**Set in Java**

* The set interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package
* extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/)
* an unordered collection of objects
* duplicate values cannot be stored
* an interface that implements the mathematical set.
* interface contains the methods inherited from the Collection interface
* adds a feature that restricts the insertion of the duplicate elements.

**Creating Set Objects**

Since Set is an [interface](https://www.geeksforgeeks.org/interfaces-in-java/), We always need a class that extends this list in order to create an object.

In order to use functionalities of the Set interface, we can use these classes:

* [HashSet](https://www.programiz.com/java-programming/hashset)
* [LinkedHashSet](https://www.programiz.com/java-programming/linkedhashset)
* [EnumSet](https://www.programiz.com/java-programming/enumset)
* [TreeSet](https://www.programiz.com/java-programming/treeset)

|  |
| --- |
| Set<Obj> set = new HashSet<Obj> (); |

**Methods:**

* + [add(element)](https://www.geeksforgeeks.org/set-add-method-in-java-with-examples/)
  + [addAll(collection)](https://www.geeksforgeeks.org/set-addall-method-in-java-with-examples/)
  + [clear()](https://www.geeksforgeeks.org/set-clear-method-in-java-with-examples/)
  + [contains(element)](https://www.geeksforgeeks.org/set-contains-method-in-java-with-examples/)
  + [containsAll(collection)](https://www.geeksforgeeks.org/set-containsall-method-in-java-with-examples/)
  + isEmpty()
  + [iterator()](https://www.geeksforgeeks.org/set-iterator-method-in-java-with-examples/)
  + [remove(element)](https://www.geeksforgeeks.org/set-remove-method-in-java-with-examples/)
  + [removeAll(collection)](https://www.geeksforgeeks.org/set-removeall-method-in-java-with-examples/)
  + [size()](https://www.geeksforgeeks.org/set-size-method-in-java-with-example/)

**Example:**

Set<String> hash\_Set = **new** HashSet<String>();

        // Adding elements to the Set

        hash\_Set.add("Geeks");

        hash\_Set.add("For");

        hash\_Set.add("Geeks");

        hash\_Set.add("Example");

        hash\_Set.add("Set");

        // Printing elements of HashSet object

        System.out.println(hash\_Set);

**Output**

[Set, Example, Geeks, For]

**Operations on the Sets**

Let set1 = [1, 3, 2, 4, 8, 9, 0] and set2 = [1, 3, 7, 5, 4, 0, 7, 5].

* Intersection = [0, 1, 3, 4]
* Union = [0, 1, 2, 3, 4, 5, 7, 8, 9]
* Difference = [2, 8, 9]

**Example:**

  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); |

**Performing Various Operations on SortedSet**

* **Adding Elements**

Set<String> hs = **new** HashSet<String>();

        // Adding elements to above object

        // using add() method

        hs.add("B");

        hs.add("B");

        hs.add("C");

        hs.add("A");

        // Printing the elements inside the Set object

        System.out.println(hs);

* **Accessing the Elements**

String check = "D";

        // Check if the above string exists in

        // the SortedSet or not

        // using contains() method

        System.out.println("Contains " + check + " "+ hs.contains(check));

* **Removing the Values**

  hs.remove("B");

        // Printing Set elements after removing an element

        // and printing updated Set elements

        System.out.println("After removing element " + hs);

* **Iterating through the Set**

**for** (String value : hs)

            // Printing all the values inside the object

            System.out.print(value + ", ");

## ***Java HashSet***

In Java, HashSet is commonly used if we have to access elements randomly. It is because elements in a hash table are **accessed using hash codes**. The hashcode of an element is a unique identity that helps to identify the element in a hash table. HashSet cannot contain duplicate elements. Hence, each hash set element has a unique hashcode.

