

# Vetores

PCII - Programação Orientada a Objetos em Java

FEG - UNESP - 2021

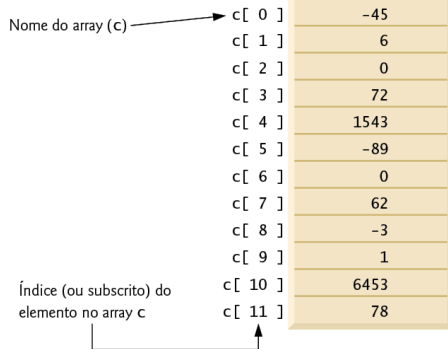
- 1 Arrays Unidimensionais
- 2 Arrays Multidimensionais
- 3 Lista de argumentos de tamanho variável
- 4 Classe Arrays
- 5 Estudo de caso: Matriz de distâncias
- 6 Coleção Genérica ArrayList<E>
- 7 Referências

## Declaração

- Um array (**vetor**) consiste em um grupo de elementos todos **de mesmo tipo** armazenados de forma **sequencial**.
- Em Java, arrays são objetos, portanto são considerados tipos referenciados.
- Os elementos de um array podem ser **tipos primitivos ou referenciados** (incluindo outros arrays).
- Para obter um certo elemento de um array, devem ser especificados o **nome da variável** que referencia o array e a **posição relativa** do elemento no array (índice).

# Arrays Unidimensionais

## Array de tamanho 12



Nome do array (c)	c[ 0 ]	-45
	c[ 1 ]	6
	c[ 2 ]	0
	c[ 3 ]	72
	c[ 4 ]	1543
	c[ 5 ]	-89
	c[ 6 ]	0
	c[ 7 ]	62
	c[ 8 ]	-3
	c[ 9 ]	1
	c[ 10 ]	6453
Índice (ou subscrito) do elemento no array c	c[ 11 ]	78

# Arrays Unidimensionais

## Declaração

- Um objeto array possui como **atributo público** seu próprio tamanho (`length`).
- O tamanho de um array não pode ser modificado (`length` é um atributo `final`).
- Assim como qualquer outro objeto, um array pode ser instanciado com a palavra chave `new`.
- Também devem ser especificados em uma instanciação o tipo e o número de posições do array. O tamanho pode ser uma variável, obtida por exemplo, a partir de uma entrada do usuário.

```
int [] b = new int[ 12 ]; // array de elementos primitivos
String [] c = new String[ 100 ]; // array de elementos referenciados
double [] values = new double[size]; //array de elementos primitivos, size é uma variável inteira
```

- Vetores podem ser criados de forma estática (o tamanho já é conhecido em tempo de compilação), nesse caso o tamanho do array é resultado do número de elementos inseridos.

```
int [] numeros = {5, 0, -2 ,4 , -1}; // array de números inteiros
```

# Arrays Unidimensionais

## Declaração

- Quando um array é criado, cada elemento recebe um valor default, de acordo com o tipo do elemento.
- É possível colocar os colchetes ( `[]` ) tanto antes quanto depois do nome da variável.

```
int b[] = new int[ 12 ]; // declaração válida  
String c[] = new String[ 100 ]; // declaração válida  
double values[] = new double[size]; // declaração válida
```

- Quando os colchetes são colocados logo após o tipo, todas as variáveis da declaração passam a ser arrays.

```
int [] a, b, c; // declarando três arrays de inteiros
```

# Arrays Unidimensionais

## Exemplo

```
// GradeBook.java
// Livro de notas contendo um array para armazenar as notas dos alunos.

public class GradeBook {
    private String courseName; // nome do curso
    private int grades[]; // array de notas dos alunos

    // construtor
    public GradeBook(String name, int gradesArray[]) {
        courseName = name;
        grades = gradesArray; // cópia superficial do vetor (não é uma boa ideia, por quê?)
    } // fim construtor

    // método para armazenar o nome do curso
    public void setCourseName(String name) {
        courseName = name;
    } // fim método setCourseName

    // método para obter o nome do curso
    public String getCourseName() {
        return courseName;
    } // fim método getCourseName

    // imprime mensagem de boas-vindas para o usuário
    public void displayMessage() {
        // getCourseName recebe o nome do curso
        System.out.printf("Bem-vindo ao livro de notas para %s!\n\n",
            getCourseName());
    } // fim método displayMessage

    /* continua na próxima página... */
}
```

