Java means DURGA SOFT..

CORE JAVA

Material

Java™

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Collectionsframework (java.util)

Pre-requisite topics for Collections framework:-

1) ***AutoBoxing.***
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.***

- Collection frame contains group of classes and interfaces by using these classes & interfaces we are representing group of objects as a single entity.
- Collection is sometimes called a **container**. And it is object that groups multiple elements into a **single unit.**
- Collections are used to store, retrieve ,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***

Note :- The root interface of Collection framework is **Collection** it contains 15 methods so all implementation classes are able to use that methods.

```
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();
```

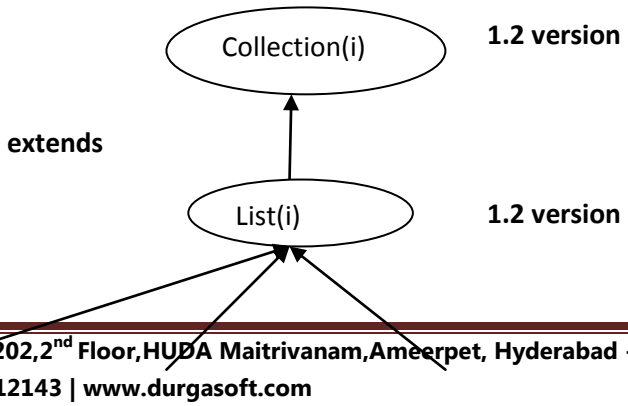
The interface contains abstract method and for that interfaces object creation is not possible hence think about implementation classes of that interfaces.

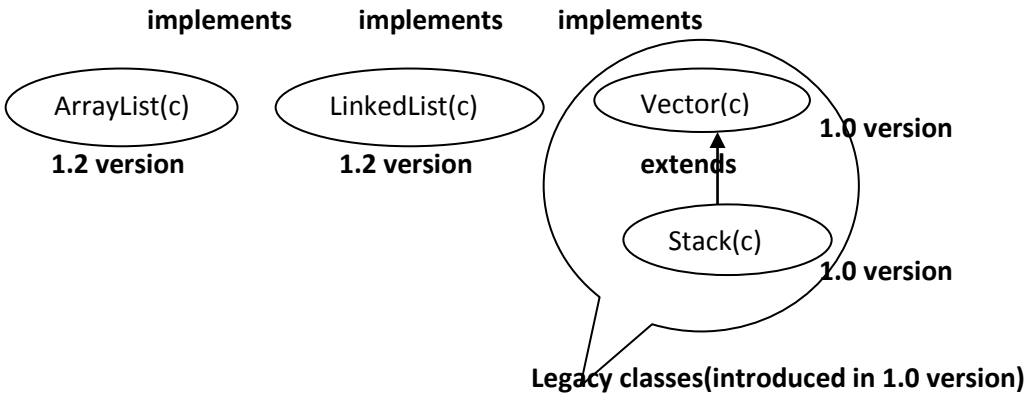
Collection vs Collections:-

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.

Characteristics of Collection frame work classes:-

1) ***The collect ion framework classes are introduced in different Versions.***
2) ***Heterogeneous data allowed or not allowed.***
   All classes allowed heterogeneous data except two classes
   i. ***TreeSet*** ii. ***TreeMap***
3) ***Null insertion is possible or not possible.***
4) ***Insertion order is preserved or not preserved.***
   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*
5) ***Collection classes' methods are synchronized or non-synchronized.***
6) ***Duplicate objects are allowed or not allowed.***
   add(e1)
   add(e1)
7) ***Collections classes underlying data structures.***
8) ***Collections classes supported cursors.***





I ----------→Interface

c----------→class

Legcy class:- The java classes which are introduced in 1.0 version are called legacy classes.

Ex :- Vector , Stack , HashTable…….etc

Implementation classes of List interface :-

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

List interface common properties:-

1) *All List interface implementation classes allows null insertion.*
2) *All classes allows duplicate objects.*
3) *All classes preserved insertion order.*

Java.util.ArrayList:-

ArrayList is implementing List interface it widely used class in projects because it is providing functionality and flexibility

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) Inside ArrayList we can insert **Null** objects.
4) ArrayList preserved Insertion order it means whatever the order we inserted the data in the same way output is printed.
   - ***a.*** Input -→e1 e2 e3 output -→e1 e2 e3 ***insertion order is preserved***
   - b. Input --→e1 e2 e3 output --→e1 e3 e2 ***insertion order is not- preserved***
5) ArrayList methods are non-synchronized methods.
6) Duplicate objects are allowed.
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:-

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

It is possible to create ArrayList with initial capacity

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

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

ArrayList al = new ArrayList(Collection c);

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
```

Example :-Collections vs Autoboxing

upto 1.4 version by using wrapper classes, create objects then add that objects in 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);

        }
}
```

But from 1.5 version onwards autoboxing concept is introduced so add the primitive value directly that is automatically converted into wrapper objects format.

```
import java.util.ArrayList;
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                al.add(10);     //primitive int value     --->Integer Object conversion   //AutoBoxing
                al.add('a');    //primitive char value    --->Integer Object conversion   //AutoBoxing
                al.add(10.5);   //primitive double value --->Integer Object conversion   //AutoBoxing
                System.out.println(al);
```

```
        }
}
```

Example:-Basic operations of ArrayList

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       ArrayList al =new ArrayList();
                al.add("A");
                al.add("B");
                al.add('a');
                al.add(190);
                al.add(null);
                System.out.println(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(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(al);
        }
}
```

- ✓ *In above example when we are adding primitive char value **al.add('c')** in ArrayList that value is automatically converted **Character** object format because ArrayList is able to store objects that is called AutoBoxing.*
- ✓ *When we print the ArrayList data for every object internally it is calling toString() method.*

Example :- *in Collection framework when we remove the data by using numeric value that is by default treated as a **index** value.*

```
ArrayList al = new ArrayList();
```

```
al.add(10);
al.add("ratan");
al.add('a');
System.out.println(al);
```

*In above example if u want remove **10**object by using object name then we are using below code.*

```
al.remove(10);
```

*But whenever we are writing above code then JVM treats that 10 is index value hence it is generating exception **java.lang.IndexOutOfBoundsException: Index: 10, Size: 3***

*To overcome above limitation if we want 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);
```

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

Emp.java:-

```
class Emp
{       //instance variables
        int eid;
        String ename;
        Emp(int eid,String ename)
        {//conversion of local - instance
        this.eid=eid;
        this.ename=ename;
        }
}
```

