Arrays em Java

Características dos Arrays em Java

Algumas características dos arrays em Java são listadas a seguir:

- Em Java, todos os arrays são alocados dinamicamente
- Uma variável array em Java pode ser declarada como qualquer outra variável, acrescentando colchetes [] após o tipo de dados declarado.
- Arrays são objetos em Java. Assim, podemos descobrir seu tamanho usando o atributo length do objeto.
- Os elementos em um array são indexados a partir de zero.
- Um array pode ter uma ou mais dimensões
- O tamanho de um array sempre deve ser um valor int, e nunca um long, short ou byte.
- Podemos usar arrays em Java como um campo estático, variável local ou ainda como parâmetros em métodos.
- Os tipos de array implementam as interfaces Cloneable e java.io.Serializable
- Um array pode conter tipos de dados primitivos ou ainda objetos de uma classe

Criando um array unidimensional: Vetor

Declaramos um array de uma dimensão (“vetor”) da seguinte forma:

```
tipo nomeArray[];
ou
tipo[] nomeArray;
```

O **tipo** determina qual será o tipo de dados de todos os elementos que serão armazenados nas posições do array (estrutura homogênea).

Exemplo de declaração de array:

```
double[] salarios;

double[] salario[]; //ambas as formas são válidas
```

Neste exemplo, declaramos um array de nome **salario** e tipo **double** em um programa. Porém, neste ponto, nenhum array ainda existe de fato. Essa declaração apenas diz ao compilador que a variável salario irá armazenar um array do tipo double. Para que essa variável seja ligada ao array real, será necessário alocar um array usando o operador new, e atribuindo-o à variável salário.

Desta forma, quando um array é declarado, apenas uma referência de array é realmente criada. Para efetivamente alocar espaço na memória para o array, devemos proceder desta forma:

```
nomeArray = new tipoDado[tamanho];
```

Onde tamanho se refere ao número de elementos do array, e nomeArray é o nome da variável que será ligada ao array físico. Quando usamos o operador ***new*** devemos especificar o número de elementos que serão alocados no array.

Portanto, criar um array em Java é um processo em duas etapas: Declaramos uma variável do tipo desejado para o array, e então alocamos memória para efetivamente armazenar o array, atribuindo-a à variável criada.

Exemplo:

```
double[] salarios; //Declara o array
salarios = new double[50]; //Alocar memória para o array
```

Podemos ainda declarar a variável de referência, criar o array, e atribuir a referência do array à variável de uma só vez, como segue:

```
tipoDado[] variavelReferencia = new tipoDado[tamanhoArray];
```

Nosso exemplo:

```
double[] salarios = new double[50];
```

Assim, declaramos e criamos um array de 50 posições para armazenar dados do tipo double.

Os elementos do array alocados com o operador new são automaticamente inicializados com valores-padrão, a saber:

- **zero** para tipos numéricos
- **falso** para tipo boolean
- **null** para tipos de referência (incluindo String

Array literal

Quando sabemos de antemão o tamanho do array e quais serão os valores que serão armazenados nas posições do array, podemos usar literais de array – ou seja, podemos atribuir esses valores diretamente na declaração do array, usando uma ***lista de inicialização***. Para isso podemos usar a seguinte sintaxe:

```
tipoDados[] variavelReferencia = {valor1, valor2, ..., valorN};
```

Por exemplo, suponha um array de 5 posições do tipo inteiro, do qual sabemos de antemão de será necessário armazenar os números 5, 9, 12, 3 e 4. podemos declarar esse array da seguinte forma:

```
int[] numeros = { 5, 9, 12, 3, 4 };
```

Desta forma o array é criado, e suas cinco posições já serão preenchidas com os dados desejados. Neste caso, não é necessário usar o operador new para criar o array (exceto em versões muito antigas do Java).

