Collectionsframework (java.util)

Pre-requisite topics for Collections framework:-

1) ***Arrays***
2) ***toString() method.***
3) ***type-casting.***
4) ***interfaces.***
5) ***for-each loop.***
6) ***implementation classes.***
7) ***compareTo() method.***
8) ***Wrapper classes.***
9) ***Marker interfaces advantages.***
10) ***Anonymous inner classes.***
11) ***For-each loop***
12) ***Auto-boxing***

Importance of collections:-

- The main objective of collections framework is to represent group of object as a single entity.
- In java Collection framework provide very good architecture to store and manipulate the group of objects.
- Collection API contains group of classes and interfaces that makes it easier to handle group of objects.
- Collections are providing flexibility to store, retrieve, and manipulate data.

The key interfaces of collection framework:-

1. ***Java.util.Collection***
2. ***Java.util.List***
3. ***Java.util.*Set**
4. ***Java.util.*SortedSet**
5. ***Java.util.*NavigablaSet**
6. ***Java.util.*Queue**
7. ***Java.util.*Map**
8. ***Java.util.*SotedMap**

9. ***Java.util.*NavigableMap**
10. ***Map.Entry***
11. ***Java.util.Enumeration***
12. ***Java.util.Iterator***
13. ***Java.util.ListIterator***
14. ***Java.lang.Comparable***
15. ***Java.util.Comparator***

- All collection framework classes and interfaces are present in ***java.util*** package.
- The root interface of Collection framework is **Collection.**
- Collection interface contains 15 methods so all collection implementation classes are able to use these methods because collections is a root interface.

Collection interface methods:-

To check the predefined support use javap command.

>javap java.util.Collection

```
public abstract int size();
public abstract boolean isEmpty();
public abstract boolean contains(java.lang.Object);
public abstract java/util/Iterator<E> iterator();
public abstract java.lang.Object[] toArray();
public abstract <T extends java/lang/Object> T[] toArray(T[]);
public abstract boolean add(E);
public abstract boolean remove(java.lang.Object);
public abstract boolean containsAll(java/util/Collection<?>);
public abstract boolean addAll(java/util/Collection<? extends E>);
public abstract boolean removeAll(java/util/Collection<?>);
public abstract boolean retainAll(java/util/Collection<?>);
public abstract void clear();
public abstract boolean equals(java.lang.Object);
public abstract int hashCode();
```

Iinterface contains abstract method and for that interfaces object creation is not possible hence think about implementation classes of that interfaces.

Collection vs Collections:-

Collection is interface it is used to represent group of objects as a single entity. Collections is utility class it contains methods to perform operations.

Arrays vs Collections:-

Both Arrays and Collections are used to represent groupof objects as a single entity but the differences are as shown below.

Limitations of Arrays

1) *Arrays are used to represent group of objects as a single entity.*
2) *Arrays are used to store homogeneous data(similar data).*
3) *Arrays are capable to store primitive & Object type data*
4) *Arrays are fixed in size, it means once we created array it is not possible to increase & decrease the size based on our requirement.*
5) *With respect to memory arrays are not recommended to use.*
6) *If you know size in advance arrays are recommended to use because it provide good performance.*
7) *Arrays does not contains underlying Data structure hence it is not supporting predefined methods.*
8) *While working with arrays operations(add,remove,update…) are become difficult because it is not supporting methods.*

Advantages of Collections

1) *Collections are used to represent group of objects as a single entity.*
2) *Collections are used to store both heterogeneous data(different type)& homogeneous data.*
3) *Collections are capable to store only object data.*
4) *Collections are growable in nature, it means based on our requirement it*

is possible to increase & decrease the size.

5) *With respect to memory collections are recommended to use.*

6) *In performance point of view collections will give low performance compare to arrays.*

7) *Collection classes contains underlying data structure hence it supports predefined methods.*

8) *Here operations are become easy because collections supports predefined methods.*

Characteristics of Collection frame work classes:-

The collections framework contains group of classes but every class is used to represent group of objects as a single entity but characteristics are different.

1) ***The collect ion framework classes introduced Versions***

Different classes are introduced in different versions.

2) ***Heterogeneous data allowed or not allowed.***

All most all collection framework classes allowed heterogeneous data except two classes

i. TreeSet ii. TreeMap

3) ***Null insertion is possible or not possible.***

Some classes are allowed null insertion but some classes are not allowed.

4) ***Duplicate objects are allowed or not allowed.***

Inserting same object more than one time is called duplication. Some classes are allowed duplicates but some classes are not allowed duplicates.

add(e1)
add(e1)

5) ***Insertion order is preserved or not preserved.***

In which order we are inserting element same order output is printed then say insertion order is preserved otherwise not.

Input --->e1 e2 e3 output --->e1 e2 e3 insertion order is preserved
Input --->e1 e2 e3 output --->e2 e1 e3 insertion order is not-preserved

6) ***Collection classes' methods are synchronized or non-synchronized.***

If the methods are synchronized only one thread is allow to access, these methods are thread safe but performance is reduced.
If the methods are non-synchronized multiple threads are able to access, these methods are not thread safe but performance is increased.

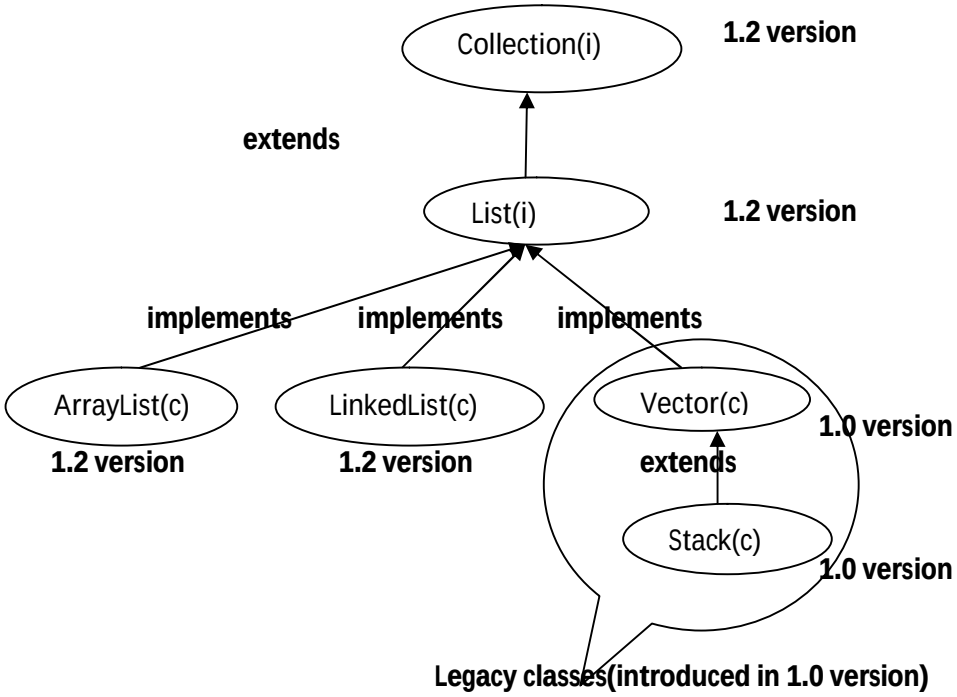
7) ***Collection classes underlying data structures.***

