos e dependências entre os mesmo.

Programa Mínimo Java

```
public class Exemplo1 {
   public static void main(String args[]) {
      // corpo do método main
   }
}
```

Acima, temos a declaração de uma classe de nome Exemplo1, na qual existe um método **main** que não contém instruções.

Considerações Iniciais sobre a Sintaxe

- Em um programa, as diversas "declarações" utilizadas devem ser separadas por um "**;**" (ponto-e-vírgula).
- Várias declarações podem ser colocadas numa mesma linha, devidamente separadas pelo "**;**".
- Blocos de declarações, delimitados por chaves, não precisam ser seguidos pelo "**;**".
- Espaços em branco, tabulações ou mesmo linhas em branco (chamados de *whitespace*) podem ser utilizados livremente para a formatação do programa, exceto quando dividem nomes declarados, palavras reservadas ou os operadores da linguagem.

Exemplos de Comentários

O comentário de uma linha utiliza duas barras (//) para marcar seu início:

```
// comentário de uma linha a partir do ínicio da linha
int x = 0; // comentário de um linha após um comando
           // tudo após as duas barras é considerado
           // comentário
```

O segundo tipo utiliza a combinação /* e */ para delimitar uma ou mais linhas, cujo conteúdo deverá ser considerado como um comentário

```
    /* comentário
       de múltiplas linhas */
```

O último tipo tem o próposito de documentar o programa. Seu marcador de início é /** e seu marcador de fim */

```
    /** comentário de documentação que também
        pode ter múltiplas linhas */
```

obs. o comentário de documentação deve ser posicionado imediatamente antes do elemento a ser documentado e tem seu conteúdo extraído automaticamente pelo utilitário **javadoc** que acompanha o kit de desenvolvimento Java.

Tipos de Dados Primitivos

Um tipo de dados é a definição de um certo conjunto de valores para os quais existe uma representação específica dentro do programa, podendo também ser definidas operações sobre tais dados. Quando a definição de um tipo e das operações aceitas para o mesmo fazem parte da linguagem, isto é, estão predefinidas, é dito ser este um "tipo primitivo".

A linguagem Java tem "oito" tipos primitivos, que podem ser agrupados em "quatro" categorias:

Tipos inteiros
Byte
Inteiro Curto
Inteiro
Inteiro Longo

Tipos em ponto flutuante
Ponto flutuante simples
Ponto flutuante duplo

Tipo caracter
Caracter

Tipo Lógico
Booleano

Tipos de Dados Inteiros

Tipo	Tamanho	Valor Mínimo..Valor Máximo
byte	8 bits	-128..127
short	16 bits	-32.768..32.767
int	32 bits	-2.147.483.648..2.147.483.647
long	64 bits	-9.223.372.036.854.775.808L..9.223.372.036.854.775.807L

Por *default*, os valores diretamente escritos em um programa são tratados com inteiros simples (int), ou seja, valores de 32 bits. O tamanho de cada um desses tipos independe do sistema operacional ou do hardware utilizado. O valores podem ser expressos em base *octal*, se precedidos por um "**0**", enquanto os valores hexadecimais devem ser precedidos por "**0x**" ou "**0X**". Valor do tipo *long* devem ser finalizados por um "**L**", como mostram os próximos exemplos:

-13	7691**L**	**0**123	**0x**CAFE10
int	long	int (octal)	int (hexadecimal)

Tipos de Dados em Ponto Flutuante

Tipo	Tamanho
float	32 bits
double	64 bits

O caracter ponto "**.**" deve ser utilizado como separador de casas decimais, enquanto expoentes podem ser escritos usando o caractere "**e**" ou "**E**", como nos seguintes valores:

1.44**E**6 3.4254**e**-2 -25.342**E**8 -0.32**E**-17

Por *default*, os valores reais são tratados como double. A indicação de valores do tipo float pode ser feita seguindo-se o valor numérico real por um caracter "**f**", tal como 55.3**f** ou 2.73E-2**f**.

Tipos de Dados de Caracter

O tipo char permite a representação de caracteres individuais e sempre devem ser delimitados por aspas simples(‘’), tal como:

‘A’ ‘c’ ‘7’ ‘\n’ (caracter especial que não tem uma representação visual, combinação de uma barra invertida (‘\’) com um código convencionado.)

