TreeMap:

- In TreeMap keys are maintained in ascending order.
- TreeMap uses balanced binary trees algorithm internally.
- Insertion order is not maintained.
- TreeMap is available since jdk1.2V.
- Null keys are not allowed where as null values are accepted.
- Duplicates values are accepted, If duplicate key is there the previous key-value will be replaced.

♪ HashMap doesn't maintain insertion order

♪ Linked HashMap maintains Insertion Order

♪ TreeMap Maintains Ascending order of keys

Hashtable

- Hashtable is available since jdk1.0V.
- Insertion order is not preserved.
- Heterogeneous objects are allowed for both keys and values.
- Null key (or) null value is not allowed.
- It allows duplicate values with unique keys.
- Every method present inside Hashtable is synchronized
- Default capacity is 11 & load factor is 0.75.

```
Hashtable h=new Hashtable();

Hashtable h=new Hashtable(int initialcapacity);
```

HashMap Vs LinkedHashMap Vs TreeMap Vs Hashtable

Property	HashMap	LinkedHashMap	TreeMap	Hashtable
Insertion order	Not Maintained	Maintained	Not Maintained (But keys are sorted in ascending order)	Not Maintained
Null Keys/Values	Allowed	Allowed	Not Allowed (Null Values are allowed)	Not Allowed
Synchronization	Not Synchronized	Not Synchronized	Not Synchronized	Synchronized

Set (Vs) List

Set	List
A set represents a collection of elements Order of the elements may change in the set.	A List represents ordered collection of elements. List preserves the order of elements in which they are entered.
Set will not allow duplicate values to be stored.	List will allow duplicate values.
Accessing elements by their index (position number) is not possible in case of sets.	Accessing elements by index is possible in lists.

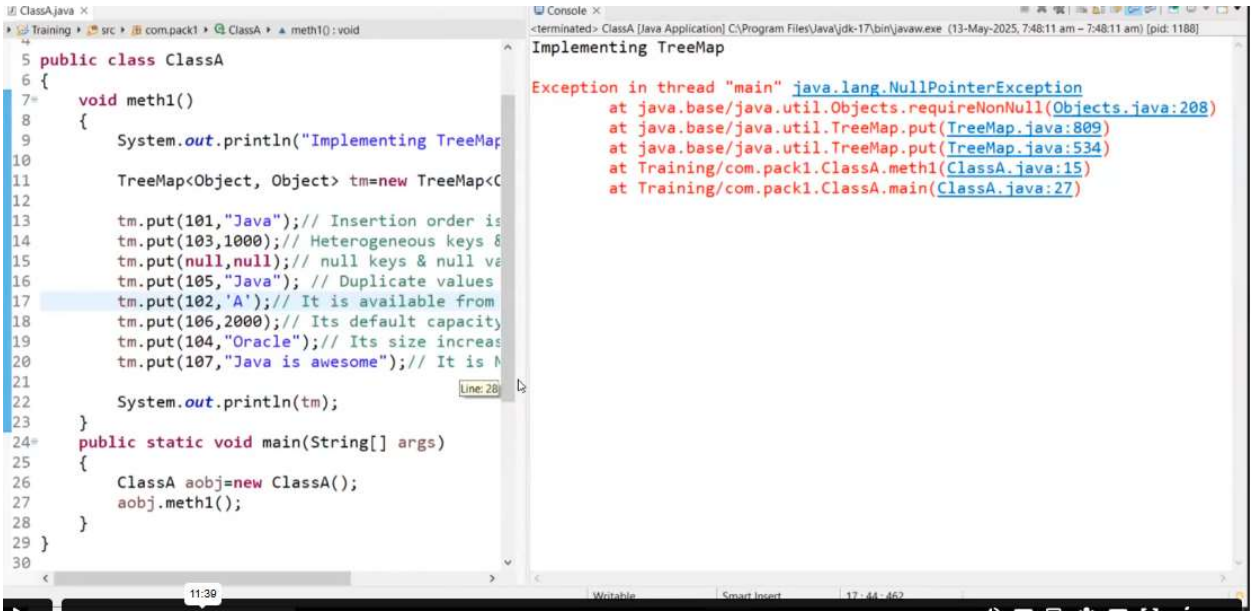
ArrayList (Vs) Vector

ArrayList	Vector
ArrayList object is not synchronized by default.	Vector object is synchronized by default.
Incase of a single thread, using ArrayList is faster than the Vector.	In case of multiple threads, using Vector is advisable. With a single thread, Vector becomes slow.
ArrayList increases its size every time by 50 percent (half).	Vector increases its size every time by doubling it.

