Operadores

Sendo a linguagem Java uma das linguagens mais utilizadas e famosas, conseguimos realizar diversas ações e aproveitar de muitos recursos. Realizar operações com a linguagem Java não é complexo; se você tem conhecimento básico de matemática e um pouco de lógica, você se dará bem.

Na aula de hoje, vamos aprender a fazer algumas operações com Java, e quem sabe você até mesmo consiga usar o Scanner, que aprendemos na aula passada, para realizar as operações matemáticas.

1. Introdução

Operador em Java é um símbolo que é usado para executar operações, como por exemplo, operações de adição, subtração, multiplicação ou divisão.

Entre os operadores, estão: aritmético, relacional, lógico, atribuição, entre outros. Veja abaixo o quadro de precedência de operações em Java:

|  |  |  |
| --- | --- | --- |
| **Tipo de Operador** | **Categoria** | **Precedência** |
| Unário | Sufixos | Expressão++  Expressão-- |
| Prefixos | ++Expressão  --Expressão  +Expressão  -Expressão ~ ! |
| Aritmética | Multiplicativo | \* / % |
| Aditivo | + - |
| Mudança | Mudança | << >> >>> |
| Relacional | Comparação | < > <= >= instanceof |
| Igualdade | == != |
| Bit a bit | Bit a bit AND | & |
| Bit a bit OR exclusivo | ^ |
| Bit a bit OU inclusive | | |
| Lógico | AND lógico | && |
| OR lógico | || |
| Ternário | Ternário | ?: |
| Atribuição | Atribuição | = += -= \*= /= = = ^= |= <<= >>= >>>= |

Vamos aprender sobre alguns operadores.

1.1 Aritmética

Os operadores Aritméticos obedecem às regras comuns da matemática:

* As expressões contidas dentro de parênteses são resolvidas antes das expressões fora dos parênteses;
* Quando há um parente dentro de outros parênteses, a solução é realizada sempre dentro do parêntese mais interno para o externo;
* Quando duas ou mais expressões tiveram a mesma prioridade, a solução é sempre iniciada da esquerda para a direita.

Veja abaixo um exemplo de como realizar operações aritméticas:

package Operadores;

import java.util.Scanner;

public class OperadoresAritmeticos

public static void main(String[] args) {

int x = 4;

int y = 2;

int z = 3;

int w = -5;

System.out.println("Soma (4+2): " + (x + y));

System.out.println("\nSoma com número negativo (4 + (-5)): " + (x + w));

System.out.println("\nSubtração (4-2): " + (x - y));

System.out.println("\nMultiplicação (4\*2): " + (x \* y));

System.out.println("\nDivisão (4/2): " + (x / y));

System.out.println("\nMódulo (resto da divisão): " + (x % 2));

System.out.println("\nMódulo (resto da divisão): " + (z % 2));

}

}

1.2 Strings

Ao realizar operações com Strings, não realizamos operações matemáticas, porque em muitos casos, trabalhamos com uma cadeia de caracteres, onde muitos não podem ser somados.

Diferentemente de somar números, com as Strings realizamos o chamado concatenar, onde realizamos a união de dois ou mais dados atribuídos às variáveis.

Veja o exemplo abaixo:

package Operadores;

public class OperadoresString {

public static void main(String[] args) {

String soberano = "São Paulo";

String timao = "Corinthians";

System.out.println(soberano + timao);

System.out.println(timao + soberano);

System.out.println(soberano + " é maior do que " + timao);

}

}

No exemplo acima, percebemos que a primeira e segunda saída dos dados estão juntas (“São PauloCorinthians” e “CorinthiansSão Paulo”), mas quando aprendemos a concatenar as variáveis, fica tudo mais organizado (“Quem nasceu Corinthians nunca será maior do que São Paulo”).

1.3 Unário

Esses operadores requerem apenas um operando. São utilizados para executar várias operações, seja incrementando ou decrementando um valor, negando uma expressão e invertendo o valor de um booleano.