Representação de caracteres especiais mais comuns

\n	pula linha (*newline* ou *linefeed*)
\r	retorno de carro (*carrige return*)
\b	retrocesso (*backspace*)
\t	tabulação (*horizontal tabulation*)
\f	nova página (*formfeed*)
\’	apóstrofe
\”	aspas
\\	barra invertida
\u223d	caracter \UNICODE 233d
\g37	Octal 37
\fca	Hexadecimal

Tipos de Dados Lógicos ou Booleanos

Em Java o tipo lógico boolean é capaz de assumir os valores falso ou verdadeiro, isto é, que equivalem aos estados *off* (desligado) ou *on* (ligado), ou *no* (não) e *yes* (sim).

Os valores que representam tais significados são: **false** e **true** (observe que devem ser usadas letras minúsculas para indicar os valores lógicos)

Cuidado: Java é *case-sensitive* (diferencia letras minúscula de letras maiúsculas).

Diferente das linguagens C/C++, não existe nenhuma equivalência entre valores do tipo lógico (boolean) e valores dos tipos inteiros (byte, short, int e long), ou seja, o valor 0 é diferente do valor false, e 1 também é diferente de true, não podendo ser utilizados em nenhum tipo de expressão por caracterizar uso indevido de tipos (*type mismatch*).

Variáveis

Uma variável é um nome definido pelo programador, ao qual pode ser associado um valor pertencente a um certo tipo de dados. Em outras palavras, uma variável é um conjunto de posições da memória do computador, tratadas como uma unidade capaz de armazenar um valor de um certo tipo.

O nome que se dá a uma variável é, na verdade, uma representação simbólica do local da memória (endereços de memória) em que se encontra o valor associado.

Como não é necessário que o programador conheça detalhes sobre a localização dos nomes, usualmente, utilizamos nomes que descrevam o significado ou o propósito de tal variável.

Dessa forma, toda variável sempre tem um nome, um tipo e um conteúdo. Também devemos considerar que é praticamente impossível implementarmos um programa de razoável utilidade sem o uso de um conjunto de variáveis.

Regras para Nomes de Variáveis

O nome de uma variável Java pode ser formado por uma seqüência de um ou mais caracteres alfabéticos e numéricos, iniciados por uma letra ou ainda pelos caracteres '_' (*underscore*, ou sublinhado) ou '**$**' (cifrão). Os nomes não podem conter outros símbolos gráficos, operadores ou espaços em branco, podendo ser arbitrariamente longos, embora apenas os primeiros 32 caracteres sejam utilizados para distinguir nomes de diferentes variáveis. Como a linguagem Java é sensível às maiúsculas e minúsculas, é importante ressaltar que as letras minúsculas são consideradas diferentes das letras maiúsculas.

Exemplos de nomes válidos:

A	total	ct	$minimo	Salario
_Soma	TOT	Sm1	Maximo	Cod_Cliente

Exemplos de nomes inválidos:

1x	Total Geral	Salario**-**Minimo	public

regras violadas: começa com algarismo, usar espaço em branco e o operador de subtração e utilizar palavras reservadas.

Palavras Reservadas

Além das regras de formação de nomes, uma variável, ou qualquer outro nome a ser definido pelo programador, não pode ser uma das palavras reservadas da linguagem. As palavras reservadas são os nomes dos tipos primitivos, as diretivas da linguagem e quaisquer outros elementos pertencentes à “sintaxe” dessa linguagem.

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

Palavras reservadas, no entanto, não são utilizadas pela linguagem:

const	generic	inner	outer	var
future	goto	operator	rest	volatile

Declaração de Variáveis

Declarar uma variável é formalizar a associação tipo – nome – valor, e deve seguir a seguinte sintaxe:
<Tipo> <nome1> [, nome2 [, nome3 [… [, nomeN]]]];

Primeiro, é obrigatória a especificação de um Tipo e depois a declaração de uma (obrigatória) ou mais variáveis (opcional, como em uma lista em que os nomes são separados por vírgulas). A declaração é terminada por "**;**" (ponto-e-vírgula). Como tipo deve-se utilizar o nome de um tipo primitivo (int, long, float, …), de uma classe padrão da linguagem ou outra que o programador tenha definido.

```
int i;                              // declaração individual
double preco, total, salario;       // declaração em conjunto
boolean ehPrimo;
char sexo;
```

inicializando uma variável em sua declaração:

```
int i, sm = 0, ct = 0;      boolean ehPrimo = true;      char sexo = 'M';
```

Outras Considerações sobre Variáveis

As variáveis podem ser declaradas “em qualquer ponto” de um programa Java, sendo válidas em todo o escopo em que foram declaradas e nos escopos internos a estes.

