Collection interface

28 January 2025 20:21

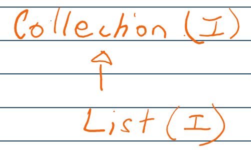
If we want to represent a group of individual objects as a single entity, then we should go for Collection.
Collection interface defines the most common methods which are applicable for any Collection object.

```
  boolean add(Object o)
  boolean addAll(Collection c)
  boolean remove(Object o)
  boolean removeAll(Collection c)
  boolean retainAll(Collection c) => to remove all objects except those present in c.
  void clear()
  boolean contains(Object o)
  boolean containsAll(Collection c)
  boolean isEmpty()
  int size()
  Object[] toArray()
  Iterator iterator()
```

Note: There is no concrete class which implements Collection interface directly.

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Collection Link for more study.



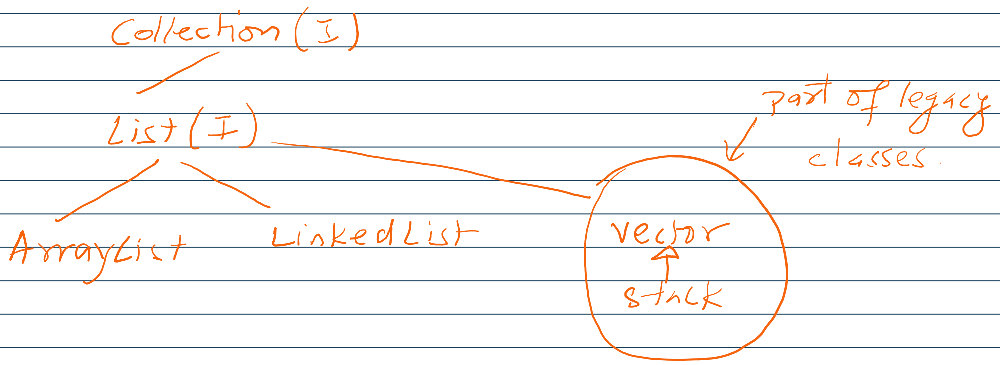
List interface:
List is a child interface of Collection interface. If we want to represent a group of individual object as a single entity, where duplicates are allowed and insertion order must be preserved, then we should go for List interface.

We can preserve insertion order via or with index. And we can differentiate, duplicate object by using index. Hence, index will play very important role in the list.
List interface defines the following specific methods:

```
    void add(int index, Object o)
    boolean addAll(int index, Collection c)
    Object get(int index)
    Object remove(int index)
    Object set(int index, Object newObj) => To replace the element present at the specified
index with provided object and returns old object.

    int indexOf(Object o) => returns index of first occurrence of 'o' and -1 if 'o' is not
present.
    int lastIndexOf(Object o)

    ListIterator listIterator()
```



stack

ArrayList is a class:
1. Resizable Array or Growable or shrinkable Array
2. Duplicates are allowed.
3. Insertion order is preserved.
4. Can hold Heterogenous objects.(except TreeSet and TreeMap objects)
5. "null" insertion is allowed.

Constructors:

ArrayList al = new ArrayList(); => Creates an empty array list object with default initial capacity 10. Once array list reaches its maximum capacity, then a new array list object will be created with new capacity.

newCapacity = currentCapacity * 3/2 + 1

ArrayList al = new ArrayList(int initalCapacity); => Creates an empty array list object with a specified initial capacity.

ArrayList al = new ArrayList(Collection c); => Creates an equivalent array list object for the given collection.

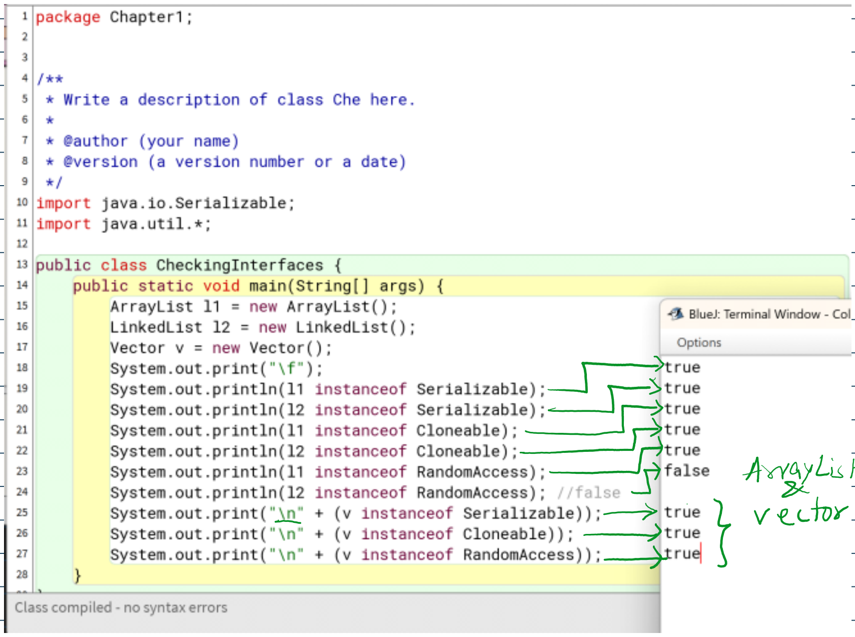
```
package ArraysProgram;
import java.util.ArrayList;
public class ArrayListOne {
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        al.add("A");
        al.add(10);
        al.add("Raghav");
        al.add(Math.PI);
        System.out.println(al);
        al.remove(1);
        System.out.println(al);
        al.add(2, Integer.MAX_VALUE);
        al.add("Shiv");
        System.out.println(al);
        System.out.println("Last element: "+al.get(al.size()-1));
    }
}
```