Student.java

```
class Student
{       //instance variables
        int sid;
        String sname;
        Student(int sid,String sname)
        {//conversion of local to instance
                this.sid=sid;
                this.sname = sname;
        }
}
```

```
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() --->it execute Integer class toString()
                al.add('a');            //toString() --->it execute Character class toString()
                al.add(e1);             //toString() --->it executes Object class toString()
                al.add(s1);             //toString() --->it executes Object class toString()
                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);
                        }
                }
        }
}
```


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);
        }
}
```

All collection classes are having 2-versions:-

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

Note :-

in java it is recommended to use generic version of collections class to store specified type of data.

Syntax:-

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

Examples:-

```
ArrayList<Integer> al = new ArrayList<Integer>();          //store only Integer objects
ArrayList<String> al = new ArrayList<String>();            //store only String objects
ArrayList<Student> al = new ArrayList<Student>();          //store only Student objects
```

ArrayList<product> al = new ArrayList<product>(); ***//store only produce objects***

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');
   al.add(10.5);
   System.out.println(al);

2) *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 holding different types of Objects.

```
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);
            al.add(true);
            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 holding only Integer data.

```
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 ArrayLiat

```
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);
                }
        }
}
```


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]
        }
}
```

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)

Example:-

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

HasSet h = new HashSet();                    //non- synchronized version of HashSet
Set h1  = Collections.synchronizedSet(h); // synchronized version of HashSet

HashMap h = new HashMap();                   //non- synchronized version of HashMap
Map m = Collections.synchronizedMap(h);      // synchronized version of HashMap
```

Conversion of Arrays to ArrayList & ArrayList to Arrays:

Conversion of ArrayList to String array by using toArray(T) methd of ArrayList class:-

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);
                //normal approach to print the data
                System.out.println(a[0]);
                System.out.println(a[1]);
                System.out.println(a[2]);
                System.out.println(a[3]);
                //for-each loop to print the data
                for (String s:a)
                {System.out.println(s);
                }
        }
}
```

Example :- 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);
                }
        }
}
```

Example :-

```
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());
                        }
                }
        }
}
```

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);
                }
        }
}
```

we are able to retrieve objects from collection classes in three ways:-

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

Cursors:-

Cursors are used to retrieve the Objects from collection classes.

There are three types of cursors present in the java.

1. Enumaration
2. Iterator
3. ListIteator

Applying Enumeration cursor on ArrayList:-

```
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       ArrayList<String> al = new ArrayList<String>();
                al.add("anu");
                al.add("Sravya");
                al.add("ratan");
                al.add("natraj");
                Enumeration<String> e = Collections.enumeration(al);
                while (e.hasMoreElements())
                {       String str = e.nextElement();
                        System.out.println(str);
                }
        }
}
```

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 ascening 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 decening order");
                for (String str2 :al)
                {       System.out.println(str2);
                }

        }
}
```


Comparable vs Comparator :-

Note :- it is possible to sort String and all wrapper objects because these objects are implementing Cloneable interface.

- If we want to sort user defined class Emp based on eid or ename then your class must implements Comparable interface.
- Comparable present in java.lang package it contains only one method **compareTo(obj)**

  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.
- By using comparable it is possible to sort the objects by using only one instance variable either eid or ename.

Emp.java:-

```
class Emp implements Comparable
{       int eid;
        String ename;
        Emp(int eid,String ename)
        {       this.eid=eid;
                this.ename=ename;
        }
        /* it is sorting the data by using ename instance variable
        public int compareTo(Object o)
        {       Emp e = (Emp)o;
                return  ename.compareTo(e.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;      }
        }
}
```

Another format:-

```
class Emp implements Comparable<Emp>
{       ***********
        public int compareTo(Emp e)
        {       ***********
        }
}
```

Test.java:-

```
import java.util.*;
```

```
class Test
{       public static void main(String[] args)
        {       ArrayList al = new ArrayList();
                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);
                }

        }
}
```


Java.utilComparator :-

- ✓ *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);
}
```

- ✓

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);
                //return   -(e1.ename).compareTo(e2.ename);  //print data descending order
        }
}
```

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 project level 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)
        {       ***************
                ***************
        }
}
```

Property

1. ***Sorting logics***
2. ***Sorting method***
3. ***Method calling to perform sorting***
4. ***package***

negative –*this object is less than o1*

3) ***Collections.sort(List)*** *Here objects will be sorted on the basis of CompareTo method*
4) *Java.lang*

Comparable

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*

Comparator

I. *Sorting logics in separate class hence we are able to sort the data by using different 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. ***Collections.sort(List, Comparator)*** *Here objects will be sorted on the basis of Compare method in Comparator*

Copying data from Vector to ArrayList:-

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]
        }
}
```

Enumaration:-0

1. Enumeration cursor introduced in 1.0 version hence it is called legacy cursor.

2. Enumeration is a legacy cursor it is used to retrieve the objects form only legacy classes (vector, Stack, HashTable…)hence it is not a universal cursor.

3. to retrieve Object from collection classes Enumeration Object uses two methods.
   public abstract boolean hasMoreElements();
   This method is used to check whether the collection class contains Objects or not, if collection class contains objects return true otherwise false.
   public abstract E nextElement();
   This method used to retrieve the objects from collection classes.

4. **elements()** method used to get Enumeration Object.
   Vector v =new Vector();
   v.addElement(10);
   v.addElement(20);
   v.addElement(30);
   Enumeration e = v.elements();

5. ***Normal version of Enumeration***

```
Vector v =new Vector();
v.addElement(10);
v.addElement(20);
v.addElement(30);
Enumeration e = v.elements();
while (e.hasMoreElements())
{//typecasting required
Integer i = (Integer)e.nextElement();
System.out.println(i);
}
```

Generic version of Enumeration

```
Vector v =new Vector();
v.addElement(10);
v.addElement(20);
v.addElement(30);
Enumeration<Integer> e = v.elements();
while (e.hasMoreElements())
{
Integer i = e.nextElement();
//type casting is not required
System.out.println(i);
}
```

6. By using this cursor it is possible to read the data only, it not possible to update the data an not possible to remove the data.

7. By using this cursor we are able to retrieve the data only in forward direction.

Iterator:-

1) Iterator cursor introduced in 1.2 versions.

2) Iterator is a universal cursor applicable for all collection classes.

3) **iterator()** method is used to get Iterator object.

ex:- ArrayList al =new ArrayList();
al.add(10);
al.add(20);
al.add(30);
Iterator itr = al.iterator();

4) The Iterator object uses three methods to retrieve the objects from collections classes.

public abstract boolean hasNext();

This is used to check whether the Objects are available in collection class or not , if available returns true otherwise false.

public abstract E next();

This method used to retrieve the objects.

public abstract void remove();

This method is used to remove the objects from collections classes.

5) ***Normal version of Iterator***

