Comparable

Vs

Comparator

21st dec

23 December 2022 23:12

Comparable vs Comparator
=======================
public TreeSet();

|=> When we use the above constructor,JVM will internally use Comparable interface method to sort the Objects
based on default natural sorting order.

What is Comparable interface?

It is a functiaonal interace present in java.lang package.

This interface is internally used by TreeSet object during sorting process of the Object.

```
@FunctionalInterface
public interface java.lang.Comparable<T> {
    public abstract int compareTo(T);
}
```

eg#1.

```
class Test
{
    public static void main(String[] args)
    {
        //Sorting of objects will happen based on default natural sorting order
        TreeSet ts = new TreeSet();
            ts.add("A");
            ts.add("Z");
            ts.add("L");
            ts.add("B");
            ts.add(null);//NullPointerException
            ts.add(10);//ClassCastException
            System.out.println(ts);//[A,B,L,Z]
    }
}
```

Note:

If we are keeping the data inside TreeSet object, then the data should be

a. Homogenous ====> because it uses compareTo() to sort the Object

b. The object should compulsorily implements an interface called "Comparable".
if we fail to do so , it would result in "ClassCastException".

eg#2.

```
class Test
{
    public static void main(String[] args)
    {
        //Sorting of objects will happen based on default natural sorting order
        TreeSet ts = new TreeSet();

        ts.add(new StringBuffer("A"));
        ts.add(new StringBuffer("Z"));
        ts.add(new StringBuffer("L"));
        ts.add(new StringBuffer("B"));
        System.out.println(ts);//ClassCastException

    }
}
```

note: All Wraper classes and String class has implemented "Comparable" interface.

StringBuffere class has not implemented Comparable interface, so the above program would result in "ClassCastException".

Note:

If we are Depending on Default Natural Sorting Order Compulsory Objects should be Homogeneous and Comparable.

Otherwise we will get RE: ClassCastException.

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

All Wrapper Classes, String Class Already Implements Comparable Interface. But StringBuffer Class doesn't Implement Comparable Interface.

```
@FunctionalInterface
public interface Comparable<T> {
  public abstract int compareTo(T);
}
obj1.compareTo(obj2)
            |=> returns -ve value, if obj1 has to come before obj2
            |=> returns +ve value, if obj1 has to come after  obj2
            |=> returns 0 value, if both obj1 and obj2 are equal
import java.util.*;
class Test {
    public static void main(String[] args) {
        TreeSet ts =new TreeSet();
        ts.add("A");
        ts.add("Z");
        ts.add("L");
        ts.add("K");
        ts.add("B");
        System.out.println(ts);
    }
}
```

Comparable (I):

Comparable Interface Present in java.lang Package and it contains Only One Method compareTo().

obj1.compareTo(obj2)

Returns –ve if and Only if obj1 has to Come Before obj2.

Returns +ve if and Only if obj1 has to Come After obj2.

Returns 0 if and Only if obj1 and obj2 are Equal.

eg#1.

```
 System.out.println("A".compareTo("Z")); //-ve value
System.out.println("Z".compareTo("K")); // +value
System.out.println("Z".compareTo("Z")); // zero
System.out.println("Z".compareTo(null));//NPE
```

Wheneverwe are Depending on Default Natural Sorting Order and if we are trying to Insert Elements then Internally JVM will

Call compareTo() to IdentifySorting Order.

```
TreeSet t = new TreeSet();
t.add("K");
t.add("Z"); "Z".compareTo("K");
t.add("A"); "A".compareTo("K");
t.add("A"); "A".compareTo("A");
System.out.println(t);//[A,K,Z] => Sorting is ascending order
```

Note:

For String default natural sorting order is "Ascending order".

For Number default natural sorting order is "Ascending order"

Comparator(I)

=============

Note: If we are Not satisfied with Default Natural Sorting Order OR if Default Natural Sorting Order is Not Already Available then

we can Define Our Own Sorting by using Comparator Object.

```
    public interface java.util.Comparator<T> {
    public abstract int compare(T, T);
    public abstract boolean equals(java.lang.Object);
  }

  Comparator (I):
  This Interface Present in java.util Package.
  Methods: It contains 2 Methodscompare() and equals().

  public  int compare(Object obj1, Object obj2);
      Returns –ve if and Only if obj1 has to Come Before obj2.
      Returns +ve if and Only if obj1 has to Come After obj2.
      Returns 0 if and Only if obj1 and obj2 are Equal.

  public boolean equals(Object o);
      Whenever we are implementing Comparator Interface Compulsory we should Provide
Implementation for compare().
Implementing equals() is Optional because it is Already Available to Our Class from
Object Class through Inheritance.

import java.util.*;
class     TreeSetDemo {
```

```
public static void main(String[] args) {
        TreeSet t = new TreeSet(new MyComparator());//Line-1
            t.add(10);
            t.add(0);
            t.add(15);
            t.add(5);
            t.add(20);
            t.add(20);
            System.out.println(t);//[20, 15, 10, 5, 0]
    }
}
class  MyComparator implements Comparator {
        public  int compare(Object obj1, Object obj2) {
                Integer i1 = (Integer)obj1;
                Integer i2 = (Integer)obj2;
                if(i1 < i2)
                    return +1;
                else if(i1 > i2)
                    return -1;
                else
                    return 0;
        }
}
```

At Line l if we are Not Passing Comparator Object as an Argument then Internally JVM will Call compareTo(),

Which is Meant for Default Natural Sorting Order (Ascending Order).

In this Case the Output is [0, 5, 10, 15, 20].

At Line l if we are Passing Comparator Object then JVM will Call compare() Instead of compareTo().

