| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/List.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/LinkedList.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ListIterator.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/List.html)    [**NO FRAMES**](http://docs.google.com/List.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

## **java.util**

Interface List<E>

**All Superinterfaces:** [Collection](http://docs.google.com/java/util/Collection.html)<E>, [Iterable](http://docs.google.com/java/lang/Iterable.html)<E> **All Known Implementing Classes:** [AbstractList](http://docs.google.com/java/util/AbstractList.html), [AbstractSequentialList](http://docs.google.com/java/util/AbstractSequentialList.html), [ArrayList](http://docs.google.com/java/util/ArrayList.html), [AttributeList](http://docs.google.com/javax/management/AttributeList.html), [CopyOnWriteArrayList](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html), [LinkedList](http://docs.google.com/java/util/LinkedList.html), [RoleList](http://docs.google.com/javax/management/relation/RoleList.html), [RoleUnresolvedList](http://docs.google.com/javax/management/relation/RoleUnresolvedList.html), [Stack](http://docs.google.com/java/util/Stack.html), [Vector](http://docs.google.com/java/util/Vector.html)

public interface **List<E>**extends [Collection](http://docs.google.com/java/util/Collection.html)<E>

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and hashCode methods. Declarations for other inherited methods are also included here for convenience.

The List interface provides four methods for positional (indexed) access to list elements. Lists (like Java arrays) are zero based. Note that these operations may execute in time proportional to the index value for some implementations (the LinkedList class, for example). Thus, iterating over the elements in a list is typically preferable to indexing through it if the caller does not know the implementation.

The List interface provides a special iterator, called a ListIterator, that allows element insertion and replacement, and bidirectional access in addition to the normal operations that the Iterator interface provides. A method is provided to obtain a list iterator that starts at a specified position in the list.

The List interface provides two methods to search for a specified object. From a performance standpoint, these methods should be used with caution. In many implementations they will perform costly linear searches.

The List interface provides two methods to efficiently insert and remove multiple elements at an arbitrary point in the list.

Note: While it is permissible for lists to contain themselves as elements, extreme caution is advised: the equals and hashCode methods are no longer well defined on such a list.

Some list implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically NullPointerException or ClassCastException. Attempting to query the presence of an ineligible element may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible element whose completion would not result in the insertion of an ineligible element into the list may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

This interface is a member of the  [Java Collections Framework](http://docs.google.com/technotes/guides/collections/index.html).

**Since:** 1.2 **See Also:**[Collection](http://docs.google.com/java/util/Collection.html), [Set](http://docs.google.com/java/util/Set.html), [ArrayList](http://docs.google.com/java/util/ArrayList.html), [LinkedList](http://docs.google.com/java/util/LinkedList.html), [Vector](http://docs.google.com/java/util/Vector.html), [Arrays.asList(Object[])](http://docs.google.com/java/util/Arrays.html#asList(T...)), [Collections.nCopies(int, Object)](http://docs.google.com/java/util/Collections.html#nCopies(int,%20T)), [Collections.EMPTY\_LIST](http://docs.google.com/java/util/Collections.html#EMPTY_LIST), [AbstractList](http://docs.google.com/java/util/AbstractList.html), [AbstractSequentialList](http://docs.google.com/java/util/AbstractSequentialList.html)

