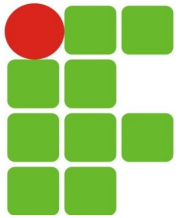


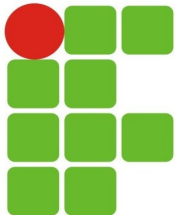
Coleções

João Paulo Q. dos Santos

joao.queiroz@ifrn.edu.br

Roteiro

- Conceitos sobre coleções;
- Tipos de coleções;
- Como são classificadas;
- Exemplos.

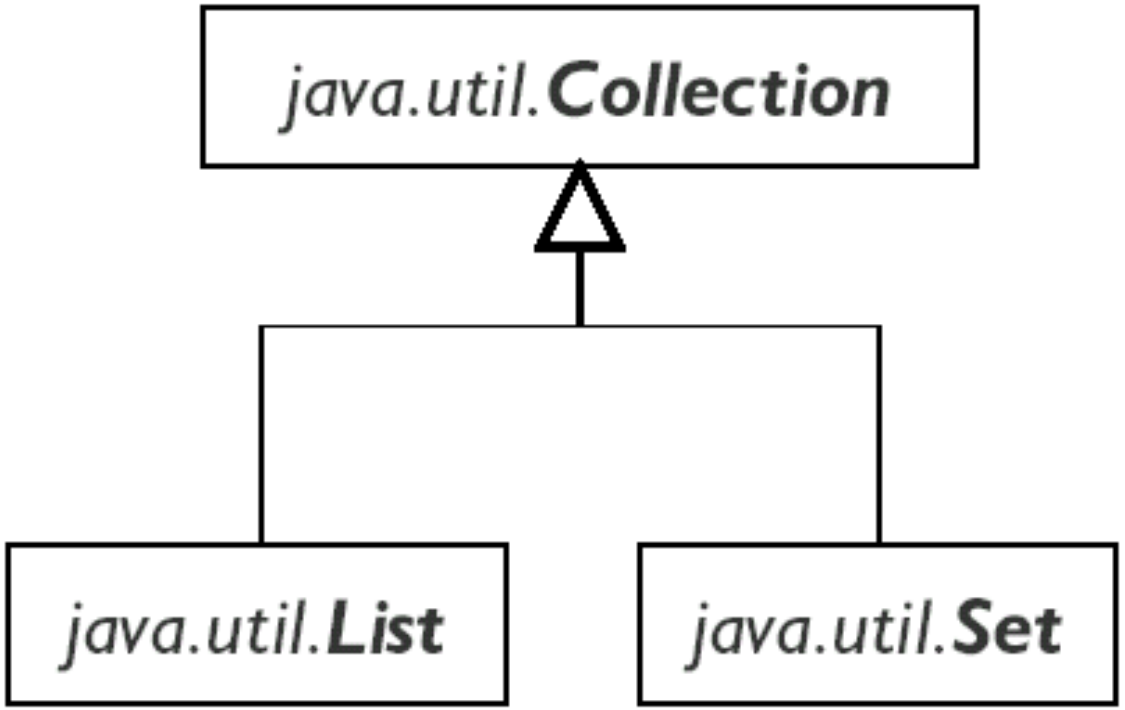


Coleções

- Classes e interfaces do pacote **java.util** que representam listas, conjuntos e mapas.
- Solução flexível para armazenar objetos.
- Quantidade armazenada de objetos não é fixa, como ocorre com arrays.
- Poucas interfaces (duas servem de base) permitem maior reuso e um vocabulário menor de métodos.
  - `add()`, `remove()`, `contains()` - principais métodos de ***`Collection`***
  - `put()`, `get()` - principais métodos de interface ***`Map`***
- Desvantagens:
  - Menos eficientes que arrays.
  - A partir da versão do j2se 1.5 foi possível incluir um tipo primitivo diretamente nas coleções devido ao recurso do autoboxing.

