JAVA - THE LINKEDLIST CLASS

The LinkedList class extends AbstractSequentialList and implements the List interface. It provides a linked-list data structure.

Given below are the constructors supported by the LinkedList class.

SN	Constructors and Description
1	**LinkedList** This constructor builds an empty linked list.
2	**LnkedList***Collectionc* This constructor builds a linked list that is initialized with the elements of the collection c.

Apart from the methods inherited from its parent classes, LinkedList defines following methods:

SN	Methods with Description
1	**void add***intindex, Objectelement* Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range $index < 0 \| \| index > size()$.
2	**boolean add***Objecto* Appends the specified element to the end of this list.
3	**boolean addAll***Collectionc* Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null
4	**boolean addAll***intindex, Collectionc* Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5	**void addFirst***Objecto* Inserts the given element at the beginning of this list.
6	**void addLast***Objecto* Appends the given element to the end of this list.
7	**void clear** Removes all of the elements from this list.

8 **Object clone**

Returns a shallow copy of this LinkedList.

9 **boolean contains*Objecto***

Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that $o == null ? e == null : o.equals(e)$.

10 **Object get*intindex***

Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range $index < 0 || index >= size()$.

11 **Object getFirst**

Returns the first element in this list. Throws NoSuchElementException if this list is empty.

12 **Object getLast**

Returns the last element in this list. Throws NoSuchElementException if this list is empty.

13 **int indexOf*Objecto***

Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

14 **int lastIndexOf*Objecto***

Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

15 **ListIterator listIterator*intindex***

Returns a list-iterator of the elements in this list $inpropersequence$, starting at the specified position in the list. Throws IndexOutOfBoundsException if the specified index is is out of range $index < 0 || index >= size()$.

16 **Object remove*intindex***

Removes the element at the specified position in this list. Throws NoSuchElementException if this list is empty.

17 **boolean remove*Objecto***

Removes the first occurrence of the specified element in this list. Throws NoSuchElementException if this list is empty. Throws IndexOutOfBoundsException if the specified index is is out of range $index < 0 || index >= size()$.

18 **Object removeFirst**

Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty.

19 **Object removeLast**

Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty.

20 **Object set***intindex, Objectelement*

Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range $index < 0 || index >= size()$.

21 **int size**

Returns the number of elements in this list.

22 **Object[] toArray**

Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.

23 **Object[] toArray***Object[]a*

Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

Example:

The following program illustrates several of the methods supported by LinkedList:

```
import java.util.*;

public class LinkedListDemo {

   public static void main(String args[]) {
      // create a linked list
      LinkedList ll = new LinkedList();
      // add elements to the linked list
      ll.add("F");
      ll.add("B");
      ll.add("D");
      ll.add("E");
      ll.add("C");
      ll.addLast("Z");
      ll.addFirst("A");
      ll.add(1, "A2");
      System.out.println("Original contents of ll: " + ll);

      // remove elements from the linked list
      ll.remove("F");
      ll.remove(2);
      System.out.println("Contents of ll after deletion: "
       + ll);

      // remove first and last elements
      ll.removeFirst();
      ll.removeLast();
      System.out.println("ll after deleting first and last: "
       + ll);

      // get and set a value
      Object val = ll.get(2);
      ll.set(2, (String) val + " Changed");
      System.out.println("ll after change: " + ll);
   }
}
```

This would produce the following result:

```
Original contents of ll: [A, A2, F, B, D, E, C, Z]
Contents of ll after deletion: [A, A2, D, E, C, Z]
ll after deleting first and last: [A2, D, E, C]
ll after change: [A2, D, E Changed, C]
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js