# Arrays Unidimensionais

## Exemplo

```
/* ... continua da página anterior */

// procura pela menor nota
public int getMinimum() {
    int lowGrade = grades[0];

    // varre o vetor
    for (int grade : grades) {

        if (grade < lowGrade)
            lowGrade = grade;
    } // fim for

    return lowGrade; // retorna a menor nota
} // fim método getMinimum

// procura pela maior nota
public int getMaximum() {
    int highGrade = grades[0];

    // varre o vetor
    for (int grade : grades) {

        if (grade > highGrade)
            highGrade = grade;
    } // fim for

    return highGrade; // retorna a maior nota
} // fim método getMaximum

/* continua na próxima página... */
```



# Arrays Unidimensionais

## Exemplo

```
/* ... continua da página anterior */

// determina a nota média
public double getAverage() {
    int total = 0;

    // soma a nota dos estudantes
    for (int grade : grades)
        total += grade;

    // retorna a nota média da turma
    return (double) total / grades.length;
} // fim método getAverage

// realiza diversas operações sobre os dados de entrada (notas)
public void processGrades() {
    // imprime as notas
    outputGrades();

    // chama método getAverage para determinar a nota média
    System.out.printf("\nA nota média é %.2f\n", getAverage());

    // chama método getMinimum e getMaximum
    System.out.printf("A menor nota é %d\nA maior nota é %d\n\n",
        getMinimum(), getMaximum());

    // chama método outputBarChart para imprimir o gráfico de distribuição de notas
    outputBarChart();
} // fim método processGrades

/* continua na próxima página... */
```

# Arrays Unidimensionais

## Exemplo

```
/* ... continua da página anterior */

// imprime gráfico de barras com a distribuição de notas
public void outputBarChart() {
    System.out.println("Distribuição de notas:");

    // armazena a frequência das notas em cada intervalo de 10
    int frequency[] = new int[11];

    // para cada grade, incremente a frequência apropriada
    for (int grade : grades)
        ++frequency[grade / 10];

    // para cada grade de frequência, imprime um gráfico de barra
    for (int count = 0; count < frequency.length; count++) {
        // imprime barra de rótulos ( "00-09: ", ..., "90-99: ", "100: " )
        if (count == 10)
            System.out.printf("%5d: ", 100);
        else
            System.out.printf("%02d-%02d: ", count * 10, count * 10 + 9);

        // imprime barra de asteriscos
        for (int stars = 0; stars < frequency[count]; stars++)
            System.out.print("*");

        System.out.println();
    } // fim for
} // fim método outputBarChart

/* continua na próxima página... */
```

# Arrays Unidimensionais

## Exemplo

```
/* ... continua da página anterior */

// imprime o conteúdo da array de notas
public void outputGrades() {
    System.out.println("As notas são:\n");

    // imprime a nota de cada aluno
    for (int student = 0; student < grades.length; student++)
        System.out.printf("Aluno %2d: %3d\n", student + 1,
                           grades[student]);
} // fim método outputGrades

public static void main(String args[]) {
    // array que armazena a nota dos alunos
    int gradesArray[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };

    GradeBook myGradeBook = new GradeBook(
        "Programação de Computadores II", gradesArray);
    myGradeBook.displayMessage();
    myGradeBook.processGrades();
} // fim main
} // fim classe GradeBook
```

# Arrays Unidimensionais

## Exemplo: impressão na tela

Bem-vindo ao livro de notas para Programação de Computadores III!