Arrays are does not contains underlying data structure hence it is not supporting predefined methods.
Every collection class contains underlying data structure hence it supports predefined methods.
Based on underlying data structure the element will be stored.

8) Collection classes supported cursors.

The collection classes are used to represent group of objects as a single entity & To retrieve the objects from collation class we are using cursors.

List interface:-



I = Interfacec=class

Implementation classes of List interface :-

1) *ArrayList*
2) *LinkedList*
3) *Vector*
4) *Stack*

Legacy classes:-

*The java classes which are introduced in 1.0 version are called legacy classes and **java.util** package contains 5 legacy classes.*

1) *Dictionary*
2) *HashTable*
3) *Properties*
4) *Stack*
5) *Vector*

Legacy interfaces:-

The java interfaces which are introduced in 1.0 version are called legacy interfaces and ***java.util*** *package contains only one interface is* ***Enumeration****.*

List interface common properties:-

1) *All list class allows heterogeneous data.*
2) *All List interface implementation classes allows null insertion.*
3) *All classes allows duplicate objects.*
4) *All classes preserved insertion order.*

Java.util.ArrayList:-

To check parent class and interface use below command.

D:\ratan>javap java.util.ArrayList

public class java.util.ArrayList<E>
extends *java.util.AbstractList<E>*
implements *java.util.List<E>,*
java.util.RandomAccess,
java.lang.Cloneable,
java.io.Serializable

ArrayList Characteristics:-

1) ArrayList Introduced in 1.2 version.
2) ArrayList stores Heterogeneous objects(different types).
3) In ArrayList it is possible to insert **Null** objects.
4) Duplicate objects are allowed.
5) ArrayList preserved Insertion order it means whatever the order we inserted the data in the same way output will be printed.
6) ArrayList methods are non-synchronized methods.
7) The under laying data structure is growable array.
8) By using cursor we are able to retrieve the data from ArrayList : ***Iterator , ListIterator***

Constructors to create ArrayList:-

Constructor-1:-

ArrayList al = new ArrayList();

The default capacity of the ArrayList is 10 once it reaches its maximum capacity then size is automatically increased by ***New capacity = (old capacity*3)/2+1***

Constructor-2:-

ArrayList al = new ArrayList (int initial-capacity);

It is possible to create ArrayList with initial capacity

Constructor-3:-

ArrayList al = new ArrayList(Collection c);

Adding one collection data into another collection(Vector data intoArrayList) use fallowing constructor.

Example :-

Collections vs Autoboxing

Up to 1.4 version we must create wrapper class object then add that object into ArrayList.

```
import java.util.ArrayList;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                Integer i = new Integer(10);            //creation of Integer Object
                Character ch = new Character('c');      //creation of Character Object
                Double d = new Double(10.5);            //creation of Double  Object
                //adding wrapper objects into ArrayList
                al.add(i);
                al.add(ch);
                al.add(d);
                System.out.println(al);

        }
}
```

From 1.5 version onwards add the primitive data into ArrayList that data is automatically converted into wrapper object format is called Autoboxing.

Code before compilation:-

```
import java.util.ArrayList;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(10); //AutoBoxing
                al.add('a'); //AutoBoxing
                al.add(10.5);//AutoBoxing
                System.out.println(al);
        }
}
```

Code after compilation:-

```
import java.io.PrintStream;
import java.util.ArrayList;
class Test
{  Test()
   {
   }
   public static void main(String args[])
   {  ArrayList arraylist = new ArrayList();
      arraylist.add(Integer.valueOf(10));
      arraylist.add(Character.valueOf('a'));
      arraylist.add(Double.valueOf(10.5));
      System.out.println(arraylist);
   }
}
```

*Example-2:-**ArrayList vs toString()***

Emp.java:-

```
class Emp
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
}
```

Student.java

```
class Student
{       int sid;
        String sname;
        Student(int sid,String sname)
        {       this.sid=sid;
                this.sname = sname;
        }
}
```

Case 1:-

In java when we print reference variable internally it calls toString() method on that object.

```
import java.util.ArrayList;
class Test
{       public static void main(String[] args)
        {       Emp e1 = new Emp(111,"ratan");
                Student s1 = new Student(222,"xxx");
                ArrayList al = new ArrayList();
                al.add(10);             //toString()
                al.add('a');            //toString()
                al.add(e1);             //toString()
                al.add(s1);             //toString()
                System.out.println(al);                 //[10, a, Emp@d70d7a, Student@b5f53a]
                System.out.println(al.toString());      //[10, a, Emp@d70d7a, Student@b5f53a]
        }
}
```

Case2:-

```
import java.util.ArrayList;
class Test
{       public static void main(String[] args)
        {       Emp e1 = new Emp(111,"ratan");
                Student s1 = new Student(222,"xxx");
                ArrayList al = new ArrayList();
                al.add(10);
                al.add('a');
                al.add(e1);
                al.add(s1);
                System.out.println(al.toString());//[10, a, Emp@d70d7a, Student@b5f53a]
                for (Object o : al)
                {       if (o instanceof Integer)
                        System.out.println(o.toString());
                        if (o instanceof Character)
                        System.out.println(o.toString());
                        if (o instanceof Emp){
                                Emp e = (Emp)o;
                                System.out.println(e.eid+"---"+e.ename);
                        }
                        if (o instanceof Student){
                                Student s = (Student)o;
                                System.out.println(s.sid+"---"+s.sname);
                        }
                }
        }
}
```

