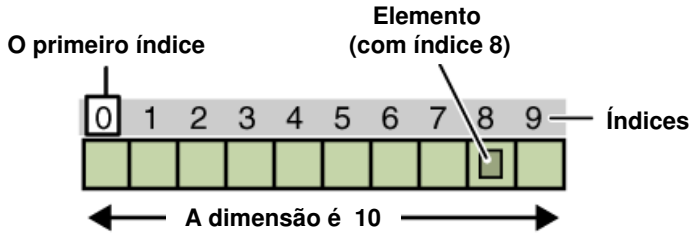
Aula 7-8

Colecções

Tabela (array)

Uma tabela é um objecto que contêm uma lista de elementos, todos do mesmo tipo.

- A **dimensão** da tabela é definida durante o processo de criação e não pode ser alterada durante a execução do programa
- Cada elemento da tabela pode ser acedido pelo seu **índice**



Declaração e criação

- A declaração e criação de uma tabela podem ser fundidas numa única instrução:

```
tipo [ ] nome_da_tabela = new tipo [dimensão] ;
```

Exemplos:

```
// cria uma tabela capaz de armazenar 50 elementos inteiros
int [ ] tabInt = new int [50];
// cria uma tabela capaz de armazenar 20 elementos doubles
double [ ] tabDouble = new double[20];
// cria uma tabela capaz de armazenar 15 elementos boolean
boolean [ ] tabBoolean = new boolean [15];
```


Dimensão definida pelo utilizador

Podemos primeiro declarar e inicializar uma variável de tipo **int** para definir a **dimensão de uma tabela**, e depois usar esta por forma a adaptar a dimensão às necessidades de cada utilizador.

Exemplo:

```
// 1º. pede e lê o número de elementos
int num = readInt("Introduza o número de elementos: ");
// 2º. declara e cria a tabela
int [ ] tabInt = new int [num];
```

A dimensão da tabela é variável e pode ser indicada pelo próprio utilizador

A variável **length**

Para além dos elementos da tabela, um objecto deste tipo possui uma variável denominada **length** que guarda o número de elementos com que a tabela foi criada

Exemplo:

```
// cria uma tabela capaz de armazenar 50 elementos reais
double [ ] notas = new double [50];
// imprime no ecrã o número máximo de elementos da tabela notas
print("Número máximo de notas é " + notas.length );
```


Acesso aos elementos de uma tabela

O aceso a cada um dos elementos da tabela é feito através de um **índice**.

O primeiro elemento tem o índice **zero**, enquanto que o último tem o índice igual à **dimensão -1**

Exemplo:

```
// cria uma tabela capaz de armazenar 50 elementos doubles
double [ ] notas = new double [50];
// coloca o valor 14,5 no primeiro elemento da tabela e 12,0 no segundo
notas[0] = 14.5;
notas[1] = 12.0;
// imprime o valor da primeira nota
print("A primeira nota é " + notas[0] );
```

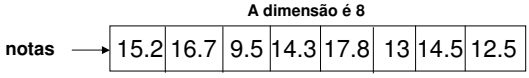
Forma abreviada de inicialização

A declaração, criação e inicialização de uma tabela podem ser fundidas numa única instrução:

```
tipo_elementos[ ] nome_da_tabela = { valor_1, valor_2, …, valor_n} ;
```

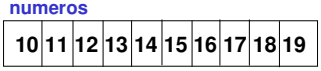
Exemplo:

```
// cria uma tabela que armazena e inicializa 8 elementos doubles
double [ ] notas = { 15.2,  16.7,  9.5, 14.3. 17.8,  13, 14.5, 12.5};
```



Exemplo 1:
utilização de ciclos para aceder aos elementos

Este ciclo inicializa o array com os seguinte valores:
numeros[0] = 10
numeros[1] = 11
numeros[2] = 12
numeros[3] = 13
…
numeros[9] = 19



Resultado que será impresso se assinalamos os espaços com um ponto:
..10..11
..12..13
..14..15
..16..17
..18..19

```
import p1.*;
class Ex_7_1_tabelaNumeros  extends P1App {
   public static void main(String args[]) {

// declara e reserva memoria para 10 elementos
    int[ ] numeros = new int[10];

//  inicializa os elementos começando pelo número 10
   for (int i=0; i<numeros.length; i++){
      numeros[i] = i + 10;
   }
//  imprime os elementos do array
  for (int i=0; i<numeros.length; i++){
    printf(4, numeros[i]);
    if (i%2 != 0) println();
  }
}
```

Problema 1

Escrever um programa que lê do teclado **uma sequência de números inteiros** positivos e que permita escrever a sequência e calcular um conjunto de estatísticas acerca da mesma, tais como: o **máximo**, **mínimo** e **média** da sequência.