As notas são:

Aluno 1: 87

Aluno 2: 68

Aluno 3: 94

Aluno 4: 100

Aluno 5: 83

Aluno 6: 78

Aluno 7: 85

Aluno 8: 91

Aluno 9: 76

Aluno 10: 87

A nota média é 84.90

A menor nota é 68

A maior nota é 100

Distribuição de notas:

00-09:

10-19:

20-29:

30-39:

40-49:

50-59:

60-69: \*

70-79: \*\*

80-89: \*\*\*\*

90-99: \*\*

100: \*

# Arrays Multidimensionais

## Declaração sem inicialização

- Um array multidimensional é um array onde **cada elemento é uma referência para outro array**.
- Trata-se de uma estrutura de dados com pelo menos duas dimensões.
- Um array bidimensional (**matriz**) pode ser utilizado para representar valores armazenados em uma tabela organizada em linhas e colunas.
- Um array bidimensional pode ser declarado da seguinte forma:

```
<tipo> <nome> = new <tipo>[<#linhas>][<#colunas>];
```

- Exemplo:

```
int [][] a = new int[3][4];
```

# Arrays Multidimensionais

## Matriz (array bi-dimensional)

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Linha 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Linha 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

Diagram illustrating the structure of a 2D array (matrix) with 3 rows and 4 columns. The array is named 'a'. The indices for the first row (Linha 0) are shown as a[ 0 ][ 0 ], a[ 0 ][ 1 ], a[ 0 ][ 2 ], and a[ 0 ][ 3 ]. The indices for the second row (Linha 1) are shown as a[ 1 ][ 0 ], a[ 1 ][ 1 ], a[ 1 ][ 2 ], and a[ 1 ][ 3 ]. The indices for the third row (Linha 2) are shown as a[ 2 ][ 0 ], a[ 2 ][ 1 ], a[ 2 ][ 2 ], and a[ 2 ][ 3 ].

Legend:

- Índice de coluna
- Índice de linha
- Nome do array

## Declaração com inicialização

- Um array bidimensional também pode ser declarado e inicializado com valores definidos pelo programador.

$$b = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
int b[2][2] = { { 1, 2 }, { 3, 4 } };
```

- Caso análogo de um array tridimensional:

$$c[0] = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad c[1] = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

```
int c[2][2][2] = { b, { { 5, 6 }, { 7, 8 } } };
```

# Arrays Multidimensionais

## Matrizes com linhas de tamanho variável

- As linhas de uma mesma matriz não precisam ter o mesmo tamanho.

```
int [][] d = { { 1, 2 }, { 3, 4, 5 } };
```

- Os elementos da matriz `d` fazem referência a arrays de tamanhos distintos.
- `d[0]` referencia um array de tamanho `2` e `d[1]` referencia um array de tamanho `3`.
- O exemplo abaixo declara uma matriz `e` sem inicialização explícita, onde a primeira linha possui `5` elementos e a segunda linha possui `3` elementos.

```
int [][] e = new int[ 2 ][ ]; // cria 2 linhas  
e[ 0 ] = new int[ 5 ]; // primeira linha possui 5 colunas  
e[ 1 ] = new int[ 3 ]; // segunda linha possui 3 colunas
```



# Arrays Multidimensionais

## Exemplo

```
// InitArray .java
// Inicializa arrays bi—dimensionais.
public class InitArray {

    public static void main(String args[]) {
        int array1 [][] = new int [2][];
        array1[0] = new int[2];
        array1[1] = new int[4];
        int array2 [][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };

        System.out.println("Valores por linha da array1 ");
        outputArrayFor(array1);

        System.out.println("\nValores por linha da array2 ");
        outputArrayEnhancedFor(array2);
    } // fim main

    // imprime os elementos de um array bi-dimensional
    public static void outputArrayFor(int array [][]) {
        // laço para iterar entre as linhas
        for (int row = 0; row < array.length; row++) {
            // laço para iterar entre as colunas
            for (int column = 0; column < array[row].length; column++)
                System.out.printf ("%d ", array[row][column]);

