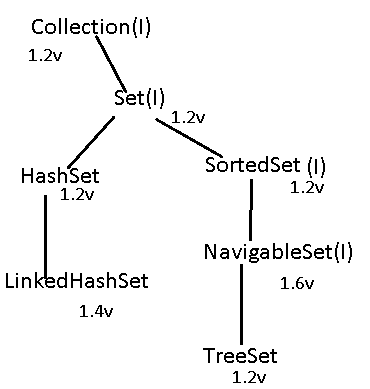
**Set (I)**



**-->** Set is child interface of collection.

**-->** If we want to represent a group of individual objects as a single entity where duplicates are not allowed and insertion order not preserved.

🡪 Set interface doesn’t contain any new method and we have to use only collection interface methods.

**HashSet():**

🡪 The underlaying data structure is Hash table.

🡪 Duplicates objects are not allowed.

🡪 Insertion order is not preserved and it is based on hash code of objects.

🡪 null insertion is possible only once.

🡪 Heterogenous objects are allowed.

🡪 Implements Serializable, Cloneable but not RandomAccess interface.

🡪 HashSet is the best choice is the best choice if our frequent operation is search operation.

**Note:**

In HashSet duplicates are not allowed.

If we are trying to insert duplicate then we won’t get any compile time or run time error and add method simply returns false.

Ex:

HashSet h = new HashSet();  
System.*out*.println(h.add('A'));  
System.*out*.println(h.add('A'));

O/P:

true

false

**Constructors:**

1. HashSet h = new HashSet();

Creates an empty HashSet with default initial capacity 16 and default fill ratio .75.

1. HashSet h = new HashSet(int initialCapacity);

Creates an empty HashSet object with specified initial capacity and default fill ratio .75.

1. HashSet h = new HashSet(int initialcapacity, float fillratio);

Creates an empty HashSet object with specified initial capacity and specified fill ratio.

1. HashSet h = new HashSet(Collection c);

Creates an equivalent HashSet for the given collection.

This constructor mean for interconversion between collection object.

Filling ratio:

After filling how much ratio a new HashSet object will be created. It is also called Load Factor.

Ex:

package com.company;  
import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 HashSet h = new HashSet();  
 h.add("B");  
 h.add("C");  
 h.add("D");  
 h.add("Z");  
 h.add(null);  
 h.add(10);  
 System.*out*.println(h.add("Z"));  
 System.*out*.println(h);  
 }  
}

O/P:

false

[null, B, C, D, Z, 10]

**LinkedHashSet() :**

🡪 It is the child class of HashSet.

🡪 It is exactly same as HashSet(including constructors and methods) except the following difference.

|  |  |
| --- | --- |
| **HashSet** | **LinkedHashSet** |
| 1. The underlaying data structure is Hash table. 2. Insertion order not preserved. 3. Introduced in 1.2 version. | 1. The underlaying data structure is a combination of Linked List Hash table. 2. Insertion order preserved. 3. Introduced in 1.4 version. |

In the above program, if we replace HashSet with LinkedHashSet then the output is

[B, C, D, Z, null, 10]

i.e. insertion order preserved.

Note: In general we can use LInkedHashSet to develop cache based application where duplicates are not allowed and insertion order preserved.

**SortedSet:**

SortedSet is the child interface of Set.

If we want to represent a group of individual objects according to some sorting order without duplicates then we should go for SortedSet.

SortedSet interface defines the following specific methods:

1. Object first()

Returns first element in the SortedSet.

1. Object last()

Returns last element in the SortedSet.

1. SortedSet headSet(Object obj)

Returns SortedSet whose elements are less than obj.

1. SortedSet tailSet(Object obj)

Returns SortedSet whose elements are greater than or equal to obj.

1. SortedSet subset(Object obj1, Object obj2)

Returns SortedSet whose elements are greater than or equal to obj1 and less then obj2.

1. Comparator comparator()

Returns Comparator Object that describes underlaying sorting technique.

If we are using default natural sorting order then we will get null.

Ex:

Let the SortedSet is [100, 101, 104, 106, 110, 115,120]

1. first() // 100
2. last() // 120
3. headset(106) //[100, 101 ,104 ]
4. tailSet(106) // [106, 110, 115, 120]
5. subset(101, 115) //[101, 104, 106, 110]
6. comparator() //null

**TreeSet:**

The underlaying data structure is balanced tree.

Duplicate objects are not allowed.

