**1 Revisão de alguns conceitos importantes de programação**

* 1. **Variáveis Primitivas**

Em linguagens de programação, as **variáveis primitivas** e as **variáveis de referência** são dois tipos fundamentais de armazenamento de dados. Vamos explorar a diferença entre elas com exemplos.

As **variáveis primitivas** armazenam diretamente valores simples, como números e caracteres. Elas são alocadas na **stack** (pilha) da memória e possuem um tamanho fixo.

### ****Exemplos de tipos primitivos em Java:****

int idade = 25; // Número inteiro

double altura = 1.75; // Número decimal

char letra = 'A'; // Caracter único

boolean ativo = true; // Valor booleano (true ou false)

Esses valores são armazenados diretamente na variável e não dependem de um objeto externo

* 1. **Variáveis de Referência**

As **variáveis de referência** armazenam um ponteiro (endereço de memória) para um objeto, ao invés de armazenar diretamente o valor. Esses objetos são alocados na **heap** (memória dinâmica) e podem conter múltiplos atributos e métodos.

### ****Exemplo em Java:****

class Pessoa {

String nome;

int idade;

Pessoa(String nome, int idade) {

this.nome = nome;

this.idade = idade;

}

}

// p1 é uma variável de referência

Pessoa p1 = new Pessoa("João", 30);

Pessoa p2 = p1; // Ambas apontam para o mesmo objeto na memória

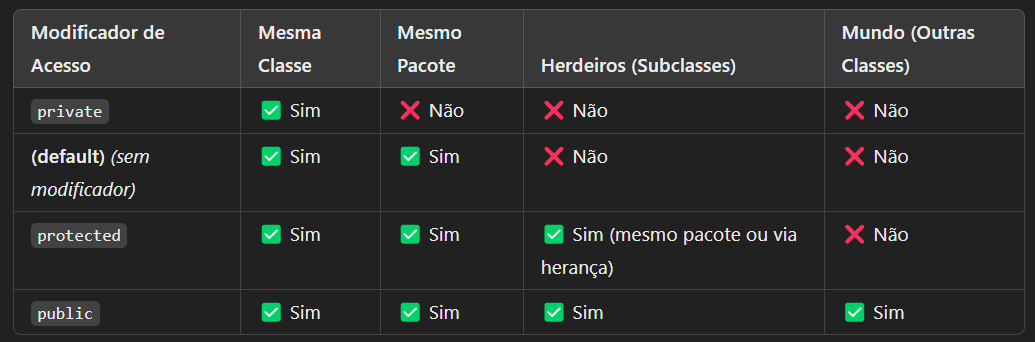
Neste caso, p1 e p2 armazenam um **endereço de memória** apontando para o mesmo objeto Pessoa na heap.

Se modificarmos p2, p1 também verá essa alteração, pois ambos referenciam o mesmo objeto.

p2.nome = "Carlos";

System.out.println(p1.nome); // Saída: Carlos

* 1. **Níveis de Acesso**

****

* 1. **Diferenças entre incrementos de variáveis**

**i++** (Pós-incremento):

* O valor da variável i é usado primeiro e depois incrementado.
* A variável é modificada após sua utilização.

Exemplo:

int i = 5;

int x = i++; // x recebe 5, depois i se torna 6

System.out.println("x = " + x); // x = 5

System.out.println("i = " + i); // i = 6

**++i** (Pré-incremento):

* O valor da variável i é primeiro incrementado e depois usado.
* A variável é modificada antes de ser usada.

Exemplo:

int i = 5;

int x = ++i; // // i se torna 6, depois x recebe 6

System.out.println("x = " + x); // x = 6

System.out.println("i = " + i); // i = 6

**i + 1** (Soma sem modificação):

* Apenas retorna o resultado da soma, sem alterar o valor de i.
* i não muda, apenas é utilizado no cálculo.

