Java/Operadores

Operadores

Operadores são símbolos que representam atribuições, cálculos e ordem dos dados. As **operações** seguem uma ordem de prioridades, ou seja, alguns cálculos (ou outros) são processados antes de outros. Por exemplo, na Álgebra podemos mostrar a seguinte ordem:

Ordem	Operadores	Operação
1	/ *	Divisão e multiplicação
2	+ -	Soma e subtração

Assim, as operações de divisão e multiplicação, por serem de ordem 1, serão executadas antes das operações de soma e subtração (ordem 2). Também, as operações de divisão e multiplicação são de mesma ordem (1) e não importa, entre si, a ordem da operação (*2 dividido por 4 vezes 9* é igual a *2 vezes 9 dividido por 4*).

Tipo

Os operadores são divididos em 3 tipos em relação à quantidade de operandos no qual operam: unário, binário e ternário.

```
int a = 5, b = 2, c = 0;
a--;                    // -- é um operador unário pois opera apenas em a;
c = a * b;              // * é um operador binário pois opera em a e b.
c = c < 0 ? a : b;      // ?: é O operador ternário. Opera em na expressão booleana (c < 0), e em a ou b.
```

Precedência

Precedência indica a ordem na qual um operador opera em relação à avaliação de uma expressão.

A tabela seguinte elenca os operadores por precedência, do maior para o menor.

Tipo de Operador	Lista de Operadores
Sufixais	expr++ expr--
Prefixais	++expr --expr +expr -expr ~ !
Multiplicativos	* / %
Aditivos	+ -
Shift Binário	<< >> >>>
Comparativos	< > <= >= instanceof
Igualdade	== !=
Bit-aBit E	&
Bit-aBit XOU OR	^
Bit-aBit OU OR	|
Lógico E	&&
Lógico OU	||
Ternário	? :
Atribuição	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Precedência 13: operadores sufixais

São operadores unários posicionados após o identificador da variável para incrementar **++** ou decrementar **--** uma variável de tipo numérico em 1. Não podem ser utilizados em variáveis do tipo string, boolean ou de referência. Também não podem ser utilizados em valores de expressão e em literais. Diferente dos operadores de pré incremento/decremento, esses operadores sufixais retornam o valor original da variável para a expressão e depois realizam a operação sobre a variável.

```
int numero = 5;                         //A variável número é inicializada com 5.
System.out.println(numero++);           //É exibido 5, o valor original, e então a variável é
atualizada para 6.
System.out.println(numero);             //É exibido 6, valor incrementado na instrução anterior.
5
6
```

Precedência 12: operadores prefixais

São operadores unários que alteram o valor de uma variável e seus sinais são posicionados antes do identificador da variável. Como exemplo, pode-se citar o *Incremento* ++, *Decremento* --, *Sinal Positivo* +, *Sinal Negativo* -, *Inversão e Incremento* ~ e *Negação !*. O incremento e decremento, já vimos o que faz. Eles estão sendo citados aqui novamente porque seus sinais podem vir antes de variáveis também e numa operação complexa (com outros operadores binários) alteram a precedência da operação. O *Sinal Positivo* +retorna a variável que vem depois dele com o mesmo sinal, o *Sinal Negativo* - inverte o sinal de variáveis transformando números positivos em negativo e vice-versa. Ele não pode ser usado em variáveis dos tipos boolean e char. O *Incremento e Inversão* ~ aumenta o número em uma unidade e inverte o seu sinal. Só pode ser usado em inteiros. Já a operação de negação *!* transforma "verdadeiro" em "falso" e vice-versa, só podendo ser usado em variáveis do tipo boolean. Também só funcionam em variáveis, não em literais. Exemplos de uso:

```
int numero=5;          //numero contém 5
boolean ligado=false;   //ligado contém "falso"
++numero;              //numero agora vale 6
--numero;              //numero passa a valer 5
numero=+numero;        //numero continua valendo 5
numero=-numero;        //numero passa a valer -5
numero=~numero;        //numero passa a valer 4
ligado=!ligado;        //ligado passa a representar o valor "true"
```

Observação: uma diferença importante entre os operadores '++' e '--' prefixais e sufixais é o *tempo* de avaliação da expressão comparado com a alteração da variável. A saber:

```
int x = 5;          // x contém 5
int y, z;           // y e z não foram definidos
y = x++;            // primeiro faz y igual ao valor (anterior) de x, e depois modifica x
z = ++x;            // primeiro modifica x,  e depois atribui a z o novo valor de x
```

Neste exemplo, temos que, ao final *x* vale 7 (duas vezes incrementado), *y* vale 5 (o valor inicial de *x*) e *z* vale 7 (o valor final de *x*). Deve-se evitar usar mais de um operador prefixal e sufixal na mesma linha, porque isto torna o código incompreensível, por exemplo: *x = (y++ + ++z - --x) + ++y*.

