

POO (MiEI/LCC)

2015/2016

Ficha Prática #02

Arrays

Conteúdo

1	Síntese teórica	3
2	Sintaxe essencial	4
2.1	Declarações, inicializações e dimensionamento	4
2.2	Comprimento e acesso aos elementos	5
2.3	Varrimento (acesso a todos os elementos)	5
2.4	Leitura de Valores para um array	6
2.5	Algoritmo de Procura	7
2.6	Cópia Entre Arrays	8
2.7	Métodos da class java.util.Arrays (tipo primitivos)	8
3	Exercícios	8

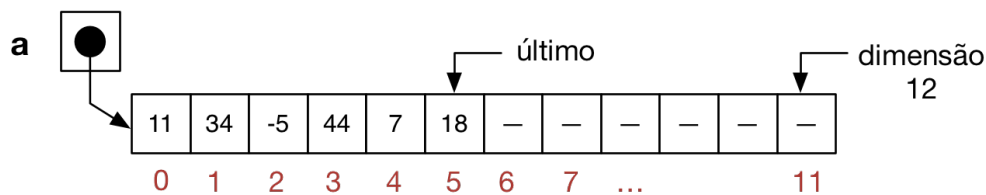
1 Síntese teórica

Os *arrays* de JAVA são estruturas lineares indexadas, ou seja, cada posição do *arrays* possui um endereço inteiro para acesso ao elemento nele contido (1º elemento no índice 0). Os *arrays* podem conter valores de tipos primitivos ou objectos. Os *arrays* de JAVA não são objectos, ou seja, não são criados por nenhuma classe nem respondem a mensagens. São no entanto de tipo referenciado, ou seja, a variável que identifica o *array* contém o endereço de memória do *array* (é uma referência).

Sendo estruturas lineares indexadas, os **elementos** de um *array* ocupam **posições** referenciáveis por um **índice** inteiro com valores a partir de 0.

A dimensão física de um *array*, também designada a sua **capacidade**, pode ser definida aquando da sua declaração ou posteriormente, mas é diferente do seu **comprimento**, que se associa ao número efectivo de elementos que, num dado momento, estão armazenados no *array*.

Para um *array* de dimensão d , o seu comprimento actual c será sempre $c \leq d$ e o índice do último elemento será sempre $c-1$. Para um *array* **a**, a instrução **a.length** devolve um inteiro que corresponde à sua dimensão actual, não o actual número de elementos. Para *arrays* numéricos, inicializados a 0 ou 0.0 há que ter cuidado com **length** pois os elementos a zero iniciais são contados também, e não correspondem a dados introduzidos. Assim, sempre que o número de elementos não coincida garantidamente com a dimensão, uma variável que conte os elementos efectivos do *array* deverá ser usada.



A dimensão física de um *array*, bem como o tipo dos seus elementos, são em geral definidos aquando da sua declaração, como em:

```
1 | int[] vector = new int[100];
```

A dimensão pode, no entanto, ser definida posteriormente, usando a construção **new**, não podendo o *array* ser usado enquanto tal dimensão não for especificada.

```
1 String[] nomes;  
2 nomes = new String[50];
```

A capacidade/dimensão definida para um *array* é fixa, ou seja, é imutável ao longo da execução do programa. A capacidade pode ser também definida de forma implícita e automática através da sua inicialização, sendo, neste caso, a capacidade do *array* igual ao número de elementos introduzidos na sua inicialização, cf. o exemplo:

```
1 int[] valores={12,56,-6,45,56,8}; //dim=6  
2 double[] notas = { 12.5, 15.6, 10.9, 15.2, 6.6, 8.7, 9.0,  
    11.1 }; // dim = 8
```

Os *arrays* podem ser multidimensionais (linhas, colunas, etc.) e não apenas a uma só dimensão (linha). Os *arrays* monodimensionais são muitas vezes referidos como vectores.

Os *arrays* multidimensionais são em geral referidos como matrizes. O número de dimensões de um *array* é clarificado na sua definição, pois cada dimensão corresponde sintacticamente a mais um `[]`.

```
1 int[][] matriz_valores = new int[20][50]; // matriz de  
    20 linhas por 50 colunas  
2 double[][] notas = new double[3][300]; // matriz 3  
    testes por 300 alunos  
3 double[][][] temps = new double[15][12][31]; // cidades x  
    meses x dias e temperaturas
```

2 Sintaxe essencial

2.1 Declarações, inicializações e dimensionamento

```
1 int lista[]; // estilo C  
2 int[] lista; // estilo Java  
3  
4 int[] turma = new int[100];  
5 double[] medias = new double[50];  
6 byte[] mem = new byte[800*600];  
7  
8 short matriz[][] = new short[10][50];
```