HashMap (Vs) HashTable

HashMap	HashTable
HashMap object is not synchronized by default.	Hashtable object is synchronized by default.
In case of a single thread, using HashMap is faster than the Hashtable.	In case of multiple threads, using Hashtable is advisable, with a single thread, Hashtable becomes slow.
HashMap allows null keys and null values to be stored.	Hashtable does not allow null keys or values.

In map we can insert the data but to retrieve the data we need to use collection implementation classes

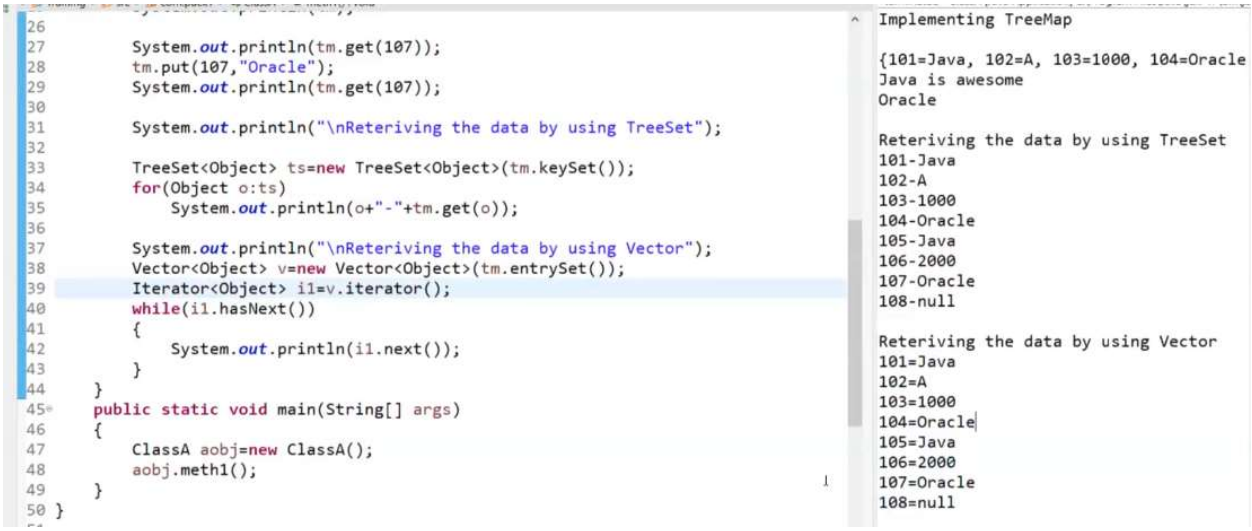


```
public class ClassA
{
    void meth1()
    {
        System.out.println("Implementing TreeMap

        TreeMap<Object, Object> tm=new TreeMap<C

        tm.put(101,"Java");// Insertion order is
        tm.put(103,1000);// Heterogeneous keys &
        tm.put(null,null);// null keys & null va
        tm.put(105,"Java"); // Duplicate values
        tm.put(102,'A');// It is available from
        tm.put(106,2000);// Its default capacity
        tm.put(104,"Oracle");// Its size increas
        tm.put(107,"Java is awesome");// It is N

        System.out.println(tm);
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.meth1();
    }
}
```

```
Implementing TreeMap

Exception in thread "main" java.lang.NullPointerException
        at java.base/java.util.Objects.requireNonNull(Objects.java:208)
        at java.base/java.util.TreeMap.put(TreeMap.java:809)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
        at Training/com.pack1.ClassA.meth1(ClassA.java:15)
        at Training/com.pack1.ClassA.main(ClassA.java:27)
```