Usually we can use collection to hold and transfer object from one location to another location (Container). To provide support for this requirement every collection class by default implements Serializable and Cloneable interface.

ArrayList & Vector are the class that implements RandomAccess Interface.

RandomAccess interface: present in java.util package and it doesn't contain any method. It is a marker interface where required ability automatically used by the JVM. The primary purpose of this interface is to allow generic algorithms to alter their behavior to provide good performance when applied to either random or sequential access.
This interface is a member of the Java Collections Framework.

Since:
1.4



```java
package Chapter1;


/**
 * Write a description of class Che here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
import java.io.Serializable;
import java.util.*;

public class CheckingInterfaces {
    public static void main(String[] args) {
        ArrayList l1 = new ArrayList();
        LinkedList l2 = new LinkedList();
        Vector v = new Vector();
        System.out.print("\f");
        System.out.println(l1 instanceof Serializable);
        System.out.println(l2 instanceof Serializable);
        System.out.println(l1 instanceof Cloneable);
        System.out.println(l2 instanceof Cloneable);
        System.out.println(l1 instanceof RandomAccess);
        System.out.println(l2 instanceof RandomAccess); //false
        System.out.print("\n" + (v instanceof Serializable));
        System.out.print("\n" + (v instanceof Cloneable));
        System.out.print("\n" + (v instanceof RandomAccess));
    }
```

Class compiled - no syntax errors

```
true
true
true
true
true
false

true
true
true
```

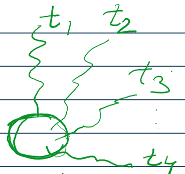
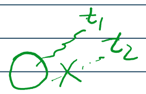
LinkedList has implemented
⇒ 1. Serializable
⇒ 2. Cloneable
it hasn't implemented RandomAccess.

ArrayList & vector has implement all 3 interfaces
1. Serializable
2. Cloneable
3. RandomAccess.

ArrayList is the best choice if our frequent operation is retrieval operation. Because ArrayList implements random access interface.
ArrayList Is the worst choice if our frequent operation is insertion or deletion in the middle.

What is the difference between ArrayList and Vector?

Threads.

ArrayList	**Vector**
Every method presented in the array list is non-synchronized.	Every method present in the vector is synchronized.
 (better-performance) non-synchronized Add a time multiple threads are allowed on arrow list object, and hence it is not thread safe.	At a time, only one thread is allowed to operate on vector object, and hence it is thread safe.  synchronized
A relatively performance is high because threats are not required to wait for the operation on array list object.	A relatively performance is low because threats are required to wait to operate on vector object.
Introduced in version 1.2. And it is not a legacy class.	Introduced in 1.0 version. And it is a legacy class.

How to get synchronized version of ArrayList object?

List (I)
⇑
ArrayList

ArrayList al = new ArrayList();
List l = Collections.synchronizedList(al);
↗
synchronized

By default, ArrayList is non synchronized, but we can get synchronized version of array list object by using synchronizedList() method of Collections class.

public static <T> List<T> synchronizedList(List<T> list)
Returns a synchronized (thread-safe) list backed by the specified list. In order to guarantee serial access, it is critical that **all** access to the backing list is accomplished through the returned list.

It is imperative that the user manually synchronize on the returned list when iterating over it:

```
  List list = Collections.synchronizedList(new ArrayList());
      ...
  synchronized (list) {
      Iterator i = list.iterator(); // Must be in synchronized block
      while (i.hasNext())
          foo(i.next());
  }
```

Failure to follow this advice may result in non-deterministic behavior.

The returned list will be serializable if the specified list is serializable.

