JAVA

VETORES

Prof. Benefrancis do Nascimento

TÓPICOS

- **Definição**
- **Criando, Inicializando e acessando um Array unidimencional (vetor)**
- **Vamos codificar!**
- **Atividades Complementares**

DEFINIÇÃO

Não dá para se falar de vetor sem antes definir o que vem a ser o ***array***.

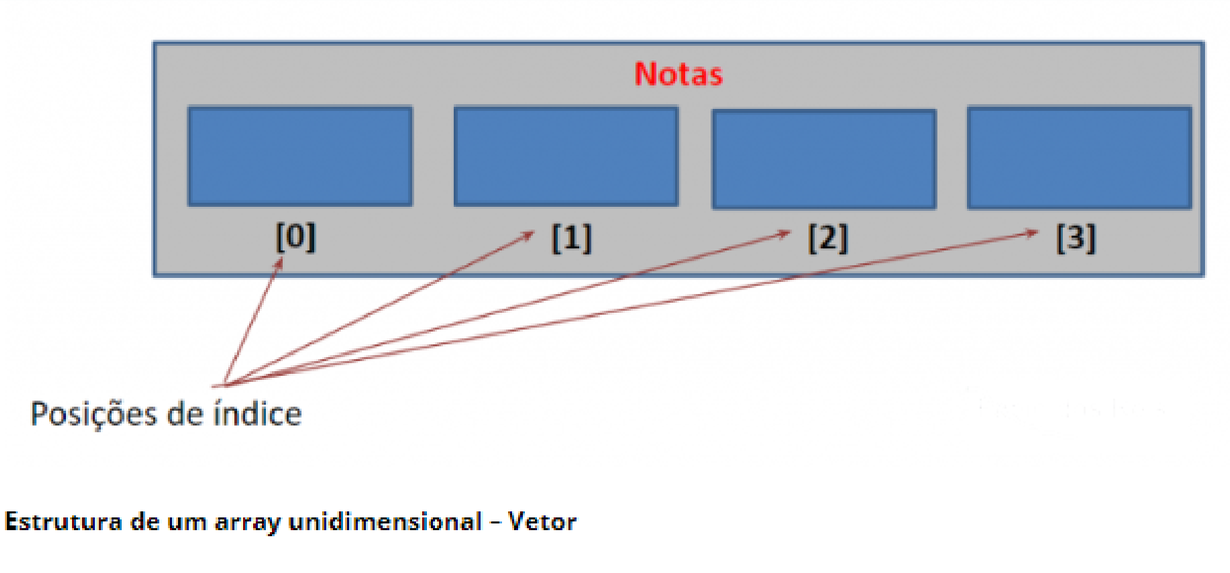
Um ***array*** é uma estrutura de dados homogênea que mantém uma série de elementos de dados de mesmo tipo. Pode-se acessar os elementos individuais armazenados no array por meio de uma posição de índice associada, geralmente numérica.

Os arrays são classificados de acordo com a sua dimensão de armazenamento de dados, como segue:

- **Unidimensional**: Vetor
- **Bidimensional**: Matriz
- **Tridimensional**: Cubo

Um vetor é um array unidimensional, ou seja, de uma única dimensão; é análogo a uma linha única de dados em uma planilha ou tabela.

DEFINIÇÃO



Estrutura de um array unidimensional – Vetor

A figura ao lado ilustra a estrutura interna de um vetor de quatro posições, que permite portanto armazenar até quatro dados, de nome Notas.

No geral a contagem das posições se inicia em zero (0), de modo que a primeira posição do vetor será a posição 0, a segunda posição será 1, e assim por diante; a última posição do vetor será a de número n – 1, onde n é o número total de posições disponíveis (tamanho do array). Assim, em um vetor de 4 posições a última posição será 4 – 1 = 3.

As posições em um vetor são sempre indicadas pelo número da posição entre colchetes [ ].

PONTOS IMPORTANTES

Um array é portanto um grupo de variáveis semelhantes (do mesmo Tipo ou Classe) referidas por um nome comum. Dito isso, devemos ater aos seguintes pontos importantes sobre arrays:

- Em Java, todos os arrays são alocadas de forma dinâmica;
- Os arrays são armazenadas na **Memória Heap;**
- Como **os arrays são objetos em Java**, podemos encontrar seu comprimento usando o comprimento da propriedade do objeto. Isto é diferente do C/C++, onde encontramos o comprimento usando sizeof;
- Uma variável de array Java também pode ser declarada como outras variáveis com **[]** após o tipo de dado. Por exemplo: int[] nomeDoArray;
- As variáveis no array são ordenadas, e cada uma tem um índice começando em 0;
- O array Java também pode ser usado como um campo estático, uma variável local ou um parâmetro de método;
- O tamanho de um array deve ser especificado por **int** ou **short**, mas não com **long;**
- A superclasse direta de um tipo de array é **Object;**
- Cada tipo de array implementa as interfaces **Cloneable** and java.io.**Serializable;**
- Este armazenamento de matrizes nos ajuda a acessar aleatoriamente os elementos de uma matriz [Support Random Access];
- O tamanho do array não pode ser alterado (uma vez inicializado). Entretanto, uma referência de array pode ser feita para apontar para outro array.