Como acessar os elementos de um array em Java (unidimensional)

Como acessar os elementos de um array em Java

Vamos falar agora sobre como atribuir valores e acessar os elementos armazenados em um array, usando técnicas variadas.
Os elementos em um array são sempre acessados por meio de seu número de índice (que é a posição do elemento). Esse índice sempre se inicia em zero, e termina em **tamanhoArray – 1**.
Por exemplo, um array de dez posições tem a sua primeira posição com índice zero e sua última posição com 10 -1 = índice 9. Portanto, se trata de um array cujas posições vão de 0 a 9.

Acessando e alterar posições individuais em um array (vetor)

Podemos acessar um elemento individual em um array simplesmente indicando o número de índice da posição desejada (entre colchetes), junto ao nome do array em si. Veja o exemplo:

```
// Criar o array e atribuir-lhe valores a partir de uma lista de inicialização:
double[] valores = { 4.5, 5.9, 4.1, 2.0, 8.9, 6.3, 7.8, 5.3, 1.2, 0.8 };

// Acessando seu quinto elemento (número de posição 4)
System.out.println("Elemento 5 do array: " + valores[4]);
```

Resultado:

```
Elemento 5 do array: 8.9
```

O mesmo vale para a atribuição de valores ao array. Por exemplo, vamos modificar os valores das posições de índices 4 e 6 para 2.3 e 7.1, respectivamente:

```
valores[4] = 2.3;
valores[6] = 7.1;
```

```
// Acessando os elementos alterados:
System.out.println("Valor alterado para: " + valores[4]);
System.out.println("Valor alterado para: " + valores[6]);
```

A seguir temos o código completo usado nos exemplos anteriores para leitura e escrita de valores em um array:

```
// Criar o array e atribuir-lhe valores a partir de uma lista de inicialização:
```

```
double[] valores = { 4.5, 5.9, 4.1, 2.0, 8.9, 6.3, 7.8, 5.3, 1.2, 0.8 };

// Acessando seu quinto elemento (número de posição 4)
System.out.println("Elemento 5 do array: " + valores[4]);

// Alterando os valores de duas posições no array:
valores[4] = 2.3;
valores[6] = 7.1;

// Acessando os elementos alterados:
System.out.println("Valor alterado para: " + valores[4]);
System.out.println("Valor alterado para: " + valores[6]);
```

Resultado:

```
Elemento 5 do array: 8.9
Valor alterado para: 2.3
Valor alterado para: 7.1
```

Acessando todas as posições de uma vez via laço for

Podemos acessar todas as posições de um array usando um simples laço for e a propriedade **length** do array, como segue:

```
for (int i = 0; i < nomeArray.length; i++) {
  Código a executar para cada elemento
}
```

Exemplo: Vamos criar um array de números double e acessar seus elementos usando um laço for.

```
// Criar array e atribuir-lhe valores a partir de uma lista de inicialização:

double[] valores = { 4.5, 5.9, 4.1, 2.0, 8.9, 6.3, 7.8, 5.3, 1.2, 0.8 };

// Mostrar todos os elementos do array:
for (int i = 0; i < valores.length; i++) {
   System.out.println("Elemento " + i + " = " + valores[i]);
}
```

Resultado:

```
Elemento 0 = 4.5
Elemento 1 = 5.9
Elemento 2 = 4.1
Elemento 3 = 2.0
Elemento 4 = 8.9
Elemento 5 = 6.3
Elemento 6 = 7.8
```

```
Elemento 7 = 5.3
Elemento 8 = 1.2
Elemento 9 = 0.8
```

Acessando todas as posições do array usando laço foreach

Vamos mostrar todos os elementos do array valores usando um laço **foreach** (*for enhanced* em Java):

```
public static void main(String[ ] args) {
        double[] valores = { 4.5, 5.9, 4.1, 2.0, 8.9, 6.3, 7.8, 5.3, 1.2, 0.8 };

    // Mostrar todos os elementos do array:
    for (double elemento: valores) {
     System.out.println(elemento);
  }
}
```

Usamos neste exemplo uma declaração ***for enhanced***, a qual itera pelos elementos de um array sem usar um contador.

Observação: Cuidado ao usar um laço foreach para iterar pelos elementos de um array em Java. Essa técnica não é apropriada quando queremos modificar o array, além de não manter registro do número de índice que está sendo acessado.