Insertion order not preserved.

Heterogenous objects are not allowed otherwise we will get runtime exception saying class caste exception.

Null insertion possible (only once).

TreeSet implements Serializable and Cloneable but not RandomAccess.

All objects will be inserted based on some it may be default natural sorting order or customized sorting order.

**Constructors:**

1. TreeSet t = new TreeSet();

It creates an empty TreeSet Object where the element will be inserted according to default natural sorting order.

1. TreeSet t = new TreeSet(Comparator c);

It creates an empty TreeSet object where the elements will be inserted according to customized sorting order specified by Comparator object.

1. TreeSet t = new TreeSet(Collection c);
2. TreeSet t = new TreeSet(SortedSet s);

Ex:

TreeSet t = new TreeSet();  
t.add("A");  
t.add("a");  
t.add("B");  
t.add("Z");  
t.add("L");  
System.*out*.println(t);

O/p:

[A, B, L, Z, a]

**null acceptance:**

Case – 1

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

Case – 2

For empty TreeSet as the 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 the first element to the empty TreeSet, but from 1.7 version onwards null is not allowed even as the first element i.e. ‘null’ such type of story not applicable for TreeSet from 1,7 version onwards.

Ex: TreeSet t = new TreeSet();  
t.add(new StringBuffer("A"));  
t.add(new StringBuffer("B"));  
t.add(new StringBuffer("L"));  
t.add(new StringBuffer("Z"));  
t.add(new StringBuffer("a"));  
System.*out*.println(t);

O/P: [A, B, L, Z, a]

If we are depending on default natural sorting order compulsory the objects should be homogenous and comparable otherwise we will get Runtime Exception saying ClassCasteException

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

String class and all Wrapper classes already implements Comparable interface but before java 11 version StringBuffer class doesn’t implements Comparable interface hence we will get ClassCasteException in the above example.

After 11 version we wont get Exception we will get output according to default natural sorting order.

**Comparable (I):**

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

Prototype: **public int compareTo(Object obj)**

obj1.compareTo(obj2)

Returns negative if obj1 has to come before obj2

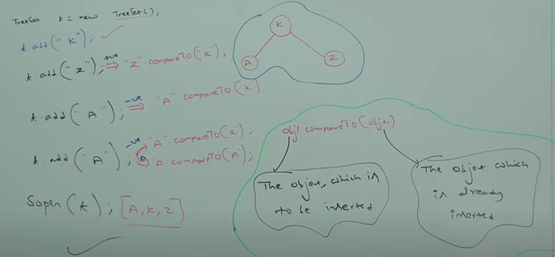
Returns positive if obj1 has to come after obj2

Returns 0 if obj1 and obj2 are equal

Ex:

System.*out*.println("A".compareTo("Z")); //-25  
System.*out*.println("Z".compareTo("K")); //15  
System.*out*.println("A".compareTo("A")); //0  
System.*out*.println("A".compareTo(null)); //RE: NullPointerException

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



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

Comparable meant for default natural sorting order where as Comparator meant for customized sorting order.

**Comparator:**

Comparator present in java.util package and it defines two method compare and equals.

Methods:

1. public int compare(Object obj1, Object obj2)

it returns negative iff obj1 has to come before obj2

it returns positive iff obj1 has to come after obj2

returns 0 iff obj1 and obj2 are equal

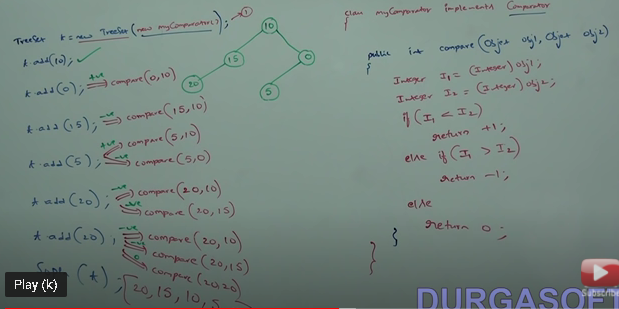
1. public boolean equals(Object obj)

Whenever we are implementing Comparable interface compulsory we have to provide implementation only for compare() method and we are not required to provide implementation for equals method because it is already available to our class from object class through inheritance.

