# Java.util.Vector Class in Java

The Vector class implements a growable array of objects. Vectors basically falls in legacy classes but now it is fully compatible with collections.

- Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index
- They are very similar to ArrayList but Vector is synchronised and have some legacy method which collection framework does not contain.
- It extends **AbstractList** and implements **List** interfaces.

**Constructor:**

- **Vector()**: Creates a default vector of initial capacity is 10.
- **Vector(int size):** Creates a vector whose initial capacity is specified by size.
- **Vector(int size, int incr):** Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time that a vector is resized upward.
- **Vector(Collection c):** Creates a vector that contains the elements of collection c.

**Important points regarding Increment of vector capacity:**
If increment is specified, Vector will expand according to it in each allocation cycle but if increment is not specified then vector's capacity get doubled in each allocation cycle. Vector defines three protected data member:

- **int capacityIncreament:** Contains the increment value.
- **int elementCount:** Number of elements currently in vector stored in it.
- **Object elementData[]:** Array that holds the vector is stored in it.

**Methods:**

1. **boolean add(Object obj)** This method appends the specified element to the end of this vector.

```
Syntax: public boolean add(Object obj)
Returns: true if the specified element is added
successfully into the Vector, otherwise it returns false.
Exception: NA.
```

```java
// Java code illustrating add() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector
        Vector v = new Vector();


        v.add(1);
        v.add(2);
        v.add("geeks");
        v.add("forGeeks");
        v.add(3);

        System.out.println("Vector is " + v);

    }
}
```

Run on IDE

Output:

```
[1, 2, geeks, forGeeks, 3]
```

2. **void add(int index, Object obj)** This method inserts the specified element at the specified position in this Vector.

```
Syntax: public void add(int index, Object obj)
Returns: NA.
Exception: IndexOutOfBoundsException, method throws this exception
if the index (obj position) we are trying to access is out of range
(index  size()).
```

```java
// Java code illustrating add() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector
        Vector v = new Vector();
        v.add(0, 1);
        v.add(1, 2);
        v.add(2, "geeks");
        v.add(3, "forGeeks");
        v.add(4, 3);

        System.out.println("Vector is " + v);

    }
}
```

Run on IDE

Output:

```
Vector is: [1, 2, geeks, forGeeks, 3]
```

3. **boolean addAll(Collection c)** This method appends all of the elements in the specified Collection to the end of this Vector.

```
Syntax: public boolean addAll(Collection c)
Returns: Returns true if operation succeeded otherwise false.
Exception: NullPointerException thrown if collection is null.
```

```
// Java code illustrating addAll()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
       ArrayList arr = new ArrayList();
       arr.add(3);
       arr.add("geeks");
       arr.add("forgeeks");
       arr.add(4);


        // createn default vector
        Vector v = new Vector();


        // copying all element of array list int0 vector
        v.addAll(arr);

        // checking vector v
        System.out.println("vector v:" + v);
    }
}
```

Run on IDE

Output:

```
vector v:[3, geeks, forgeeks, 4]
```

4. **boolean addAll(int index, Collection c)** This method inserts all of the elements in the specified Collection into this Vector at the specified.
   position.

```
Syntax: public boolean addAll(int index, Collection c)
Returns: true if this list changed as a result of the call.
Exception: IndexOutOfBoundsException -- If the index is out of range,
NullPointerException -- If the specified collection is null.
```

```java
// Java code illustrating addAll()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
       ArrayList arr = new ArrayList();
       arr.add(3);
       arr.add("geeks");
       arr.add("forgeeks");
       arr.add(4);


        // createn default vector
        Vector v = new Vector();

        v.add(2);
        // copying all element of array list int0 vector
        v.addAll(1, arr);

        // checking vector v
        System.out.println("vector v:" + v);
     }
}
```

Run on IDE

Output:

```
vector v:[2, 3, geeks, forgeeks, 4]
```

5. **void clear()** This method removes all of the elements from this vector.

```
Syntax: public void clear()
Returns: NA.
Exception: NA.
```

```java
// Java code illustrating clear() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        // create default vector
        Vector v = new Vector();


        v.add(0, 1);
        v.add(1, 2);
        v.add(2, "geeks");
        v.add(3, "forGeeks");
        v.add(4, 3);

        System.out.println("Vector is: " + v);

        //clearing the vector
        v.clear();

        // checking vector
        System.out.println("after clearing: " + v);

    }
}
```