TOME NOTA!

Os elementos de um array alocados pelo comando new serão automaticamente inicializados com o valor zero (para tipos numéricos), falsos (para booleanos), ou nulos (para tipos de referência).

A obtenção de um array é um processo de dois passos. Primeiro, você deve declarar uma variável do tipo de array desejado. Segundo, você deve alocar a memória para manter o array, usando o comando new, e atribuí-la à variável array.

Assim, em Java, todos os vetores são alocadas dinamicamente.

CRIANDO, INICIALIZANDO E ACESSANDO UM ARRAY UNIDIMENCIONAL (VETOR)

tipo varName[];

ou

tipo[] varName;

Uma declaração de array tem dois componentes: o **tipo** e o **nome**. tipo declara o tipo de elemento do array. O tipo de elemento determina o tipo de dados de cada elemento que compreende o vetor. Como um array de inteiros, também podemos criar um array de outros tipos de dados primitivos como **char**, **float**, **double**, **short**, **long**, etc., ou tipos de dados definidos pelo usuário (objetos de uma classe). Assim, o tipo de elemento para o array determina que tipo de dados o array irá conter.

CRIANDO, INICIALIZANDO E ACESSANDO UM ARRAY UNIDIMENCIONAL (VETOR)

Veja o código a seguir:

intArray[] int;

Embora a primeira declaração estabeleça que o intArray é uma variável de vetor, não existe um vetor real. Ela simplesmente diz ao compilador que esta variável (intArray) conterá um array do tipo inteiro. Para ligar o intArray a um array de inteiros real e físico, você deve alocar um usando um novo e atribuí-lo ao intArray.

```
int intArray[];              //declarando um array de inteiros
intArray = new int[20];  // Alocando memória para o array declarado anteriormente.
```

ou

```
int[] intArray = new int[20]; // combinando os dois comandos (declaração e instanciação) em um só.
```

CRIANDO, INICIALIZANDO E ACESSANDO UM ARRAY UNIDIMENCIONAL (VETOR)

Array Literal

Em uma situação onde o tamanho da matriz e as variáveis da matriz já são conhecidos, podem ser usados os literais do array.

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
```

A quantidade de itens separados por vírgula, determina o **lenght** do vetor criado.

Não há necessidade de escrever **new int[]** nas versões mais recentes de Java.

CRIANDO, INICIALIZANDO E ACESSANDO UM ARRAY UNIDIMENCIONAL (VETOR)

Podemos acessar os elementos do vetor utilizando um laço de repetição (looping). Cada elemento do array é acessado através de seu índice. O índice começa com 0 e termina em (tamanho total da matriz)-1. Todos os elementos do array podem ser acessados usando o looping For.

Veja código abaixo:

```
int[] arr = {0,1,2,3,4,5,6,7,8,9};

for (int i = 0; i < arr.length; i++) System.out.println("Element at index " + i + : "+ arr[i]);
```

Pontos Importantes sobre Vetores

DIFERENTES FORMAS DE SE DECLARAR E INICIALIZAR

```
int[] intArray;
int intArray[];
int intArray[] = {0,1,2,3};
int[] intArray = new int[9];
```

DEFINIÇÃO

Os arrays são classificados de acordo com a sua dimensão de armazenamento de dados:

- Unidimensional: **Vetor**
- Bidimensional: **Matriz**
- Tridimensional: **Cubo**

ARRAYS SÃO TRATADOS COMO OBJETOS

Os arrays são armazenadas na Memória Heap;

A superclasse direta de um tipo de array é **Object;**

Os elementos de um array alocados pelo comando new serão automaticamente inicializados com o valor zero (para tipos numéricos), falsos (para booleanos), ou nulos (para tipos de referência).

TUDO TEM UM LIMITE!

O tamanho de um array deve ser especificado por valor short ou int e não long. Veja:

```
long valor;
int[] vetor = new int[valor]; //Erro de compilação
```

Copiando o Conteúdo de um Array em Java

Em diversas situações práticas, quando trabalhamos com arrays torna-se necessário realizar uma cópia de seu conteúdo para realizar algum tipo de processamento.
A linguagem Java oferece diferentes formas para copiar o conteúdo de um array para outro. Nós vamos exercitar este conceito agora. Primeiramente, vamos criar dois vetores com o código abaixo:

```
int [] a = new int[9999999];
int [] b = new int[a.length];
Random numeros = new Random();
for (int i=0; i < a.length; i++) a[i] = numeros.nextInt();
```

- IMPLEMENTANDO UM LAÇO COM O COMANDO FOR
- UTILIZANDO O MÉTODO CLONE.
- UTILIZANDO SYSTEM.ARRAYCOPY
- UTILIZANDO ARRAYS.COPYOF

Copiando o Conteúdo de um Array em Java

- IMPLEMENTANDO UM LAÇO COM O COMANDO FOR

```
for (int i=0; i < a.length; i++) b[i] = a[i];
```

- UTILIZANDO SYSTEM.ARRAYCOPY

```
System.arraycopy(a, 0, b, 0, a.length);
```

- UTILIZANDO O MÉTODO CLONE.

