**Instruções em Java**

As instruções em Java são comandos ou blocos de código que controlam o fluxo de execução de um programa, permitindo que ele tome decisões, repita ações e manipule dados de forma estruturada e eficiente.

Neste módulo, vamos explorar as principais instruções e estruturas de controle de fluxo em Java. Elas são fundamentais para o desenvolvimento de programas eficientes e dinâmicos. O conteúdo está dividido em:

* estruturas condicionais,
* estruturas de seleção,
* laços de repetição e
* manipulação de vetores e matrizes.

**Estruturas Condicionais**

As estruturas condicionais em Java permitem a execução de blocos de código diferentes com base em determinadas condições. São essenciais para controlar o fluxo do programa. São estruturas simples e úteis para garantir que o código responda de maneira apropriada a diferentes cenários. A seguir, são apresentadas as principais estruturas condicionais:

**1. Estrutura IF**: Avalia uma condição e executa o bloco de código caso a condição seja verdadeira.

*int idade = 18;  
if (idade >= 18) {  
    System.out.println(“Você é maior de idade.”);  
}*

Neste exemplo, se a variável idade for maior ou igual a 18, a mensagem “Você é maior de idade.” será exibida**.**

**2. Estrutura IF-ELSE**: Caso a condição não seja atendida, um segundo bloco de código é executado.

*int idade = 16;  
if (idade >= 18) {  
    System.out.println(“Você é maior de idade.”);  
} else {  
    System.out.println(“Você é menor de idade.”);  
}*

Se a variável idade for menor que 18, o programa exibirá “Você é menor de idade.”.

**3. Estrutura** **IF- ELSE IF -ELSE**: Permite testar múltiplas condições em sequência, com um bloco de código executado para cada condição atendida.

*int nota = 85;  
if (nota >= 90) {  
    System.out.println(“Excelente!”);  
} else if (nota >= 70) {  
    System.out.println(“Bom trabalho!”);  
} else {  
    System.out.println(“Precisa melhorar.”);  
}*

Aqui, o código verificará se nota é maior ou igual a 90, entre 70 e 89, ou menor que 70, e exibirá uma mensagem correspondente a cada faixa de valor.  
Esses exemplos demonstram como utilizar as estruturas condicionais em Java para controlar o fluxo de um programa com base em diferentes condições. Ao dominar essas construções, você estará apto a criar programas que respondem de forma adequada a diferentes situações e requisitos.

**Estruturas de Seleção**

A estrutura switch permite selecionar um bloco de código a ser executado com base no valor de uma variável, facilitando o controle de fluxo quando há várias opções específicas. Essa estrutura é ideal para cenários em que é necessário avaliar múltiplos valores de uma única variável e é uma alternativa ao uso de múltiplos ‘if-else’, oferecendo uma sintaxe mais limpa e eficiente em alguns casos. É preciso avaliar a necessidade de uso para cada caso.

No exemplo abaixo, a variável diaDaSemana é avaliada e o programa imprime o nome do dia correspondente ao valor fornecido. Se o valor não estiver entre 1 e 7, a instrução default exibirá “Dia inválido”.

Ao contrário de uma sequência de ‘if-else’, que pode se tornar extensa e difícil de ler, o switch proporciona uma sintaxe simplificada para lidar com múltiplas opções específicas. Seu uso é recomendado quando há muitas condições baseadas no valor de uma variável, tornando o código mais organizado e de fácil manutenção.

*public class ExemploSwitch {  
    public static void main(String[] args) {  
        int diaDaSemana = 3;  
  
        switch (diaDaSemana) {  
            case 1:  
                System.out.println(“Domingo”);  
                break;  
            case 2:  
                System.out.println(“Segunda-feira”);  
                break;  
            case 3:  
                System.out.println(“Terça-feira”);  
                break;  
            case 4:  
                System.out.println(“Quarta-feira”);  
                break;  
            case 5:  
                System.out.println(“Quinta-feira”);  
                break;  
            case 6:  
                System.out.println(“Sexta-feira”);  
                break;  
            case 7:  
                System.out.println(“Sábado”);  
                break;  
            default:  
                System.out.println(“Dia inválido”);  
                break;  
        }  
    }  
}*

**Instruções de Repetição**

As instruções de repetição são usadas para executar blocos de código repetidamente enquanto uma condição for verdadeira. Essas estruturas são fundamentais para o controle de fluxo em laços de repetição. Em Java, as principais estruturas de repetição são:

**1. Estrutura FOR:** Executa um bloco de código um número específico de vezes, normalmente usado quando o número de iterações é conhecido.

*public class ExemploRepeticoes {  
    public static void main(String[] args) {  
        // Exemplo de FOR - imprime números de 1 a 5  
        System.out.println(“Exemplo FOR:”);  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(i);  
        }  
}*

Aqui, o número de iterações é conhecido antecipadamente (números de 1 a 5).

