**Collection:**

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).

What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The java.util package contains all the [classes](https://www.javatpoint.com/object-and-class-in-java) and [interfaces](https://www.javatpoint.com/interface-in-java) for the Collection framework.

Collections have 2 Categories: is also called collections interfaces

1. List
2. Set
3. Map

List Interface

**List** in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList. The ArrayList and LinkedList are widely used in Java programming

.

**Operations in a Java List Interface**

Since List is an interface, it can be used only with a class that implements this interface. Now, let’s see how to perform a few frequently used operations on the List.

* **Operation 1:**Adding elements to List class using add() method
* **Operation 2:**Updating elements in List class using set() method
* **Operation 3:** Searching for elements using indexOf(), lastIndexOf methods
* **Operation 4:**Removing elements using remove() method
* **Operation 5:**Accessing Elements in List class using get() method
* **Operation 6:**Checking if an element is present in the List class using contains() method

List interface is implemented by the classes ArrayList, LinkedList.

Above mentioned operations are achieved through ArrayList and LinkedList

**Data Structure of the List:**

1. ArrayList <data-type> list1= new ArrayList<dtata-type>();
2. List <data-type> list2 = new LinkedList();

ArrayList:

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

When To Use

Use an ArrayList for storing and accessing data, and LinkedList to manipulate data.

How the ArrayList works

The ArrayList class has a regular array inside it. When an element is added, it is placed into the array. If the array is not big enough, a new, larger array is created to replace the old one and the old one is removed.

**Methods of the List Interface**

Since the main concept behind the different types of lists is the same, the list interface contains the following methods:

| **Method** | **Description** |
| --- | --- |
| [**add(int index, element)**](https://www.geeksforgeeks.org/list-addint-index-e-element-method-in-java/) | This method is used with Java List Interface to add an element at a particular index in the list. When a single parameter is passed, it simply adds the element at the end of the list. |
| [**addAll(int index, Collection collection)**](https://www.geeksforgeeks.org/list-addall-method-in-java-with-examples/) | This method is used with List Interface in Java to add all the elements in the given collection to the list. When a single parameter is passed, it adds all the elements of the given collection at the end of the list. |
| [**size()**](https://www.geeksforgeeks.org/list-size-method-in-java-with-examples/) | This method is used with Java List Interface to return the size of the list. |
| [**clear()**](https://www.geeksforgeeks.org/list-clear-method-in-java-with-examples/) | This method is used to remove all the elements in the list. However, the reference of the list created is still stored. |
| [**remove(int index)**](https://www.geeksforgeeks.org/list-removeint-index-method-in-java-with-examples/) | This method removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1. |
| [**remove(element)**](https://www.geeksforgeeks.org/list-removeobject-obj-method-in-java-with-examples/) | This method is used with Java List Interface to remove the first occurrence of the given element in the list. |
| [**get(int index)**](https://www.geeksforgeeks.org/list-get-method-in-java-with-examples/) | This method returns elements at the specified index. |
| [**set(int index, element)**](https://www.geeksforgeeks.org/arraylist-set-method-in-java-with-examples/) | This method replaces elements at a given index with the new element. This function returns the element which was just replaced by a new element. |
| [**indexOf(element)**](https://www.geeksforgeeks.org/list-indexof-method-in-java-with-examples/) | This method returns the first occurrence of the given element or *-1* if the element is not present in the list. |
| [**lastIndexOf(element)**](https://www.geeksforgeeks.org/list-lastindexof-method-in-java-with-examples/) | This method returns the last occurrence of the given element or *-1* if the element is not present in the list. |
| [**equals(element)**](https://www.geeksforgeeks.org/list-equals-method-in-java-with-examples/) | This method is used with Java List Interface to compare the equality of the given element with the elements of the list. |
| [**hashCode()**](https://www.geeksforgeeks.org/list-hashcode-method-in-java-with-examples/) | This method is used with List Interface in Java to return the hashcode value of the given list. |
| [**isEmpty()**](https://www.geeksforgeeks.org/list-isempty-method-in-java-with-examples/) | This method is used with Java List Interface to check if the list is empty or not. It returns true if the list is empty, else false. |
| [**contains(element)**](https://www.geeksforgeeks.org/list-contains-method-in-java-with-examples/) | This method is used with List Interface in Java to check if the list contains the given element or not. It returns true if the list contains the element. |
| [**containsAll(Collection collection)**](https://www.geeksforgeeks.org/list-containsall-method-in-java-with-examples/) | This method is used with Java List Interface to check if the list contains all the collection of elements. |

**Linked list:**

Linked List is a part of the [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) present in [java.util package](https://www.geeksforgeeks.org/java-util-package-java/). This class is an implementation of the [LinkedList data structure](https://www.geeksforgeeks.org/data-structures/linked-list/) which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.