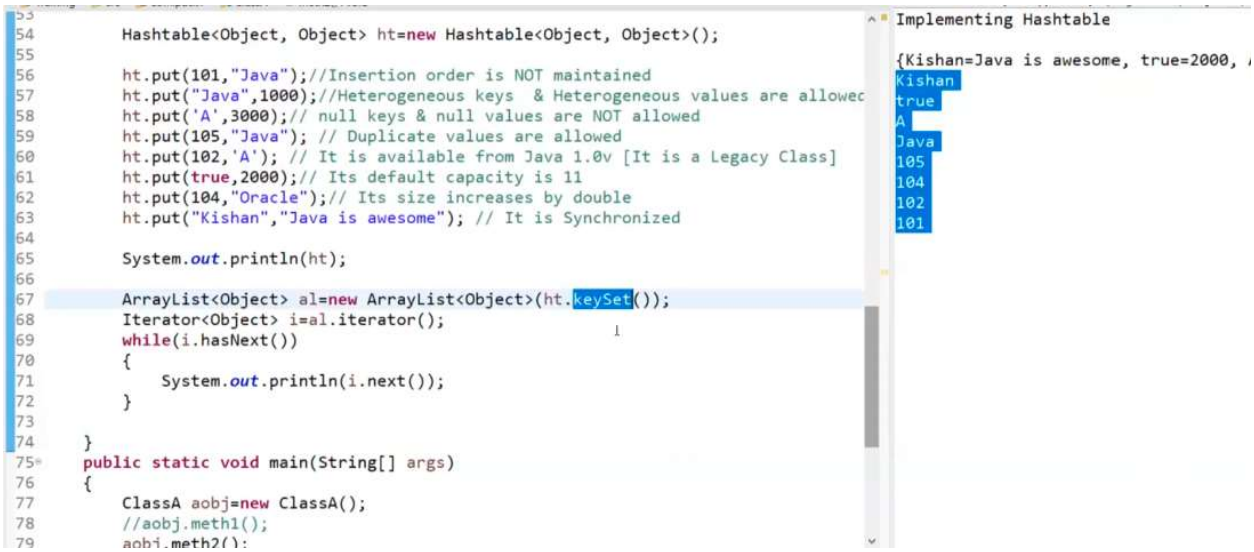
```
public class ClassA
{
    void meth1()
    {
        System.out.println("Imp

        TreeMap<Object, Object>

        tm.put(101,"Java");// I
        tm.put("Java",1000);//
        tm.put(null,null);// nu
        tm.put(105,"Java"); //
        tm.put(102,'A');// It i
        tm.put(true,2000);// Its
        tm.put(104,"Oracle");//
        tm.put("Kishan","Java i

        System.out.println(tm);
    }
    public static void main(Str
    {
        ClassA aobj=new ClassA(
        aobj.meth1();
    }
}
```

```
Exception in thread "main" Implementing TreeMap

java.lang.ClassCastException: class java.lang.Integer cannot be cast to class java.lang.S
        at java.base/java.lang.String.compareTo(String.java:140)
        at java.base/java.util.TreeMap.put(TreeMap.java:814)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
        at Training/com.pack1.ClassA.meth1(ClassA.java:14)
        at Training/com.pack1.ClassA.main(ClassA.java:27)
```



