JAVA - THE VECTOR CLASS

Vector implements a dynamic array. It is similar to ArrayList, but with two differences:

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

The Vector class supports four constructors. The first form creates a default vector, which has an initial size of 10:

```
Vector( )
```

The second form creates a vector whose initial capacity is specified by size:

```
Vector(int size)
```

The third form creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward:

```
Vector(int size, int incr)
```

The fourth form creates a vector that contains the elements of collection c:

```
Vector(Collection c)
```

Apart from the methods inherited from its parent classes, Vector defines the following methods:

SN	Methods with Description
1	**void add(int index, Object element)** Inserts the specified element at the specified position in this Vector.
2	**boolean add(Object o)** Appends the specified element to the end of this Vector.
3	**boolean addAll(Collection 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.
4	**boolean addAll(int index, Collection c)** Inserts all of the elements in in the specified Collection into this Vector at the specified position.
5	**void addElement(Object obj)** Adds the specified component to the end of this vector, increasing its size by one.

6 **int capacity()**

Returns the current capacity of this vector.

7 **void clear()**

Removes all of the elements from this Vector.

8 **Object clone()**

Returns a clone of this vector.

9 **boolean contains(Object elem)**

Tests if the specified object is a component in this vector.

10 **boolean containsAll(Collection c)**

Returns true if this Vector contains all of the elements in the specified Collection.

11 **void copyInto(Object[] anArray)**

Copies the components of this vector into the specified array.

12 **Object elementAt(int index)**

Returns the component at the specified index.

13 **Enumeration elements()**

Returns an enumeration of the components of this vector.

14 **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.

15 **boolean equals(Object o)**

Compares the specified Object with this Vector for equality.

16 **Object firstElement()**

Returns the first component (the item at index 0) of this vector.

17 **Object get(int index)**

Returns the element at the specified position in this Vector.

18 **int hashCode()**

Returns the hash code value for this Vector.

19 **int indexOf(Object elem)**

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

20 **int indexOf(Object elem, int index)**

Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the equals method.

21 **void insertElementAt(Object obj, int index)**

Inserts the specified object as a component in this vector at the specified index.

22 **boolean isEmpty()**

Tests if this vector has no components.

23 **Object lastElement()**

Returns the last component of the vector.

24 **int lastIndexOf(Object elem)**

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

25 **int lastIndexOf(Object elem, int index)**

Searches backwards for the specified object, starting from the specified index, and returns an index to it.

26 **Object remove(int index)**

Removes the element at the specified position in this Vector.

27 **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.

28 **boolean removeAll(Collection c)**

Removes from this Vector all of its elements that are contained in the specified Collection.

29 **void removeAllElements()**

Removes all components from this vector and sets its size to zero.

30 **boolean removeElement(Object obj)**

Removes the first (lowest-indexed) occurrence of the argument from this vector.

31 **void removeElementAt(int index)**

removeElementAt(int index)

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

Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

33 **boolean retainAll(Collection c)**

Retains only the elements in this Vector that are contained in the specified Collection.

34 **Object set(int index, Object element)**

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

35 **void setElementAt(Object obj, int index)**

Sets the component at the specified index of this vector to be the specified object.

36 **void setSize(int newSize)**

Sets the size of this vector.

37 **int size()**

Returns the number of components in this vector.

38 **List subList(int fromIndex, int toIndex)**

Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.

39 **Object[] toArray()**

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

40 **Object[] toArray(Object[] 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.

41 **String toString()**

Returns a string representation of this Vector, containing the String representation of each element.

42 **void trimToSize()**

Trims the capacity of this vector to be the vector's current size.

Example:

The following program illustrates several of the methods supported by this collection:

```
import java.util.*;

public class VectorDemo {

   public static void main(String args[]) {
      // initial size is 3, increment is 2
      Vector v = new Vector(3, 2);
      System.out.println("Initial size: " + v.size());
      System.out.println("Initial capacity: " +
      v.capacity());
      v.addElement(new Integer(1));
      v.addElement(new Integer(2));
```

```
      v.addElement(new Integer(3));
      v.addElement(new Integer(4));
      System.out.println("Capacity after four additions: " +
          v.capacity());

      v.addElement(new Double(5.45));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Double(6.08));
      v.addElement(new Integer(7));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Float(9.4));
      v.addElement(new Integer(10));
      System.out.println("Current capacity: " +
      v.capacity());
      v.addElement(new Integer(11));
      v.addElement(new Integer(12));
      System.out.println("First element: " +
         (Integer)v.firstElement());
      System.out.println("Last element: " +
         (Integer)v.lastElement());
      if(v.contains(new Integer(3)))
         System.out.println("Vector contains 3.");
      // enumerate the elements in the vector.
      Enumeration vEnum = v.elements();
      System.out.println("\nElements in vector:");
      while(vEnum.hasMoreElements())
         System.out.print(vEnum.nextElement() + " ");
      System.out.println();
   }
}
```

This would produce the following result:

```
Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.

Elements in vector:
1 2 3 4 5.45 6.08 7 9.4 10 11 12
```