- O programa deve escrever um menu de operações
- As diferentes operações deverão ser implementadas usando subprogramas
- A leitura da sequência termina quando aparecer o número zero como indicador de paragem ou quando tiverem sido lidos 50 números.

Problema 1: Implementação do Menu de Operações

```
public static void main(String args[]) {
  int seq [] = new int [50];
  int n=0, op, max, min;
  double media;
  do {  // repetir enquanto opção <> 6
   println("Menu");
   println(" 1 – Ler a sequência");
   println(" 2 – Escrever a sequência");
   println(" 3 -  Calcular o máximo");
   println(" 4 -  Calcular o mínimo");
   println(" 5 -  Calcular a média");
   println(" 6 -  Terminar o programa");
   op = readInt("Seleccione uma opção ", 1, 6);
   switch (op) {
    case 1:
      n = lerSequencia(seq);
      break;
    case 2:
      if (n > 0) escreverSequencia (seq, n);
     break;
```

```
    case 3:
      if (n > 0){
        max = calcMax(seq, n);
        println("O máximo é ", max);
      } else println("Não há dados");
      break;
    case 4:
     if (n > 0){
       min = calcMin(seq, n);
       println("O mínimo é ", min);
     } else println("Não há dados");
      break;
    case 5:
      if (n > 0){
        media = calcMedia(seq, n);
        println("A média é ", media);
     } else println("Não há dados");
      break;
  } }
  while (op <>6);
  } // fecha main
```

Problema 1: Análises de Sequência

Subprogramas

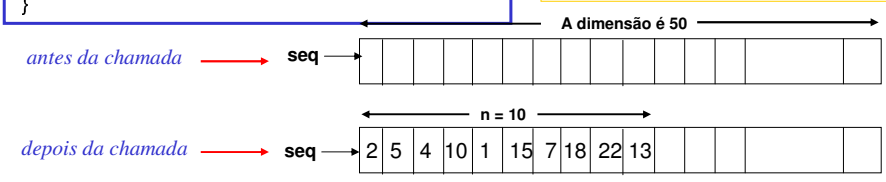
Nome da Função	Parâmetros (entrada)	Tipo do valor de retorno	Valor de retorno
lerSequencia	**int** seq[]	**int**	o número de elementos lidos
escreverSequencia	**int** seq[], **int** n	**void**	nenhum
calcMax	**int** seq[], **int** n	**int**	o máximo dos elementos lidos
calcMin	**int** seq[], **int** n	**int**	o mínimo dos elementos lidos
calcMedia	**int** seq[], **int** n	**double**	a média dos elementos lidos

Problema 1: lerSequencia

```
static int lerSequencia (int seq[]) {
 int n=0;
 do {
  do  {
    seq[n] = readInt("Insira o "+(n+1)+"º numero: ");
    }  while (seq[n]<0);
   if (seq[n]==0)
     break;
    n++;
  } while (n < 50);
 return n;
}
```

```
public static void main(String args[]) {
  int seq [] = new int [50],
  int n=0, op, max, min;
  println(" 1 – Ler a sequência");
...
  do {  // repetir enquanto opção <> 6
  op = readInt("Seleccione uma opção ", 1, 6);
  switch (op) {
    case 1:
      n = lerSequencia(seq);
      break;
  }.
}
```

O array **seq** é passado como parâmetro do subprograma. A diferencia do que acontece quando os parâmetros são de tipo primitivo (estes são passados por valor), alterar um elemento de uma tabela dentro de um subprograma, **modifica o conteúdo da tabela original**



Problema 1: Subprograma escreverSequencia

```
static void escreverSequencia (int seq[], int n) {
   printf(" Listagem da sequência com %d
        números inteiros positivos\n", n);
   for (int i=0; i<n; i++)  {
     printf(" %5d", seq[i]);
      if ((i+1)%5 == 0)
       println();
   }
   println();
}
```

Quando imprimi é substituído **%d** pelo valor de **n**

\n – muda de linha

%5d – imprime seq[i] usando 5 posições (de esquerda à direita)

