java.util

# Class LinkedList<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.AbstractSequentialList<E>
        java.util.LinkedList<E>

### Type Parameters:

`E` - the type of elements held in this collection

### All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, List<E>, Queue<E>

---

```
public class LinkedList<E>
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

Doubly-linked list implementation of the `List` and `Deque` interfaces. Implements all optional list operations, and permits all elements (including `null`).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

**Note that this implementation is not synchronized.** If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the `Collections.synchronizedList` method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

```
List list = Collections.synchronizedList(new LinkedList(...));
```

The iterators returned by this class's `iterator` and `listIterator` methods are *fail-fast*: if the list is structurally modified at any time after the iterator is created, in any way except through the Iterator's own `remove` or `add` methods, the iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

This class is a member of the Java Collections Framework.

### Since:

1.2

### See Also:

`List`, `ArrayList`, Serialized Form

## Field Summary

### Fields inherited from class java.util.AbstractList

| |
|---|
| `modCount` |

## Constructor Summary

**Constructors**

| Constructor and Description |
| --- |
| `LinkedList()`<br>Constructs an empty list. |
| `LinkedList(Collection<? extends E> c)`<br>Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. |

## Method Summary

**Methods**

| Modifier and Type | Method and Description |
| --- | --- |
| boolean | `add(E e)`<br>Appends the specified element to the end of this list. |
| void | `add(int index, E element)`<br>Inserts the specified element at the specified position in this list. |
| boolean | `addAll(Collection<? extends E> c)`<br>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. |
| boolean | `addAll(int index, Collection<? extends E> c)`<br>Inserts all of the elements in the specified collection into this list, starting at the specified position. |
| void | `addFirst(E e)`<br>Inserts the specified element at the beginning of this list. |
| void | `addLast(E e)`<br>Appends the specified element to the end of this list. |
| void | `clear()`<br>Removes all of the elements from this list. |
| `Object` | `clone()`<br>Returns a shallow copy of this `LinkedList`. |
| boolean | `contains(Object o)`<br>Returns `true` if this list contains the specified element. |
| `Iterator<E>` | `descendingIterator()`<br>Returns an iterator over the elements in this deque in reverse sequential order. |
| `E` | `element()`<br>Retrieves, but does not remove, the head (first element) of this list. |
| `E` | `get(int index)`<br>Returns the element at the specified position in this list. |
| `E` | `getFirst()`<br>Returns the first element in this list. |
| `E` | `getLast()`<br>Returns the last element in this list. |
| int | `indexOf(Object o)`<br>Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| int | `lastIndexOf(Object o)`<br>Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| `ListIterator<E>` | `listIterator(int index)`<br>Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. |
| boolean | `offer(E e)` |

| | |
|---|---|
| | Adds the specified element as the tail (last element) of this list. |
| boolean | **offerFirst**(E e)<br>Inserts the specified element at the front of this list. |
| boolean | **offerLast**(E e)<br>Inserts the specified element at the end of this list. |
| E | **peek**()<br>Retrieves, but does not remove, the head (first element) of this list. |
| E | **peekFirst**()<br>Retrieves, but does not remove, the first element of this list, or returns null if this list is empty. |
| E | **peekLast**()<br>Retrieves, but does not remove, the last element of this list, or returns null if this list is empty. |
| E | **poll**()<br>Retrieves and removes the head (first element) of this list. |
| E | **pollFirst**()<br>Retrieves and removes the first element of this list, or returns null if this list is empty. |
| E | **pollLast**()<br>Retrieves and removes the last element of this list, or returns null if this list is empty. |
| E | **pop**()<br>Pops an element from the stack represented by this list. |
| void | **push**(E e)<br>Pushes an element onto the stack represented by this list. |
| E | **remove**()<br>Retrieves and removes the head (first element) of this list. |
| E | **remove**(int index)<br>Removes the element at the specified position in this list. |
| boolean | **remove**(Object o)<br>Removes the first occurrence of the specified element from this list, if it is present. |
| E | **removeFirst**()<br>Removes and returns the first element from this list. |
| boolean | **removeFirstOccurrence**(Object o)<br>Removes the first occurrence of the specified element in this list (when traversing the list from head to tail). |
| E | **removeLast**()<br>Removes and returns the last element from this list. |
| boolean | **removeLastOccurrence**(Object o)<br>Removes the last occurrence of the specified element in this list (when traversing the list from head to tail). |
| E | **set**(int index, E element)<br>Replaces the element at the specified position in this list with the specified element. |
| int | **size**()<br>Returns the number of elements in this list. |
| Object[] | **toArray**()<br>Returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| <T> T[] | **toArray**(T[] a)<br>Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. |

## Methods inherited from class java.util.AbstractSequentialList

iterator

## Methods inherited from class java.util.AbstractList

equals, hashCode, listIterator, removeRange, subList

### Methods inherited from class java.util.AbstractCollection

containsAll, isEmpty, removeAll, retainAll, toString

### Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

### Methods inherited from interface java.util.List

containsAll, equals, hashCode, isEmpty, iterator, listIterator, removeAll, retainAll, subList

### Methods inherited from interface java.util.Deque

iterator

## Constructor Detail

### LinkedList