Escopo = locais em que uma certa declaração tem validade.

O uso de variáveis ou objetos fora de seu escopo constitui um erro de programação detectado pelo compilador e que deve ser obrigatoriamente corrigido.

Variáveis declaradas dentro de métodos são denominadas “variáveis locais”, e o escopo das mesmas é entendido como local, seguindo as regras apresentadas.

Por outro lado, se as variáveis forem declaradas dentro do corpo da classe, elas serão entendidas como campos (*fields*) da classe, tendo então escopo “global”, isto é, serão válidas para toda a classe.

Demonstração de Escopo

O escopo de uma variável tem seu início a partir do "ponto" em que ocorrer a declaração dessa variável e "dentro" do bloco de comandos, que são delimitados por chaves ("**{**" e "**}**").

```
public class Exemplo2 {
  public static void main(String args[]) {
    // início do Bloco 1
    ini i = 0;
    {  // início do Bloco 2
       int j = 0;
       System.out.println(i); // dentro do escopo de i
       System.out.println(j); // dentro do escopo de j
       {  // início do Bloco 3
          System.out.println(k); // ERRO, a variável k ainda não foi declarada
          int k = 0;
          System.out.println(i); // dentro do escopo de i
          System.out.println(j); // dentro do escopo de j
          System.out.println(k); // dentro do escopo de k
       } // fim do Bloco 3
       System.out.println(i); // dentro do escopo de i
       System.out.println(j); // dentro do escopo de j
       System.out.println(k); // ERRO, fora do escopo de k
    } // fim do Bloco 2
    System.out.println(i); // dentro do escopo de i
    System.out.println(j); // ERRO, fora do escopo de j
  } // fim do Bloco 1
}
```

Convenções para Nomes de Variáveis

A “especificação” da linguagem Java tem regras bem definidas para a criação de identificadores, ou seja, nomes para os elementos definidos pelo programador. Lembrando, os nomes de variáveis podem ser compostos de uma ou mais letras, números e também os símbolos “_” (caracter sublinhado) e “$” (cifrão)

