**Java - The 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.

Following is the list of constructors provided by the vector class.

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **Vector( )**  This constructor creates a default vector, which has an initial size of 10. |
| 2 | **Vector(int size)**  This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size. |
| 3 | **Vector(int size, int incr)**  This constructor 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. |
| 4 | **Vector(Collection c)**  This constructor creates a vector that contains the elements of collection c. |

Apart from the methods inherited from its parent classes, Vector defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & 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. |
| 9 | **boolean contains(Object elem)**  Tests if the specified object is a component in this vector. |
| 10 | **boolean containsAll(Collection c)**  Returns true if this vector contains all of the elements in the specified Collection. |
| 11 | **void copyInto(Object[] anArray)**  Copies the components of this vector into the specified array. |
| 12 | **Object elementAt(int index)**  Returns the component at the specified index. |
| 13 | **Enumeration elements()**  Returns an enumeration of the components of this vector. |
| 14 | **void 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. |
| 15 | **boolean equals(Object o)**  Compares the specified Object with this vector for equality. |
| 16 | **Object firstElement()**  Returns the first component (the item at index 0) of this vector. |
| 17 | **Object get(int index)**  Returns the element at the specified position in this vector. |
| 18 | **int hashCode()**  Returns the hash code value for this vector. |
| 19 | **int indexOf(Object elem)**  Searches for the first occurence of the given argument, testing for equality using the equals method. |
| 20 | **int indexOf(Object elem, int index)**  Searches for the first occurence of the given argument, beginning the search at index, and testing for equality using the equals method. |
| 21 | **void insertElementAt(Object obj, int index)**  Inserts the specified object as a component in this vector at the specified index. |
| 22 | **boolean isEmpty()**  Tests if this vector has no components. |
| 23 | **Object lastElement()**  Returns the last component of the vector. |
| 24 | **int lastIndexOf(Object elem)**  Returns the index of the last occurrence of the specified object in this vector. |
| 25 | **int lastIndexOf(Object elem, int index)**  Searches backwards for the specified object, starting from the specified index, and returns an index to it. |
| 26 | **Object remove(int index)**  Removes the element at the specified position in this vector. |
| 27 | **boolean 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. |
| 28 | **boolean removeAll(Collection c)**  Removes from this vector all of its elements that are contained in the specified Collection. |
| 29 | **void removeAllElements()**  Removes all components from this vector and sets its size to zero. |
| 30 | **boolean removeElement(Object obj)**  Removes the first (lowest-indexed) occurrence of the argument from this vector. |
| 31 | **void removeElementAt(int index)**  removeElementAt(int index). |
| 32 | **protected void removeRange(int fromIndex, int toIndex)**  Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive. |
| 33 | **boolean retainAll(Collection c)**  Retains only the elements in this vector that are contained in the specified Collection. |
| 34 | **Object set(int index, Object element)**  Replaces the element at the specified position in this vector with the specified element. |
| 35 | **void setElementAt(Object obj, int index)**  Sets the component at the specified index of this vector to be the specified object. |
| 36 | **void setSize(int newSize)**  Sets the size of this vector. |
| 37 | **int size()**  Returns the number of components in this vector. |
| 38 | **List subList(int fromIndex, int toIndex)**  Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive. |
| 39 | **Object[] toArray()**  Returns an array containing all of the elements in this vector in the correct order. |
| 40 | **Object[] toArray(Object[] 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. |
| 41 | **String toString()**  Returns a string representation of this vector, containing the String representation of each element. |
| 42 | **void trimToSize()**  Trims the capacity of this vector to be the vector's current size. |

# Difference between ArrayList and Vector In java 🡺

[ArrayList](https://beginnersbook.com/2013/12/java-arraylist/) and [Vector](https://beginnersbook.com/2013/12/vector-in-java/) both use Array as a data structure internally. However there are few differences in the way they store and process the data. In this post we will discuss the difference and similarities between ArrayList and Vector.