Since a LinkedList acts as a dynamic array and we do not have to specify the size while creating it, the size of the list automatically increases when we dynamically add and remove items. And also, the elements are not stored in a continuous fashion. Therefore, there is no need to increase the size. Internally, the LinkedList is implemented using the [doubly linked list data structure](https://www.geeksforgeeks.org/doubly-linked-list/).

**Methods for Java LinkedList**

| **Method** | **Description** |
| --- | --- |
| [add(int index, E element)](https://www.geeksforgeeks.org/java-util-linkedlist-add-method-in-java/) | This method Inserts the specified element at the specified position in this list. |
| [add(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-add-method-in-java/) | This method Appends the specified element to the end of this list. |
| [addAll(int index, Collection<E> c)](https://www.geeksforgeeks.org/java-util-linkedlist-addall-method-in-java/) | This method Inserts all of the elements in the specified collection into this list, starting at the specified position. |
| [addAll(Collection<E> c)](https://www.geeksforgeeks.org/java-util-linkedlist-addall-method-in-java/) | This method Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection’s iterator. |
| [addFirst(E e)](https://www.geeksforgeeks.org/linkedlist-addfirst-method-in-java/) | This method Inserts the specified element at the beginning of this list. |
| [addLast(E e)](https://www.geeksforgeeks.org/linkedlist-addlast-method-in-java/) | This method Appends the specified element to the end of this list. |
| [clear()](https://www.geeksforgeeks.org/linkedlist-clear-method-in-java/) | This method removes all of the elements from this list. |
| [clone()](https://www.geeksforgeeks.org/linkedlist-clone-method-in-java/) | This method returns a shallow copy of this LinkedList. |
| [contains(Object o)](https://www.geeksforgeeks.org/linkedlist-contains-method-in-java/) | This method returns true if this list contains the specified element. |
| [descendingIterator()](https://www.geeksforgeeks.org/linkedlist-descendingiterator-method-in-java-with-examples/) | This method returns an iterator over the elements in this deque in reverse sequential order. |
| [element()](https://www.geeksforgeeks.org/linkedlist-element-method-in-java-with-examples/) | This method retrieves but does not remove, the head (first element) of this list. |
| [get(int index)](https://www.geeksforgeeks.org/linkedlist-get-method-in-java/) | This method returns the element at the specified position in this list. |
| [getFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-get-getfirst-getlast-java/) | This method returns the first element in this list. |
| [getLast()](https://www.geeksforgeeks.org/linkedlist-getlast-method-in-java/) | This method returns the last element in this list. |
| [indexOf(Object o)](https://www.geeksforgeeks.org/linkedlist-indexof-method-in-java/) | This method returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| [lastIndexOf(Object o)](https://www.geeksforgeeks.org/linkedlist-lastindexof-method-in-java/) | This method returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| [listIterator(int index)](https://www.geeksforgeeks.org/linkedlist-listiterator-method-in-java/) | This method returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. |
| [offer(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | This method Adds the specified element as the tail (last element) of this list. |
| [offerFirst(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | This method Inserts the specified element at the front of this list. |
| [offerLast(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | This method Inserts the specified element at the end of this list. |
| [peek()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | This method retrieves but does not remove, the head (first element) of this list. |
| [peekFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty. |
| [peekLast()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty. |
| [poll()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-examples-java/) | This method retrieves and removes the head (first element) of this list. |
| [pollFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-examples-java/) | This method retrieves and removes the first element of this list, or returns null if this list is empty. |
| [pollLast()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-examples-java/) | This method retrieves and removes the last element of this list, or returns null if this list is empty. |
| [pop()](https://www.geeksforgeeks.org/linkedlist-pop-method-in-java/) | This method Pops an element from the stack represented by this list. |
| [push(E e)](https://www.geeksforgeeks.org/linkedlist-push-method-in-java/) | This method pushes an element onto the stack represented by this list. |
| [remove()](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | This method retrieves and removes the head (first element) of this list. |
| [remove(int index)](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | This method removes the element at the specified position in this list. |
| [remove(Object o)](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | This method removes the first occurrence of the specified element from this list if it is present. |
| [removeFirst()](https://www.geeksforgeeks.org/linkedlist-removefirst-method-in-java/) | This method removes and returns the first element from this list. |
| [removeFirstOccurrence(Object o)](https://www.geeksforgeeks.org/linkedlist-removefirstoccurrence-method-in-java/) | This method removes the first occurrence of the specified element in this list (when traversing the list from head to tail). |
| [removeLast()](https://www.geeksforgeeks.org/linkedlist-removelast-method-in-java/) | This method removes and returns the last element from this list. |
| [removeLastOccurrence(Object o)](https://www.geeksforgeeks.org/linkedlist-removelastoccurrence-method-in-java-with-example/) | This method removes the last occurrence of the specified element in this list (when traversing the list from head to tail). |
| [set(int index, E element)](https://www.geeksforgeeks.org/linkedlist-set-method-in-java/) | This method replaces the element at the specified position in this list with the specified element. |
| [size()](https://www.geeksforgeeks.org/linkedlist-size-method-in-java/) | This method returns the number of elements in this list. |
| [spliterator()](https://www.geeksforgeeks.org/linkedlist-spliterator-method-in-java/) | This method creates a late-binding and fail-fast Spliterator over the elements in this list. |
| [toArray()](https://www.geeksforgeeks.org/linkedlist-toarray-method-in-java-with-example/) | This method returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| [toArray(T[] a)](https://www.geeksforgeeks.org/linkedlist-toarray-method-in-java-with-example/) | This method returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. |
| [toString()](https://www.geeksforgeeks.org/object-tostring-method-in-java/) | This method returns a string containing all of the elements in this list in proper sequence (from first to the last element), each element is separated by commas and the String is enclosed in square brackets. |

