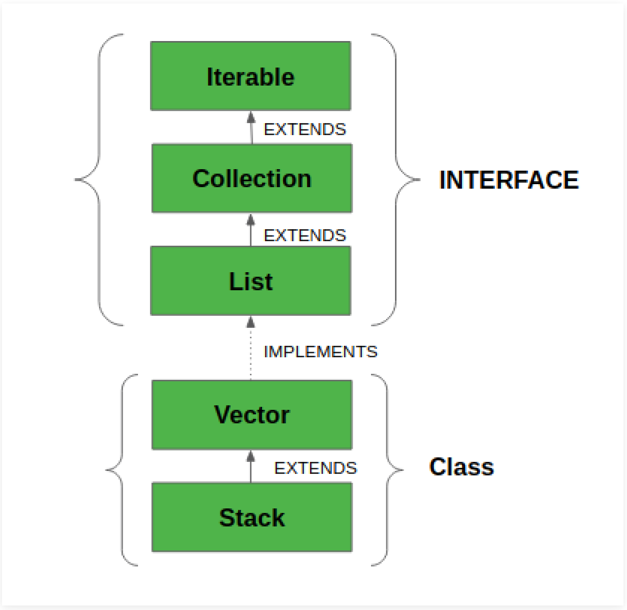
Vector Class in Java

Difficulty Level : Easy Last Updated : 07 Dec, 2021

The Vector class implements a growable array of objects. Vectors fall in legacy classes, but now it is fully compatible with collections. It is found in java.util package and implement the List interface, so we can use all the methods of List interface as shown below as follows:



- 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 synchronized and has some legacy methods that the collection framework does not contain.
- It also maintains an insertion order like an ArrayList. Still, it is rarely used in a non-thread environment as it is **synchronized**, and due to this, it gives a poor performance in adding, searching, deleting, and updating its elements.
- The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails and throws the **ConcurrentModificationException.**

Syntax:

```
public class Vector<E> extends AbstractList<E> implements List<E>, RandomAcce
```

Here, **E** is the type of element.

- It extends AbstractList and implements List interfaces.
- It implements Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>,

RandomAccess interfaces.
- The directly known subclass is Stack.

Important points regarding the Increment of vector capacity are as follows:

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

- ***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.

Common Errors in the declaration of Vectors are as follows**:**

- Vector throws an **IllegalArgumentException** if the InitialSize of the vector defined is negative.
- If the specified collection is null, It throws **NullPointerException**.

Constructors

1. Vector(): Creates a default vector of the initial capacity is 10.

```
Vector<E> v = new Vector<E>();
```

2. Vector(int size): Creates a vector whose initial capacity is specified by size.

```
Vector<E> v = new Vector<E>(int size);
```

3. 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 a vector is resized upward.

```
Vector<E> v = new Vector<E>(int size, int incr);
```

4. Vector(Collection c): Creates a vector that contains the elements of collection c.

```
Vector<E> v = new Vector<E>(Collection c);
```

Methods in Vector Class

METHOD	DESCRIPTION
add(E e)	Appends the specified element to the end of this Vector.
add(int index, E element)	Inserts the specified element at the specified position in this Vector.
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.
addAll(int index, Collection<? extends E> c)	Insert all of the elements in the specified Collection into this Vector at the specified position.
addElement(E obj)	Adds the specified component to the end of this vector, increasing its size by one.
capacity()	Returns the current capacity of this vector.
clear()	Removes all of the elements from this Vector.
clone()	Returns a clone of this vector.
contains(Object o)	Returns true if this vector contains the specified element.
containsAll(Collection<?> c)	Returns true if this Vector contains all of the elements in the specified Collection.
copyInto(Object[] anArray)	Copies the components of this vector into the specified array.
elementAt(int index)	Returns the component at the specified index.
elements()	Returns an enumeration of the components of this vector.
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.

METHOD	DESCRIPTION
equals(Object o)	Compares the specified Object with this Vector for equality.
firstElement()	Returns the first component (the item at index 0) of this vector.
forEach(Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
get(int index)	Returns the element at the specified position in this Vector.
hashCode()	Returns the hash code value for this Vector.
indexOf(Object o)	Returns the index of the first occurrence of the specified element in this vector, or -1 if this vector does not contain the element.
indexOf(Object o, int index)	Returns the index of the first occurrence of the specified element in this vector, searching forwards from the index, or returns -1 if the element is not found.
insertElementAt(E obj, int index)	Inserts the specified object as a component in this vector at the specified index.
isEmpty()	Tests if this vector has no components.
iterator()	Returns an iterator over the elements in this list in a proper sequence.
lastElement()	Returns the last component of the vector.
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.
lastIndexOf(Object o, int index)	Returns the index of the last occurrence of the specified element in this vector, searching backward from the index, or returns -1 if the element is not found.