```
ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        al.add(30);
Iterator itr = al.iterator();
while (itr.hasNext())
{
Integer i = (Integer)itr.next();
//normal version typecasting is required
System.out.println(i);
}
```

Generic version of ArrayList

```
ArrayList<Integer> al = new
ArrayList<Integer>();
al.add(10);
al.add(20);
al.add(30);
Iterator<Integer> itr = al.iterator();
        while (itr.hasNext())
        {
        Integer i = itr.next();
//generic version type casting is not required
        System.out.println(i);
        }
```

6) By using Iterator cursor we are able to perform read and remove operations but it is not possible to perform update operation.

7) By using Iterator we are able to read the data only in forward direction.

Property

1. *Purpose*
2. *Legacy or not*
3. *Applicable for which type of classes*
4. *How to get the object*
5. *How many methods*

6. Operations	4) Get the Enumeration Object by using elements() method. ***Vector v =new Vector();*** ***v.add(10);*** ***v.add(20);*** ***Enumeration e = v.elements();***
7. Cursor moment	5) It contains two methods a. hasMoreElements() b. nextElement()
8. Universal cursor or not	6) only read operations.
9. Class or interface	7) Only forward direction.
10. Versions supports	8) Not a universal cursor because it is applicable for only legacy classes.
	9) Interface
	10) It supports both normal and generic version.

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

LIstIterator:-

1. ListIterator cursor Introduced in 1.2 version
2. This cursor is applicable only for List type of classes(ArrayList,LinkedList,Vector,Stack...etc) hence it is not a universal cursor.
3. listIterator() method used to get ListIterator object

ex:- LinkedList ll = new LinkedList();
ll.add(10);

```
ll.add(20);
ll.add(30);
ListIterator lstr = ll.listIterator();
```

4. ListIterator contains fallowing methods

```
public abstract boolean hasNext();---->to check the Objects
public abstract E next();          ----->to retrieve the objects top to bottom
public abstract boolean hasPrevious(); ---→check the objects in previous direction
public abstract E previous();      ---->to retrieve the Objects from previous direction
public abstract int nextIndex();---->to get index
public abstract int previousIndex();--->to get the index from previous direction.
public abstract void remove(); --->to remove the Objects
public abstract void set(E); ----->to replace the particular Object
public abstract void add(E);---->to add new Objects
```

5. By using this cursor we are able to read & remove & update the data.
6. By using this cursor we are able to read the data both in forward and backward direction.

Differences between Enumeration & Iterator & ListIterator:-

Characterstics	5. Methods		1..0 version
1. Version		10.Return which object	
	6. Operations		legacy
2.Legacy or not			
	7.Class or interface		By using elements() method
3. How to get object			
	8.Applicable to which type of classes		Normal version & generic version
4. How many versions			
	9.Cursor moment	Enumeration	2 methods

	1.2 version	Implementation class of Iterator interface.	Read & remove &update
Read operations	Not a legacy		interface
interface	By using iterator() method		only for List type of classes
legacy classes	Normal version & generic version		both forward and backward directions.
only forward direction	3 methods	***ListIterator***	Implementation class of ListIterator interface.
implementation class of Enumeration Interface	Read & remove	1.2 version	
	interface	Not a legacy	
	all collection classes	By using listIterator()method	
	only forward direction.	Normal version & generic version	
Iterator		9 methods	

Synchronized Collection Methods of Collections class:-

Collections.synchronizedSortedSet(SortedSet<T> s)
Collections.synchronizedSortedMap(SortedMap<K,V> m)
Collections.synchronizedSet(Set<T> s)
Collections.synchronizedCollection(Collection<T> c)
Collections.synchronizedList(List<T> list)
Collections.synchronizedMap(Map<K,V> m)

Application shows implementation class object of cursor interfaces(Enumeration ,Iterator,ListIterator)

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       Vector v = new Vector();
                v.addElement(10);
                v.addElement(20);
                v.addElement(30);
                //it  returns implementation class object of Enumeration interface
                Enumeration e = v.elements();
                System.out.println(e.getClass().getName());

                //it  returns implementation class object of Iterator interface
                Iterator itr = v.iterator();
                System.out.println(itr.getClass().getName());

                //it  returns implementation class object of ListIterator  interface
                ListIterator lstr = v.listIterator();
                System.out.println(lstr.getClass().getName());
        }
};
```

D:\>java Test
java.util.Vector$1
java.util.Vector$Itr
java.util.Vector$ListItr

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 cursors.
3) By using get() method.

Example application:-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       //ArrayList able to store only String Objects
                ArrayList<String> al =new ArrayList<String>();
                al.add("A");
                al.add("B");
                al.add("C");
                al.add("D");
                al.add(null);

                //1st appraoch to print Collection class elements  (by using for-each loop)
                for (String  a : al)
                {       System.out.println(a);          }

                //2nd approach to print Collection class elements   (by using cursors)
                Iterator itr1 = al.iterator();//normal version of Iterator
                while (itr1.hasNext())
                {       String str =(String)itr1.next();//type casting required because normal version
                        System.out.println(str);
                }
                Iterator<String> itr2 = al.iterator();//generic  version of Iterator
                while (itr2.hasNext())
                {       String str =itr2.next();//type castingnot  required because generic  version
                        System.out.println(str);
                }
        //3rd approach to print objects by using get() method
        int size = al.size();
        for (int i=0;i<size;i++)
        {       System.out.println(al.get(i));
        }
        }
}
```

Example :-[ListIterator VS ArrayList]

add(E);	***---->to add the Object***
remove(java.lang.Object);	***----->to remove the object***
size();	***----->to find size***
isEmpty();	***----->returns true if empty otherwise false***
clear();	***------>to remove all objects***
addAll();	***------>to add one collection object into another collection***
removeAll();	***----->to remove all the elements of particular collection***
retainAll();	***---->to remove all elements except particular collections***

Example:-

```
import java.util.*;
class Emp
{       //instance variables
        int eid;
        String ename;
        Emp(int eid ,String ename)  //local variables
        {       //conversion of local variables to instance variables
                this.eid = eid;
                this.ename = ename;
        }
        public static void main(String[] args)
        {       Emp main1 = new Emp(111,"ratan");
                Emp main2 = new Emp(222,"Sravya");
                Emp main3 = new Emp(333,"aruna");
                Emp sub1 = new Emp(444,"anu");
                Emp sub2 = new Emp(555,"banu");

