Arrays

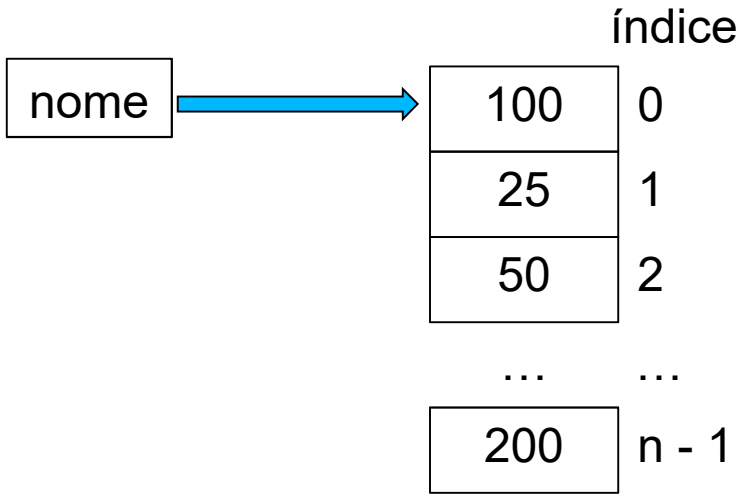
- Introdução aos *arrays*
- Declaração de variáveis do tipo *array*
- Acesso aos valores de um *array*
- Arrays como argumentos de funções
- Arrays bidimensionais
- Exemplos

Introdução

- Vimos anteriormente que é possível criar novos tipos de dados referência que permitem declarar variáveis onde é possível guardar mais do que um valor.
- No entanto, existem aplicações informáticas que precisam de lidar com grandes volumes de dados, pelo que não é eficiente ter uma variável para cada "valor" a armazenar.
- A linguagem JAVA disponibiliza outro tipo de dados referência, os *arrays,* (podemos descrever em português como sequências, vetores ou tabelas) onde é possível numa só variável armazenar vários valores do mesmo tipo.
- Como variável do tipo referência, o nome da variável é apenas uma referência para uma zona de memória que será reservada posteriormente com o operador `new`.

Arrays (sequências)

- Um *array* é uma organização de memória que se caracteriza pelo facto de ser um agregado de células contíguas, capaz de armazenar um conjunto de valores do mesmo tipo e aos quais se pode aceder de forma indexada.



- Um array é identificado pelo nome da variável e o acesso a cada elemento é feito através da respetiva posição.

Declaração de arrays

- A declaração de uma *array* faz-se da seguinte forma:

```
tipo identificador[]; // qualquer tipo...
tipo[] identificador; // ou
identificador = new tipo[dimensão];
```

- Exemplos:

```
double x[];
x = new double[3]; // array com 3 elementos reais
```

  - o *array* `x` tem os seguintes elementos: `x[0]`, `x[1]`, `x[2]`

```
int[] y = new int[4]; // array com 4 elementos inteiros
```

  - `y` tem os seguintes elementos: `y[0]`, `y[1]`, `y[2]`, `y[3]`

```
char[] z = new char[2]; // array com dois caracteres
```

  - `z` tem os seguintes elementos: `z[0]`, `z[1]`

Acesso aos elementos de um array (1)

- O tipo *array* é concebido como um conjunto de tipos base, sendo apenas possível processar um elemento de cada vez.
- Um elemento do *array* é acedido da forma:

```
identificador[índice]
```

- Em JAVA, os índices são sempre valores numéricos inteiros positivos sendo o primeiro elemento o zero e o último (dimensão-1).
- O índice pode também ser dado através de uma expressão cujo resultado tem que ser inteiro.
- Caso se tente referenciar um elemento fora do array (índice inferior a zero ou superior a (dimensão-1) gera um erro na execução do programa ("acesso fora dos limites").

Acesso aos elementos de um array (2)

- Uma variável do tipo *array* distingue-se de uma variável simples devido ao uso do operador "`[]`" na sua declaração.
- A linguagem JAVA associa a cada *array* um campo de dimensão (`length`) que pode ser usado sempre que seja necessário determinar a sua capacidade de armazenamento. Pode ser usado da forma: `identificador.length`
- É também possível declarar e atribuir um conjunto de valores a um *array* através de uma expressão de inicialização da seguinte forma:

```
int diasDoMes[] = {31, 28, 31, 30, 31, 30, 31, ..., 31};
char letras[] = {'a', 'e', 'i', 'o', 'u'};
// A dimensão é dada pelo número de elementos dentro de {}
```

Acesso ao conteúdo de *arrays*

```
final int DIM = 10;
int a[];
a = new int[DIM];
// leitura:
for(int i = 0; i < DIM; i++){
  System.out.print("Valor para posicao " + i);
  a[i] = sc.nextInt();
}
// escrita:
for(int i = 0; i < DIM; i++){
  printf("a[%d] contém o valor %d\n", i, a[i]);
}
```

Passagem de *arrays* a funções

- Uma variável do tipo *array* é sempre passada por referência a uma função. De facto estamos a passar o endereço do inicio do *array* em memória (como nos registos).
- Deste modo, uma variável do tipo *array* é sempre um argumento de entrada-saída, podendo o seu conteúdo ser modificado dentro da função.
- Dentro da função podemos saber com quantos elementos foi criado um *array* através do campo `length`.
- No entanto, nem sempre uma sequência está preenchida, pelos que nessas circunstâncias é usual utilizar uma variável inteira adicional, para além do *array*, onde armazenamos o número de elementos preenchidos.