            System.out.println();
        } // fim for
    } // fim método outputArray

    /* continua na próxima página... */
```

# Arrays Multidimensionais

## Exemplo

```
/* ... continua da página anterior */

// imprime os elementos de um array bi-dimensional com for aprimorado
public static void outputArrayEnhancedFor(int array[][]) {
    // laço para iterar entre as linhas
    for (int [] row : array) {
        // laço para iterar entre os elementos
        for (int element : row)
            System.out.printf ("%d ", element);

        System.out.println ();
    } // fim for
} // fim método outputArray

} // fim classe InitArray
```

# Arrays Multidimensionais

## Exemplo: impressão na tela

Valores por linha da array1

0 0

0 0 0 0

Valores por linha da array2

1 2

3

4 5 6

# Lista de argumentos de tamanho variável

## Declaração

- Com uma lista de argumentos de tamanho variável é possível criar métodos que recebem um número de argumentos **definidos em tempo de execução**.
- A lista de argumentos deve conter o tipo do argumento seguido de reticências (`...`) e a identificação da lista de parâmetros.

```
public PrintStream printf(String format, Object... args)
```

- Um método pode conter **no máximo uma lista** variável de parâmetros.
- As reticências devem estar na última posição da lista de parâmetros.

# Lista de argumentos com tamanho variável

## Exemplo

```
// VarargsTest.java
// Usando lista de argumentos de tamanho variável.

public class VarargsTest {
    // determina a média aritmética
    public static double average(double... numbers) {
        double total = 0.0;

        // somando o valor total com o laço for aperfeiçoado
        for (double d : numbers)
            total += d;

        return total / numbers.length;
    } // fim método average

    public static void main(String args[]) {
        double d1 = 5.0;
        double d2 = 10.0;
        double d3 = 20.0;

        System.out.printf("d1 = %.1fnd2 = %.1fnd3 = %.1fn\n", d1, d2, d3);

        System.out.printf("Média de d1 e d2 é %.1fn", average(d1, d2));
        System.out.printf("Média de d1, d2 e d3 é %.1fn", average(d1, d2, d3));
    } // fim main
} // fim classe VarargsTest
```

# Lista de argumentos com tamanho variável

## Exemplo: impressão na tela

```
d1 = 5.0  
d2 = 10.0  
d3 = 20.0
```

```
Média de d1 e d2 é 7.5  
Média de d1, d2 e d3 é 11.7
```

## API Java

- A classe `Arrays`<sup>1</sup> encontra-se no pacote `java.util` e possui muitos métodos estáticos para manipulações típicas de arrays.
- Esses métodos foram sobrecarregados para arrays de tipos primitivos e de objetos.
- Exemplos de métodos pertencentes a essa API são **ordenação, busca binária, preenchimento, comparação e cópia**.
- Passar uma referência `null` para algum método dessa classe resulta em um **erro em tempo de execução**.

---

<sup>1</sup><https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

# Classe Arrays

## Exemplo

```
// ArrayManipulations.java
// Exemplos de uso dos métodos da classe Arrays
import java.util.Arrays;

public class ArrayManipulations {

    // imprime vetor
    public static void displayArray(double[] array, String description) {
        System.out.println(description + ": " + Arrays.toString(array));
    } // fim método displayArray

    /* continua na próxima página... */
```



# Classe Arrays

```
/* ... continua da página anterior */
public static void main(String[] args) {
    double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };

    // ordena o vetor em ordem crescente
    Arrays.sort(doubleArray);
    displayArray(doubleArray, "doubleArray");

    // preenche um vetor com 10 elementos de números 7s
    double[] filledDoubleArray = new double[10];
    Arrays.fill(filledDoubleArray, 7.0);
    displayArray(filledDoubleArray, "filledIntArray ");

    // copia conteúdo de um vetor para outro
    double[] doubleArrayCopy = Arrays.copyOf(doubleArray, doubleArray.length * 2);
    displayArray(doubleArrayCopy, "doubleArrayCopy");

    // compara doubleArray com doubleArrayCopy
    boolean b = Arrays.equals(doubleArray, doubleArrayCopy);
    System.out.printf("\ndoubleArray %s doubleArrayCopy\n", (b ? "==" : "!="));