                ArrayList<Emp> al1 = new ArrayList<Emp>();      //generic  version of ArrayList
                al1.add(main1);
                al1.add(main2);
                al1.add(main3);

                ArrayList<Emp> al2 = new ArrayList<Emp>();      //generic version of ArrayList
                al2.add(sub1);
                al2.add(sub2);
                al1.addAll(al2);            //add all objects of al2 into al1
                al1.remove(main2);          //it removes main1 object from al1
                al1.removeAll(al2);         //it removes all objects of al2
                //it checks wether main2 available or not
                System.out.println(al1.contains(main2));
                System.out.println(al1.size());             //print size

//printing elements by using for-each loop
for (Emp o1 : al1)
{
        System.out.println(o1.eid+" "+o1.ename);
}

        System.out.println("printing objects in forward direction");
        ListIterator<Emp>   lstr  = al1.listIterator();       //generic version of ListIterator cursor
                while (lstr.hasNext())
                {       Emp e = lstr.next();        //type casting not required becaue it is generic version
                        System.out.println(e.eid+" "+e.ename);

                }
```

```
        System.out.println("printing objects in  backword direction");
                while (lstr.hasPrevious())
                {       Emp e1 = lstr.previous();
                        System.out.println(e1.eid+" "+e1.ename);
                }
        }
}
```


ArrayList vs Iterator vs ListIterator:-

```
import java.util.*;
class Student
{       //instance varaibles
        int sno;        String sname;    int smarks;
        Student(int sno,String sname,int smarks)        //local variables
        { //conversion of local variables to instance variables
                this.sno = sno;
                this.sname = sname;
                this.smarks = smarks;
        }
        public static void main(String[] args)
        {       Student s1 = new Student(111,"ratan",100);
                Student s2 = new Student(222,"anu",99);
                Student s3 = new Student(333,"aruna",98);
                Student s4 = new Student(444,"pavan",97);
                ArrayList<Student> ar1 = new ArrayList<Student>();//generic version  of ArrayList
                ar1.add(s1);
                ar1.add(s2);    //ar1 contains 2 objects
                ArrayList<Student> ar2 = new ArrayList<Student>();//generic version  of ArrayList
                ar2.add(s3);
                ar2.add(s4);    //ar2 contains 2 object
                ar1.addAll(ar2);        //it's adding all objects ar2 into ar1
                ar1.retainAll(ar2);     //it removes all objects of  ar1 except ar2
                Iterator<Student> itr = ar1.iterator();     //generic version of Iterator

                System.out.println("using Iterator  retrieving  objects ony forward direction");
```

```
                while (itr.hasNext())
                {Student st =itr.next(); //type casting is not required because it is genric
                        System.out.println(st.sno+" "+st.sname+" "+st.smarks);
                }
                ListIterator<Student> ltr = ar1.listIterator();       //generic version of ListIterator

                System.out.println("print objects forward direction by using ListIterator");
                while (ltr.hasNext())
                {       Student stt = ltr.next();  //type casting is not required because it is generic
                        System.out.println(stt.sno+" "+stt.sname+" "+stt.smarks);
                }
                System.out.println("print objects backword direction by using ListIterator");
                while (ltr.hasPrevious())
                {       Student sttt =ltr.previous();
                        System.out.println(sttt.sno+" "+sttt.sname+" "+sttt.smarks);
                }
        }
}
```


LinkedList:-

Class LinkedList extends AbstractSequentialList implements List,Deque,Queue

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

if we are using AL to store the data at that situation whenever we are adding one new object middle of ArrayList then number of shift operations are requires it will degrade the p

Ex:-

```
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]

Vector:- (legacy class introduced in 1.0 version)

1) Introduced in 1.0 v legacy classes.
2) Duplicate objects are allowed.
3) Null insertion is possible.
4) Heterogeneous objects are allowed.
5) The under laying data structure is growable array.
6) Insertion order is preserved.
7) Every method present in the Vector is synchronized and hence vector object is Thread safe.
8) Cursors: - Enumeration.

Vector :-

Case:-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
```

Case 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
```

Case 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
```

Adds the specified component to the end of this vector

public void addElement(Object obj) --→Vector class method

Appends the specified element to the end of this Vector

public boolean add(Object o) --→List interface method

```
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>();
                v.add("ratan");
                v.add("aruna");
                v.add("Sravya");
                v.addElement("ccc");
                System.out.println(v);
                System.out.println(v.size());
                v.add(2,"xxx");
                v.remove("Sravya");    //removed based on object
                v.remove(1);           //removed based on index
                System.out.println(v);
                v.addAll(al);          //adding ArrayList data into Vector
                System.out.println(v);
                v.removeAll(al);       //it removes all objects of al
                System.out.println(v);
                System.out.println(v.firstElement());       //to retrieve first element
                System.out.println(v.lastElement());        //to retrieve last  element
        }
}
D:\vikram>java Test
[ratan, aruna, Sravya, ccc]
4
[ratan, xxx, ccc]
[ratan, xxx, ccc, no1, no2]
[ratan, xxx, ccc]
ratan
ccc
```

Example :-

```
//product.java
class Product
{       //instance variables
        int pid;
        String pname;
        double pcost;
        Product(int pid,String pname,double pcost)      //local variables
        {       //conversion [passing local variable values to instance variable]
                this.pid = pid;
```

```
                this.pname = pname;
                this.pcost = pcost;
        }
};
//ArrayListDemo.java
import java.util.*;
class ArrayListDemo
{       public static void main(String[] args)
        {       Product p1 = new Product(111,"pen",1300);
                Product p2 = new Product(222,"laptop",13000);
                Product p3 = new Product(333,"bag",1000);
                Product p4 = new Product(444,"java",5000);
                Product p5 = new Product(555,".net",4000);
                Vector<Product> v = new Vector<Product>();
                v.addElement(p1);
                v.addElement(p2);
                v.addElement(p3);
                System.out.println("***Enumeration cursor only read operations***");
                Enumeration<Product> e = v.elements();
                while (e.hasMoreElements())
                {       Product p = e.nextElement();
                        System.out.println(p.pid+"----"+p.pname+"----"+p.pcost);
                }
                System.out.println("***Iterator cursor both read & remove operations***");
                Iterator<Product> itr = v.iterator();
                while (itr.hasNext())
                {       Product pp = itr.next();
                        if ((pp.pname).equals("pen"))
                        itr.remove();           //pen object removed
                }

