

Programação Orientada a Objetos

Array, Coleções e Exceções

Rone Ilídio
Thiago Oliveira



Array

- O *array* é um grupo de posições contíguas que possuem o mesmo nome e o mesmo tipo.
- Quando se aloca um *array*, seus elementos recebem, por *default*:
 - 0 para variáveis numéricas,
 - *false* para *boolean* ou
 - *null* para qualquer tipo não primitivo.



Array – Declaração

- O operador *new* é utilizado para alocar espaço na memória.
- Para declarar um *array* de inteiros de 12 posições:
 - `int vetor[] = new int[12];`
- Similar à operação anterior:
 - `int[] vetor;`
 - `vetor = new int[12];`



Array – Inicialização e Tamanho

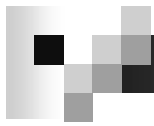
- Para criar um vetor com valores, utiliza-se a sintaxe:
 - `int[] valores = {10,20,30,40,50};`
 - Resultado é um *array* de 5 posições (0 a 4).
 - A primeira posição é sempre 0.

- Todos os *arrays* em Java conhecem seu próprio comprimento, através de:
 - `nome_array.length`



Array – Acesso a Posições

- Para referir-se a uma localização ou elemento particular no *array*:
 - Especificamos o nome do *array* e o número da posição (índice ou subscrito) do elemento.
 - Índice deve ser um inteiro ou expressão inteira.
- Exemplo:
 - `int valores[] = {10,20,30,40,50};`
 - `int posicao = valores[3];`
 - A variável `posicao` terá valor igual a 40.




```
import javax.swing.*;
public class Vetor{
    public static void main(String[] args){
        int[] vet = new int[5];
        String entrada;

        for (int i=0; i<=4; i++ ){
            entrada = JOptionPane.showInputDialog("Informe um valor");
            vet[i] = Integer.parseInt(entrada);
        }
        for (int i=0; i<5; i++ ){
            System.out.print( i + " - " + vet[i] + "\n");
        }
    }
}
```



Arrays Multidimensionais

- Também são chamados de matrizes.
- Java não aceita diretamente array multidimensionais.
- Porém, aceita arrays formados por elementos que também são arrays.
- Exemplo: `int[][] matriz = new int[3][4];`
- A declaração gera um array multidimensional com 3 linhas e 4 colunas.



```
import javax.swing.*;
public class Matriz{
    public static void main(String[] args){
        int[][] matriz = new int[3][2];
        String entrada;
        for (int i=0; i<3; i++ ) {
            for (int j=0; j<2; j++) {
                entrada = JOptionPane.showInputDialog("Informe um valor");
                matriz[i][j] = Integer.parseInt(entrada);
            }
        }
        String saida = "";
        for (int i=0; i<3; i++ ) {
            for (int j=0; j<2; j++) {
                saida += matriz[i][j] + " ";
            }
            saida += "\n";
        }
        JOptionPane.showMessageDialog(null, saida);
    }
}
```




Exercícios

- Crie um programa onde o usuário informe todos os elementos de um vetor (10 posições) e obtenha como resultado qual é o maior elemento.
- Crie um aplicativo com um vetor de elementos preenchidos na sua declaração.
 - ☐ Deve receber do usuário um número e verificar se tal número existe ou não no vetor.
- Crie um *applet* que receba do usuário uma sequência de números. Ele deverá imprimir essa sequência de números ordenada.




Exercícios

- Crie um aplicativo que contenha uma matriz 3x3, de inteiros.
 - ☐ Deve ser inicializada com valores vindos do usuário.
 - ☐ Ao final, essa matriz deverá ser impressa na tela.
- Crie um *applet* com uma matriz 4x4 de inteiros.
 - ☐ Preencha-a gerando números aleatórios entre 0 e 100.
 - ☐ Ao final, deve ser impressa a soma da diagonal principal desta matriz.



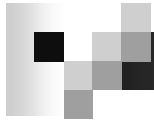
Array – Parâmetro de métodos

- Quando um *array* é passado como parâmetro para um método, essa passagem é sempre feita por referência.
- Na declaração do método não é necessário especificar o tamanho do vetor.
- Toda alteração feita no método será gravada em memória.
- Veja um exemplo:



```
import javax.swing.*;

public class MetodoArray extends JApplet{
    public void init(){
        int[] vet_teste = {3,34,3,87,3,4};
        dobra_array(vet_teste);
        String saida = "";
        for(int i=0; i<vet_teste.length; i++) {
            saida = saida + " " + vet_teste[i];
        }
        JOptionPane.showMessageDialog(null, saida);
    }
    public void dobra_array(int[] vetor){
        for(int i=0; i<vetor.length; i++)
            vetor[i] = vetor[i] * 2;
        }
    }
}
```



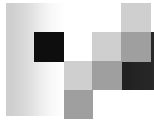
Coleções

- Coleção simples: Array

- ☐ Precisa definir tamanho.
- ☐ Remoção altera posições.

