TreeSet Examples

Below example shows how to create TreeSet with other collection. By passing another collection to the TreeSet constructor, you can copy entire collections elements to the TreeSet.

```
import java.util.ArrayList;
import java.util.List;
import java.util.TreeSet;

public class MySetWithCollection {
    public static void main(String a[]){
        List<String> li = new ArrayList<String>();
        li.add("one");
        li.add("two");
        li.add("three");
        li.add("four");
        System.out.println("List: "+li);
        //create a treeset with the list
        TreeSet<String> myset = new TreeSet<String>(li);
        System.out.println("Set: "+myset);
    }
}
```

Output:

```
List: [one, two, three, four]
Set: [four, one, three, two]
```

Below example shows how to create TreeSet with other collection. By passing another collection to the TreeSet constructor, you can copy entire collections elements to the TreeSet.

```
import java.util.ArrayList;
import java.util.List;
import java.util.TreeSet;

public class MySetWithCollection {
    public static void main(String a[]){
        List<String> li = new ArrayList<String>();
        li.add("one");
        li.add("two");
        li.add("three");
        li.add("four");
        System.out.println("List: "+li);
        //create a treeset with the list
        TreeSet<String> myset = new TreeSet<String>(li);
        System.out.println("Set: "+myset);
    }
}
```

```
Output:
List: [one, two, three, four]
Set: [four, one, three, two]
```

Below example shows how to read objects using Iterator. By calling iterator() method you will get Iterator object, through which you can iterate through all the elements of the TreeSet.

```
import java.util.Iterator;
import java.util.TreeSet;

public class MySetIteration {

    public static void main(String a[]){

        TreeSet<String> ts = new TreeSet<String>();
        ts.add("one");
        ts.add("two");
        ts.add("three");
        Iterator<String> itr = ts.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

```
Output:
one
three
two
```

The easiest way to remove duplicate entries from the given array is, create TreeSet object and add array entries to the TreeSet. Since the set doesnot support duplicate entries, you will get only unique elements left with TreeSet.

```
import java.util.Arrays;
import java.util.List;
import java.util.TreeSet;
```

```
public class MyArrayDuplicates {

        public static void main(String a[]){
                String[] strArr = {"one","two","three","four","four","five"};
                //convert string array to list
                List<String> tmpList = Arrays.asList(strArr);
                //create a treeset with the list, which eliminates duplicates
                TreeSet<String> unique = new TreeSet<String>(tmpList);
                System.out.println(unique);
        }
}
```

Output:

```
[five, four, one, three, two]
```

The easiest way to find duplicate entries from the given array is, create TreeSet object and add array entries to the TreeSet. Since the set doesnot support duplicate entries, you can easily findout duplicate entries. Below example add each element to the set, and checks the returns status.

```
import java.util.TreeSet;

public class MyDuplicateEntry {

  public static void main(String a[]){
    String[] strArr = {"one","two","three","four","four","five"};
    TreeSet<String> unique = new TreeSet<String>();
    for(String str:strArr){
      if(!unique.add(str)){
        System.out.println("Duplicate Entry is: "+str);
      }
    }
  }
}
```

Output:

```
Duplicate Entry is: four
```

To implement your own sorting functionality with TreeSet, you have to pass Comparator object along with TreeSet constructor call. The Comparator implementation holds the sorting logic. You have to override compare() method to provide the sorting logic. Below example shows how to sort TreeSet using comparator.

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

public class MySetWithCompr {

    public static void main(String a[]){

        TreeSet<String> ts = new TreeSet<String>(new MyComp());
        ts.add("RED");
        ts.add("ORANGE");
        ts.add("BLUE");
        ts.add("GREEN");
        System.out.println(ts);
    }
}

class MyComp implements Comparator<String>{

    @Override
    public int compare(String str1, String str2) {
        return str1.compareTo(str2);
    }

}
```

Output:

```
[BLUE, GREEN, ORANGE, RED]
```

To implement your own sorting functionality with TreeSet on user defined objects, you have to pass Comparator object along with TreeSet constructor call. The Comparator implementation holds the sorting logic. You have to override compare() method to provide the sorting logic on user defined objects. Below example shows how to sort TreeSet using comparator with user defined objects.

