Class java.util.Vector

```
java.lang.Object
   |
   +----java.util.Vector
```

public class **Vector**
extends Object

Vector class (a growable array).

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 elements are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. Setting the capacity to what you want before inserting a large number of objects will reduce the amount of incremental reallocation. You can safely ignore the capacity and the vector will still work correctly.

Version:
1.27, 08/18/95
Author:
Jonathan Payne, Lee Boynton

Variable Index

o **capacityIncrement**
The size of the increment.
o **elementCount**
The number of elements in the buffer.
o **elementData**
The buffer where elements are stored.

Constructor Index

o **Vector**(int, int)
Constructs an empty vector with the specified storage capacity and the specified capacityIncrement.

o **Vector**(int)
    Constructs an empty vector with the specified storage capacity.
o **Vector**()
    Constructs an empty vector.

Method Index

o **addElement**(Object)
    Adds the specified object as the last element of the vector.
o **capacity**()
    Returns the current capacity of the vector.
o **clone**()
    Clones this vector.
o **contains**(Object)
    Returns true if the specified object is a value of the collection.
o **copyInto**(Object[])
    Copies the elements of this vector into the specified array.
o **elementAt**(int)
    Returns the element at the specified index.
o **elements**()
    Returns an enumeration of the elements.
o **ensureCapacity**(int)
    Ensures that the vector has at least the specified capacity.
o **firstElement**()
    Returns the first element of the sequence.
o **indexOf**(Object)
    Searches for the specified object, starting from the first position and returns an index to it.
o **indexOf**(Object, int)
    Searches for the specified object, starting at the specified position and returns an index to it.
o **insertElementAt**(Object, int)
    Inserts the specified object as an element at the specified index.
o **isEmpty**()
    Returns true if the collection contains no values.
o **lastElement**()
    Returns the last element of the sequence.
o **lastIndexOf**(Object)
    Searches backwards for the specified object, starting from the last position and returns an index to it.
o **lastIndexOf**(Object, int)
    Searches backwards for the specified object, starting from the specified position and returns an index to it.
o **removeAllElements**()
    Removes all elements of the vector.
o **removeElement**(Object)
    Removes the element from the vector.
o **removeElementAt**(int)

Deletes the element at the specified index.

o **setElementAt**(Object, int)
    Sets the element at the specified index to be the specified object.

o **setSize**(int)
    Sets the size of the vector.

o **size**()
    Returns the number of elements in the vector.

o **toString**()
    Converts the vector to a string.

o **trimToSize**()
    Trims the vector's capacity down to size.

Variables

o **elementData**

```
protected Object elementData[]
```

The buffer where elements are stored.

o **elementCount**

```
protected int elementCount
```

The number of elements in the buffer.

o **capacityIncrement**

```
protected int capacityIncrement
```

The size of the increment. If it is 0 the size of the the buffer is doubled everytime it needs to grow.

Constructors

o **Vector**

```
public Vector(int initialCapacity,
              int capacityIncrement)
```

Constructs an empty vector with the specified storage capacity and the specified capacityIncrement.

Parameters:
initialCapacity – the initial storage capacity of the vector
capacityIncrement – how much to increase the element's size by.

o **Vector**

```
public Vector(int initialCapacity)
```

Constructs an empty vector with the specified storage capacity.
Parameters:
initialCapacity – the initial storage capacity of the vector

o **Vector**

```
public Vector()
```

Constructs an empty vector.

