Sumario

[Operadores 3](#_Toc130843484)

[Estruturas condicionais 4](#_Toc130843485)

[A Estrutura Condicional If 4](#_Toc130843486)

[Switch 5](#_Toc130843487)

[As Palavras Reservadas – Case, Default e Break 7](#_Toc130843488)

[Laços de Repetição 7](#_Toc130843489)

[While Loop 7](#_Toc130843490)

[For Loop 8](#_Toc130843491)

[For Loop Aninhado 8](#_Toc130843492)

[For-each ou Enhanced For Loop 8](#_Toc130843493)

[O Uso da Palavra-Chave Continue 9](#_Toc130843494)

# Operadores

Em Java, um operador é um símbolo ou conjunto de símbolos que é usado para realizar uma operação em um ou mais valores. Eles podem ser usados para realizar cálculos matemáticos, comparar valores, atribuir valores a variáveis e muito mais.

**Operadores Aritméticos**: São usados para realizar operações matemáticas básicas, como adição (+), subtração (-), multiplicação (\*), divisão (/) e resto da divisão (%) entre dois valores numéricos. Por exemplo: int resultado = 5 + 3;

**Operadores de Atribuição**: São usados para atribuir valores a variáveis. O operador de atribuição básico é o =, mas também há operadores compostos, como += (atribuição com adição), -= (atribuição com subtração), \*= (atribuição com multiplicação), /= (atribuição com divisão) e %= (atribuição com resto da divisão). Por exemplo: int a = 2; a += 3; // o valor de 'a' agora é 5

**Operadores Relacionais**: São usados para comparar valores e retornar um valor booleano (verdadeiro ou falso). Alguns exemplos de operadores relacionais são >, <, >=, <=, == (igualdade) e != (diferente). Por exemplo: boolean resultado = (5 > 3); // o valor de 'resultado' será 'true'

**Operadores Lógicos**: São usados para combinar expressões booleanas e retornar um valor booleano. Os principais operadores lógicos são && (e lógico), || (ou lógico) e ! (negação lógica). Por exemplo: boolean resultado = (5 > 3) && (2 < 4); // o valor de 'resultado' será 'true'

**Operadores Bitwise**: São usados para realizar operações de bits em valores numéricos. Alguns exemplos de operadores bitwise são & (and bitwise), | (or bitwise) e ^ (xor bitwise). Por exemplo: int resultado = 5 & 3; // o valor de 'resultado' será 1

**Operadores Shift**: São usados para deslocar os bits de um valor numérico para a esquerda ou para a direita. Alguns exemplos de operadores de deslocamento são << (deslocamento para a esquerda), >> (deslocamento para a direita com preenchimento de sinal) e >>> (deslocamento para a direita sem preenchimento de sinal). Por exemplo: int resultado = 5 << 2; // o valor de 'resultado' será 20

**Operadores Ternários**: São usados para fazer uma avaliação condicional em uma única linha de código. O operador ternário é representado por ? :, e tem a forma condição ? expressão1 : expressão2. Por exemplo: int resultado = (5 > 3) ? 1 : 0; // o valor de 'resultado' será 1

**Operador de Instânciaof**: O operador instanceof é usado para verificar se um objeto é uma instância de uma determinada classe ou de uma de suas subclasses. Ele retorna um valor booleano true se o objeto for uma instância da classe especificada e false caso contrário. Por exemplo: if (objeto instanceof MinhaClasse) { // fazer alguma coisa }

**Operador de Referência de Método**: O operador de referência de método :: é usado para referenciar um método existente sem invocá-lo. Ele é geralmente usado em expressões lambda e em programação funcional. Existem três tipos de referência de método: referência de método para um método estático, referência de método para um método de instância de um objeto específico e referência de método para um método de instância de um tipo arbitrário. Por exemplo: lista.forEach(System.out::println);

**Operador de Lançamento** (Casting): O operador de lançamento é usado para converter um tipo de dado em outro tipo. Existem dois tipos de lançamento em Java: o lançamento implícito, que ocorre automaticamente em algumas situações, e o lançamento explícito, que é feito pelo programador usando os operadores de lançamento. Por exemplo: double d = 1.23; int i = (int) d;

**Operador de Condicional** **Nulo**: O operador ?. é usado para chamar um método ou acessar um campo em um objeto que pode ser nulo. Ele retorna null se o objeto for nulo, em vez de lançar uma exceção NullPointerException. Por exemplo: String resultado = objeto?.getNome();

**Operador de Concatenação** **de Strings**: O operador + pode ser usado para concatenar duas strings. Quando pelo menos um dos operandos é uma string, o operador + é usado para concatenar as duas strings. Por exemplo: String nome = "João"; String sobrenome = "Silva"; String nomeCompleto = nome + " " + sobrenome;

# Estruturas condicionais

Uma estrutura condicional em Java é uma forma de controle de fluxo que permite que o programa execute diferentes instruções com base em uma ou mais condições.