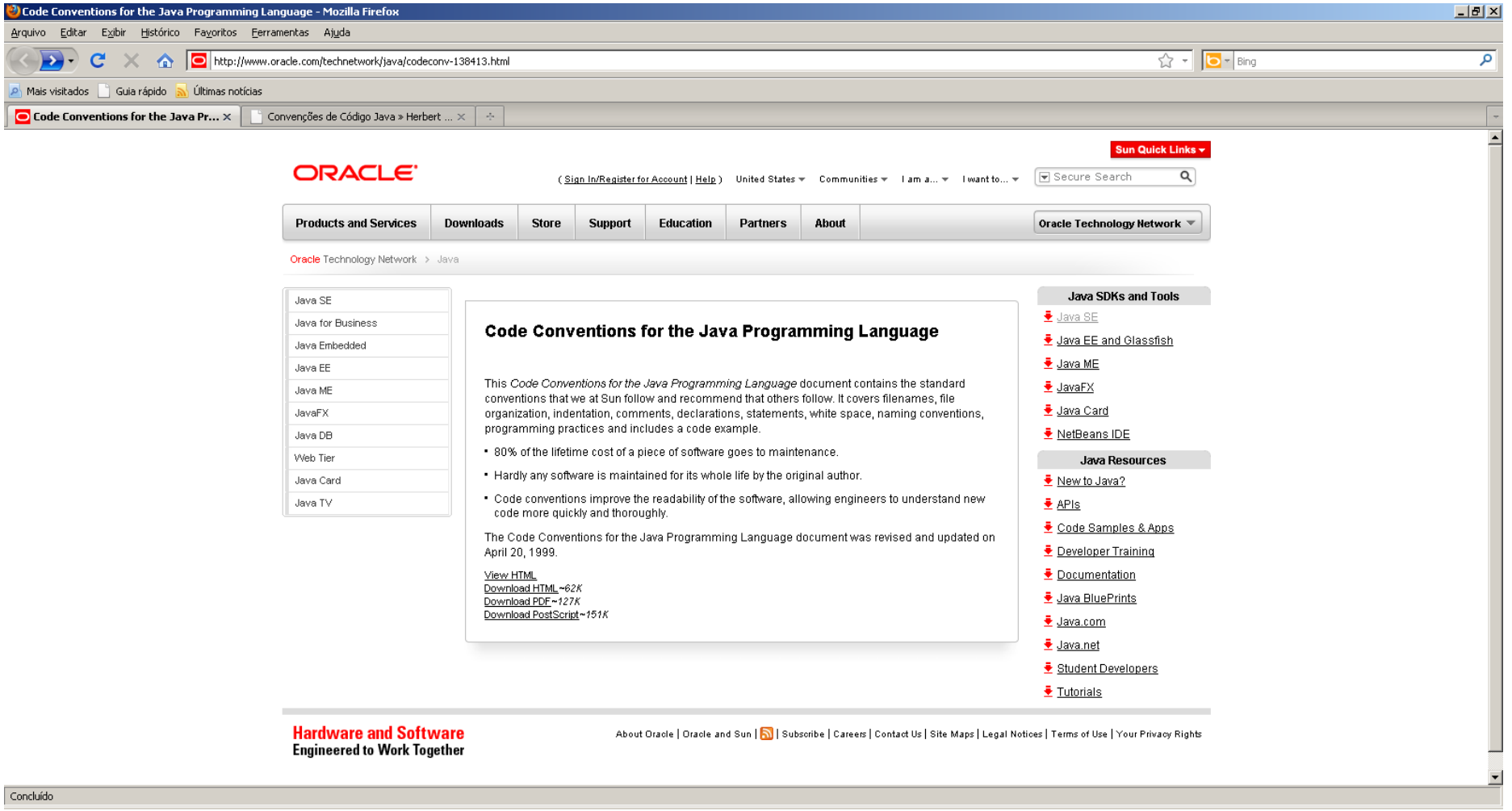
Para a declaração de variáveis, recomenda-se que os nomes sejam iniciados com letras minúsculas. Caso o nome seja constituído de uma única letra, sugere-se a inicial de seu próprio tipo, acompanhada eventualmente de números para diferenciar as variáveis. Caso o nome seja composto de mais de uma palavra, a primeira inicial deverá ser minúscula e as demais devem ser letras maiúsculas.

i	total	salario01
salarioMinimo	mediaAcademicoJava	

A utilização de caracteres numéricos no nome é livre, enquanto o uso do traço de sublinhar e o cifrão, embora permitido, não seja recomendado. A aplicação dessas recomendações faz com que o código tenha uma nomenclatura consistente com a API do Java, o que facilita seu entendimento por outros programadores.

