**Collection Framework – Part\_10**

* **Write a program to insert string objects in to the TreeSet where all elements should be inserted according to reverse of alpahabetical order**

import java.util.\*;

class TreeSetDemo{

public static void main(String[] args){

TreeSet t = new TreeSet(new MyComparator());

t.add(“Roja”);

t.add(“Shobakumari”);

t.add(“Rajakumari”);

t.add(“GangaBhavani”);

t.add(“Ramulamma”);

System.out.println(t);

}

}

class MyComparator implements Comparator{

public int compare(Object obj1, Object obj2){

String s1 = obj1.toString();

String s2 = (String)obj2;

return s2.compareTo(s1);

}

}

Output: [Shobarani, Roja, Ramullamma, Rajakumari, GangaBhavani]

* **Write a program to insert StringBuffer objects into the TreeSet where sorting order is alphabetical order:**

import java.util.\*;

class TreeSetDemo{

TreeSet t = new TreeSet(new MyComparator());

t.add(new StringBuffer(“A”));

t.add(new StringBuffer(“Z”));

t.add(new StringBuffer(“K”));

t.add(new StringBuffer(“L”));

System.out.println(t);

}

class MyComparator implements Comparator{

public int compare(Object obj1, Object obj2){

String s1 = obj1.toString();

String s2 = obj2.toString();

return s1.compareTo(s2);

}

}

Output: [A,K,L,Z]

Note:

If we are depending on default natural sorting order, compulsory the objects should be homogeneous and comparable otherwise we will get runtime exception saying: ClassCastException

If we are defining our own sorting by Comparator then objects need not to be comparable and homogeneous that is we can add heterogeneous non comparable objects also.

* **Write a program to insert String and StringBuffer objects into TreeSet where sorting order is increasing length order. If two objects having same length then consider their alphabetical order.**

import java.util.\*;

class TreeSetDemo{

public static void main(String[] args){

TreeSet t = new TreeSet(new MyComparator());

t.add(“A”);

t.add(new StringBuffer(“ABC”));

t.add(new StringBuffer(“AA”);

t.add(“XX”);

t.add(“ABCD”);

t.add(“A”);

System.out.println(t);

}

}

class MyCompartor implement Comparator{

public int compare(Object obj1, Object obj2){

String s1 = obj1.toString();

String s2 = obj2.toString();

int l1 = s1.length();

int l2 = s2.length();

if(l1 < l2)

return -1;

else if(l1 > l2)

return 1;

else

return s1.compareTo(s2);