                System.out.println("***ListIterator cursor read & remove & update operations***");
                ListIterator<Product> lstr = v.listIterator();
                lstr.add(p4);   //p4 object is added by ListIterator
                while (lstr.hasNext())
                {       Product p = lstr.next();
                        if (p.pid==333)
                                lstr.remove();  //bag object removed
                        if ((p.pname).equals("laptop"))
                                lstr.set(p5);  //laptop is replaced by .net
                }

                System.out.println("***printing remaining objects ***");
                Iterator<Product> it = v.iterator();
                while (it.hasNext())
                {       Product p = it.next();
                        System.out.println(p.pid+"---"+p.pname+"---"+p.pcost);
```

```
                }
        }
};
```

Example:-

```
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);
        }
}
```

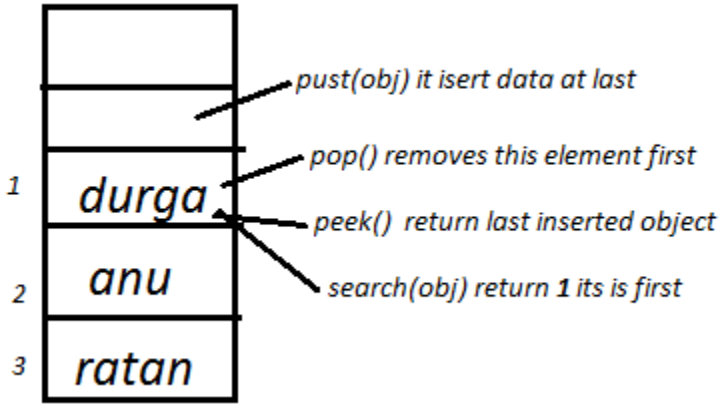
Stack:- (legacy class introduced in 1.0 version)

1) It is a child class of vector
2) Introduce in 1.0 v 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");
                }
```

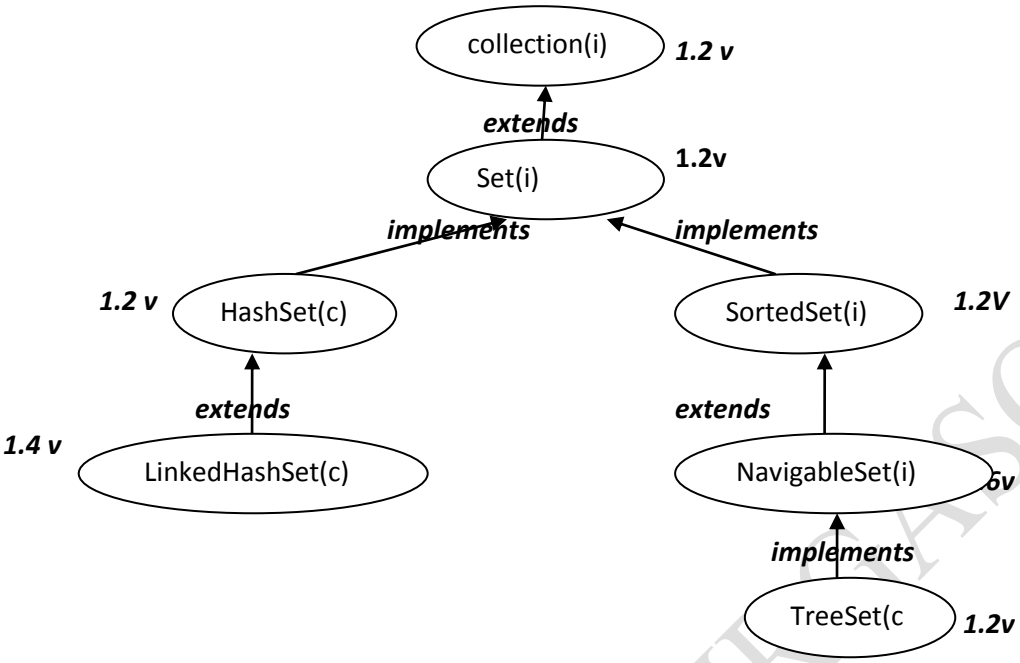
```
        }
}
```


5) by using ListIterator we are able to read & remove & update the data.

1. It is applicable for only list type of objects.
2. By using this it is possible to read the data upate the data and delete data also.
3. By using listIterator() method we are getting LIstIterator object

EmpBean.java:-

```
public class EmpBean implements Comparable<EmpBean>
{       private int eid;
        private String ename;
        public void setEid(int eid)
        {       this.eid=eid;
        }
        public void setEname(String ename)
        {       this.ename=ename;
        }
        public int getEid()
        {return eid;
        }
        public String getEname()
        {return ename;
        }
        public int compareTo(EmpBean o)
        {       if (eid==o.eid)
                {return 0;
                }
                if (eid>o.eid)
                {return 1;
                }
                else{return -1;}
        }
};
```



HashSet:-

1) *Introduced in 1.2 v.*
2) *Duplicate objects are not allowed if we are trying to insert duplicate values then we won't get any compilation errors an won't get any Execution errors simply add method return old value.*
3) *Null insertion is possible but if we are inserting more than one null it return only one null value.*
4) *Heterogeneous objects are allowed.*
5) *The under laying data structure is HashTable.*
6) *It is not maintain any order the elements are return in any random order .[Insertion order is not preserved].*
7) *Methods are non-synchronized.*
8) *cursor : Iterator*

Example:-

```
import java.util.*;
class  Test
{       public static void main(String[] args)
        {       //HashSet  object creation
```

```
                HashSet<String> h = new HashSet<String>();
                h.add("A");
                h.add("B");
                h.add("C");
                h.add("D");
                h.add("D");
                //creation of Iterator Object
                Iterator<String> itr = h.iterator();
                while (itr.hasNext())
                {       String str  = itr.next();
                        System.out.println(str);
                }
        }
}

import java.util.*;
class Test
{       public static void main(String[] args)
        {       HashSet<String> h = new HashSet<String>();
                h.add("ratan");
                h.add("anu");
                h.add("Sravya");
                HashSet<String> hsub = new HashSet<String>();
                hsub.add("no1");
                hsub.add("no2");
                hsub.addAll(h);
                System.out.println(hsub.contains("anu"));
                hsub.remove("anu");
                System.out.println(hsub.containsAll(h));
                System.out.println(hsub);
                hsub.removeAll(h);
                System.out.println(hsub);
                hsub.retainAll(h);
                System.out.println(hsub);
        }
}
```

LinkedHashSet:-

1. Introduced in 1.4 version and It is a child class of HashSet.
2. Duplicate objects are not allowed if we are trying to insert duplicate values then we won't get any compilation errors an won't get any Execution errors simply add method return false.
3. Null insertion is possible.
4. Heterogeneous objects are allowed
5. The under laying data structure is LinkedList & hashTable.
6. Insertion order is preserved.
7. Methods are non-synchronized.
8. Cursors :- Iterator.