Example:- Basic operations of ArrayList

add() *to add the objects into ArrayList & bydefault it add the data at last but it is possible to insert the data at specified index.*

remove()*it removes Objects from ArrayList based on Object & index.*
(for the remove(10) method if we are passing integer value that is always treted as index)

size() *to find the size of ArrayList.*

isEmpty()*to check the objects are available or not.*

Clear()*to remove all objects from ArrayList.*

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al =new ArrayList();
                al.add(10);
                al.add("ratan");
                al.add("anu");
                al.add('a');
                al.add(10);
                al.add(null);
                System.out.println("ArrayList data="+al);
                System.out.println("ArrayList size-->"+al.size());
                al.add(1,"A1");                 //add the object at first index
                System.out.println("after adding objects ArrayList size-->"+al.size());
                System.out.println("ArrayList Data="+al);
                al.remove(1);           //remove the object index base
                al.remove("A");         //remove the object on object base
                System.out.println("after removeing elemetns arrayList size "+al.size());
                System.out.println("ArrayList data="+al);
                System.out.println(al.isEmpty());
                al.clear();
                System.out.println(al.isEmpty());
        }
}
```

```
E:\>java Test
ArrayList data=[10, ratan, anu, a, 10, null]
ArrayList size-->6
after adding objects ArrayList size-->7
ArrayList Data=[10, A1, ratan, anu, a, 10, null]
after removeing elemetns arrayList size 6
ArrayList data=[10, ratan, anu, a, 10, null]
false
true
```

observation:-

in above example when we remove the data by passing numeric value that is by default treated as a indexvalue.

```
ArrayList al = new ArrayList();
al.add(10);
al.add("ratan");
```

```
al.add('a');
System.out.println(al);
al.remove(10);   // 10 is taken as index value
```

whenever we are executing above code then JVM treats that 10 is index value but 10th position value is not available hence it is generating exception ***java.lang.IndexOutOfBoundsException: Index: 10, Size: 3***

in above case to remove ***10*** *Integer object then use below code.*

```
ArrayList al = new ArrayList();
Integer i = new Integer(10);
al.add(i);
al.remove(i);
System.out.println(al);
```

<u>All collection classes are having 2-formats:-</u>

1) *Normal version (no type safety).*
2) *Generic version. (provide type safety)*

The main purpose of the generics is to provide the type safety to avoid type casting problems.

<u>Arrays [Type safety]:-</u>

Arrays are always type safe it means we can give guarantee for the type of element present in arrays.

For example if we want to store String objects create String[]. By mistake if we are trying to add any other data compiler will generate compilation error.

Example:-

```
String[] str= new String[50];
str[0]="ratan";
str[1]="anu";
str[2]=new Integer(10);
```

E:\>javac Test.java

```
incompatible types
 str[2]=new Integer(10);
required: String
found:    Integer
```

Based on above error we can give guarantee String[] is able to store only String type of objects. Hence with respect to the type arrays are recommended to use because it is type safe.

<u>[Collection]Not type safe:-</u>

Collections are not type safe it means we can't give guarantee for the type of elements present in collection.

If programming requirement is to hold the String type of the objects we are choosing ArrayList , by mistake if we are adding any type of data then compiler is unable to generate compilation error but runtime program is failed.

Example :-

```
ArrayList al = new ArrayList();
al.add("ratan");
al.add("anu");
al.add(new Integer(10)); //allowed
String s1 = (String)al.get(0);
String s2 = (String)al.get(1);
String s3 = (String)al.get(2);//java.lang.ClassCastException
```

Based on above exception we can decide collection is not type safe.

Example :-

```
ArrayList al = new ArrayList();
al.add("ratan");
String str = l.get(0); // java.lang.ClassCastException
String str =(String l.get(0); //type casting is mandatory
```

In above example type-casting is mandatory it is bigger problem in collection.

To overcome above problems use generics,

1) **To provide type safety.**
2) **To overcome type casting problems.**

in java it is recommended to use generic version of collections to provide the type safety.

Syntax:-

ArrayList**<type-name> al =** new ArrayList**<type-name>();**

The ArrayList is able to store only String data if we are trying to add any other data compiler will generate error message.

Example :-

```
ArrayList<String> al = new ArrayList<String>();
al.add("ratan");
al.add("anu");
al.add(new Integer(10)); //compilation error
```

if we are using generics we will get type safety. At the time retrieval not required to perform type casting.

```
ArrayList<String> al = new ArrayList<String>();
al.add("ratan");
String s1 = al.get(0);
```

Normal version of ArrayList(no type safety)

1) Normal version is able to hold any type of data(heterogeneous data) hence it is not a type safe.

```
ArrayList al = new ArrayList();
al.add(10);
al.add('a');
System.out.println(al);
```

2) At the time of retrieval Always check the type of the object by using **instanceof** operator.

3) In normal it is holding different types of data hence while retrieving data must perform **type casting.**

4) If we are using normal version while compilation compiler generate worning message like**unchecked or unsafe operations.**

Example:- normal version of ArrayList

```
import java.util.*;
class Test
{   public static void main(String[] args)
    {       ArrayList al = new ArrayList();
            al.add(10);
```

```
        al.add('a');
        al.add(10.4);
        System.out.println(al);
    }
}
```

Generic version of ArrayList(type safety)

1) Generic version is able to hold specified type of data hence it is a type safe.

ArrayList<tye-name> al = new ArrayList<type-name>( );

```
ArrayList<Integer> al = new ArrayList<Integer>();
    al.add(10);
    al.add(20);
    al.add("ratan");//compilation error
    System.out.println(al);
```

2) Type checking is not required because it contains only one type of data.

3) It is holding specific data hence at the time of retrieval type casting is not required.

4) If we are using generic version compiler won't generate worning messages.

Example :- generic version of ArrayList.

```
import java.util.*;
class Test
{   public static void main(String[] args)
{ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(10);
        al.add(20);
        al.add(30);
        al.add(40);
        System.out.println(al);
            }
        }
```

Example :- retrieving data from generic version of ArrayList.

