11- ARRAYS E COLLECTIONS

Francisco Barretto - francisco.barretto@udf.edu.br

VARIAVEIS COMPOSTAS HOMOGENEAS

Variáveis Compostas Homogêneas

- Quando uma estrutura de dados é composta pelo mesmo tipo.
- Ex: uma alcatéia, um bando, um cardume.

- Unidimensionais: arrays
- Bidimensionais: matrizes (damas, xadrez, ...)

1. Arrays Unidimensionais

- arrays são variáveis compostas que podem armazenar um conjunto de valores.

- Todos estes valores são referenciados através do **nome do array** (o mesmo para todo o conjunto de valores) e de um **índice** (distinto para cada valor.)

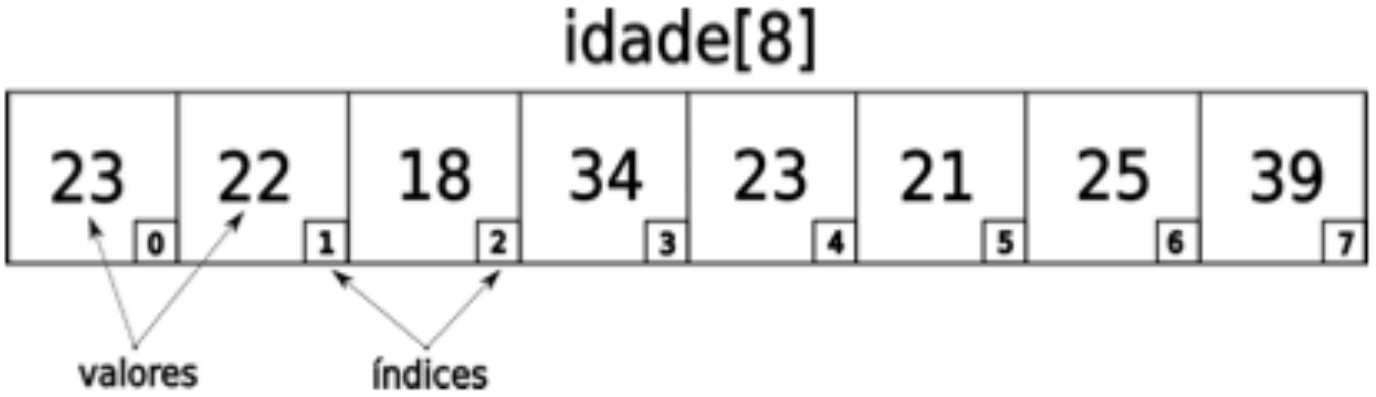
1. Arrays Unidimensionais

- As valores armazenados numa variável arrayial são todos do mesmo tipo, por isso os arrays são chamados de **variáveis compostas homogêneas**.

- Os arrays são utilizados quando se quer armazenar diversos valores de um mesmo tipo e referenciá-los com o mesmo nome.

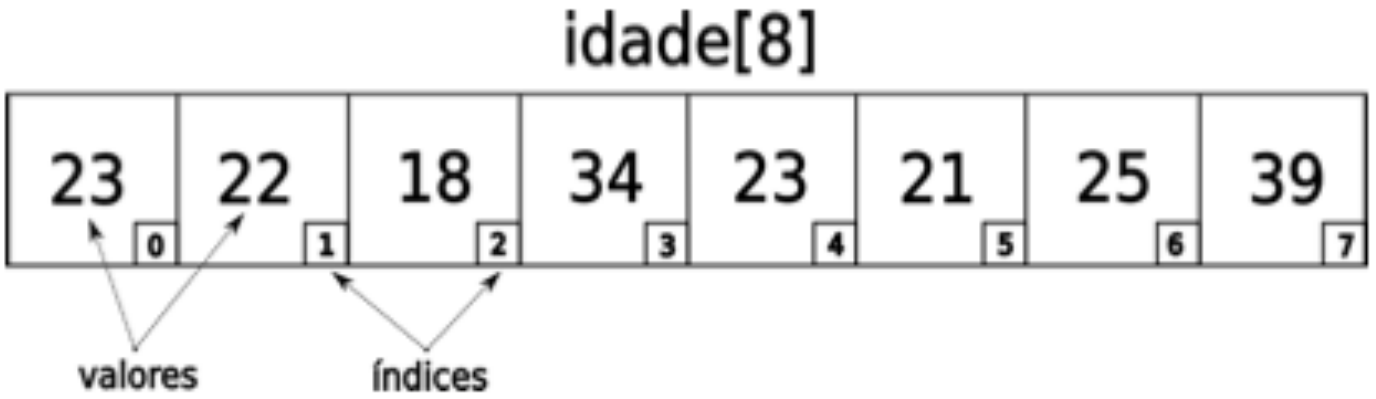
1. Arrays Unidimensionais

- Por exemplo, para armazenar as idades de vários funcionários, poderia-se criar um array com 8 posições; cada índice de 0 a 7 corresponderia a um funcionário.