Exemplo:

int i = 5;

int x = i + 1; // x recebe 6, mas i continua 5

System.out.println("x = " + x); // x = 6

System.out.println("i = " + i); // i = 5

**2 Vetores**

Como já aprendemos, um dos recursos mais simples de qualquer linguagem de programação é a declaração de variáveis. Porém, para programas que requerem uma quantidade muito grande de variáveis, a forma simples como aprendemos até então pode não ser suficiente. Um vetor é um tipo de coleção que pode resolver este problema. Para tratar sobre este assunto, levaremos em conta os seguintes aspectos:

- Assim como ocorre com variáveis, vetores precisam ser **declarados** antes de serem utilizados.

-Para caracterizar a declaração de um vetor, utilizamos o operador **[ ]** em java.

- Vetores são **objetos**. Isso quer dizer que eles precisam ser construídos com o operador **new**, ou seja, precisam de **variáveis de referência** para serem manipulados.

- Os valores armazenados por um vetor possuem sempre um **mesmo tipo**. Por essa razão, vetores são chamados de estruturas **homogêneas**. Note como isso contrasta com classes, estruturas capazes de “armazenar” variáveis de tipos diferentes, que são chamadas de estruturas **heterogêneas**.

- Vetores possuem um **tamanho fixo**. Quando um vetor é declarado, o seu tamanho é especificado e nunca pode ser alterado. O tamanho de um vetor pode ser sempre descoberto por meio de sua propriedade **length**.

- As variáveis armazenadas por um vetor ficam sempre armazenadas de forma contínua na memória.

- Dizemos que os valores armazenados por um vetor ocupam suas “**posições**”. As posições são **números** **inteiros**, começando do **zero**. Assim, o primeiro valor de um vetor está em sua posição zero, o segundo está em sua posição um e assim por diante.

- Se um vetor é referenciado por uma variável **v**, utilizamos a expressão **v[i]** para acessar a sua i-ésima posição.

- As posições de um vetor são **sempre inicializadas pelo compilador** com o valor padrão referente ao tipo de suas variáveis. Por exemplo, um vetor de inteiros tem todas as suas posições inicializadas como zero. Um vetor de Strings, tem todas as suas posições inicializadas como **null**. E assim por diante.

**2.1 Exemplos básicos envolvendo vetores.** Nesta seção, veremos o uso elementar de um vetor. Para isso, crie um novo projeto e uma nova classe contendo o método **main**. A Listagem 1 mostra as operações mais elementares envolvendo um vetor. Veja os comentários para melhor entendimento.

Listagem 1

|  |
| --- |
| public class TestesElementaresComVetores {  public static void main(String[] args) {  //essa é a declaração de um vetor de ints, chamado v1.  int [] v1;  //também podemos declarar assim:  int v2 [];  //observe, contudo, que somente temos um vetor de fato quando o construímos com o operador new  int [] v3 = new int [4];  //aqui acessamos a primeira posição do vetor, no índice zero, atribuindo o valor 2  v3[0] = 2;  //também podemos exibi-lo  System.out.println (v3[0]);  //aqui exibimos o valor 0, já que o compilador inicializa as posições automaticamente  System.out.println (v3[2]);  //podemos solicitar ao usuário que digite um valor a ser guardado em uma posição de um vetor  v3[1] = Integer.parseInt(JOptionPane.showInputDialog("Digite um inteiro"));  //podemos exibir o valor de uma posição específica do vetor  JOptionPane.showMessageDialog(null, v3[1]);  //podemos checar o tamanho do vetor assim  System.out.println(v3.length);  //Também podemos fazer operações aritméticas utilizando as posições do vetor  v3[0] = 2;  v3[1] = 3;  int r = v3[0] + v3[1];  //o indice pode ser qualquer expressão inteira  System.out.println(v3[3 - 2]);  System.out.println(v3[3 - v3.length + 2]);  }  } |

**2.2 Algoritmos elementares sobre vetores.** Nesta seção, estudaremos sobre algoritmos elementares que podemos aplicar sobre vetores. A Listagem 2 mostra como **preencher** **um** **vetor** com dados digitados pelo usuário, exibindo-os a seguir.

