Working with List<E> interface implemented classes :

As we know, in List interface we have 4 implemented classes which are as follows :

1) Vector<E>
2) Stack<E>
3) **ArrayList<E>**
4) **LinkedList<E>**

Vector<E>
----
public class Vector<E> extends AbstractList<E> implements List<E>, Cloneable, Serializable, RandomAccess

* It is a predefined **legacy class** available in java.util package from JDK 1.0V
* It is the implemented class of List<E> interface.
* It can accept duplicate, null, homogenous and hetrogeneous elements.
* It stores the elements based on the index position.
* Internally, It uses **dynamic array data structure.**
* Initial default capacity is 10.
* Next capacity will depend upon the formula :
  New Capacity = (Current Capacity * 2)

* Methods are synchronized so, performance wise not good.
* It is similar to ArrayList, only difference is, Vector methods are synchronized but ArrayList methods are not synchronized.
* It is mainly used to perform retrieval operation when thread safety is required.
* It implements from List, Cloneable, Serializable, RandomAccess.
* RandomAccess is a marker interaface available from JDK 1.4V, due to RandomAccess interface Vector can randomly fetch the element by using get(int index) method.
* Here Iteartor is **Fail Fast Iterator**.
* equals() and hashCode() method is overridden in the Vector class so It follows the contract.

Constructors of Vector :
----
* We have 4 types of Constructors :

1) Vector v1 = new Vector();
   It will create the vector object with default capacity is 10.

2) Vector v2 = new Vector(int initialCapacity);
   Will create the vector object with user specified capacity.

3) Vector v3 = new Vector(int initialCapacity, int capacityIncrement);
   Eg :- Vector v = new Vector(1000,5);

   Initially It will create the Vector Object with initial capacity 1000 and then when the capacity will be full then increment by 5 so the next capacity would be 1005, 1010 and so on.

4) Vector v4 = new Vector(Collection c);
   We can achieve loose coupling

//Programs :

```java
package com.ravi.vector;

import java.util.Vector;

public class VectorDemo
{
    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();
        listOfCity.add("Hyderabad");
        listOfCity.add("Pune");
        listOfCity.add("Indore");
        listOfCity.add("Bhubneswar");
        listOfCity.add("Kolkata");

        System.out.println("Before Sorting :"+listOfCity);


        System.out.println("Sorting the city Name in Ascending Order :");

        //Collections.sort(listOfCity);     [Old technique]
        //System.out.println(listOfCity);

        listOfCity.sort((s1,s2)-> s1.compareTo(s2));  //[New Technique]
        System.out.println(listOfCity);

        System.out.println("Remove the element using Object :");
        listOfCity.remove("Pune");
        System.out.println(listOfCity);

        System.out.println("Remove the element using index :");
        listOfCity.remove(2);
        System.out.println(listOfCity);
    }

}
```

```java
package com.ravi.vector;

//Vector program on capacity() method
import java.util.Vector;

public class VectorDemo1
{
    public static void main(String[] args)
    {
        Vector<Integer> v = new Vector<>(100, 10);

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

        for (int i = 0; i < 100; i++)
        {
            v.add(i);
        }

        System.out.println("After adding 100 elements  capacity is :" + v.capacity());

        v.add(101);
        System.out.println("After adding 101th elements  capacity is :" + v.capacity());

        for(int i=0; i<v.size(); i++)
        {
            if(i%5==0)
            {
                System.out.println();
            }
            System.out.print(v.get(i)+"\t");
        }


    }
}
```

