|  | FTCE  Faculdade de Tecnologia e Ciências Exatas |
| --- | --- |

USJT – 2021/2 – Programação de Soluções Computacionais

**Aula: 11 - Vetores**

**1 Introdução**

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. Neste material, estudaremos sobre diferentes tipos de **coleções** e sobre suas peculiaridades.

**1.1 Vetores**

Um vetor é um tipo de coleção. 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 **[ ]**.

- 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 e vivem sempre na memória **heap**.

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

- Vetores têm 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 contígua na memória.

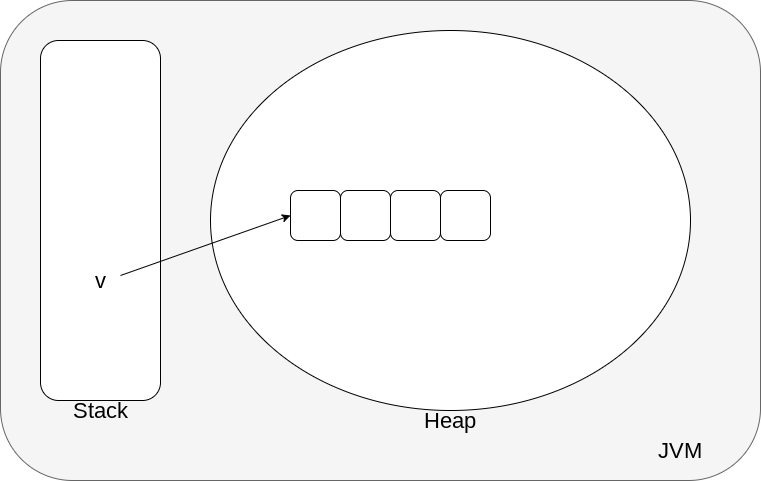
- Dizemos que os valores armazenados por um vetor ocupam suas “**posições**”. As posições são **números** **inteiros**, começando do **zero**. Assim, o primeiro valor de um vetor está em sua posição zero, o segundo está em sua posição um e assim por diante.

- 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.

A Figura 1.1.1 mostra uma variável de referência e um vetor que armazena inteiros, de acordo com o modelo de memória da JVM.

Figura 1.1.1



**1.1.1 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 1.1.1.1 mostra as operações mais elementares envolvendo um vetor. Veja os comentários para melhor entendimento.

Listagem 1.1.1.1

| public class TestesElementaresComVetores {  public static void main(String[] args) {  //essa é uma variável de referência capaz de referenciar 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  //aqui, v3 é uma variável de referência que referencia um vetor de quatro ints  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]);  //Também 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"));  //e exibir também  JOptionPane.showMessageDialog(null, v3[1]);  //podemos checar o tamanho do vetor assim  System.out.println(v3.length);  //Também podemos fazer continhas  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]);  }  } |
| --- |

**1.1.2 Algoritmos elementares sobre vetores.** Nesta seção, estudaremos sobre algoritmos elementares que podemos aplicar sobre vetores. A Listagem 1.1.2.1 mostra como **preencher** **um** **vetor** com dados digitados pelo usuário, exibindo-os a seguir.

Listagem 1.1.2.1

| public class PreencherEExibir {  public static void main(String[] args) {  int [] v = new int [10];  //assim fica repetitivo...  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"));    //Podemos 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, talvez usando um while  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 depois do for  JOptionPane.showMessageDialog(null, s);    }    } |
| --- |

A Listagem 1.1.2.2 mostra como podemos calcular a soma dos elementos em um vetor.

Listagem 1.1.2.2

| 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  //ele continua sendo um objeto e vive na heap  //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[1];  }  //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 1.1.2.3 mostra como fazê-lo.

Listagem 1.1.2.3

| 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");  }  } |
| --- |

Em geral, quando calculamos a média de uma coleção de valores, estamos interessados em obter um resumo dela. Um número que traga uma ideia geral da coleção para que não tenhamos que analisá-la elemento por elemento. Contudo, duas coleções que possuem média igual podem ser muito diferentes. A Tabela 1.1.2.1 mostra um exemplo. Temos duas turmas de alunos nas quais as notas foram muito diferentes. Apesar disso, a média delas é igual.

Tabela 1.1.2.1

| **Turma 1** | | **Turma 2** | |
| --- | --- | --- | --- |
| **Aluno** | **Nota** | **Aluno** | **Nota** |
| João | 5 | Marina | 9 |
| Maria | 5 | José | 5 |
| Pedro | 5 | Afonso | 1 |

Embora tenham a mesma média, as turmas da Tabela 1.1.2.1 são diferentes na variação que possuem em relação a sua própria média. Na Turma 1 não houve variação alguma. Todos os alunos têm nota igual à média. Na turma 2, houve variação razoável. Marina ficou 4 pontos acima da média, José ficou na média e Afonso ficou 4 pontos abaixo da média. Junto com a média, essa ideia de **dispersão** pode trazer ainda mais informações de interesse para quem analise os dados.