## ArrayList Vs Vector:

1) **Synchronization**: ArrayList is non-synchronized which means multiple threads can work on ArrayList at the same time. For e.g. if one thread is performing an add operation on ArrayList, there can be an another thread performing remove operation on ArrayList at the same time in a multithreaded environment

while Vector is synchronized. This means if one thread is working on Vector, no other thread can get a hold of it. Unlike ArrayList, only one thread can perform an operation on vector at a time.

2) **Resize:** Both ArrayList and Vector can grow and shrink dynamically to maintain the optimal use of storage, however the way they resized is different. ArrayList grow by half of its size when resized while Vector doubles the size of itself by default when grows.

3) **Performance**: ArrayList gives better performance as it is non-synchronized. Vector operations gives poor performance as they are thread-safe, the thread which works on Vector gets a lock on it which makes other thread wait till the lock is released.

4) **fail-fast**: First let me explain what is fail-fast: If the collection (ArrayList, vector etc) gets structurally modified by any means, except the **add or remove methods** of iterator, after creation of iterator then the iterator will throw [ConcurrentModificationException](http://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html" \t "_blank). Structural modification refers to the addition or deletion of elements from the collection.

As per the [Vector javadoc](http://docs.oracle.com/javase/7/docs/api/java/util/Vector.html) the Enumeration returned by Vector is not fail-fast. On the other side the iterator and listIterator returned by ArrayList are fail-fast.

5) **Who belongs to collection framework really?**The vector was not the part of collection framework, it has been included in collections later. It can be considered as Legacy code. There is nothing about Vector which List collection cannot do. Therefore Vector should be avoided. If there is a need of thread-safe operation make ArrayList synchronized as discussed in the next section of this post or use [CopyOnWriteArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html" \t "_blank) which is a thread-safe variant of ArrayList.

**There are few similarities between** **these classes which are as follows:**

1. Both Vector and ArrayList use growable array data structure.
2. The iterator and listIterator returned by these classes (Vector and ArrayList) are fail-fast.
3. They both are ordered collection classes as they maintain the elements insertion order.
4. Vector & ArrayList both allows duplicate and null values.
5. They both grows and shrinks automatically when overflow and deletion happens.

### **When to use ArrayList and when to use vector?**

It totally depends on the requirement. If there is a need to perform “thread-safe” operation the vector is your best bet as it ensures that only one thread access the collection at a time.

**Performance:** Synchronized operations consumes more time compared to non-synchronized ones so if there is no need for thread safe operation, ArrayList is a better choice as performance will be improved because of the concurrent processes.

**How to make ArrayList synchronized?**  
As I stated above ArrayList methods are non-synchronized but still if there is a need you can make them synchronized like this –

//Use Collecions.synzhonizedList method

List list = Collections.synchronizedList(new ArrayList());

...

//If you wanna use iterator on the synchronized list, use it

//like this. It should be in synchronized block.

synchronized (list) {

Iterator iterator = list.iterator();

while (iterator.hasNext())

...

iterator.next();

...

}

**Vector(int size, int incr)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.util.Enumeration; **import** java.util.Vector;  **public class** Main {   **public static void** main(String[] args) {  *// initial size is 3, increment is 2* Vector v = **new** Vector(3, 2);  System.***out***.println(**"Initial size: "** + v.size());  System.***out***.println(**"Initial capacity: "** + v.capacity());   v.addElement(**new** Integer(1));  v.addElement(**new** Integer(2));  v.addElement(**new** Integer(3));  v.addElement(**new** Integer(4));  System.***out***.println(**"Capacity after four additions: "** + v.capacity());   v.addElement(**new** Double(5.45));  System.***out***.println(**"Current capacity: "** + v.capacity());   v.addElement(**new** Double(6.08));  v.addElement(**new** Integer(7));  System.***out***.println(**"Current capacity: "** + v.capacity());   v.addElement(**new** Float(9.4));  v.addElement(**new** Integer(10));  System.***out***.println(**"Current capacity: "** + v.capacity());   v.addElement(**new** Integer(11));  v.addElement(**new** Integer(12));  System.***out***.println(**"First element: "** + (Integer)v.firstElement());  System.***out***.println(**"Last element: "** + (Integer)v.lastElement());   **if**(v.contains(**new** Integer(3)))  System.***out***.println(**"Vector contains 3."**);   *// enumerate the elements in the vector.* Enumeration vEnum = v.elements();  System.***out***.println(**"\nElements in vector:"**);   **while**(vEnum.hasMoreElements())  System.***out***.print(vEnum.nextElement() + **" "**);  System.***out***.println();  }  } } | Initial size: 0  Initial capacity: 3  Capacity after four additions: 5  Current capacity: 5  Current capacity: 7  Current capacity: 9  First element: 1  Last element: 12  Vector contains 3.  Elements in vector:  1 2 3 4 5.45 6.08 7 9.4 10 11 12 |

