# Programação Estruturada

## Aula 06 - Vetores e Matrizes

# Apresentação

Como já foi discutido na disciplina de Lógica de Programação, há situações em que os tipos de dados básicos não são suficientes para resolver os problemas encontrados na prática. Em muitas dessas situações, podemos utilizar estruturas de dados homogêneas do tipo **vetor** ou **matriz**. Nesta aula, você irá aprender a declarar e manipular vetores e matrizes na linguagem Java.

Tenha uma boa leitura!

**Vídeo 01** - Apresentação

## Objetivos

Ao final desta aula, você será capaz de:

- Saber estruturar dados através de vetores e matrizes na linguagem Java;
- Ler e escrever dados em matrizes e vetores;
- Aplicar corretamente o uso de vetores e matrizes.

# 1. Vetores

Durante seus estudos, você já viu que uma variável ocupa um espaço na memória reservado para armazenar valores definidos de acordo com determinado tipo de dados (veja Aula 2). Por exemplo, variáveis do *int* podem armazenar valores numéricos inteiros, mas apenas um valor por vez, certo? Pense na situação em que você queira escrever um programa que solicite ao usuário a entrada das notas de 10 alunos de uma turma de sua escola. Como você faria isso?

Uma possibilidade de programa é mostrada a seguir:

```
1  import java.util.Scanner;
2
3  public class Programa10Notas {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite a nota de número 1");
7          double nota1 = leitor.nextDouble();
8          System.out.println("Digite a nota de número 2");
9          double nota2 = leitor.nextDouble();
10         System.out.println("Digite a nota de número 3");
11         double nota3 = leitor.nextDouble();
12         System.out.println("Digite a nota de número 4");
13         double nota4 = leitor.nextDouble();
14         System.out.println("Digite a nota de número 5");
15         double nota5 = leitor.nextDouble();
16         System.out.println("Digite a nota de número 6");
17         double nota6 = leitor.nextDouble();
18         System.out.println("Digite a nota de número 7");
19         double nota7 = leitor.nextDouble();
20         System.out.println("Digite a nota de número 8");
21         double nota8 = leitor.nextDouble();
22         System.out.println("Digite a nota de número 9");
23         double nota9 = leitor.nextDouble();
24         System.out.println("Digite a nota de número 10");
25         double nota10 = leitor.nextDouble();
26         // ... aqui, faz o processamento das notas
27     }
28 }
```

Note que o código, apesar de simples, fica repetitivo (duplicação de código) e muito extenso. Imagine declarar variáveis para turmas de 50 alunos!!! E o pior de tudo é que, na prática, vamos ter turmas com quantidade de alunos diferentes.

Dessa forma, não sabemos previamente a quantidade de variáveis que precisaríamos declarar.

Você viu na aula anterior que comandos de iteração podem eliminar duplicação de código, reduzindo assim o tamanho do código. Vamos ver uma versão desse mesmo programa agora usando laços. Observe o código e verifique se ele está correto.

```
1  import java.util.Scanner;
2
3  public class Programa10NotasFor {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          double nota;
7          for (int i = 1; i <= 10; i++) {
8              System.out.println("Digite a nota de número " + i);
9              nota = leitor.nextDouble();
10         }
11         // ... aqui, faz o processamento das notas
12     }
13 }
```

Você conseguiu detectar algum problema nesse código? Ele está bem enxuto, porém o problema é que temos apenas uma única variável. Toda vez que uma nota é lida, é armazenada nessa mesma variável. Como a variável só consegue armazenar um valor por vez, a nota anteriormente lida é perdida. Para resolver esse problema, fazemos uso dos chamados **vetores (ou arrays)**, uma estrutura de dados *homogênea* e *indexada* que pode armazenar diversos valores de um mesmo tipo de dado.

> Vetor é uma estrutura de dados homogênea, ou seja, que armazena valores de um mesmo tipo.