Run on IDE

Output:

```
Vector is: [1, 2, geeks, forGeeks, 3]
after clearing: []
```

6. **Object clone()** This method returns a clone of this vector.

```
Syntax: public Object clone()
Returns: a clone of this ArrayList instance.
Exception: NA.
```

```
// Java code illustrating clone()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector
        Vector v = new Vector();

        Vector v_clone = new Vector();

        v.add(0, 1);
        v.add(1, 2);
        v.add(2, "geeks");
        v.add(3, "forGeeks");
        v.add(4, 3);

        v_clone = (Vector)v.clone();

        // checking vector
        System.out.println("Clone of v: " + v_clone);
    }
}
```

Run on IDE

Output:

```
Clone of v: [1, 2, geeks, forGeeks, 3]
```

7. **boolean contains(Object o)** This method returns true if this vector contains the specified element.

```
Syntax: public boolean contains(object o)
Returns: true if the operation is succeeded otherwise false.
Exception: NA.
```

```
// Java code illustrating contains() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector
        Vector v = new Vector();


        v.add(1);
        v.add(2);
        v.add("geeks");
        v.add("forGeeks");
        v.add(3);

        // check whether vector contains "forGeeks"
        if(v.contains("forGeeks"))
        System.out.println("forGeeks exists");


    }
}
```

Run on IDE

Output:

```
forGeeks exists
```

8. **void ensureCapacity(int minCapacity):** This method increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the m[illegible]um capacity argument .

```
Syntax: public void ensureCapacity(int minCapacity)
Returns: NA.
Exception: NA.
```

```
// Java code illustrating ensureCapacity() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        // ensuring capacity
        v.ensureCapacity(22);

        // cheking capacity
        System.out.println("Minimum capacity: " + v.capacity());

    }
}
```

Run on IDE

Output:

```
Minimum capacity: 22
```

9. **Object get(int index):** This method returns the element at the specified position in this Vector.

```
Syntax: public void ensureCapacity(int minCapacity)
Returns: returns the element at specified positions .
Exception: IndexOutOfBoundsException -- if the index is out of range.
```

```
// Java code illustrating get() methods
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // get the element at index 2
        System.out.println("element at indexx 2 is: " + v.get(2));

    }
}
```

Run on IDE

Output:

```
element at indexx 2 is: Geeks
```

10. **int indexOf(Object o):** This method returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

```
Syntax: public int indexOf(Object o)
Returns: the index of the first occurrence of the specified
element in this list, or -1 if this list does not contain the element.
Exception: NA.
```

```
// Java code illustrating indexOf() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // get the element at index of Geeks
        System.out.println("index of Geeks is: " + v.indexOf("Geeks"));

    }
}
```

Run on IDE

Output:

```
index of Geeks is: 2
```

11. **boolean isEmpty():** This method tests if this vector has no components.

```
Syntax: public boolean isEmpty()
Returns: true if vector is empty otherwise false.
Exception: NA.
```

```
// Java code illustrating isEmpty() method
import java.util.*;
class Vector_demo
```

```
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        v.clear();

        // check whether vector is empty or not
        if(v.isEmpty())
        System.out.println("Vector is clear");

    }
}
```

Run on IDE

Output:

```
Vector is clear
```

12. **int lastIndexOf(Object o):** This method returns the index of the last occurrence of the specified element in this vector, or -1 if this vector does not contain the element.

```
Syntax: public int lastIndexOf(Object o)
Returns: returns the index of the last occurrence of the
specified element in this list, or -1 if this list does not contain the element.
Exception: NA.
```

```
// Java code illustrating lastIndexof()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // checking last occurance of 2
        System.out.println("last occurance of 2 is: " + v.lastIndexOf(2));

    }
}
```

Run on IDE

Output:

```
last occurance of 2 is: 1
```

13. **boolean remove(Object o):** This method removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.

```
Syntax: public boolean remove(Object o)
Returns: Returns the first occurrence of element.
Exception: NA.
```

```
// Java code illustrating remove method()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

       // removing first occurance element at 1
       v.remove(1);

       // checking vector
       System.out.println("after removal: " + v);

    }
}
```

Run on IDE

Output:

```
after removal: [1, Geeks, forGeeks, 4]
```

14. **boolean equals(Object o):** This method compares the specified Object with this Vector for equality.

```
Syntax: public boolean equal(Object o)
Returns: true if operation succeeded otherwise false.
Exception: NA.
```

```
// Java code illustrating equals() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

       // second vector
       Vector v_2nd = new Vector();

        v_2nd.add(1);
        v_2nd.add(2);
        v_2nd.add("Geeks");
        v_2nd.add("forGeeks");
        v_2nd.add(4);

     if(v.equals(v_2nd))
         System.out.println("both vectors are equal");
    }
}
```