| **Method Summary** | |
| --- | --- |
| boolean | [**add**](http://docs.google.com/java/util/List.html#add(E))([E](http://docs.google.com/java/util/List.html) e)            Appends the specified element to the end of this list (optional operation). |
| void | [**add**](http://docs.google.com/java/util/List.html#add(int,%20E))(int index, [E](http://docs.google.com/java/util/List.html) element)            Inserts the specified element at the specified position in this list (optional operation). |
| boolean | [**addAll**](http://docs.google.com/java/util/List.html#addAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/List.html)> 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 (optional operation). |
| boolean | [**addAll**](http://docs.google.com/java/util/List.html#addAll(int,%20java.util.Collection))(int index, [Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/List.html)> c)            Inserts all of the elements in the specified collection into this list at the specified position (optional operation). |
| void | [**clear**](http://docs.google.com/java/util/List.html#clear())()            Removes all of the elements from this list (optional operation). |
| boolean | [**contains**](http://docs.google.com/java/util/List.html#contains(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Returns true if this list contains the specified element. |
| boolean | [**containsAll**](http://docs.google.com/java/util/List.html#containsAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<?> c)            Returns true if this list contains all of the elements of the specified collection. |
| boolean | [**equals**](http://docs.google.com/java/util/List.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Compares the specified object with this list for equality. |
| [E](http://docs.google.com/java/util/List.html) | [**get**](http://docs.google.com/java/util/List.html#get(int))(int index)            Returns the element at the specified position in this list. |
| int | [**hashCode**](http://docs.google.com/java/util/List.html#hashCode())()            Returns the hash code value for this list. |
| int | [**indexOf**](http://docs.google.com/java/util/List.html#indexOf(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) 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. |
| boolean | [**isEmpty**](http://docs.google.com/java/util/List.html#isEmpty())()            Returns true if this list contains no elements. |
| [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/List.html)> | [**iterator**](http://docs.google.com/java/util/List.html#iterator())()            Returns an iterator over the elements in this list in proper sequence. |
| int | [**lastIndexOf**](http://docs.google.com/java/util/List.html#lastIndexOf(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) 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. |
| [ListIterator](http://docs.google.com/java/util/ListIterator.html)<[E](http://docs.google.com/java/util/List.html)> | [**listIterator**](http://docs.google.com/java/util/List.html#listIterator())()            Returns a list iterator over the elements in this list (in proper sequence). |
| [ListIterator](http://docs.google.com/java/util/ListIterator.html)<[E](http://docs.google.com/java/util/List.html)> | [**listIterator**](http://docs.google.com/java/util/List.html#listIterator(int))(int index)            Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list. |
| [E](http://docs.google.com/java/util/List.html) | [**remove**](http://docs.google.com/java/util/List.html#remove(int))(int index)            Removes the element at the specified position in this list (optional operation). |
| boolean | [**remove**](http://docs.google.com/java/util/List.html#remove(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Removes the first occurrence of the specified element from this list, if it is present (optional operation). |
| boolean | [**removeAll**](http://docs.google.com/java/util/List.html#removeAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<?> c)            Removes from this list all of its elements that are contained in the specified collection (optional operation). |
| boolean | [**retainAll**](http://docs.google.com/java/util/List.html#retainAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<?> c)            Retains only the elements in this list that are contained in the specified collection (optional operation). |
| [E](http://docs.google.com/java/util/List.html) | [**set**](http://docs.google.com/java/util/List.html#set(int,%20E))(int index, [E](http://docs.google.com/java/util/List.html) element)            Replaces the element at the specified position in this list with the specified element (optional operation). |
| int | [**size**](http://docs.google.com/java/util/List.html#size())()            Returns the number of elements in this list. |
| [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/List.html)> | [**subList**](http://docs.google.com/java/util/List.html#subList(int,%20int))(int fromIndex, int toIndex)            Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. |
| [Object](http://docs.google.com/java/lang/Object.html)[] | [**toArray**](http://docs.google.com/java/util/List.html#toArray())()            Returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| | <T> T[] | | --- | | [**toArray**](http://docs.google.com/java/util/List.html#toArray(T%5B%5D))(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. |

| **Method Detail** |
| --- |

### size

int **size**()

Returns the number of elements in this list. If this list contains more than Integer.MAX\_VALUE elements, returns Integer.MAX\_VALUE.

**Specified by:**[size](http://docs.google.com/java/util/Collection.html#size()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Returns:**the number of elements in this list

### isEmpty

boolean **isEmpty**()

Returns true if this list contains no elements.

**Specified by:**[isEmpty](http://docs.google.com/java/util/Collection.html#isEmpty()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Returns:**true if this list contains no elements

### contains

boolean **contains**([Object](http://docs.google.com/java/lang/Object.html) 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](http://docs.google.com/java/util/Collection.html#contains(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**o - element whose presence in this list is to be tested **Returns:**true if this list contains the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the type of the specified element is incompatible with this list (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements (optional)

### iterator

[Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/List.html)> **iterator**()

Returns an iterator over the elements in this list in proper sequence.

**Specified by:**[iterator](http://docs.google.com/java/util/Collection.html#iterator()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)>**Specified by:**[iterator](http://docs.google.com/java/lang/Iterable.html#iterator()) in interface [Iterable](http://docs.google.com/java/lang/Iterable.html)<[E](http://docs.google.com/java/util/List.html)> **Returns:**an iterator over the elements in this list in proper sequence

### toArray

[Object](http://docs.google.com/java/lang/Object.html)[] **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 even if this list is backed by an 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](http://docs.google.com/java/util/Collection.html#toArray()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Returns:**an array containing all of the elements in this list in proper sequence**See Also:**[Arrays.asList(Object[])](http://docs.google.com/java/util/Arrays.html#asList(T...))

### toArray

<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()](http://docs.google.com/java/util/List.html#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](http://docs.google.com/java/util/Collection.html#toArray(T%5B%5D)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**a - the array into which the elements of this 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 this list **Throws:** [ArrayStoreException](http://docs.google.com/java/lang/ArrayStoreException.html) - if the runtime type of the specified array is not a supertype of the runtime type of every element in this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified array is null

### add

boolean **add**([E](http://docs.google.com/java/util/List.html) e)

Appends the specified element to the end of this list (optional operation).

