**Java Collections – Set**

A Set is a Collection that cannot contain duplicate elements. There are three main implementations of Set interface: HashSet, TreeSet, and LinkedHashSet.

* HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration.
* TreeSet, which stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashSet.
* LinkedHashSet, which is implemented as a hash table with a linked list running through it, orders its elements based on the order in which they were inserted into the set (insertion-order).

**HashSet Class in Java with example**

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class is not synchronized. However it can be synchronized explicitly like this: Set s = Collections.synchronizedSet(new HashSet(...));

**Points to Note about HashSet:**

1. HashSet doesn’t maintain any order, the elements would be returned in any random order.
2. HashSet doesn’t allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
3. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
4. HashSet is non-synchronized.
5. The iterator returned by this class is fail-fast which means iterator would throw ConcurrentModificationException if HashSet has been modified after creation of iterator, by any means except iterator’s own remove method.

**HashSet Example**

import java.util.HashSet;

public class HashSetExample {

public static void main(String args[]) {

// HashSet declaration

HashSet<String> hset =

new HashSet<String>();

// Adding elements to the HashSet

hset.add("Apple");

hset.add("Mango");

hset.add("Grapes");

hset.add("Orange");

hset.add("Fig");

//Addition of duplicate elements

hset.add("Apple");

hset.add("Mango");

//Addition of null values

hset.add(null);

hset.add(null);

//Displaying HashSet elements

System.out.println(hset);

}

}

**Output:**

[null, Mango, Grapes, Apple, Orange, Fig]

As you can see there all the duplicate values are not present in the output including the duplicate null value.

**HashSet Methods:**

1. **boolean add(Element  e)**: It adds the element e to the list.
2. **void clear()**: It removes all the elements from the list.
3. **Object clone()**: This method returns a shallow copy of the HashSet.
4. **boolean contains(Object o)**: It checks whether the specified Object o is present in the list or not. If the object has been found it returns true else false.
5. **boolean isEmpty()**: Returns true if there is no element present in the Set.
6. **int size()**: It gives the number of elements of a Set.
7. **boolean(Object o)**: It removes the specified Object o from the Set.

**LinkedHashSet Class in Java with Example**

 LinkedHashSet is also an implementation of Set interface, it is similar to the HashSet and TreeSet except the below mentioned differences:

1. HashSet doesn’t maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

**Example of LinkedHashSet:**

import java.util.LinkedHashSet;

public class LinkedHashSetExample {

public static void main(String args[]) {

// LinkedHashSet of String Type

LinkedHashSet<String> lhset = new LinkedHashSet<String>();

// Adding elements to the LinkedHashSet

lhset.add("Z");

lhset.add("PQ");

lhset.add("N");

lhset.add("O");

lhset.add("KK");

lhset.add("FGH");

System.out.println(lhset);

// LinkedHashSet of Integer Type

LinkedHashSet<Integer> lhset2 = new LinkedHashSet<Integer>();

// Adding elements

lhset2.add(99);

lhset2.add(7);

lhset2.add(0);

lhset2.add(67);

lhset2.add(89);

lhset2.add(66);

System.out.println(lhset2);

}

}

**Output:**

[Z, PQ, N, O, KK, FGH]

[99, 7, 0, 67, 89, 66]

Observe the output: Both types of LinkedHashSet have preserved the insertion order.

**TreeSet Class in Java with example**

TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn’t maintain any order. TreeSet allows null element but like HashSet it doesn’t allow. Like most of the other collection classes this class is also not synchronized, however it can be synchronized explicitly like this: SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

**TreeSet Example:**

In this example we have two TreeSet (TreeSet<String> & TreeSet<Integer>). We have added the values to both of them randomly however the result we got is sorted in ascending order.

import java.util.TreeSet;

public class TreeSetExample {

public static void main(String args[]) {

// TreeSet of String Type

TreeSet<String> tset = new TreeSet<String>();

// Adding elements to TreeSet<String>

tset.add("ABC");

tset.add("String");

tset.add("Test");

tset.add("Pen");

tset.add("Ink");

tset.add("Jack");

//Displaying TreeSet

System.out.println(tset);

// TreeSet of Integer Type

TreeSet<Integer> tset2 = new TreeSet<Integer>();

// Adding elements to TreeSet<Integer>

tset2.add(88);

tset2.add(7);

tset2.add(101);

tset2.add(0);

tset2.add(3);

tset2.add(222);

System.out.println(tset2);

}

}

**Output:** You can see both the TreeSet have been sorted in ascending order implicitly.

[ABC, Ink, Jack, Pen, String, Test]

[0, 3, 7, 88, 101, 222]