Run on IDE

Output:

```
both vectors are equal
```

15. **Object firstElement():** This method returns the first component (the item at index 0) of this vector.

```
Syntax: public Object firstElement()
Returns: NA.
Exception: NoSuchElementException -- This exception is returned
if this vector has no components.
```

```
// Java code illustrating firstElement() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // first element of vector
        System.out.println("first element of vector is: " + v.firstElement());
    }
}
```

Run on IDE

Output:

```
first element of vector is: 1
```

16. **void trimToSize():** This method trims the capacity of this vector to be the vector's current size.

```
Syntax: public void trimToSize()
Returns: NA.
Exception: NA.
```

```
// Java code illustrating trimToSize() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

       // checking initial capacity
       System.out.println("Initial capacity: " + v.capacity());

       // trim capacity to size
       v.trimToSize();

       // checking capacity after triming
      System.out.println("capacity after triming: " + v.capacity());
    }
}
```

Run on IDE

Output:

```
Initial capacity: 10
capacity after triming: 5
```

17. **String toString():** The toString() method is used to return a string representation of this Vector, containing the String representation of each element.

```
Syntax: public String toString()
Returns:   a string representation of this collection.
Exception: NA
```

```
// Java code illustrating toString() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

        // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);


       // string equivalent of vector
       System.out.println(" String equivalent of vector: " + v.toString());
    }
}
```

Run on IDE

Output:

```
String equivalent of vector: [1, 2, Geeks, forGeeks, 4]
```

18. **object[ ] toArray():** This method returns a array representation of this Vector, containing the String representation of each element.

```
Syntax: public  object[ ] toArray()
Returns: an array containing all of the elements in this collection.
Exception: NA.
```

```
// Java code illustrating toArray() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {

         String elements[] = { "M", "N", "O", "P", "Q" };
         Set set = new HashSet(Arrays.asList(elements));

         String[] strObj = new String[set.size()];
```

```
            strObj = (String[]) set.toArray(strObj);

            for (int i = 0; i < strObj.length; i++)
           {
            System.out.println(strObj[i]);
           }
            System.out.println(set);
        }
    }
```

Run on IDE

Output:

```
P
Q
M
N
O
[P, Q, M, N, O]
```

19. **int size():** This method returns the number of components in this vector.

```
Syntax: public int size()
Returns: returns the number of components in this vector.
Exception: NA
```

```
// Java code illustrating size() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
     // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // size of vector
        System.out.println(" size of vector: " + v.size());
   }
}
```

Run on IDE

Output:

```
size of vector: 5
```

20. **void setSize(int newSize):** This method sets the size.

```
Syntax: public void setSize(int newSize)
Returns: NA.
Exception: ArrayIndexOutOfBoundsException -- This exception is thrown
if the new size is negative.
```

```java
// Java code illustrating setSize() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
     // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // setting new size of vector
        v.setSize(13);

        // size of vector
        System.out.println("size of vector: " + v.size());
   }
}
```

Run on IDE

Output:

```
size of vector: 13
```

21. **void setElementAt(Object obj, int index):** This method sets the component at the specified index of this vector to be the specified object.

```
Syntax: public void setElementAt(E obj, int index)
Returns: NA.
Exception: ArrayIndexOutOfBoundsException -- This exception is thrown
if the accessed index is out of range.
```

```java
// Java code illustrating setElementAt() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
     // create default vector of capacity 10
        Vector v = new Vector();

        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

        // set 4 at the place of 2
        v.setElementAt(4, 1);

        System.out.println("vector: " + v);
    }
}
```

Run on IDE

Output:

```
vector: [1, 4, Geeks, forGeeks, 4]
```

22. **retainAll(Collection c):** This method retains only the elements in this Vector that are contained in the specified Collection.

```
Syntax: public boolean retainAll(Collection c)
Returns: true if this Vector is changed as a result of the call.
Exception: NullPointerException -- This exception is thrown if the
specified collection is null.
```

```
// Java code illustrating retainAll() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);
        Vector vecretain = new Vector(4);

        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // this elements will be retained
        vecretain.add(5);
        vecretain.add(3);
        vecretain.add(2);

        System.out.println("Calling retainAll()");
        vec.retainAll(vecretain);

        // let us print all the elements available in vector
        System.out.println("Numbers after removal :- ");

        Iterator itr = vec.iterator();

        while(itr.hasNext())
        {
         System.out.println(itr.next());

        }
    }
}
```