    // compara doubleArray com filledDoubleArray
    b = Arrays.equals(doubleArray, filledDoubleArray);
    System.out.printf("doubleArray %s filledDoubleArray\n", (b ? "==" : "!="));

    // procura em doubleArray pelo valor 3.4
    int location = Arrays.binarySearch(doubleArray, 3.4);
    if (location >= 0)
        System.out.printf("Achei valor 3.4 na posição %d do vetor doubleArray\n", location);
    else
        System.out.println("O valor 3.4 não foi encontrado no vetor doubleArray");
} // fim main
} // fim classe ArrayManipulations
```

## Exemplo: impressão na tela

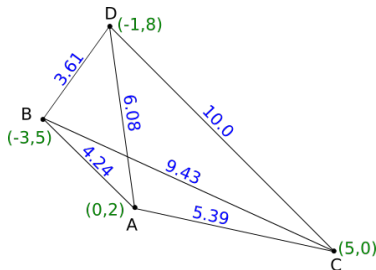
```
doubleArray: [0.2, 3.4, 7.9, 8.4, 9.3]  
filledIntArray : [7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0, 7.0]  
doubleArrayCopy: [0.2, 3.4, 7.9, 8.4, 9.3, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
doubleArray != doubleArrayCopy  
doubleArray != filledDoubleArray  
Achei valor 3.4 na posição 1 no vetor doubleArray
```

## Distância entre pontos

No estudo a seguir deseja-se construir uma classe que permita obter a distância entre diferentes localidades. A partir destas distâncias podem-se obter informações como pontos mais próximos de uma determinada localidade, pontos mais distantes, etc. Estas informações podem ser relevantes em engenharia, aplicações na área de logística, alocação de recursos, dentre outros. Observe a seguinte figura:

Representação de quatro pontos e a distância entre eles.



# Estudo de caso: Matriz de distâncias

## Classe Ponto

Representa um ponto no plano.

```
public class Ponto {  
    private int coordX, coordY;  
  
    public Ponto(Ponto pnt) { // construtor de cópia  
        this(pnt.coordX, pnt.coordY); // chamada ao construtor com dois parâmetros inteiros  
    }  
    public Ponto(int x, int y) { // construtor com dois parâmetros inteiros  
        this.coordX = x;  
        this.coordY = y;  
    }  
    public void setCoordenadaX(int x) {  
        coordX = x;  
    }  
    public void setCoordenadaY(int y) {  
        coordY = y;  
    }  
    public int getCoordenadaX() {  
        return coordX;  
    }  
    public int getCoordenadaY() {  
        return coordY;  
    }  
    public String toString() {  
        return "(" + coordX + ", " + coordY + ")";  
    }  
}
```

# Estudo de caso: Matriz de distâncias

## Classe `DistanciaEntrePontos`

A classe `DistanciaEntrePontos` é utilizada para criar uma matriz de distâncias entre pontos armazenados num arrays de objetos de tipo `Ponto`.

```
public class DistanciaEntrePontos {
    private double[][] matDistancias; //matriz de distâncias
    private Ponto[] pontos;           //array de pontos
    public DistanciaEntrePontos(Ponto[] pnts) {
        // pontos = pnts // cópia superficial evite fazer!
        // pontos = Arrays.copyOf(pnts, pnts.length) // cópia superficial !
        pontos = new Ponto[pnts.length]; //copiar pontos para o novo atributo pontos da forma tradicional
        for(int i=0; i<pnts.length; i++)
            pontos[i] = new Ponto(pnts[i]); // chamada ao construtor de cópia
        calculaMatrizDistancias(); // calcula as distâncias e preenche a matriz
    }
    // método estático que calcula a distância entre duas coordenadas no plano
    public static double distancia(double x1, double y1, double x2, double y2){
        return Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
    }
    // retorno o ponto dentro do vetor com índice ind
    public Ponto getPonto(int ind) {
        if (ind < pontos.length)
            return pontos[ind];
        return null;
    }
    // Cria uma matriz de distâncias como uma matriz quadrada simétrica, forma trivial (consome mais memória)
    public void calculaMatrizDistancias(){
        matDistancias = new double[pontos.length][pontos.length];
        for(int i=0; i<pontos.length; i++)
            for(int j=0; j<pontos.length; j++)
                matDistancias[i][j] = distancia(pontos[i].getCoordenadaX(), pontos[i].getCoordenadaY(),
                    pontos[j].getCoordenadaX(), pontos[j].getCoordenadaY());
    }
    /* ... continua na próxima página */
}
```

# Estudo de caso: Matriz de distâncias