Coleções

Coleções de elementos individuais



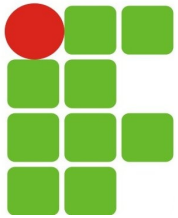
- seqüência definida
- elementos indexados

- seqüência arbitrária
- elementos não repetem

Coleções de pares de elementos

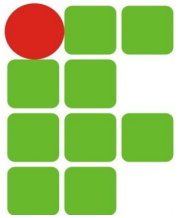
java.util.**Map**

- Pares chave/valor (vetor associativo)
- **Collection** de valores (podem repetir)
- **Set** de chaves (unívocas)



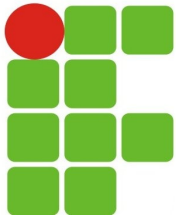
Collection

- Uma coleção representa um grupo de objetos.
  - Algumas coleções permitem valores repetidos, outras não.
  - Algumas coleções são ordenadas, outras não.
  - Principais subinterfaces:
    - **List**
    - **Set**
  - Principais métodos:
    - **boolean add(Object o)**: adiciona objeto na coleção
    - **boolean contains(Object o)**
    - **boolean isEmpty()**
    - **Iterator iterator()**: retorna iterator
    - **boolean remove(Object o)**
    - **int size()**: retorna o número de elementos
    - **Object[] toArray(Object[])**: converte coleção em Array



List

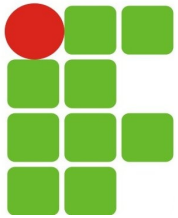
- Estende a interface ***`Collection`***.
- Principais subclasses:
  - **`Vector`** – ideal para acesso randômico. Sincronizado.
  - **`ArrayList`** – ideal para acesso randômico. Não sincronizada.
  - **`LinkedList`** – ideal para acesso sequencial. Não sincronizada.
- Principais métodos adicionais
  - **`void add(int index, Object o)`**: adiciona objeto na posição indicada (empurra elementos existentes para a frente)
  - **`Object get(int index)`**: recupera objeto pelo índice
  - **`int indexOf(Object o)`**: procura objeto e retorna índice da primeira ocorrência
  - **`Object set(int index, Object o)`**: grava objeto na posição indicada (apaga qualquer outro que ocupava a posição).
  - **`Object remove(int index)`**
  - ***`ListIterator`*** **`listIterator()`**: retorna uma ***`ListIterator`***



Vector

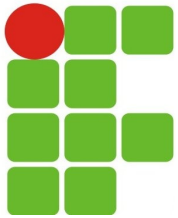
- Implementa um array redimensionável bastante eficiente para leitura.
- Possui um construtor na forma: `Vector(int capacidadeInicial, int incremento)`
  - **capacidadeInicial** – o tamanho inicial do vector (se não especificado no construtor, assume o tamanho de 10 elementos).
  - **incremento** – especifica em quantos elementos o vector deverá crescer quando sua capacidade for extrapolada (Se não especificado no construtor, o vector terá seu tamanho duplicado).
- Implementado internamente com arrays.
- Otimiza o espaço alocado pelo array que encapsula.
- Ideal para acesso aleatório.
- É sincronizado, permitindo que somente uma *thread* por vez acesse um método sincronizado do objeto.
- Métodos comumente utilizados:

`add(Object o)`	`contains(Object element)`
`elementAt(int index)`	`toArray()`
`setElementAt(Object obj, int index)`	`size()`
`remove(Object o)`	`remove(int index)`
`lastIndexOf(Object element)`	`isEmpty()`
`indexOf(Object element)`	



LinkedList

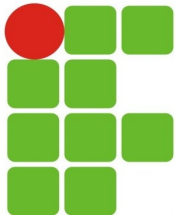
- Muito mais eficiente que **ArrayList** e **Vector** para remoção e inserção no meio da lista.
- Pode ser usada para implementar pilhas, filas unidirecionais (queue) e bidirecionais (deque – double -ended queue). Possui métodos para manipular essas estruturas.
- Ideal para acesso sequencial.
- Não sincronizado.
- Principais métodos:
  - **add(int index, Object element)**
  - **remove(int index)**
  - **get(int index)**
  - **indexOf(Object o)**
  - **lastIndexOf(Object o)**



Listas

- A implementação mais utilizada da interface List é ArrayList.
- ArrayList é ideal para pesquisa
- LinkedList é ideal para inserção e remoção de itens nas pontas.
- A partir do Java 5 podemos usar o recurso de Generics para restringir as listas a um determinado tipo de objetos (e não qualquer Object):

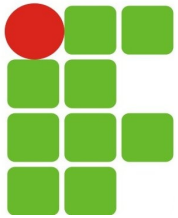
```
List<ContaCorrente> contas = new ArrayList<ContaCorrente>();

ContaCorrente c1, c2, c3;

contas.add(c1);
contas.add(c3);
contas.add(c2);
```