Lists that support this operation may place limitations on what elements may be added to this list. In particular, some lists will refuse to add null elements, and others will impose restrictions on the type of elements that may be added. List classes should clearly specify in their documentation any restrictions on what elements may be added.

**Specified by:**[add](http://docs.google.com/java/util/Collection.html#add(E)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**e - element to be appended to this list **Returns:**true (as specified by [Collection.add(E)](http://docs.google.com/java/util/Collection.html#add(E))) **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the add operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of this element prevents it from being added to this list

### remove

boolean **remove**([Object](http://docs.google.com/java/lang/Object.html) o)

Removes the first occurrence of the specified element from this list, if it is present (optional operation). 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](http://docs.google.com/java/util/Collection.html#remove(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**o - element to be removed from this list, if present **Returns:**true if this list contained the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the type of the specified element is incompatible with this list (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements (optional) [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the remove operation is not supported by this list

### containsAll

boolean **containsAll**([Collection](http://docs.google.com/java/util/Collection.html)<?> c)

Returns true if this list contains all of the elements of the specified collection.

**Specified by:**[containsAll](http://docs.google.com/java/util/Collection.html#containsAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**c - collection to be checked for containment in this list **Returns:**true if this list contains all of the elements of the specified collection **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the types of one or more elements in the specified collection are incompatible with this list (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection contains one or more null elements and this list does not permit null elements (optional), or if the specified collection is null**See Also:**[contains(Object)](http://docs.google.com/java/util/List.html#contains(java.lang.Object))

### addAll

boolean **addAll**([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/List.html)> 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 (optional operation). 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](http://docs.google.com/java/util/Collection.html#addAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**c - collection containing elements to be added to this list **Returns:**true if this list changed as a result of the call **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the addAll operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of the specified collection prevents it from being added to this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection contains one or more null elements and this list does not permit null elements, or if the specified collection is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of an element of the specified collection prevents it from being added to this list**See Also:**[add(Object)](http://docs.google.com/java/util/List.html#add(E))

### addAll

boolean **addAll**(int index,  
 [Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/List.html)> c)

Inserts all of the elements in the specified collection into this list at the specified position (optional operation). 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 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.)

**Parameters:**index - index at which to insert the first element from the specified collectionc - collection containing elements to be added to this list **Returns:**true if this list changed as a result of the call **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the addAll operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of the specified collection prevents it from being added to this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection contains one or more null elements and this list does not permit null elements, or if the specified collection is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of an element of the specified collection prevents it from being added to this list [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index > size())

### removeAll

boolean **removeAll**([Collection](http://docs.google.com/java/util/Collection.html)<?> c)

Removes from this list all of its elements that are contained in the specified collection (optional operation).

**Specified by:**[removeAll](http://docs.google.com/java/util/Collection.html#removeAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**c - collection containing elements to be removed from this list **Returns:**true if this list changed as a result of the call **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the removeAll operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of this list is incompatible with the specified collection (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if this list contains a null element and the specified collection does not permit null elements (optional), or if the specified collection is null**See Also:**[remove(Object)](http://docs.google.com/java/util/List.html#remove(java.lang.Object)), [contains(Object)](http://docs.google.com/java/util/List.html#contains(java.lang.Object))

### retainAll

boolean **retainAll**([Collection](http://docs.google.com/java/util/Collection.html)<?> c)

Retains only the elements in this list that are contained in the specified collection (optional operation). In other words, removes from this list all the elements that are not contained in the specified collection.

**Specified by:**[retainAll](http://docs.google.com/java/util/Collection.html#retainAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Parameters:**c - collection containing elements to be retained in this list **Returns:**true if this list changed as a result of the call **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the retainAll operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of an element of this list is incompatible with the specified collection (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if this list contains a null element and the specified collection does not permit null elements (optional), or if the specified collection is null**See Also:**[remove(Object)](http://docs.google.com/java/util/List.html#remove(java.lang.Object)), [contains(Object)](http://docs.google.com/java/util/List.html#contains(java.lang.Object))

### clear

void **clear**()

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

**Specified by:**[clear](http://docs.google.com/java/util/Collection.html#clear()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)> **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the clear operation is not supported by this list

### equals

boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) o)

Compares the specified object with this list for equality. Returns true if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are *equal*. (Two elements e1 and e2 are *equal* if (e1==null ? e2==null : e1.equals(e2)).) In other words, two lists are defined to be equal if they contain the same elements in the same order. This definition ensures that the equals method works properly across different implementations of the List interface.