2.2 Comprimento e acesso aos elementos

```
9 short matx[][] = new short[10][]; // 2a dimensão é variável
10 matx[0] = new short[15]; matx[1] = new short[40];
11
12 String[] nomes = new String[20];
13 String[] alunos = { "Pedro", "Rita", "Ana" };
14 String[][] linhas = {{ "A", "minha"}, {"casa", "tem", "um"},
    {"rio"}};
15 String[][] galo = { {"O", "O", "X"},
16                     {"X", "X", "O"},
17                     {"O", "X", "O"} };
18 Ponto[] plano = new Ponto[200];
19 Object obj[] = new Object[100];
```

2.2 Comprimento e acesso aos elementos

```
1 // comprimento
2 int comp = lista.length;
3 int numAlunos = alunos.length;
4
5 // acesso
6 int val = lista[0];
7 int num = lista[val*2];
8 short snum = matx[5][3];
9 String nome = nomes[index];
10 String pal = linhas[l][c];
11 out.println(lista[i]);
12 out.println(nomes[i]);
13 out.printf("Val = %d\n", lista[i]);
```

2.3 Varrimento (acesso a todos os elementos)

```
1 for(int i = 0; i <= a.length-1; i++) { ...a[i]....} // por
    índice
2 for(IdTipo elem : IdArray) { ...elem ... } //
    for(each)
3
4 // Imprimir todos os elementos de um array
5 for(int i=0; i< lista.length; i++) out.println(lista[i]);
6 for(int elem : lista) out.println(elem);
7
8 // Exemplos de somatórios
9 int soma = 0;
10 for(int i=0; i< lista.length; i++) soma = soma + lista[i];
11
12 int soma1 = 0;
13 for(int elem : lista) soma1 += elem;
14
15 // Exemplos de concatenação de strings
16 String total = "";
```

2.4 Leitura de Valores para um array (MiEI+LCC) Ficha Prática #02

```
17 for(int i=0; i < nomes.length; i++) { total = total +
    nomes[i]; }
18
19 String total = "";
20 for(String nome : nomes) { total += nome; }
21
22 // Contagem de pares e ímpares num array de inteiros
23 int par = 0, impar = 0;
24 for(int i = 0; i < a.length; i++)
25     if (a[i]%2 == 0) par++;
26     else impar++;
27 out.printf("Pares = %d - Impares = %d\n", par, impar);
28
29 // Total de inteiros > MAX de um array de arrays de inteiros
30 int maiores = 0;
31 int MAX = Integer.MIN_VALUE;
32 for(int l = 0; l < nums.length; l++) {
33     for(int c = 0; c < nums[l].length; c++)
34         if (nums[l][c] > MAX) maiores++;
35 }
36
37 // Concatenação de strings de um array bidimensional
38 String[][] nomes = { {"Rita", "Pedro"}, ..... };
39 String sfinal = "";
40 for(int l = 0; l < nomes.length; l++) {
41     for(int c = 0; c < nomes[l].length; c++) sfinal +=
        nomes[l][c];
42 }
43
44 // usando for(each)
45 sfinal = "";
46 for(String[] lnomes : nomes)
47     for(String nome : lnomes) sfinal += nome;
```

2.4 Leitura de Valores para um array

```
1 // Ler um número n, dado pelo utilizador, de valores de dado
   tipo, e guardá-los sequencialmente num array
2 Scanner sc = new Scanner(System.in);
3 int valor = 0;
4 out.print("Quantos números inteiros quer introduzir ? ");
5 int n = sc.nextInt();
6 for(int i = 0; i <= n-1; i++) {
7     valor = sc.nextInt();
8     lista[i] = valor;
9 }
10 // ou ainda, de forma mais simples mas equivalente:
11 int n = sc.nextInt();
12 int valor = 0;
```

```
13 for(int i = 0; i <= lista.length-1; i++) lista[i] =
    sc.nextInt();
14
15 // Ler um número não previamente fixado de valores de dado
    tipo e guardá-los num array pela sua ordem de leitura;
    Terá sempre que existir uma condição de paragem da
    leitura, seja porque foi lido um valor definido como
    valor de paragem (flag), seja porque o array já não tem
    mais capacidade.
16 int VALOR_STOP = -9999; // valor que serve de sentinela/flag
    para parar a leitura
17 int[] lista = new int[MAXDIM]; // MAXDIM é uma constante
    predefinida no programa
18 boolean stop = false;
19 int conta = 0;
20 int valor;
21 while(!stop && conta<=MAXDIM-1) {
22     valor = sc.nextInt();
23     if(valor == VALOR_STOP)
24         stop = true;
25     else {
26         lista[conta] = valor;
27         conta++;
28     }
29 }
```