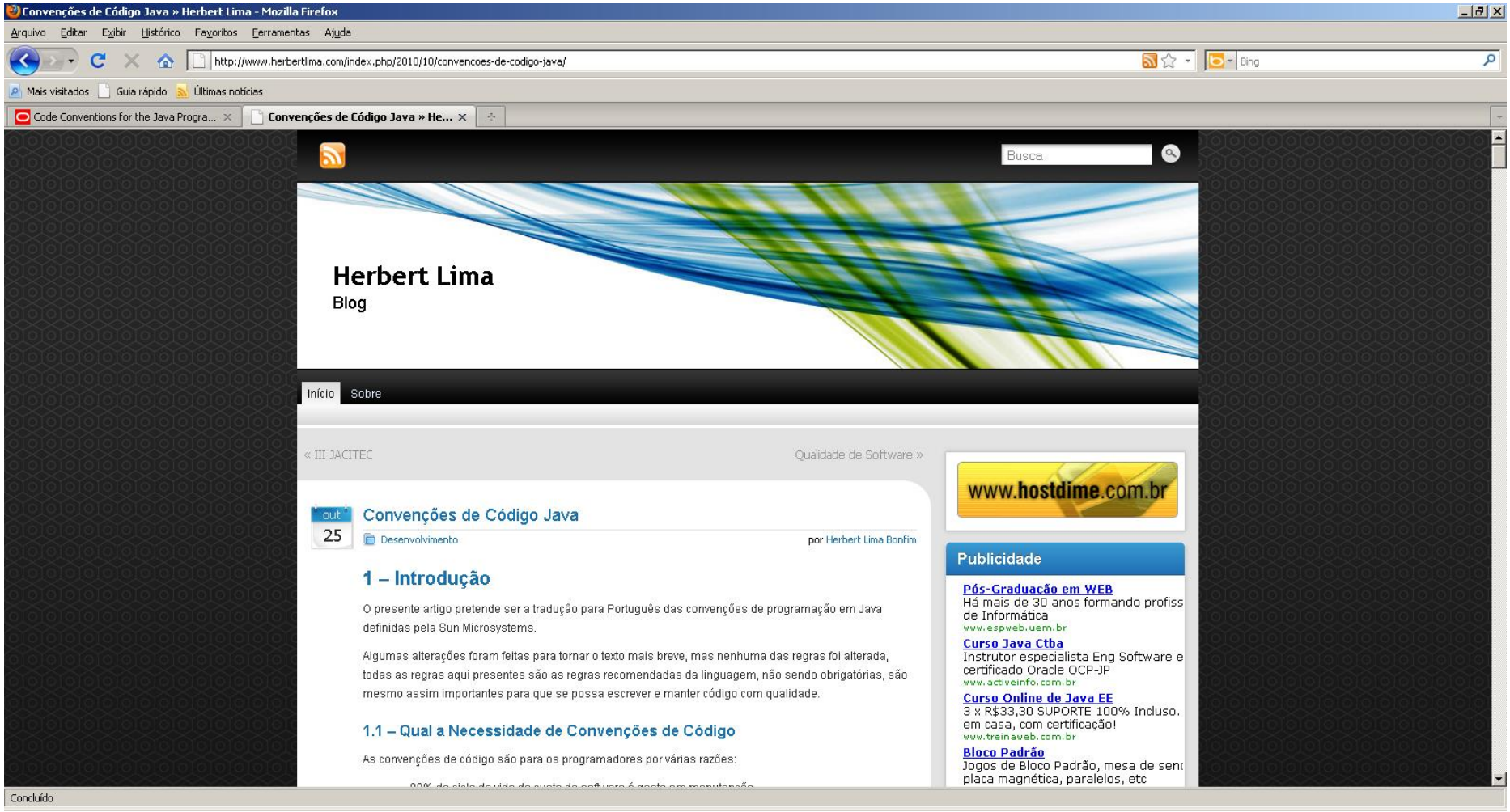
Convenções de Código Java (1/2)

http://www.oracle.com/technetwork/java/codeconv-138413.html



Convenções de Código Java (2/2)

http://www.herbertlima.com/index.php/2010/10/convencoes-de-codigo-java/



Operadores

A linguagem Java oferece um conjunto bastante amplo de operadores destinados à realização de operações de atribuição, aritméticas, lógicas e relacionais (são identicos aos encontrados em C/C++).

Operador de Atribuição (=)

A atribuição é a operação que permite definir o valor de uma variável, por meio de uma constante ou do resultado de uma expressão que envolve operações diversas, seguindo a seguinte sintaxe:

```
variável = expressão;
```

Exemplos de operações de atribuição:

```
boolean ehPrimo = true;
i = 0;
volume = base * (altura / 2);
```

Em Java, é válido o encadeamento de atribuições:

```
byte i, j, k;
i = j = k = 0;      // o que equivale a:      i = 0; j = 0; k = 0;
```

Operadores Aritméticos

Operador	Significado	Exemplo
+	adição	x + y
-	subtração	x - y
*	multiplicação	x * y
/	divisão	x / y
%	resto da divisão inteira	x % y
++	incremento unitário	++x ou y++
--	decremento unitário	--x ou y--