Map:

In Java, Map Interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value. Java Map interface is not a subtype of the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/). Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.

Geeks, the brainstormer should have been why and when to use Maps.

Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some common scenarios are as follows:

* A map of error codes and their descriptions.
* A map of zip codes and cities.
* A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
* A map of classes and students. Each class (key) is associated with a list of students (value).
* Map hm = new HashMap();   
  // Obj is the type of the object to be stored in Map

Methods in Java Map Interface

| Method | Action Performed |
| --- | --- |
| [clear()](https://www.geeksforgeeks.org/map-clear-method-in-java-with-example/) | This method is used in Java Map Interface to clear and remove all of the elements or mappings from a specified Map collection. |
| [containsKey(Object)](https://www.geeksforgeeks.org/map-containskey-method-in-java-with-examples/) | This method is used in Map Interface in Java to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map. |
| [containsValue(Object)](https://www.geeksforgeeks.org/map-containsvalue-method-in-java-with-examples/) | This method is used in Map Interface to check whether a particular value is being mapped by a single or more than one key in the Map. It takes the value as a parameter and returns True if that value is mapped by any of the keys in the map. |
| [entrySet()](https://www.geeksforgeeks.org/map-entryset-method-in-java-with-examples/) | This method is used in Map Interface in Java to create a set out of the same elements contained in the map. It basically returns a set view of the map or we can create a new set and store the map elements into them. |
| [equals(Object)](https://www.geeksforgeeks.org/map-equals-method-in-java-with-examples/) | This method is used in Java Map Interface to check for equality between two maps. It verifies whether the elements of one map passed as a parameter is equal to the elements of this map or not. |
| [get(Object)](https://www.geeksforgeeks.org/map-get-method-in-java-with-examples/) | This method is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key. |
| [hashCode()](https://www.geeksforgeeks.org/map-hashcode-method-in-java-with-examples/) | This method is used in Map Interface to generate a hashCode for the given map containing keys and values. |
| [isEmpty()](https://www.geeksforgeeks.org/map-isempty-method-in-java-with-examples/) | This method is used to check if a map is having any entry for key and value pairs. If no mapping exists, then this returns true. |
| [keySet()](https://www.geeksforgeeks.org/map-keyset-method-in-java-with-examples/) | This method is used in Map Interface to return a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. |
| [put(Object, Object)](https://www.geeksforgeeks.org/map-put-method-in-java-with-examples/) | This method is used in Java Map Interface to associate the specified value with the specified key in this map. |
| [putAll(Map)](https://www.geeksforgeeks.org/map-putall-method-in-java-with-examples/) | This method is used in Map Interface in Java to copy all of the mappings from the specified map to this map. |
| [remove(Object)](https://www.geeksforgeeks.org/map-remove-method-in-java-with-examples/) | This method is used in Map Interface to remove the mapping for a key from this map if it is present in the map. |
| [size()](https://www.geeksforgeeks.org/hashmap-size-method-in-java/) | This method is used to return the number of key/value pairs available in the map. |
| [values()](https://www.geeksforgeeks.org/hashmap-values-method-in-java/) | This method is used in Java Map Interface to create a collection out of the values of the map. It basically returns a Collection view of the values in the HashMap. |
| [getOrDefault(Object key, V defaultValue)](https://www.geeksforgeeks.org/hashmap-getordefaultkey-defaultvalue-method-in-java-with-examples/) | Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key. |
| [merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)](https://www.geeksforgeeks.org/hashmap-mergekey-value-bifunction-method-in-java-with-examples/) | If the specified key is not already associated with a value or is associated with null, associate it with the given non-null value. |
| [putIfAbsent(K key, V value)](https://www.geeksforgeeks.org/hashmap-putifabsentkey-value-method-in-java-with-examples/) | If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current associate value. |

