**Collection Framework – Part\_08**

* **TreeSet:**

1. The underlying data structure is “Balanced Tree”.
2. Duplicate objects are not allowed.
3. Insertion order is not preserved.
4. Heterogeneous objects are not allowed, otherwise we will get runtime exception saying ClassCastException.
5. null insertion possible (only once).
6. TreeSet implements Serializable and Cloneable but not RandomAccess.
7. All objects will be inserted based on some sorting order it may be default natural sorting order or customized sorting order.

**Constructors:**

TreeSet t = new TreeSet();

Creates an empty TreeSet object where the elements will be inserted according to default natural sorting order.

TreeSet t = new TreeSet(Comparator c);

Creates an empty TreeSet object where the elements will

be inserted according to customized sorting order specified

by Compartor object.

TreeSet t = new TreeSet(Collection c);

TreeSet t = new TreeSet(SortedSet s);

Example:

import java.util\*;

class TreeSetDemo{

public static void main(String[] args){

TreeSet t = new TreeSet();

t.add(“A”);

t.add(“a”);

t.add(“B”);

t.add(“Z”);

t.add(“L”);

//t.add(new Integer(10));//CCE

//t.add(null); // NPE

System.out.println(t); [A,B,L,Z,a]

}

}

* **Null Acceptance:**

Rule\_01:

For non-empty TreeSet if we are trying insert null, then we will get NullPointerException.

Rule\_02:

For empty TreeSet as a first element null is allowed but after inserting that null, if we are trying to insert any other then we will get runtime exception saying: NullPointerException

Note\*\*\*

Until 1.6 version null is allowed as first element to the empty TreeSet. But, 1.7 version onwards null is not allowed even as first element that is “null” such type of story not applicable for TreeSet from 1.7 version onwards.

* **Example\_02:**

import java.util.\*;

class TreeSetDemo{

public static void main(String[] args){

TreeSet t = new TreeSet();

t.add(new StringBuffer(“A”));

t.add(new StringBuffer(“Z”));

t.add(new StringBuffer(“L”));

t.add(new StringBuffer(“B”));

System.out.println(t);

}

}

Output:

RE: ClassCastException

Note:

If we are depending on default natural sorting order compulsory the objects should be homogeneous & comparable. Otherwise we will get runtime exception saying: ClassCastException

An object is said to be comparable if and only if corresponding class implements Comparable interface.

String class and all Wrapper classes already implement Comparable interface. But StringBuffer class doesn’t implement Comparable interface. Hence we got ClassCastException in the above example.

* **Comparable (I):**

It is present in java.lang package and it contains only one method

compareTo()

public int compareTo(Object obj)

obj1.compareTo(obj2)

Returns -ve iff obj1 has to come before obj2.

Returns +ve iff obj1 has to come after obj2.

Returns 0 iff obj1 & obj2 are equal.

**Example:**

class Test{

public static void main(String[] args){

System.out.println(“A”.compareTo(“Z”)); // -ve

System.out.println(“Z”.compareTo(“K”)); // +ve

System.out.println(“A”.compareTo(“A”)); //0

System.out.println(“A”.compareTo(null)); //NPE

}

}

* **TreeSet Internal Working:**

If we are depending on default natural sorting order, then while adding objects into the TreeSet JVM will call compareTo() method.

TreeSet t = new TreeSet();

t.add(“K”);