Type Parameters:
T - the class of the objects in the list
Parameters:
list - the list to be "wrapped" in a synchronized list.
Returns:
a synchronized view of the specified list.

From <https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#synchronizedList-java.util.List->

Vector:
1. Underline data structure is resizable array or growable array.
2. Insertion order is preserved.
3. Duplicates are allowed.
4. Heterogeneous objects are allowed.
5. Null insertion is possible.
6. It implements Serializable, Cloneable and RandomAccess interfaces.
7. Every method present in the vector is synchronized, and hence vector object is thread safe.

Constructors:
Vector v = new Vector(); => Creates an empty vector object with default initial capacity of 10. Once vector reaches its max capacity, then a new vector object will be created with. new capacity: (new_capacity = current_capcity * 2) Double the capacity.

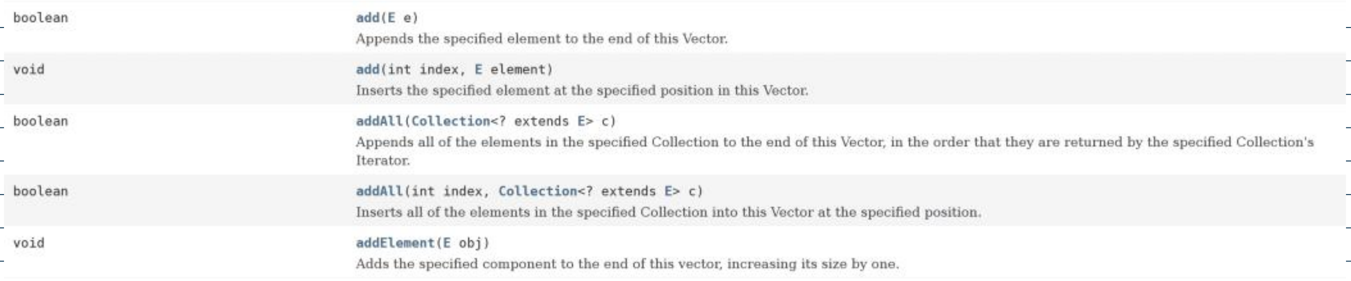
Vector v = new Vector(int initialCapacity); => Creates an empty vector object with the specified initial capacity.

Vector v = new Vector(int initialCapacity, int incrementalCapacity);

Vector v = new Vector(Collection c); => Creates an equivalent object for the given collection. This constructor meant for interconversion between collection objects.

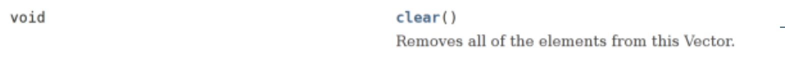
Vector Specific Methods

To add Objects:

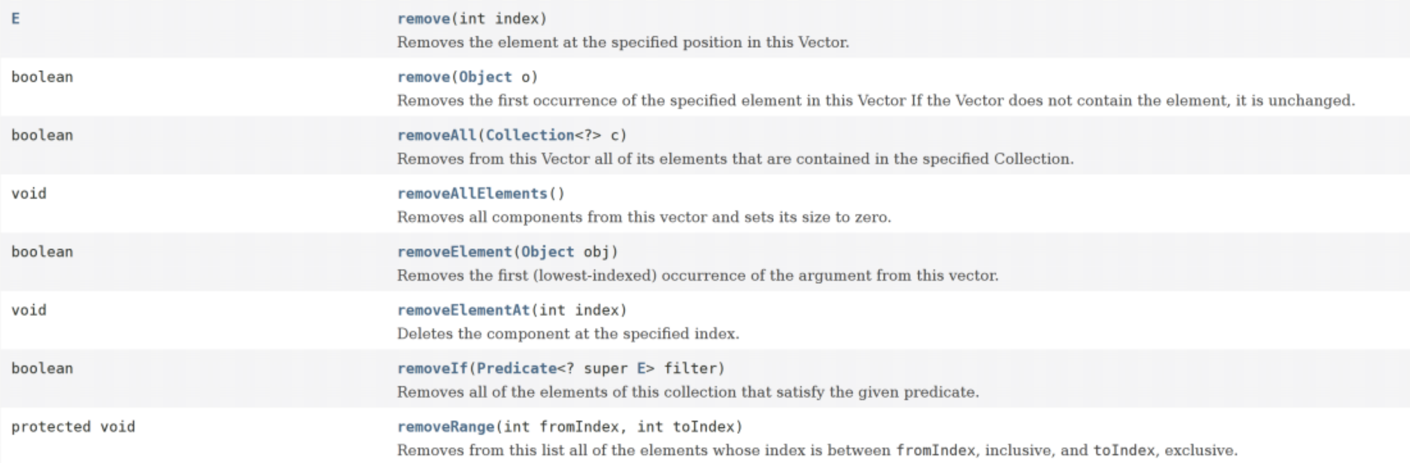


boolean	add(E e) Appends the specified element to the end of this Vector.
void	add(int index, E element) Inserts the specified element at the specified position in this Vector.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified Collection into this Vector at the specified position.
void	addElement(E obj) Adds the specified component to the end of this vector, increasing its size by one.