**Vector(int size)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.util.Enumeration; **import** java.util.Vector;  **public class** Main {   **public static void** main(String[] args) {  *// Vector(int size)* Vector v = **new** Vector(5);  System.***out***.println(**"Initial size: "** + v.size());  System.***out***.println(**"Initial capacity: "** + v.capacity());   v.addElement(5); *//int* v.addElement(5.3); *//float* v.addElement(655555555555555555555555.888888888888888888); *//long* v.addElement(**"A"**); *//String* System.***out***.println(**"elements are "** + v);   *//adding more elements* v.addElement(90);  System.***out***.println(**"elements are 2nd time "** + v);   *//adding elements more that initiated* v.addElement(**"55d"**);  System.***out***.println(**"elements are more than declared"** + v);   *//adding elements more that initiated* v.addElement(**"55.3332"**);  System.***out***.println(**"elements are more than declared"** + v);  }  } | Initial size: 0  Initial capacity: 5  elements are [5, 5.3, 6.555555555555556E23, A]  elements are 2nd time [5, 5.3, 6.555555555555556E23, A, 90]  elements are more than declared[5, 5.3, 6.555555555555556E23, A, 90, 55d]  elements are more than declared[5, 5.3, 6.555555555555556E23, A, 90, 55d, 55.3332] |

**Vector(Collection c)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.util.ArrayList; **import** java.util.Enumeration; **import** java.util.Vector;  **public class** Main {   **public static void** main(String[] args) {  *// Vector(int size)* ArrayList<Integer> arr = **new** ArrayList<>();  arr.add(10);  arr.add(15); *// arr.add(10.6); //error* arr.add(16);  arr.add(20);    Vector v = **new** Vector(arr);  System.***out***.println(**"Initial size: "** + v.size());  System.***out***.println(**"Initial capacity: "** + v.capacity());   System.***out***.println(**"passing collection as parameter "** + v);  }  } | Initial size: 4  Initial capacity: 4  passing collection as parameter [10, 15, 16, 20] |

**boolean add(Object o) & boolean addAll(Collection c) & boolean addAll(int index, Collection c)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// Vector(int size)* Collection<Integer> arr = **new** TreeSet<>();  arr.add(10);  arr.add(9);  arr.add(16);  arr.add(11);    Vector v = **new** Vector();  v.addElement(**"a"**);  v.addElement(**"b"**);  v.addElement(100);  v.addElement(5.5);   v.addAll(2, arr);   System.***out***.println(**"passing collection as parameter "** + v);  }  }  passing collection as parameter [a, b, 9, 10, 11, 16, 100, 5.5] | **package** com.company;  **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// Vector(int size)* Collection<Integer> arr = **new** TreeSet<>();  arr.add(10);  arr.add(9);  arr.add(16);  arr.add(11);    Vector v = **new** Vector();  v.addElement(**"a"**);  v.addElement(**"b"**);  v.addElement(100);  v.addElement(5.5);   v.addAll(arr);   System.***out***.println(**"passing collection as parameter "** + v);  } }  passing collection as parameter [a, b, 100, 5.5, 9, 10, 11, 16] |
| **package** com.company;  **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// Vector(int size)* Collection<Integer> arr = **new** TreeSet<>();  arr.add(10);  arr.add(9);  arr.add(16);  arr.add(11);    Vector v = **new** Vector();  v.addElement(**"a"**);  v.addElement(**"b"**);  v.addElement(100);  v.addElement(5.5);   v.add(arr);  System.***out***.println(**"passing collection as parameter "** + v); *//  // v.add(1, arr); // System.out.println("passing collection as parameter " + v);* } } | passing collection as parameter [a, b, 100, 5.5, [9, 10, 11, 16]] |
| v.add(1, arr); System.***out***.println(**"passing collection as parameter "** + v);  passing collection as parameter [a, [9, 10, 11, 16], b, 100, 5.5] |