t.add(“Z”); 🡺 “Z”.compareTo(“K”) +ve

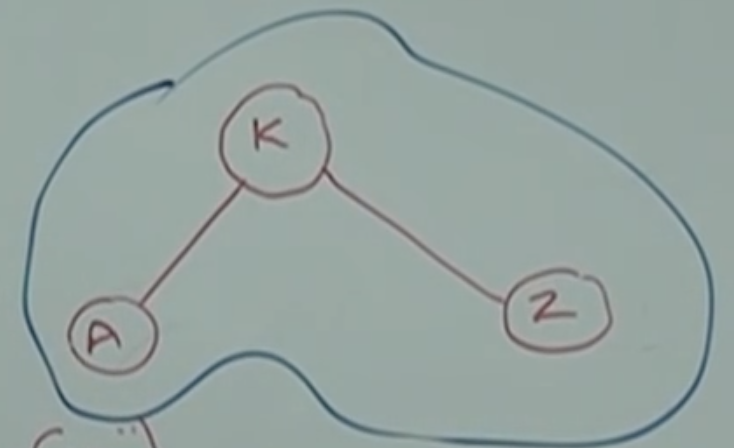
t.add(“A”); 🡺 “A”.compareTo(“K”); -ve

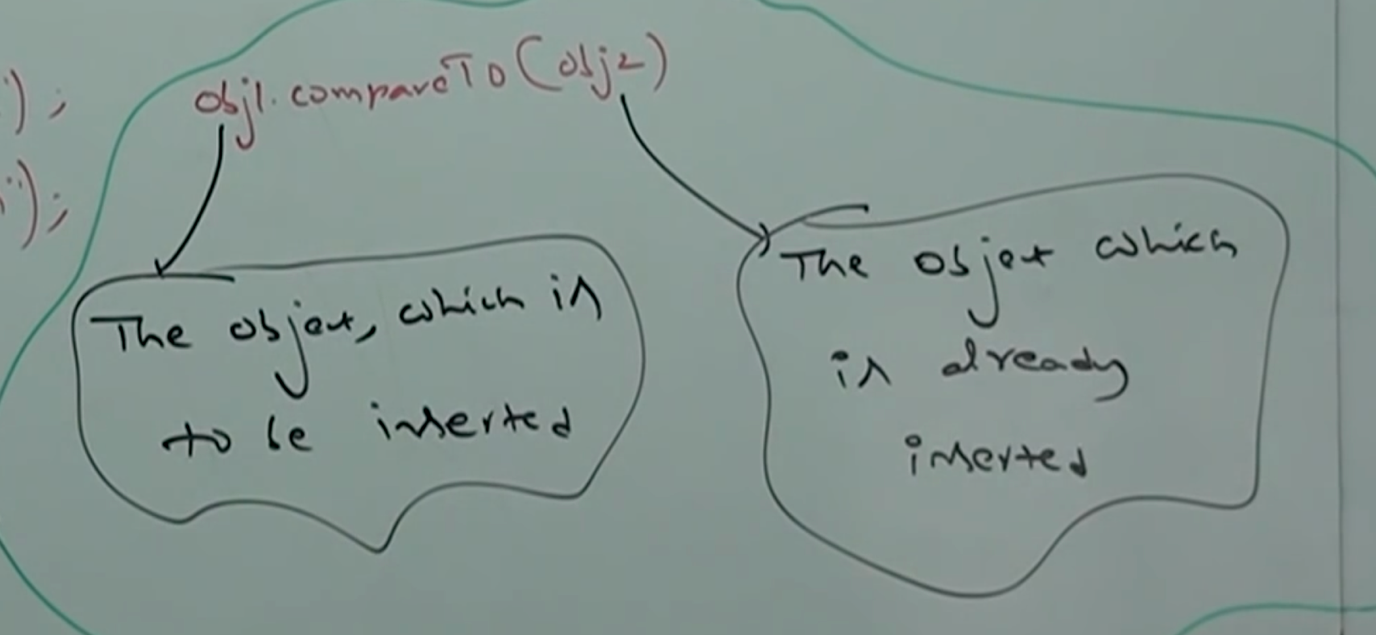
t.add(“A”); 🡺 “A”.compareTo(“K “); -ve

“A”.compareTo(“A”); 0

t.add(“A”);

System.out.println(t); [A,K,Z]



****

**Note:**

If default natural sorting order not available or if we are not satisfied with default natural order then we can go for customized sorting by using Comparator.

Comparable meant for default natural sorting order. Whereas Comparator meant for customized sorting order.