**9.4 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 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. |

***Vectors***

Vectors (the java.util.Vector class) are commonly used instead of arrays, because they expand automatically when new data is added to them. The Java 2 Collections API introduced the similar ArrayList data structure. ArrayLists are unsynchronized and therefore faster than Vectors, but less secure in a multithreaded environment. The Vector class was changed in Java 2 to add the additional methods supported by ArrayList. See below for a reasons to use each. The description below is for the (new) Vector class.

Vectors can hold only Objects and not primitive types (eg, int). If you want to put a primitive type in a Vector, put it inside an object (eg, to save an integer value use the Integer class or define your own class). If you use the Integer wrapper, you will not be able to change the integer value, so it is sometimes useful to define your own class.

**To Create a Vector**

You must import either import java.util.Vector; or import java.util.\*;. Vectors are implemented with an array, and when that array is full and an additional element is added, a new array must be allocated. Because it takes time to create a bigger array and copy the elements from the old array to the new array, it is a little faster to create a Vector with a size that it will commonly be when full. Of course, if you knew the final size, you could simply use an array. However, for non-critical sections of code programmers typically don't specify an initial size.

* **Create a Vector with default initial size**   
  Vector v = new Vector();
* **Create a Vector with an initial size**   
  Vector v = new Vector(300);

**To Add elements to the end of a Vector**

v.add(s); // adds s to the end of the Vector v

**To get the elements from a Vector (ListIterator)**

You can use a for loop to get all the elements from a Vector, but another very common way to go over all elements in a Vector is to use a ListIterator. The advantage of an iterator is that it it can be used with other data structures, so that if you later change to using a linked list for example, you won't have to change your code. Here is an example of using an iterator to print all elements (Strings) in a vector. The two most useful methods are **hasNext()**, which returns true if there are more elements, and **next()**, which returns the next element.

ListIterator iter = v.listIterator();

while (iter.hasNext()) {

System.out.println((String)iter.next());

}

**Common Vector Methods**

There are many useful methods in the Vector class and its parent classes. Here are some of the most useful. **v** is a Vector, **i** is an int index, **o** is an Object.