```
b = a.clone();
```

- UTILIZANDO ARRAYS.COPYOF

```
b = Arrays.copyOf(a, a.length);
```

Qual abordagem obtem o melhor desempenho?

Let's code!

Vamos codificar!

Vamos exercitar os conceitos sobre Vetores.

Copiando o Conteúdo de um Array em Java

Qual abordagem obtem o melhor desempenho?

Metodo 1 (comando for) -> tempo de processamento -> 11
Metodo 2 (método clone) -> tempo de processamento -> 22
Metodo 3 (método System.arraycopy) -> tempo de processamento -> 9
Metodo 4 (método Arrays.copyOf) -> tempo de processamento -> 13

A abordagem 1 e 3 **não reservam espaço em memória para o vetor destino** (ou seja, elas exigem que este espaço já tenha sido alocado, o que foi feito logo no início do nosso programa).

Desta forma, elas não gastam tempo com essa operação. Sendo assim, elas são indicadas no caso em que precisaremos copiar muitos vetores para um mesmo vetor temporário, pois nesse caso basta reservar espaço uma vez para o vetor temporário e usá-lo sempre que for preciso copiar qualquer array.

Para qualquer outra situação, não faz muita diferença utilizar qualquer um dos métodos.

Copiando o Conteúdo de um Array em Java

- ## IMPLEMENTANDO UM LAÇO COM O COMANDO FOR

A abordagem que faz a cópia usando o comando for é mais tradicional e tão eficiente quanto qualquer outra.

```
for (int i=0; i < a.length; i++) b[i] = a[i];
```

Copiando o Conteúdo de um Array em Java

- ## UTILIZANDO O MÉTODO CLONE.

  A sintaxe do método clone() é mais econômica e ele faz a duplicação do conteúdo do vetor origem automaticamente (não há necessidade de reservar espaço para o vetor destino) e não requer nenhum parâmetro. Porém se queremos copiar apenas parte do vetor origem não vale a pena – é melhor usar System.arraycopy. Se precisamos realizar diversas cópias para um vetor temporário instanciado uma única vez, também não vale a pena usar o método clone().

```
b = a.clone();
```

Copiando o Conteúdo de um Array em Java

- ## UTILIZANDO SYSTEM.ARRAYCOPY

  O método System.arraycopy é o mais flexível de todos, pois permite copiar qualquer seção do array origem para o array destino. Além disso, é o método mais eficiente na situação em que precisamos realizar diversas cópias para um vetor temporário instanciado uma única vez. A desvantagem do método é que ele necessita de muitos parâmetros.

  System.arraycopy(a, 0, b, 0, a.length);

Copiando o Conteúdo de um Array em Java

- ## UTILIZANDO ARRAYS.COPYOF

  O método Arrays.copyOf funciona de maneira similar ao método clone, oferecendo apenas o recurso de possibilitar a cópia dos n primeiros elementos do vetor origem.

  ```
  B = ARRAYS.COPYOF(A, A.LENGTH);
  ```

fonte: https://www.devmedia.com.br/copiando-o-conteudo-de-um-array-em-java/26732

Ordenando o conteúdo do Array

A maneira mais fácil

ORDEM CRESCENTE

```
int[] numbers = { 5, 4, 2, 1, 3 };

Arrays.sort(numbers);
for (int num : numbers){
    System.out.println(num);
}
```

ORDEM DECRESCENTE

```
int[] numbers = { 5, 4, 2, 1, 3 };

Arrays.sort(numbers, Collections.reverseOrder());
for (int num : numbers){
    System.out.println(num);
}
```

PARA SUA REFLEXÃO

Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende.

Leonardo da Vinci

Atividades Complementares

Conheça a sequência de Fibonacci:

A Sequência de Fibonacci consiste em uma sucessão de números, tais que, definindo os dois primeiros números da sequência como 0 e 1, os números seguintes serão obtidos por meio da soma dos seus dois antecessores. Portanto, os números são:

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181...

Fórmula

$$F(n) = \begin{cases} 0\,, & \text{se } n = 0\,; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Atividades Complementares

Assista aos vídeos abaixo

1) O que é a sequência de Fibonacci e o por que é chamada de 'código secreto da natureza'

[https://youtu.be/cHZWZhHQq4g]

2) O número usado por DEUS para criar o MUNDO

[https://youtu.be/e1-rL3KZALk]

3) Donald no País da Matemágica e O Número de Ouro

[https://youtu.be/g8oqgrVhA_8]

Atividades Complementares

Agora que você já conheceu a sequência de fibonacci, implemente programa para calcular a sequência de fibonacci guardando os valoes em um array numérico. Respeite a fórmula abaixo:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Bons estudos.

Agradeço!

profbenefrancis.nascimento@fiap.com.br

Baixe os códigos desta aula em:

https://github.com/Benefrancis/fiap