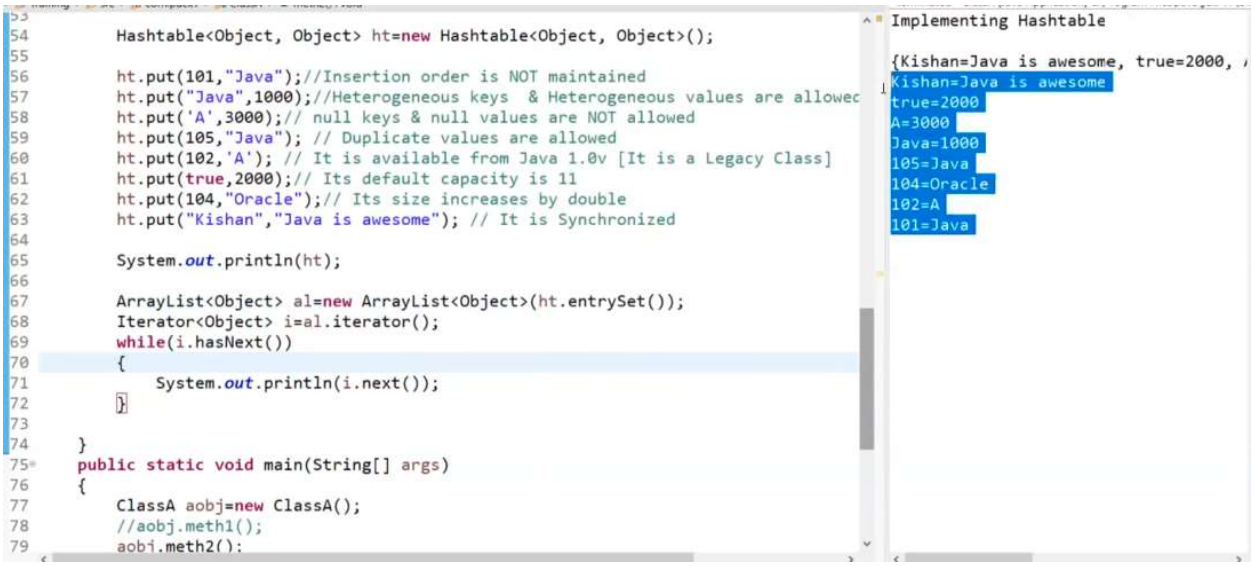
```
        System.out.println(tm.get(107));
        tm.put(107,"Oracle");
        System.out.println(tm.get(107));

        System.out.println("\nReteriving the data by using TreeSet");

        TreeSet<Object> ts=new TreeSet<Object>(tm.keySet());
        for(Object o:ts)
            System.out.println(o+"-"+tm.get(o));

        System.out.println("\nReteriving the data by using Vector");
        Vector<Object> v=new Vector<Object>(tm.entrySet());
        Iterator<Object> i1=v.iterator();
        while(i1.hasNext())
        {
            System.out.println(i1.next());
        }
    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        aobj.meth1();
    }
}
```

```
Implementing TreeMap

{101=Java, 102=A, 103=1000, 104=Oracle
Java is awesome
Oracle

Reteriving the data by using TreeSet
101-Java
102-A
103-1000
104-Oracle
105-Java
106-2000
107-Oracle
108-null

Reteriving the data by using Vector
101=Java
102=A
103=1000
104=Oracle
105=Java
106=2000
107=Oracle
108=null
```



```
        Hashtable<Object, Object> ht=new Hashtable<Object, Object>();

        ht.put(101,"Java");//Insertion order is NOT maintained
        ht.put("Java",1000);//Heterogeneous keys  & Heterogeneous values are allowed
        ht.put('A',3000);// null keys & null values are NOT allowed
        ht.put(105,"Java"); // Duplicate values are allowed
        ht.put(102,'A'); // It is available from Java 1.0v [It is a Legacy Class]
        ht.put(true,2000);// Its default capacity is 11
        ht.put(104,"Oracle");// Its size increases by double
        ht.put("Kishan","Java is awesome"); // It is Synchronized

        System.out.println(ht);

        ArrayList<Object> al=new ArrayList<Object>(ht.keySet());
        Iterator<Object> i=al.iterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }

    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        //aobj.meth1();
        aobj.meth2();
```

```
Implementing Hashtable

{Kishan=Java is awesome, true=2000,
Kishan
true
A
Java
105
104
102
101
```



```
        Hashtable<Object, Object> ht=new Hashtable<Object, Object>();

        ht.put(101,"Java");//Insertion order is NOT maintained
        ht.put("Java",1000);//Heterogeneous keys  & Heterogeneous values are allowed
        ht.put('A',3000);// null keys & null values are NOT allowed
        ht.put(105,"Java"); // Duplicate values are allowed
        ht.put(102,'A'); // It is available from Java 1.0v [It is a Legacy Class]
        ht.put(true,2000);// Its default capacity is 11
        ht.put(104,"Oracle");// Its size increases by double
        ht.put("Kishan","Java is awesome"); // It is Synchronized

        System.out.println(ht);

        ArrayList<Object> al=new ArrayList<Object>(ht.entrySet());
        Iterator<Object> i=al.iterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }

    }
    public static void main(String[] args)
    {
        ClassA aobj=new ClassA();
        //aobj.meth1();
        aobj.meth2();
```

```
Implementing Hashtable

{Kishan=Java is awesome, true=2000,
Kishan=Java is awesome
true=2000
A=3000
Java=1000
105=Java
104=Oracle
102=A
101=Java
```

```java
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.Vector;

public class ClassA
{
    void meth1()
    {
        System.out.println("Implementing TreeMap\n");

        TreeMap<Object, Object> tm=new TreeMap<Object, Object>();

        tm.put(101,"Java");// Insertion order is NOT maintained but sorting order of keys is maintained
        tm.put(103,1000);// Heterogeneous keys are NOT allowed & Heterogeneous values are allowed
        tm.put(108,null);// null keys are NOT allowed & null values are allowed
        tm.put(105,"Java"); // Duplicate values are allowed
        tm.put(102,'A');// It is available from Java 1.2v
        tm.put(106,2000);// Its default capacity is 16
        tm.put(104,"Oracle");// Its size increases  by DOUBLE
        tm.put(107,"Java is awesome");// It is NOT Synchronized

        System.out.println(tm);

        System.out.println(tm.get(107));
        tm.put(107,"Oracle");
        System.out.println(tm.get(107));

        System.out.println("\nReteriving the data by using TreeSet");

        TreeSet<Object> ts=new TreeSet<Object>(tm.keySet());
        for(Object o:ts)
            System.out.println(o+"-"+tm.get(o));

        System.out.println("\nReteriving the data by using Vector");