**void clear() & Object clone() & boolean contains(Object ele)**

**& boolean containsAll(Collection c)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  Vector v = **new** Vector();  v.addElement(**"a"**);  v.addElement(**"b"**);  v.addElement(100);  v.addElement(5.5);  System.***out***.println(**"main collection is : "** + v);   Object x = v.clone();  System.***out***.println(**"Cloned collection is : "** + x);   ArrayList<Object> arrn = **new** ArrayList<>();  arrn.add(**"a"**);  arrn.add(**"b"**);  arrn.add(100);  arrn.add(5.5);   **if** (v.contains(100))  System.***out***.println(**"element found"**);  **else** System.***out***.println(**"No element found"**);   **if** (v.containsAll(arrn))  System.***out***.println(**"similar"**);  **else** System.***out***.println(**"not similar"**);   v.clear();  System.***out***.println(**"printing v after clear : "** + v);  } } | main collection is : [a, b, 100, 5.5]  Cloned collection is : [a, b, 100, 5.5]  element found  similar  printing v after clear : [] |

**void copyInto(Object[] anArray) & Object elementAt(int index)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// create an empty Vector vec with an initial capacity of 4* Vector<Integer> vec = **new** Vector<Integer>(4);   Integer anArray[] = **new** Integer[4];   anArray[0] = 100;  anArray[1] = 100;  anArray[2] = 100;  anArray[3] = 100;   *// use add() method to add elements in the vector* vec.add(4);  vec.add(3);  vec.add(2);  vec.add(1);   *// numbers in the array before copy* System.***out***.println(**"Numbers in the array before copy"**);  **for** (Integer number : anArray) {  System.***out***.println(**"Number = "** + number);  }  *// copy into the array* vec.copyInto(anArray);   *// numbers in the array after copy* System.***out***.println(**"Numbers in the array after copy"**);   **for** (Integer number : anArray) {  System.***out***.println(**"Number = "** + number);  }   Object object = vec.elementAt(2);  System.***out***.println(**"mal at that index is : "** + object);  } } | Numbers in the array before copy  Number = 100  Number = 100  Number = 100  Number = 100  Numbers in the array after copy  Number = 4  Number = 3  Number = 2  Number = 1  mal at that index is : 2 |

**Enumeration elements()**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// create an empty Vector vec with an initial capacity of 4* Vector<Integer> vec = **new** Vector<Integer>(4);   vec.add(4);  vec.add(3);  vec.add(2);  vec.add(1);   Enumeration ele = vec.elements();  **while** (ele.hasMoreElements())  System.***out***.println(**"the next element is : "** + ele.nextElement());  } } | the next element is : 4  the next element is : 3  the next element is : 2  the next element is : 1 |

**boolean equals(Object o) & Object firstElement() & Object get(int index)** **& int hashCode()** **& int indexOf(Object elem)** **& int indexOf(Object elem, int index)**

**& void insertElementAt(Object obj, int index)** **& boolean isEmpty()** **& Object lastElement()** **& int lastIndexOf(Object elem)** **& int lastIndexOf(Object elem, int index)** **& Object remove(int index)** **& boolean remove(Object o)** **& boolean removeAll(Collection c)** **& void removeAllElements()** **&**