```
/* ... continua da página anterior */
// Cria uma matriz de distâncias como uma matriz triangular inferior (economiza memória)
public void calculaMatrizTriangularDistancias() {
    matDistancias = new double[pontos.length-1][];
    for(int j=0; j<pontos.length-1; j++)
        matDistancias[j] = new double[j+1];
    for(int i=0; i<pontos.length; i++)
        for(int j=i+1; j<pontos.length; j++)
            matDistancias[j-1][i] = distancia(pontos[i].getCoordenadaX(), pontos[i].getCoordenadaY(),
                pontos[j].getCoordenadaX(), pontos[j].getCoordenadaY());
}

// retorna a distância entre os pontos com índices ind1 e ind2
public double getDistancia(int ind1, int ind2) {
    if (ind1 < matDistancias.length && ind2 < matDistancias.length)
        return matDistancias[ind1][ind2];

    System.err.println("Índice inválido!");
    return Double.NEGATIVE_INFINITY; // -Infinity (podia ser outro número negativo, ex: -1)
}

// retorna a distância entre os pontos com índices ind1 e ind2 quando a matriz é triangular inferior
public double getDistanciaTriangular(int ind1, int ind2) {
    if (ind1 < matDistancias.length+1 && ind2 < matDistancias.length+1) {
        if (ind1 == ind2)
            return 0;
        if (ind1 > ind2) {
            return matDistancias[ind1-1][ind2];
        }
        return matDistancias[ind2-1][ind1];
    }

    System.err.println("Índice inválido!");
    return Double.NEGATIVE_INFINITY; // -Infinity (podia ser outro número negativo, ex: -1)
}

// retorna o tamanho do vetor pontos
public int getNumPontos() {
    return pontos.length;
}

/* ... continua na próxima página */
```

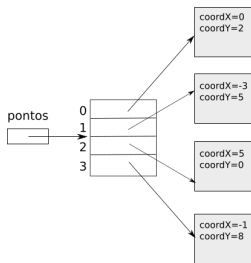
# Estudo de caso: Matriz de distâncias

```
/* ... continua da página anterior */
// Imprime uma matriz na tela com os elementos alinhados à esquerda
public void imprimirMatrizDistancias() {
    for(int i=0; i<matDistancias.length; i++) {
        for(int j=0; j<matDistancias[i].length; j++)
            System.out.printf("%-5.2f ", matDistancias[i][j]); // (-) alinhar à esquerda
        System.out.println();
    }
}

