

POO (MiEI/LCC)

2020/2021

Ficha Prática #02

Arrays

Conteúdo

1	Arrays em Java	3
2	Sintaxe essencial	4
2.1	Declarações, inicialização e dimensionamento	4
2.2	Comprimento e acesso aos elementos	5
2.3	Percorrer um array	5
2.4	Máximo e mínimo de arrays de inteiros	6
2.5	Leitura de Valores para um array	7
2.6	Algoritmo de Procura	8
2.7	Cópia Entre Arrays	8
2.8	Métodos da class java.util.Arrays (tipo primitivos)	9
3	Exercícios	9

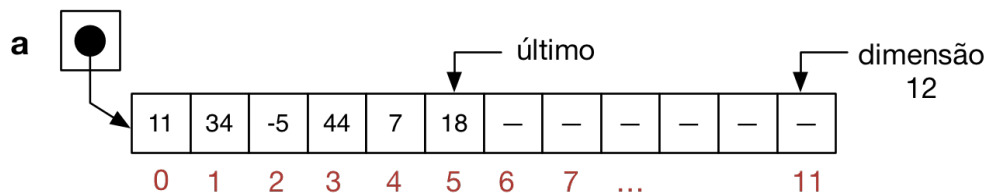
1 Arrays em Java

Os *arrays* em Java são a estrutura de dados básica para agregar uma colecção de entidades do mesmo tipo de dados. Os *arrays* são estruturas lineares indexadas, em que a cada posição do mesmo está associado um índice para aceder ao elemento nele contido. Tal como em C, os *arrays* em Java começam no índice 0, onde está guardado o primeiro elemento do *array*. Em Java o limite do *array* é testado aquando do acesso e se o índice não existir obtém-se um erro, ao invés de outras linguagens em que teremos uma referência para null.

Os *arrays* podem conter valores de tipos primitivos ou objectos. Tal como em C o nome do *array* não é mais do que um apontador para a posição de memória onde este está guardado.

A dimensão física de um *array* é determinada aquando da sua construção e determina a capacidade máxima que o *array* consegue armazenar. Note-se que este valor é diferente do número de elementos que, num determinado momento, se encontram no *array*.

Para um *array* **alunos**, a invocação **alunos.length** devolve um inteiro que corresponde à sua dimensão, isto é, o número de elementos máximo com que foi alocado. Quando for necessário efectivamente saber quantos elementos estão no *array* será necessário ter uma variável que faça essa contagem.



O tamanho de um *array*, bem como o tipo de dados dos seus elementos, são definidos aquando da sua declaração, como em:

```
1 | int[] colecao = new int[100];  
  
1 | String[] turma;  
2 | turma = new String[30];
```

Neste caso, o *array* *colecao* foi criado para conter elementos do tipo inteiro e foi alocado o espaço para 100 elementos. O *array* é inicializado com o valor por omissão dos inteiros, isto é, com 0 (zeros).

Não é possível alterar a dimensão de um *array* depois de este ter sido criado. A única forma de fazer um *array* crescer é alocar um novo *array* e copiar para lá os elementos do *array* original.

Os *arrays* podem ser definidos passando desde logo na sua inicialização os valores associados, como em:

```
1 | int[] temperaturas={12,26,-2,15,32,19}; //  
    temperaturas.length = 6  
2 | double[] notas = { 17.0, 12.4, 7.9, 19.1, 13.4, 7.5, 15.3,  
    16.1 }; // notas.length = 8
```

Até agora vimos apenas *arrays* com uma dimensão, mas tal como em C (e outras linguagens) podemos ter *arrays* multi-dimensionais.

