# Collections

## A API de Coleções do Java

Givanaldo Rocha de Souza
givanaldo.rocha@ifrn.edu.br
http://docente.ifrn.edu.br/givanaldorocha

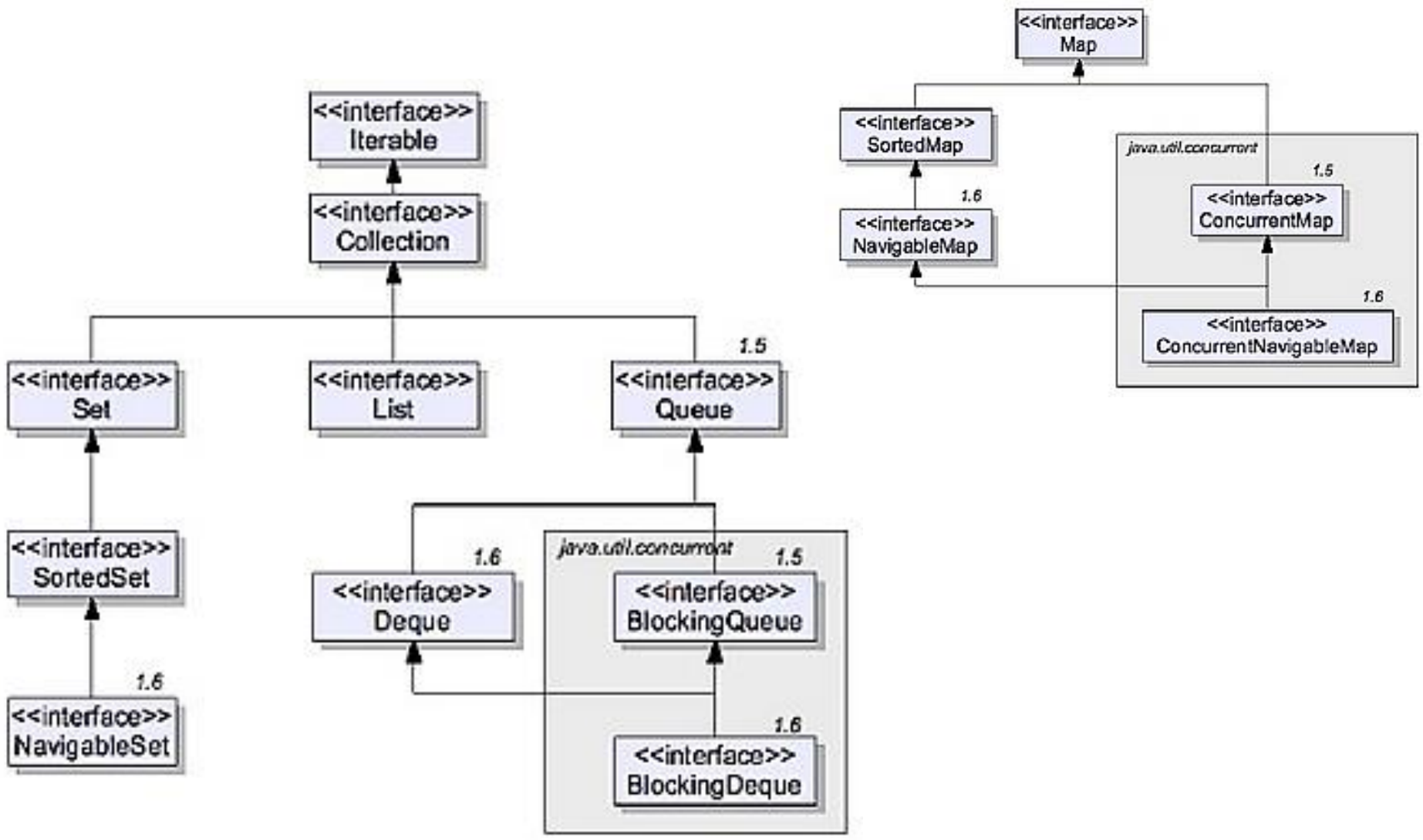
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE
Campus Natal - Zona Norte

