# Collection Classes:

1. **What are the differences between List and Set?**

**------------------------------------------------------------------**

**Ans:**

**----**

1. **List is index based, it able to allow all the elements as per indexing.**

**Set is not index based, it able to allow all the elements on the basis of elements hashcode values.**

1. **List is able to allow duplicate elements.**

**Set is not allowing duplicate elements.**

1. **List is able to allow any no. of null values.**

**Set is able to allow only one null value.**

1. **List is following insertion order.**

**Set is not following insertion order bydefault.**

**Note: LinkedHashSet is following insertion order.**

1. **List is not following sorting order.**

**Sets are also not following sorting order bydefault.**

**Note: SortedSet, NavigableSet and TreeSet are following Sorting order.**

1. **List is able to allow heterogeneous elements.**

**Sets are able to allow hetergeneous elements bydefault.**

**Note: SortedSett, NavigableSet and TreeSet are allowing only Homogeneous elements.**

**List :**

**------**

**-->List is a direct child interface to Collection interface.**

**-->List was provided by JAVA along with JDK1.2 version**

**-->List is index based, it able to arrange all the elements as per indexing.**

**-->List is able to allow duplicate elements.**

**-->List is following insertion order.**

**-->List is not following Sorting order.**

**-->List is able to allow any no. of null values.**

**-->List is able to allow heterogeneous elements.**

**List interface has provided the following methods common to all of its implementation classes.**

**1.public void add(int index, Object obj)**

**-->It able to add the specified element at the specified index value.**

**2.public Object set(int index, Object obj)**

**-->It able to set the specified element at the specified index value.**

**Q)What is the difference between add(--) method and set(--) metod?**

**----------------------------------------------------------**

**Ans:**

**----**

**add(--) method is able to perform insert operation. if any element is existed at the specified index then add() method will insert the specified new element at the specified index value and add() method will adjust the existed element to next index value. If no element is existed at the specified index then add() method add the specified element at the specified index.**

**set(--) method is able to perform replace operation. If any element is existed at the specified index then set() method will remove the existed element and set(-) method will add the specified element to the specified index and set() method will return the removed element. If no element is existed at the specified index value then set() method will rise an exception like java.lang.indexOutOfBoundsException.**

