Core Java Session Assignments

Topic: Set **Topic Owner: Awadhesh K & Urmil S**

SET :

The SET interface is present in the java.util package and extends the Collection Interface, Hence SET is a collection that contains no duplicate elements. To put it more formal terms , SETS contains no pair of elements E1 and E2 , such that E1.equals(E2), and at the most one null elements.

The Set interface places additional stipulations, beyond those inherited from the Collection interface, on the contracts of all constructors and on the contracts of the add, equals and hashCode methods. Additionally the interface contains a feature that will restrict the insertion of the duplicate elements.

Super Interfaces for the SET interface:

Collection<E>, Iterable<E>

Sub Interface for the SET interface : There are two interfaces which extend the set implementation namely SortedSet and NavigableSet.

The navigable set extends the sorted set interface. Since a set doesn't retain the insertion order, the navigable set interface provides the implementation to navigate through the Set.

Declaration: The Set interface is declared as:

public interface Set extends Collection

Some set implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically NullPointerException or ClassCastException.

Creating Set Objects:

```
Set<Obj> set = new HashSet<Obj> ();
```

Obj is the type of the object to be stored in Set

Classes which implement the Set interface in Java Collections

1. HashSet: Hashset class which is implemented in the collection framework is an inherent implementation of the hash table datastructure. The objects that we insert into the hashset does not guarantee to be inserted in the same order. The objects are inserted based on their hashcode. This class also allows the insertion of NULL elements.

2. EnumSet: EnumSet class which is implemented in the collections framework is one of the specialized implementation of Set interface for use with the enumeration type. It is a high-performance set implementation, much faster than HashSet. All of the elements in an enum set must come from a single enumeration type that is specified when the set is created either explicitly or implicitly.

3. LinkedHashSet: LinkedHashSet class which is implemented in the collections framework is an ordered version of HashSet that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class is used. When iterating through a HashSet the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted.

4. TreeSet: TreeSet class which is implemented in the collections framework an implementation of the SortedSet Interface and SortedSet extends Set Interface. It behaves like simple set with the exception that it stores elements in sorted format. TreeSet uses tree data structure for storage. Objects are stored in sorted, ascending order. But we can iterate in descending order using method TreeSet.descendingIterator().

Example:

```
Import java.util.*;
class Main
{
public static void main(String args[])
      {
        Set<String> hashset = new  HashSet<String>();
  Set<String> linkedhashset = new LinkedHashSet<String>();
         // Adding elements into the HashSet
        // using add()
        hashset.add("India");
        hashset.add("Australia");
        hashset.add("South Africa");
       // Adding the duplicate element
        hashset.add("India");
       // Displaying the HashSet
        System.out.println("implementation using HashSet"+hashset);

linkedhashset.add("India");
        linkedhashset.add("Australia");
        linkedhashset.add("South Africa");
       // Adding the duplicate element
        linkedhashset.add("India");
       // Displaying the HashSet
     System.out.println("Implementation using LinkedHashSet"+linkedhashset);

       // Iterating over hash set items
        System.out.println("Iterating over set:");
```

```
        Iterator<String> i = h.iterator();
        while (i.hasNext())
            System.out.println(i.next());
      }
}
  //Similarly we could do for TreeSet
```

OUTPUT:

```
      Implementation using HashSet [South Africa, Australia, India]
Implementation using LinkedHashSet [South Africa, Australia, India]
      Iterating over set:
      South Africa
Australia
      India
```

All the functions available with Set Interface:

1. **addAll():** We use this function to take union of two set. It will take argument one set which one you want to union with the calling set and returns a s with the calling set and returns a set.
2. **retainAll():** It is similar to addAll() and is used to take the intersection b/w to set.
3. **removeAll():** It is also similar to the addAll() and is used for taking difference b/w two set.
4. **add():** used to add an element into the set.
5. **remove():** is used to remove an element from the set.
6. **contains(Element):** Is used to check whther the Element exist in the set or not.

Example:

```
/* package codechef; // don't place package name! */
import java.util.*;
import java.lang.*;
import java.io.*;
/* Name of the class has to be "Main" only if the class is public. */
class Codechef
{
	public static void main (String[] args) throws java.lang.Exception
	{
//Adding the element into the set using add() method.
                        Set set = new  HashSet<Integer>();
                        set.add(1);
                        set.add(2);
                        set.add(3);
                        set.add(4);
                        set.add(5);
                        //printing the set.
                        System.out.println(set);
                        //checking whether 3 and then 6 exist in the set or not.
                        System.out.println("3 exist in the set: "+set.contains(3));
                        System.out.println("6 exist in the set: "+set.contains(6));
                        //Removing 3 from the set.
                        set.remove(3);
                        System.out.println("after removing 3 from original set"+set);
                        //Take Union
                        //Create another set
                        Set set2 = new LinkedHashSet<Integer>();
```

```
                set2.add(1);
                set2.add(7);
                set2.add(8);
                set2.add(9);
                set2.add(10);
                Set union = new HashSet<Integer>(set);
                union.addAll(set2);
                System.out.println("Union of set1"+set+", set2 "+set2+" is: "+union);
                //Taking Intersection
                Set intersection = new HashSet<Integer>(set);
                intersection.retainAll(set2);
                System.out.println("intersection of set1"+set+", set2 "+set2+" is: "+intersection);
               //Taking Difference Operation
                Set diff = new HashSet<Integer>(set);
                diff.removeAll(set2);
                System.out.println("Difference of set2"+set2+", from set1 "+set+" is: "+diff);
                //We could use enhanced for loop for iterating


                for(Object value : set)
                   System.out.println(value);

}
}
```

Output:

```
[1, 2, 3, 4, 5]
        3 exist in the set: true
        6 exist in the set: false
        after removing 3 from original set[1, 2, 4, 5]
```

Union of set1[1, 2, 4, 5], set2 [1, 7, 8, 9, 10] is: [1, 2, 4, 5, 7, 8, 9, 10]

intersection of set1[1, 2, 4, 5], set2 [1, 7, 8, 9, 10] is: [1]

Difference of set2[1, 7, 8, 9, 10], from set1 [1, 2, 4, 5] is: [2, 4, 5]

1

2

4

5