// É criado um vetor com 4 pontos, cria-se um objeto da classe e se imprime a matriz de distâncias assim como a distância
// entre os pontos, muda-se a coordenada de um ponto (3) e as distâncias são visualizadas novamente.
public static void main(String[] args) {
    Ponto[] pontos = {new Ponto(0, 2), new Ponto(-3, 5), new Ponto(5, 0), new Ponto(-1, 8)};
    DistanciaEntrePontos distancias = new DistanciaEntrePontos(pontos); // pontos está fora do objeto!
    distancias.imprimirMatrizDistancias();
    for(int i=0; i<distancias.getNumPontos(); i++) // imprime as distancias entre todos os pares de pontos
        for(int j=0; j<distancias.getNumPontos(); j++)
            System.out.printf("Distância entre: %s e %s: %.2f\n", distancias.getPonto(i), distancias.getPonto(j),
                               distancias.getDistancia(i, j));
    pontos[3].setCoordenadaX(0); // modificaria o ponto acidentalmente se o construtor fizesse uma cópia superficial!
    distancias.getPonto(3).setCoordenadaX(0); // tem que mudar chamando o método do objeto, aí não é acidentalmente!
    distancias.calculaMatrizDistancias(); // tem que recalcular!
    System.out.println("Depois da mudança!");
    for(int i=0; i<distancias.getNumPontos(); i++)
        for(int j=0; j<distancias.getNumPontos(); j++)
            System.out.printf("Distância entre: %s e %s: %.2f\n", distancias.getPonto(i), distancias.getPonto(j),
                               distancias.getDistancia(i, j));
}
} // fim da classe
```

# Estudo de caso: Matriz de distâncias

Armazenamento dos quatro pontos no vetor.



Acesso à matriz quadrada para 4 pontos (método `getDistancia`). Acesso à matriz triangular para 4 pontos (método `getDistanciaTriangular`).

	0	1	2	3
0	0.0	4.24	5.39	6.08
1	4.24	0.0	9.43	3.61
2	5.39	9.43	0.0	10.0
3	6.08	3.61	10.0	0.0

distância entre pontos 3 e 2 → `matDistancias[3,2]`

	0	1	2
0	4.24		
1	5.39	9.43	
2	6.08	3.61	10.0

distância entre pontos 3 e 2 → `matDistancias[2,2]`



# Estudo de caso: Matriz de distâncias

## Observações

- Para utilizar a matriz triangular basta substituir as chamadas aos métodos `calculaMatrizDistancias` e `getDistancia` pelos métodos `calculaMatrizTriangularDistancias` e `getDistanciaTriangular`.
- Note que se uma cópia superficial do vetor passado como argumento é feita no construtor da classe, qualquer modificação externa de dito vetor acarretará numa modificação interna dos atributos do objeto (como comentado no `main`).
- Se o vetor é copiado corretamente então para modificar os dados é preciso chamar métodos do objeto, por exemplo o método `getPonto`.
- Quando é feita alguma modificação nos atributos (pontos) é obrigatório recalcular a matriz de distâncias.

## Desafio: Complete a classe `DistanciaEntrePontos`

Escreva um:

- 1 Método que retorne o índice do ponto mais próximo do ponto com seu índice passado como argumento do método. Para o exemplo mostrado no `main` (4 pontos) com o argumento (0) deve retornar: 1.
- 2 Método que retorne o índice do ponto mais longínquo do ponto cujo índice foi passado como argumento. Exemplo: para o ponto (0) deve retornar 3.
- 3 Método que retorne um vetor com os índices dos pontos ordenados do mais próximo ao mais distante do ponto com seu índice passado como argumento. Exemplo: para o argumento (0) deve retornar o vetor [1, 2, 3].
- 4 Método que retorne um vetor com os índices dos pontos ordenados do mais distante ao mais próximo do ponto com seu índice passado como argumento. Exemplo: para o argumento (0) deve retornar o vetor [3, 2, 1].
- 5 Método que retorne um vetor com os índices dos pontos que estão no raio da distância de um ponto com o índice também passado como argumento. Exemplo: para o argumento (0, 5.0) deve retornar o vetor [1].

## API Java

- A classe `ArrayList`<sup>2</sup> (`java.util`) consiste em uma estrutura de dados alternativa para armazenar dados de modo sequencial, como um array.
- A diferença do `ArrayList` está no fato de que essa estrutura **modifica seu tamanho automaticamente** para comportar novos dados ou após a remoção de dados.
- `ArrayList<E>` é uma das **coleções genéricas** disponíveis para um aplicativo Java.
- É uma **coleção** pois foi implementada para armazenar grupos de objetos relacionados (de mesmo tipo).
- É **genérica** pois foi implementada para armazenar um tipo genérico `E` (não primitivo), definido pelo programador na declaração de uma variável e instanciação de um objeto `ArrayList<E>`.