```
import java.util.*;
class Test
{   public static void main(String[] args)
    {   ArrayList<Emp> al = new ArrayList<Emp>();
        al.add(new Emp(111,"ratan"));
        al.add(new Emp(222,"anu"));
        al.add(new Emp(333,"Sravya"));
        for (Emp e : al)
        {   System.out.println(e.eid+"---"+e.ename);
        }
    }
}
```

Example :-

add(E); ***---->to add the Object.***
remove(java.lang.Object); ***----->to remove the object.***
addAll(); ***------>to add one collection object into another collection.***
contains() ***----->to check object is available or not.***
containsAll() ***---->to check entire collection data is available or not.***

Example:-

```
import java.util.*;
class Test
{   public static void main(String[] args)
    {   Emp e1 = new Emp(111,"ratan");
        Emp e2 = new Emp(222,"Sravya");
        Emp e3 = new Emp(333,"aruna");
        Emp e4 = new Emp(444,"anu");

        ArrayList<Emp> a1 = new ArrayList<Emp>();
        a1.add(e1);
```

```
                a1.add(e2);

                ArrayList<Emp> a2 = new ArrayList<Emp>();
                a2.addAll(a1);
                a2.add(e3);
                a2.add(e4);

                System.out.println(a2.contains(e1));
                System.out.println(a2.containsAll(a1));
                a2.remove(e1);
                System.out.println(a2.contains(e1));
                System.out.println(a2.containsAll(a1));
                //printing the data
                for (Emp e:a2)
                {       System.out.println(e.eid+"---"+e.ename);
                }
        }
}
E:\>java Test
true
true
false
false
222---Sravya
333---aruna
444---anu
```

Example :-

removeAll(Obj):-

*a2.removeAll(a1); // it removes all **a1** data.*

retainAll(Obj):-

*a2.retainAll(a1); // it removes all **a2** data except **a1***

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<Emp> a1 = new ArrayList<Emp>();
                a1.add(new Emp(111,"ratan"));
                a1.add(new Emp(222,"Sravya"));

                ArrayList<Emp> a2 = new ArrayList<Emp>();
                a2.addAll(a1);
                a2.add(new Emp(333,"aruna"));
                a2.add(new Emp(444,"anu"));

                a2.removeAll(a1);
                a2.retainAll(a1);
                for (Emp e:a2)
```

```
                {       System.out.println(e.eid+"---"+e.ename);
                }
        }
}
```

Creation of sub ArrayList & swapping data :-

Create sub ArrayList by using ***subList(int,int)*** *method of ArrayList.*

public java.util.List<E> subList(int, int);

to swap the data from one index position to another index position then use ***swap()****method of Collections class.*

public static void swap(java.util.List<?>, int, int);

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> a1 = new ArrayList<String>();
                a1.add("ratan");
                a1.add("anu");
                a1.add("Sravya");
                a1.add("yadhu");
                ArrayList<String> a2 = new ArrayList<String>(a1.subList(1,3));
                System.out.println(a2);         //[anu,Sravya]
                ArrayList<String> a3 = new ArrayList<String>(a1.subList(1,a1.size()));
                System.out.println(a3);         //[anu,Sravya,yadhu]

                //java.lang.IndexOutOfBoundsException: toIndex = 7
                //ArrayList<String> a4 = new ArrayList<String>(a1.subList(1,7));
                System.out.println("before swapping="+a1);//[ratan, anu, Sravya, yadhu]
                Collections.swap(a1,1,3);
                System.out.println("after swapping="+a1);// [ratan, yadhu, Sravya, anu]
        }
}
```

ArrayList Capacity:-

```
import java.util.*;
import java.lang.reflect.Field;
class Test
{       public static void main(String[] args)throws Exception
        {       ArrayList<Integer> al  = new ArrayList<Integer>(5);
                for (int i=0;i<10 ;i++)
                {       al.add(i);
                        System.out.println("size="+al.size()+" capacity="+getcapacity(al));
                }
        }
        static int getcapacity(ArrayList l)throws Exception
        {       Field f = ArrayList.class.getDeclaredField("elementData");
```

```
                f.setAccessible(true);
                return ((Object[])f.get(l)).length;
        }
}
D:\>java Test
size=1 capacity=5
size=2 capacity=5
size=3 capacity=5
size=4 capacity=5
size=5 capacity=5
size=6 capacity=8
size=7 capacity=8
size=8 capacity=8
size=9 capacity=13
size=10 capacity=13
```

Different ways to initialize values to ArrayList:-

Case 1:initializing ArrayList by using asList()

```
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>(
                Arrays.asList("ratan","Sravya","anu"));
                System.out.println(al);
        }
}
```

Case 2:- adding objects into ArrayList by using anonymous inner classes.

```
import java.util.ArrayList;
class ArrayListDemo
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>()
                {       {add("anu");
                         add("ratan");
                        }
                };//semicolan is mandatory
                System.out.println(al);
        }
}
```

Case 3:- normal approach to initialize the data

```
import java.util.ArrayList;
class ArrayListDemo
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
```

```
                al.add("anu");
                al.add("Sravya");
                System.out.println(al);
        }
}
```

Case 4:-

ArrayList<Type> obj = new ArrayList<Type>(Collections.nCopies(count, object));

```
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       Emp e1 = new Emp(111,"ratan");
                ArrayList<Emp> al = new ArrayList<Emp>(Collections.nCopies(5,e1));
                for (Emp e:al)
                {       System.out.println(e.ename+"---"+e.eid);
                }
        }
}
```

Case 5:-adding Objects into ArrayList by using addAll() method of Collections class.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                String[] strArray={"ratan","anu","Sravya"};
                Collections.addAll(al,strArray);
                System.out.println(al);
        }
}
```

Q. ***How to get synchronized version of ArrayList?***

Ans:- by default ArrayList methods are synchronized but it is possible to get synchronized version of ArrayList by using fallowing method.