Exemplo (1)

```
// Leitura de valores até aparecer o zero
public static int lerSequencia(int a[]){
  int n = 0, tmp;
  do{
    System.out.print("Valor inteiro: ");
    tmp = sc.nextInt();
    if(tmp != 0){
      a[n] = tmp; // armazenamos o valor na posição n
      n++;   // "avançamos" para a próxima posição
    }
  }while(tmp != 0 && n < a.length); // Atenção!
  return n; // devolvemos o número de valores lidos
}
```

Exemplo (2)

```
public static void imprimeSequencia(int a[], int n){
   for(int i = 0 ; i < n ; i++){
     System.out.printf("a[%d] contém o valor %d\n", i, a[i]);
   }
 }
public static double calcMedia(int a[], int n){
   int soma = 0;
   double m;
   for(int i = 0 ; i < n ; i++){
     soma += a[i];
   }
   m = (double)soma / n;
   return m;
 }
```

Arrays como valor de retorno de uma função

- O valor de retorno de uma função pode ser de qualquer tipo de dados (primitivo ou referência).
- Supondo que queríamos copiar o conteúdo de um *array* para outro, podíamos implementar a seguinte função:

```
public static int[] copiaArrays(int[] a, int n){
    int tmp[] = new int[n];
    for(int i = 0 ; i < n ; i++){
      tmp[i] = a[i];
    }
    return tmp;
  }
```

- A outra alternativa seria ter uma função que recebia como argumento dois *arrays* já criados (experimentem!).

Ciclo *For-each* (iterador para coleções de objetos)

for(type itr-var : collection) statement-block

Todos os elementos de uma coleção de objetos são percorridos do inicial ao final.

```
.....
Random rand = new Random(47);
float f[] = new float[10];
for (int i = 0; i < 10; i++) {
    f[i] = rand.nextFloat();
}
for (float x : f) {
    System.out.println(x);
}
```

Ciclo *For-each* (iterador para coleções de objetos) (2)

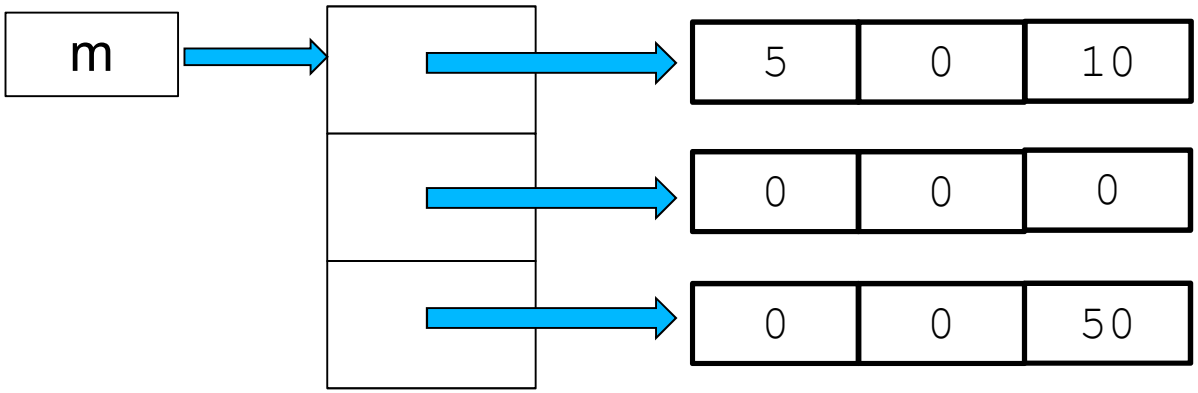
```
char[] t;   // = new char[100];
for(char c : "Aveiro tem moliceiros".toCharArray() )
    System.out.print(c + " ");
System.out.printf("\n");
// converte frase para um array de carateres
t = "Aveiro tem moliceiros".toCharArray();
for(char c : t )
     System.out.print(c + ".");
System.out.printf("\n%d\n",t.length);
```

Arrays bidimensionais

- Existem problemas em que a informação a ser processada é melhor representada através de uma estrutura de dados com um fomato bidimensional, como por exemplo uma matriz.
- Uma sequência bidimensional é na prática uma sequência de sequências.
- A sua declaração respeita as regras de uma sequência unidimensional, sendo a única diferença o facto de usar dois operadores sequência seguidos `[][]`.
- Pode ser vista como uma estutura matricial de elementos, composta por linhas e colunas, sendo o acesso a cada um dos elementos feito através de dois índices.

Exemplo (1)

```
// Declaração de uma matriz de valores inteiros com 3
   linhas e 3 colunas
int m[][] = new int[3][3];
m[0][0] = 5; // elemento na linha 0, coluna 0
m[0][2] = 10; // elemento na linha 0, coluna 2
m[2][2] = 50; // elemento na linha 2, coluna 2
m.length // número de linhas
m[0].length // número de colunas para a linha 0
```



Exemplo (2)

```
public static void lerMatriz(int m[][]){
   for(int l = 0 ; l < m.length ; l++){
     for(int c = 0 ; c < m[l].length ; c++){
        System.out.print("pos [" + l + "][" + c + "]: ");
        m[l][c] = sc.nextInt();
     }
   }
 }
 public static void imprimirMatriz(int m[][]){
    for(int l = 0 ; l < m.length ; l++){
     for(int c = 0 ; c < m[l].length ; c++){
       System.out.printf("  %5d", m[l][c]);
     }
     System.out.println();
   }
```