**Specified by:**[equals](http://docs.google.com/java/util/Collection.html#equals(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)>**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**o - the object to be compared for equality with this list **Returns:**true if the specified object is equal to this list**See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### hashCode

int **hashCode**()

Returns the hash code value for this list. The hash code of a list is defined to be the result of the following calculation:

int hashCode = 1;  
 Iterator<E> i = list.iterator();  
 while (i.hasNext()) {  
 E obj = i.next();  
 hashCode = 31\*hashCode + (obj==null ? 0 : obj.hashCode());  
 }

This ensures that list1.equals(list2) implies that list1.hashCode()==list2.hashCode() for any two lists, list1 and list2, as required by the general contract of [Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()).

**Specified by:**[hashCode](http://docs.google.com/java/util/Collection.html#hashCode()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/List.html)>**Overrides:**[hashCode](http://docs.google.com/java/lang/Object.html#hashCode()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**the hash code value for this list**See Also:**[Object.equals(Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [equals(Object)](http://docs.google.com/java/util/List.html#equals(java.lang.Object))

### get

[E](http://docs.google.com/java/util/List.html) **get**(int index)

Returns the element at the specified position in this list.

**Parameters:**index - index of the element to return **Returns:**the element at the specified position in this list **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index >= size())

### set

[E](http://docs.google.com/java/util/List.html) **set**(int index,  
 [E](http://docs.google.com/java/util/List.html) element)

Replaces the element at the specified position in this list with the specified element (optional operation).

**Parameters:**index - index of the element to replaceelement - element to be stored at the specified position **Returns:**the element previously at the specified position **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the set operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this list [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index >= size())

### add

void **add**(int index,  
 [E](http://docs.google.com/java/util/List.html) element)

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

**Parameters:**index - index at which the specified element is to be insertedelement - element to be inserted **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the add operation is not supported by this list [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the class of the specified element prevents it from being added to this list [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if some property of the specified element prevents it from being added to this list [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index > size())

### remove

[E](http://docs.google.com/java/util/List.html) **remove**(int index)

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

**Parameters:**index - the index of the element to be removed **Returns:**the element previously at the specified position **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if the remove operation is not supported by this list [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index >= size())

### indexOf

int **indexOf**([Object](http://docs.google.com/java/lang/Object.html) 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.

**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 **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the type of the specified element is incompatible with this list (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements (optional)

### lastIndexOf

int **lastIndexOf**([Object](http://docs.google.com/java/lang/Object.html) 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.

**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 **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the type of the specified element is incompatible with this list (optional) [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null and this list does not permit null elements (optional)

### listIterator

[ListIterator](http://docs.google.com/java/util/ListIterator.html)<[E](http://docs.google.com/java/util/List.html)> **listIterator**()

Returns a list iterator over the elements in this list (in proper sequence).

**Returns:**a list iterator over the elements in this list (in proper sequence)

### listIterator

[ListIterator](http://docs.google.com/java/util/ListIterator.html)<[E](http://docs.google.com/java/util/List.html)> **listIterator**(int index)

Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list. The specified index indicates the first element that would be returned by an initial call to [next](http://docs.google.com/java/util/ListIterator.html#next()). An initial call to [previous](http://docs.google.com/java/util/ListIterator.html#previous()) would return the element with the specified index minus one.

**Parameters:**index - index of first element to be returned from the list iterator (by a call to the next method) **Returns:**a list iterator of the elements in this list (in proper sequence), starting at the specified position in this list **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - if the index is out of range (index < 0 || index > size())

### subList

[List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/List.html)> **subList**(int fromIndex,  
 int toIndex)

Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. (If fromIndex and toIndex are equal, the returned list is empty.) The returned list is backed by this list, so non-structural changes in the returned list are reflected in this list, and vice-versa. The returned list supports all of the optional list operations supported by this list.

This method eliminates the need for explicit range operations (of the sort that commonly exist for arrays). Any operation that expects a list can be used as a range operation by passing a subList view instead of a whole list. For example, the following idiom removes a range of elements from a list:

list.subList(from, to).clear();

Similar idioms may be constructed for indexOf and lastIndexOf, and all of the algorithms in the Collections class can be applied to a subList.

The semantics of the list returned by this method become undefined if the backing list (i.e., this list) is *structurally modified* in any way other than via the returned list. (Structural modifications are those that change the size of this list, or otherwise perturb it in such a fashion that iterations in progress may yield incorrect results.)

**Parameters:**fromIndex - low endpoint (inclusive) of the subListtoIndex - high endpoint (exclusive) of the subList **Returns:**a view of the specified range within this list **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - for an illegal endpoint index value (fromIndex < 0 || toIndex > size || fromIndex > toIndex)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/List.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/LinkedList.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ListIterator.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/List.html)    [**NO FRAMES**](http://docs.google.com/List.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).