1. Arrays Unidimensionais

- Neste caso será criada uma variável que conterá 8 posições – índices 0 a 7 – onde poderão ser armazenados números inteiros.



1. Arrays Unidimensionais

- Cada uma das posições do array é referenciada através do nome do array seguido do respecivo índice colocado entre colchetes.

- idade[0] = 23;
- Idade[1] = 22;

1. Arrays Unidimensionais

- É importante notar que uma variável de **N** posições possui índices de **0** a **N-1**. Na variável de 8 posições usam-se os índices 0 a 7;
  - qualquer índice fora desta faixa resulta em erro.
- A grande vantagem de se usar índices dentro do nome da variável é a possibilidade de referenciar um dado indice do array através de um índice variável.

1. Arrays Unidimensionais

- Por exemplo, para imprimir todos os valores da variável *idade*, ao invés de fazer:

  Syso(idade[0]), **syso**(idade[1]), ...

- Podemos fazer:

```
for(int i = 0; i < idade.length; i++)
        System.out.printf("%d \n", idade[ i ]);
}
```

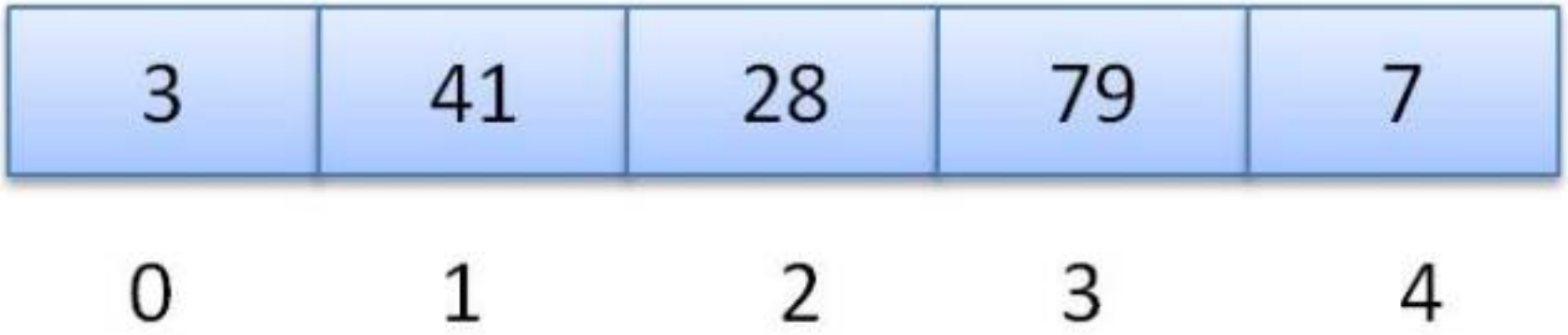
ARRAYS

Arrays

- Os arrays ou matrizes, como são conhecidos pelo Java, fazem parte do pacote java.util na coleção da API do Java. São objetos de recipientes que contém um número fixo de valores de um único tipo. O comprimento de um array é estabelecido quando criado, sendo que após a criação o seu comprimento fica fixo.

Arrays

- Cada item em um array é chamado de indice , e cada indice é acessado pelo número, o índice. Abaixo é mostrado se dá esse acesso aos seus indice s, lembrando que sempre sua numeração começa em 0.



Declarando Arrays

- Na declaração de um array, cada indice recebe um valor padrão, sendo 0 (zero) para números de tipo primitivo, falso (false) para indices booleanos e nulo (null) para referências.