Example:-

```
import java.util.*;
class  Test
{       public static void main(String[] args)
        {       Set<String> h = new LinkedHashSet<String>();
                h.add("A");
                h.add("B");
                h.add("C");
                h.add("D");
                h.add("D");
        //retrieving objects by using Iterator cursor
                Iterator<String> itr = h.iterator();
                while (itr.hasNext())
                {String str  = itr.next();
                 System.out.println(str);
                }
                //retrieving objects by using Enumeration cursor
                Enumeration<String> e = Collections.enumeration(h);
                while (e.hasMoreElements())
                {       System.out.println(e.nextElement());
                }
        }
}
import java.util.*;
class Test
{       public static void main(String[] args)
        {       HashSet<String> h = new HashSet<String>();
                h.add("ratan");
                h.add("anu");
                h.add("Sravya");
                //passing data from HashSet to LinkedHashSet
                LinkedHashSet<String> lh = new LinkedHashSet<String>(h);
                //lh.addAll(h);
                lh.add("xxx");
```

```
                lh.add("yyy");
                System.out.println(lh);
                //passing data from LinkedHashSet to HashSet
                HashSet<String> hh = new HashSet<String>(lh);
                //hh.addAll(lh);
                hh.add("zzz");
                System.out.println(hh);
        }
}
```

TreeSet:-

1. TreeSet is same as HashSet but TreeSet sorts the elements is ascending order but HashSet does not maintain any order.
2. The underlying data Structure is BalencedTree.
3. Insertion order is not preserved it is based some sorting order.
4. Heterogeneous data is not allowed.
5. Duplicate objects are not allowed
6. Null insertion is possible only once.

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       TreeSet<String> t = new TreeSet<String>();
                t.add("ratan");
                t.add("anu");
                t.add("Sravya");
                System.out.println(t);
                TreeSet<Integer> t1 = new TreeSet<Integer>();
                t1.add(10);
                t1.add(12);
                t1.add(8);
                System.out.println(t1);
        }
}
import java.util.*;
class Test
{       public static void main(String[] args)
        {       TreeSet<String> t = new TreeSet<String>();
                t.add("ratan");
                t.add("anu");
                t.add("Sravya");
                System.out.println(t);
                System.out.println(t.size());
                t.remove("ratan");
                System.out.println(t.contains("ratan"));

                TreeSet<String> t1 = new TreeSet<String>(t);
                //t1.addAll(t);
```

```
                t1.add("xxx");
                t1.removeAll(t);
                System.out.println(t1);
        }
}


import java.util.*;
class Fruit
{       public static void main(String[] args)
        {       TreeSet<String> t = new TreeSet<String>(new MyComp());
                t.add("orange");
                t.add("bananna");
                t.add("apple");
                System.out.println(t);
        }
}
class MyComp implements Comparator<String>
{       public int compare(String s1,String s2)
        {
                return s1.compareTo(s2);//[apple, bananna, orange]
                //return -s1.compareTo(s2);//[orange, bananna, apple]
        }
};

Example :-Elimination duplicate objects
import java.util.*;
class Test
{       public static void main(String[] args)
        {       String[] str={"ratan","anu","Sravya","anu"};
                //converion of String[] to List
                List<String> l = Arrays.asList(str);
                //conversion of List to Set [it eliminates duplicates]
                TreeSet<String> t = new TreeSet<String>(l);
                System.out.println(t);
        }
}
```

Example :-

```
import java.util.Iterator;
import java.util.TreeSet;
public class Test {
  public static void main(String[] args) {
    // creating a TreeSet Object
    TreeSet <Integer>treeadd = new TreeSet<Integer>();
// adding object  in the tree set
    treeadd.add(10);
    treeadd.add(30);
    treeadd.add(70);
    treeadd.add(20);
// create  iterator Object
    Iterator iterator;
    iterator = treeadd.iterator();

   // displaying the Tree set data
    System.out.println("Tree set data in ascending order: ");
    while (iterator.hasNext()){
     System.out.println(iterator.next() + " ");
    }
  }
}
```

Example :-

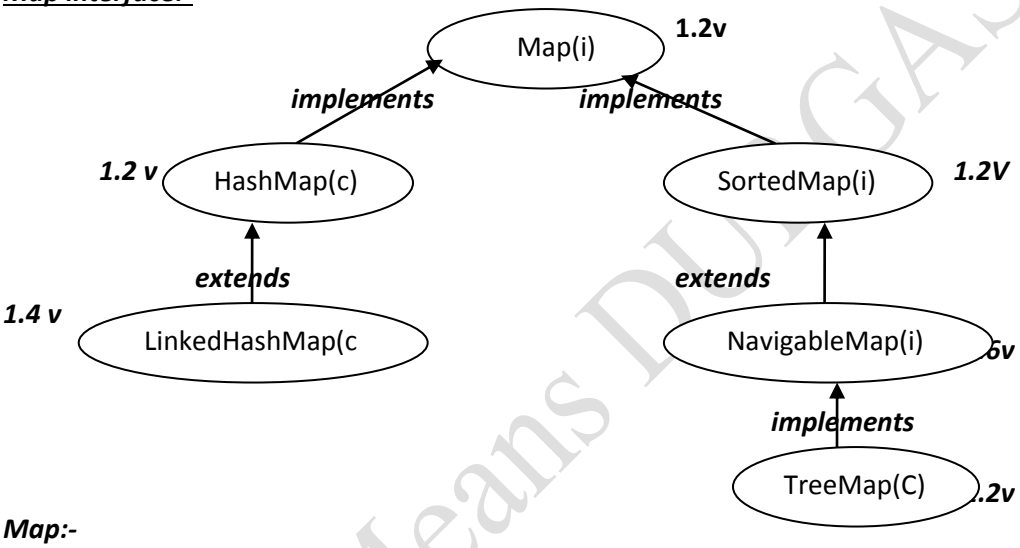
```
public E first();                          it print first element
 public E last();                          it print last element
 public E lower(E);                        it print lower object of specified object
 public E higher(E);                       it print higher object of specified object
  public java/util/SortedSet<E> subSet(E, E);   it print subset
 public java/util/SortedSet<E> headSet(E);     it print specified object above objects
 public java/util/SortedSet<E> tailSet(E);     it print specified objects below values
public E pollFirst();                      it print and remove first
public E pollLast();                       it print and remove last.
```

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       //creating TreeSet object
                TreeSet t=new TreeSet();
                //adding object in TreeSet
                t.add(50);          t.add(20);          t.add(40);
                t.add(10);          t.add(30);
                System.out.println(t);
                SortedSet s1=t.headSet(50);
```

```
                System.out.println(s1);          //[10,20,30,40]
                SortedSet s2=t.tailSet(30);
                System.out.println(s2);          //[30,40,50]
                SortedSet s3=t.subSet(20,50);
                System.out.println(s3);          //[20,30,40]
                System.out.println("last element="+t.last());
                System.out.println("first element="+t.first());
                System.out.println("lower element="+t.lower(50));
                System.out.println("higher element="+t.higher(20));
                System.out.println("print & remove first element="+t.pollFirst());
                System.out.println("print & remove last  element="+t.pollLast());
                System.out.println("final elements="+t);
                System.out.println("TreeSet size="+t.size());
                System.out.println("TreeSet size="+t.remove(10));
                System.out.println("TreeSet size="+t.remove(30));
        }
}
```

```
D:\morn11>java Test
[10, 20, 30, 40, 50]
[10, 20, 30, 40]
[30, 40, 50]
[20, 30, 40]
last element=50
first element=10
lower element=40
igher element=30
print & remove first element=10
print & remove last  element=50
final elements=[20, 30, 40]
TreeSet size=3
TreeSet size=false
TreeSet size=true
```