```
public LinkedList()
```

Constructs an empty list.

### LinkedList

```
public LinkedList(Collection<? extends E> c)
```

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

**Parameters:**

c - the collection whose elements are to be placed into this list

**Throws:**

NullPointerException - if the specified collection is null

## Method Detail

### getFirst

```
public E getFirst()
```

Returns the first element in this list.

**Specified by:**

getFirst in interface Deque<E>

**Returns:**

the first element in this list

**Throws:**

NoSuchElementException - if this list is empty

## getLast

```
public E getLast()
```

Returns the last element in this list.

**Specified by:**

`getLast` in interface `Deque<E>`

**Returns:**

the last element in this list

**Throws:**

`NoSuchElementException` - if this list is empty

## removeFirst

```
public E removeFirst()
```

Removes and returns the first element from this list.

**Specified by:**

`removeFirst` in interface `Deque<E>`

**Returns:**

the first element from this list

**Throws:**

`NoSuchElementException` - if this list is empty

## removeLast

```
public E removeLast()
```

Removes and returns the last element from this list.

**Specified by:**

`removeLast` in interface `Deque<E>`

**Returns:**

the last element from this list

**Throws:**

`NoSuchElementException` - if this list is empty

## addFirst

```
public void addFirst(E e)
```

Inserts the specified element at the beginning of this list.

**Specified by:**

`addFirst` in interface `Deque<E>`

**Parameters:**

`e` - the element to add

## addLast

```
public void addLast(E e)
```

Appends the specified element to the end of this list.

This method is equivalent to `add(E)`.

**Specified by:**

`addLast` in interface `Deque<E>`

**Parameters:**

`e` - the element to add

## contains

```
public boolean contains(Object o)
```

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))`.

**Specified by:**

`contains` in interface `Collection<E>`

**Specified by:**

`contains` in interface `Deque<E>`

**Specified by:**

`contains` in interface `List<E>`

**Overrides:**

`contains` in class `AbstractCollection<E>`

**Parameters:**

`o` - element whose presence in this list is to be tested

**Returns:**

`true` if this list contains the specified element

## size

```
public int size()
```

Returns the number of elements in this list.

**Specified by:**

`size` in interface `Collection<E>`

**Specified by:**

`size` in interface `Deque<E>`

**Specified by:**

`size` in interface `List<E>`

**Specified by:**

`size` in class `AbstractCollection<E>`

**Returns:**

the number of elements in this list

## add

```
public boolean add(E e)
```

Appends the specified element to the end of this list.

This method is equivalent to `addLast(E)`.

**Specified by:**

`add` in interface `Collection<E>`

**Specified by:**

`add` in interface `Deque<E>`

**Specified by:**

`add` in interface `List<E>`

**Specified by:**

`add` in interface `Queue<E>`

**Overrides:**

`add` in class `AbstractList<E>`

**Parameters:**

`e` - element to be appended to this list

**Returns:**

`true` (as specified by `Collection.add(E)`)

## remove