Cada 5 números muda de linha

Parâmetros (entrada) → **seq** → | 2 | 5 | 4 | 10 | 1 | 15 | 7 | 18 | 22 | 13 |

n = 10

Saída no ecrã →

```
Listagem da sequência com 10 números inteiros positivos
....2....5....4...10....1
...15....7...18...22...13
```


Problema 1: Subprograma calMax

```
static int calMax (int seq[], int n) {
   int max = seq[0];
   for (int i=1; i<n; i++) {
     if (seq[i]>max)
       max = seq[i];
   }
  return max;
  }
```

Inicializar a variável local **max** com o valor do primeiro elemento de **seq**

Se o valor actual é maior do que **max**, então **max** toma este valor

```
public static void main(String args[]) {
   int seq [] = new int [50],
   int n=0, op, max, min;
   ...
   println(" 3 -  Calcular o máximo");
...
  do {  // repetir enquanto opção <> 6
   op = readInt("Seleccione uma opção ", 1, 6);
   switch (op) {
...
     case  3:
       if (n > 0){
         max = calcMax(seq, n);
         println("O máximo é ", max);
       } else
          println("Não há dados");
   break;
...
  }
}
```

Parâmetros (entrada) → **seq** → | 2 | 5 | 4 | 10 | 1 | 15 | 7 | 18 | 22 | 13 |

n = 10

Saída (tipo int) → O valor da variável local **max** (22)

Problema 1: Subprograma calMin

```
static int calMin (int seq[], int n) {
  int min = seq[0];
  for (int i=1; i<n; i++) {
    if (seq[i]<min)
      min = seq[i];
  }
 return min;
}
```

Inicializar a variável local **min** com o valor do primeiro elemento de **seq**

Se o valor actual é menor do que **min**, então **min** toma este valor

```
public static void main(String args[]) {
  int seq [] = new int [50],
  int n=0, op, max, min;
  …
  println(" 4 -  Calcular o mínimo");
…
 do { // repetir enquanto opção <> 6
  op = readInt("Seleccione uma opção ", 1, 6);
  switch (op) {
…
    case 4:
      if (n > 0){
        min = calcMin(seq, n);
       println("O mínimo é ", min);
      } else
        println("Não há dados");
  break;
…
  }
}
```

Parâmetros (entrada) → **seq** → | 2 | 5 | 4 | 10 | 1 | 15 | 7 | 18 | 22 | 13 | | | | | |

n = 10

Saída (tipo int) → O valor da variável local **min** (1)

Problema 1: Subprograma calMedia

```
static double calMedia (int seq[], int n) {
  double soma=0,  media;
  for (int i=0; i<n; i++) {
    soma = soma + seq[i];
  }
  media = soma/n;
  return media;
}
```

```
public static void main(String args[]) {
  int seq [] = new int [50],
  int n=0, op, max, min;
  double media;
…
  println(" 5 -  Calcular a media");
…
 do { // repetir enquanto opção <> 6
  op = readInt("Seleccione uma opção ", 1, 6);
  switch (op) {
…
    case 5:
      if (n > 0){
        media = calcMedia(seq, n);
       println("A média é ", media);
      } else
        println("Não há dados");
  break;
…
  }
}
```

Parâmetros (entrada) → **seq** → | 2 | 5 | 4 | 10 | 1 | 15 | 7 | 18 | 22 | 13 | | | | | |

n = 10

Saída (tipo double) → O valor da variável local **media** (9.7)

Problema 2

Escrever um programa que lê do teclado **uma sequência de números inteiros** positivos e que permita detectar se é :

- uma sequência só constituída por números pares
- uma sequência contínua de números ímpares
- uma sequência com um espaçamento constante
- uma sequência contínua de potências de 2

- O programa deve escrever um menu de operações
- As diferentes operações deverão ser implementadas usando subprogramas
- A leitura da sequência termina quando aparecer o número zero como indicador de paragem ou quando tiverem sido lidos 50 números.

