| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AbstractList.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/AbstractCollection.html)   [**NEXT CLASS**](http://docs.google.com/java/util/AbstractMap.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/AbstractList.html)    [**NO FRAMES**](http://docs.google.com/AbstractList.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#2s8eyo1) | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

## **java.util**

Class AbstractList<E>

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.util.AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<E>  
 **java.util.AbstractList<E>**

**All Implemented Interfaces:** [Iterable](http://docs.google.com/java/lang/Iterable.html)<E>, [Collection](http://docs.google.com/java/util/Collection.html)<E>, [List](http://docs.google.com/java/util/List.html)<E> **Direct Known Subclasses:** [AbstractSequentialList](http://docs.google.com/java/util/AbstractSequentialList.html), [ArrayList](http://docs.google.com/java/util/ArrayList.html), [Vector](http://docs.google.com/java/util/Vector.html)

public abstract class **AbstractList<E>**extends [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<E>implements [List](http://docs.google.com/java/util/List.html)<E>

This class provides a skeletal implementation of the [List](http://docs.google.com/java/util/List.html) interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array). For sequential access data (such as a linked list), [AbstractSequentialList](http://docs.google.com/java/util/AbstractSequentialList.html) should be used in preference to this class.

To implement an unmodifiable list, the programmer needs only to extend this class and provide implementations for the [get(int)](http://docs.google.com/java/util/AbstractList.html#get(int)) and [size()](http://docs.google.com/java/util/List.html#size()) methods.

To implement a modifiable list, the programmer must additionally override the [set(int, E)](http://docs.google.com/java/util/AbstractList.html#set(int,%20E)) method (which otherwise throws an UnsupportedOperationException). If the list is variable-size the programmer must additionally override the [add(int, E)](http://docs.google.com/java/util/AbstractList.html#add(int,%20E)) and [remove(int)](http://docs.google.com/java/util/AbstractList.html#remove(int)) methods.

The programmer should generally provide a void (no argument) and collection constructor, as per the recommendation in the [Collection](http://docs.google.com/java/util/Collection.html) interface specification.

Unlike the other abstract collection implementations, the programmer does *not* have to provide an iterator implementation; the iterator and list iterator are implemented by this class, on top of the "random access" methods: [get(int)](http://docs.google.com/java/util/AbstractList.html#get(int)), [set(int, E)](http://docs.google.com/java/util/AbstractList.html#set(int,%20E)), [add(int, E)](http://docs.google.com/java/util/AbstractList.html#add(int,%20E)) and [remove(int)](http://docs.google.com/java/util/AbstractList.html#remove(int)).

The documentation for each non-abstract method in this class describes its implementation in detail. Each of these methods may be overridden if the collection being implemented admits a more efficient implementation.

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

**Since:** 1.2

| **Field Summary** | |
| --- | --- |
| protected  int | [**modCount**](http://docs.google.com/java/util/AbstractList.html#modCount)            The number of times this list has been *structurally modified*. |

| **Constructor Summary** | |
| --- | --- |
| protected | [**AbstractList**](http://docs.google.com/java/util/AbstractList.html#AbstractList())()            Sole constructor. |