1. **public Object get(int index)**

**-->It will return an element available at the specified index value.**

1. **public Object remove(int index)**

**-->It will remove and return an element available at the specified index value.**

1. **public int indexOf(Object obj)**

**-->It will return an index value where the first occurence of the specified element.**

**6.public int lastIndexOf(Object obj)**

**-->It will return an index value where the last occurence of the specified element.**

**EX:**

**---**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**ArrayList al=new ArrayList();**

**al.add("A");**

**al.add("B");**

**al.add("C");**

**al.add("D");**

**al.add("E");**

**System.out.println(al);**

**al.add(1,"X");**

**System.out.println(al);**

**al.add(6,"F");**

**System.out.println(al);**

**al.set(3,"Y");**

**System.out.println(al);**

**//al.set(7,"Z");--->IndexOutOfBoundsException**

**System.out.println(al.get(4));**

**System.out.println(al.remove(6));**

**System.out.println(al);**

**al.add(6,"X");**

**al.add(7,"B");**

**al.add(8,"X");**

**System.out.println(al);**

**System.out.println(al.indexOf("X"));**

**System.out.println(al.lastIndexOf("X"));**

**}**

**}**

**ArrayList:**

**--> It was provided by JAVA along with JDK1.2 version.**

**--> It is a direct implementation class to List interface.**

**--> It is index based.**

**--> It allows duplicate elements.**

**--> It follows insertion order.**

**--> It will not follow sorting order.**

**--> It allows heterogeneous elements.**

**--> It allows any no. of null values.**

**--> Its internal data structer is "Resizable Array".**

**--> Its initial capacity is 10 elements.**

**--> Its incremental capacity ration is**

**new\_Capacity=(Current\_Capacity\*3/2)+1**

**--> It is best option for frequent retrival operations.**

**--> It is not synchronized.**

**--> No method is synchronized method in ArrayList.**

**--> It allows more than one thread to access data.**

**--> It follows parallel execution.**

**--> It will reduce execution time.**

**--> It will improve application performance.**

**--> It will not give guarantee for data consistency.**

**--> It is not threadsafe.**

**--> It is not Legacy Collection.**

**Constructors:**

**-------------**

1. **public ArrayList()**

**-->It can be used to create an empty ArrayList object with 10 elements as default capacity value.**

**EX: ArrayList al=new ArrayList();**

**2.public ArrayList(int capacity)**

**-->It can be used to create an empty ArrayList object with the specified capacity.**

**EX: ArrayList al=new ArrayList(20);**

**3.public ArrayList(Collection c)**

**-->It can be used to create an ArrayList object with all the elements of the specified Collection object.**

**EX:**

**---**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**ArrayList al1=new ArrayList();**

**al1.add("AAA");**

**al1.add("BBB");**

**al1.add("CCC");**

**al1.add("DDD");**

**System.out.println(al1);**

**ArrayList al2=new ArrayList(al1);**

**System.out.println(al2);**

**}**

**}**

**OP:**

**---**

**[AAA,BBB,CCC,DDD]**

**[AAA,BBB,CCC,DDD]**

**EX:**

**----**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**ArrayList al=new ArrayList();**

**al.add("A");**

**al.add("B");**

**al.add("C");**

**al.add("D");**

**al.add("E");**

**System.out.println(al);**

**al.add("B");**

**System.out.println(al);**

**al.add(new Integer(10));**

**System.out.println(al);**

**al.add(null);**

**al.add(null);**

**System.out.println(al);**

**}**

**}**

**Vector:**

**-------**

**-->It was introduced in JDK1.0 version.**

**-->It is Legacy Collection.**

**-->It is a direct implementation class to List interface.**

**-->It is index based.**

**-->It allows duplicate elements.**

**-->It follows insertion order.**

**-->It will not follow sorting order.**

**-->It allows heterogeneous elements.**

**-->It allows any no. of null values.**

**-->Its internal data structer is "Resizable Array".**

**-->Its initial capacity is 10 elements.**

**-->It is best choice for frequent retrival operations.**

**-->It is not good for frequent insertions and deletion operations.**

**-->Its incremental capacity is double the current capacity.**

**New\_capacity=2\*Current\_Capacity**

**--> It is synchronized element.**

**--> All the methods of vector class are synchronized.**

**--> It allows only one thread at a time.**

**--> It follows sequential execution.**

**--> It will increase execution time.**

**-->It will reduce application performance.**

**--> It is giving guarantee for data consistency.**

**--> It is threadsafe.**

**Constructors:**

**--------------**

**1.public Vector()**

**--> It can be used to create an empty Vector object with the initial capacity 10 elements.**

**EX: Vector v=new Vector();**

**System.out.println(v.capacity());**

**OP: 10**

1. **public Vector(int capacity)**

**--> It can be used to create an empty vector object with the specified capacity value.**

**EX: Vector v=new Vector(20);**

**System.out.println(v.capacity());**

**OP: 20**

**EX:**

**----**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**Vector v=new Vector(5,5);**

**System.out.println(v.capacity());**

**for(int i=1;i<=6;i++)**

**{**

**v.add(i);**

**}**

**System.out.println(v.capacity());**

**for(int i=7;i<=11;i++)**

**{**

**v.add(i);**

**}**

**System.out.println(v.capacity());**

**}**

**}**

**OP:**

**---**

**5**

**10**

**15**

**4.public Vector(Collection c)**

**-->This constructor can be used to create Vector object with all the elements of the specified Collection object.**

**EX:**

**----**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**Vector v=new Vector();**

**v.add("A");**

**v.add("B");**

**v.add("C");**

**v.add("D");**

**System.out.println(v);**

**Vector v1=new Vector(v);**

**System.out.println(v1);**

**}**

**}**

**OP:**

**---**

**[A,B,C,D]**

**[A,B,C,D]**

**Methods:**

**---------**

**1.public void addElement(Object obj)**

**-->It will add the specified element to Vector.**

**2.public Object firstElement()**

**-->It will return first element of the Vector.**

**3.public Object lastElement()**

**-->It will return last element of the Vector.**

**4.public Object elementAt(int index)**

**-->It will return an element available at the specified index.**

**5.public void removeElement(Object obj)**

**-->It will remove the specified element from Vector.**

**6.public void removeElementAt(int index)**

**-->It will remove an element existed at the specified index value.**

**7.public void removeAllElements()**

**-->It will remove all elements from Vector.**

**EX:**

**---**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**Vector v=new Vector();**

**v.addElement("A");**

**v.addElement("B");**

**v.addElement("C");**

**v.addElement("D");**

**v.addElement("E");**

**System.out.println(v);**

**System.out.println(v.firstElement());**

**System.out.println(v.lastElement());**

**System.out.println(v.elementAt(3));**

**v.removeElement("D");**

**System.out.println(v);**

**v.removeElementAt(2);**

**System.out.println(v);**

**v.removeAllElements();**

**System.out.println(v);**

**}**

**}**

**Q)What are the differences between ArrayList and Vector?**

**--------------------------------------------------------**

**Ans:**

**----**

**1.ArrayList class was introduced in JDK1.2 version.**

**Vector class was introduced in JDK1.0 version.**

**2.ArrayList is not Legacy Collection.**

**Vector is Legacy Collection.**

**3.ArrayList is not synchronized.**

**Vector is synchronized.**

**4.No method is synchronized method in ArrayList.**

**Almost all the methods are synchronized methods in vector.**

**5.ArrayList allows more than one thread at a time to access data.**

**Vector allows only one thread at a time to access data.**

**6.ArrayList follows parallel execution.**

**Vector follows sequential execution.**

**7.ArrayList is able to reduce application execution time.**

**Vector is able to increase application execution time.**

**8.ArrayList is able to improve application performance.**

**Vector is able to reduce application performance.**

**9.ArrayList is not giving guarantee for data consistency.**

**vector is giving guarantee for data consistency.**

**10.ArrayList is not threadsafe.**

**Vector is threadsafe.**

1. **We are unable to get capacity value of ArrayList, because, no capacity() method in ArrayList class.**

