**Sintaxe do Java**

**Principais características da sintaxe do Java**

As principais características da sintaxe do Java incluem a orientação a objetos, tipagem estática, case sensitivity, blocos de código delimitados por chaves, uso de estruturas de controle de fluxo e um sistema robusto de tratamento de exceções. Estas são as principais características da sintaxe do Java:

* Bloco de Código: delimitado por {};
* Controle de Fluxo: utiliza estruturas como if, else, for, while;
* Tratamento de Exceções: realizado com try, catch, finally;
* Tipagem Estática: tipos de variáveis são definidos na compilação;
* Comentários: usam // para comentários de linha única e /\* … \*/ para múltiplas linhas;
* Case Sensitive: diferencia maiúsculas de minúsculas;
* Estrutura de Programa: composta por classes e métodos;
* Classes e Métodos: todo programa é formado por classes que contêm métodos.

**Exemplo de Estrutura de Programa Java**

public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println(“Hello, World!”);  
    }  
}

1. public class HelloWorld: declara uma classe pública chamada HelloWorld.

2. public static void main(String[] args): método principal, ponto de entrada do programa.

3. System.out.println(“Hello, World!”): imprime “Hello, World!” no console (na tela).

**Tipos de Dados**

Tipos de dados primitivos no Java são os tipos básicos de dados que representam valores simples e indivisíveis, como números, caracteres e valores booleanos, sendo diretamente suportados pela linguagem sem a necessidade de objetos adicionais. Java possui oito tipos de dados primitivos:

1. byte: 8 bits, intervalo de -128 a 127.

2. short: 16 bits, intervalo de -32,768 a 32,767.

3. int: 32 bits, intervalo de -2^31 a 2^31-1.

4. long: 64 bits, intervalo de -2^63 a 2^63-1.

5. float: 32 bits, ponto flutuante de precisão simples.

6. double: 64 bits, ponto flutuante de precisão dupla.

7. char: 16 bits, caractere Unicode.

8. boolean: true ou false.

Para praticar, crie um programa Java criando variáveis de desses tipos e imprimir seus valores usando System.out.println.

**Vetores (Arrays)**

Vetores em Java são estruturas de dados que armazenam uma sequência de elementos do mesmo tipo, permitindo acesso e manipulação eficiente dos dados por meio de índices.

A declaração de vetores acontece com essa sintaxe:

*int[] numbers; // Declaração de um vetor de inteiros  
String[] names; // Declaração de um vetor de strings*

A inicialização de vetores pode acontecer das seguintes formas:

- Estática:

*int[] numbers = {1, 2, 3, 4, 5};  
String[] names = {“Alice”, “Bob”, “Charlie”};*

- Dinâmica:

*int[] numbers = new int[5];  
String[] names = new String[3];*

A inicialização estática de vetores em Java define os elementos do vetor no momento da declaração, enquanto a inicialização dinâmica apenas reserva o espaço de memória, permitindo que os valores sejam atribuídos posteriormente.

Para praticar, considere o exemplo abaixo:

int[] numeros = new int[5];  
numeros[0] = 10;  
numeros[1] = 20;  
numeros[2] = 30;  
numeros[3] = 40;  
numeros[4] = 50;

System.out.println(“Valores do vetor usando loop for:”);  
for (int i = 0; i < numeros.length; i++) {  
    System.out.println(“Elementos na posição " + i + “: " + numeros[i]);  
}

String[] nomes = {“Ana”, “Bruno”, “Carlos”, “Diana”, “Eduardo”};  
System.out.println(”\nNomes no vetor usando loop for-each:”);  
for (String nome : nomes) {  
    System.out.println(nome);

Este código em Java demonstra a criação e manipulação de vetores de inteiros e strings, além de mostrar como iterar sobre eles usando diferentes tipos de*loops*(repetições).

1. Criação e Inicialização do Vetor de Inteiros:

a.*int[] numeros = new int[5];* : declara e inicializa um vetor de inteiros chamado ‘numeros’ com tamanho 5, reservando espaço para 5 elementos.

b. As linhas seguintes atribuem valores específicos a cada posição do vetor. Cada elemento do vetor é acessado e modificado usando o índice correspondente, que vai de 0 a 4.