Logo a seguir, vemos a declaração e inicialização de um vetor do tipo double de 10 posições e sua representação na memória:

```
1  double notas[] = new double[10]
```


O índice (número do elemento, acima do desenho) representa a posição onde o dado está armazenado no vetor. Por exemplo, na figura anterior, o valor armazenado na segunda posição do vetor é 9,0. Por isso, diz-se que o vetor é uma estrutura **indexada**. A forma geral de declaração de vetores segue abaixo:

```
1 tipo nome_da_variavel[];
```

Ao declarar um vetor, indicamos o tipo do array e o nome da variável. O uso do [] é para indicar que a variável é um array. Existe, porém, uma diferença quando declaramos um array em relação às declarações de variáveis de tipos primitivos. Para que seja alocado o espaço na memória relativo ao array, é necessário usar a palavra chave **new**, seja na própria declaração do array, ou inicializando-o posteriormente. Veja o formato geral para a inicialização na própria declaração:

```
1 tipo nome_da_variavel[] = new tipo[tamanho];
```

O *tipo*, como o próprio nome já diz, irá declarar o tipo de cada elemento do vetor: inteiro (**int**), real (**double** e **float**), caractere (**char**), etc. Já o *tamanho* irá definir quantos elementos o vetor irá guardar. Quando usamos a palavra chave **new**, é reservado um espaço de memória suficiente para armazenar a quantidade de elementos especificada. Fazendo uma analogia a grosso modo, declarar um vetor do tipo double de 50 posições equivaleria a declarar 50 variáveis do tipo double, só que é muito mais prático trabalhar com vetores. Principalmente porque o tamanho do vetor pode ser determinado pelo usuário durante a execução do programa! Veja como isso acontece:

```
1  import java.util.Scanner;
2
3  public class ProgramaNotasArray {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite a quantidade de notas a serem lidas");
7          int quantidade = leitor.nextInt();
8          double notas[] = new double[quantidade];
9          for (int i = 1; i <= quantidade; i++) {
10             System.out.println("Digite a nota de número " + i);
11             notas[i] = leitor.nextDouble();
12         }
13         // ... aqui, faz o processamento das notas
14     }
15 }
```

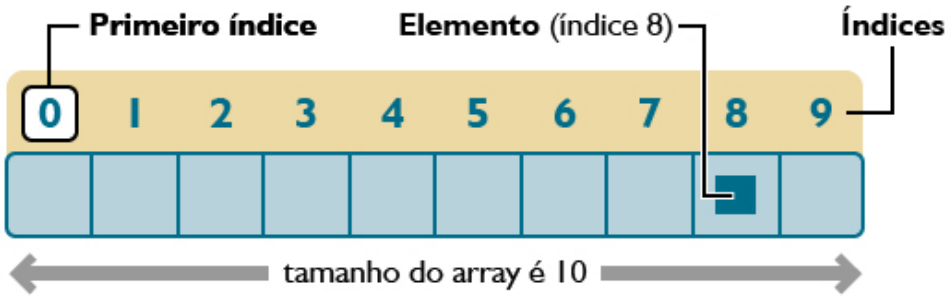
Observe no código que primeiro é lida a quantidade de notas a serem processadas. Essa quantidade é armazenada na variável *quantidade*, a qual é utilizada para determinar o tamanho do array e a condição de continuidade do laço (i <= quantidade). Isso quer dizer que, se o usuário digitar 45 para a quantidade de notas, o programa irá ler e alocar espaço para guardar essas 45 notas. Não é legal?

Bom, você deve ter percebido que dentro do laço estamos atribuindo o valor lido a uma determinada posição do array. Isso está sendo feito pelo seguinte comando:

```
1  notas[i] = leitor.nextDouble();
```

Observe o uso do [i]. Como um vetor geralmente tem várias posições, então é necessário informar em qual posição você quer ler ou escrever. Isso é feito passando-se o índice da posição. Mas, cuidado, pois na maioria das linguagens de programação, os índices começam pelo número 0 (zero)! Veja isso na Figura 1, que mostra o espaço na memória de um array de 10 posições. O índice do primeiro elemento do array é 0, e não 1. Já o índice da última posição é 9, e não 10. Observe que, se você quiser acessar o n-ésimo elemento de um array, é só usar o índice (n - 1). É fácil, mas exige atenção.