Map interface:-



Map:-

1. Map is a child interface of collection.
2. Up to know we are working with single object and single value where as in the map collections we are working with two objects and two elements.
3. The main purpose of the collection is to compare the key value pairs and to perform necessary operation.
4. The key and value pairs we can call it as map Entry.
5. Both keys and values are objects only.
6. In entire collection keys can't be duplicated but values can be duplicate.

HashMap:-

1) *interdicted in 1.2 version*
2) *Heterogeneous data allowed.*
3) *Underlying data Structure is HashTable.*
4) *Duplicate keys are not allowed but values can be duplicated.*
5) *Insertion order is not preserved.*
6) *Null is allowed for key(only once)and allows for values any number of times.*

7) *Every method is non-synchronized so multiple Threads are operate at a time hence permanence is high.*
8) *cursor :- Iterator.*

```
//public java/util/Set<K> keySet(); method syntax
// public java/util/Collection<V> values(); method syntax
// public java/util/Set<java/util/Map$Entry<K, V>> entrySet(); method syntax
```

Example :-

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       //creation of HashMap Object
                HashMap h = newHashMap();
                h.put("ratan",111);             //h.put(key,value);
                h.put("anu",111);
                h.put("banu",111);
                Set s1=h.keySet();              //used to get all keys
                System.out.println("all keys:--->"+s1);
                Collection c = h.values();      //used to get all values
                System.out.println("all values--->"+c);
                Set ss = h.enrySet();           //it returns all entryes nathing but [key,value]
                System.out.println("all entries--->"+ss);
                //get the Iterator Object
                Iterator itr = ss.iterator();
                while (itr.hasNext())
                {
                //next() method retrun first entry to represent that entery do typeCasting
                Map.Entry  m= (Map.Entry)itr.next();
                System.out.println(m.getKey()+"----"+m.getValue()); //printing key and value
                }
        }
};
```

LinkedHashMap:-

1) *interdicted in 1.4 version*
2) *Heterogeneous data allowed.*
3) *Underlying data Structure is HashTable & linkedlist.*
4) *Duplicate keys are not allowed but values can be duplicated.*
5) *Insertion order is preserved.*
6) *Null is allowed for key(only once)and allows for values any number of times.*
7) *Every method is non-synchronized so multiple Threads are operate at a time hence permanence is high.*
8) *cursor :- Iterator*

Emp.java:

```
class Emp
{       int eid;
        String ename;
```

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

//Student.java

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

Test.java:-

```
import java.util.*;
class  Test
{       public static void main(String[] args)
        {
        //creates LinkedList object with generic version
LinkedHashMap<Emp,Student> h = new LinkedHashMap<Emp,Student>();
                h.put(new Emp(111,"ratan"), new Student(1,"budha"));
                h.put(new Emp(222,"anu"), new Student(2,"ashok"));
                //get Set interface  before getting Iterator object
                Set s = h.entrySet();
                //creates iterator Object
                Iterator itr = s.iterator();
                while (itr.hasNext())
                {//holding Entry  by using Entry  interface
                Map.Entry m = (Map.Entry)itr.next();
                Emp e = (Emp)m.getKey();        //getting Emp object
                System.out.println(e.ename+"--"+e.eid);
                Student ss = (Student)m.getValue();//getting Student object
                System.out.println(ss.sname+"--"+ss.sid);
                }
        }
}
```

```
import java.util.*;
class  Test
{       public static void main(String[] args)
        {       HashMap h = new LinkedHashMap();
                h.put(111,"ratan");
                h.put(222,"anu");
                h.put(333,"sravya");
                System.out.println(h);

                Set s1 = h.keySet();
                System.out.println(s1);

                Collection  c = h.values();
                System.out.println(c);
```

```
                Set s2 = h.entrySet();
                Iterator itr = s2.iterator();
                while (itr.hasNext())
                {       Map.Entry m = (Map.Entry)itr.next();
                        System.out.println(m.getKey()+"---"+m.getValue());
                }
        }
}
```

HashTable:-

1. Introduced in the 1.0 version it's a legacy class.
2. Every method is synchronized hence only one thread is allowed to access it is a Thread safe but performance is decreased.
3. Null insertion is not possible if we are trying to insert null values we are getting NullPointerException.

h.put(null,"ratan");
h.put("4",null);

Ex:-

```
import java.util.Hashtable;
import java.util.Collection;
import java.util.Set;
class Test
{       public static void main(String[] args)
        {       Hashtable<String,String> h = new Hashtable<String,String>();
                //adding data in HashTable
                h.put("1","one");
                h.put("2","two");
                h.put("3","three");
                System.out.println(h);
                System.out.println(h.get("1"));//one
                System.out.println(h.isEmpty());
                h.remove("3");
                System.out.println(h.containsKey("1"));
```

```
                System.out.println(h.containsKey("3"));
                System.out.println(h.containsValue("one"));
                System.out.println(h.size());
                //to get all values objects
                Collection<String> c = h.values();
                for (String i : c)
                {       System.out.println(i);
                }
                //to get all key objects
                Set<String> s = h.keySet();
                for (String ss : s)
                {       System.out.println(ss);
                }
        }
}
```

Example :-

We are able to add one class data into another class in two ways

1) *Passing one class reference variable to another class*

   Hashtable h = new Hashtable();

   HashMap<String,String> h1 = new HashMap<String,String>(h);

2) *By using **putAll()** method*

   h1.putAll(h);