|  |  |
| --- | --- |
| **Class** | **Description** |
| [HashMap](https://www.javatpoint.com/java-hashmap) | HashMap is the implementation of Map, but it doesn't maintain any order. |
| [LinkedHashMap](https://www.javatpoint.com/java-linkedhashmap) | LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order. |
| [TreeMap](https://www.javatpoint.com/java-treemap) | TreeMap is the implementation of Map and SortedMap. It maintains ascending order. |

Java HashMap

In the [ArrayList](https://www.w3schools.com/java/java_arraylist.asp) chapter, you learned that Arrays store items as an ordered collection, and you have to access them with an index number (int type). A HashMap however, store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values:

Create a HashMap object called capitalCities that will store String keys and String values:

import java.util.HashMap; // import the HashMap class

HashMap<String, String> capitalCities = new HashMap<String, String>();

Linked hashmap

The **LinkedHashMap** **Class** is just like [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion, which the LinkedHashMap provides where the elements can be accessed in their insertion order.

**Features of a LinkedHashMap:**

* A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.
* It contains only unique elements.
* It may have one null key and multiple null values.
* It is non-synchronized.
* It is the same as HashMap with an additional feature that it maintains insertion order. For example, when we run the code with a HashMap, we get a different order of elements.

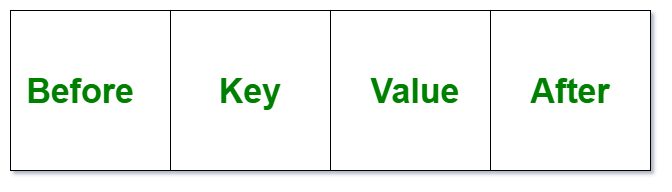
**Declaration of LinkedHashMap:**

public class LinkedHashMap<K,​V> extends HashMap<K,​V> implements Map<K,​V>

Here, **K** is the key Object type and **V** is the value Object type

* **K:** The type of the keys in the map.
* **V:** The type of values mapped in the map.
* A **LinkedHashMap** is an extension of the **HashMap** class and it implements the **Map** interface. Therefore, the class is declared as:
* public class LinkedHashMap   
  extends HashMap   
  implements Map

n this class, the data is stored in the form of nodes. The implementation of the LinkedHashMap is very similar to a [doubly-linked list](https://www.geeksforgeeks.org/doubly-linked-list/). Therefore, each node of the LinkedHashMap is represented as:



* [**Hash:**](https://www.geeksforgeeks.org/hashing-set-1-introduction/) All the input keys are converted into a hash which is a shorter form of the key so that the search and insertion are faster.
* **Key:** Since this class extends HashMap, the data is stored in the form of a key-value pair. Therefore, this parameter is the key to the data.
* **Value:** For every key, there is a value associated with it. This parameter stores the value of the keys. Due to generics, this value can be of any form.
* **Next:** Since the LinkedHashMap stores the insertion order, this contains the address to the next node of the LinkedHashMap.
* **Previous:** This parameter contains the address to the previous node of the LinkedHashMap.

**1. LinkedHashMap():** This is used to construct a default LinkedHashMap constructor.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>();

**2. LinkedHashMap(int capacity):** It is used to initialize a particular LinkedHashMap with a specified capacity.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity);

**3. LinkedHashMap(Map**<? extends **K**,​? extends **V**> map**):** It is used to initialize a particular LinkedHashMap with the elements of the specified map.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(Map<? extends K,​? extends V> map);

**4. LinkedHashMap(int capacity, float fillRatio):** It is used to initialize both the capacity and fill ratio for a LinkedHashMap. A fillRatio also called as **loadFactor** is a metric that determines when to increase the size of the LinkedHashMap automatically. By default, this value is 0.75 which means that the size of the map is increased when the map is 75% full.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio);

**5. LinkedHashMap(int capacity, float fillRatio, boolean Order):** This constructor is also used to initialize both the capacity and fill ratio for a LinkedHashMap along with whether to follow the insertion order or not.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio, boolean Order);

Here, For the ***Order attribute***, true is passed for the last access order and false is passed for the insertion order.