# Coleções

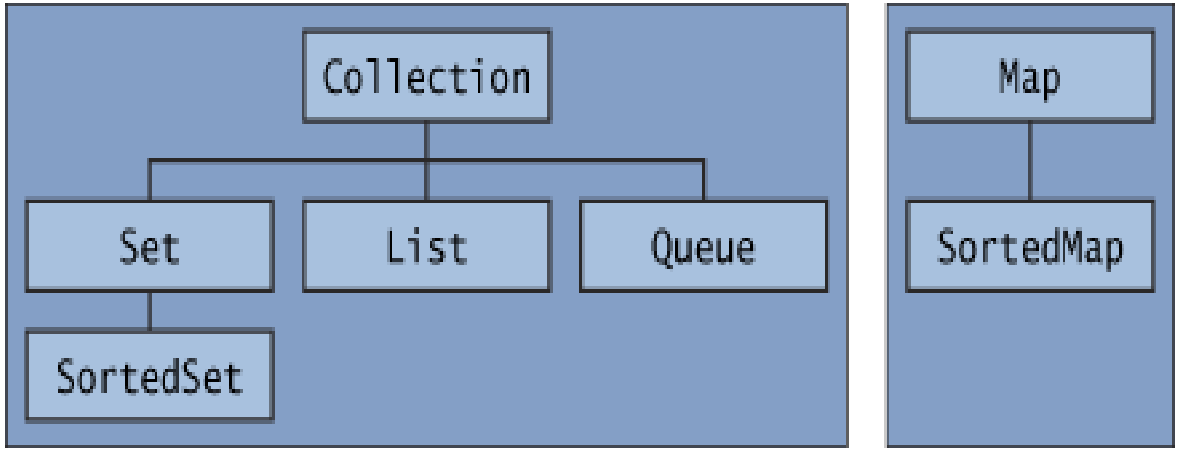
- É comum usarmos um objeto que armazene vários outros
  - ✓ Agenda é um exemplo
- API Collections forneces interfaces e classes para coleções
  - ✓ pacote java.util
- Uma coleção é um objeto que agrupa vários outros
  - ✓ Também chamado contêiner
- Interface Collection

***API - Application Programming Interface***

# Interfaces


<<interface>>
Iterable
<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
1.5
<<interface>>
Queue
<<interface>>
SortedSet
1.6
<<interface>>
NavigableSet
1.6
<<interface>>
Deque
java.util.concurrent
1.5
<<interface>>
BlockingQueue
1.6
<<interface>>
BlockingDeque
<<interface>>
Map
<<interface>>
SortedMap
1.6
<<interface>>
NavigableMap
java.util.concurrent
1.5
<<interface>>
ConcurrentMap
1.6
<<interface>>
ConcurrentNavigableMap

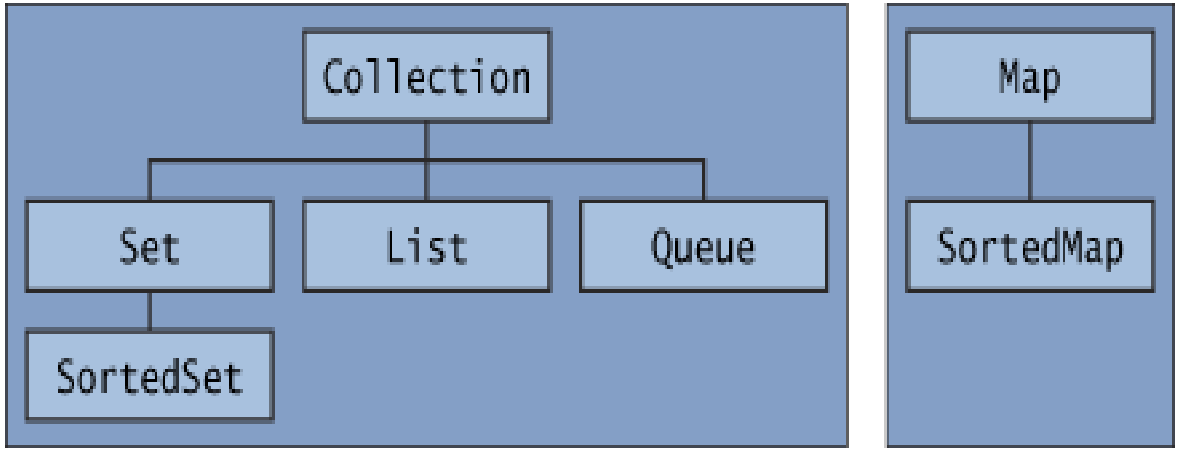
# Interfaces




**A API oferece alguns tipos de coleções: conjuntos, listas e mapas.**

- ***Conjunto (Set e SortedSet)***
  - Uma coleção de elementos que não mantém uma ordem nem uma contagem dos elementos .
  - Cada elemento ou está no conjunto ou não (não há elementos repetidos)