## A Estrutura Condicional If

if (condicao) {

// código a ser executado se a condição for verdadeira

}

A "condicao" pode ser qualquer expressão booleana, ou seja, uma expressão que é avaliada como "verdadeira" ou "falsa". Se a condição for verdadeira, o código dentro do bloco de chaves {} é executado; caso contrário, o bloco é ignorado.

Exemplo:

int nota = 67;

if(nota >= 60) {

System.out.prinln(“Passou de Ano”);

}

Também é possível adicionar blocos de código alternativos a serem executados se a condição do "if" não for verdadeira. Isso é feito com a palavra-chave "else", que é seguida por outro bloco de código. A sintaxe básica do "if-else" é a seguinte:

if (condicao) {

// código a ser executado se a condição for verdadeira

} else {

// código a ser executado se a condição não for verdadeira

}

Exemplo:

int nota = 59;

int frequencia = 85;

int recuperação = 61;

if(nota >= 60 && frequencia >= 75) {

System.out.println(“Passou de Ano”);

} else if((nota+recuperacao)/2 >= 60) {

System.out.println(“Passou de Ano”);

} else {

System.out.println(“Reprovou”);

}

Há também a possibilidade de adicionar múltiplas condições usando a palavra-chave "else if". Isso permite que o programa execute diferentes blocos de código com base em uma série de condições. A sintaxe básica do "if-else if-else" é a seguinte:

if (condicao1) {

// código a ser executado se a condição1 for verdadeira

} else if (condicao2) {

// código a ser executado se a condição2 for verdadeira

} else {

// código a ser executado se nenhuma das condições anteriores for verdadeira

}

O "if" aninhado é uma outra estrutura condicional que pode ser utilizada em Java. Ela permite que um "if" seja colocado dentro do bloco de código de outro "if". A sintaxe básica do "if" aninhado é a seguinte:

if (condicao1) {

if (condicao2) {

// código a ser executado se a condicao1 e a condicao2 forem verdadeiras

} else {

// código a ser executado se a condicao1 for verdadeira, mas a condicao2 não

}

} else {

// código a ser executado se a condicao1 não for verdadeira

}

No exemplo acima, o primeiro "if" verifica a "condicao1" e, se for verdadeira, entra no bloco de código. Dentro desse bloco, há um segundo "if" que verifica a "condicao2". Se a "condicao2" for verdadeira, o bloco de código dentro do segundo "if" será executado. Se a "condicao2" não for verdadeira, o bloco de código dentro do "else" do segundo "if" será executado. Se a "condicao1" não for verdadeira, o bloco de código dentro do "else" do primeiro "if" será executado.

O "if" aninhado pode ser útil quando precisamos verificar mais de uma condição em um determinado momento. No entanto, é importante tomar cuidado com o uso excessivo dessa estrutura, pois ela pode tornar o código difícil de ler e entender.

## Switch

O "switch" é outra estrutura de controle de fluxo que permite que o programa execute diferentes blocos de código com base em um valor específico de uma variável. O "switch" é usado principalmente quando há múltiplas condições a serem avaliadas em um único bloco de código, o que tornaria o uso de vários "if-else" desnecessário e pouco eficiente.

A sintaxe básica do "switch" é a seguinte:

switch (variavel) {

case valor1:

// código a ser executado se variável for igual a valor1

break;

case valor2:

// código a ser executado se variável for igual a valor2

break;

case valor3:

// código a ser executado se variável for igual a valor3

break;

default:

// código a ser executado se nenhum dos casos anteriores for verdadeiro

}

A variável é o valor que será avaliado, enquanto os "casos" representam os diferentes valores possíveis que essa variável pode ter. Se a variável for igual a um dos valores listados em um dos casos, o bloco de código correspondente será executado. É importante notar que, assim como o "if", o "switch" também suporta o uso da palavra-chave "break", que é usada para sair do bloco de código e continuar a execução após o "switch". O "default" é opcional e é executado se nenhum dos casos anteriores for verdadeiro.

Um exemplo simples de uso do "switch" é a conversão de um número inteiro em um mês do ano:

int mes = 3;

String nomeMes;

switch (mes) {

case 1:

nomeMes = "Janeiro";

break;

case 2:

nomeMes = "Fevereiro";

break;

case 3:

nomeMes = "Março";

break;

case 4:

nomeMes = "Abril";

break;

case 5:

nomeMes = "Maio";

break;

case 6:

nomeMes = "Junho";

break;

case 7:

nomeMes = "Julho";

break;

case 8:

nomeMes = "Agosto";

break;

case 9:

nomeMes = "Setembro";

break;

case 10:

nomeMes = "Outubro";

break;

case 11:

nomeMes = "Novembro";

break;

case 12:

nomeMes = "Dezembro";

break;

default:

nomeMes = "Mês inválido";

}

System.out.println("O mês correspondente ao número " + mes + " é " + nomeMes);

No exemplo acima, a variável "mes" é igual a 3, então o bloco de código correspondente ao "case 3" será executado, resultando em "nomeMes" sendo definido como "Março".