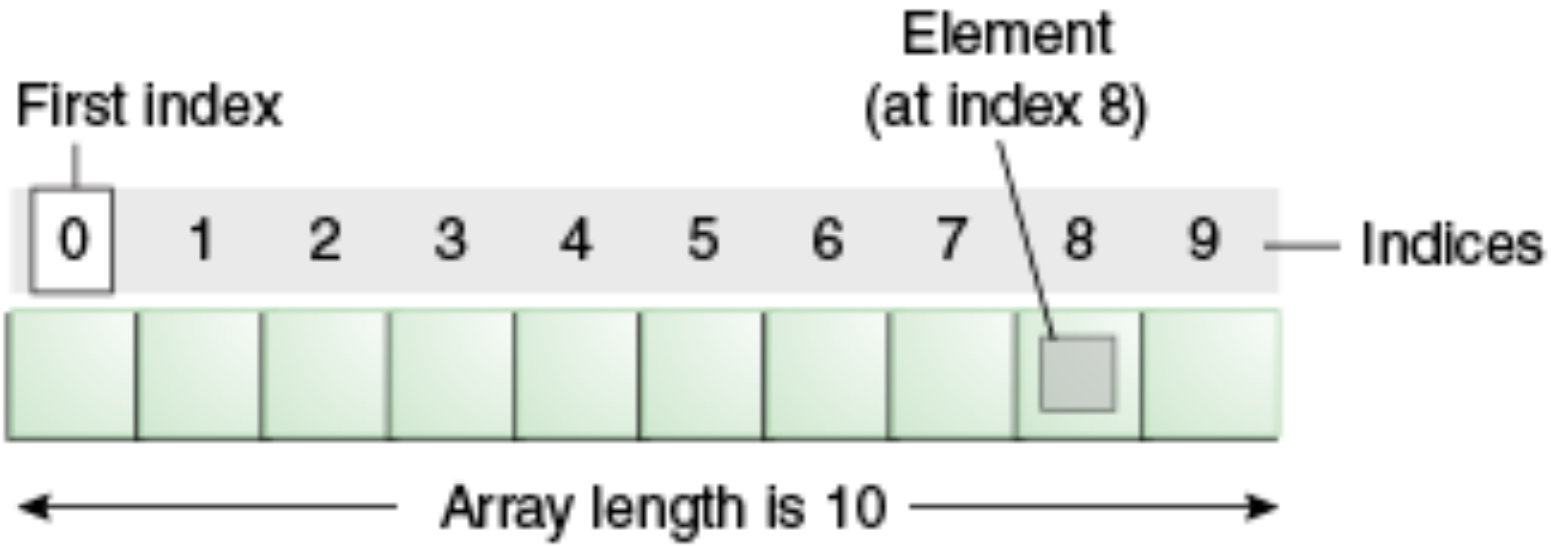
- ```
  int[] idades;
  ```
  ```

Declarando Arrays

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

Criando Arrays

- O int[] é um tipo. Uma array é sempre um objeto, portanto, a variável idades é uma referência. Vamos precisar criar um objeto para poder usar a array. Como criamos o objeto-array?
  - `idades = new int[10];`



Criando Arrays

- Alternativamente, podemos criar um array já declarando quais serão os seus elementos:

```
int[] anArray = {
        100, 200, 300,
        400, 500, 600,
        700, 800, 900, 1000
};
```

Inicializando os elementos do Array

```
// declara um array de inteiros
int[] anArray;

//aloca memoria para 10 inteiros
anArray = new int[10];

// inicializa o primeiro indice
anArray[0] = 100;

// inicializa o segundo
anArray[1] = 200;

// e assim por diante…
anArray[2] = 300;
anArray[3] = 400;
anArray[4] = 500;
anArray[5] = 600;
...
```

Acessando os elementos do Array

```
System.out.println("indice 0: ” + anArray[0]);
System.out.println("indice 1: “ + anArray[1]);
System.out.println("indice 2: “ + anArray[2]);
System.out.println("indice 3: “ + anArray[3]);
                          ...
System.out.println("indice 7: “ + anArray[7]);
System.out.println("indice 8: “ + anArray[8]);
System.out.println("indice 9: “ + anArray[9]);
```

Percorrendo os elementos do Array

```
public static void main(String args[]) {
   int[] idades = new int[10];
   for (int i = 0; i < 10; i++) {
     idades[i] = (i + 1)* 100;
   }

   for (int i = 0; i < 10; i++) {
     System.out.println(idades[i]);
   }
}
```

Tamanho do Array (length)

- Por padrão, cada array sabe seu próprio tamanho, independente de quantos valores forem inseridos. O array armazena na variável de instância o método length, que retorna o tamanho do array especificado.

