POO (LEI/LCC)

2014/2015

Ficha Prática #02

Arrays

Conteúdo

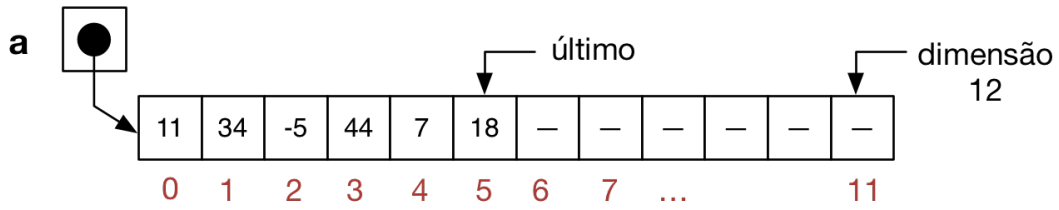
1 Síntese teórica

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 <= 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 quer 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:

```
int[]valores={12,56,-6,45,56,8}; //dim=6
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 `[]`.

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

2 Sintaxe essencial

2.1 Declarações, inicializações e dimensionamento

```
int lista[]; // estilo C
int[] lista; // estilo Java

int[] turma = new int[100];
double[] medias = new double[50];
byte[] mem = new byte[800*600];

short matriz[][] = new short[10][50];
short matx[][] = new short[10][]; // 2a dimensão é variável
matx[0] = new short[15]; matx[1] = new short[40];

String[] nomes = new String[20];
String[] alunos = { "Pedro", "Rita", "Ana" };
String[][] linhas = {{"A", "minha"}, {"casa", "tem", "um"},
   {"rio"}};
String[][] galo = { {"O", "O", "X"},
                    {"X", "X", "O"},
                    {"O", "X", "O"} };
Ponto[] plano = new Ponto[200];
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 = "";
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.lenght; 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;
```

```
for(int l = 0; l < nums.length; l++) {
    for(int c = 0; c < nums[l].length; c++)
        if (nums[l][c] > MAX) maiores++;
}

// Concatenação de strings de um array bidimensional
String[][] nomes = { {"Rita", "Pedro"}, ................. };
String sfinal = "";
for(int l = 0; l < nomes.length; l++) {
    for(int c = 0; c < nomes[l].length; c++) sfinal +=
       nomes[l][c];
}

// usando for(each)
sfinal = "";
for(String[] lnomes : nomes)
    for(String nome : lnomes) sfinal += nome;
```

2.4 Leitura de Valores para um array

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

// 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.
int VALOR_STOP = -9999; // valor que serve de sentinela/flag para
   parar a leitura
int[] lista = new int[MAXDIM]; // MAXDIM é uma constante
   predefinida no programa
boolean stop = false;
int conta = 0;
int valor;
while(!stop && conta<=MAXDIM-1) {
    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  System.out.println("--------------------------");
7
8  // solução alternativa usando método da classe Arrays
9  int[] lista = {12, 2, 45, 66, 7, 23, 99};
10 out.println(Arrays.toString(lista));
```

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

3. Modificar o programa anterior de modo a que a leitura dos N elementos para um *array* de inteiros seja realizada usando um método auxiliar que recebe o valor de N como parâmetro.

4. Modificar o programa anterior de modo a que quer a leitura dos N elementos quer a determinação do máximo elemento do *array* sejam realizados em métodos auxiliares do método `main()`.

5. Escrever um programa que faça a leitura de N elementos inteiros para um *array*, mas que os insira de forma a que o *array* se mantenha sempre ordenado por ordem crescente.

6. Escrever um programa que faça a leitura de N elementos inteiros para um *array*, receba dois índices válidos do *array* lido e crie um *array* apenas com os elementos entre esses dois índices. Usar um método auxiliar.

7. Escrever um programa que leia uma série de palavras terminada por "zzz" para um *array*, aceite repetidamente uma palavra até que seja introduzida a palavra "xxx" e verifique se tal palavra existe no *array*. Caso exista o programa deverá removê-la do *array*.

8. Escrever um programa que leia para um *array* os vencimentos mensais brutos (ilíquidos) dos 20 funcionários de uma empresa. O programa possuirá uma tabela de retenção de IRS constante, do tipo

Salário Ilíquido	**% Retenção do IRS**
0 a 500 Euros	5
501 a 1000 Euros	10
1001 a 2000	20
2001 a 4000	30
4001 ou mais	40

Pretende-se que o programa crie um *array* no qual, para o respectivo funcionário cujo vencimento bruto se encontra no *array* lido, sejam introduzidos as respectivas retenções para IRS. No final, o programa deve apresentar uma listagem com os vencimento bruto, retenção de IRS e vencimento líquido para os 20 funcionários.

9. 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.

10. Modifique o programa do exemplo 9 de modo a que no final o programa apresente o somatório de todos os vencimentos e de todos os impostos retidos aos funcionários.