To get synchronized version of List interface use fallowing Collections class static method

public static List synchronizedList(List l)

To get synchronized version of Set interface use fallowing Collections class static method

public static Set synchronizedSet(Set s)

To get synchronized version of Map interface use fallowing Collections class static method

public static Map synchronized Map(Map m)

to get synchronized version of TreeSet use fallowing Collections class static method

Collections.synchronizedSortedSet(SortedSet<T> s)

to get synchronized version of TreeMap use fallowing Collections class static method

Collections.synchronizedSortedMap(SortedMap<K,V> m)

Example:-

```
ArrayList al = new ArrayList();                 //non- synchronized version of ArrayList
List l = Collections.synchronizedList(al);      // synchronized version of ArrayList
```

Conversion of Arrays to ArrayList & ArrayList to Arrays:

Example-1:

Conversion of String array to ArrayList (by using asList() method):-

```
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       String[] str={"ratan","Sravya","aruna"};
                ArrayList<String> al = new ArrayList<String>(Arrays.asList(str));
                al.add("newperson-1");
                al.add("newperson-2");
                //printing data by using enhanced for loop
                for (String s: al)
                {       System.out.println(s);
                }
        }
}
```

Example-2:-

Conversion of ArrayList to String array by using toArray(T)

public abstract <T extends java/lang/Object> T[] toArray(T[]);

```
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       //interface ref-var & implementaiton  class Object
                List<String> al = new ArrayList<String>();
```

```
                al.add("anu");
                al.add("Sravya");
                al.add("ratan");
                al.add("natraj");
                String[] a = new String[al.size()];
                al.toArray(a);
                //for-each loop to print the data
                for (String s:a)
                {System.out.println(s);
                }
        }
}
```

Example-3:-

Case-1 :- conversion of ArrayList to Array

public abstract java.lang.Object[] toArray();

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(10);
                al.add('c');
                al.add("ratan");
                //converison of ArrayList to array
                Object[] o = al.toArray();
                for (Object oo :o)
                {       System.out.println(oo);
                }
        }
}
```

Case-2 :-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(new Emp(111,"ratan"));
                al.add(new Student(1,"xxx"));
                al.add("ratan");
```

```
                ***//converison of ArrayList to array***
                Object[] o = al.toArray();
                for (Object oo :o)
                {       if (oo instanceof Emp)
                        {       Emp e = (Emp)oo;
                                System.out.println(e.eid+"---"+e.ename);
                        }
                        if (oo instanceof Student)
                        {       Student s = (Student)oo;
                                System.out.println(s.sid+"---"+s.sname);
                        }
                        if (oo instanceof String)
                        {       System.out.println(oo.toString());
                        }
                }
        }
}
```

Example : Reverse of Collection data by using reverse() method Collections class.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(10);
                al.add("ratan");
                al.add(30);
                al.add("anu");
                Collections.reverse(al);
                System.out.println(al);
        }
};
```

Cursors in Collections:

Property

1. ***Purpose***
2. ***Legacy or not***
3. ***Applicable for which type of classes***
4. ***Universal cursor or not***
5. ***How to get the object***
6. ***How many methods***
7. ***Operations***
8. ***Cursor moment***
9. ***Class or interface***
10. ***Versions supports***

6) It contains two methods
hasMoreElements(): to check objects.
nextElement() : to retrieve the objects.

7) Only read operations.

8) Only forward direction.

Enumeration

1) *Used to retrieve the data from collection classes.*

2) *Introduced in 1.0 version it is legacy*

3) It is used to retrieve the data from only legacy classes like vector, Stack…etc

4) Not a universal cursor because it is applicable for only legacy classes.

5) Get the Enumeration Object by using elements() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
   Enumeration e = v.elements();
```

9) Interface

10) It supports both normal and generic version.

Iterator

1) *Used to retrieve the objects from collection classes.*

2) *Introduced in 1.2 version it is not a legacy*

3) It is used to retrieve the data from all collection classes.

4) It is a universal cursor because it is applicable for all collection classes.

5) Get the iterator Object by using iterator() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
   Enumeration e = v.iterator();
```

6) It contains two methods
hasNext(): to check the objects available or not.
Next() : to retrieve the objects.

7) read & remove operations are possible.

8) Only forward direction.

9) Interface

10) It supports both normal and generic version.

ListIterator

1) *Used to retrieve the data from collection classes.*

2) *Introduced in 1.2 version it is not a legacy*

3) It is used to retrieve the data from only List type of classes like ArrayList,LinkedList,Vector,Stack.

4) Not a universal cursor because it is applicable for only List interface classes.

5) Get the ListIterator Object by using listIterator() method.

```
Vector v =new Vector();
        v.add(10);
        v.add(20);
    Enumeration e = v.listIterator();
```

6) It contains 9 methods

7) Read, remove, add, and replace operations.

8) Bidirectional cursor direction.

9) Interface

10) It supports both normal and generic version.

ListIterator methods:-

```
public abstract boolean hasNext();
public abstract E next();
public abstract boolean hasPrevious();
public abstract E previous();
public abstract int nextIndex();
public abstract int previousIndex();
public abstract void remove();
public abstract void set(E);//replacement
public abstract void add(E);
```

Retrieving objects of collections classes:-

We are able to retrieve the objects from collection classes in 3-ways

1) **By using for-each loop.**
2) **By using get() method.**
3) **By using cursors.**

Example application:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al =new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("sravya");
                //1st appraoch to print Collection data
                for (String  a  :  al )
                {       System.out.println(a);
                }

                //2nd approach to print Collection data
                int size = al.size();
```

```
        for (int i=0;i<size;i++)
        {       System.out.println(al.get(i));
        }

        //3rd approach to print Collection data
        //normal version of Iterator(type casting required at the time of retrieving)
        Iterator itr1 = al.iterator();
        while (itr1.hasNext())
        {       String str =(String)itr1.next();
                System.out.println(str);
        }
        //generic  version of Iterator(type casting not required at the time of retrieving)
        Iterator<String> itr2 = al.iterator();
        while (itr2.hasNext())
        {       String str =itr2.next();
                System.out.println(str);
        }
    }
}
```

Example:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al =new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("sravya");
                ListIterator<String> lstr = al.listIterator();
                lstr.add("suneel");
                while(lstr.hasNext())
                {       if ((lstr.next()).equals("anu"))
                        {       lstr.set("Anushka");
                        }
                }
                lstr.add("aaa");
                for (String str:al)
                {       System.out.println(str);        }
        }
}
```

E:\>java Test

suneel
ratan
Anushka
sravya
aaa

if we want remove the data:-

```
import java.util.*;
class Test
{	public static void main(String[] args)
	{	ArrayList<String> al =new ArrayList<String>();
		al.add("ratan");
		al.add("anu");
		al.add("sravya");
		ListIterator<String> lstr = al.listIterator();
		while(lstr.hasNext())
		{	if ((lstr.next()).equals("ratan"))
			{	lstr.remove();
			}
		}
		for (String str:al)
		{	System.out.println(str);
		}
	}
}
```

E:\>java Test
anu
sravya

Example:-printing data in forward and backward directions.