Run on IDE

Output:

```
Calling retainAll()
Numbers after removal :-
2
3
5
```

23. **void removeAllElements():** This method removes all components from this vector and sets its size to zero.

```
Syntax: public void removeAllElements()
Returns: NA.
Exception: NA.
```

```
// Java code illustrating removeAllElements() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
         Vector vec = new Vector(7);


        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // remove all elements
        vec.removeAllElements();

        //checking vector's size
        System.out.println("Size: " + vec.size());

        //checking vector's componenets
        System.out.println("vector's componenets: " + vec);
    }
}
```

Run on IDE

Output:

```
Size: 0
vector's componenets: []
```

24. **Object lastElement():** This method returns the last component of the vector.

```
Syntax: public Object lastElement()
Returns: returns the last component of the vector,
i.e., the component at index size() - 1.
Exception: NoSuchElementException -- This exception is thrown
if this vector is empty
```

```
// Java code illustrating lastElement() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);


        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // checking last element of vector
        System.out.println("vector's last componenets: " + vec.lastElement());
    }
}
```

Run on IDE

Output:

```
vector's last componenets: 7
```

25. **int hashCode():** This method returns the hash code value for this Vector.

```
Syntax: public int hashCode()
Returns: returns the hash code value(int) for this list.
Exception: NA.
```

```
// Java code illustrating hashCode() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);
        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // checking hash code
        System.out.println("Hash code: " + vec.hashCode());
    }
}
```

Run on IDE

Output:

```
Hash code: -1604500253
```

26. **boolean removeElement(Object obj):** This method removes the first occurrence of the argument from this vector.

```
Syntax: public boolean removeElement(Object obj)
Returns: true if operation is succeeded otherwise false.
Exception:
```

```
// Java code illustrating removeElement()
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);


        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // remove an element
        vec.removeElement(5);

        // checking vector
        System.out.println("Vector after removal: " + vec);
    }
}
```

Run on IDE

Output:

```
Vector after removal: [1, 2, 3, 4, 6, 7]
```

27. **void copyInto(Object[ ] anArray):** This method copies the components of this vector into the specified array.

```
Syntax: public void copyInto(Object[] anArray)
Returns: NA.
Exception: NullPointerException -- if the given array is null.
```

```
// Java code illustrating copyInto() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);


        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        Integer[] arr=new Integer[7];

        // copy componnent of vector int array arr
        vec.copyInto(arr);

        System.out.println("elements in array arr: ");
        for(Integer num : arr)
        {
          System.out.println(num);
        }
    }
}
```

Run on IDE

Output:

```
elements in array arr:
1
2
3
4
5
6
7
```

28. **int capacity():** This method returns the current capacity of this vector.

```
Syntax: public int capacity()
returns:  returns the current capacity of the
vector as an integer value. Here capacity means the length of
its internal data array, kept in the field elementData of this vector.
 Exception: NA.
```

```
// Java code illustrating capacity() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);


        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

        // checking capacity
        System.out.println("Capacity of vector: " + vec.capacity());
    }
}
```

Run on IDE

Output:

```
Capacity of vector: 7
```

29. **void insertElementAt(Object obj, int index):** This method inserts the specified object as a component in this vector at the specified index.

```
Syntax: public void insertElementAt(E obj, int index)
Returns: NA.
Exception: ArrayIndexOutOfBoundsException -- This exception is thrown
 if the index is invalid.
```

```
// Java code illustrating insertElementAt() method
import java.util.*;
class Vector_demo
{
    public static void main(String[] arg)
    {
        Vector vec = new Vector(7);
        // use add() method to add elements in the vector
        vec.add(1);
        vec.add(2);
        vec.add(3);
        vec.add(4);
        vec.add(5);
        vec.add(6);
        vec.add(7);

         // insert 10 at the index 7
        vec.insertElementAt(10, 7);

        // checking vector
        System.out.println(" Vector: " + vec);
    }
}
```

Run on IDE

Output:

```
Vector: [1, 2, 3, 4, 5, 6, 7, 10]
```

## GATE CS Corner  Company Wise Coding Practice

Java  Java - util package

## Recommended Posts:

Stack Class in Java
LinkedList in java
Set in Java
ArrayList in Java
Vector vs ArrayList in Java
Java.lang.String.compareTo()
Templates in C++ vs Generics in Java
Converting Text to Speech in Java
Difference between x++ and x=x+1 in Java