- Exemplos:

- ☐ `int[] vetorInteiros = { 1, 2, 3, 4, 5, 6 };`
- ☐ `double[] vetorDouble = { 8.4, 9.3, 0.2, 3.4, 7.9 };`
- ☐ `Pessoa[] pessoas = new Pessoa[12];`



Coleções

- **Collection:** interface base (“pai”) de Set, List e Queue.
- **List:** uma coleção ordenada onde podem existir elementos duplicados.
- **Set:** conjunto onde elementos duplicados não são permitidos.
- **Map:** conjunto com chave primária e elementos, chaves duplicadas não são permitidas.
- **Queue:** fila de elementos, estilo “primeiro que entra é o primeiro que sai”.



Interface List

```
/*  
 * Classes ArrayList, TreeSet e Vector  
 */  
List< String > lista = new ArrayList<String>();  
  
lista.add("PROVA");  
lista.add("LISTA");  
lista.add("TRABALHO");  
lista.add("PROVASUB");  
  
System.out.println("LISTA:: Meu ArrayList: " );  
for (String elemento: lista)  
{  
    System.out.println(elemento);  
}  
  
lista.remove("PROVASUB");
```



Interface List

// Continua...

```
System.out.println("LISTA:: Meu ArrayList modificado: ");
```

```
for (int i=0; i<lista.size(); ++i)
{
    System.out.print(lista.get(i) + "\n");
}
```

// OU ainda...

```
Iterator<String> it = lista.iterator();
```

```
while ( it.hasNext() )
{
    System.out.println(it.next());
}
```



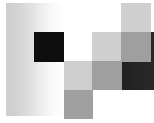

Interface Map

```
/*  
 * Classes HashMap, HashTable e TreeMap  
 */
```

```
Map< String, Integer > tabela = new HashMap< String, Integer >();
```

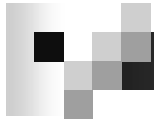
```
tabela.put("PROVA", 30);  
tabela.put("LISTA", 10);  
tabela.put("TRABALHO", 20);
```

```
System.out.println("MAP:: Exemplo de Map: ");  
System.out.println("Valor da PROVA: " + tabela.get("PROVA"));
```



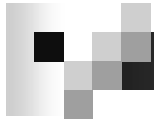
Tratamento de Exceções

- Nem todos os erros podem ser previstos em tempo de compilação.
- Existem situações inesperadas que podem ocorrer durante uma execução, como:
 - acesso a posições inválidas de vetores, utilização de objetos que ainda não foram inicializados, etc.
- Tais situações podem ser tratadas com a utilização de exceções.



Tratamento de Exceções

- Existem funcionalidades Java que devem obrigatoriamente ser utilizadas controlando-se exceções.
 - ☐ JDBC, banco de dados.
- Ações básicas que devem ser tomadas:
 - ☐ levantar uma exceção;
 - ☐ tratar uma exceção;
 - ☐ finalizar situação (opcional).



Tratamento de Exceções


- Uma exceção é:
 - levantada através de uma chamada de `try{...}`,
 - tratada por uma chamada de `catch (Exception){...}`
 - e finalizada com uma chamada para `finally{...}`.
- Se ocorrer uma exceção dentro do blocos de comandos delimitados por `try{...}`
 - Tal bloco terminará sua execução e os comandos do `catch` serão executados.
 - Ocorrendo ou não uma exceção, `finally` é chamado no final: operação opcional.



Tratamento de Exceções

■ Try/catch – sintaxe:

```
try {  
    ...comandos que podem ou não lançar exceções...  
} catch(ExcecaoTipoEspecifico parametro) {  
    ...tratamento para exceções desse tipo...  
} catch(ExcecaoTipo... parametro) {  
    ...tratamento para exceções de outros tipos...  
}  
finally {  
    ...comandos executados com ou sem lançar exceção...  
}
```



```
public class TesteTry {
    public static void main(String args[]) {
        int vetor[]=new int[4];
        try {
            for(int i=0; i<=4; i++) {
                vet[i]= i*i;
            }
            //repetição gera excecao, pois os indices válidos são: 0,1,2,3
            //qualquer codigo escrito depois desse for nunca será executado!
            System.out.println("Não imprime essa mensagem.");
        }
        catch(Exception e) {
            System.out.println("Acesso a posição inválida.");
        }
        finally{
            System.out.println("Executa de qualquer forma.");
        }
    }
}
```



```
import javax.swing.JOptionPane;
```

```
public class Excecao {  
    public static void main(String[] args) {
```

```
        boolean repete = false;
```

```
        do {
```

```
            try {
```

```
                String entrada = JOptionPane.showInputDialog("Digite um número");
```

```
                int numero = Integer.parseInt(entrada);
```

```
                JOptionPane.showMessageDialog(null, numero);
```

```
                repete = false;
```

```
            }
```

```
            catch (NumberFormatException e) {
```

```
                JOptionPane.showMessageDialog(null, "É pra digitar um número!");
```

```
                repete = true;
```

```
            }
```

```
            catch (Exception e) {
```

```
                JOptionPane.showMessageDialog(null, "Deu outro erro.");
```

```
            }
```

```
        } while (repete);
```

```
    }
```

```
}
```