```
import java.util.*;
class Test
{	public static void main(String[] args)
	{	ArrayList<String> al =new ArrayList<String>();
		al.add("ratan");
		al.add("anu");
		al.add("sravya");
		ListIterator<String> lstr = al.listIterator();
		System.out.println("printing data forward direction");
		while(lstr.hasNext())
		{	System.out.println(lstr.next());
		}
		System.out.println("printing data backward direction");
		while(lstr.hasPrevious())
		{	System.out.println(lstr.previous());
		}
	}

}
```

```
E:\>java Test
printing data forward direction


ratan
anu
sravya
printing data backword
direction sravya
anu
ratan
```

Sorting data by using sort() method of Collections class:-

we are able to sort ArrayList data by using sort() method of Collections class and by default it perform ascending order .

public static <T extends java/lang/Comparable<? super T>> void sort(java.util.List<T>);

if we want to person ascending order your class must implements Comparable interface of java.lang package.

If we want to perform descending order use ***Collections.reverseOrder()*** *method along with* ***Collection.sort()*** *method.*

Collections.sort(list ,Collections.reverseOrder());

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("Sravya");
```

```
            //printing ArrayList data
            System.out.println("ArrayList data before sorting");
            for (String str :al)
            {       System.out.println(str);
            }
            //sorting ArrayList in ascending order
            Collections.sort(al);
            System.out.println("ArrayList data after sorting ascending order");
            for (String str1 :al)
            {       System.out.println(str1);
            }
            //sorting ArrayList in decending order
            Collections.sort(al,Collections.reverseOrder());
            System.out.println("ArrayList data after sorting descending order");
            for (String str2 :al)
            {       System.out.println(str2);
            }

        }
}
```

Example:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("ratan");
                al.add("anu");
                al.add("Sravya");
                Collections.sort(al);
                System.out.println("ArrayList data after sorting");
                for (String str1 :al)
                {       System.out.println(str1);
                }
        }
}
```

- ✓ *in above example to perform the sorting of data by using natural sorting order then your objects must be homogeneous and must implements comparable interface.*
- ✓ *The default natural sorting order internally uses compareTo() method to perform sorting and it compare to objects and it return int value as a return value.*

```
"ratan".compareTo("anu")        ==>     +ve     ==>change the order
"ratan".compareTo("ratan")      ==>     0       ==>no change
"anu".compareTo("ratan")        ==>     -ve     ==>no change
```

Example:-

The sorting object(Emp) Not implementing Comparable interface hence it does not perform sorting.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(new Emp(111,"ratan"));
                Collections.sort(al);
        }
}
```

When we execute the above example JVM will generate Exception,

*"**java.lang.ClassCastException: Emp cannot be cast to java.lang.Comparable"***

Example :-

***If the Class contains Heterogeneous data sorting is not possible**.*

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add("ratan");
                al.add(10);
                Collections.sort(al);//java.lang.ClassCastException
                System.out.println(al);
        }
}
```

To overcome above two cases exception use Comparable or Comparator interfaces to perform sorting.

Comparable vs Comparator :-

- ✓ *If we want to perform default natural sorting order then your objects must be homogeneous & comparable.*
- ✓ *Comparable objects are nothing but the objects which are implements comparable interface.*
- ✓ *All wrapper classes & String objects are implementing Comparable interface hence it is possible to perform sorting.*

- ❖ *If we want to sort user defined class like Emp based on eid or ename with default natural sorting order then your class must implements Comparable interface.*
- ❖ *Comparable interface present in java.lang package it contains only one method compareTo(obj) then must override that method to write the sorting logics.*

  public abstract int compareTo(T);

- ❖ If your class is implementing Comparable interface then that objects are sorted automatically by using **Collections.sort().** And the objects are sorted by using compareTo() method of that class.

Normal version of comparable:-

Emp.java:-

```
class Emp implements Comparable
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
        public int compareTo(Object o)
        {       Emp e = (Emp)o;
                if (eid == e.eid )
                {       return 0;       }
                else if (eid > e.eid)
                {       return 1;       }
                else
                {       return -1;      }
        }
}
```

Test.java:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<Emp> al = new ArrayList<Emp>();
                al.add(new Emp(333,"ratan"));
                al.add(new Emp(222,"anu"));
                al.add(new Emp(111,"Sravya"));
                Collections.sort(al);
                Iterator itr = al.iterator();
                while (itr.hasNext())
                {       Emp e = (Emp)itr.next();
                        System.out.println(e.eid+"---"+e.ename);
                }
        }
}
```

Generic version of Comparable:-

```
class Emp implements Comparable<Emp>
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
        public int compareTo(Emp e)
        {       return  ename.compareTo(e.ename);
        }
```

```
}
```

Java.utilComparator :-

- ✓ *For the default sorting order use comparable but for customized sorting order we can use Comparator.*
- ✓ *The class whose objects are stored do not implements this interface some third party class can also implements this interface.*
- ✓ *Comparable present in **java.lang**package but Comparator present in **java.util** package.*
- ✓ *Comparator interface contains two methods,*

```
public interface java.util.Comparator<T> {
  public abstract int compare(T, T);
  public abstract boolean equals(java.lang.Object);
 }
```

Normal version of Comparator:-

Emp.java:-

```
class Emp
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
}
```

EidComp.java:-

```
import java.util.Comparator;
class EidComp implements Comparator
{       public int compare(Object o1,Object o2)
        {               Emp e1 = (Emp)o1;
                        Emp e2 = (Emp)o2;
                        if (e1.eid==e2.eid)
                        {       return 0;       }
                        else if (e1.eid>e2.eid)
                        {       return 1;       }
                        else
                        {       return -1;      }
        }
}
```