Ex: Wap to insert integer object into the TreeSet where the sorting is descending order.

package com.company;  
import java.util.\*;  
public class Main {  
 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);  
 }  
}  
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;  
 }  
}

Explanation:



At line 1 if we are not passing Comparator object then internally jvm will call compareTo() which is meant for default natural sorting order. In this case the output is [0, 5, 10, 15, 20]

At line 1 if we are passing comparator object then jvm will call compare() method which is meant for customized sorting in this case the output is [20, 15, 10, 5, 0].

**Various possible implementation of compare() method:**

1. return I1.compareTo(I2); Default natural sorting order

Ascending order [0, 5, 10, 15, 20]

1. return I1.compareTo(I2); Descending order [20, 15, 10, 5, 0]
2. return I2.compareTo(I1); Descending order [20, 15, 10, 5, 0]
3. return -I2.compareTo(I1); Ascending order[0, 5, 10, 15, 20]
4. return +1; [Insertion Order] [10, 0, 15, 5, 20, 20]
5. return -1; [Reverse of Insertion Order] [20, 20, 5, 15, 0, 10]
6. return 0; [Only first element will be inserted & all remaining are duplicates] [10]

Ex: WAP to insert String object in to the TreeSet where all elements should be inserted according to reverse of alphabetical order.

package com.company;  
import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 TreeSet t = new TreeSet(new MyComparator());  
 t.add("Roja");  
 t.add("ShobhaRani");  
 t.add("RajaKumari");  
 t.add("GangaBhabani");  
 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);  
 }  
}

O/P:

[ShobhaRani, Roja, Ramulamma, RajaKumari, GangaBhabani]

Ex: WAP to insert StringBuffer objects into the TreeSet where sorting order is alphabetical order.

package com.company;  
import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 TreeSet t = new TreeSet();  
 t.add(new StringBuffer("A"));  
 t.add(new StringBuffer("Z"));  
 t.add(new StringBuffer("k"));  
 t.add(new StringBuffer("L"));  
 t.add(new StringBuffer("O"));  
 System.*out*.println(t);  
 }  
}

O/P:

[A, L, O, Z, k]

Or;

package com.company;  
import java.util.\*;  
public class Main {  
 public static void main(String[] args) {  
 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"));  
 t.add(new StringBuffer("O"));  
 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);  
 }  
}

O/P:

[A, L, O, Z, k]

**Note :** We won’t get any ClassCasteException in the first program because from java 11 StringBuffer also implementing comparable interface. So according to default natural sorting order we will get output.

The second program is valid for previous versions of java. Otherewise we will get runtime Exception saying ClassCasteException.

Ex:

WAP 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.

package com.company;  
import java.util.\*;  
public class Main {  
 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 MyComparator implements 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);  
 }  
}

O/P:

[A, AA, XX, ABC, ABCD]

**Comparable vs Comparator:**

1. For predefined Comparable classes default natural sorting order already available.

If we are not satisfied with that default natural sorting order then we can define our sorting by using Comparator.

1. For predefined non-comparable classes default natural sorting order not already available. We can define our own sorting by using comparator.
2. 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 not satisfied with default natural sorting then he can define his own sorting by using comparator.

Ex:

import java.util.\*;  
public class Employee {  
 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 e3 = new Employee("chiru",50);  
 Employee e4 = new Employee("venki",150);  
 Employee e5 = 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 EmpComparator());  
 t1.add(e1);  
 t1.add(e2);  
 t1.add(e3);  
 t1.add(e4);  
 t1.add(e5);  
 System.*out*.println(t1);  
 }  
}  
class EmpComparator 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 | Comparator |
| 1. It is meant for default natural sorting order. 2. Present in java.lang package. 3. It defines only one method i.e. compareTo(). 4. String and StringBuffer and all wrapper classes implements this. | 1. It is meant for customized sorting order. 2. Present in java.util package. 3. It defines two methods i.e. compare() and equals(). 4. The only implemented classes are Comparator or collector or rule based colletor. |

**Comparison table of Set related classes :**

|  |  |  |  |
| --- | --- | --- | --- |
| Property | HashSet | LinkedHashSet | TreeSet |
| 1. Underlaying DS 2. Duplicate objects 3. Insertion order 4. Sorting order 5. Heterogenous Objects 6. null acceptance | Hashtable  NA  Not preserved  NA  Allowed  Allowed | LinkedList + Hashtable  NA  Preserved  NA  Allowed    Allowed | Balanced Tree  NA  Not preserved  Applicable  NA  For Empty TreeSet as first element until 1.6 version  From 1.7 version onwards not allowed. |