**We can get capacity value of Vector, because, capacity() method is existed in vector class.**

# LinkedList:

**-----------**

**-->It was introduced in JDK1.2 version.**

**-->It is not Legacy Collection.**

**-->It is a direct implementation class to List interface.**

**-->It is index based.**

**-->It allows duplicate elements.**

**-->It follows insertion order.**

**-->It is not following sorting order.**

**-->It allows heterogeneous elements.**

**-->It allows null values in any number.**

**-->Its internal data structer is "Double Linked List".;**

**-->It is best choice for frequent insertions and deletions.**

**-->It is not synchronized Collection.**

**-->No method is synchronized in LinkedList.**

**-->It allows more than one thread to access data.**

**-->It will follow parallel execution.**

**-->It will decrese execution time.**

**-->It will improve application performance.**

**-->It is not giving guarantee for data consistency.**

**-->It is not threadsafe.**

**Constructors:**

**-------------**

1. **public LinkedList()**

**-->It will create an empty LinkedList object.**

**EX: LinkedList ll=new LinkedList();**

**2.public LinkedList(Collection c)**

**-->It will create LinkedList object with all the elements of the specified Collection object.**

**EX:**

**---**

**LinkedList ll1=new LinkedList();**

**ll1.add("A");**

**ll1.add("B");**

**ll1.add("C");**

**ll1.add("D");**

**System.out.println(ll1);**

**LinkedList ll2=new LinkedList(ll1);**

**System.out.println(ll2);**

**OP:[A, B, C, D]**

**[A, B, C, D]**

**Methods:**

**--------**

**1.public void addFirst(Object obj)**

**-->It will add the specified element as first element to LinkedList.**

**2.public void addLast(Object obj)**

**-->It will add the specified element as last element to LinkedList.**

**3.public Object getFirst()**

**-->It will return first element from LinkedList.**

**4.public Object getLast()**

**-->It will return last element from LinkedList.**

**5.public void removeFirst()**

**-->It will remove first element from LinkedList.**

**6.public void removeLast()**

**-->It will remove last element from LinkedList.**

**EX:**

**---**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**LinkedList ll=new LinkedList();**

**ll.add("A");**

**ll.add("B");**

**ll.add("C");**

**ll.add("D");**

**ll.add("E");**

**System.out.println(ll);**

**ll.addFirst("X");**

**ll.addLast("Y");**

**System.out.println(ll);**

**ll.removeFirst();**

**ll.removeLast();**

**System.out.println(ll);**

**System.out.println(ll.getFirst());**

**System.out.println(ll.getLast());**

**}**

**}**

**Cursors / Iterators in Collections**

**----------------------------------**

**In java applications, when we pass Collection object reference variable as parameter to System.out.println(-) method, then, JVM will execute toString() method internally. Initially toString() method was implemented in java.lang.Object class, it was implemented in such a way that to return a String contains "Class\_Name@Ref\_val" . In java applications, Collection classes are not depending on Object class toString() method, they are having their own toString() method , which are implemented in such a way to return a String contains all the elements of the Collection object by enclosed with [].**

**EX:**

**---**

**ArrayList al=new ArrayList();**

**al.add("A");**

**al.add("B");**

**al.add("C");**

**al.add("D");**

**System.out.println(al);**

**OP: [A, B, C, D]**

**As per the requirement, we dont want to display all the Elements at a time on command prompt, we want to retrive elements one by one individually from Collection objects and we want to display all the elements one by one on Command prompt.**

**To achieve the above requirment, Collection Framework has provided the following three Cursors or Iterators.**

**1.Enumeration**

**2.Iterator**

**3.ListIterator**

**1.Enumeration**

**-------------**

**It is a Legacy Cursor, it is applicable for only Legacy Collections to retrieve elements in one by one fashion.**

**To retrieve elements from Collections by using Enumeration we have to use the following steps.**

1. **Create Enumeration object:**

**To create Enumeration object we have to use the following method from Legacy Collections.**

**public Enumeration elements()**

1. **Retrieve elements from Enumeration:**
2. **Check whether more elements are available or not from Current cursor position by using the following method.**

**public boolean hasMoreElements()**

**--> It will return true value if atleast next element is existed.**

**-->It will return false value if no element is existed from current cursor position.**

1. **If atleast next element is existed then read next element and move cursor to next position by using the following method.**

**public Object nextElement()**

**EX:**

**---**

**import java.util.\*;**

**class Test**

**{**

**public static void main(String[] args)**

**{**

**Vector v=new Vector();**

**v.add("A");**

**v.add("B");**

**v.add("C");**

**v.add("D");**

**v.add("E");**

**System.out.println(v);**

**Enumeration e=v.elements();**

**while(e.hasMoreElements())**

**{**

**System.out.println(e.nextElement());**

**}**

**}**

**}**

**Drawbacks:**

**-----------**

**1.Enumeration is applicable for only Legacy Collections.**

**2.Enumeration is able to allow only read operation while iterating elements.**