java.util

Class ArrayList

```
java.lang.Object
  └java.util.AbstractCollection
      └java.util.AbstractList
          └java.util.ArrayList
```

All Implemented Interfaces:

Cloneable, Collection, List, RandomAccess, Serializable

---

public class **ArrayList**
extends AbstractList
implements List, RandomAccess, Cloneable, Serializable

Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)

The `size`, `isEmpty`, `get`, `set`, `iterator`, and `listIterator` operations run in constant time. The `add` operation runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the `LinkedList` implementation.

Each `ArrayList` instance has a *capacity*. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

An application can increase the capacity of an `ArrayList` instance before adding a large number of elements using the `ensureCapacity` operation. This may reduce the amount of incremental reallocation.

Note that this implementation is not synchronized. If multiple threads access an `ArrayList` instance 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, or explicitly resizes the backing array; 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 ArrayList(...));
```

The iterators returned by this class's `iterator` and `listIterator` methods are *fail-fast*: if 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:

`Collection`, `List`, `LinkedList`, `Vector`, `Collections.synchronizedList(List)`, Serialized Form

Field Summary

Fields inherited from class java.util.AbstractList
`modCount`

Constructor Summary

`ArrayList``()` Constructs an empty list with an initial capacity of ten.
`ArrayList``(Collection c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayList``(int initialCapacity)` Constructs an empty list with the specified initial capacity.

Method Summary

void	**`add`**`(int index, Object element)` Inserts the specified element at the specified position in this list.
boolean	**`add`**`(Object o)` Appends the specified element to the end of this list.
boolean	**`addAll`**`(Collection 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.
boolean	**`addAll`**`(int index, Collection c)` Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void	**`clear`**`()` Removes all of the elements from this list.
Object	**`clone`**`()` Returns a shallow copy of this `ArrayList` instance.
boolean	**`contains`**`(Object elem)` Returns `true` if this list contains the specified element.
void	**`ensureCapacity`**`(int minCapacity)` Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

Object	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method.
boolean	**isEmpty**() Tests if this list has no elements.
int	**lastIndexOf**(Object elem) Returns the index of the last occurrence of the specified object in this list.
Object	**remove**(int index) Removes the element at the specified position in this list.
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in the correct order.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

Methods inherited from class java.util.AbstractList
equals, hashCode, iterator, listIterator, listIterator, subList

Methods inherited from class java.util.AbstractCollection
containsAll, remove, 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, iterator, listIterator, listIterator, remove, removeAll, retainAll, subList

Constructor Detail

ArrayList

```
public ArrayList(int initialCapacity)
```

Constructs an empty list with the specified initial capacity.

Parameters:

`initialCapacity` - the initial capacity of the list.

Throws:

`IllegalArgumentException` - if the specified initial capacity is negative

ArrayList

```
public ArrayList()
```

Constructs an empty list with an initial capacity of ten.

ArrayList

```
public ArrayList(Collection c)
```

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. The `ArrayList` instance has an initial capacity of 110% the size of the specified collection.

Parameters:

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

Throws:

`NullPointerException` - if the specified collection is null.

Method Detail

trimToSize

```
public void trimToSize()
```

Trims the capacity of this `ArrayList` instance to be the list's current size. An application can use this operation to minimize the storage of an `ArrayList` instance.

---

ensureCapacity

```
public void ensureCapacity(int minCapacity)
```

Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

Parameters:

`minCapacity` - the desired minimum capacity.

---

size

```
public int size()
```

Returns the number of elements in this list.

Specified by:

`size` in interface `List`

Specified by:

`size` in class `AbstractCollection`

Returns:

the number of elements in this list.

---

isEmpty

```
public boolean isEmpty()
```

Tests if this list has no elements.

Specified by:

`isEmpty` in interface `List`

Overrides:

`isEmpty` in class `AbstractCollection`

Returns:

`true` if this list has no elements; `false` otherwise.

---

contains