| **Method Summary** | |
| --- | --- |
| boolean | [**add**](http://docs.google.com/java/util/AbstractList.html#add(E))([E](http://docs.google.com/java/util/AbstractList.html) e)            Appends the specified element to the end of this list (optional operation). |
| void | [**add**](http://docs.google.com/java/util/AbstractList.html#add(int,%20E))(int index, [E](http://docs.google.com/java/util/AbstractList.html) element)            Inserts the specified element at the specified position in this list (optional operation). |
| boolean | [**addAll**](http://docs.google.com/java/util/AbstractList.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/AbstractList.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/AbstractList.html#clear())()            Removes all of the elements from this list (optional operation). |
| boolean | [**equals**](http://docs.google.com/java/util/AbstractList.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Compares the specified object with this list for equality. |
| abstract  [E](http://docs.google.com/java/util/AbstractList.html) | [**get**](http://docs.google.com/java/util/AbstractList.html#get(int))(int index)            Returns the element at the specified position in this list. |
| int | [**hashCode**](http://docs.google.com/java/util/AbstractList.html#hashCode())()            Returns the hash code value for this list. |
| int | [**indexOf**](http://docs.google.com/java/util/AbstractList.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. |
| [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/AbstractList.html)> | [**iterator**](http://docs.google.com/java/util/AbstractList.html#iterator())()            Returns an iterator over the elements in this list in proper sequence. |
| int | [**lastIndexOf**](http://docs.google.com/java/util/AbstractList.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/AbstractList.html)> | [**listIterator**](http://docs.google.com/java/util/AbstractList.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/AbstractList.html)> | [**listIterator**](http://docs.google.com/java/util/AbstractList.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/AbstractList.html) | [**remove**](http://docs.google.com/java/util/AbstractList.html#remove(int))(int index)            Removes the element at the specified position in this list (optional operation). |
| protected  void | [**removeRange**](http://docs.google.com/java/util/AbstractList.html#removeRange(int,%20int))(int fromIndex, int toIndex)            Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| [E](http://docs.google.com/java/util/AbstractList.html) | [**set**](http://docs.google.com/java/util/AbstractList.html#set(int,%20E))(int index, [E](http://docs.google.com/java/util/AbstractList.html) element)            Replaces the element at the specified position in this list with the specified element (optional operation). |
| [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> | [**subList**](http://docs.google.com/java/util/AbstractList.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. |

| **Methods inherited from class java.util.**[**AbstractCollection**](http://docs.google.com/java/util/AbstractCollection.html) |
| --- |
| [addAll](http://docs.google.com/java/util/AbstractCollection.html#addAll(java.util.Collection)), [contains](http://docs.google.com/java/util/AbstractCollection.html#contains(java.lang.Object)), [containsAll](http://docs.google.com/java/util/AbstractCollection.html#containsAll(java.util.Collection)), [isEmpty](http://docs.google.com/java/util/AbstractCollection.html#isEmpty()), [remove](http://docs.google.com/java/util/AbstractCollection.html#remove(java.lang.Object)), [removeAll](http://docs.google.com/java/util/AbstractCollection.html#removeAll(java.util.Collection)), [retainAll](http://docs.google.com/java/util/AbstractCollection.html#retainAll(java.util.Collection)), [size](http://docs.google.com/java/util/AbstractCollection.html#size()), [toArray](http://docs.google.com/java/util/AbstractCollection.html#toArray()), [toArray](http://docs.google.com/java/util/AbstractCollection.html#toArray(T%5B%5D)), [toString](http://docs.google.com/java/util/AbstractCollection.html#toString()) |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Methods inherited from interface java.util.**[**List**](http://docs.google.com/java/util/List.html) |
| --- |
| [addAll](http://docs.google.com/java/util/List.html#addAll(java.util.Collection)), [contains](http://docs.google.com/java/util/List.html#contains(java.lang.Object)), [containsAll](http://docs.google.com/java/util/List.html#containsAll(java.util.Collection)), [isEmpty](http://docs.google.com/java/util/List.html#isEmpty()), [remove](http://docs.google.com/java/util/List.html#remove(java.lang.Object)), [removeAll](http://docs.google.com/java/util/List.html#removeAll(java.util.Collection)), [retainAll](http://docs.google.com/java/util/List.html#retainAll(java.util.Collection)), [size](http://docs.google.com/java/util/List.html#size()), [toArray](http://docs.google.com/java/util/List.html#toArray()), [toArray](http://docs.google.com/java/util/List.html#toArray(T%5B%5D)) |

| **Field Detail** |
| --- |

### modCount

protected transient int **modCount**

The number of times this list has been *structurally modified*. Structural modifications are those that change the size of the list, or otherwise perturb it in such a fashion that iterations in progress may yield incorrect results.

This field is used by the iterator and list iterator implementation returned by the iterator and listIterator methods. If the value of this field changes unexpectedly, the iterator (or list iterator) will throw a ConcurrentModificationException in response to the next, remove, previous, set or add operations. This provides *fail-fast* behavior, rather than non-deterministic behavior in the face of concurrent modification during iteration.