**Figura 01** - Indexação de um array




**Fonte**: Adaptado de <http://download.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>. Acesso em: 5 dez. 2011.

Para ler ou escrever em uma posição do array, é só usar o nome da variável e indicar o índice da posição a ser lida ou escrita, como nos seguintes exemplos:

```
1 // escreve a nota no índice 4, ou seja, na 5a posição do array
2 notas[4] = 7.8;
3
4 // escreve a nota no índice i
5 notas[i] = 8.3;
6
7 // imprime a nota de índice 8, ou seja, a 9a nota do array
8 System.out.println(notas[8]);
```

**Vídeo 02** - Vetor

# Atividade 01

1. Implemente e execute o ProgramaNotasArray mostrado na aula.
2. Altere o ProgramaNotasArray para que imprima as notas na tela, depois de lidas, mostrando sua localização no array em termos de índice (de zero à quantidade - 1) e de ordem (de um a quantidade).

# 2. Calculando o espaço ocupado na memória por um vetor

Na aula 2, você aprendeu que cada variável ocupa um espaço determinado, em número de bits, de acordo com o seu tipo. Por exemplo, uma variável char ocupa 2 bytes. Caso queira relembrar o tamanho ocupado por variáveis de tipos primitivos, visite novamente essa aula. Agora, quanto espaço de memória você acha que é ocupado por um vetor?

Pois bem, a quantidade de espaço necessário para armazenar os dados de um vetor na memória está relacionada a duas coisas: seu tamanho e seu tipo de dados. Para calcular o espaço de memória ocupado em bytes por um vetor, temos a seguinte fórmula:

```
1 Espaço ocupado = tamanho ocupado por cada elemento do vetor * tamanho do vetor
```

Dessa forma, considerando um vetor com 10 inteiros, ele irá ocupar 10 x 4 bytes, ou seja, 40 bytes, pois cada inteiro ocupa 4 bytes na memória

# Atividade 02

1. Calcule o espaço ocupado na memória por vetores do tamanho do vetor mostrado na Figura 1, para os tipos de dados char, short, long, float e double.

# 3. Conhecendo o tamanho de um vetor em tempo de execução

Como programador, você deve ter sempre o cuidado de assegurar que está utilizando índices válidos ao manipular um vetor. Por exemplo, observe o programa mostrado a seguir. Você consegue ver algum problema nele?

```
import java.util.Scanner;

public class ProgramaForArray {
    public static void main(String[] args) {
        int[] valores = new int[10];
        Scanner leitor = new Scanner(System.in);
        for (int i = 1; i <= 10; i++) {
            valores[i] = leitor.nextInt();
        }
    }
}
```

O programa usa a variável *i* para indexar o vetor, certo? Tudo bem, mas quais os valores usados para essa variável? Note que eles variam de 1 a 10, e não de 0 a 9. Isso fará com que o primeiro valor seja lido e armazenado na segunda posição do vetor (índice 1), ao invés de na primeira posição (índice 0). Porém, o maior problema é com o último valor lido, pois o programa tentará armazená-lo no índice de número 10, o qual não existe no vetor. Isso fará com que seja levantada uma exceção e o seu programa será finalizado automaticamente. Você verá no módulo de Orientação a Objetos como tratar exceções para que elas não finalizem automaticamente seu programa.

Portanto, muito cuidado ao manipular vetores. Você tem que usar os índices corretos. Além disso, existe uma outra situação que é quando não temos uma variável armazenando o tamanho do vetor, como no caso de rotinas que recebem um vetor como parâmetro (falaremos em outra aula sobre rotinas) e no caso do exemplo a seguir:

```
import java.util.Scanner;

public class ProgramaForArray2 {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite a quantidade de valores a digitar:");
        int[] valores = new int[leitor.nextInt()];
        System.out.println("Digite agora os valores:");
        for (int i = 0; i < 10; i++) {
            valores[i] = leitor.nextInt();
        }
        System.out.println("Valores digitados:");
        for (int i = 0; i < 10; i++) {
            System.out.println(valores[i]);
        }
    }
}
```

Cuidado! O código anterior irá levantar uma exceção caso o número digitado pelo usuário seja menor que 10. Veremos adiante como solucionar esse problema.