Duas medidas de dispersão muito comuns são a **variância** e o **desvio** **padrão**. A variância consiste na soma dos quadrados das diferenças de cada nota em relação à média, dividido pelo total de notas (menos 1, no caso de amostras pequenas). Assim, a variância de cada turma é a seguinte.

Turma 1: ((5 – 5)2 + (5 - 5)2 + (5 – 5 )2) / (3 – 1) = (0 + 0 + 0) / 2 = 0

Turma 2: ((9 – 5)2 + (5 – 5)2 + (1 – 5)2) / (3 – 1) = 16 + 0 + 16 = 32 /2 = 16.

Note, porém, que a variância está em uma unidade diferente em relação aos valores originais pelo fato de termos feito a soma dos quadrados. Assim, é comum extrair a raiz quadrada da variância, obtendo o que é conhecido como **desvio padrão**. O desvio padrão de cada turma é dado a seguir.

Turma 1:

Turma 2:

A Listagem 1.1.2.5 mostra a implementação da média (requisito para a variância), variância e desvio padrão dos elementos armazenados em um vetor.

Listagem 1.1.2.5

| public class MediaVarianciaDesvioPadrao {    public static void main(String[] args) {  double [] colecao = {5, 5, 5};    //media  double soma = 0;  for (int i = 0; i < colecao.length; i++){  soma += colecao[i];  }  double media = soma / colecao.length;  //variancia  double somaDiferencas = 0;  for (int i = 0; i < colecao.length; i++){  somaDiferencas += Math.pow((colecao[i] -media), 2);  }  double variancia = somaDiferencas / (colecao.length - 1);  //desvio padrao  double desvioPadrao = Math.sqrt(variancia);  System.out.printf("Media: %.2f, Variancia: %.2f, Desvio Padrao: %.2f\n", media, variancia, desvioPadrao);  }    } |
| --- |

- Percebeu como escrevemos o código para calcular a média duas vezes ao longo deste material? As operações que implementamos têm grande potencial de reusabilidade. Uma vez implementadas, muitos programas podem desejar utilizá-las. Podemos viabilizar isso escrevendo cada funcionalidade potencialmente reutilizável como um método. Ou seja, um bloco de código que possui um nome e que promete cumprir alguma função. Ele pode receber dados sobre os quais operar e pode devolver valores que eventualmente calcular. Vamos ilustrar essa ideia criando a nossa própria classe de utilitários matemáticos. No momento, ela oferecerá três operações: cálculo de média de uma coleção, cálculo de variância de uma coleção e cálculo de desvio padrão de uma coleção. Veja a Listagem 1.1.2.6.

Listagem 1.1.2.6

| public class Matematica {      //calcula a média da colecão recebida  public static double media (double [] valores){  double soma = 0;  for (int i = 0; i < valores.length; i++){  soma += valores[i];  }  return soma / valores.length;  }    //calcula a variancia da coleção recebida  //note que aqui já reutilizamos a media  public static double variancia (double [] valores){  double m = media (valores);  double somaDiferencas = 0;  for (int i = 0; i < valores.length; i++){  somaDiferencas += Math.pow ((valores[i] - m), 2);  }  return somaDiferencas / (valores.length - 1);  }    //calcula o desvio padrao de uma colecao  //aqui reutilizamos a variancia  public static double desvioPadrao (double [] valores){  return Math.sqrt(variancia(valores));  }      } |
| --- |

Podemos testar a classe Matematica com o código da Listagem 1.1.2.7.

Listagem 1.1.2.7

| public class TesteMatematica {  public static void main(String[] args) {  double[] turma1 = {5, 5, 5};  double[] turma2 = {9, 5, 1};  double media1, media2, variancia1, variancia2, desvio1, desvio2;  media1 = Matematica.media(turma1);  variancia1 = Matematica.variancia(turma1);  desvio1 = Matematica.desvioPadrao(turma1);  media2 = Matematica.media(turma2);  variancia2 = Matematica.variancia(turma2);  desvio2 = Matematica.desvioPadrao(turma2);  System.out.printf("Turma 1: (Média: %.2f, Variância: %.2f, Desvio Padrão: %.2f)\n", media1, variancia1, desvio1);  System.out.printf("Turma 2: (Média: %.2f, Variância: %.2f, Desvio Padrão: %.2f)\n", media2, variancia2, desvio2);  }  } |
| --- |

***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.