To remove objects:

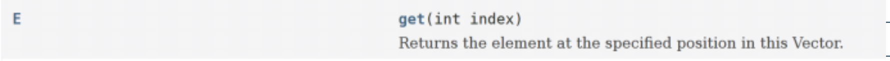


void	`clear()` Removes all of the elements from this Vector.

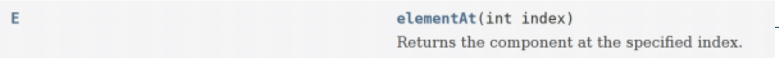


E	`remove(int index)` Removes the element at the specified position in this Vector.
boolean	`remove(Object o)` Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
boolean	`removeAll(Collection<?> c)` Removes from this Vector all of its elements that are contained in the specified Collection.
void	`removeAllElements()` Removes all components from this vector and sets its size to zero.
boolean	`removeElement(Object obj)` Removes the first (lowest-indexed) occurrence of the argument from this vector.
void	`removeElementAt(int index)` Deletes the component at the specified index.
boolean	`removeIf(Predicate<? super E> filter)` Removes all of the elements of this collection that satisfy the given predicate.
protected void	`removeRange(int fromIndex, int toIndex)` Removes from this list all of the elements whose index is between `fromIndex`, inclusive, and `toIndex`, exclusive.

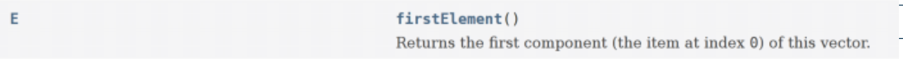
To get Objects:



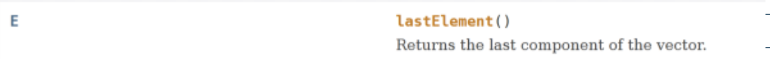
E	`get(int index)` Returns the element at the specified position in this Vector.



E	`elementAt(int index)` Returns the component at the specified index.



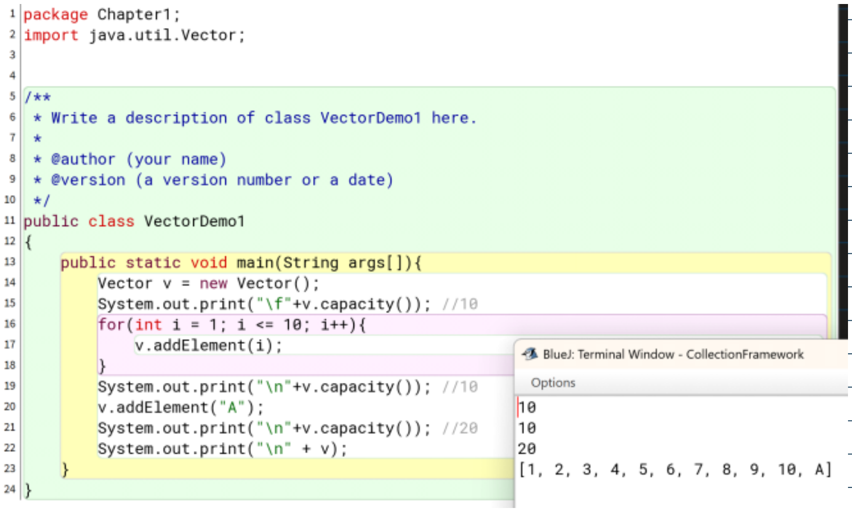
E	`firstElement()` Returns the first component (the item at index `0`) of this vector.



E	`lastElement()` Returns the last component of the vector.

Other methods:

```
int size();
int capacity();
Enumeration elements();
```



```java
package Chapter1;
import java.util.Vector;


/**
 * Write a description of class VectorDemo1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class VectorDemo1
{
    public static void main(String args[]){
        Vector v = new Vector();
        System.out.print("\f"+v.capacity()); //10
        for(int i = 1; i <= 10; i++){
            v.addElement(i);
        }
        System.out.print("\n"+v.capacity()); //10
        v.addElement("A");
        System.out.print("\n"+v.capacity()); //20
        System.out.print("\n" + v);
    }
}
```

```
10
10
20
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A]
```