Listagem 2

|  |
| --- |
| public class PreencherEExibir {  public static void main(String[] args) {  int [] v = new int [10];  //assim fica inviável para vetores com muitas posições  v[0] = Integer.parseInt(JOptionPane.showInputDialog("Digite um valor"));  v[1] = Integer.parseInt(JOptionPane.showInputDialog("Digite um valor"));  v[2] = Integer.parseInt(JOptionPane.showInputDialog("Digite um valor"));    //devemos automatizar a repetição de código com uma estrutura de repetição  for (int i = 0; i < v.length; i++){  v[i] = Integer.parseInt(JOptionPane.showInputDialog("Digite um valor"));  }    //para exibir também devemos automatizar a repetição  int i = 0;  while (i < v.length){  JOptionPane.showMessageDialog(null, v[i]);  i++;  }  //para exibir todos de uma vez poderíamos concatenar os valores em uma string  String s = "";  for (i = 0; i < v.length; i++){  s = s + v[i] + " ";  }  //exibimos a String completa depois do for  JOptionPane.showMessageDialog(null, s);    }    } |

A Listagem 3 mostra como podemos calcular a soma dos elementos em um vetor.

Listagem 3

|  |
| --- |
| public class SomarElementosDeUmVetor {    public static void main(String[] args) {  //usamos um vetor de reais dessa vez, só para ilustrar  //o mesmo se aplica para inteiros  //perceba a forma como o vetor foi construído  //trata-se somente de um "syntax sugar" que o compilador nos oferece  double [] v = {2.5, 6.4, 7.5, 1, 1, 2, 7, 5};  //começamos declarando um acumulador, incializado com zero, valor identidade da soma  double soma = 0;  //depois, somamos elemento a elemento, acumulando um a um  soma = soma + v[0];  soma = soma + v[1];  soma = soma + v[2];  //estamos repetindo manualmente de novo, melhor automatizar  soma = 0;  for (int i = 0; i < v.length; i++){  soma = soma + v[i];  }  //depois do for, exibimos o resultado  JOptionPane.showMessageDialog(null, "Soma: " + soma);    }  } |

Também podemos estar interessados em calcular a média aritmética de elementos armazenados em um vetor. A Listagem 4 mostra como fazê-lo.

Listagem 4

|  |
| --- |
| public class CalcularMediaDeElementosEmUmVetor {    public static void main(String[] args) {  //um vetor  //talvez queiramos inicializar suas posições com valores aleatórios, veja  Random gerador = new Random();  double [] v = new double [100];  for (int i = 0; i < v.length; i++){  //estude essa expressão e identifique o intervalo...  v[i] = gerador.nextDouble() \* 5 + 3;  }  //uma vez preenchido o vetor, podemos fazer a soma  double soma = 0;  for (int i = 0; i < v.length; i++){  soma += v[i];  }  //finalmente, calculamos a média  double media = soma / v.length;  //e exibimos  JOptionPane.showMessageDialog(null, "Média: " + media);  }  } |

Que tal fazer uma busca em um vetor? Ou seja, dado um valor, responder se ele existe ou não no vetor? Veja como fazê-lo na Listagem 1.1.2.4.

Listagem 1.1.2.4

|  |
| --- |
| public class BuscaValorEmUmVetor {  public static void main(String[] args) {  int [] v = {1, 2, 5, 4, 3};  int valor = Integer.parseInt (JOptionPane.showInputDialog("Qual valor deseja buscar"));  int i;  for (i = 0; i < v.length; i++){  if (v[i] == valor)  break;  }  JOptionPane.showMessageDialog(null, i < v.length ? "Achou na posição " + i : "Não achou");  }  } |

***Exercícios***

1. Escreva um algoritmo que leia 10 números inteiros do teclado e armazene-os em um vetor. Em seguida, imprima o valor armazenado no início e no fim do vetor.

***Referências***

DEITEL, P. e DEITEL, H. **Java Como Programar**. 8ª Edição. São Paulo, SP: Pearson, 2010.

LOPES, A. e GARCIA, G. **Introdução à Programação – 500 Algoritmos Resolvidos**. 1ª Edição. São Paulo, SP: Elsevier, 2002.