```
int[] arrayUm = {43,42,4,8,55,21,2,45};
System.out.println("\nTamanho do ArrayUm =
"+arrayUm.length);
```

Percorrendo Arrays (length)

```
void imprimeArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
  }
}
```

Percorrendo Arrays (Java 5)

- No caso de você não ter necessidade de manter uma variável com o índice que indica a posição do elemento no array (que é uma grande parte dos casos), podemos usar o enhanced-for.

```
// imprimindo toda a array
for (int x : idades) {
  System.out.println(x);
}
```

Exercicio 0:

- Crie um array de inteiros com 10 posições;
- Preencha o array, considere que cada elemento deve ser igual ao seu índice elevado ao quadrado;
- Imprima todos os elementos do array;

Exercicio 1

- Leia 10 numeros inteiros do usuário e os armazene em um array;
- Implemente um método que recebe um array como parâmetro e retorne quantos numeros primos existem no array;
- Exiba para o usuário quantos numeros primos há dentre os numeros digitados.

Exercicio 2

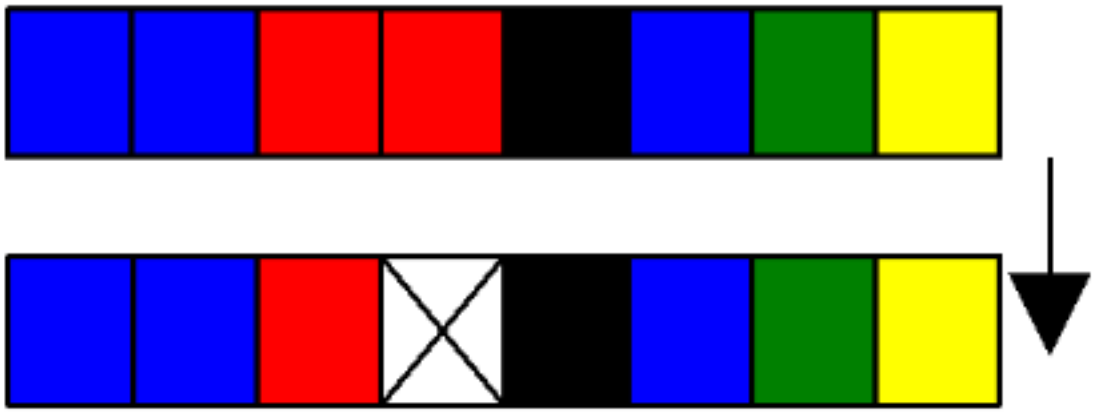
- Crie um array, cujo tamanho deve ser definido pelo usuário;
- Depois, leia do usuário os numeros necessários para preencher o array;
- Percorra o array e identifique qual foi o menor e o maior numero digitado;

Java Collections Framework

ARRAYS vs COLLECTIONS

- Como vimos, manipular arrays é bastante trabalhoso. Essa dificuldade aparece em diversos momentos:
  - não podemos redimensionar um array em Java;
  - é impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
  - não conseguimos saber quantas posições do array já foram populadas sem criar, para isso, métodos auxiliares.

ARRAYS vs COLLECTIONS



Retire a quarta Conta

conta[3] = **null**;

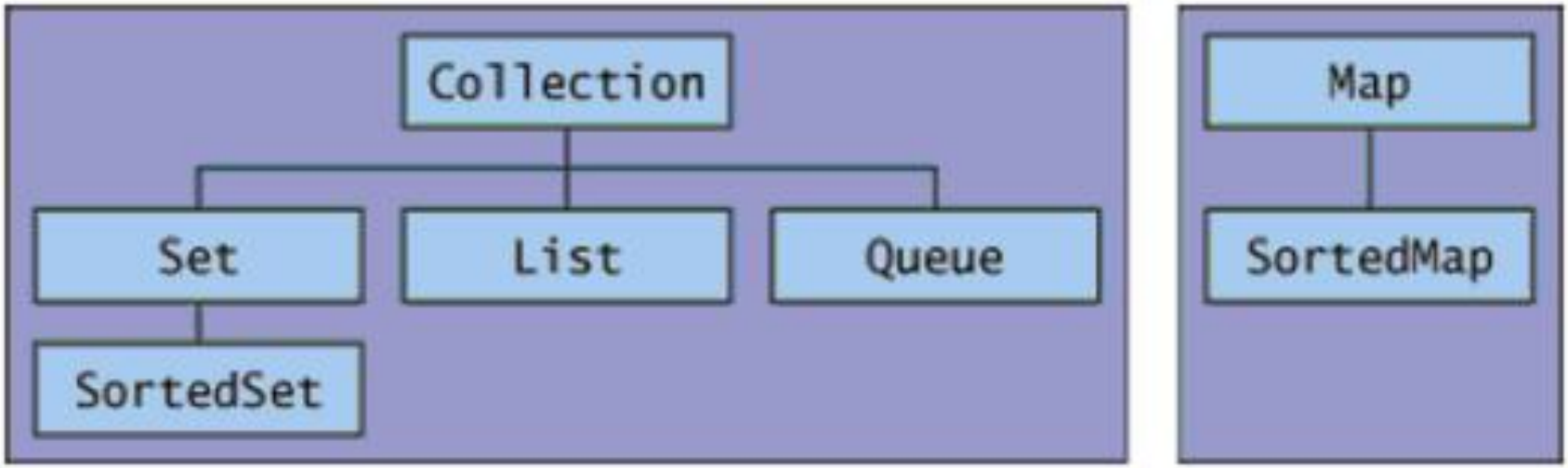
O que são COLLECTIONS?

- É um objeto onde podemos agrupar vários elementos. No dia-a-dia nos deparamos com várias situações onde as coleções estão presentes: uma ***fila*** de banco, uma ***lista*** de compras, uma ***pilha*** de livros, um ***conjunto*** de elementos, etc.

O que são COLLECTIONS?

- Em Java, existe uma arquitetura para representar e manipular coleções:
  - **Interfaces:** Permitem que as coleções sejam manipuladas independentes de suas implementações;