}

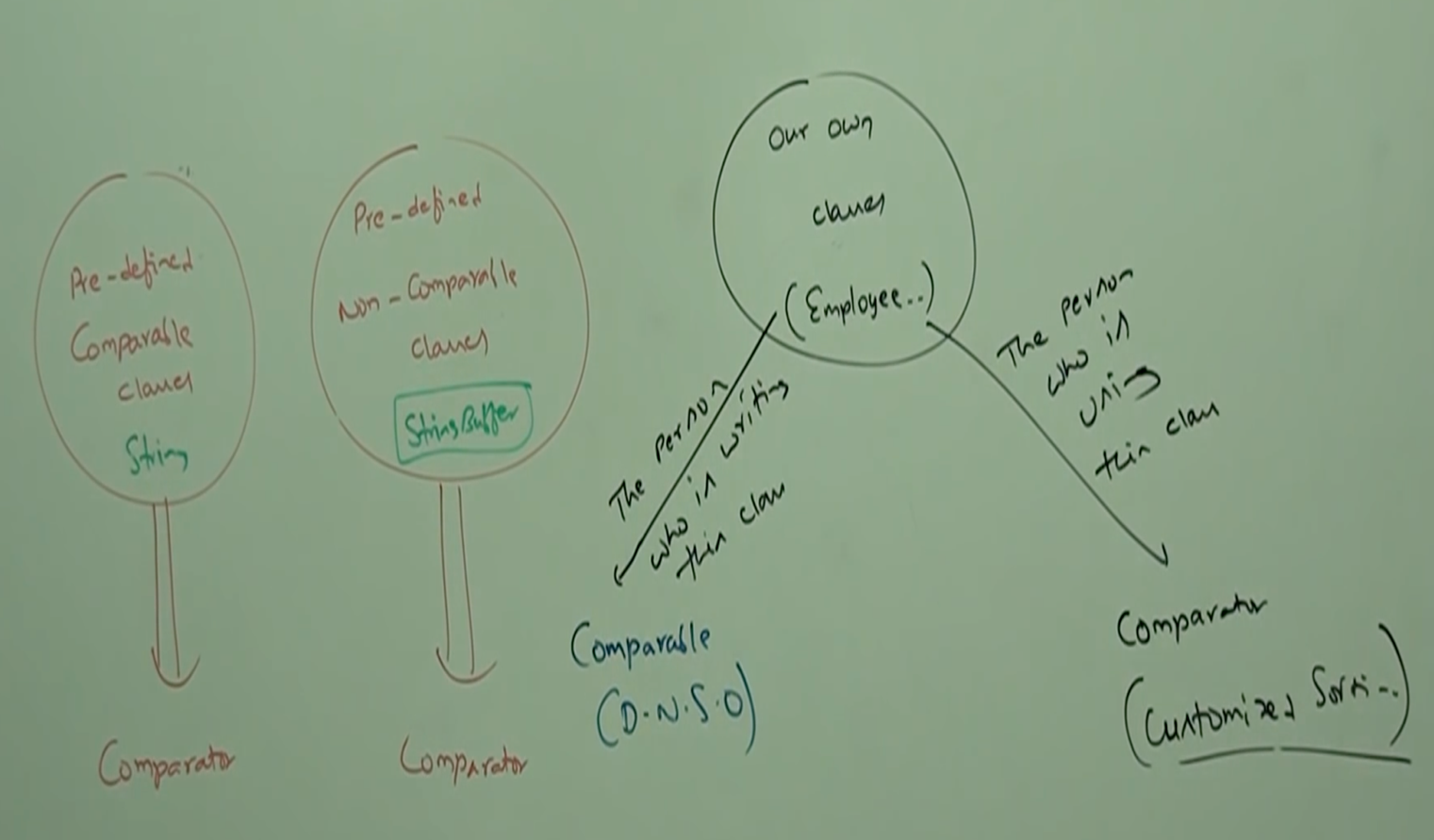
}

Output: [A, AA, XX, ABC, ABCD]

* **Comparable vs Comparator:**

1. For pre-defined Comparable classes default natural sorting order already available. If we are not satisfied with that default natural sorting order then we can define our own sorting by using comparator.
2. For pre-defined non-comparable classes (like StringBuffer) default natural sorting order not already available we can define our own sorting by using Comparator.
3. For our own classes like Employee, the person who is writing the class is responsible to define default natural sorting order by implementing Comparable interface.

The person who is using our class, if he is not satisfied with default natural sorting order then he define his own sorting by using Comparator.



Example

import java.util.\*;

class Employee implements Comparable{

String name;

int eid;

Employee(String name, int eid){

this.name = name;

this.eid = eid;

}

public String toString(){

return name+”---“+eid;

}

public int compareTo(Object obj){

int eid1 = this.eid;

Employee e = (Employee) obj;

int eid2 = e.eid;

if(eid1 < eid2){

return -1;

}

else if(eid1 > eid2){

return 1;

}

else return 0;

}

}

class CompComp{

public static void main(String[] args){

Employee e1 = new Employee(“nag”, 100);

Employee e2 = new Employee(“balaiah”, 200);

Employee e1 = new Employee(“chiru”, 50);

Employee e1 = new Employee(“venki”, 150);

Employee e1 = new Employee(“nag”, 100);

TreeSet t = new TreeSet();

t.add(e1);

t.add(e2);

t.add(e3);

t.add(e4);

t.add(e5);

System.out.println(t);

TreeSet t1 = new TreeSet(new MyComparator());

t.add(e1);

t.add(e2);

t.add(e3);

t.add(e4);

t.add(e5);

System.out.println(t);

}

}

class MyComparator implement Comparator{

public int compare(Object obj1, Object obj2){

Employee e1 = (Employee) obj1;

Employee e2 = (Employee) obj2;

String s1 = e1.name;

String s2 = e2.name;

return s1.compareTo(s2);

}

}

Output:

[chiru--50, nag--100, venki--150, balaiah -- 200]

[balaiah -- 200,chiru -- 50, nag -- 100, venki -- 150]

* **Comparison of Comparable and Comparator:**

|  |  |  |
| --- | --- | --- |
| S.No | Comparable | Comparator |
| 1 | It is meant for default natural sorting order. | It is meant for customized sorting order. |
| 2 | Present in java.lang package | Present in java.util package |
| 3 | It defines only one method compareTo() | It defines two methods compare() and equals() |
| 4 | String and all wrapper classes implement Comparable interface | The only implemented classes of Comparator are Collator RuleBasedCollator |

* **Comparison table of Set implemented classes:**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | HashSet | LinkedHashSet | TreeSet |
| Underlying data structure | Hashtable | LL + Hashtable | Balanced Tree |
| Duplicate objects | Not allowed | Not allowed | Not allowed |
| Insertion Order | Not preserved | Preserved | Not preserved |
| Sorting order | NA | NA | Applicable |
| Heterogeneous objects | Allowed | Allowed | Allowed |
| Null acceptance | Allowed | Allowed | For empty TreeSet as first element null is allowed. |

Note:

For empty TreeSet as the first element null is allowed but this rule is applicable util 1.6 version only from 1.7 version onwards, null is not allowed even as the first element.