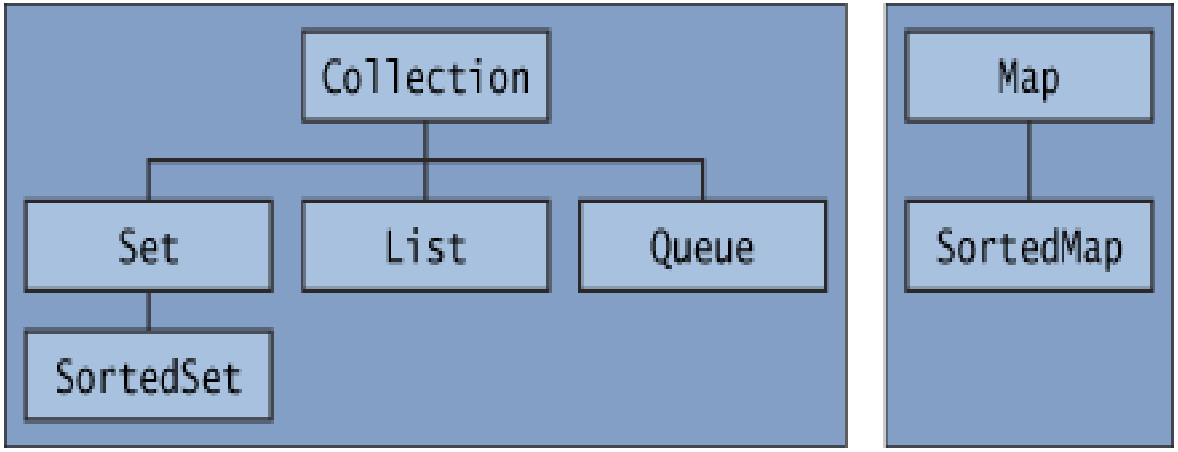
# Interfaces




**A API oferece alguns tipos de coleções: conjuntos, listas e mapas.**

- **Lista (List)**
  - uma sequência de elementos.
  - mantém dados a respeito de ambos, a ordem e a contagem.
- **Fila (Queue)**
  - Fila de elementos.
  - Modelo FIFO (First in, first out)

# Interfaces




*Map e SortedMap armazenam pares (chave, valor) de Elementos.*

**A API oferece alguns tipos de coleções: conjuntos, listas e mapas.**

- **Mapa (Map e SortedMap)**
  - Uma associação entre chaves e valores.
  - Mantém um conjunto de chaves e mapeia cada chave para um único valor.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE
Campus Natal - Zona Norte

# Coleções

A API organiza suas classes da seguinte forma:

- Uma hierarquia de interfaces
- As especificações dos vários tipos
- Uma hierarquia de implementação de classes

| | | Implementations | | | | |
|---|---|---|---|---|---|---|
| | | Hash Table | Resizable Array | Balanced Tree | Linked List | Hash Table + Linked List |
| Interfaces | Set | HashSet | | TreeSet | | LinkedHashSet |
| | List | | ArrayList | | LinkedList | |
| | Map | HashMap | | TreeMap | | LinkedHashMap |

# Interface Collection

- É a raiz da hierarquia da coleções.
- Representa um grupo de objetos conhecidos como seus elementos.
- Contém o denominador comum do que todas as coleções implementam.
- Usada para passagem de coleções como parâmetros e para manipulação genérica de grupos de dados.
- A plataforma Java não fornece implementação direta desta interface.
- Implementações são de suas sub-interfaces.
- Possui métodos para acessar, buscar e remover elementos.

# Collection

```
public interface Collection<E> extends Iterable<E> {
    //Operações básicas
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator iterator();

    //Operações na coleção
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();

    //Operações de arrays
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

# Set

- Um conjunto é uma coleção que não tem elementos duplicados.
- Podem ser mantidos ordenados (SortedSet) ou não.
- Modela a abstração matemática para conjuntos.
- Três implementações
  - HashSet: implementado com tabela hash.
  - TreeSet: implementado com árvore.
  - LinkedHashSet: implementado com tabela hash e listas ligadas.

# Set