Methods

o **copyInto**

```
public final synchronized void copyInto(Object anArray[])
```

Copies the elements of this vector into the specified array.
Parameters:
anArray – the array where elements get copied into

o **trimToSize**

```
public final synchronized void trimToSize()
```

Trims the vector's capacity down to size. Use this operation to minimize the storage of a vector. Subsequent insertions will cause reallocation.

o **ensureCapacity**

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

Ensures that the vector has at least the specified capacity.
Parameters:
minCapacity – the desired minimum capacity

o **setSize**

```
public final synchronized void setSize(int newSize)
```

Sets the size of the vector. If the size shrinks, the extra elements (at the end of the vector) are lost; if the size increases, the new elements are set to null.
Parameters:
newSize – the new size of the vector

o **capacity**

```
public final int capacity()
```

Returns the current capacity of the vector.

o **size**

```
public final int size()
```

Returns the number of elements in the vector. Note that this is not the same as the vector's capacity.

o **isEmpty**

```
public final boolean isEmpty()
```

Returns true if the collection contains no values.

o **elements**

```
public final synchronized Enumeration elements()
```

Returns an enumeration of the elements. Use the Enumeration methods on the returned object to fetch the elements sequentially.

o **contains**

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

Returns true if the specified object is a value of the collection.
Parameters:
elem – the desired element

o **indexOf**

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

Searches for the specified object, starting from the first position and returns an index to it.
Parameters:
elem – the desired element
Returns:
the index of the element, or −1 if it was not found.

o **indexOf**

```
public final synchronized int indexOf(Object elem,
                                      int index)
```

Searches for the specified object, starting at the specified position and returns an

index to it.
Parameters:
elem – the desired element
index – the index where to start searching
Returns:
the index of the element, or –1 if it was not found.

o **lastIndexOf**

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

Searches backwards for the specified object, starting from the last position and returns an index to it.
Parameters:
elem – the desired element
Returns:
the index of the element, or –1 if it was not found.

o **lastIndexOf**

```
public final synchronized int lastIndexOf(Object elem,
                                          int index)
```

Searches backwards for the specified object, starting from the specified position and returns an index to it.
Parameters:
elem – the desired element
index – the index where to start searching
Returns:
the index of the element, or –1 if it was not found.

o **elementAt**

```
public final synchronized Object elementAt(int index)
```

Returns the element at the specified index.
Parameters:
index – the index of the desired element
Throws: ArrayIndexOutOfBoundsException
If an invalid index was given.

o **firstElement**

```
public final synchronized Object firstElement()
```

Returns the first element of the sequence.
Throws: NoSuchElementException
If the sequence is empty.

o **lastElement**

```
public final synchronized Object lastElement()
```

Returns the last element of the sequence.
Throws: NoSuchElementException
If the sequence is empty.

o **setElementAt**

```
public final synchronized void setElementAt(Object obj,
                                            int index)
```

Sets the element at the specified index to be the specified object. The previous element at that position is discarded.
Parameters:
obj – what the element is to be set to
index – the specified index
Throws: ArrayIndexOutOfBoundsException
If the index was invalid.

o **removeElementAt**

```
public final synchronized void removeElementAt(int index)
```

Deletes the element at the specified index. Elements with an index greater than the current index are moved down.
Parameters:
index – the element to remove
Throws: ArrayIndexOutOfBoundsException
If the index was invalid.

o **insertElementAt**

```
public final synchronized void insertElementAt(Object obj,
                                               int index)
```

Inserts the specified object as an element at the specified index. Elements with an index greater or equal to the current index are shifted up.
Parameters:
obj – the element to insert
index – where to insert the new element
Throws: ArrayIndexOutOfBoundsException
If the index was invalid.

o **addElement**

```
public final synchronized void addElement(Object obj)
```

Adds the specified object as the last element of the vector.
Parameters:
obj – the element to be added

o **removeElement**

```
public final synchronized boolean removeElement(Object obj)
```

Removes the element from the vector. If the object occurs more than once, only the first is removed. If the object is not an element, returns false.
Parameters:
obj – the element to be removed
Returns:
true if the element was actually removed; false otherwise.

o **removeAllElements**

```
public final synchronized void removeAllElements()
```

Removes all elements of the vector. The vector becomes empty.

o **clone**

```
public synchronized Object clone()
```

Clones this vector. The elements are **not** cloned.
Overrides:
clone in class Object

o **toString**

```
public final synchronized String toString()
```

Converts the vector to a string. Useful for debugging.
Overrides:
toString in class Object

All Packages   This Package   Previous   Next