Listas

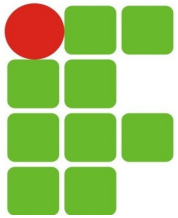
- O uso de Generics também elimina a necessidade de casting, já que seguramente todos os objetos inseridos na lista serão do tipo ContaCorrente:

```
for(int i = 0; i < contas.size(); i++) {

   ContaCorrente cc = contas.get(i); // sem casting!
   System.out.println(cc.getSaldo());

}
```



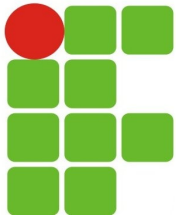
Collection

- Esta classe contém métodos estáticos que retornam ou operam sobre coleções.
- Principais métodos:
  - `binarySearch(List list, Object key)` – Busca um determinado elemento de uma lista em ordem crescente usando o algoritmo de busca binária.
  - `copy(List dest, List src)` – Copia todos os elementos de uma lista para outra.
  - `fill(List list, Object obj)` – Substitui todos os elementos da lista pelo objeto especificado.
  - `sort(List list)` – Orderna uma lista usando uma modificação do algoritmo *Mergesort*.
  - `shuffle(List list)` – Randomicamente permuta os elementos de uma lista.
  - `reverse(List list)` – inverte a ordem dos elementos da lista.
  - `max(Collection coll)` – retorna o maior elemento da coleção.
  - `min(Collection coll)` – retorna o menor elemento da coleção.



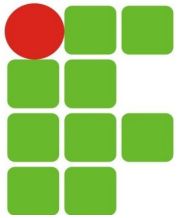
Ordenação

```
List<String> lista = new ArrayList<String>();

lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");

System.out.println(lista);

Collections.sort(lista);

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



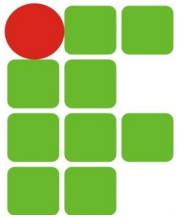
Set

- Set representa um conjunto matemático.
- Não possui valores repetidos.
- Pode possuir um único elemento `null`.
- Principais subclasses:
  - `HashSet` (implements *`Set`*)
  - `TreeSet` (implements *`SortedSet`*)
- Principais métodos:
  - `add(Object o)`
  - `contains(Object o)`
  - `equals(Object o)`
  - `isEmpty()`
  - `iterator()`
  - `remove(Object o)`
  - `size()`
  - `toArray()`



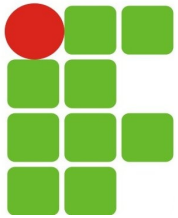
Map

- Objetos Map são semelhantes a arrays mas, em vez de índices numéricos, usam objetos como chaves.
- Chaves são únicas.
- Valores podem ser duplicados.
- Principais subclasses:
  - **`Hashtable`** – Sincronizada. Não aceita **`null`** como chave. Acesso rápido devido ao uso do método **`hashCode()`**.
  - **`HashMap`** – Não sincronizada. Aceita **`null`** como chave. Acesso rápido devido ao uso do método **`hashCode()`**.
  - **`TreeMap`** – Não sincronizada. Mapa ordenado. Contém métodos para manipular elementos ordenados.
- Métodos principais:
  - **`void put(Object key, Object value)`**: acrescenta um objeto
  - **`Object get(Object key)`**: recupera um objeto
  - **`Set keySet()`**: retorna um **`Set`** de chaves
  - **`Collection values()`**: retorna uma **`Collection`** de valores.



Exemplo Prático - ArrayList

```
public class Pessoa {

    private String nome, endereco;

    private int telefone;

    public String getEndereco() {
        return this.endereco;
    }

    public String getNome() {
        return this.nome;
    }

    public int getTelefone() {
        return this.telefone;
    }

    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setTelefone(int i) {
        this.telefone = i;
    }
}
```



Exemplo Prático - Vector

```
package br.edu.ifrn.arraylist;

import java.util.ArrayList;

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

        Pessoa pessoa = new Pessoa();

        ArrayList<Pessoa> contatos = new ArrayList<Pessoa>();
        pessoa.setNome("Joao Jose");
        pessoa.setEndereco("Av. Sao Jose");
        pessoa.setTelefone(32334545);

        contatos.add(pessoa);

        pessoa = new Pessoa();
        pessoa.setNome("Maria");
        pessoa.setEndereco("Av. Santos");
        pessoa.setTelefone(36224545);

        contatos.add(pessoa);

        for (int i = 0; i < contatos.size(); i++) {

            System.out.println("Nome : "+contatos.get(i).getNome());
            System.out.println("End. : "+contatos.get(i).getEndereco());
            System.out.println("Fone : "+contatos.get(i).getTelefone());

            System.out.println(" ");

        }
    }
}
```

Dúvidas