O número de dimensões de um *array* é visível na sua definição, na medida em que cada [] corresponde a uma dimensão.

```
1 | int[][] matriz_valores = new int[20][50]; // matriz de 20  
    linhas por 50 colunas  
2 | double[][] notasCurso = new double[5][12]; // 5 anos x 12  
    notas de UC  
3 | double [][][] temps = new double[15][12][31]; // cidades x  
    meses x dias e temperaturas
```

2 Sintaxe essencial

2.1 Declarações, inicialização 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[] memoriaVideo = new byte[1920*1080];  
7 |  
8 | short matriz[][] = new short[10][50];  
9 | short matrix[][] = new short[10][]; // A segunda dimensão é  
    variável,  
10 | // mas tem de ser alocada antes de inserir valores  
11 | matrix[0] = new short[15]; matrix[1] = new short[40];  
12 |  
13 | String[] nomes = new String[20];  
14 | String[] jogadores = { "Deco", "Hulk", "Falcao" };  
15 | String[][] texto = {{"0", "trabalho", "de", "POO"}, {"foi",  
    "disponibilizado", "hoje"}, {"Os", "Professores"}}};  
16 | String[][] galo = { {"0", "0", "X"},  
17 |                     {"X", "X", "0"},  
18 |                     {"0", "X", "0"} };  
19 | Aluno[] alunos = new Aluno[165];  
20 | Object obj[] = new Object[25];
```

2.2 Comprimento e acesso aos elementos

```
1 // determinar o comprimento
2 int numeroCraques = jogadores.length;
3 int numAlunos = alunos.length;
4
5 // acesso a posições existentes do array
6 int val = lista[3];
7 int num = lista[val*2];
8 short snum = matrix[5][3];
9 String nome = nomes[index];
10 String pal = texto[1][c];
11 System.out.println(lista[i]);
12 System.out.println(nomes[i]);
13 System.out.printf("Val = %d\n", lista[i]);
```

2.3 Percorrer um array

```
1 for(int i = 0; i < a.length; i++) { ...a[i]....} //
   acedendo a cada posição dado o i
2 // a condição de paragem poderia ser também i <= a.length-1
3 for(IdTipo elem : IdArray) { ...elem ... } // percorrer todo
   o array (do princípio ao fim)
4
5 // Imprimir todos os elementos de um array
6 for(int i=0; i< lista.length; i++)
   System.out.println(lista[i]);
7 for(int m : medias) System.out.println(m);
8
9 // Exemplos de somatórios
10 int soma = 0;
11 for(int i=0; i< lista.length; i++) soma = soma + lista[i];
12
13 int soma1 = 0;
14 for(int elem : lista) soma1 += elem;
15
16 // Exemplos de concatenação de strings.
17 // Criar uma String com o nome de todos os alunos
18 String total = "";
19 for(int i=0; i < alunos.length; i++) { total = total +
   alunos[i]; }
20
21 String total = "";
22 for(String nome : alunos) { total += nome; }
23
24 // Contagem de pares e ímpares num array de inteiros
25 int par = 0, impar = 0;
26 for(int i = 0; i < a.lenght; i++)
27     if (a[i]%2 == 0) par++;
```

```
28     else impar++;
29 out.printf("Pares = %d - Impares = %d\n", par, impar);
30
31 // Determinar o número de inteiros > __valorMaximo__ de um
    array de arrays de inteiros
32 int maiores = 0;
33 int valorMaximo = ...
34 for(int i = 0; i < numeros.length; i++) {
35     for(int c = 0; c < numeros[1].length; c++)
36         if (numeros[1][c] > valorMaximo) maiores++;
37 }
38
39 // Concatenação de strings de um array bidimensional
40 String[][] condutores = { {"Senna", "Prost"}, {"Alesi",
    "Berger"}, ..... };
41 String todosOsNomes = "";
42 for(int i = 0; i < condutores.length; i++) {
43     for(int c = 0; c < condutores[1].length; c++)
44         todosOsNomes += condutores[1][c];
45 }
46 // o mesmo algoritmo com o ciclo for()
47 todosOsNomes = "";
48 for(String[] nomes : condutores)
49     for(String nome : nomes) todosOsNomes += nome;
```

2.4 Máximo e mínimo de arrays de inteiros

Cálculo de mínimo de um array - com recurso ao Integer.MAX_VALUE