**Methods of LinkedHashMap**

| **Method** | **Description** |
| --- | --- |
| containsValue​(Object value) | Returns true if this map maps one or more keys to the specified value. |
| entrySet() | Returns a Set view of the mappings contained in this map. |
| [get​(Object key)](https://www.geeksforgeeks.org/linkedhashmap-get-method-in-java/) | Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| keySet() | Returns a Set view of the keys contained in this map. |
| [removeEldestEntry​(Map.Entry<K,​V> eldest)](https://www.geeksforgeeks.org/linkedhashmap-removeeldestentry-method-in-java/) | Returns true if this map should remove its eldest entry. |
| values() | Returns a Collection view of the values contained in this map. |

Set: ignore the duplicate value

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

Hashset: ¬ It will print random order

LinkedHashset: ¬ Insertion order

Tress set-> Accedning order

Let us discuss methods present in the Set interface provided below in a tabular format below as follows:

| **Method** | **Description** |
| --- | --- |
| [add(element)](https://www.geeksforgeeks.org/set-add-method-in-java-with-examples/) | This method is used to add a specific element to the set. The function adds the element only if the specified element is not already present in the set else the function returns False if the element is already present in the Set. |
| [addAll(collection)](https://www.geeksforgeeks.org/set-addall-method-in-java-with-examples/) | This method is used to append all of the elements from the mentioned collection to the existing set. The elements are added randomly without following any specific order. |
| [clear()](https://www.geeksforgeeks.org/set-clear-method-in-java-with-examples/) | This method is used to remove all the elements from the set but not delete the set. The reference for the set still exists. |
| [contains(element)](https://www.geeksforgeeks.org/set-contains-method-in-java-with-examples/) | This method is used to check whether a specific element is present in the Set or not. |
| [containsAll(collection)](https://www.geeksforgeeks.org/set-containsall-method-in-java-with-examples/) | This method is used to check whether the set contains all the elements present in the given collection or not. This method returns true if the set contains all the elements and returns false if any of the elements are missing. |
| [hashCode()](https://www.geeksforgeeks.org/set-hashcode-method-in-java-with-examples/) | This method is used to get the hashCode value for this instance of the Set. It returns an integer value which is the hashCode value for this instance of the Set. |
| isEmpty() | This method is used to check whether the set is empty or not. |
| [iterator()](https://www.geeksforgeeks.org/set-iterator-method-in-java-with-examples/) | This method is used to return the [iterator](https://www.geeksforgeeks.org/iterators-in-java/) of the set. The elements from the set are returned in a random order. |
| [remove(element)](https://www.geeksforgeeks.org/set-remove-method-in-java-with-examples/) | This method is used to remove the given element from the set. This method returns True if the specified element is present in the Set otherwise it returns False. |
| [removeAll(collection)](https://www.geeksforgeeks.org/set-removeall-method-in-java-with-examples/) | This method is used to remove all the elements from the collection which are present in the set. This method returns true if this set changed as a result of the call. |
| [retainAll(collection)](https://www.geeksforgeeks.org/set-retainall-method-in-java-with-example/) | This method is used to retain all the elements from the set which are mentioned in the given collection. This method returns true if this set changed as a result of the call. |
| [size()](https://www.geeksforgeeks.org/set-size-method-in-java-with-example/) | This method is used to get the size of the set. This returns an integer value which signifies the number of elements. |
| [toArray()](https://www.geeksforgeeks.org/set-toarray-method-in-java-with-example/) | This method is used to form an array of the same elements as that of the Se |

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

**Operations on the Set Interface**

The set interface allows the users to perform the basic mathematical operation on the set. Let’s take two arrays to understand these basic operations. Let set1 = [1, 3, 2, 4, 8, 9, 0] and set2 = [1, 3, 7, 5, 4, 0, 7, 5]. Then the possible operations on the sets are:

**1. Intersection:** This operation returns all the common elements from the given two sets. For the above two sets, the intersection would be: **Linked set**

Intersection = [0, 1, 3, 4]

**2. Union:** This operation adds all the elements in one set with the other. For the above two sets, the union would be:

Union = [0, 1, 2, 3, 4, 5, 7, 8, 9]  - **Tree set**

**3. Difference:** This operation removes all the values present in one set from the other set. For the above two sets, the difference would be:

Difference = [2, 8, 9] - **Hashset**