List [Interface]: Vector

Definition

Vector implements a dynamic array that means it can grow or shrink as required. It contains components that can be accessed using an integer index.

Is similar to ArrayList but vector is synchronized and has some legacy methods that the collection framework does not contain. Maintains an insertion order, it is rarely used in non-thread environment as it is synchronized, this gives him poor performance in adding, searching, deleting, and updating its elements.

Vector capacity

If the increment is specified, Vector will expand according to it each allocation cycle. If the increment is not specified, the vector capacity gets doubled in each allocation cycle. Vector defines three protected data members.

- Int capacityIncreament: Contains the increment value
- Int elementCount: Number of the element in vector
- Object elementData[]: Array that holds the vector is stored in it.

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 storage increases in chunks the size of capacityIncrement. An application can increase the capacity of a vector before inserting many components; this reduces the amount of incremental reallocation.

Field summary

Modifier and type	**Field and description**
protected int	capacityIncreament
protected int	elementCount
protected Object[]	elementData

Constructor summary

Vector<E> v = new **Vector**<E>()	Creates a default vector of the initial capacity is 10.
Vector<E> v = new **Vector**<E>(int size)	Creates a vector whose initial capacity is specified by size.
Vector<E> v = new **Vector**<E>(int size, int incr)	Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specified the number of element to allocate each time a vector is resized upward.

Vector<E> v = new **Vector**<E>(Collection c)	Creates a vector that contains the elements of collection C.

Methods summary

Modifier and Type	**Method and Description**
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.
E	**elementAt(int index)** Returns the component at the specified index
boolean	**equals(Object** o) Compares the specified Object with this vector for equality
void	**forEach(Consumer** < ? super **E** > action) Peforms the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
E	**get(int** index**)** Returns the element at the specified position in this Vector.
void	**insertElementAt(E** obj, int index)

	Inserts the specified object as a component in this vector at the specified index.
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.
ListIterator<E>	**listIterator()** Returns a list iterator over the elements in this list(in proper sequence)
ListIterator<E>	**listIterator (int** index**)** Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	remove(**Object o)** Removes the first occurrence of the specified element in this Vector if the Vector does no contain the element, it is unchanged.
E	**set(**int index, **E** element**)** Replaces the element at the specified position in this Vector with the specified element.
int	**size()** Returns the number of components in this vector
void	**sort(Comparator** < ? super **E** > c) Sorts this list according to the order induced by the specified **Comparator**
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 Vector's current size.

When is its use recommended?

- When we use a single thread process. Vector class is synchronized, which means that only one thread at a time can access the code.
- As the vector class is using list Interface. Vector may be used to store a list of products sold or list of supplements available at a local store.
- Vector can be used in local programs when the time of searching elements it’s not important as when is a web service or similar.