Veja um exemplo abaixo:

package Operadores;

public class Unario {

public static void main(String[] args) {

int a = 10;

System.out.println(a++);

System.out.println(++a);

System.out.println(a--);

System.out.println(--a);

}

}

No exemplo acima, temos uma variável que recebe o valor 10.

Na primeira linha temos um operador pós-incremento, onde a variável A terá o seu valor impresso após a impressão, sendo o seu novo valor o número 11 porque a instrução usada “a++” pede ao programa para acrescentar mais um número.

Na segunda linha temos um operador pré-incremento, onde a variável A terá o seu valor incrementado antes da impressão, sendo o seu novo valor o número 12 porque a instrução usada “++a” pede para o programa acrescentar mais um número. Tenha atenção na lógica: o valor da variável passa a ser 12 porque na primeira linha á foi acrescentado o valor de 1 a variável A.

Ficou um pouco confuso? Então vamos tentar deixar mais fácil de entender e resumir o que foi feito na operação:

**Pós-incremento (a++)**: Retorna o valor original de a e, em seguida, incrementa a em 1.

**Pré-incremento (++a)**: Incrementa a em 1 e, em seguida, retorna o valor incrementado.

**Pós-decremento (a--)**: Retorna o valor original de a e, em seguida, decrementa a em 1.

**Pré-decremento (--a)**: Decrementa a em 1 e, em seguida, retorna o valor decrementado.

Vamos fazer outro exemplo para praticar, mas esse vai ser um pouquinho mais difícil:

package Operadores;

public class Unario {

public static void main(String[] args) {

int c = 10;

int d = -10;

boolean e = true;

boolean f = false;

// Invertendo o sinal e subtrai com 1

System.out.println(~c);

// Inverte o sinal e subtrai com 1

System.out.println(~d);

// Falso - oposto do valor booleano

System.out.println(!e);

// Verdadeiro - oposto do valor booleano

System.out.println(!f);

}

}

O resultado será: -11, 9, false, true.

Vamos a explicação deste resultado:

Na primeira linha temos o sinal “~c” que inverterá os valores em bit. Por exemplo, o número 10 em binário passa a ser 0000 1010; após a conversão, o número binário passa a inverter, ficando 1111 0101, que corresponde ao número -11.

Na segunda linha temos o sinal “~d” que inverterá os valores em bit. Por exemplo, o número -10 em binário passa a ser 1111 0110; após a conversão, o número binário passa a inverter, ficando 000 1001, que corresponde ao número 9.

Nas linhas quatro e cinco temos os seguintes sinais: “!e” e “!f”, que inverte os valores booleanos, onde E passa a ser false e F passam a ser true.

Conclusão

Ao realizar operações com linguagens de programação, notamos que a lógica faz parte do dia a dia de um programador. O ambiente da programação é muito rico e amplo, onde podemos realizar, de maneiras diferentes, a mesma atividade ou realizar atividades diferentes.

Os operadores ajudam os profissionais a diminuir linhas de códigos e na tomada de decisão. Você pode praticar o que aprende nas aulas, e aproveitar para unir o que você aprendeu nas últimas aulas sobre a linguagem Java.

Não se esqueça que este é um conteúdo básico, e mais conteúdos devem ser consultados em livros, vídeos e fóruns na internet. Tire tempo, pesquise, estude bem, assim será mais fácil você absorver o conteúdo dessa linguagem.

Referências

Java: operadores de atribuição, aritméticos, relacionais e lógicos. Acesso em 31 de maio de 2024

https://www.devmedia.com.br/java-operadores-de-atribuicao-aritmeticos-relacionais-e-logicos/38289

Operadores lógicos e matemáticos da linguagem Java. Acesso em 31 de maio de 2024.

https://www.devmedia.com.br/operadores-logicos-e-matematicos-da-linguagem-java/25248