```
1 int min = Integer.MAX_VALUE; // o primeiro mínimo é o maior
    valor que é possível representar
2 // a próxima comparação garantidamente dá um novo valor de
    mínimo
3
4 int pos = -1; // índice do mínimo. -1 caso o array seja
    vazio
5
6 for(int i=0; i < a.length; i++) {
7     if (a[i] < min) {
8         min = a[i];
9         pos = i;
10    }
11 }
12
13 if (pos == -1)
14     System.out.println("O array está vazio.");
15 else {
16     System.out.println("Mínimo = " + min + " na posição " +
        pos);
```

```
17 | }
```

Cálculo de mínimo de um array - sem recurso ao Integer.MAX_VALUE

```
1 | min = a[0];    // o primeiro mínimo é o primeiro elemento do
   | array
2 | pos = 0;      // a posição do primeiro mínimo
3 | for(int i=1; i < a.length; i++) {
4 |     if (a[i] < min) {
5 |         min = a[i];
6 |         pos = i;
7 |     }
8 | }
9 |
10 | System.out.println("Mínimo = " + min + " na posição " + pos);
```

2.5 Leitura de Valores para um array

Ler um número n , dado pelo utilizador, de valores de dado tipo, e guardá-los sequencialmente num array:

```
1 | Scanner sc = new Scanner(System.in);
2 | int valor = 0;
3 | System.out.print("Número de inteiros a ler?: ");
4 | int n = sc.nextInt();
5 |
6 | int[] valores = new int[n]
7 |
8 | for(int i = 0; i < n; i++) {
9 |     valor = sc.nextInt();
10 |    valores[i] = valor;
11 | }
12 |
13 |
14 | // o mesmo algoritmo mas sem necessidade de testar a
   | variável "n", porque utilizamos o length
15 |
16 | int n = sc.nextInt();
17 | int valor = 0;
18 | for(int i = 0; i < valores.length; i++) valores[i] =
   | sc.nextInt();
```

Ler valores para um array até ser lido um valor definido como condição de paragem. A leitura faz-se até se ler esse valor ou quando se esgota o tamanho do array. Não faz sentido continuar a tentar inserir mais elementos no array porque a máquina virtual originará uma excepção.

```
1 | Scanner sc = new Scanner(System.in);
2 |
```

```
3 | int fim = sc.nextInt() ; // quando se ler este valor
   |     interrompe-se a leitura de valores
4 | int tamanhoArray = sc.nextInt(); //tamanho do array
5 | int[] valores = new int[tamanhoArray];
6 | boolean flag = false;
7 | int i = 0;
8 | int valor;
9 | while(!flag && i <= tamanhoArray-1) {
10 |     valor = sc.nextInt();
11 |     if(valor == fim)
12 |         flag = true;
13 |     else {
14 |         valores[i] = valor;
15 |         i++;
16 |     }
17 | }
```

2.6 Algoritmo de Procura

Procurar um valor num array de inteiros e devolver a posição em que o encontrou. Caso não encontre o valor devolve -1. Assume-se que os valores foram lidos com o código apresentado atrás.

```
1 |
2 | int valor;
3 | boolean encontrada = false; //flag que determina se
   |     encontrou ou não o valor. Inicializa-se a false.
4 | int i = 0;
5 | int posicao = -1;
6 | Scanner sc = new Scanner(System.in);
7 | System.out.print("Qual o valor a procurar no array? : ");
8 | valor = sc.nextInt();
9 | while(!encontrada && i < tamanhoArray) {
10 |     if(valores[i] == valor) {
11 |         encontrada = true;
12 |         posicao = i;
13 |     }
14 |     i++;
15 | }
16 | System.out.println("Valor: " + valor + " encontrado na
   |     posição "+ posicao);
```

2.7 Cópia Entre Arrays

A classe `java.lang.System` tem um método, `arraycopy`, que pode ser utilizado de forma muito fácil para efectuar cópias de *arrays*. A invocação desse método

permite não termos de construir manualmente o ciclo para copiar n posições de um *array* origem para um *array* destino. Este método tem um comportamento similar ao *memcpy* do C.