Note que o usuário irá digitar não só os valores do vetor, mas também o tamanho dele. Isso está sendo feito através da seguinte instrução:

```
1  int[] valores = new int[leitor.nextInt()];
```

Observe que o valor lido pelo usuário é usado diretamente para inicializar o tamanho do vetor, não sendo armazenado em nenhuma variável auxiliar. Já a variável *i* continua sendo usada novamente para indexar esse vetor, e seu valor está variando de 0 a 9. Porém, não sabemos qual valor o usuário digitou, então, como realizar essa indexação? A variável i tem que variar de 0 até que valor? Para descobrir isso, usamos o comando **length** que é usado depois do nome do vetor. Veja como fica isso no ProgramaForArray3:

```
1  import java.util.Scanner;
2
3  public class ProgramaForArray3 {
4     public static void main(String[] args) {
5        Scanner leitor = new Scanner(System.in);
6        System.out.println("Digite a quantidade de valores a digitar:");
7        int[] valores = new int[leitor.nextInt()];
8        System.out.println("Digite agora os valores:");
9        for (int i = 0; i < valores.length; i++) {
10          valores[i] = leitor.nextInt();
11       }
12       System.out.println("Valores digitados:");
13          for (int i = 0; i < valores.length; i++) {
14       System.out.println(valores[i]);
15    }
16 }
```

A expressão valores.**length** retorna o tamanho do vetor valores. Dessa forma, sempre é possível saber o tamanho de um vetor dada a variável que o referencia.

**Vídeo 03** - Vetor Dinâmico

# Atividade 03

1. Execute o ProgramaForArray mostrado na aula e comente o que acontece em sua execução.
2. Execute o ProgramaForArray2 mostrado na aula, digitando o valor 12 como o tamanho do vetor, e comente o que acontece em sua execução.
3. Execute o ProgramaForArray3 mostrado na aula, digitando o valor 12 como o tamanho do vetor, e comente o que acontece em sua execução.

# 4. Matrizes

## Você lembra o que são matrizes?

Se ainda tiver dúvida ou não lembra, revise a disciplina de Matemática Aplicada para entender melhor o que você vai aprender aqui.

Comecemos o assunto falando que um vetor pode ter mais de uma dimensão. Os vetores que vimos anteriormente são unidimensionais, ou seja, possuem apenas uma dimensão. Isso quer dizer que para percorrer o conteúdo do vetor você precisa fazer uso de apenas um índice. Se o vetor é bidimensional, são necessários dois índices para percorrer seu conteúdo. Se ele for tridimensional, você precisará de três índices e assim por diante.

Além dos vetores unidimensionais, é comum você precisar usar em certos momentos vetores bidimensionais. Esses tipos de vetores são chamados de matrizes. Imagine um programa para armazenar os nomes dos proprietários dos apartamentos do prédio onde você mora. Considerando que fosse um prédio com 10 andares, 3 apartamentos por andar, como você faria para armazenar os nomes dos 30 proprietários de apartamentos? Com certeza você não criaria mais 30 variáveis! Se você usasse um vetor, teríamos algo como:

```
1 String nomes[] = new String[30];
```

Esse vetor aí consegue armazenar 30 nomes, já que ele tem 30 posições do tipo String. Entretanto, onde estará armazenado o apartamento 402, ou seja, o segundo apartamento do quarto andar? Isso depende de como utilizaremos o nosso índice. Como organizaremos os proprietários dentro desse vetor? Iremos organizar por andar, colocando nas três primeiras posições os nomes dos proprietários do primeiro andar? Ou iremos organizar por numeração de apartamento, colocando primeiro os 10 proprietários de apartamentos com terminação 1 e assim por diante?