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

public class MyCompUserDefine {

    public static void main(String a[]){
        //By using name comparator (String comparison)
        TreeSet<Empl> nameComp = new TreeSet<Empl>(new MyNameComp());
        nameComp.add(new Empl("Ram",3000));
        nameComp.add(new Empl("John",6000));
        nameComp.add(new Empl("Crish",2000));
        nameComp.add(new Empl("Tom",2400));
        for(Empl e:nameComp){
            System.out.println(e);
        }
        System.out.println("===========================");
        //By using salary comparator (int comparison)
        TreeSet<Empl> salComp = new TreeSet<Empl>(new MySalaryComp());
        salComp.add(new Empl("Ram",3000));
        salComp.add(new Empl("John",6000));
        salComp.add(new Empl("Crish",2000));
        salComp.add(new Empl("Tom",2400));
        for(Empl e:salComp){
            System.out.println(e);
        }
    }
}

class MyNameComp implements Comparator<Empl>{

    @Override
    public int compare(Empl e1, Empl e2) {
        return e1.getName().compareTo(e2.getName());
    }
}

class MySalaryComp implements Comparator<Empl>{

    @Override
    public int compare(Empl e1, Empl e2) {
        if(e1.getSalary() > e2.getSalary()){
            return 1;
        } else {
            return -1;
        }
    }
}

class Empl{

    private String name;
    private int salary;
    public Empl(String n, int s){
        this.name = n;
        this.salary = s;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    public String toString(){
        return "Name: "+this.name+"-- Salary: "+this.salary;
    }
}
```

Output:

```
Name: Crish-- Salary: 2000
Name: John-- Salary: 6000
Name: Ram-- Salary: 3000
Name: Tom-- Salary: 2400
===========================
Name: Crish-- Salary: 2000
Name: Tom-- Salary: 2400
Name: Ram-- Salary: 3000
Name: John-- Salary: 6000
```

To avoid duplicate user defined objects in TreeSet, you have to implement Comparator interface with equality verification. Below example gives a sample code to implement it.

```
import java.util.Comparator;
import java.util.Set;
import java.util.TreeSet;

public class MyUserDuplicates {

    public static void main(String a[]){
        Set<Emp> ts = new TreeSet<Emp>(new EmpComp());
        ts.add(new Emp(201,"John",40000));
        ts.add(new Emp(302,"Krish",44500));
        ts.add(new Emp(146,"Tom",20000));
        ts.add(new Emp(543,"Abdul",10000));
        ts.add(new Emp(12,"Dinesh",50000));
        //adding duplicate entry
        ts.add(new Emp(146,"Tom",20000));
        //check duplicate entry is there or not
        for(Emp e:ts){
            System.out.println(e);
        }
    }
}

class EmpComp implements Comparator<Emp>{

    @Override
    public int compare(Emp e1, Emp e2) {
        if(e1.getEmpId() == e2.getEmpId()){
            return 0;
        } if(e1.getEmpId() < e2.getEmpId()){
            return 1;
        } else {
            return -1;
        }
    }
}

class Emp {

    private int empId;
    private String empName;
    private int empSal;

    public Emp(int id, String name, int sal){
        this.empId = id;
        this.empName = name;
        this.empSal = sal;
    }

    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public int getEmpSal() {
        return empSal;
    }
    public void setEmpSal(int empSal) {
        this.empSal = empSal;
    }

    public String toString(){
        return empId+" : "+empName+" : "+empSal;
    }
}
```

Output:

```
543 : Abdul : 10000
302 : Krish : 44500
201 : John : 40000
146 : Tom : 20000
12 : Dinesh : 50000
```