Set in Java

* Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
* Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
* Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

|  |
| --- |
| // Java code for adding elements in Set  import java.util.\*;  public class Set\_example  {      public static void main(String[] args)      {          // Set deonstration using HashSet          Set<String> hash\_Set = new HashSet<String>();          hash\_Set.add("Geeks");          hash\_Set.add("For");          hash\_Set.add("Geeks");          hash\_Set.add("Example");          hash\_Set.add("Set");          System.out.print("Set output without the duplicates");            System.out.println(hash\_Set);            // Set deonstration using TreeSet          System.out.print("Sorted Set after passing into TreeSet");          Set<String> tree\_Set = new TreeSet<String>(hash\_Set);          System.out.println(tree\_Set);      }  } |

(Please note that we have entered a duplicate entity but it is not displayed in the output. Also, we can directly sort the entries by passing the unordered Set in as the parameter of TreeSet).

**Output:**

Set output without the duplicates[Geeks, Example, For, Set]

Sorted Set after passing into TreeSet[Example, For, Geeks, Set]

**Note:** As we can see the duplicate entry “Geeks” is ignored in the final output, Set interface doesn’t allow duplicate entries.

Now we will see some of the basic operations on the Set i.e. Union, Intersection and Difference.

Let’s take an example of two integer Sets:

* [1, 3, 2, 4, 8, 9, 0]
* [1, 3, 7, 5, 4, 0, 7, 5]

**Union**  
In this, we could simply add one Set with other. Since the Set will itself not allow any duplicate entries, we need not take care of the common values.

**Expected Output:**

Union : [0, 1, 2, 3, 4, 5, 7, 8, 9]

**Intersection**  
We just need to retain the common values from both Sets.

**Expected Output:**

Intersection : [0, 1, 3, 4]

**Difference**  
We just need to remove all the values of one Set from the other.  
 **Expected Output:**

Difference : [2, 8, 9]

|  |
| --- |
| // Java code for demonstrating union, intersection and difference  // on Set  import java.util.\*;  public class Set\_example  {      public static void main(String args[])      {          Set<Integer> a = new HashSet<Integer>();          a.addAll(Arrays.asList(new Integer[] {1, 3, 2, 4, 8, 9, 0}));          Set<Integer> b = new HashSet<Integer>();          b.addAll(Arrays.asList(new Integer[] {1, 3, 7, 5, 4, 0, 7, 5}));            // To find union          Set<Integer> union = new HashSet<Integer>(a);          union.addAll(b);          System.out.print("Union of the two Set");          System.out.println(union);            // To find intersection          Set<Integer> intersection = new HashSet<Integer>(a);          intersection.retainAll(b);          System.out.print("Intersection of the two Set");          System.out.println(intersection);            // To find the symmetric difference          Set<Integer> difference = new HashSet<Integer>(a);          difference.removeAll(b);          System.out.print("Difference of the two Set");          System.out.println(difference);      }  } |

**Output:**

Union of the two Set[0, 1, 2, 3, 4, 5, 7, 8, 9]

Intersection of the two Set[0, 1, 3, 4]

Difference of the two Set[2, 8, 9]

Runtime:

Add Remove Contains Next Size DS

|  |  |
| --- | --- |
|  | HashSet | O(1) | O(1) | O(1) | O(h/n) | O(1) | Hash Table |
|  | LinkedHashSet | O(1) | O(1) | O(1) | O(1) | O(1) | Hash Table + Linked List |
|  | EnumSet | O(1) | O(1) | O(1) | O(1) | O(1) | Bit Vector |
|  | TreeSet | O(log n) | O(log n) | O(log n) | O(log n) | O(1) | Red-black tree |
|  | CopyOnWriteArraySet | O(n) | O(n) | O(n) | O(1) | O(1) | Array |
|  | ConcurrentSkipListSet | O(log n) | O(log n) | O(log n) | O(1) | O(n) | Skip List |