```
public boolean contains(Object elem)
```

Returns `true` if this list contains the specified element.

Specified by:

`contains` in interface `List`

Overrides:

`contains` in class `AbstractCollection`

Parameters:

`elem` - element whose presence in this List is to be tested.

Returns:

`true` if the specified element is present; `false` otherwise.

---

indexOf

```
public int indexOf(Object elem)
```

Searches for the first occurence of the given argument, testing for equality using the `equals` method.

Specified by:

`indexOf` in interface `List`

Overrides:

`indexOf` in class `AbstractList`

Parameters:

`elem` - an object.

Returns:

the index of the first occurrence of the argument in this list; returns `-1` if the object is not found.

See Also:

`Object.equals(Object)`

---

lastIndexOf

```
public int lastIndexOf(Object elem)
```

Returns the index of the last occurrence of the specified object in this list.

Specified by:
lastIndexOf in interface List
Overrides:
lastIndexOf in class AbstractList
Parameters:
elem - the desired element.
Returns:
the index of the last occurrence of the specified object in this list; returns -1 if the object is not found.

---

clone

```
public Object clone()
```

Returns a shallow copy of this ArrayList instance. (The elements themselves are not copied.)

Overrides:
clone in class Object
Returns:
a clone of this ArrayList instance.
See Also:
Cloneable

---

toArray

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

Returns an array containing all of the elements in this list in the correct order.

Specified by:
toArray in interface List
Overrides:
toArray in class AbstractCollection
Returns:
an array containing all of the elements in this list in the correct order.

---

toArray

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

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

Specified by:
`toArray` in interface `List`
Overrides:
`toArray` in class `AbstractCollection`
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 a is not a supertype of the runtime type of every element in this list.

---

get

```
public Object get(int index)
```

Returns the element at the specified position in this list.

Specified by:
`get` in interface `List`
Specified by:
`get` in class `AbstractList`
Parameters:
`index` - index of element to return.
Returns:
the element at the specified position in this list.
Throws:
`IndexOutOfBoundsException` - if index is out of range `(index < 0 || index >= size())`.

---

set

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

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

Specified by:
`set` in interface `List`
Overrides:
`set` in class `AbstractList`
Parameters:
`index` - index of element to replace.
`element` - element to be stored at the specified position.
Returns:
the element previously at the specified position.
Throws:
`IndexOutOfBoundsException` - if index out of range `(index < 0 || index >= size())`.

---

add

```
public boolean add(Object o)
```

Appends the specified element to the end of this list.

Specified by:
add in interface List
Overrides:
add in class AbstractList
Parameters:
o - element to be appended to this list.
Returns:
true (as per the general contract of Collection.add).

---

add

```
public void add(int index,
                Object 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
Overrides:
add in class AbstractList
Parameters:
index - index at which the specified element is to be inserted.
element - element to be inserted.
Throws:
IndexOutOfBoundsException - if index is out of range (index < 0 || index > size()).

---

remove

```
public Object 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).

Specified by:
remove in interface List
Overrides:
remove in class AbstractList
Parameters:
index - the index of the element to removed.
Returns:
the element that was removed from the list.
Throws:
IndexOutOfBoundsException - if index out of range (index < 0 || index >= size()).

---

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 List
Overrides:
clear in class AbstractList

---

addAll

```
public boolean addAll(Collection 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. (This implies that the behavior of this call is undefined if the specified Collection is this list, and this list is nonempty.)

Specified by:
addAll in interface List
Overrides:
addAll in class AbstractCollection
Parameters:
c - the elements to be inserted into 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 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
Overrides:
addAll in class AbstractList
Parameters:
index - index at which to insert first element from the specified collection.
c - elements to be inserted into this list.
Returns:
true if this list changed as a result of the call.
Throws:
IndexOutOfBoundsException - if index out of range (index < 0 || index > size()).
NullPointerException - if the specified Collection is null.

---

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 list by `(toIndex - fromIndex)` elements. (If `toIndex==fromIndex`, this operation has no effect.)

Overrides:

`removeRange` in class `AbstractList`

Parameters:

`fromIndex` - index of first element to be removed.
`toIndex` - index after last element to be removed.
