Vectors in Java

(Vector)

Packages → Java.util import Java.util.*;
Classes → Vector
Interfaces → Enumeration

What is Java **Vector** Class?

- **Java class Vector provides the capabilities of array-like data structures that can dynamically resize themselves.**
- **Vector can hold objects of any type and any number.**
- **At any time the Vector contains a certain number of elements which can be different types of objects and the size is less than or equal to its *capacity*.**
- **The *capacity* is the space that has been reserved for the array.**
- **If a Vector needs to grow, it grows by an increment that you specify.**
- **If you do not specify a capacity increment, the system will automatically double the size of the Vector each time additional capacity is needed.**

Class **Vector** constructors

public Vector();

Constructs an empty vector so that its internal data array has size 10.

e.g. Vector v = new Vector();

public Vector(int initialCapacity);

Constructs an empty vector with the specified initial capacity.

e.g. Vector v = new Vector(10);

public Vector(int initialCapacity, int capacityIncrement);

Constructs an empty vector with the specified initial capacity and capacity increment.

e.g. Vector v = new Vector(4, 2);

Vector v = new Vector(Collection c);
It constructs a vector that contains the elements of a collection c.

Class **Vector** methods

public void addElement(Object obj)

Adds the specified component to the end of this vector, increasing its size by one. The capacity of this vector is increased if its size becomes greater than its capacity.

e.g. v.addElement(item);

public boolean removeElement(Object obj)

Removes the first (lowest-indexed) occurrence of the argument from this vector. If the object is found in this vector, each component in the vector with an index greater or equal to the object's index is shifted downward to have an index one smaller than the value it had previously.

e.g. v.removeElement(item)

Class **Vector** methods

public Object firstElement()

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

e.g. v.firstElement();

public Object lastElement()

Returns the last component of the vector.

e.g. v.lastElement();

public boolean isEmpty()

Tests if this vector has no components.

e.g. v.isEmpty();

Class **Vector** methods

public boolean contains(Object elem)

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

e.g. if (v.contains(item))

public int indexOf(Object elem)

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

e.g. v.indexOf(item) ;

public int size()

Returns the number of components in this vector.

e.g. v.size();

Class **Vector** methods

v.elementAt(n)

Gives the name of the nth object.

v.removeElementAt(n)

Removes the item stored in the nth position of the list.

v.removeAllElements()

Removes all the elements in the vector.

v.insertElementAt(item,n)

Inserts the item at nth position.

v.copyInto(array)

Copies all items from vector to array.

Class **Vector** methods

public int capacity()

Returns the current capacity of this vector.

e.g. v.capacity();

public Enumeration elements()

Returns an enumeration of the components of this vector. The returned Enumeration object will enumerate all items in this vector. The first item generated is the item at index 0, then the item at index 1, and so on.

Interface **Enumeration** and its methods

public abstract interface Enumeration

Enumeration means one at a time.

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the nextElement method return successive elements of the series.

e.g. Enumeration enum = v.elements();

public Object nextElement()

Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

public boolean hasMoreElements()

Tests if this enumeration contains more elements.

e.g. While (enum.hasMoreElements())

Problem statement

Implement a menu driven program for the following:
Accepts a shopping list (name, price and quantity)from the command line and stores them in a vector.
To delete specific item (given by user) in the vector
Add item at the end of the vector
Add item at specific location
Print the contents of vector using enumeration interface.

Collections.sort() function

- **java.util.Collections.sort()** method is present in java.util.Collections class. It is used to sort the elements present in the specified list of Collection in ascending order.
- It can sort the elements of Array as well as linked list, queue and many more present in it.

- public static void sort(Vector v)

- v : A Vector type object we want to sort. This method doesn't return anything

Comparator interface in java

- A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class**.** Following function compare obj1 with obj2.
- Syntax: public int compare(Object obj1, Object obj2)
- Suppose we have an Array/ArrayList/Vector of our own class type, containing fields like roll no, name, address, DOB, etc, and we need to sort the array based on Roll no or name?

Comparator interface in java

- **Method 1**: One obvious approach is to write our own sort() function using one of the standard algorithms. This solution requires rewriting the whole sorting code for different criteria like Roll No. and Name.

- **Method 2**: Using comparator interface- Comparator interface is used to order the objects of a user-defined class. This interface is present in java.util package and contains 2 methods
  - compare(Object obj1, Object obj2)
  - equals(Object element).
-
- Using a comparator, we can sort the elements based on data members. For instance, it may be on roll no, name, age, or anything else.
- Method of Collections class for sorting Vector elements is used to sort the elements of Vector by the given comparator.

Comparator interface in java

- public void sort(Vector v, ComparatorClass c)To sort a given Vector, ComparatorClass must implement a Comparator interface.
- Internally the Sort method calls Compare method of the classes it is sorting.
- To compare two elements, it asks “Which is greater?” Compare method returns -1, 0, or 1 to say if it is less than, equal, or greater to the other.
- It uses this result to then determine if they should be swapped for their sort.