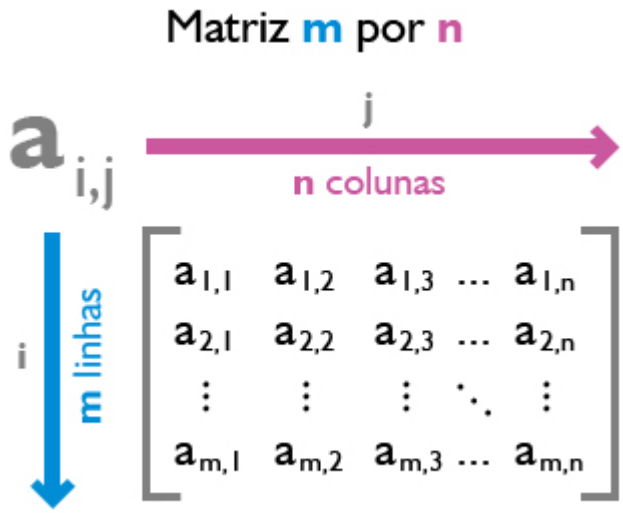
Veja que existem várias possibilidades de organizarmos o conteúdo desse vetor. Isso porque estamos usando um vetor unidirecional (indexado por apenas uma dimensão), mas estamos trabalhando com um problema que tem duas dimensões. Cada apartamento é identificado pela dimensão andar (1 a 10) e posição do apartamento no andar (1 a 3), correto? Nessas situações, fazemos uso das matrizes. Veja a seguir como declarar o vetor nomes de forma bidimensional, ou seja, como uma matriz:

```
1 String nomes[][] = new String[10][3];
```

Observe o uso de [][]. A ocorrência do [] em duas vezes indica que o vetor tem duas dimensões. Se o vetor tivesse três dimensões, o [] ocorreria três vezes, uma para cada dimensão. Note também que, no lado direito da atribuição, temos dois tamanhos. O primeiro é o tamanho da primeira dimensão, no caso, da quantidade de andares. Já o segundo valor é o tamanho da segunda dimensão, que no caso é a quantidade de apartamentos por andar.

Relembrando o conceito de matrizes na matemática, olhe a figura 1. Perceba que as informações dentro de uma matriz são indexadas por duas dimensões e que a quantidade de elementos que cabem na matriz é dada pela multiplicação do tamanho de cada dimensão. No caso da matriz nomes, temos 10 x 3 que é igual aos 30 apartamentos do prédio.

**Figura 02** - Matrizes na matemática




**Fonte**: <http://pt.wikipedia.org/wiki/Matriz_(matemática)>. Acesso em: 5 dez. 2011.

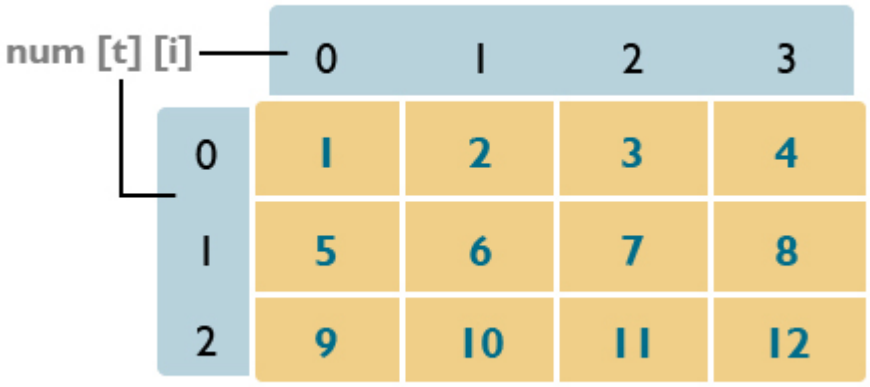
A forma geral da declaração de uma matriz é muito parecida com a declaração de uma matriz unidimensional:

```
1 tipo_da_variável nome_da_variável [altura][largura];
```

Quando trabalhamos com matriz, é comum chamarmos os índices de linhas e colunas. No caso, usualmente consideramos que o índice da esquerda indexa as linhas e o da direita indexa as colunas. Assim como no vetor unidirecional, os índices variam de 0 ao tamanho da dimensão menos um. Veja isso exemplificado na matriz mostrada na Figura 3, cuja declaração é mostrada a seguir.

```
1 int num[][] = new int[3][4];
```