Os operadores de incremento unitário e decremento unitário permitem, respectivamente, acrescentar ou diminir uma unidade da variável operada, e tem uma característica adicional muito especial: podem ser posicionados antes ou depois de seus operadores, pois tal posicionamento influi no resultado das expressões em que figuram.

```
int x = 0, y;        // variável "x" passará a valer 1, enquanto a variável "y" receberá
y = ++x;             // o valor 1

int x = 0, y;        // variável "x" passará a valer 1, da mesma forma; mas a variável "y"
y = x++;             // receberá o valor 0, ou seja, o valor contido pela variável "x" anterior
                     // à realização da operação de incremento.
```

```
// Vários exemplos de operações aritméticas sobre variáveis

public class Aritmetica {

  public static void main(String args[]) {

  // Declaração e inicialização de duas variáveis "x", "y"
    int x = 5;
    int y = 2;

    System.out.println("x = " + x);                  // x = 5
    System.out.println("y = " + y);                  // y = 2
    System.out.println("-x = " + (-x));              // -x = -5
    System.out.println("x + y = " + (x + y));        // x + y = 7
    System.out.println("x - y = " + (x - y));        // x - y = 3
    System.out.println("x * y = " + (x * y));        // x * y = 10
    System.out.println("x / y = " + (x / y));        // x / y = 2
    // (double) x / y = 2.5
    System.out.println("(double) x / y = " + (double) x / y);
    System.out.println("x % y = " + (x % y));        // x % y = 1
    System.out.println("++x = " + (x = ++x));        // ++x = 6
    System.out.println("--y = " + (y = --y));        // --y = 1
  }
}
```

Operadores Relacionais

Os operadores relacionais permitem comparar valores, variáveis ou o resultado de expressoões retornando um “resultado do tipo lógico”, isto é, um resultado falso ou verdadeiro. Os operados relacionais disponíveis em Java são:

Operador	Significado	Exemplo
==	igual	x == y
!=	diferente	x != y
>	maior que	x > y
>=	maior ou igual a	x >= y
<	menor que	x < y
<=	menor ou igual a	x <= y

Note que o operador de igualdade é definido como um duplo sinal de igualdade (==), que não deve ser confundido com o operador de atribuição que corresponde a um sinal de igualdade (=).

Tanto o operador de igualdade (==) como o de desigualdade (!=) são semelhantes aos existentes na linguagem C/C++. Os demais são idênticos à grande maioria das linguagens de programação em uso.

Operadores Lógicos

Operador	Significado	Exemplo
`!`	`negação (not)`	`!a`
`&&`	`e lógico (and)`	`a && b`
`||`	`ou lógico (or)`	`a || b`

Os operadores lógicos duplos (**&&** e ||), isto é, definidos por dois caracteres, não podem conter espaços em branco entre os símbolos.

Esta observação também é válida para outros tipos de operadores da linguagem Java, como por exemplo, os operadores aritméticos e os operadores relacionais.

Operadores Compactos

Durante a programação, é muito comum a ocorrência de expressões do tipo:

```
i = i + 1;
ctHomens = ctHomens + 1;
totSalario = totSalario + salario;
```

Em todas estas expressões, é possível identificar a seguinte estrutura comum:

```
variável = variável operador expressão;
```

Ou seja, a variável que recebe o resultado da atribuição é utilizada como o primeiro termo da própria expressão que produz seu valor, indicando operações de totalização ou acumulação. Como forma de otimizar a execução do programa e reduzir o código escrito, podemos utilizar a notação compacta dessas expressões, que usa novos operadores denominados compactos ou de atribuição.

```
variável operador= expressão;
```

por exemplo,

```
i += 1;                  // com efeito semântico idêntico
ctHomens += 1;           // mas com execução mais rápida
totSalario += salario;
```

Precedência dos Operadores em Java

Como é possível construir uma expressão com diversos operadores diferentes, é necessário ter um critério para determinarmos qual desses operadores será primeiramente processado, tal como ocorre em expressões matemáticas comuns. O conceito de precedência representa esse critério, isto é, a ordem ou hierarquia de avaliação dos operadores.

Nível	Operador
1	.(seletor) [] ()
2	++ -- ~ instanceof new clone -(unário) (cast)
3	* / %
4	+ -
5	<< >> >>>
6	< > <= >=
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= operador=
15	,

Os operadores de um mesmo nível de precedência são avaliados conforme a ordem da esquerda para a direita. Os parênteses são operadores que podem ser utilizados para alterar a precedência natural, pois, primeiramente, avalia-se o conteúdo de um parêntese, antes de se proceder com o restante da avaliação de uma expressão.

Referência

- Introdução ao Java.
  - Peter Jandl Junior.
  - São Paulo: Berkeley - 2002.
  - Capítulo 2: Java e sua Sintaxe, pág. 19..36.