```
public boolean remove(Object o)
```

Removes the first occurrence of the specified element from this list, if it is present. If this list does not contain the element, it is unchanged. More formally, removes the element with the lowest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))` (if such an element exists). Returns `true` if this list contained the specified element (or equivalently, if this list changed as a result of the call).

**Specified by:**

`remove` in interface `Collection<E>`

**Specified by:**

`remove` in interface `Deque<E>`

**Specified by:**

`remove` in interface `List<E>`

**Overrides:**

`remove` in class `AbstractCollection<E>`

**Parameters:**

`o` - element to be removed from this list, if present

**Returns:**

`true` if this list contained the specified element

## addAll

```
public boolean addAll(Collection<? extends E> c)
```

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. The behavior of this operation is undefined if the specified collection is modified while the operation is in progress. (Note that this will occur if the specified collection is this list, and it's nonempty.)

**Specified by:**

`addAll` in interface `Collection<E>`

**Specified by:**

`addAll` in interface `List<E>`

**Overrides:**

`addAll` in class `AbstractCollection<E>`

**Parameters:**

`c` - collection containing elements to be added to this list

**Returns:**

`true` if this list changed as a result of the call

**Throws:**

`NullPointerException` - if the specified collection is null

**See Also:**

`AbstractCollection.add(Object)`

## addAll

```
public boolean addAll(int index,
              Collection<? extends E> c)
```

Inserts all of the elements in the specified collection into this list, starting at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in the list in the order that they are returned by the specified collection's iterator.

**Specified by:**

`addAll` in interface `List<E>`

**Overrides:**

`addAll` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - index at which to insert the first element from the specified collection

`c` - collection containing elements to be added to this list

**Returns:**

`true` if this list changed as a result of the call

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

`NullPointerException` - if the specified collection is null

## clear

```
public void clear()
```

Removes all of the elements from this list. The list will be empty after this call returns.

**Specified by:**

`clear` in interface `Collection<E>`

**Specified by:**

`clear` in interface `List<E>`

**Overrides:**

`clear` in class `AbstractList<E>`

## get

```
public E get(int index)
```

Returns the element at the specified position in this list.

**Specified by:**

`get` in interface `List<E>`

**Overrides:**

`get` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - index of the element to return

**Returns:**

the element at the specified position in this list

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

## set

```
public E set(int index,
    E element)
```

Replaces the element at the specified position in this list with the specified element.

**Specified by:**

`set` in interface `List<E>`

**Overrides:**

`set` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - index of the element to replace

`element` - element to be stored at the specified position

**Returns:**

the element previously at the specified position

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

## add

```
public void add(int index,
       E element)
```

Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

**Specified by:**

`add` in interface `List<E>`

**Overrides:**

`add` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - index at which the specified element is to be inserted

`element` - element to be inserted

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

## remove

```
public E remove(int index)
```

Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list.

**Specified by:**

`remove` in interface `List<E>`

**Overrides:**

`remove` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - the index of the element to be removed

**Returns:**

the element previously at the specified position

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

## indexOf

```
public int indexOf(Object o)
```

Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the lowest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))`, or -1 if there is no such index.

**Specified by:**

`indexOf` in interface `List<E>`

**Overrides:**

`indexOf` in class `AbstractList<E>`

**Parameters:**

`o` - element to search for

**Returns:**

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

## lastIndexOf

```
public int lastIndexOf(Object o)
```

Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. More formally, returns the highest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))`, or -1 if there is no such index.

**Specified by:**

`lastIndexOf` in interface `List<E>`

**Overrides:**

`lastIndexOf` in class `AbstractList<E>`

**Parameters:**

o - element to search for

**Returns:**

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

## peek