```
ArrayList< Integer > list = new ArrayList<Integer>();
```

# Classe ArrayList

## Alguns métodos <sup>3</sup> da classe ArrayList

Método	Descrição
<code>add</code>	Sobrecarregado. Adiciona um elemento ao final do <code>ArrayList</code> ou adiciona um elemento no índice especificado.
<code>clear</code>	Remove todos os elementos do <code>ArrayList</code> .
<code>contains</code>	Retorna <code>true</code> se o <code>ArrayList</code> contém o elemento especificado; <code>false</code> caso contrário.
<code>get</code>	Retorna o elemento no índice especificado.
<code>indexOf</code>	Retorna o índice dentro do <code>ArrayList</code> da primeira ocorrência do elemento especificado.
<code>remove</code>	Sobrecarregado. Remove a primeira ocorrência do valor especificado ou o elemento armazenado no índice especificado.
<code>size</code>	Retorna o número de elementos armazenados no <code>ArrayList</code> .

<sup>3</sup><https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

# Coleção Genérica ArrayList<E>

```
// ArrayListCollection.java
// Demonstração da coleção genérica ArrayList<E>.
import java.util . ArrayList;

public class ArrayListCollection {
    public static void main(String[] args) {

        // cria uma nova ArrayList que armazena Strings. Capacidade inicial igual a 10 (default) .
        ArrayList<String> items = new ArrayList<String>();
        items.add("vermelho"); // anexa um novo elemento à lista
        items.add(0, "amarelo"); // insere um elemento na posição 0
        System.out.println("Imprime conteúdo da lista: " + items);

        items.add("verde"); // anexa um elemento à lista
        items.add("amarelo"); // anexa um elemento à lista
        System.out.println("Imprime a lista com os dois novos elementos: " + items); // igual que items.toString()

        items.remove("amarelo"); // remove a primeira ocorrência "amarelo"
        System.out.println("Remove a primeira ocorrência de amarelo: " + items); // igual que items.toString()

        items.remove(1); // remove item na posição 1 (segundo elemento da lista)
        System.out.println("Remove o segundo elemento da lista (verde): " + items); // igual que items.toString()

        // verifica se um certo elemento se encontra na lista
        System.out.printf("%" + "vermelho" + "%sestá na lista\n",
            items.contains("vermelho") ? "" : "não ");

        // imprime o número de elementos contidos na lista
        System.out.printf("Tamanho da lista: %s\n", items.size());

    } // fim main
} // fim classe ArrayListCollection
```

## Exemplo: impressão na tela

```
Imprime conteúdo da lista: [amarelo, vermelho]
Imprime a lista com os dois novos elementos: [amarelo, vermelho, verde, amarelo]
Remove a primeira ocorrência de amarelo: [vermelho, verde, amarelo]
Remove o segundo elemento da lista (verde): [vermelho, amarelo]
"vermelho" está na lista
Tamanho da lista: 2
```

# Coleção Genérica ArrayList<E>

## Outro Exemplo

```
import java.util . ArrayList ;
import java.util . Arrays ;

public class IntegerArrayListDemo {

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(5,8,-9,15,0,12,2)); // inicializa list com valores predeterminados
        System.out.print("Lista: ");
        System.out.println( list );
        list.remove(2); // remove o elemento com indice 2 de list i.e o -9
        System.out.print("Lista: ");
        System.out.println( list );
        list.remove(Integer.valueOf(2)); // remove o elemento 2 de list
        System.out.print("Lista: ");
        System.out.println( list );
    }
}
```

## Impressão na tela

```
Lista: [5, 8, -9, 15, 0, 12, 2]
Lista: [5, 8, 15, 0, 12, 2]
Lista: [5, 8, 15, 0, 12]
```

# Matriz de Distâncias com ArrayList<E>

## Desafio:

Reescreva a classe `DistanciaEntrePontos` utilizando a classe `ArrayList` em lugar de arrays.



# Referências

- 1 Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 8a. Ed.
- 2 The Java Tutorials (Oracle)  
<http://docs.oracle.com/javase/tutorial/>
- 3 Java Tutorial (w3school)  
<https://www.w3schools.com/java/>
- 4 Eckel, B. Thinking in Java. 2. ed.  
<http://mindview.net/Books>
- 5 Introduction to Computer Science using Java  
<http://chortle.ccsu.edu/java5/index.html>