Precedência 11: operadores multiplicativos

São operadores que realizam uma operação igual ou semelhante à multiplicação. Exemplos de operações do tipo são a Multiplicação (*), a Divisão (/) e o Resto (%). O primeiro pode realizar a multiplicação entre dois valores que não sejam do tipo boolean e nem do tipo char. O segundo pode dividir o primeiro número pelo segundo. Também não pode ser usado em valores booleans ou char. O terceiro retorna o resto da divisão do primeiro pelo segundo. Exemplos de uso:

```
int numero=5;        //numero passa a valer 5
numero=numero*4;     //numero assume o valor 20
numero=200/numero;   //numero assume o valor 10
numero=5%12;         //numero assume o valor 5
```

Precedência 10: operadores aditivos

São operadores que realizam alguma operação igual ou equivalente à adição. Assim como os Operadores Multiplicativos, os Aditivos podem ser usados tanto em variáveis como em literais (quando fazem a concatenação de strings). Mas também não podem ser usados em variáveis char e boolean. Eles também não alteram as variáveis passadas para eles. No lugar disso, eles retornam um número que deve ser direcionado par uma variável por meio da operação de atribuição (veja abaixo). Exemplos de uso:

```
int numero=5;           //numero passa a valer 5
numero=numero+8;        //numero passa a valer 13
numero=numero-numero;   //numero passa a valer zero
String x="Alo";         // x é inicializado com a string "Alo"
String y="Mundo!";      // y é inicializado com a string "Mundo!"
x = x + ", " + y;       // x passa a valer "Alo, Mundo!"
```

Precedência 9: operadores de shift

São operadores que deslocam os bits de um número de modo a alternar o seu valor. Exemplos de operadores deste tipo são o Shift para a Direita (>>), o Shift para a Direita Sem-Sinal(>>>) e o Shift para a Esquerda (<<). O primeiro valor a ser recebido pelo operador é o número sobre o qual será realizado um Shift e o segundo número é a quantidade de posições de bits a serem deslocados. Exemplos de uso:

```
int numero=-3;         //numero vale -3
numero=numero>>1;      //numero vale -2
numero=numero<<1;      //numero vale -4
numero=numero>>>1;     //numero vale 2147483646
numero=numero<<1;      //numero vale -4
```

Precedência 8: operadores comparativos

São operadores que comparam dois números e retornam em seguido o valor booleano "verdadeiro" ou "falso". Como exemplo, pode-se citar o Menor que(<), Maior que (>), Menor ou Igual que(<=), Maior ou Igual que (>=) e Exemplo de (instanceof). O significado dos quatro primeiros operadores é evidente. Já a operação Exemplo de, retorna "verdadeiro" se o primeiro operando for um Objeto pertencente à classe passada como segundo operando e "falso" caso contrário. Exemplos de uso:

```
boolean variavel;
variavel=(4<4);   //variavel recebe "falso"
variavel=(4<=4);  //variavel recebe "verdadeiro"
variavel=(-1>-3); //variavel recebe "verdadeiro"
variavel=(-4>=0); //variavel recebe "falso"
```

Precedência 7: operadores de igualdade

São semelhantes aos Operadores Comparativos. Eles também recebem números como operandos e retornam um valor boolean. A diferença é que estes operadores apenas verificam se as variáveis são iguais ou não. Como exemplos de operadores assim, pode-se citar o Igual a (==) e Diferente de (!=). Estes operadores podem ser usados em qualquer tipo de variável, desde que elas sejam do mesmo tipo. Exemplos de uso:

```
boolean variavel;
variavel=(-5==5);    //variavel recebe "falso"
variavel=(2!=45674); //variavel recebe "verdadeiro"
```

Ao utilizar operadores de igualdade com objetos, a comparação é feita entre suas referências. Dessa forma, dois objetos cognitivamente iguais, podem ser avaliados como diferentes. Exemplo:

```
1 class Pessoa{
```

```
2     String nome;
3
4     public Pessoa(String nome){
5         this.nome = nome;
6     }
7 }
8 new Pessoa("miguel") == new Pessoa("miguel") // comparação avaliada como falsa
```

Precedência 6, 5 e 4: operadores Bit-aBit

Os Operadores Bit-a-Bit são todos aqueles que realizam suas operações sobre os bits de um número, e não sobre o seu valor. Existem ao todo três destes operadores e cada um deles tem um valor de precedência diferente. O que tem precedência mais alta é o AND bit-a-bit (&). Ele analisa dois bits e retorna 1 se ambos forem iguais à 1 e 0 caso contrário. Depois vem o OR exclusivo bit-a-bit (^) que retorna 1 se os bits forem diferentes e 0 caso contrário. Por último, vem o operador OR inclusivo (|), que retorna 0 caso ambos os bits valerem 0 e retorna 1 caso contrário. Estes operadores podem ser usados em qualquer tipo de dados, desde que possuam o mesmo tamanho em bits. Exemplos de uso:

```
int numero;
numero=34&435;  //numero passa a valer 34
numero=34^46;   //numero passa a valer 12
numero=436|547; //numero passa a valer 951
```