```
public static void main(String[] args) {
  Set<String> nomes = new HashSet();
  nomes.add("Joao");
  nomes.add("Jose");
  nomes.add("Maria");
  nomes.add("Bianca");
  System.out.println("Quantidade de elementos: " + nomes.size());
  if (nomes.contains("Maria"))
      System.out.println("Contém Maria");
  nomes.add("Jose");
  System.out.println("Quantidade de elementos: " + nomes.size());
}
```

Ao ser criado, informamos o tipo (classe) do elemento a ser armazenado na coleção:

HashSet<String> nomes = new HashSet()

Isto porque a coleção é implementada com tipos genéricos, onde o tipo é determinado no momento da execução

# Percorrendo coleções

- Iterator: objeto que permite que todos os elementos da coleção sejam acessados

  ✓ Não se sabe sobre a implementação
  ✓ boolean hasNext() - informa se ainda há elementos a serem “visitados”
  ✓ <T> next() - retorna o próximo elemento a ser visitado

- for-each

  ✓ É implementado usando o iterator
  ✓ for (Tipo objeto: Coleção)
  ✓ Exemplo: for (String nome: nomes) { ... }

# Exemplo

*Com for-each:*

```
public static void main(String[] args) {
  Collection<String> nomes = new HashSet();
  ...
  System.out.println("Qtd elementos: “ + nomes.size());
  for (String string : nomes) {
    System.out.println(string);
  }
}
```

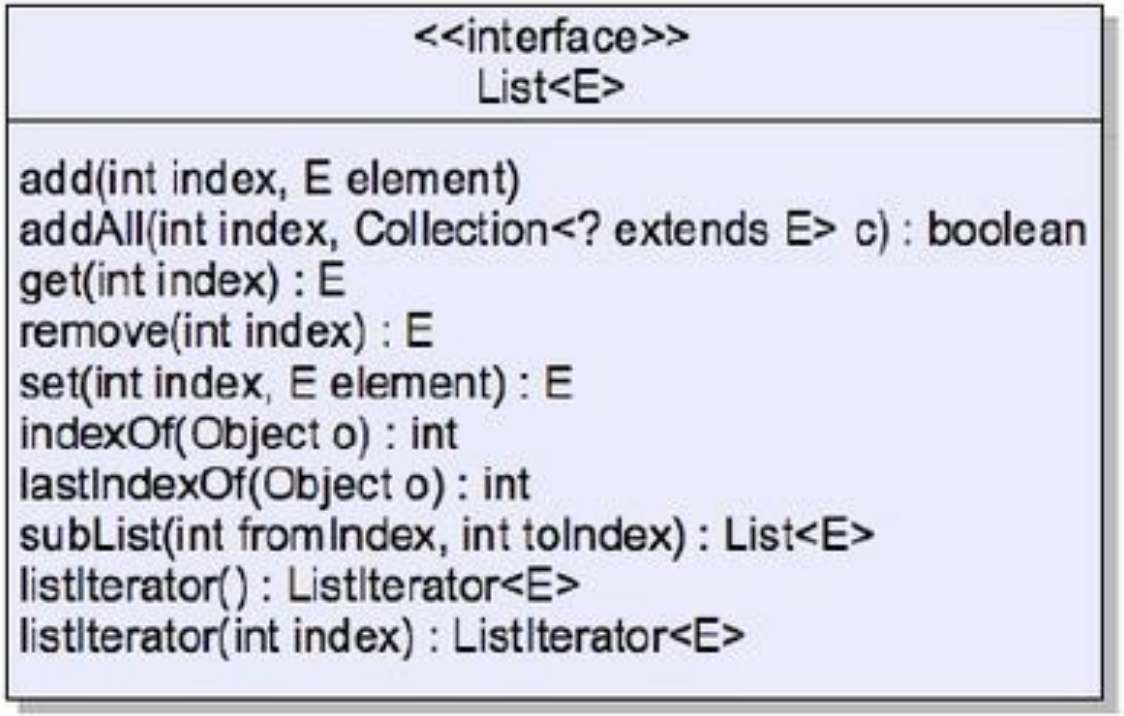
*Com iterator:*