**2. Estrutura WHILE:** Executa o bloco de código enquanto uma condição for verdadeira, sendo mais flexível em relação ao número de repetições.

*public class ExemploRepeticoes {  
    public static void main(String[] args) {  
       // Exemplo de WHILE - imprime números de 1 a 5  
        System.out.println(“\nExemplo WHILE:”);  
        int j = 1;  
        while (j <= 5) {  
            System.out.println(j);  
            j++;  
        }  
}*

Neste caso, o laço continua até que a condição j <= 5 não seja mais verdadeira.

**3. Estrutura DO-WHILE**: Similar ao ‘while’, mas garante que o bloco de código seja executado pelo menos uma vez antes da condição ser verificada.

*public class ExemploRepeticoes {  
    public static void main(String[] args) {  
        // Exemplo de DO-WHILE - imprime números de 1 a 5  
        System.out.println(“\nExemplo DO-WHILE:”);  
        int k = 1;  
        do {  
            System.out.println(k);  
            k++;  
        } while (k <= 5);  
    }  
}*

E neste exemplo, o bloco de código é executado pelo menos uma vez, mesmo que a condição seja falsa inicialmente.

As instruções de repetição são fundamentais para automatizar tarefas repetitivas em Java. Compreender quando e como utilizar cada uma dessas estruturas permite que você crie programas mais eficientes para lidar com repetições.

**Manipulação de Vetores**

Vetores em Java são coleções de elementos do mesmo tipo. A manipulação de vetores pode ser feita por meio de loops (repetições) que percorrem todos os elementos. Um exemplo básico é o uso do laço ‘for’ para iterar sobre cada elemento do vetor.

*int[] numeros = {1, 2, 3, 4, 5};  
for (int i = 0; i < numeros.length; i++) {  
    System.out.println(numeros[i]);  
}*

Outra forma de percorrer o vetor é utilizando o ‘for-each’, que simplifica a iteração:

*for (int numero : numeros) {  
    System.out.println(numero);  
}*

Além disso, é possível acessar o comprimento de um vetor usando a propriedade ‘*length*’, como foi no código anterior em *numeros.length.*

**Manipulação de Matrizes**

Matrizes são vetores bidimensionais e permitem a organização de dados em linhas e colunas. A seguir, um exemplo básico de criação e manipulação de matrizes:

int[][] matriz = {  
    {1, 2, 3},  
    {4, 5, 6}  
};  
System.out.println(matriz[0][1]); // acessa o elemento na primeira linha e segunda coluna

Você também pode modificar valores específicos e percorrer todos os elementos da matriz usando repetições aninhadas:

for (int i = 0; i < matriz.length; i++) {  
    for (int j = 0; j < matriz[i].length; j++) {  
        System.out.print(matriz[i][j] + " ");  
    }  
    System.out.println();  
}

**Instruções de Controle de Repetição**

As instruções ‘*break*’ e ‘continue’ são usadas para controlar o comportamento das repetições em Java:

**1. Instrução *break***: Encerra o código, seja ele uma repetição ou não, imediatamente, como no exemplo abaixo, em que o loop para quando a variável i atinge o valor 5:

for (int i = 0; i < 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    System.out.println("Iteração: " + i);  
}

**2. Instrução *continue***: Pula para a próxima iteração, ignorando o restante do código dentro da iteração atual. Abaixo, o laço ignora os números pares:

*for (int i = 0; i < 10; i++) {  
    if (i % 2 == 0) {  
        continue;  
    }  
    System.out.println("Iteração ímpar: " + i);  
}*

**Conteúdo Bônus**

**Título:** Lógica de programação 09 - Estruturas condicionais

**Canal:** Cataline

**Plataforma:** YouTube

**Descrição**: Neste vídeo, abordamos o conceito de estruturas condicionais, essenciais para controlar o fluxo de execução de programas em diversas linguagens de programação. Uma excelente explicação para quem está aprendendo lógica de programação.

**Referências Bibliográficas**

ASCENCIA, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação**: algoritmos, Pascal, C/C++ e Java. 2. ed. Pearson, 2007.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. **Sistemas operacionais**. 3. ed. Pearson, 2005.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação**: a construção de algoritmos e estruturas de dados. 3. ed. Pearson, 2005.

GUEDES, S. (Org.). **Lógica de programação algorítmica**. Pearson, 2014.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações móveis**: arquitetura, projetos e desenvolvimento. Pearson, 2005.

MELO, A. C. V. de; SILVA, F. S. C. da. **Princípios de linguagens de programação**. Blucher, 2014.

MENEZES, A. M. de.**Os paradigmas de aprendizagem de algoritmo computacional**. Blucher, 2015.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados, com aplicações em Java**. Pearson, 2016.