Problema 2: Implementação do Menu de Operações

```
public static void main(String args[]) {
 int seq [] = new int [50], n=0;
 do {  // repetir enquanto opção <> 6
  println("Menu");
  println(" 1 – Ler a sequência");
  println(" 2 – Sequência só constituída por núm.pares?");
  println(" 3 - Sequência contínua núm. impares?");
  println(" 4 - Sequência c/ espaçamento constante?");
  println(" 5 - Sequência contínua de potências de 2?");
  println(" 6 -  Terminar o programa");
  op = readInt("Seleccione uma opção ", 1, 6);
  switch (op) {
   case 1:
     n = lerSequencia(seq);
     break;
   case 2:
     if (seqNumPares(seq, n))
       println("É só constituída por números pares");
     else
       println("Não é só constituída por números pares");
     break;
   case 3:
     if (seqNumImpares(seq, n))
       println("Sequência contínua de num. Impares");
     else
       println("Não é contínua de num. Impares");
     break;
   case 4:
     if (seqEspConstante(seq, n))
       println("Sequência c/ espaçamento constante");
     else
       println("Não tem espaçamento constante");
     break;
   case 5:
     if (seqPotencia2(seq, n))
       println("Sequência de potências de 2");
     else
       println("Não é sequência de potencias de 2" );
     break;
   } // fecha switch
 } while (op <>6);
 } // fecha main
```

Problema 2: Análises de Sequência

Subprogramas

Nome da Função	Parâmetros (entrada)	Tipo do valor de retorno	Valor de retorno
lerSequencia	**int** seq[]	**int**	o número de elementos lidos
seqNumPares	**int** seq[], **int** n	**boolean**	**true** se a sequência é só constituída por números pares; senão **false**
seqNumImpares	**int** seq[], **int** n	**boolean**	**true** se é uma sequência contínua de números ímpares; senão **false**
seqEspConstante	**int** seq[], **int** n	**boolean**	**true** se é uma sequência c/ espaçamento constante; senão **false**
seqPotencia2	**int** seq[], **int** n	**boolean**	**true** se é uma sequência de números potência de 2; senão **false**

Problema 2: Subprograma seqNumPares

Versão 1: não optimizada

```
static boolean seqNumPares (int seq[], int n){
  boolean pares = true;
  for (int i=0; i<n; i++) {
    if (seq[i] % 2 != 0)
      pares = false;
  }
  return pares;
}
```

O ciclo percorre todos os elementos lidos de **seq**. Se o elemento actual é impar, a variável **pares** toma o valor **false**

Versão 2: optimizada

```
static boolean seqNumPares (int seq[], int n){
  for (int i=0; i<n; i++) {
    if (seq[i] % 2 != 0)
      return false;
  }
  return true;
}
```

Basta encontrar um número da sequência ímpar, que o programa termina e manda para fora o valor **false**; caso contrário, o ciclo percorre todos os elementos (pares) e manda para fora **true**

Exemplo:

Parâmetros (entrada) → **seq** → | 2 | 10 | 6 | 8 | 12 | 16 | | | | | | | | | |

n = 6

Saída (tipo boolean) → O valor lógico **true**

Problema 2: Subprograma seqNumImpares

Versão 1: não optimizada

```
static boolean seqNumImpares (int seq[], int n){
  boolean impares = true;
  for (int i=0; i<n; i++) {
    if (seq[i] % 2  == 0)
      impares = false;
  }
  return impares;
}
```

O ciclo percorre todos os elementos lidos de **seq**. Se o elemento actual é par, a variável **impares** toma o valor **false**

Versão 2: optimizada

```
static boolean seqNumImpares (int seq[], int n){
  for (int i=0; i<n; i++) {
    if (seq[i] % 2 == 0)
      return false;
  }
  return true;
}
```

Basta encontrar um número da sequência par, que o programa termina e manda para fora o valor **false**; caso contrário, o ciclo percorre todos os elementos (impares) e manda para fora **true**

Exemplo:

Parâmetros (entrada) → **seq** → | 3 | 11 | 7 | 13 | 1 | 15 | | | | | | | | |

n = 6

Saída (tipo boolean) → O valor lógico **true**

Problema 2: Subprograma seqEspConstante

```
static boolean seqEspConstante(int seq[], int n){
  if (n<=2) return false;
  int delta = seq[1]-seq[0];
  for (int i=1; i<(n-1); i++){
    if (seq[i+1]-seq[i] != delta)
      return false;
  }
  return true;
}
```

Se n<=2 a execução do subprograma termina e retorna o valor lógico **false**

Se o actual espaçamento é diferente de **delta** a execução da função termina e retorna **false**

Retorna **true** se nunca foram detectadas as situações anteriores

Exemplo:

Parâmetros (entrada) → **seq** → | 5 | 10 | 15 | 20 | 25 | 30 | | | | | | | | |

n = 6

Saída (tipo boolean) → O subprograma retorna o valor lógico **true**

Problema 2: Subprograma seqPotencia2

Versão 1: não optimizada

```
static boolean seqPotencia2 (int seq[], int n){
 boolean potencias2 =true;
 for (int i=0; i<n; i++){
   if (numPotencia2(seq[i]) == false)
     potencias2 = false;
 }
 return potencias2;
}
```

Falta por definir este subprograma

Versão 2: optimizada

```
static boolean seqPotencia2 (int seq[], int n){
 for (int i=0; i<n; i++){
   if (numPotencia2(seq[i]) == false)
     return false;
 }
 return true;
}
```

Se o número não é potência de 2, terminar a execução e mandar para fora o valor **false**

Exemplo:

Parâmetros (entrada) → seq → | 2 | 16 | 32 | 8 | 64 | 4 | | | | | | | | | |

n = 6

Saída (tipo boolean) → O subprograma retorna o valor booleano **true**

Problema 2: Subprograma numPotencia2

Dado um número inteiro positivo **n** determinar se este número é potencia de 2

```
static boolean numPotencia2 (int n){
 do {
   if (n % 2 == 1)
     return false;
   n = n / 2;
 } while (n> 2);
 return true;
}
```

Se o resto da divisão por 2 é igual a 1, termina a execução da função e mandar para fora o valor **falso**, o que significa que o número **n** não é potencia de 2

Exemplo:

```
10 | 2
10   5 | 2
 0     4   2
       1
```

Se existir um resto =1 então o número não é potência de 2

Iter.	n	n % 2	n % 2 == 1 ?	actualização n =n /2	pos-validação n >2 ?
1	10	0	false	5	SIM
2	5	1	true		

FIM: retorna **false** (n=10 não é potência de 2)

Tabela Bidimensional

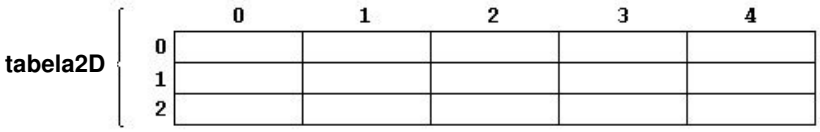
Uma tabela bidimensional é um objecto que contêm uma lista de elementos, que a sua vez são também tabelas.

A declaração e criação de uma tabela podem ser fundidas numa única instrução:

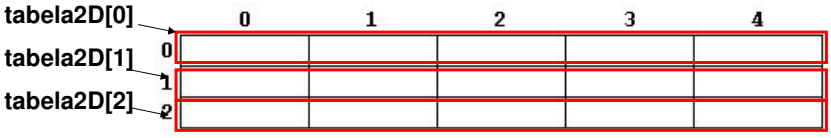
```
tipo [ ] [ ] nome_da_tabela = new tipo [numLinhas] [numColunas];
```

Exemplo:

```
// cria uma tabela de números inteiros com 3 linhas e 5 colunas
int [ ] [ ]tabela2D = new int [3] [5];
```



A variável **length**



Exemplo:

```
// cria uma tabela de números inteiros com 3 linhas e 5 colunas
int [ ] [ ]tabela2D = new int [3] [5];
// imprime no ecrã o número de linhas
println("Número de linhas: " + tabela2.length  );
// imprime no ecrã o número de colunas
println("Número de colunas: " + tabela2[0].length );
```

tabela2D[0] é uma referência para uma tabela unidimensional de dimensão 5 que representa a primeira linha

Acesso aos elementos de uma tabela bidimensional

Cada elemento de uma tabela bidimensional é referenciado utilizando 2 índices:

- ✓ o primeiro para as **linhas** (um número entre 0 e num. linhas -1)
- ✓ o segundo para as **colunas** (um número entre 0 e num. colunas -1)

Exemplo:

```
// cria uma tabela de números inteiros com 3 linhas e 5 colunas
int [ ] [ ] tabela2D = new int [3] [5];
// atribui o valor 25 ao elemento da 3ª linha e a 2ª coluna
tabela2D [2] [1] = 25;
```

tabela2D

	0	1	2	3	4
0					
1					
2		25			

Forma abreviada de inicialização

A declaração, criação e inicialização de uma tabela bidimensional com **m** linhas e **n** colunas podem ser fundidas numa única instrução:

```
tipo [] [] nome_da_tabela = { (n valores linha 1), ... (n valores linha m) } ;
```