```
public E peek()
```

Retrieves, but does not remove, the head (first element) of this list.

**Specified by:**

`peek` in interface `Deque<E>`

**Specified by:**

`peek` in interface `Queue<E>`

**Returns:**

the head of this list, or `null` if this list is empty

**Since:**

1.5

## element

```
public E element()
```

Retrieves, but does not remove, the head (first element) of this list.

**Specified by:**

`element` in interface `Deque<E>`

**Specified by:**

`element` in interface `Queue<E>`

**Returns:**

the head of this list

**Throws:**

`NoSuchElementException` - if this list is empty

**Since:**

1.5

## poll

```
public E poll()
```

Retrieves and removes the head (first element) of this list.

**Specified by:**

`poll` in interface `Deque<E>`

**Specified by:**

`poll` in interface `Queue<E>`

**Returns:**

the head of this list, or `null` if this list is empty

**Since:**

1.5

## remove

```
public E remove()
```

Retrieves and removes the head (first element) of this list.

**Specified by:**

`remove` in interface `Deque<E>`

**Specified by:**

`remove` in interface `Queue<E>`

**Returns:**

the head of this list

**Throws:**

`NoSuchElementException` - if this list is empty

**Since:**

1.5

## offer

```
public boolean offer(E e)
```

Adds the specified element as the tail (last element) of this list.

**Specified by:**

`offer` in interface `Deque<E>`

**Specified by:**

`offer` in interface `Queue<E>`

**Parameters:**

`e` - the element to add

**Returns:**

`true` (as specified by `Queue.offer(E)`)

**Since:**

1.5

## offerFirst

```
public boolean offerFirst(E e)
```

Inserts the specified element at the front of this list.

**Specified by:**

`offerFirst` in interface `Deque<E>`

**Parameters:**

`e` - the element to insert

**Returns:**

`true` (as specified by `Deque.offerFirst(E)`)

**Since:**

1.6

## offerLast

```
public boolean offerLast(E e)
```

Inserts the specified element at the end of this list.

**Specified by:**

`offerLast` in interface `Deque<E>`

**Parameters:**

`e` - the element to insert

**Returns:**

`true` (as specified by `Deque.offerLast(E)`)

**Since:**

1.6

## peekFirst

```
public E peekFirst()
```

Retrieves, but does not remove, the first element of this list, or returns `null` if this list is empty.

**Specified by:**

`peekFirst` in interface `Deque<E>`

**Returns:**

the first element of this list, or `null` if this list is empty

**Since:**

1.6

## peekLast

```
public E peekLast()
```

Retrieves, but does not remove, the last element of this list, or returns `null` if this list is empty.

**Specified by:**

`peekLast` in interface `Deque<E>`

**Returns:**

the last element of this list, or `null` if this list is empty

**Since:**

1.6

## pollFirst

`public E pollFirst()`

Retrieves and removes the first element of this list, or returns `null` if this list is empty.

**Specified by:**

`pollFirst` in interface `Deque<E>`

**Returns:**

the first element of this list, or `null` if this list is empty

**Since:**

1.6

## pollLast

`public E pollLast()`

Retrieves and removes the last element of this list, or returns `null` if this list is empty.

**Specified by:**

`pollLast` in interface `Deque<E>`

**Returns:**

the last element of this list, or `null` if this list is empty

**Since:**

1.6

## push

`public void push(E e)`

Pushes an element onto the stack represented by this list. In other words, inserts the element at the front of this list.

This method is equivalent to `addFirst(E)`.

**Specified by:**

`push` in interface `Deque<E>`

**Parameters:**

`e` - the element to push

**Since:**

1.6

## pop

`public E pop()`

Pops an element from the stack represented by this list. In other words, removes and returns the first element of this list.

This method is equivalent to `removeFirst()`.

**Specified by:**

`pop` in interface `Deque<E>`

**Returns:**

the element at the front of this list (which is the top of the stack represented by this list)

**Throws:**

`NoSuchElementException` - if this list is empty

**Since:**

1.6

## removeFirstOccurrence