**Use of this field by subclasses is optional.** If a subclass wishes to provide fail-fast iterators (and list iterators), then it merely has to increment this field in its add(int, E) and remove(int) methods (and any other methods that it overrides that result in structural modifications to the list). A single call to add(int, E) or remove(int) must add no more than one to this field, or the iterators (and list iterators) will throw bogus ConcurrentModificationExceptions. If an implementation does not wish to provide fail-fast iterators, this field may be ignored.

| **Constructor Detail** |
| --- |

### AbstractList

protected **AbstractList**()

Sole constructor. (For invocation by subclass constructors, typically implicit.)

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

### add

public boolean **add**([E](http://docs.google.com/java/util/AbstractList.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.

This implementation calls add(size(), e).

Note that this implementation throws an UnsupportedOperationException unless [add(int, E)](http://docs.google.com/java/util/AbstractList.html#add(int,%20E)) is overridden.

**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/AbstractList.html)>**Specified by:**[add](http://docs.google.com/java/util/List.html#add(E)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)>**Overrides:**[add](http://docs.google.com/java/util/AbstractCollection.html#add(E)) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/AbstractList.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

### get

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

Returns the element at the specified position in this list.

**Specified by:**[get](http://docs.google.com/java/util/List.html#get(int)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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

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

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

This implementation always throws an UnsupportedOperationException.

**Specified by:**[set](http://docs.google.com/java/util/List.html#set(int,%20E)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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

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

This implementation always throws an UnsupportedOperationException.

**Specified by:**[add](http://docs.google.com/java/util/List.html#add(int,%20E)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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

public [E](http://docs.google.com/java/util/AbstractList.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.

This implementation always throws an UnsupportedOperationException.

**Specified by:**[remove](http://docs.google.com/java/util/List.html#remove(int)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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

public 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.

This implementation first gets a list iterator (with listIterator()). Then, it iterates over the list until the specified element is found or the end of the list is reached.

**Specified by:**[indexOf](http://docs.google.com/java/util/List.html#indexOf(java.lang.Object)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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

public 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.

This implementation first gets a list iterator that points to the end of the list (with listIterator(size())). Then, it iterates backwards over the list until the specified element is found, or the beginning of the list is reached.

**Specified by:**[lastIndexOf](http://docs.google.com/java/util/List.html#lastIndexOf(java.lang.Object)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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)

### clear

public void **clear**()

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

This implementation calls removeRange(0, size()).

Note that this implementation throws an UnsupportedOperationException unless remove(int index) or removeRange(int fromIndex, int toIndex) is overridden.

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

### addAll

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

This implementation gets an iterator over the specified collection and iterates over it, inserting the elements obtained from the iterator into this list at the appropriate position, one at a time, using add(int, E). Many implementations will override this method for efficiency.

Note that this implementation throws an UnsupportedOperationException unless [add(int, E)](http://docs.google.com/java/util/AbstractList.html#add(int,%20E)) is overridden.

**Specified by:**[addAll](http://docs.google.com/java/util/List.html#addAll(int,%20java.util.Collection)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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())

### iterator

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

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

This implementation returns a straightforward implementation of the iterator interface, relying on the backing list's size(), get(int), and remove(int) methods.

Note that the iterator returned by this method will throw an UnsupportedOperationException in response to its remove method unless the list's remove(int) method is overridden.

This implementation can be made to throw runtime exceptions in the face of concurrent modification, as described in the specification for the (protected) modCount field.

**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/AbstractList.html)>**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/AbstractList.html)>**Specified by:**[iterator](http://docs.google.com/java/util/List.html#iterator()) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)>**Specified by:**[iterator](http://docs.google.com/java/util/AbstractCollection.html#iterator()) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **Returns:**an iterator over the elements in this list in proper sequence**See Also:**[modCount](http://docs.google.com/java/util/AbstractList.html#modCount)

### listIterator

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

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

This implementation returns listIterator(0).

**Specified by:**[listIterator](http://docs.google.com/java/util/List.html#listIterator()) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **Returns:**a list iterator over the elements in this list (in proper sequence)**See Also:**[listIterator(int)](http://docs.google.com/java/util/AbstractList.html#listIterator(int))

### listIterator

public [ListIterator](http://docs.google.com/java/util/ListIterator.html)<[E](http://docs.google.com/java/util/AbstractList.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.