Which is Meant for Customized Sorting (can be Ascending /Descending Order).

In this Case the Ouput is [20, 15, 10, 5, 0]

Various Possible Implementations of compare():

==

```
public int compare(Object obj1, Object obj2) {
    Integer I1 = (Integer)obj1;
    Integer I2 = (Integer)obj2;
    return I1.compareTo(I2);
    return -I1.compareTo(I2);
    return I2.compareTo(I1);
    return -I2.compareTo(I1);
    return +1;
    return -1;
    return 0;
}
Output:
1. Ascending order
2. Descending order
```

```
3. Descending order
4. Ascending order
5. insertion order
6. reverse of insertion order
7. only first element will be inserted.
```

Write a Program to Insert String Objects into the TreeSet where the Sorting Order is of Reverse of Alphabetical Order.

```
import java.util.*;
class  TreeSetDemo {
    public static void main(String[] args) {
            TreeSet t = new TreeSet(new MyComparator());
            t.add("sachin");
            t.add("ponting");
            t.add("sangakara");
            t.add("fleming");
            t.add("lara");
            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);
            //return -s1.compareTo(s2);
    }
}
```

Write a Program to Insert StringBuffer Objects into the TreeSet where Sorting Order is Alphabetical Order.

```
import java.util.*;
class  TreeSetDemo {
    public static void main(String[] args) {
            TreeSet t = new TreeSet(new MyComparator1());
            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 MyComparator1 implements Comparator {
    publicint compare(Object obj1, Object obj2) {
            String s1 = obj1.toString();
            String s2 = obj2.toString();
            return s1.compareTo(s2); //[A, K, L, Z]
}
}
```

Write a Program to Insert String and StringBuffer Objects into the TreeSet where Sorting Order is Increasing Length Order.

If 2 Objects having Same Length then Consider their Alphabetical Order.

eg: A,ABC,AA,XX,ABCE,A

output: A,AA,XX,ABC,ABCE

```
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("ABCE");
            t.add("A");
            System.out.println(t);
    }
}
class  MyComparator implements Comparator {
    public  int compare(Object obj1, Object obj2) {
        String s1 = obj1.toString();
        String s2 = obj2.toString();
        int i1 = s1.length();
        int i2 = s2.length();
        if(i1 < i2)return -1;
        else if(i1 > i2)return 1;
        else    return s1.compareTo(s2);
    }
}
```

Note:

if we are use TreeSet(), then the condition is

a. Object should be homogenous.

b. Object should be comparable(class should implement Comparable(I)).

if we are use TreeSet(Comparator c) then what is the condition?

a. Object need not be homogenous.

b. Object need not implement Comparable.

When to go for Comparable interface and When to go Comparator interface?

Ans. Predefined Comparable classes like String,Wrapper class ====> Default natural sorting is already available

if we are not happy with natural sorting order, we want customization then we need to go for "Comparator(I)".

For Predefined Non-Comparable class like StringBuffer => Comparator(I) is used for both natural sorting order and customized sorting order.

For userdefined class like Employee,Student =====> Developer if he comes up with own logic of sorting,then he should implement Comparable(I) and give it as a ready made logic.

Nitin.M
======

```
class Employee implements Comparable
{
    int id;
    String name;
    int age;

    public int compareTo(Object obj){
        //sorting is done based on "id"
        ;;;;;
    }
}
```

If the developer who is using Employee class, if he is not interested with sorting based on "id" given by the api, then he can
use "Comparator".

When we go for Comparable and When we go for Comparator.
Comparable Vs Comparator.

=> For Predefined Comparable Classes (Like String) Default Natural Sorting Order is Already Available. If we are Not satisfied
    with that we can Define Our Own Sorting by Comparator Object.
=> For Predefine Non- Comparable Classes (Like StringBuffer) Default Natural Sorting Order is Not Already Available.
    If we want to Define Our Own Sorting we can Use Comparator Object.
=> For Our Own Classes (Like Employee) the Person who is writing Employee Class he is Responsible to Define Default Natural
    Sorting Order by implementing Comparable Interface.
=> The Person who is using Our Own Class if he is Not satisfied with Default Natural Sorting Order he can Define his Own
    Sorting by using Comparator Object.
    If he is satisfied with Default Natural Sorting Order then he can Use Directly Our Class.

Write a Program to Insert Employee Objects into the TreeSet where DNSO is Based on Ascending Order of EmployeeId and

Customized Sorting Order is Based on Alphabetical Order of Names:

DNSO -> Default Natural Sorting Order

```
import  java.util.*;
class Employee implements Comparable {
        String name;
        int eid;
        Employee(String name, inteid) {
            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     Test {
    public static void main(String[] args) {

        Employee e1 = new Employee("sachin", 10);
        Employee e2 = new Employee("ponting", 14);
        Employee e3 = new Employee("lara", 9);
        Employee e4 = new Employee("flintoff", 17);
        Employee e5 = new Employee("anwar", 23);

        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());
        t1.add(e1);
        t1.add(e2);
        t1.add(e3);
        t1.add(e4);
        t1.add(e5);
        System.out.println(t1);
    }
}
class    MyComparator implements 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);
    }
}
```
Comparison of Comparable and Comparator.

Comparable(I)

Present in java.lang Package

It is Meant for Default Natural Sorting Order.

Defines Only One Method compareTo()

All Wrapper Classes and String Class implements Comparable Interface.

Comparator(I)

Present in java.util Package

It is Meant for Customized Sorting Order.

Defines 2 Methods compare() and equals().

The Only implemented Classes of Comparator are Collator and RuleBaseCollator.