Methods :
----
public Object toArray() : Collection to array
Collections.max()
Collections.min()
Collections.reverse()

```java
package com.ravi.vector;

import java.util.Arrays;
import java.util.Collections;
import java.util.Vector;

public class VectorDemo2
{
    public static void main(String args[])
    {
        Vector<Integer> v = new Vector<>();

        int x[]={22,20,10,40,15,58};

       //Adding array element into Vector

        for(int i=0; i<x.length; i++)
        {
            v.add(x[i]);
        }
        Collections.sort(v);
        System.out.println("Maximum element is :"+Collections.max(v));
        System.out.println("Minimum element is :"+Collections.min(v));
        System.out.println("Vector Elements :");

        v.forEach(y -> System.out.println(y));

        System.out.println(".....................");
        Collections.reverse(v);
        v.forEach(y -> System.out.println(y));

        //How to convert Collection to Array
        Object[] array = v.toArray();
        System.out.println(Arrays.toString(array));


    }
}
```

```java
package com.ravi.vector;

import java.util.Collections;
import java.util.Vector;

//To work with Custom Object

record Employee(Integer id, String name, Double salary)
{

}

public class VectorDemo3
{
    public static void main(String[] args)
    {
            Vector<Employee> listOfEmployees = new Vector<>();
            listOfEmployees.add(new Employee(333, "Scott", 800D));
            listOfEmployees.add(new Employee(555, "Smith", 1200D));
            listOfEmployees.add(new Employee(111, "Martin", 1500D));
            listOfEmployees.add(new Employee(222, "John", 1800D));
            listOfEmployees.add(new Employee(444, "Alen", 900D));

            System.out.println("Original Employee Data :");
            listOfEmployees.forEach(System.out::println);

            System.out.println("Sorting Employee Data based on the Id");
            Collections.sort(listOfEmployees,(e1, e2)-> e1.id().compareTo(e2.id()));

            System.out.println("Employee Data After sorting based on the Id:");
            listOfEmployees.forEach(System.out::println);
    }
}
```

```java
package com.ravi.vector;

import java.util.ArrayList;
import java.util.Vector;

public class VectorDemo4
{
    public static void main(String[] args)
    {
        long startTime = System.currentTimeMillis();

        ArrayList<Integer> al = new ArrayList<Integer>();

        for(int i=0; i<1000000; i++)
        {
            al.add(i);
        }

        long endTime = System.currentTimeMillis();

        System.out.println("Total Time taken by ArrayList class :"+(endTime -
startTime)+" ms");

       startTime = System.currentTimeMillis();

        Vector<Integer> v1 = new Vector<Integer>();

        for(int i=0; i<1000000; i++)
        {
            v1.add(i);
        }

        endTime = System.currentTimeMillis();

        System.out.println("Total Time taken by Vector class :"+(endTime - startTime)+"
ms");


    }
}
```

Note : From the above program, It is clear that performance wise ArrayList is more better than Vector.

```java
package com.ravi.vector;


import java.util.Arrays;
import java.util.Collections;
import java.util.Vector;

public class VectorDemo5
{

    public static void main(String[] args)
    {
        Vector<String> listOfCity = new Vector<>();
        listOfCity.add("Surat");
        listOfCity.add("Pune");
        listOfCity.add("Ahmadabad");
        listOfCity.add("Vanaras");

        Collections.sort(listOfCity);

        listOfCity.forEach(System.out::println);


        System.out.println(".............");

        Vector<Integer> listOfNumbers = new Vector<>();
        listOfNumbers.add(500);
        listOfNumbers.add(900);
        listOfNumbers.add(400);
        listOfNumbers.add(300);
        listOfNumbers.add(800);
        listOfNumbers.add(200);
        listOfNumbers.add(100);

        System.out.println("Original Data...");
        System.out.println(listOfNumbers);


        System.out.println("Ascending Order...");
        Collections.sort(listOfNumbers);
        System.out.println(listOfNumbers);

        System.out.println("Descending Order...");

        Collections.sort(listOfNumbers, ((i1, i2)-> Integer.compare(i2, i1)));
        System.out.println(listOfNumbers);

        //Converting Our Vector(Collection Object) into Array
        Vector<String> listOfFruits = new Vector<>();
        listOfFruits.add("Orange");
        listOfFruits.add("Apple");
        listOfFruits.add("Mango");

        Object[] fruits = listOfFruits.toArray();
        System.out.println(Arrays.toString(fruits));


    }

}
```

*** What is FAIL FAST ITERATOR ?

While retrieving the object from the collection by using Itearor interface or for each loop, if at any point of time the original structure is going to modify after the creation of Iearator then we will get java.util.ConcurrentModificationExacption.

```java
package com.ravi.vector;

import java.util.Iterator;
import java.util.Vector;

class Concurrent extends Thread
{
    private Vector<String> cities;

    public Concurrent(Vector<String> cities)
    {
        super();
        this.cities = cities;
    }

    @Override
    public void run()
    {
        try
        {
            Thread.sleep(2000);
        }
        catch(InterruptedException e)
        {
            System.err.println("Thread is interrupted");
        }

        cities.add("GOA");
    }
}



public class FailFastIterator
{

    public static void main(String[] args) throws InterruptedException
    {
        Vector<String> listOfCity = new Vector<>();
        listOfCity.add("Hyderabad");
        listOfCity.add("Pune");
        listOfCity.add("Indore");
        listOfCity.add("Bhubneswar");
        listOfCity.add("Kolkata");
        listOfCity.add("Chennai");

        Concurrent concurrent = new Concurrent(listOfCity);
        concurrent.start();

        Iterator<String> iterator = listOfCity.iterator();

        while(iterator.hasNext())
        {
            System.out.println(iterator.next());
            Thread.sleep(500);
        }

    }

}
```