2.5 Algoritmo de Procura

```
1 // Procura de um valor lido (chave) num array, dando como
    resultado a sua posição, ou -1 caso não seja encontrado.
2 int[] lista = new int[MAXDIM]; // MAXDIM é uma constante
    predefinida no programa
3 ..... // leitura ou inicialização
4 int chave;
5 boolean encontrada = false;
6 int indice = 0;
7 int pos = -1;
8 Scanner sc = new Scanner(System.in);
9 out.print("Qual o valor a procurar no array? : ");
10 chave = sc.nextInt();
11 while(!encontrada && indice<=MAXDIM-1) {
12     if(lista[indice] == chave) {
13         encontrada = true;
14         pos = indice;
15     }
16 }
17 out.println("Valor " + chave + " encontrado na posição " +
    pos);
```

2.6 Cópia Entre Arrays

```
1 System.arraycopy(array_fonte, índice_inicial_f,
   array_destino, índice_inicial_d, quantos);
2 System.arraycopy(a, 0, b, 0, a.length); // a.length
   elementos de a[0] para b desde b[0]
3 System.arraycopy(lista, 5, lista1, 1, 4); // 4 elems a
   partir de lista[5] para lista1 desde 1
```

2.7 Métodos da class java.util.Arrays (tipo primitivos)

```
1 int binarySearch(tipo[] a, tipo chave); // devolve índice da
   chave, se existir, ou < 0;
2 boolean equals(tipo[] a, tipo[] b); // igualdade de arrays
   do mesmo tipo;
3 void fill(tipo[] a, tipo val); // inicializa o array com o
   valor parâmetro;
4 void sort(tipo[] a); // ordenação por ordem crescente;
5 String toString(tipo[] a); // representação textual dos
   elementos;
6 String deepToString(array_multidim); // repres. textual para
   multidimensionais;
7 boolean deepEquals(array_multi1, array_multi2); // igualdade
   de arrays multidim;
```

3 Exercícios

1. Declarar, inicializar e imprimir os elementos de um *array* de inteiros.

```
1 // declarar, inicializar e imprimir os elementos de um
   array
2 int[] lista = {12, 2, 45, 66, 7, 23, 99};
3 System.out.println("--- ELEMENTOS DO ARRAY ---");
4 for(int i = 0; i < lista.length; i++) {
5     System.out.println("Elemento "+ i + " = " + lista[i]);
6 }
7 System.out.println("-----");
8
9 // solução alternativa usando método da classe Arrays
10 int[] lista = {12, 2, 45, 66, 7, 23, 99};
11 out.println(Arrays.toString(lista));
```

2. Escrever um programa que faça a leitura de N valores inteiros para um *array* e determine qual o menor valor introduzido, e qual a sua posição no *array*.

O programa deverá estruturado recorrendo a dois métodos auxiliares:

- `private static int[] lerArrayInt(int n)` – para ler um *array* de N inteiros;
- `private static int minPos(int[] arr)` – para determinar a posição do mínimo de um *array* de inteiros.

3. Altere o método `lerArrayInt` por forma a que, durante a leitura e inserção dos N elementos, o *array* se mantenha sempre ordenado por ordem decrescente.

4. Escreva e teste o método

`private static int[] subArray(int[] arr, int i, int f)`

que, dado um *array* e dois índices válidos do *array*, retorna um novo *array* apenas com os elementos entre esses dois índices.

5. Escreva e teste o método

`private static int[] arrayConcat(int[] arr1, int[] arr2)`

que recebe dois *arrays* e retorna o *array* resultado de os concatenar. O *array* resultante deve estar ordenado por ordem crescente.

6. Escrever um programa que leia uma série de palavras para um *array*, até ser introduzida a palavra "fim". De seguida, deve ser efectuada a leitura de um caracter. Se o caracter lido for 's', então ler duas palavras, e substituir a primeira pela segunda no *array*. Se o caracter lido for 'r', ler uma palavra e remover essa palavra do *array*. Utilizar métodos auxiliares.

7. Escrever um programa que leia para um *array* os vencimentos mensais dos 20 funcionários de uma empresa, e para um outro *array* da mesma dimensão os cargos respectivos dos funcionários. O programa possuirá uma tabela de bonificação salarial constante, como apresentado abaixo.

Cargo	% benefício
Developer	5
Gestor	10
Administrador	20
CEO	30

Pretende-se que o programa crie um *array* com os salários totais dos 20 funcionários.

8. Escrever um programa que simule o jogo do Euromilhões. O programa gera aleatoriamente uma chave contendo 5 números (de 1 a 50) e duas estrelas (1 a 9). Em seguida são pedidos ao utilizador 5 números e duas estrelas (a aposta). O programa deverá em seguida apresentar a chave gerada e o número de números e estrelas certos da aposta do utilizador. Naturalmente devem ser usados *arrays* para guardar os dados.
9. Modifique o programa do Exercício 7 de modo a que no final o programa apresente o somatório de todos os vencimentos e de todos os benefícios dos funcionários.