```
import java.util.Hashtable;
import java.util.*;
class Test
{       public static void main(String[] args)
        {       Hashtable<String,String> h = new Hashtable<String,String>();
                h.put("1","one");
                h.put("2","two");
                h.put("3","three");
                //passing Hashtable data into HashMap
                HashMap<String,String> h1 = new HashMap<String,String>(h);
                //h1.putAll(h);
                h1.put("hm","ratan");
                System.out.println(h1);
                //passing HashMap data into LinkedHashMap
                LinkedHashMap<String,String> lhm =  new LinkedHashMap<String,String>(h1);
                //lhm.putAll(h1);
                lhm.put("lhm","anu");
                System.out.println(lhm);
        }
}
```

TreeMap:-

Example-1:-

```
import java.util.TreeMap;
```

```
class Test
{	public static void main(String[] args)
	{	TreeMap<String,String> tmain = new TreeMap<String,String>();
		tmain.put("ratan","no1");
		tmain.put("anu","no2");

		TreeMap<String,String> tsub = new TreeMap<String,String>();
		tsub.putAll(tmain);
		tsub.put("x","no3");
		tsub.put("4","no4");
		System.out.println(tsub);
	}
}
```

Example -2:-

```
import java.util.TreeMap;
import java.util.Set;
import java.util.Collection;
import java.util.Map.Entry;
class Test
{	public static void main(String[] args)
	{	TreeMap<String,String> tmain = new TreeMap<String,String>();
		tmain.put("ratan","no1");
		tmain.put("anu","no2");

		TreeMap<String,String> tsub = new TreeMap<String,String>();
		tsub.putAll(tmain);
		tsub.put("x","no3");
		tsub.put("y","no4");
		System.out.println(tsub);

		if (tmain.containsKey("ratan"))
		{System.out.println("ratan is great");
		}
		if (tsub.containsValue("no1"))
		{System.out.println("no1 ratan only");
		}
		//printing all the keys
		Set<String> s = tsub.keySet();
		for (String ss : s)
		{	System.out.println(ss);
		}
		//printing all the values
		Collection<String> s1 = tsub.values();
		for (String ss1 : s1)
		{	System.out.println(ss1);
		}
		Set<Entry<String,String>> s2 = tsub.entrySet();
```

```
                for (Entry<String,String> ss2 : s2)
                {       System.out.println(ss2);
                }
                tsub.clear();
                System.out.println(tsub);
        }
}
```

Java.util.Properties:-

Abc.properties :-

username = system
password = manager
driver = oracle.jdbc.driver.OracleDriver
trainer = Ratan

Test.java:-

```
import java.util.*;
import java.io.*;
class  Test
{       public static void main(String[] args) throws FileNotFoundException,IOException
        {
//locate properties file
FileInputStream fis=new FileInputStream("abc.properties");
//load the properties file by using load() method of Properties class
Properties p = new Properties();
p.load(fis);
//get the data from properties class by using getProperty()
String username = p.getProperty("username");
String driver = p.getProperty("driver");
String password = p.getProperty("password");
String trainer = p.getProperty("trainer");
```

```
//use the properties file data
System.out.println("DataBase username="+username);
System.out.println("DataBase password ="+password);
System.out.println("driver ="+driver);
System.out.println("trainer="+trainer);
        }
}
```

Collections:-

```
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");
                //to perform sorting use sort method of collections class
                Collections.sort(al);
                Iterator itr =al.iterator();
                while (itr.hasNext())
                {System.out.println(itr.next());
                }
        }
}
```

Comparable interface :-

- *Comparable interface used to perform sorting of user defined class objects.*
- *Comparable present in java.lang package and it contains only method*
  public abstract int compareTo(Object obj-name);
- *By using comparable We areable sort the object by suing single data member like* ***sid,sname****.*
- *String & all Wrapper classes are implement Comparable interface hence if we are storing these Objects these are comparable.*

If first object sid value is greater than existing object then it returns positive//no change in data
If the object sid values is less than existing object then it returns negative.//change location
If any negative or both are equals then it returns zero. //no change in data

Student.java

```
class Student implements Comparable
{       int sid;
        String sname;
        Student(int sid,String sname)//local var
        { this.sname=sname; this.sid=sid;
        }
        public int compareTo(Object obj)
```

```
        {       Student s = (Student)obj;
                if (sid>s.sid)
                {return 1;
                }
                if (sid<s.sid)
                {return -1;
                }
                If(sid==0){
                return 0;}
        }
}
```

Test.java:-

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                ArrayList<Student> al = new ArrayList<Student>();
                al.add(new Student(11,"ratan"));
                al.add(new Student(2,"Sravya"));
                al.add(new Student(333,"anu"));
                Collections.sort(al);
                Iterator<Student> itr =al.iterator();
                while (itr.hasNext())
                {       Student s = itr.next();
                        System.out.println(s.sid+"----"+s.sname);
                }
        }
}
import java.util.*;
class Comp  implements Comparator
{
        public int compare(Object o1,Object o2)
        {
                EmpBean e1 = (EmpBean)o1;
                EmpBean e2 = (EmpBean)o2;
                if (e1.eid==e2.eid)
                {return 0;
                }
                if (e1.eid>e1.eid)
                {return 1;
                }
                else{return -1;}
```

```
	}
}
import java.util.*;
class Test
{	public static void main(String[] args)
	{
		TreeSet<EmpBean> s = new TreeSet<EmpBean>(new Comp());
		EmpBean e1 = new EmpBean();
		e1.setEid(111);
		e1.setEname("ratan");
		EmpBean e2 = new EmpBean();
		e2.setEid(22);
		e2.setEname("anu");

		s.add(e1);
		s.add(e2);

		for (EmpBean e:s)
		{System.out.println(e.eid+"---"+e.ename);
		}

	}
}
public class EmpBean implements Comparable<EmpBean>
{	int eid;
	String ename;
	public void setEid(int eid)
	{	this.eid=eid;
	}
	public void setEname(String ename)
	{	this.ename=ename;
	}
	public int getEid()
	{return eid;
	}
	public String getEname()
	{return ename;
	}
	public int compareTo(EmpBean o)
	{
		if (eid==o.eid)
		{return 0;
		}
```

```
                if (eid>o.eid)
                {return 1;
                }
                else{return -1;}
        }
};
```