**Figura 03** - Indexação de matrizes.




A manipulação de uma matriz é feita de forma equivalente à manipulação de vetores unidimensionais, só que agora você fará uso de duas variáveis de indexação. Veja o seguinte código:

```
1  import java.util.Scanner;
2
3  public class ProgramaMatriz {
4     public static void main(String[] args) {
5        Scanner leitor = new Scanner(System.in);
6        String nomes[][] = new String[10][3];
7        System.out.println(
8        "Digite o nome dos proprietários dos apartamentos");
9        for (int andar = 1; andar <= nomes.length; andar++) {
10          for (int posicao = 1; posicao <= nomes[andar – 1].length; posicao++) {
11             System.out.println("Apto. " + andar + "0" + posicao);
12             nomes[andar - 1][posicao - 1] = leitor.next();
13          }
14       }
15       System.out.println("Lista dos proprietários dos apartamentos:");
16       for (int andar = 0; andar < nomes.length; andar++) {
17          for (int posicao = 0; posicao < nomes[andar].length; posicao++) {
18             System.out.println("Apto. " + (andar + 1) + "0" +
19             (posicao + 1) + ": " + nomes[andar][posicao]);
20          }
21       }
22    }
23 }
```

O código mostrado declara um leitor (Scanner) e uma matriz para armazenar os nomes dos proprietários dos 30 apartamentos. Em seguida, temos um primeiro bloco de código responsável por ler os nomes dos 30 proprietários. Note que o fato

de termos dois índices (andar e posição do apartamento no andar) requer que você use dois laços for, um para cada dimensão. O primeiro está variando os valores do índice andar. Já o segundo, varia o valor do índice posição. Dentro do laço mais interno, temos uma mensagem sendo impressa com o número do apartamento, e o leitor sendo utilizado para ler o nome do proprietário. Note que os índices estão variando no for de 1 ao tamanho de cada dimensão, e não de 0 até o tamanho -1. Isso foi proposital, pois como os andares começam do 1 (101, 102, 103, 201, ...), na hora de imprimir o valor do índice, podemos fazer de forma direta se eles começarem com o valor 1. Entretanto, a indexação da matriz requer o valor do índice -1, para fazer com que ele comece do 0 e vá até o tamanho da dimensão -1.

Já a segunda parte do programa imprime os nomes dos proprietários dos apartamentos, fazendo, porém, o uso de índices que variam de 0 ao tamanho da dimensão -1. Isso faz com que essas variáveis possam ter seus valores usados diretamente para indexar a matriz. Porém, na hora de imprimir na tela o número do apartamento, você precisa usar o valor da variável + 1, entendeu? As duas abordagens são válidas, você pode escolher e usar aquela que você acha mais simples de entender.

**Vídeo 04** - Matriz 5x5

## Atividade 04

1. Faça um programa que leia uma matriz MxN, onde M e N são informados pelo usuário, e que imprima o maior valor encontrado na matriz.

# 5. Leitura Complementar

Para um maior aprofundamento do assunto, leia o conteúdo do seguinte site, o qual descreve o funcionamento dos arrays, mostrando alguns exemplos de uso simples:

<http://download.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

# 6. Resumo

Nesta aula, você estudou dois assuntos, vetores e matrizes. Você viu e estudou os exemplos da inicialização dessas estruturas de dados homogêneas, seu funcionamento e seu uso. Esperamos que ao final desta aula você tenha executado, sem grandes problemas, os exercícios que foram propostos ao longo da aula.

Para a próxima aula, nos aprofundaremos no tipo de dados String, que pode ser visto como um vetor de caracteres. Até lá!

# 7. Autoavaliação

1. Qual a estrutura geral para declararmos vetores unidimensionais?
2. Qual a estrutura geral para declararmos matrizes?
3. Quando devemos usar vetores unidimensionais e quando devemos usar matrizes?
4. Realize a leitura de duas matrizes de inteiros, a primeira chamada de A de dimensões 3 x 4 e a segunda chamada de B de dimensões 3 x 2. Construa uma matriz C de dimensões 3 x 6, sendo essa a junção das duas outras matrizes. Para isso, você deve copiar os elementos das matrizes A e B para a matriz C.

# 8. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.