**boolean removeElement(Object obj)** **& void removeElementAt(int index)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// create an empty Vector vec with an initial capacity of 4* Vector<Integer> vec = **new** Vector<Integer>(4);   vec.add(4);  vec.add(3);  vec.add(2);  vec.add(1);   Object x = vec.clone();   **if** (vec.equals(x))  System.***out***.println(**"element is found"**);  **else** System.***out***.println(**"element not found"**);   Object firstElement = vec.firstElement();  System.***out***.println(**"first element of the vector is : "** + firstElement);   Object get = vec.get(2);  System.***out***.println(**"the element is : "** + get);   **int** hashCode = vec.hashCode();  System.***out***.println(**"the hashcode value of the Vector is : "** + hashCode);   **int** indexOf = vec.indexOf(1); *//finds the element* **int** indexOf2 = vec.indexOf(3, 2); *//finds 3 from 2nd index of vec Vector* System.***out***.println(**"index of the vector is : "** + indexOf);  System.***out***.println(**"index of the vector is : "** + indexOf2);*//-1 : not found* vec.insertElementAt(5, 4);  System.***out***.println(**"the new vector will be : "** + vec);  *// vec.insertElementAt(6, 6); //living index 5 as null  // System.out.println("the new vector will be : " + vec);  //Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException   //boolean isEmpty()   //Object lastElement()   //int lastIndexOf(Object elem, int index)  //int lastIndexOf(Object elem)* System.***out***.println(**" Removingb particular element by index: "** +vec.remove(3)); *//Object remove(int index)* System.***out***.println(**"after removing the vector will be like : "** + vec);   *//boolean remove(Object o)   //adding some more into vec to check the remove all* vec.add(10);  vec.add(15);  vec.add(20);  vec.add(26);  System.***out***.println(**"the new Vector list will be like : "** + vec);  *//lets have an arraylist* ArrayList<Integer> integers = **new** ArrayList<>();  integers.add(10);  integers.add(20);  integers.add(3);   *//boolean removeAll(Collection c)* vec.removeAll(integers);  System.***out***.println(**"boolean removeAll(Collection c) does it like so "** + vec);   *//boolean removeElement(Object obj)* vec.removeElement(2);  System.***out***.println(**"boolean removeElement(Object obj) does so : "** + vec);  *//void removeElementAt(int index)* vec.removeElementAt(0);  System.***out***.println(**"void removeElementAt(int index) makes the list like this "** + vec);   *//void removeAllElements()* vec.removeAllElements();  System.***out***.println(**"void removeAllElements() does this like so "** + vec);   } } | element is found  first element of the vector is : 4  the element is : 2  the hashcode value of the Vector is : 1045631  index of the vector is : 3  index of the vector is : -1  the new vector will be : [4, 3, 2, 1, 5]  Removingb particular element by index: 1  after removing the vector will be like : [4, 3, 2, 5]  the new Vector list will be like : [4, 3, 2, 5, 10, 15, 20, 26]  boolean removeAll(Collection c) does it like so [4, 2, 5, 15, 26]  boolean removeElement(Object obj) does so : [4, 5, 15, 26]  void removeElementAt(int index) makes the list like this [5, 15, 26]  void removeAllElements() does this like so [] |

**protected void removeRange(int fromIndex, int toIndex)**

**List subList(int fromIndex, int toIndex)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {  *// create an empty Vector vec with an initial capacity of 7* Vector<Integer> vec = **new** Vector<Integer>(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);   */\*\*  \* Removing range of elements is not directly supported. However, it  \* can be done by using subList and clear methods.  \*/* System.***out***.println(**"Sublist of the vector will be like : "** + vec.subList(2,4));   System.***out***.println(**"Remove elements from 2nd to 4th"**);  vec.subList(2,4).clear();   *// let us print all the elements available in vector* System.***out***.println(**"Numbers after removal :- "**);   **for** (Integer number : vec) {  System.***out***.println(**"Number = "** + number);  }   } } | Sublist of the vector will be like : [3, 4]  Remove elements from 2nd to 4th  Numbers after removal :-  Number = 1  Number = 2  Number = 5  Number = 6  Number = 7 |