2. Impressão dos Valores do Vetor de Inteiros:

a. *System.out.println(“Valores do vetor usando loop for:”)*; : imprime uma mensagem indicando que os valores do vetor serão mostrados.

b. Um *loop*, em português: repetição, for é utilizado para iterar sobre cada elemento do vetor:

for (int i = 0; i < numeros.length; i++) {  
System.out.println(“Elementos na posição " + i + “: " + numeros[i]);  
}

c. *numeros.length*: retorna o tamanho do vetor, garantindo que o loop percorra todas as posições.

d. Em cada iteração, o valor do elemento na posição ‘i’ é impresso.

3. Criação e Inicialização do Vetor de Strings:

a. *String[] nomes = {“Ana”, “Bruno”, “Carlos”, “Diana”, “Eduardo”}*; : declara e inicializa um vetor de strings chamado nomes com os valores “Ana”, “Bruno”, “Carlos”, “Diana” e “Eduardo”.

4. Impressão dos Valores do Vetor de Strings:

a.*System.out.println(”\nNomes no vetor usando loop for-each:”);* : imprime uma mensagem indicando que os nomes do vetor serão mostrados.

b. Um *loop*, em português: repetição, for-each é utilizado para iterar sobre cada elemento do vetor de strings:

for (String nome : nomes) {  
System.out.println(nome);  
}

c. Esse *loop* simplifica a iteração sobre todos os elementos do vetor, imprimindo cada nome contido no vetor ‘nomes’.

**Matrizes**

Matrizes são vetores multidimensionais. Em Java, as matrizes são estruturas de dados que armazenam múltiplos valores do mesmo tipo, organizados em forma de tabela, permitindo acesso e manipulação eficiente de grandes volumes de informações.

A declaração de uma matriz de duas dimensões em Java é feita especificando o tipo de dados seguido de dois pares de colchetes, como:

*int[][] matriz = new int[linhas][colunas];*

onde linhas e colunas representam o tamanho da matriz.

Em Java, existem várias formas de declarar e inicializar matrizes. Aqui estão algumas das sintaxes mais comuns:

* Declaração e inicialização com tamanho definido:

int[] matriz1D = new int[5];          // Matriz unidimensional  
int[][] matriz2D = new int[3][4];     // Matriz bidimensional  
int[][][] matriz3D = new int[2][3][4]; // Matriz tridimensional

* Declaração e inicialização com valores explícitos:

int[] matriz1D = {1, 2, 3, 4, 5}; // Matriz unidimensional com valores iniciais  
int[][] matriz2D = {              // Matriz bidimensional com valores iniciais  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};

* Declaração sem inicialização imediata:

int[] matriz1D;     // Declaração sem inicialização  
int[][] matriz2D;   // Declaração sem inicialização  
matriz1D = new int[5];      // Inicialização posterior  
matriz2D = new int[3][4];   // Inicialização posterior

* E outra possibilidade é uma declaração de matriz de tamanho irregular, onde apenas uma das posições do pode ter valor definido, como este exemplo:

int[][] matrizJagged = new int[3][]; // Declaração de matriz 2D com linhas de tamanhos diferentes  
matrizJagged[0] = new int[2];  
matrizJagged[1] = new int[4];  
matrizJagged[2] = new int[3];

Essas sintaxes permitem criar e manipular matrizes em Java de maneira flexível, dependendo da necessidade do código.

Inicializar matrizes em Java envolve a criação da matriz e a atribuição de valores a seus elementos. Dependendo das necessidades do seu programa, a inicialização pode ser feita de várias maneiras.

É possível fazer a inicialização com tamanho definido, sem valores iniciais:

int[][] matriz2D = new int[3][4];

Também é possível realizar a inicialização com valores explícitos, para situações em que você já conhece os valores que deseja armazenar:

int[][] matriz2D = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
}; // Matriz 3x3 com valores específicos.

E é possível realizar a inicialização posteriormente, onde a declaração e a inicialização acontecem em momentos distintos.

É importante lembrar que ao inicializar matrizes numéricas sem valores explícitos, todos os elementos são automaticamente preenchidos com zero, ou com null para objetos e false para booleanos. Além disso, é importante também se certificar de especificar as dimensões corretamente durante a inicialização, pois uma vez criada, o tamanho da matriz não pode ser alterado.