A HashSet is a collection of items where every item is unique, and it is found in the java.util package:

|  |
| --- |
| import java.util.HashSet; // Import the HashSet class  HashSet<String> cars = new HashSet<String>(); |

## **Add Items**

cars.add("Ford");

cars.add("BMW");

cars.add("Mazda");

System.out.println(cars);

## **Check If an Item Exists**

cars.contains("Mazda");

## **Remove an Item**

cars.remove("Volvo");

* To remove all items, cars.clear();

## **HashSet Size**

cars.size();

## **Loop Through a HashSet**

for (String i : cars) {

System.out.println(i);

}

**Example Program:**

|  |
| --- |
| **import** java.util.\*;    **class** HashSetSample  {    **public** **static** **void** main(String[] args)  {  Set<String> h = **new** HashSet<String>();    h.add("India");  h.add("Australia");  h.add("South Africa");    // Adding the duplicate element  h.add("India");    // Displaying the HashSet  System.out.println(h);    h.remove("Australia");  System.out.println("Set after removing " + "Australia:" + h);    System.out.println("Iterating over set:");    Iterator<String> i = h.iterator();    **while** (i.hasNext())    System.out.println(i.next());  }  } |

***LinkedHashSet***

\*maintains a [doubly-linked List](https://www.geeksforgeeks.org/doubly-linked-list/)

\*when the iteration order is needed

\* 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.

|  |
| --- |
| **import** java.util.\*;  **class** LinkedHashsetExample {    **public** **static** **void** main(String[] args)  {  Set<String> lh = **new** LinkedHashSet<String>();    // Adding elements into the LinkedHashSet using add()  lh.add("India");  lh.add("Australia");  lh.add("South Africa");    // Adding the duplicate element  lh.add("India");    // Displaying the LinkedHashSet  System.out.println(lh);    // Removing items from LinkedHashSet  // using remove()  lh.remove("Australia");  System.out.println("Set after removing "+ "Australia:" + lh);    // Iterating over linked hash set items  System.out.println("Iterating over set:");  Iterator<String> i = lh.iterator();  **while** (i.hasNext())  System.out.println(i.next());  }  } |

***TreeSet***

* it stores elements in a sorted format.
* TreeSet uses a tree data structure for storage.
* Objects are stored in sorted, ascending order.
* TreeSet.descendingIterator() - iterate in descending order

|  |
| --- |
| **import** java.util.\*;  **class** TreeSetExample  {  **public** **static** **void** main(String[] args)  {  Set<String> ts = **new** TreeSet<String>();    ts.add("India");  ts.add("Australia");  ts.add("South Africa");    // Adding the duplicate element  ts.add("India");    // Displaying the TreeSet  System.out.println(ts);    // Removing items from TreeSet  ts.remove("Australia");  System.out.println("Set after removing "+ "Australia:" + ts);    // Iterating over Tree set items  System.out.println("Iterating over set:");  Iterator<String> i = ts.iterator();  **while** (i.hasNext())  System.out.println(i.next());  }  } |

**Difference between List,** **Set**,**and Map in Java**

| [List](https://www.geeksforgeeks.org/list-interface-java-examples/) | [Set](https://www.geeksforgeeks.org/set-in-java/) | [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) |
| --- | --- | --- |
| The list interface allows duplicate elements | Set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | Set do not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in set almost only one null value. | The map allows a single null key at most and any number of null values. |
| List implementation classes are [Array List](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/). | Set implementation classes are [HashSet](https://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-in-java-with-examples/), and [TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/). | Map implementation classes are [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/), [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/), [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/), and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/). |
| The list provides get() method to get the element at a specified index. | Set does not provide get method to get the elements at a specified index | The map does not  provide get method to get the elements at a specified index |
| If you need to access the elements frequently by using the index then we can use the list | If you want to create a collection of unique elements then we can use set | If you want to store the data in the form of key/value pair then we can use the map. |
| To traverse the list elements by using Listlterator. | Iterator can be used traverse the set elements | Through keyset, value, and entry set. |