EnameComp.java:-

```
import java.util.Comparator;
class EnameComp  implements Comparator
{       public int compare(Object o1,Object o2)
        {       Emp e1 = (Emp)o1;
                Emp e2 = (Emp)o2;
```

```
                return (e1.ename).compareTo(e2.ename);
        }
}
```

Test.java:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<Emp> al = new ArrayList<Emp>();
                al.add(new Emp(333,"ratan"));
                al.add(new Emp(222,"anu"));
                al.add(new Emp(111,"Sravya"));
                al.add(new Emp(444,"xxx"));

                System.out.println("sorting by eid");
                Collections.sort(al,new EidComp());
                Iterator<Emp> itr = al.iterator();
                while (itr.hasNext())
                {       Emp e = itr.next();
                        System.out.println(e.eid+"---"+e.ename);
                }
                System.out.println("sorting by ename");
                Collections.sort(al,new EnameComp());
                Iterator<Emp> itr1 = al.iterator();
                while (itr1.hasNext())
                {       Emp e = itr1.next();
                        System.out.println(e.eid+"---"+e.ename);
                }
        }
}
D:\vikram>java Test
sorting by eid
111---Sravya
222---anu
333---ratan
444---xxx
sorting by ename
222---anu
111---Sravya
333---ratan
444---xxx
```

The above example code:-(with generic version)

EnameComp.java:-

```
import java.util.Comparator;
```

```
class EnameComp  implements Comparator<Emp>
{       public int compare(Emp e1,Emp e2)
        {               return (e1.ename).compareTo(e2.ename);
        }
}
```

EidComp.java:-

```
import java.util.Comparator;
class EidComp implements Comparator<Emp>
{       public int compare(Emp e1,Emp e2)
        {       ***************
                ***************
        }
}
```

Java.lang.Comparable vs java.util.Comparator:-

Property	***Comparable***	***Comparator***
1. Sorting logics	*1) Sorting logics must be in the class whose class objects are sorting.*	***2) Int compareTo(Object o1)*** *This method compares this object with o1 object and returns a integer.Its value has following meaning* ***positive –*** *this object is greater than o1* ***zero*** *– this object equals to o1* ***negative –****this object is less than o1*
2. Sorting method		***3) Collections.sort(List)*** *Here objects will be sorted on the basis of CompareTo method.*
3. Method calling to perform sorting		*4) Java.lang*
4. package		
5. which type of sorting		

5) *Default natual sorting order*

Comparator

I. *Sorting logics in separate class hence we are able to sort the data by using dif attributes.*

II. *int compare(Object o1,Object o2)*
This method compares o1 and o2 objects. and returns a integer.Its value has following meaning.
positive *– o1 is greater than o2*
zero –*o1 equals to o2*
negative*– o1 is less than o1*

III. ***Collections.sort(List, Comparator)***
Here objects will be sorted on the basis of Compare method in Comparator

IV. ***Java.util***

V. ***For customized sorting order.***

java.util.LinkedList:-

public class java.util.LinkedList ***extends*** *java.util.AbstractSequentialList*
implements *java.util.List<E>,*
java.util.Deque<E>,
java.lang.Cloneable,
java.io.Serializable

1) Introduced in 1.2 version.
2) Heterogeneous objects are allowed.
3) Null insertion is possible.
4) Insertion order is preserved.
5) LinkedList methods are non-synchronized.
6) Duplicate objects are allowed.
7) The under laying data structure is double linkedlist.
8) cursors :- Iterator,ListIterator

constructors:-

LinkedList(); *it builds a empty LinkedList.*

LinkedList(java/util/Collection<? extends E>);
it builds a LinkedList that initialized with the collection data.

Example:- LinkedList basic operations.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       LinkedList<String> l=new LinkedList<String>();
                l.add("B");
                l.add("C");
                l.add("D");
                l.add("E");
                l.addLast("Z");//it add object in last position
                l.addFirst("A");//it add object in first position
                l.add(1,"A1");//add the Object spcified index
                System.out.println("original content:-"+l);
                l.removeFirst();          //remove first  Object
```

```
                l.removeLast();          //remove last t Object
                System.out.println("after deletion first & last:-"+l);
                l.remove("E");           //remove specified Object
                l.remove(2);             //remove the object of specified index
                System.out.println("after deletion :-"+l);//A1 B D
                String val = l.get(0);  //get method used to get the element
                l.set(2,val+"cahged");//set method used to replacement
                System.out.println("after seting:-"+l);
        }
};
```

D:\>java Test
original content:-[A, A1, B, C, D, E, Z]
after deletion first & last:-[A1, B, C, D, E]
after deletion :-[A1, B, D]
after seting:-[A1, B, A1cahged]

Example:-Adding one collection data into another Collection.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("ratan");
                al.add("balu");

                LinkedList<String> linked = new LinkedList<String>(al);
                linked.add("anu");
                linked.add("simran");
                System.out.println(linked);
        }
}
```

E:\>java Test
[ratan, balu, anu, simran]

Example :- LinkedList cloneing process:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {
         LinkedList<String> arrl = new LinkedList<String>();
         arrl.add("First");
        arrl.add("Second");
        arrl.add("Third");
        arrl.add("Random");
        System.out.println("Actual LinkedList:"+arrl);
        LinkedLis copy = (LinkedLis) arrl.clone();
        System.out.println("Cloned LinkedList:"+copy);
   }
}
```

E:\>java Test
Actual LinkedList:[First, Second, Third, Random]
Cloned LinkedList:[First, Second, Third, Random]

Vector:- (legacy class introduced in 1.0 version)

public class java.util.Vector ***extends*** *java.util.AbstractList*
implements *java.util.List<E>,*
java.util.RandomAccess,
java.lang.Cloneable,
java.io.Serializable

1) Introduced in 1.0 version it is a legacy class.
2) Heterogeneous objects are allowed.
3) Duplicate objects are allowed.
4) Null insertion is possible.
5) Insertion order is preserved.
6) The under laying data structure is growable array.
7) Vector methods are synchronized.
8) Applicable cursors are Iterator,Enumeration,ListIterator.