  - **Implementações:** Implementam uma ou mais interfaces do *framework;*
  - **Algoritmos:** São métodos que realizam operações (*sort, reverse, binarysearch, etc*) sobre as coleções;

Arquitetura Collection



Classes/Interfaces do Collection

- Collection:
  - O framework não possui implementação direta desta inferface, porém, ela está no topo da hierarquia definindo operações que são comuns a todas as coleções;
    - Set;
    - List;
    - Queue;

Classes/Interfaces do Collection

- Collection:
  - Set
    - Está diretamente relacionada com a idéia de conjuntos. Assim como um conjunto, as classes que implementam esta interface não podem conter elementos repetidos.
  - List
  - Queue

Classes/Interfaces do Collection

- Collection:
  - Set
  - List
    - Também chamada de seqüência. É uma coleção ordenada, que ao contrário da inferface Set, pode conter valores duplicados.
    - Além disso, temos controle total sobre a posição onde se encontra cada elemento de nossa coleção, podendo acessar cada um deles pelo índice.
  - Queue

Classes/Interfaces do Collection

- Collection:
  - Set
  - List
  - Queue
    - Normalmente utilizamos esta interface quando queremos uma coleção do tipo FIFO (First-In-First-Out), também conhecida como fila.

Classes/Interfaces do Map

- Map
  - Util quando se quer armazenar uma relação de chave-valor entre os elementos. Cada chave pode conter apenas um único valor associado.
  - **SortedMap** para situações onde desejarmos ordenar os elementos

Implementações Collections

Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

Listas: java.util.List

- A API de Collections traz a interface java.util.List, que especifica o que uma classe deve ser capaz de fazer para ser uma lista. Há diversas implementações disponíveis, cada uma com uma forma diferente de representar uma lista.
- A implementação mais utilizada da interface List é a ArrayList, que trabalha com um array interno para gerar uma lista. Portanto, ela é mais rápida na pesquisa do que sua concorrente, a LinkedList, que é mais rápida na inserção e remoção de itens nas pontas.

Listas: java.util.List

- Para criar um ArrayList, basta chamar o construtor:

```
ArrayList lista = new ArrayList();
```

- É sempre possível abstrair a lista a partir da interface List:

```
List lista = new ArrayList();
```

- Para criar uma lista de nomes (String), podemos fazer:

```
List lista = new ArrayList();
lista.add("Manoel");
lista.add("Joaquim");
lista.add("Maria");
```

Listas: java.util.List

- Para saber quantos elementos há na lista, usamos o método size():

```
System.out.println(lista.size());
```

- Há ainda um método get(int) que recebe como argumento o índice do elemento que se quer recuperar.
  - Através dele, podemos fazer um for para iterar na lista:

```
for (int i = 0; i < lista.size(); i++) {
  lista.get(i); // código não muito útil....
}
```

Listas: java.util.List

```
for (int i = 0; i < lista.size(); i++)
{
  lista.get(i); // código não muito
útil....
}

for (String nome: lista) {
  System.out.println(nome);
}
```

Listas: java.util.List

- Em qualquer lista, é possível colocar qualquer Object. Com isso, é possível misturar objetos:

```
List lista = new ArrayList();
lista.add("Uma string");
lista.add(2);
...
```

Listas: java.util.List

- Mas e depois, na hora de recuperar esses objetos? Como o método get devolve um Object, precisamos fazer o cast. Mas com uma lista com vários objetos de tipos diferentes, isso pode não ser tão simples...
- No Java 5.0, podemos usar o recurso de Generics para restringir as listas a um determinado tipo de objetos (e não qualquer Object):

```
List<String> lista= new ArrayList<String>();
lista.add("Joao");
lista.add("Jose");
lista.add(3); //não compila mais
```

Listas: java.util.List

- Vale ressaltar a importância do uso da interface List: quando desenvolvemos, procuramos sempre nos referir a ela, e não às implementações específicas. Por exemplo, se temos um método que vai buscar uma série de contas no banco de dados, poderíamos fazer assim:

```
class Agencia {
  public ArrayList<Conta> buscaTodasContas() {
    ArrayList<Conta> contas = new ArrayList<>();
    // para cada conta do banco de dados, contas.add
    return contas;
  }
}
```

Listas: java.util.List

- O ideal é sempre trabalhar com a interface mais genérica possível:

```
class Agencia {

  // modificação apenas no retorno:
  public List<Conta> buscaTodasContas() {
    ArrayList<Conta> contas = new ArrayList<>();
    // para cada conta do banco de dados,
contas.add
    return contas;
  }
}
```

Listas: java.util.List

- A classe Collections traz um método estático sort que recebe um List como argumento e o ordena por ordem crescente. Por exemplo:

```
List<String> lista = new ArrayList<>();
lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");

// repare que o toString de ArrayList foi sobrescrito:
System.out.println(lista);

Collections.sort(lista);

System.out.println(lista);
```

Exercicio 3

- Leia 10 numeros inteiros do usuário e os armazene em um ArrayList;
- Implemente um método que recebe um List como parâmetro e retorne quantos numeros primos existem no List;
- Exiba para o usuário quantos numeros primos há dentre os numeros digitados.

Exercicio 4

- Crie uma classe Pessoa com os atributos nome e idade
- Leia os nomes e as idades de 10 pessoas e aramazene em um ArrayList;
- Mostre o nome e a idade da pessoa mais velha e mais nova.
- Mostre também a média de idade;

Bibliografia

Complementar:

- ARNOLD, Ken; GOSLING, James; HOLMES, David. A linguagem de programação Java. 4. ed. Porto Alegre: Bookman, 2007. xiii, 799 p. ISBN 9788560031641
- RODRIGUES FILHO, R. Desenvolva aplicativos com Java 2. São Paulo: Érica, 2005
- ROMAN, Ed; AMBLER, Scott W.; JEWELL, Tyler. Dominando Enterprise Javabeans. 2. ed. Porto Alegre: Grupo A, 2014
- RUTTER, Jake. Smashing jQuery: Interatividade Avançada com JavaScript Simples. Porto Alegre: Grupo A, 2012
- SIERRA, K.; BATES, B. Use a Cabeça! Java. 2. ed. Rio de Janeiro: Alta Books, 2007.

Bibliografia

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/#16-5-ordenacao-collections-sort

http://www.devmedia.com.br/utilizando-collections-parte-i/3162

Bibliografia

Básica:

- DEITEL, H. M.; DEITEL, P. J. Java: Como Programar. 8. ed. São Paulo: Pearson Education, 2010
- MANZANO, José Augusto N. G.; COSTA JUNIOR, Roberto Afonso da. JAVA II: programação e computadores - guia básico de introdução, orientação e desenvolvimento. 1. ed. São Paulo: Érica, 2006
- SANTOS, R. Introdução à Programação Orientada a Objetos Usando Java. 1. Ed. Rio de Janeiro: Campus, 2003.

Contato

francisco.barretto@udf.edu.br