**Visualização dos dados de Matrizes**

O código Java abaixo tem como objetivo percorrer e exibir os valores armazenados em uma matriz bidimensional.

System.out.println(“\nValores da matriz:”);  
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();  
}

Alguns detalhes importantes para entendimento correto, são:

* f*or (int i = 0; i < matriz.length; i++)* é o início de um laço for que percorre cada linha da matriz onde i é uma variável de controle que começa em 0 e vai até o último índice da matriz (ou seja, *matriz.length* - 1) e *matriz.length* retorna o número de linhas da matriz.
* *for (int j = 0; j < matriz[i].length; j++)*é o laço interno e ele percorre cada coluna da matriz, ou seja, cada elemento dentro de uma linha onde**j** é uma variável de controle que começa em 0 e vai até o último índice da linha atual da matriz (ou seja, *matriz[i].length* - 1) e *matriz[i].length* retorna o número de colunas na linha**i** da matriz.

O código percorre toda a matriz linha por linha e coluna por coluna, imprimindo todos os valores na tela. Cada linha da matriz é impressa em uma nova linha do console, e os valores de cada linha são separados por espaços. Isso permite visualizar os valores da matriz de forma organizada.

**Estruturas de Dados**

Em Java, a escolha da estrutura de dados correta é fundamental para a eficiência e clareza de um programa. O Java fornece uma ampla variedade de estruturas de dados, cada uma projetada para atender a diferentes necessidades de armazenamento e manipulação de dados tanto integradas na biblioteca padrão (Java Collections Framework) quanto personalizadas. Essas estruturas permitem organizar e acessar dados de maneira eficiente, dependendo do caso de uso específico.

Algumas dessas estruturas são:

1. Arrays: estruturas de dados mais básica em Java. Um array é uma coleção de elementos do mesmo tipo, armazenados em posições contíguas de memória, acessíveis através de índices.

2. Listas: estruturas que permitem o armazenamento dinâmico de elementos, onde a ordem dos elementos é preservada. As duas implementações principais são:

a. *ArrayList*: Uma lista baseada em array, que oferece acesso rápido aos elementos através de índices, mas pode ser menos eficiente para inserções e remoções frequentes.

b. *LinkedList:* Uma lista baseada em nós encadeados, ideal para cenários onde inserções e remoções são frequentes, porém, com acesso menos eficiente aos elementos individuais.

3. Pilhas (*Stacks*): a estrutura de pilha segue o princípio LIFO (*Last In, First Out*), onde o último elemento inserido é o primeiro a ser removido. Em Java, a classe *Stack* oferece métodos para manipular esse tipo de estrutura.

4. Filas (*Queues*): estas seguem o princípio FIFO (*First In, First Out*), onde o primeiro elemento inserido é o primeiro a ser removido. A interface *Queue* e suas várias implementações permitem o gerenciamento de tarefas em uma ordem linear, típica em cenários de processamento de tarefas ou gerenciamento de buffers.

5. Conjuntos (*Sets*): são estruturas que não permitem elementos duplicados. Java oferece duas implementações principais:

a. *HashSet*: conjunto que não mantém ordem específica dos elementos e oferece operações de inserção, remoção e busca muito rápidas.

b. *TreeSet*: conjunto que mantém os elementos ordenados conforme uma ordem natural ou uma ordem especificada por um comparador.

6. Mapas (*Maps*): são coleções que associam chaves a valores, permitindo um acesso rápido ao valor correspondente a uma chave específica. As implementações comuns incluem:

a. *HashMap*: mapa que não mantém ordem específica das chaves, oferecendo uma performance eficiente em operações de busca e inserção.

b. *TreeMap*: mapa que mantém as chaves ordenadas, seja na ordem natural ou conforme um comparador personalizado.

**Conteúdo Bônus**

**Título:** Sintaxe Java | Curso de Java Completo | Aula 01

**Canal:**Noturama Fut

**Plataforma:**YouTube

**Descrição:** Esse é o primeiro vídeo de uma série de aulas sobre Java completo! Uma excelente introdução para quem deseja aprender a sintaxe básica da linguagem Java e se aprofundar no desenvolvimento com essa linguagem poderosa.

**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.