BCA – 401: Java Programming

Rahul Kumar Singh

In today's Class we have discussed on Vectors in Java.

Java Vector:-

Vector is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the List interface, so we can use all the methods of List interface here.

It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.

The Iterators returned by the Vector class are fail-fast. In case of concurrent modification, it fails and throws the ConcurrentModificationException.

The Vector class is an implementation of the List interface that allows us to create resizable-arrays similar to the ArrayList class.

It is similar to the ArrayList, but with two differences-

- Vector is synchronized.
- Java Vector contains many legacy methods that are not the part of a collections framework.

Java Vector vs. ArrayList:-

In Java, both ArrayList and Vector implements the List interface and provides the same functionalities. However, there exist some differences between them.

The Vector class synchronizes each individual operation. This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.

It is because when one thread is accessing a vector, and at the same time another thread tries to access it, an exception called ConcurrentModificationException is generated. Hence, this continuous use of lock for each operation makes vectors less efficient.

However, in array lists, methods are not synchronized. Instead, it uses the Collections.synchronizedList() method that synchronizes the list as a whole.

Java Vector Constructors:-

Vector class supports four types of constructors. These are given below:

1) **vector() -** It constructs an empty vector with the default size as 10.

2) **vector(int initialCapacity) -** It constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.

3) **vector(int initialCapacity, int capacityIncrement) -** It constructs an empty vector with the specified initial capacity and capacity increment.

4) **Vector(Collection<? extends E> c) -** It constructs a vector that contains the elements of a collection c.

Creating a Vector:-

Here is how we can create vectors in Java.

```
Vector<Type> vector = new Vector<>();
```

Here, Type indicates the type of a linked list.

For example,

```
// create Integer type linked list
Vector<Integer> vector= new Vector<>();
// create String type linked list
Vector<String> vector= new Vector<>();
```

Methods of Vector:-

The Vector class also provides the resizable-array implementations of the List interface (similar to the ArrayList class). Some of the Vector methods are:

Add Elements to Vector:-

add(element) - adds an element to vectors.

add(index, element) - adds an element to the specified position.

addAll(vector) - adds all elements of a vector to another vector.

Example:-

```
import java.util.Vector;
class VectorExample
{
   public static void main(String[] args)
    {
      Vector<String> mammals= new Vector<>();
      // Using the add() method
      mammals.add("Dog");
      mammals.add("Horse");
      // Using index number
      mammals.add(2, "Cat");
      System.out.println("Vector: " + mammals);
      // Using addAll()
      Vector<String> animals = new Vector<>();
      animals.add("Crocodile");
```

```
        animals.addAll(mammals);
        System.out.println("New Vector: " + animals);
    }
}
```

Output:-

```
Vector: [Dog, Horse, Cat]
New Vector: [Crocodile, Dog, Horse, Cat]
```

Access Vector Elements:-

get(index) - returns an element specified by the index

iterator() - returns an iterator object to sequentially access vector elements

For example,

```
import java.util.Iterator;
import java.util.Vector;
class VectorExample
{
    public static void main(String[] args)
    {
        Vector<String> animals= new Vector<>();
        animals.add("Dog");
```

```
        animals.add("Horse");
        animals.add("Cat");
        // Using get()
        String element = animals.get(2);
        System.out.println("Element at index 2: " + element);
        // Using iterator()
        Iterator<String> iterate = animals.iterator();
        System.out.print("Vector: ");
        while(iterate.hasNext())
        {
            System.out.print(iterate.next());
            System.out.print(", ");
        }
    }
}
```

Output:-

```
Element at index 2: Cat
Vector: Dog, Horse, Cat,
```

Remove Vector Elements:-

remove(index) - removes an element from specified position.

removeAll() - removes all the elements.

clear() - removes all elements. It is more efficient than removeAll().

For example,

```
import java.util.Vector;

class  VectorExample

{

   public static void main(String[] args)

  {

      Vector<String> animals= new Vector<>();

      animals.add("Dog");

      animals.add("Horse");

      animals.add("Cat");

      System.out.println("Initial Vector: " + animals);

      // Using remove()

      String element = animals.remove(1);

      System.out.println("Removed Element: " + element);

      System.out.println("New Vector: " + animals);
```

```
        // Using clear()
        animals.clear();
        System.out.println("Vector after clear(): " + animals);
    }
}
```

Output:-

Initial Vector: [Dog, Horse, Cat]

Removed Element: Horse

New Vector: [Dog, Cat]

Vector after clear(): []

Others Vector Methods:-

contains() - searches the vector for specified element and returns a boolean result.

capacity() - It is used to get the current capacity of this vector.

clone() - It returns a clone of this vector.

contains() - It returns true if the vector contains the specified element.

containsAll() - It returns true if the vector contains all of the elements in the specified collection.

copyInto() - It is used to copy the components of the vector

into the specified array.

elementAt() - It is used to get the component at the specified index.

elements() - It returns an enumeration of the components of a vector.

ensureCapacity() - It is used to increase the capacity of the vector which is in use, if necessary. It ensures that the vector can hold at least the number of components specified by the minimum capacity argument.

equals() - It is used to compare the specified object with the vector for equality.

firstElement() - It is used to get the first component of the vector.

forEach() - It is used to perform the given action for each element of the Iterable until all elements have been processed or the action throws an exception.

hashCode() - It is used to get the hash code value of a vector.

indexOf() - It is used to get the index of the first occurrence of the specified element in the vector. It returns -1 if the vector does not contain the element.

insertElementAt() - It is used to insert the specified object as a component in the given vector at the specified index.

isEmpty() - It is used to check if this vector has no

components.

lastElement() - It is used to get the last component of the vector.

lastIndexOf() - It is used to get the index of the last occurrence of the specified element in the vector. It returns -1 if the vector does not contain the element.

listIterator()- It is used to get a list iterator over the elements in the list in proper sequence.

remove() - It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged.

removeAll()- It is used to delete all the elements from the vector that are present in the specified collection.

removeAllElements() - It is used to remove all elements from the vector and set the size of the vector to zero.

removeElement() - It is used to remove the first (lowest-indexed) occurrence of the argument from the vector.

removeElementAt() - It is used to delete the component at the specified index.

removeIf() - It is used to remove all of the elements of the collection that satisfy the given predicate.

removeRange() - It is used to delete all of the elements from the vector whose index is between fromIndex, inclusive and toIndex, exclusive.

replaceAll() - It is used to replace each element of the list with the result of applying the operator to that element.

retainAll() - It is used to retain only that element in the vector which is contained in the specified collection.

set() - It is used to replace the element at the specified position in the vector with the specified element.

setElementAt() - It is used to set the component at the specified index of the vector to the specified object.

setSize() - It is used to set the size of the given vector.

size() - It is used to get the number of components in the given vector.

sort() - It is used to sort the list according to the order induced by the specified Comparator.

spliterator() - It is used to create a late-binding and fail-fast Spliterator over the elements in the list.

subList() - It is used to get a view of the portion of the list between fromIndex, inclusive, and toIndex, exclusive.

toArray() - It is used to get an array containing all of the elements in this vector in correct order.

toString() - It is used to get a string representation of the vector.

trimToSize() - It is used to trim the capacity of the vector to the vector's current size.