Exemplo:

```
// cria uma tabela bidimensional de dimensão 3 x 5
int [][] tabela2D = { (1, 0, 1, 0, 1), (0, 0, 1, 1, 1), (0, 1, 0, 1, 0)};
```

tabela2D

	0	1	2	3	4
0	1	0	1	0	1
1	0	0	1	1	1
2	0	1	0	1	0

Exemplo 2:
utilização de ciclos para aceder aos elementos

Este ciclo inicializa a tabela referenciada por **tabela2D** de tal forma que cada elemento seja igual à soma do número da linha com o número da coluna em que se encontra
ex: tabela2D [1][2] = 3

Ciclo externo pelas linhas

Ciclo interno pelas colunas

Resultado que será impresso se assinalarmos os espaços com um ponto:
...0...1...2...3...4
...1...2...3...4...5
...2...3...4...5...6

```
import p1.*;
class Ex_7_2_tabela2D  extends P1App {
   public static void main(String args[]) {
// declara e reserva memoria para 15 elementos
    int [ ][ ] tabela2D = new int [3] [5];
//  inicializa os elementos da tabela bidimensional
   for (int i=0; i<tabela2D.length; i++){
     for (int j=0; j<tabela2D[0].length; j++){
       tabela2D[i][j] = i + j; }
   }
//  imprime os elementos da tabela bidimensional
   for (int i=0; i<tabela2D.length; i++){
     for (int j=0; j<tabela2D[0].length; j++) {
       printf(4, tabela2D[i][j]); }
     println();
   }
} }
```


Problema 3

Escrever um programa que lê do teclado **a dimensão de uma matriz quadrada** e logo cria e imprime o triângulo de Pascal, definido como uma matriz triangular inferior com elementos:

$$TP_{i,j} = \begin{cases} 1 \text{ se } j = 0 \text{ ou } j = i \text{ (elemento da diagonal)} \\ TP_{i-1\,j-1} + TP_{i-1\,j}, \text{ caso contrário} \end{cases}$$

	[0]	[1]	[2]	[3]	[4]
[0]	1				
[1]	1	1			
[2]	1	2	1		
[3]	1	3	3	1	
[4]	1	4	6	4	1

Sugestão: Fazer Função* tPascal *que dada a dimensão n (parâmetro) construa e retorne a matriz de Pascal (saída)

```
static int [][] tPascal(int n){
// código do subprograma
}
```

Problema 4

Escrever um programa que lê do teclado **uma sequência de números inteiros** positivos e que permita escrever a sequência ordenada em ordem crescente e decrescente.

- usar as funções **lerSequencia** e **escreverSequencia** desenvolvidas no problema 1
- usar a função predefinida do Java: **java.util.Arrays.sort()** para ordenar a sequência

Problema 4: Programa em Java

Para poder usar o subprograma predefinido **sort** devemos importar o pacote java.util.Arrays

Arrays.sort ordena crescentemente um array. Podemos usar: **Arrays.sort(seq)** se todos os elementos foram definidos. Como no nosso exemplo apenas foram lidos **n** elementos usamos: **Arrays.sort(seq, 0, n)** – para ordenar os elementos desde seq[0] até seq[n-1]

```
import p1.*;
import java.util.Arrays;
public class Pr_7_4_ordenarSeq extends P1App {
  public static void main (String [] args){
     int seq [] = new int [50];
     int n =lerSequencia(seq);
     printf("Sequência antes de ordenar:\n");
     escreverSequencia(seq, n);
     Arrays.sort(seq, 0, n);
     printf("Sequência ordenada crescentemente:\n");
     escreverSequencia(seq, n);
     printf("Sequência ordenada
decrescentemente:\n");
    // inverter a sequência e guarda-la em seqInv
    int seqInv [] = new int [n];
    for (int i=(n-1); i>=0; i--)
      seqInv [i] = seq[n-1-i];
    escreverSequencia(seqInv, n);
  } // fecho main
 static lerSequencia(int [] seq){
```

Bibliografia

- António José Mendes, Maria José Marcelino. *Fundamentos de programação em JAVA* 2.FCA – Editora de informática, 2003.
- Tutorial de Java
  http://java.sun.com/docs/books/tutorial/java/nutsandbolts/arrays.html
- Java Sorting Arrays
  http://leepoint.net/notes-java/data/arrays/70sorting.html