This implementation returns a straightforward implementation of the ListIterator interface that extends the implementation of the Iterator interface returned by the iterator() method. The ListIterator implementation relies on the backing list's get(int), set(int, E), add(int, E) and remove(int) methods.

Note that the list iterator returned by this implementation will throw an UnsupportedOperationException in response to its remove, set and add methods unless the list's remove(int), set(int, E), and add(int, E) methods are overridden.

This implementation can be made to throw runtime exceptions in the face of concurrent modification, as described in the specification for the (protected) modCount field.

**Specified by:**[listIterator](http://docs.google.com/java/util/List.html#listIterator(int)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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())**See Also:**[modCount](http://docs.google.com/java/util/AbstractList.html#modCount)

### subList

public [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.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.)

This implementation returns a list that subclasses AbstractList. The subclass stores, in private fields, the offset of the subList within the backing list, the size of the subList (which can change over its lifetime), and the expected modCount value of the backing list. There are two variants of the subclass, one of which implements RandomAccess. If this list implements RandomAccess the returned list will be an instance of the subclass that implements RandomAccess.

The subclass's set(int, E), get(int), add(int, E), remove(int), addAll(int, Collection) and removeRange(int, int) methods all delegate to the corresponding methods on the backing abstract list, after bounds-checking the index and adjusting for the offset. The addAll(Collection c) method merely returns addAll(size, c).

The listIterator(int) method returns a "wrapper object" over a list iterator on the backing list, which is created with the corresponding method on the backing list. The iterator method merely returns listIterator(), and the size method merely returns the subclass's size field.

All methods first check to see if the actual modCount of the backing list is equal to its expected value, and throw a ConcurrentModificationException if it is not.

**Specified by:**[subList](http://docs.google.com/java/util/List.html#subList(int,%20int)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.html)> **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) - endpoint index value out of range (fromIndex < 0 || toIndex > size) [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the endpoint indices are out of order (fromIndex > toIndex)

### equals

public 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 implementation first checks if the specified object is this list. If so, it returns true; if not, it checks if the specified object is a list. If not, it returns false; if so, it iterates over both lists, comparing corresponding pairs of elements. If any comparison returns false, this method returns false. If either iterator runs out of elements before the other it returns false (as the lists are of unequal length); otherwise it returns true when the iterations complete.

**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/AbstractList.html)>**Specified by:**[equals](http://docs.google.com/java/util/List.html#equals(java.lang.Object)) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.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

public int **hashCode**()

Returns the hash code value for this list.

This implementation uses exactly the code that is used to define the list hash function in the documentation for the [List.hashCode()](http://docs.google.com/java/util/List.html#hashCode()) method.

**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/AbstractList.html)>**Specified by:**[hashCode](http://docs.google.com/java/util/List.html#hashCode()) in interface [List](http://docs.google.com/java/util/List.html)<[E](http://docs.google.com/java/util/AbstractList.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(java.lang.Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### removeRange

protected void **removeRange**(int fromIndex,  
 int toIndex)

Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. Shifts any succeeding elements to the left (reduces their index). This call shortens the ArrayList by (toIndex - fromIndex) elements. (If toIndex==fromIndex, this operation has no effect.)

This method is called by the clear operation on this list and its subLists. Overriding this method to take advantage of the internals of the list implementation can *substantially* improve the performance of the clear operation on this list and its subLists.

This implementation gets a list iterator positioned before fromIndex, and repeatedly calls ListIterator.next followed by ListIterator.remove until the entire range has been removed. **Note: if ListIterator.remove requires linear time, this implementation requires quadratic time.**

**Parameters:**fromIndex - index of first element to be removedtoIndex - index after last element to be removed

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/AbstractList.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/AbstractCollection.html)   [**NEXT CLASS**](http://docs.google.com/java/util/AbstractMap.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/AbstractList.html)    [**NO FRAMES**](http://docs.google.com/AbstractList.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#2s8eyo1) | [CONSTR](#3rdcrjn) | [METHOD](#lnxbz9) |

[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).