java.util

Class Vector<E>

```
java.lang.Object
  └ java.util.AbstractCollection<E>
      └ java.util.AbstractList<E>
          └ java.util.Vector<E>
```

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:
Stack

```
public class Vector<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

The `Vector` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Vector` can grow or shrink as needed to accommodate adding and removing items after the `Vector` has been created.

Each vector tries to optimize storage management by maintaining a `capacity` and a `capacityIncrement`. The `capacity` is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of `capacityIncrement`. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

The Iterators returned by Vector's iterator and listIterator methods are *fail-fast*: if the Vector 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. The Enumerations returned by Vector's elements method are *not* fail-fast.

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.*

As of the Java 2 platform v1.2, this class was retrofitted to implement the `List` interface, making it a member of the Java Collections Framework. Unlike the new collection implementations, `Vector` is synchronized.

Since:
JDK1.0

See Also:

Collection, List, ArrayList, LinkedList, Serialized Form

Field Summary

protected int	**capacityIncrement** The amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity.
protected int	**elementCount** The number of valid components in this `Vector` object.
protected Object[]	**elementData** The array buffer into which the components of the vector are stored.

Fields inherited from class java.util.AbstractList
modCount

Constructor Summary

Vector() Constructs an empty vector so that its internal data array has size `10` and its standard capacity increment is zero.
Vector(Collection<? extends E> c) Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.
Vector(int initialCapacity) Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.
Vector(int initialCapacity, int capacityIncrement) Constructs an empty vector with the specified initial capacity and capacity increment.

Method Summary

boolean	**add**(E e) Appends the specified element to the end of this Vector.
void	**add**(int index, E element) Inserts the specified element at the specified position in this Vector.
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified Collection into this Vector at the specified position.
void	**addElement**(E obj) Adds the specified component to the end of this vector, increasing its size by one.
int	**capacity**() Returns the current capacity of this vector.

void	`clear()` Removes all of the elements from this Vector.
Object	`clone()` Returns a clone of this vector.
boolean	`contains(Object o)` Returns `true` if this vector contains the specified element.
boolean	`containsAll(Collection<?> c)` Returns true if this Vector contains all of the elements in the specified Collection.
void	`copyInto(Object[] anArray)` Copies the components of this vector into the specified array.
E	`elementAt(int index)` Returns the component at the specified index.
Enumeration<E>	`elements()` Returns an enumeration of the components of this vector.
void	`ensureCapacity(int minCapacity)` Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
boolean	`equals(Object o)` Compares the specified Object with this Vector for equality.
E	`firstElement()` Returns the first component (the item at index `0`) of this vector.
E	`get(int index)` Returns the element at the specified position in this Vector.
int	`hashCode()` Returns the hash code value for this Vector.
int	`indexOf(Object o)` Returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int	`indexOf(Object o, int index)` Returns the index of the first occurrence of the specified element in this vector, searching forwards from `index`, or returns -1 if the element is not found.
void	`insertElementAt(E obj, int index)` Inserts the specified object as a component in this vector at the specified `index`.
boolean	`isEmpty()` Tests if this vector has no components.
E	`lastElement()` Returns the last component of the vector.
int	`lastIndexOf(Object o)` Returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
int	`lastIndexOf(Object o, int index)` Returns the index of the last occurrence of the specified element in this vector, searching backwards from `index`, or returns -1 if the element is not found.
E	`remove(int index)`

	Removes the element at the specified position in this Vector.
`boolean`	**`remove`**`(Object o)` Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
`boolean`	**`removeAll`**`(Collection<?> c)` Removes from this Vector all of its elements that are contained in the specified Collection.
`void`	**`removeAllElements`**`()` Removes all components from this vector and sets its size to zero.
`boolean`	**`removeElement`**`(Object obj)` Removes the first (lowest-indexed) occurrence of the argument from this vector.
`void`	**`removeElementAt`**`(int index)` Deletes the component at the specified index.
`protected void`	**`removeRange`**`(int fromIndex, int toIndex)` Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
`boolean`	**`retainAll`**`(Collection<?> c)` Retains only the elements in this Vector that are contained in the specified Collection.
`E`	**`set`**`(int index, E element)` Replaces the element at the specified position in this Vector with the specified element.
`void`	**`setElementAt`**`(E obj, int index)` Sets the component at the specified `index` of this vector to be the specified object.
`void`	~~**`setSize`**`(int newSize)`~~ [illegible] ~~Sets the size of this vector.~~
`int`	**`size`**`()` Returns the number of components in this vector.
`List<E>`	**`subList`**`(int fromIndex, int toIndex)` Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.
`Object[]`	**`toArray`**`()` Returns an array containing all of the elements in this Vector in the correct order.
`<T> T[]`	**`toArray`**`(T[] a)` Returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array.
`String`	**`toString`**`()` Returns a string representation of this Vector, containing the String representation of each element.
`void`	**`trimToSize`**`()` Trims the capacity of this vector to be the vector's current size.

Methods inherited from class java.util.AbstractList