```
public boolean removeFirstOccurrence(Object o)
```

Removes the first occurrence of the specified element in this list (when traversing the list from head to tail). If the list does not contain the element, it is unchanged.

**Specified by:**

`removeFirstOccurrence` in interface `Deque<E>`

**Parameters:**

`o` - element to be removed from this list, if present

**Returns:**

`true` if the list contained the specified element

**Since:**

1.6

## removeLastOccurrence

```
public boolean removeLastOccurrence(Object o)
```

Removes the last occurrence of the specified element in this list (when traversing the list from head to tail). If the list does not contain the element, it is unchanged.

**Specified by:**

`removeLastOccurrence` in interface `Deque<E>`

**Parameters:**

`o` - element to be removed from this list, if present

**Returns:**

`true` if the list contained the specified element

**Since:**

1.6

## listIterator

```
public ListIterator<E> listIterator(int index)
```

Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Obeys the general contract of `List.listIterator(int)`.

The list-iterator is *fail-fast*: if the list is structurally modified at any time after the Iterator is created, in any way except through the list-iterator's own `remove` or `add` methods, the list-iterator will throw a `ConcurrentModificationException`. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

**Specified by:**

`listIterator` in interface `List<E>`

**Specified by:**

`listIterator` in class `AbstractSequentialList<E>`

**Parameters:**

`index` - index of the first element to be returned from the list-iterator (by a call to `next`)

**Returns:**

a ListIterator of the elements in this list (in proper sequence), starting at the specified position in the list

**Throws:**

`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index > size()`)

**See Also:**

`List.listIterator(int)`

## descendingIterator

```
public Iterator<E> descendingIterator()
```

**Description copied from interface:** `Deque`
Returns an iterator over the elements in this deque in reverse sequential order. The elements will be returned in order from last (tail) to first (head).

**Specified by:**

`descendingIterator` in interface `Deque<E>`

**Returns:**

an iterator over the elements in this deque in reverse sequence

**Since:**

1.6

## clone

```
public Object clone()
```

Returns a shallow copy of this `LinkedList`. (The elements themselves are not cloned.)

**Overrides:**

`clone` in class `Object`

**Returns:**

a shallow copy of this `LinkedList` instance

**See Also:**

`Cloneable`

## toArray

```
public Object[] toArray()
```

Returns an array containing all of the elements in this list in proper sequence (from first to last element).

The returned array will be "safe" in that no references to it are maintained by this list. (In other words, this method must allocate a new array). The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based APIs.

**Specified by:**

`toArray` in interface `Collection<E>`

**Specified by:**

`toArray` in interface `List<E>`

**Overrides:**

`toArray` in class `AbstractCollection<E>`

**Returns:**

an array containing all of the elements in this list in proper sequence

**See Also:**

`Arrays.asList(Object[])`

## toArray

```
public <T> T[] toArray(T[] a)
```

Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list.

If the list fits in the specified array with room to spare (i.e., the array has more elements than the list), the element in the array immediately following the end of the list is set to `null`. (This is useful in determining the length of the list *only* if the caller knows that the list does not contain any null elements.)

Like the `toArray()` method, this method acts as bridge between array-based and collection-based APIs. Further, this method allows precise control over the runtime type of the output array, and may, under certain circumstances, be used to save allocation costs.

Suppose `x` is a list known to contain only strings. The following code can be used to dump the list into a newly allocated array of `String`:

```
String[] y = x.toArray(new String[0]);
```

Note that `toArray(new Object[0])` is identical in function to `toArray()`.

**Specified by:**

`toArray` in interface `Collection<E>`

**Specified by:**

`toArray` in interface `List<E>`

**Overrides:**

`toArray` in class `AbstractCollection<E>`

**Parameters:**

a - the array into which the elements of the list are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

**Returns:**

an array containing the elements of the list

**Throws:**

`ArrayStoreException` - if the runtime type of the specified array is not a supertype of the runtime type of every element in this list

`NullPointerException` - if the specified array is null

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.