```
public static void main(String[] args) {
  Collection<String> nomes = new TreeSet();
  ...
  System.out.println("Qtd elementos: “ + nomes.size());
  Iterator<String> iterator = nomes.iterator();
  while (iterator.hasNext()){
    System.out.println(iterator.next());
  }
}
```

# List

- É uma coleção ordenada.
- Podem conter elementos duplicados.
- Em adição as operações de Collection, inclui:
  - ✓ Acesso posicional.
  - ✓ Manipula elementos baseado na sua posição numérica na lista (índice).
  - ✓ Pesquisa por um objeto específico na lista e retorna a sua posição numérica.
  - ✓ Iteração.

# List


<<interface>>
List<E>
add(int index, E element)
addAll(int index, Collection<? extends E> c) : boolean
get(int index) : E
remove(int index) : E
set(int index, E element) : E
indexOf(Object o) : int
lastIndexOf(Object o) : int
subList(int fromIndex, int toIndex) : List<E>
listIterator() : ListIterator<E>
listIterator(int index) : ListIterator<E>

# List

## Duas implementações:

- ✓ ArrayList: Lista implementada com Array.
- ✓ LinkedList: Lista implementada com lista ligada.

## Métodos de acesso através de índice

- ✓ get(int i): retorna elemento no índice i.
- ✓ remove(int i): remove elemento no índice i e reorganiza os elementos para preencher o espaço deixados.
- ✓ add(<T> elemento): Adiciona elemento depois do último índice.

**OBS: Elementos começam no índice zero**

# List

- Cada elemento tem o seu sucessor (menos o último) e o seu antecessor (menos o primeiro).

- As operações mais importantes em listas são:
  - ✓ Adicionar um objeto em qualquer lugar da lista;
  - ✓ Remover um objeto de qualquer lugar da lista;
  - ✓ Obter o elemento de qualquer lugar da lista;
  - ✓ Percorrer os elementos da lista;
  - ✓ Verificar se um elemento está na lista;
  - ✓ Descobrir o índice de um elemento na lista;
  - ✓ Obter o número de elementos da coleção.

# Exemplo

```
public static void main(String[] args) {
  ArrayList<String> nomes = new ArrayList();
  nomes.add("João");
  nomes.add("José");
  nomes.add("Maria");
  nomes.add("Bianca");
  System.out.println("Quantidade de elementos: “ + nomes.size());
  String nome = nomes.get(3);
  System.out.printf(“Nome[3]: %s \n”, nome);
}
```

# Ordenação de uma lista simples

```
public static void main(String[] args) {
  ArrayList<String> nomes = new ArrayList();
  nomes.add("João");
  nomes.add("José");
  nomes.add("Maria");
  nomes.add("Bianca");
  Collections.sort(nomes);
  for (String nome : nomes)
        System.out.println(nome);
}
```

# Exemplo

```
public static void main(String[] args) {
  ArrayList nomes = new ArrayList();
  nomes.add("João");
  nomes.add("José");
  nomes.add(12345);
  nomes.add("Maria");
  nomes.add("Bianca");
  System.out.println("Quantidade de elementos: “ + nomes.size());
  String nome = nomes.get(3).toString();
  System.out.printf(“Nome[3]: %s \n”, nome);
}
```

# Map

- Um mapa armazena pares (chave, valor) chamados de itens.
- Chaves e valores podem ser de qualquer tipo.
- As chaves armazenadas nos mapas podem estar ordenadas ou não.
- A chave é utilizada para achar o elemento de forma rápida, utilizando estruturas especiais.
- Um mapa não pode conter chaves duplicadas
- Modela a abstração matemática de função.
- Também são conhecidos como dicionários.

# Map

As operações mais importantes em mapas são:

- Adicionar um item no mapa, fornecendo chave e valor;
- Remover um item baseado em uma chave;
- Percorrer os itens da coleção;
- Verificar se um item está na coleção, fornecendo a chave;
- Obter o número de elementos da coleção.

# Map

- A plataforma Java contém três implementações:
  - ✓ HashMap
  - ✓ TreeMap
  - ✓ LinkedHashMap

- O comportamento e a performance são análogos ao de HashSet, TreeSet e LinkedHashSet.

- **Principais métodos**
  - ✓ put (chave, valor): coloca par chave/elemento no mapa
  - ✓ get (chave): retorna o valor correspondente a chave passada como parâmetro

# Exemplo

```
public static void main(String[] args) {
  HashMap<String,String> nomes = new HashMap();
  nomes.put("joao", "Joao Da Silva");
  nomes.put("jose", "Jose Matos");
  nomes.put("maria", "Maria das Dores");
  nomes.put("bianca","Bianca Patrícia");
  System.out.println("Quantidade de elementos: "+nomes.size());
  String nome = nomes.get("maria");
  System.out.println(nome);
}
```

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE
Campus Natal - Zona Norte

# Dúvidas

# Exercícios

- Entrar com uma lista de valores inteiros, até que o usuário digite 0. Exibir o menor e o maior números da lista.
- Entrar com uma lista de valores inteiros, até que o usuário digite 0. Exibir a soma do números e o tamanho dessa lista.
- Cria uma classe Banco com diversas contas e use o nome do usuário como chave para encontrar uma conta específica.
- Faça o exercício do link:

http://www.inf.furb.br/~poo/tutoriais/poo-lab5.html