Vector constructors:-

Vector();
Vector(int initialCapacity);
Vector(int intialCapacity, int increment);
Vector(java/util/Collection<? extends E>);

Constructor 1:-

The default initial capacity of the Vector is 10 once it reaches its maximum capacity it means when we trying to insert 11 element that capacity will become double[20].

```
Vector v = new Vector();
System.out.println(v.capacity());      //10
v.add("ratan");
System.out.println(v.capacity());      //10
System.out.println(v.size());          //1
```

Constructor 2:-

It is possible to create vector with specified capacity by using fallowing constructor. in this case once vector reaches its maximum capacity then size is double based on provided initial capacity.

```
Vector v = new Vector(int initial-capacity);
Vector<String> vv = new Vector<String>(3);
    System.out.println(vv.capacity());//3
    vv.add("aaa");
    vv.add("bbb");
    vv.add("ccc");
    vv.add("ddd");
```

```
System.out.println(vv.capacity());      //6
System.out.println(vv.size());          //4
```

Constructor 3:-

It is possible to create vector with initial capacity and providing increment capacity by using fallowing constructor.

Vector v = new Vector(int initial-capacity, int increment-capacity);

```
Vector<String> v = new Vector<String>(2,5);
System.out.println(v.capacity());       //2
v.add("ratan");
v.add("aruna");
v.add("Sravya");
System.out.println(v.capacity());       //7
System.out.println(v.size());           //3
```

Constructor 4:-

Vector(java/util/Collection<? extends E>);

It creates the Vector that contains another Collection data.

Example:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("no1");
                al.add("no2");

                Vector<String> v = new Vector<String>(al);
                v.add("ratan");
                v.add("aruna");
                System.out.println(v);

                ArrayList<String> a2 = new ArrayList<String>(v);
                a2.add("xxx");
                a2.add("yyy");
                System.out.println(a2);
        }
}
E:\>java Test
[no1, no2, ratan, aruna]
[no1, no2, ratan, aruna, xxx, yyy]
```

Example:-

`In below example Vector class removeElement() method removes the data always based on object but not index.`

```
Vector v=new Vector();                  v.removeElement("ratan");
v.addElement("ratan");                  System.out.println(v);//[] empty output
```

```
                                                    v.removeElement(0);
Vector v=new Vector();                              System.out.println(v); //[ratan]
v.addElement("ratan");
```

The List interface remove() method removes the data based on index and object.

```
Vector v=new Vector();
v.addElement("ratan");
v.remove(0);
System.out.println(v);//[] empty output

import java.util.*;
class  Test
{       public static void main(String[] args)
        {
        Vector<Integer> v=new Vector<Integer>();//generic version of vector
        for (int i=0;i<5 ;i++ )
        {       v.addElement(i);
        }
        v.addElement(6);
        v.removeElement(1);     //it removes element object  based
        Enumeration<Integer> e = v.elements();
        while (e.hasMoreElements())
        {       Integer i = e.nextElement();
                System.out.println(i);
        }
        v.clear();          //it removes all objects of vector
        System.out.println(v);
        }
}
```

E:\>java Test
01246[]

<u>Copying data from Vector to ArrayList:-</u>

To copy data from one class to another class use ***copy()*** method of Collections class.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("10");
                al.add("20");
                al.add("30");
                Vector<String>  v = new Vector<String>();
                v.add("ten");
                v.add("twenty");
                //copy data from vector to ArrayList
                Collections.copy(al,v);
                System.out.println(al);
        }
```

```
}
D:\vikram>java Test
[ten, twenty, 30]
```

Passing data {ArrayList to Vector} & Vector to ArrayList:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList<String> a1 = new ArrayList<String>();
                a1.add("ratan");
                a1.add("anu");
                a1.add("Sravya");
                a1.add("yadhu");
                //ArrayList - Vector
                Vector<String> v = new Vector<String>(a1);
                v.add("xxx");
                v.add("yyy");
                System.out.println(v);          //[ratan, anu, Sravya, yadhu, xxx, yyy]
                //Vector-ArrayList
                ArrayList<String> a2 = new ArrayList<String>(v);
                a2.add("suneel");
                System.out.println(a2);         //[ratan, anu, Sravya, yadhu, xxx, yyy, suneel]
        }
}
```

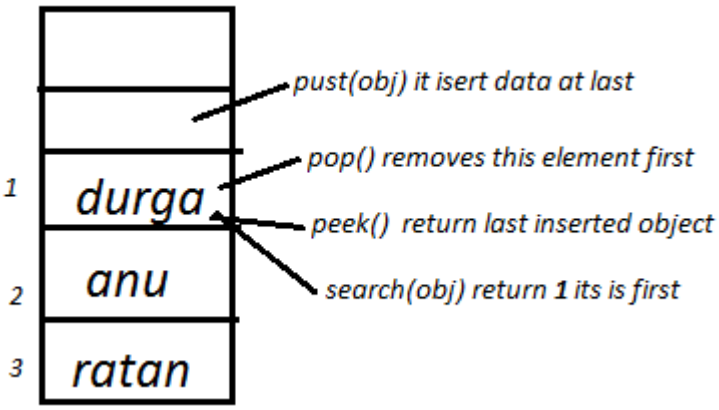
Stack:- (legacy class introduced in 1.0 version)

1) *It is a child class of vector.*
2) *Introduce in 1.0 versionit is a legacy class.*
3) *It is designed for LIFO(last in fist order).*

Example:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       Stack<String> s = new Stack<String>();
                s.push("ratan");        //insert the data top of the stack
                s.push("anu");          //insert the data top of the stack
                s.push("Sravya");
                System.out.println(s);
                System.out.println(s.search("Sravya"));   //1   last added object will become first
                System.out.println(s.size());
                System.out.println(s.peek());     //to return last element of the Stack
                s.pop();                          //remove the data top of the stack
                System.out.println(s);
```

```
                System.out.println(s.isEmpty());
                s.clear();
                System.out.println(s.isEmpty());
        }
}
```



Example :-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       String reverse="";
                Scanner s = new Scanner(System.in);
                System.out.println("enter input string to check palendrome or not");
                String str = s.nextLine();
                Stack stack = new Stack();
                for (int i=0;i<str.length();i++)
                {       stack.push(str.charAt(i));
                }
                while (!stack.isEmpty())
                {       reverse=reverse+stack.pop();
                }
                if (str.equals(reverse))
                {       System.out.println("the input String palindrome");
                }
                else
                {       System.out.println("the input String not- palindrome");
                }
        }
}
```