Coleções de Objetos
Interface List
Classe ArrayList

Profa. Karen Selbach Borges

Última Atualização: Jun/2021

Interface List

- Implementa uma seqüência ordenada de objetos: controle sobre a posição em que o objeto é inserido através de um índice
- Pode conter objetos duplicados
- Principais Implementações:
  - java.util.ArrayList
  - java.util.LinkedList
  - java.util.Deque

Métodos

- Implementa todos os métodos da interface Collection, mais alguns que permitem a manipulação da coleção a partir do índice.

Adição de Elementos

- **Object set(int index, E element)** : seta um elemento em uma dada posição.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add(“Jose”);
l.add(“Maria”);
l.add(“João”); // l = { “José”, “Maria”, “João”}
l.set(1, “Ana”); // l = { “José”, “Ana”, “João”}
```

Adição de Elementos

- **Object add(int index, E element)** : adiciona um elemento em uma dada posição.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add(“Jose”);
l.add(“Maria”);
l.add(“João”); // l = { “José”, “Maria”, “João”}
l.add(1, “Ana”); // l = { “José”, “Ana”, “Maria”, “João”}
```

Adição de Elementos

- **boolean addAll(int index, Collection c)** : adiciona uma coleção de elementos a partir do índice passado como argumento.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add(“Jose”);
l.add(“Maria”); // l = {“José”, “Maria”}
Collection<String> c2 = new ArrayList<String> ();
c2.add(“Roberto”);
c2.add(“Ana”); c2 = {“Roberto”, “Ana”}
l.addAll(1, c2); // l = { “José”, “Roberto”, “Ana”, “Maria”}
```

Remoção de Elementos

- **Object remove(int index)** : remove um objeto dada sua posição.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("João"); // l = {"José", "Maria", "João"}
l.remove(1); // l = {"José", "João"}
```

Recuperação de Elementos

- **E get(int index)** : retorna o elemento de uma dada posição.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("João");
int tamanho = l.size();
for (int i=0; i<tamanho; i++) {
  String nome = l.get(i);
  System.out.println(nome);
}
```

Recuperação de Elementos

- **int indexOf (Object o)** : retorna a posição de um objeto.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add(“Jose”);
l.add(“Maria”);
l.add(“João”);
System.out.println(l.indexOf(“Maria”)); //1
```

Recuperação de Elementos

- **int lastIndexOf (Object o)** : retorna o último índice de um objeto.
- Exemplo :

```
List<String> l = new ArrayList<String> ();
l.add(“Maria”);
l.add(“Pedro”);
l.add(“José”);
l.add(“Maria”);
System.out.println(l.lastIndexOf(“Maria”)); //3
```

Listagem dos Elementos

- **ListIterator listIterator():** retorna o iterador da lista.
- Um ListIterator, assim como um Iterator, possui um método para recuperar o próximo elemento (hasNext() e next()), mas possibilita recuperar o elemento anterior (hasPrevious() e previous()).

Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("Joao");
ListIterator li = l.ListIterator();
while(li.hasNext()) {
  String proximo = li.next();
  System.out.println(proximo);
}
while(li.hasPrevious()) {
  String anterior = li.previous();
  System.out.println(anterior);
}
```

Listagem dos Elementos

- **ListIterator listiterator(int index):** retorna o ListIterator da lista a partir de uma determinada posição.

- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("Joao");
ListIterator li = l.LisIterator(1);
while(li.hasNext()) {
  String nome = li.next();
  System.out.println(nome);
}
```

Listagem dos Elementos

- **List<E> subList (int fromIndex, int toIndex):** retorna uma sublista.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("João");
l.add("Gilberto");
l.add("Alfredo"); //l= { "José", "Maria", "João", "Gilberto", "Alberto"}
List subList = l.subList(1,3); //sublist = {"Maria", "João", "Gilberto" }
```

Classe ArrayList

- Encapsula um array de objetos realocado dinamicamente
- Não implementa controle de sincronismo:
  - Menos tempo de processamento 👍
  - Não pode ser usado com threads 👎
- Implementa os métodos das interfaces :
  - Collection
  - List

Classe ArrayList

Construtor	Descrição
ArrayList()	Cria um ArrayList com 10 posições
ArrayList(int initialCapacity)	Cria um ArrayList com o número de posições definidas no parâmetro
ArrayList(Collection c)	Cria um ArrayList e já preenche ele com os elementos contidos na coleção

Métodos da Classe ArrayList

- **Object clone ()** : cria uma cópia da coleção.
- Exemplo :

```
List<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria"); //l = {"José", "Maria"}
List<String> l2 = l.clone(); //l2 = {"José", "Maria"}
```

Métodos da Classe ArrayList

- **removeRange**(int fromIndex, int toIndex): remove do ArrayList os elementos que se encontram posicionados entre os índices definidos no parâmetro.
- Exemplo

```
ArrayList<String> l = new ArrayList<String>();
l.add("Jose");
l.add("Maria");
l.add("João");
l.add("Gilberto");
l.add("Alfredo"); //l= { "José", "Maria", "João", "Gilberto", "Alberto"}
l.removeRange(1,3); //l= {"José", "Alberto"}
```