        Vector<Object> v=new Vector<Object>(tm.entrySet());
        Iterator<Object> i1=v.iterator();
        while(i1.hasNext())
        {
            //System.out.println(i1.next());
            Entry e=(Entry)i1.next();
            System.out.println(e.getKey()+" "+e.getValue());
        }
    }
```

```
50      void meth2()
51      {
52          System.out.println("Implementing Hashtable\n");
53
54          Hashtable<Object, Object> ht=new Hashtable<Object, Object>();
55
56          ht.put(101,"Java");//Insertion order is NOT maintained
57          ht.put("Java",1000);//Heterogeneous keys  & Heterogeneous values are allowed
58          ht.put('A',3000);// null keys & null values are NOT allowed
59          ht.put(105,"Java"); // Duplicate values are allowed
60          ht.put(102,'A'); // It is available from Java 1.0v [It is a Legacy Class]
61          ht.put(true,2000);// Its default capacity is 11
62          ht.put(104,"Oracle");// Its size increases by double
63          ht.put("Kishan","Java is awesome"); // It is Synchronized
64
65          System.out.println(ht);
66
67          ArrayList<Object> al=new ArrayList<Object>(ht.entrySet());
68          Iterator<Object> i=al.iterator();
69          while(i.hasNext())
70          {
71              System.out.println(i.next());
72          }
73
74      }
75      public static void main(String[] args)
```

```
76      {
77          ClassA aobj=new ClassA();
78          //aobj.meth1();
79          aobj.meth2();
80      }
81  }
82
```

```
1  package com.pack1;
2
3  public class ClassA
4  {
5      void meth1()
6      {
7          System.out.println("Implementing Sorting in Array Vs Collection classes\n");
8
9          int arr[]= {1,4,8,2,9,3,7,6,5};
10         System.out.println("Before : "+arr);
11     }
12
13
14     public static void main(String[] args)
15     {
16         ClassA aobj=new ClassA();
17         aobj.meth1();
18
19     }
20 }
```

```
Implementing Sorting in Array Vs Col

Before : [I@3feba861
```

```
Training ▸ src ▸ com.pack1 ▸ ClassA ▸
 1 package com.pack1;
 2
 3 import java.util.Arrays;
 4
 5 public class ClassA
 6 {
 7     void meth1()
 8     {
 9         System.out.println("Implementing Sorting in Array Vs Collection classes\r
10
11         int arr[]= {1,4,8,2,9,3,7,6,5};
12         System.out.println("Before : "+Arrays.toString(arr));
13     }
14
15
16     public static void main(String[] args)
17     {
18         ClassA aobj=new ClassA();
19         aobj.meth1();
20
21     }
22 }
```

```
<terminated> ClassA [Java Application] C:\Program Files\Java\jdk-17\bin\java
Implementing Sorting in Array Vs Collect

Before : [1, 4, 8, 2, 9, 3, 7, 6, 5]
```

```
 1 package com.pack1;
 2
 3 import java.util.Arrays;
 4
 5 public class ClassA
 6 {
 7     void meth1()
 8     {
 9         System.out.println("Implementing Sorting in Array Vs Collection
10
11         int arr[]= {1,4,8,2,9,3,7,6,5};
12         System.out.println("Before sorting: "+Arrays.toString(arr));
13         Arrays.sort(arr);
14         System.out.println("After sorting : "+Arrays.toString(arr));
15
16     }
17
18
19     public static void main(String[] args)
20     {
21         ClassA aobj=new ClassA();
22         aobj.meth1();
23
24     }
25 }
```

```
Implementing Sorting in Array Vs Collection class

Before sorting: [1, 4, 8, 2, 9, 3, 7, 6, 5]
After sorting : [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
 8 {
 9     void meth1()
10     {
11         System.out.println("Implementing Sorting in Array Vs Colle
12
13         int arr[]= {1,4,8,2,9,3,7,6,5};
14         System.out.println("Before sorting: "+Arrays.toString(arr)
15         Arrays.sort(arr);
16         System.out.println("After sorting : "+Arrays.toString(arr)
17
18         ArrayList<Integer> al=new ArrayList<Integer>();
19         al.add(10);
20         al.add(50);
21         al.add(20);
22         al.add(40);
23         al.add(30);
24         System.out.println("\nBefore Sorting : "+al);
25         Collections.sort(al);
26         System.out.println("After Sorting : "+al);
27
28
29     }
30
31
32     public static void main(String[] args)
33     {
```

```
Implementing Sorting in Array Vs Collection classes

Before sorting: [1, 4, 8, 2, 9, 3, 7, 6, 5]
After sorting : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Before Sorting : [10, 50, 20, 40, 30]
After Sorting : [10, 20, 30, 40, 50]
```