### As Palavras Reservadas – Case, Default e Break

Acima é possível observar três palavras reservadas que são primordiais para a estrutura do comando switch.

O **case** indica um bloco de código no switch.

O **default** é apenas executado quando nenhum caso corresponde ao valora da variável inserida, ele não é necessário, porém é muito útil pois pode alcançar alguma condição específica ou evitar erros posteriores.

O **break** é utilizado para interromper um for loop, um while loop ou um switch, quando lançado para imediatamente o bloco de código, se houver algo código abaixo dele, este não será executado.

# Laços de Repetição

Os laços de repetição, também conhecidos como loops, são estruturas de controle de fluxo em linguagens de programação que permitem que um bloco de código seja executado repetidamente, enquanto uma condição definida for verdadeira. Isso é útil quando queremos executar a mesma operação várias vezes, sem a necessidade de repetir manualmente o código várias vezes.

## While Loop

O laço de repetição while é usado para executar um bloco de código repetidamente enquanto a condição definida no início do laço é verdadeira. A sintaxe básica é:

while (condição) {

// código a ser repetido

}

O bloco de código dentro do laço while só é executado se a condição definida no início do laço for verdadeira. Caso contrário, o bloco de código não será executado e a execução do programa continuará com a próxima instrução após o laço.

Do While Loop

Já o laço de repetição do-while é semelhante ao while, mas com uma diferença importante: o bloco de código é executado pelo menos uma vez, independentemente da condição definida no início do laço. A sintaxe básica é:

do {

// código a ser repetido

} while (condição);

Nesse caso, o bloco de código dentro do laço é executado primeiro, e depois a condição é verificada. Se a condição for verdadeira, o bloco de código é executado novamente. Caso contrário, a execução do programa continua com a próxima instrução após o laço.

O laço do-while é útil quando queremos garantir que um bloco de código seja executado pelo menos uma vez, independentemente da condição definida no início do laço.

## For Loop

Os loops são estruturas fundamentais em programação que permitem repetir um conjunto de instruções várias vezes. Existem vários tipos de loops em Java, sendo os mais comuns o for loop, o for loop aninhado e o enhanced for loop (ou for-each loop).

O for loop é utilizado para percorrer uma sequência de valores (normalmente números inteiros) e executar um conjunto de instruções para cada um desses valores. A sintaxe básica do for loop em Java é a seguinte:

for (inicialização; condição; incremento) {

// instruções a serem executadas

}

A inicialização define uma variável de controle e seu valor inicial, a condição é uma expressão booleana que determina se o loop deve continuar ou não, e o incremento é uma expressão que é executada após cada iteração do loop, geralmente incrementando a variável de controle.

### For Loop Aninhado

O for loop aninhado é uma extensão do for loop que permite percorrer uma sequência de valores em várias dimensões, ou seja, um loop dentro de outro. A sintaxe básica do for loop aninhado em Java é a seguinte:

for (inicialização1; condição1; incremento1) {

for (inicialização2; condição2; incremento2) {

// instruções a serem executadas

}

}

Nesse caso, o loop interno é executado para cada iteração do loop externo. Isso é útil, por exemplo, para percorrer uma matriz bidimensional.

### For-each ou Enhanced For Loop

Por fim, o enhanced for loop (ou for-each loop) é uma sintaxe mais simples para percorrer uma sequência de valores, especialmente para coleções como arrays ou listas. A sintaxe básica do enhanced for loop em Java é a seguinte:

for (tipo variável : sequência) {

// instruções a serem executadas

}

Nesse caso, a variável é atribuída a cada elemento da sequência, um de cada vez, e o loop é executado para cada elemento. Isso é útil para percorrer uma lista de objetos sem a necessidade de acessá-los pelo índice.

### O Uso da Palavra-Chave Continue

Em Java, continue é uma palavra-chave usada dentro de loops (por exemplo, for, while ou do-while) para pular a iteração atual e continuar com a próxima iteração do loop.

Quando o comando continue é executado dentro de um loop, o programa irá ignorar qualquer código que esteja abaixo dele dentro do bloco de instruções do loop, e imediatamente passar para a próxima iteração do loop, sem executar as instruções restantes na iteração atual.

A principal finalidade do continue é permitir que o programador evite a execução de certas instruções durante uma iteração do loop, sem ter que encerrar completamente o loop. Em outras palavras, o continue permite ao programador pular o processamento de certas condições que não atendem a certas exigências, melhorando a eficiência do programa.

Um exemplo simples de uso do continue seria um loop que itera sobre uma lista de números inteiros e precisa encontrar somente os números pares:

for (int i = 1; i <= 10; i++) {

if (i % 2 != 0) {

continue; // pula a iteração atual se i for ímpar

}

System.out.println(i + " é um número par.");

}

Nesse exemplo, a condição dentro do loop verifica se o número i é ímpar e, se for, o comando continue é executado, saltando a iteração atual e passando para a próxima. Como resultado, apenas os números pares são impressos na tela.