Precedência 3 e 2: operadores AND e OR

Os operadores AND e OR só podem ser usados em variáveis e literais do tipo boolean. O operador AND (&&) retorna "verdadeiro" quando seus dois operandos também valem "verdadeiro" e retorna "falso" caso contrário. O operador OR (||) retorna "falso" quando seus dois operandos são falsos e retorna "verdadeiro" caso contrário. Exemplos de uso:

```
boolean variavel;
variavel=(2<45)&&(45<2)  //variavel passa a valer "falso"
variavel=(2<45)||(45<2)  //variavel passa a valer "verdadeiro"
```

Precedência 1: operadores ternários

O operador ternário **? :** recebe ao todo três operandos. O primeiro operando deve possuir necessariamente um valor do tipo boolean. Os outros dois operandos podem ser de qualquer tipo. Caso o valor do primeiro operando seja "verdadeiro", o operador retorna um valor igual ao do segundo operando. Caso o seu valor seja "falso", ele retorna um valor idêntico ao terceiro operando. Exemplos de uso:

```
int numero1=245;
int numero2=123;
numero1=(numero1>numero2)?numero1:numero2; /* Faz com que a variavel numero 1 receba sempreo
maior valor entre ela mesma e a numero2. Neste caso, ela
receberá o seu prório valor por ele ser maior*/
```

Precedência 0: atribuições

Os operadores de atribuição são os mais numerosos e os que tem uma prioridade menor de serem interpretados. Um exemplo deste operador (=)foi bastante usado neste capítulo. Ele armazena o valor que aparecer à direita na variável presente à esquerda. Caso deseje-se que a variável da esquerda receba o valor dela mesma após passar por alguma operação com um segundo valor, basta colocar o símbolo da operação antes do sinal "=" e colocar o segundo valor à direita. Exemplos de uso:

```
int numero = 3; //numero recebe o valor 3
```

```
numero += 7;    //numero recebe 3+7. Ou seja, 10.
numero -= 32;   //numero recebe o seu valor menos 32. Ou seja, -22.
numero %= -3;   //numero recebe o resto da divisão entre seu valor e -3. Ou seja, -1.
numero *= 6;    //numero recebe o seu valor vezes 6. Ou seja, -6.
numero /= 2;    //numero recebe o seu valor dividido por 2. Ou seja, -3.
```

Quando em uma mesma linha forem encontrados vários operadores diferentes, serão executados primeiro aqueles que tiverem maior precedência. Se existirem operadores com o mesmo valor de precedência, será realizado primeiro aquele cujo símbolo aparecer primeiro. É possível alterar a ordem natural com que são feitas as operações através do uso de parênteses. As operações entre parênteses sempre são realizadas antes.

Separadores

Os *separadores* são sinais que separam, ou sejam, indicam/modificam a ordem das operações (ou atribuições, ou interpretações etc.) que podem ou não ser diferentes da comum. Em Álgebra, temos alguns separadores como os seguintes:

Ordem	Separadores	Descrição
1	,	Vírgula
2	( )	Parênteses
3	[ ]	Colchetes
4	{ }	Chaves

Separadores em Java

Ordem	Separadores	Descrição
1	;	Ponto-e-vírgula
1	.	Ponto
1	,	Vírgula
1	( )	Parênteses
2	[ ]	Colchetes
2	{ }	Chaves

Ponto-e-vírgula, ponto e vírgula

O ponto-e-vírgula serve para separar sentenças ou instruções. A quebra de linha não separa instruções. Por exemplo:

```
int
A;
```

É o mesmo que:

```
int A;
```

E

```
int A;
float B;
```

É o mesmo que:

```
int A; float B;
```

O ponto serve para separar a parte inteira da fracionária em um número, tal como na notação inglesa. Ou seja, enquanto escrevemos 2,5 para representar dois e meio, em Java escrevemos:

```
2.5
```

A vírgula serve para separar itens, elementos, membros, como o que ocorre na atribuição de valores aos vetores, por exemplo:

```
int[] vetor;
vetor={34, 27, 3, 2};
int JANELAS=10, PORTAS=4;
class Casa implements Lavar, Pintar {
}
```

Parênteses, colchetes e chaves

Na maioria das linguagens de programação de computadores, os separadores colchetes e chaves são utilizados para outros propósitos diferentes dos em Álgebra. Para uso semelhante, em programação se utiliza o aninhamento (inclusão dentro de outros) de parênteses. Por exemplo:

Em Álgebra:

{13x(4+12)/[12+(13+76/2)x(1+2)]+5}

Equivale a, em Java:

(13*(4+12)/(12+(13+76/2)*(1+2))+5)

Em Java, as chaves são utilizadas para separar blocos de programação e os colchetes são utilizados para indicar/separar os índices de vetores (e, também, na declaração dos mesmos vetores).

Outros separadores/delimitadores

Através de uma visão mais ampla, podemos encontrar muitos outros símbolos que podem atuar como separadores em Java.

Quando atuam em dupla, podem ser chamados de delimitadores, tais como:

- aspas " " - são usadas para delimitar uma cadeia de caracteres;
- apóstrofos ' ' - são usados para delimitar um literal do tipo caracter (char).

Alguns outros tipos de separadores:

- e ou E - usados na notação científica de números fracionários;