METHOD	DESCRIPTION
listIterator()	Returns a list iterator over the elements in this list (in proper sequence).
listIterator(int index)	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
remove(int index)	Removes the element at the specified position in this Vector.
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.
removeAll(Collection<?> c)	Removes from this Vector all of its elements contained in the specified Collection.
removeAllElements()	Removes all components from this vector and sets its size to zero.
removeElement(Object obj)	Removes the first (lowest-indexed) occurrence of the argument from this vector.
removeElementAt(int index)	Deletes the component at the specified index.
removeIf(Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
removeRange(int fromIndex, int toIndex)	Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
replaceAll(UnaryOperator<E> operator)	Replaces each element of this list with the result of applying the operator to that element.
retainAll(Collection<?> c)	Retains only the elements in this Vector contained in the specified Collection.
set(int index, E element)	Replaces the element at the specified position in this Vector with the specified element.

METHOD	DESCRIPTION
setElementAt(E obj, int index)	Sets the component at the specified index of this vector to be the specified object.
setSize(int newSize)	Sets the size of this vector.
size()	Returns the number of components in this vector.
sort(Comparator<? super E> c)	Sorts this list according to the order induced by the specified Comparator.
spliterator()	Creates a late-binding and fail-fast Spliterator over the elements in this list.
subList(int fromIndex, int toIndex)	Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive.
toArray()	Returns an array containing all of the elements in this Vector in the correct order.
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.
toString()	Returns a string representation of this Vector, containing the String representation of each element.
trimToSize()	Trims the capacity of this vector to be the vector's current size.

Let us first discuss and implement how to create and use a Vector prior to landing upon the methods of this class.

Example:

```
// Java Program to Demonstrate Working of Vector
// Via Creating and Using It

// Importing required classes
```

```java
import java.io.*;
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Size of the Vector
        int n = 5;

        // Declaring the Vector with
        // initial size n
        Vector<Integer> v = new Vector<Integer>(n);

        // Appending new elements at
        // the end of the vector
        for (int i = 1; i <= n; i++)
            v.add(i);

        // Printing elements
        System.out.println(v);

        // Remove element at index 3
        v.remove(3);

        // Displaying the vector
        // after deletion
        System.out.println(v);

        // iterating over vector elements
        // usign for loop
        for (int i = 0; i < v.size(); i++)

            // Printing elements one by one
            System.out.print(v.get(i) + " ");
    }
}
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```

Note:

- *If the vector increment is not specified then it’s capacity will be doubled in every increment cycle.*
- *The capacity of a vector cannot be below the size, it may equal to it.*

Performing Various Operations on Vector class in Java

Let us discuss various operations on Vector class that are listed as follows:

1. Adding elements
2. Updating elements
3. Removing elements
4. Iterating over elements

Operation 1: Adding Elements

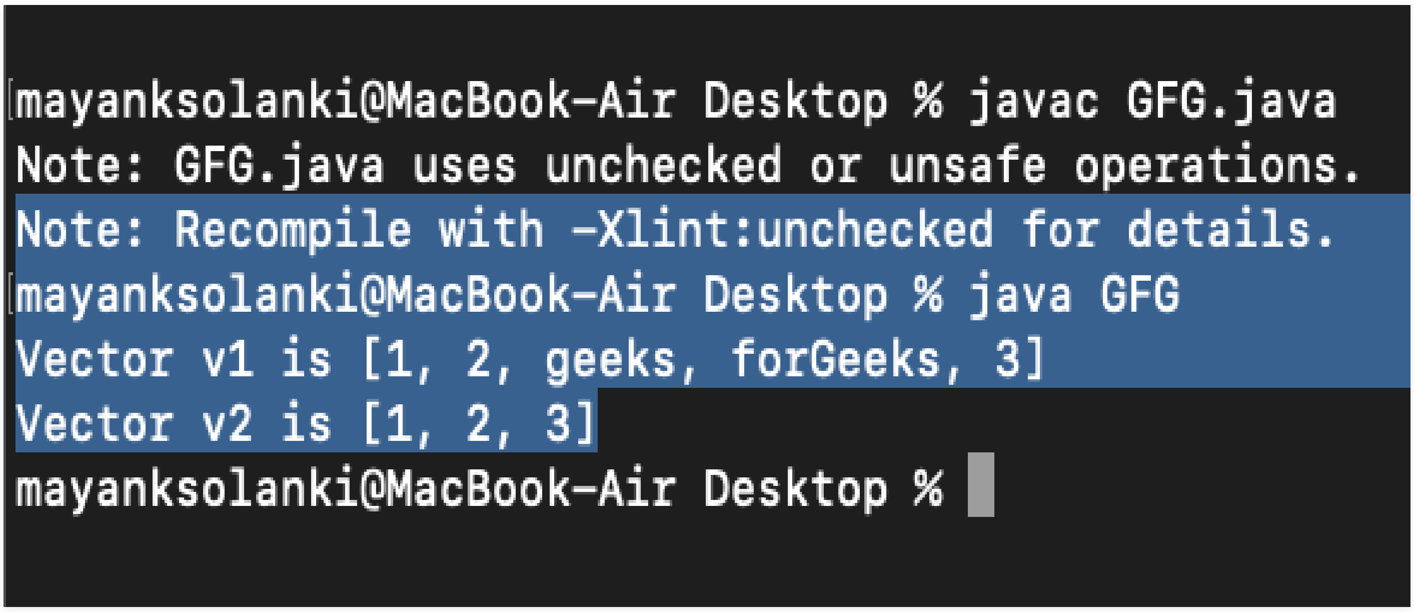
In order to add the elements to the Vector, we use the add() method. This method is overloaded to perform multiple operations based on different parameters. They are listed below as follows:

- **add(Object):** This method is used to add an element at the end of the Vector.
- **add(int index, Object):** This method is used to add an element at a specific index in the Vector.

Example:

```
// Java Program to Add Elements in Vector Class

// Importing required classes
import java.io.*;
import java.util.*;

// Main class
// AddElementsToVector
class GFG {

    // Main driver method
    public static void main(String[] arg)
    {
        // Case 1
        // Creating a default vector
        Vector v1 = new Vector();

        // Adding custom elements
        // using add() method
        v1.add(1);
        v1.add(2);
        v1.add("geeks");
        v1.add("forGeeks");
        v1.add(3);

        // Printing the vector elements to the console
        System.out.println("Vector v1 is " + v1);

        // Case 2
        // Creating generic vector
        Vector<Integer> v2 = new Vector<Integer>();

        // Adding custom elements
        // using add() method
        v2.add(1);
        v2.add(2);
        v2.add(3);

        // Printing the vector elements to the console
        System.out.println("Vector v2 is " + v2);
    }
}
```

Output:



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
Note: GFG.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
mayanksolanki@MacBook-Air Desktop % java GFG
Vector v1 is [1, 2, geeks, forGeeks, 3]
Vector v2 is [1, 2, 3]
mayanksolanki@MacBook-Air Desktop %
```

```
Vector v1 is [1, 2, geeks, forGeeks, 3]
Vector v2 is [1, 2, 3]
```

Operation 2: Updating Elements

After adding the elements, if we wish to change the element, it can be done using the set() method. Since a Vector is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element to be inserted at that index.

Example

```java
// Java code to change the
// elements in vector class

import java.util.*;

public class UpdatingVector {

    public static void main(String args[])
    {
        // Creating an empty Vector
        Vector<Integer> vec_tor = new Vector<Integer>();

        // Use add() method to add elements in the vector
        vec_tor.add(12);
        vec_tor.add(23);
        vec_tor.add(22);
        vec_tor.add(10);
        vec_tor.add(20);

        // Displaying the Vector
        System.out.println("Vector: " + vec_tor);

        // Using set() method to replace 12 with 21
        System.out.println("The Object that is replaced is: "
                           + vec_tor.set(0, 21));

        // Using set() method to replace 20 with 50
        System.out.println("The Object that is replaced is: "
                           + vec_tor.set(4, 50));

        // Displaying the modified vector
        System.out.println("The new Vector is:" + vec_tor);
    }
}
```

Output

```
Vector: [12, 23, 22, 10, 20]
The Object that is replaced is: 12
The Object that is replaced is: 20
The new Vector is:[21, 23, 22, 10, 50]
```

Operation 3: Removing Elements

In order to remove an element from a Vector, we can use the remove() method. This method is overloaded to perform multiple operations based on different parameters. They are:

- **remove(Object):** This method is used to remove an object from the Vector. If there are multiple such objects, then the first occurrence of the object is removed.
- **remove(int index):** Since a Vector is indexed, this method takes an integer value which simply removes the element present at that specific index in the Vector. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

Example

```java
// Java code illustrating the removal
// of elements from vector

import java.util.*;
import java.io.*;

class RemovingElementsFromVector {

    public static void main(String[] arg)
    {

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

          // Add elements using add() method
        v.add(1);
        v.add(2);
        v.add("Geeks");
        v.add("forGeeks");
        v.add(4);

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

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

Output:

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

Operation 4: Iterating the Vector

There are multiple ways to iterate through the Vector. The most famous ways are by using the basic for loop in combination with a get() method to get the element at a specific index and the advanced for a loop.

Example

```
// Java program to iterate the elements
// in a Vector

import java.util.*;

public class IteratingVector {

    public static void main(String args[])
    {
          // create an instance of vector
        Vector<String> v = new Vector<>();

          // Add elements using add() method
        v.add("Geeks");
        v.add("Geeks");
        v.add(1, "For");

        // Using the Get method and the
        // for loop
        for (int i = 0; i < v.size(); i++) {

            System.out.print(v.get(i) + " ");
        }

        System.out.println();

        // Using the for each loop
        for (String str : v)
            System.out.print(str + " ");
    }
}
```

Output

```
Geeks For Geeks
Geeks For Geeks
```

Note: *Do give a read to the ArrayList vs Vector class in Java to grasp it better.*