```
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(valoresIniciais, 5, valoresFinais, 1, 4);
   // 4 elementos a partir de valoresIniciais[5] para
   valoresFinais desde a sua posição 1
```

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

Apesar de os *arrays* não serem objectos, existe uma classe *Arrays* (no package *java.util*) que providencia alguns métodos interessantes para lidar com arrays.

```
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

A metodologia de resolução destes exercícios é a mesma que foi apresentada na Ficha 1. Dever-se-á ter uma classe com o método *main*, em que se faz todo o *input/output* e uma outra classe em que se guarda o array que se pretende manusear. Esta segunda classe deve ter um método para cada alínea que é solicitada nas perguntas abaixo.

1. Criar um programa que permita efectuar as seguintes operações:
 - (a) ler inteiros para um *array* e depois determinar o valor mínimo desse *array*.
 - (b) ler um array de inteiros e dois índices e determinar o *array* com os valores entre esses índices.

- (c) ler dois *arrays* de inteiros e determinar o array com os elementos comuns aos dois arrays.
2. Considerando que temos uma pauta de 5 alunos e que todos os alunos tem notas a 5 unidades curriculares, define-se o *array* `int [5] [5] notasTurma (Alunos X UnidadesCurriculares)`. Crie um programa que permita:
- (a) ler as notas dos alunos e actualiza o array da pauta;
 - (b) calcular a soma das notas a uma determinada unidade curricular;
 - (c) calcular a média das notas de um aluno (fornecendo o índice da sua posição no *array*);
 - (d) calcular a média das notas de uma unidade curricular, dado o índice da unidade curricular;
 - (e) calcular a nota mais alta a todas as unidades curriculares de todos os alunos;
 - (f) idem para a nota mais baixa;
 - (g) devolver o *array* com as notas acima de um determinado valor;
 - (h) calcular uma *String* com as notas de todos os alunos do curso a todas as unidades curriculares;
 - (i) determinar o índice da unidade curricular com a média mais elevada.
3. Crie um programa que mantenha um *array* de objectos **LocalDate** (com representação de datas, cf. Ficha1). Escreva os seguintes métodos:
- (a) inserir uma nova data, `public void insereData(LocalDate data)`
 - (b) dada uma data, determinar a data do *array* que está mais próxima (em termos de proximidade de calendário), `public LocalDate dataMaisProxima(LocalDate data)`
 - (c) devolver uma *String* com todas as datas do array, `public String toString()`
4. Crie um programa que para um array de inteiros, disponibilize os seguinte métodos:
- (a) método que ordene um array de inteiros por ordem crescente;
 - (b) método que implemente a procura binária de um elemento num array de inteiros;
5. Crie um programa que leia *Strings* para um *array*. De seguida, implemente os seguintes métodos:

- (a) determinar o array com as Strings existentes (sem repetições)
 - (b) determinar a maior String inserida;
 - (c) determinar um *array* com as Strings que aparecem mais de uma vez;
 - (d) determinar quantas vezes uma determinada String ocorre no *array*.
6. Considere que se representam matrizes de inteiros como *arrays* bidimensionais. Efectue as seguintes operações:
- (a) crie um método para ler uma matriz;
 - (b) crie um método que implemente a soma de matrizes e devolva a matriz resultado;
 - (c) crie um método que determine se duas matrizes são iguais;
 - (d) crie um método que determine a matriz oposta de uma matriz (nota: chama-se matriz oposta de A a matriz $-A$, cuja soma com A resulta na matriz nula).
7. Crie um programa que permita simular o Euromilhões. O programa deverá gerar aleatoriamente uma chave contendo 5 números (de 1 a 50) e duas estrelas (1 a 9). Para tal, pode utilizar o método `Random` da classe `java.lang.Math`.
- Posteriormente deverá ser pedido ao utilizador que introduza 5 números e duas estrelas. O programa deve comparar a aposta com a chave gerada e apresentar os resultados de números e estrelas coincidentes. Caso o utilizador tenha acertado em toda a chave, deverá ser impressa no écran 50 vezes a chave, sendo que em cada iteração a chave deve começar a ser impressa duas colunas mais à direita.