Sort() is used for list implementation classes only

```
19
20        ArrayList<Integer> al=new ArrayList<Integer>();
21        al.add(10);
22        al.add(50);
23        al.add(20);
24        al.add(40);
25        al.add(30);
26        System.out.println("\nBefore Sorting : "+al);
27        Collections.sort(al);
28        System.out.println("After Sorting : "+al);
29
30        LinkedHashSet<Integer> lhs=new LinkedHashSet<Integer>();
31        lhs.add(111);
32        lhs.add(444);
33        lhs.add(555);
34        lhs.add(222);
35        lhs.add(333);
36        System.out.println("\nBefore Sorting : "+lhs);
37        //Collections.sort(lhs); // C.E
38        TreeSet<Integer> ts=new TreeSet<Integer>(lhs);
39        System.out.println("After Sorting : "+ts);
40
41
42    }
```

```
Implementing Sorting in Array Vs Collection classes

Before sorting: [1, 4, 8, 2, 9, 3, 7, 6, 5]
After sorting : [1, 2, 3, 4, 5, 6, 7, 8, 9]

Before Sorting : [10, 50, 20, 40, 30]
After Sorting : [10, 20, 30, 40, 50]

Before Sorting : [111, 444, 555, 222, 333]
After Sorting : [111, 222, 333, 444, 555]
```

```
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.Collections;
6 import java.util.LinkedHashSet;
7 import java.util.TreeSet;
8
9 public class ClassA
10 {
11     void meth1()
12     {
13         System.out.println("Implementing Sorting in Array Vs Collection classes\n");
14
15         int arr[]= {1,4,8,2,9,3,7,6,5};
16         System.out.println("Before sorting: "+Arrays.toString(arr));
17         Arrays.sort(arr);
18         System.out.println("After sorting : "+Arrays.toString(arr));
19
20         ArrayList<Integer> al=new ArrayList<Integer>();
21         al.add(10);
22         al.add(50);
23         al.add(20);
24         al.add(40);
25         al.add(30);
26         System.out.println("\nBefore Sorting : "+al);
27         Collections.sort(al);
28         System.out.println("After Sorting : "+al);
29
30         LinkedHashSet<Integer> lhs=new LinkedHashSet<Integer>();
31         lhs.add(111);
32         lhs.add(444);
33         lhs.add(555);
```

```
34          lhs.add(222);
35          lhs.add(333);
36          System.out.println("\nBefore Sorting : "+lhs);
37          //Collections.sort(lhs); // C.E
38          TreeSet<Integer> ts=new TreeSet<Integer>(lhs);
39          System.out.println("After Sorting : "+ts);
40      }
41      public static void main(String[] args)
42      {
43          ClassA aobj=new ClassA();
44          aobj.meth1();
45      }
46  }
```