**boolean retainAll(Collection c)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   Vector<Integer> vec = **new** Vector<Integer>(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);   ArrayList<Integer> veca = **new** ArrayList<>();  veca.add(3);  veca.add(7);   System.***out***.println(**"before retaining all the elements from the Vector "** + vec);  *//boolean retainAll(Collection c)* vec.retainAll(veca);  System.***out***.println(**"After retaining all the elements from the Vector "** + vec);  } } | before retaining all the elements from the Vector [1, 2, 3, 4, 5, 6, 7]  After retaining all the elements from the Vector [3, 7] |

**Object set(int index, Object element)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   Vector<Integer> vec = **new** Vector<Integer>(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);   System.***out***.println(**"before using Object set(int index, Object element "** + vec);  *//Object set(int index, Object element) -- UPDATES ELEMENTS* vec.set(3, 100);  System.***out***.println(**"After using Object set(int index, Object element "** + vec);  *//AT THE END* vec.set(7, 150); //ERROR  System.***out***.println(**"After using Object set(int index, Object element "** + vec);  *//SKIPPING 8th AS NULL* vec.set(9, 200); //ERROR  System.***out***.println(**"After using Object set(int index, Object element "** + vec);  WHERE AS .ADD WILL WORK  vec.add(8);  System.***out***.println(**"new Vector will be like "** + vec);  } } | before using Object set(int index, Object element [1, 2, 3, 4, 5, 6, 7]  After using Object set(int index, Object element [1, 2, 3, 100, 5, 6, 7]  new Vector will be like [1, 2, 3, 100, 5, 6, 7, 8]  Object set(int index, **Object element**)   * GRANTS element AS OBJECT |

**void setElementAt(Object obj, int index)**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   Vector<Integer> vec = **new** Vector<Integer>(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);   vec.setElementAt(100, 3);  System.***out***.println(**"New Vector will be like "** + vec);  } } | New Vector will be like [1, 2, 3, 100, 5, 6, 7]  void setElementAt(**Object obj**, int index)   * GRANTS OBJECT AS OBJECT |

**int size()**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   Vector<Integer> vec = **new** Vector<Integer>(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);   **int** size = vec.size();  System.***out***.println(**"size of the new vector will be "** + size);  } } | size of the new vector will be 7 |

**Object[] toArray() & Object[] toArray(Object[] a) & String toString() & void trimToSize()**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   *// create an empty Vector vec with an initial capacity of 4* Vector<Integer> vec = **new** Vector<Integer>(4);   *// create an array* Integer[] anArray = **new** Integer[4];   *// use add() method to add elements in the vector* vec.add(4);  vec.add(3);  vec.add(2);  vec.add(1);   *// fill the array from the vector* vec.toArray(anArray);   *// check the content of the array* System.***out***.println(**"Elements are: "**);   **for** (**int** i = 0; i < anArray.**length**; i++) {  System.***out***.println(anArray[i]);  }  } } | Elements are:  4  3  2  1 |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.\*;  **public class** Main {   **public static void** main(String[] args) {   *// create an empty Vector vec with an initial capacity of 4* Vector<Integer> vec = **new** Vector<Integer>(4);   *// use add() method to add elements in the vector* vec.add(4);  vec.add(3);  vec.add(2);  vec.add(1);   *// convert the contents into string* vec.toString();   *// let us print all the elements available in vector* System.***out***.println(**"Added numbers are :- "**);   **for** (Integer number : vec) {  System.***out***.println(**"Number = "** + number);  }  } } | Added numbers are :-  Number = 4  Number = 3  Number = 2  Number = 1 |