**Java Collection Framework**

The **Collection in Java** is a framework that provides 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).

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

## Java Collections

1. [ArrayList](https://beginnersbook.com/java-collections-tutorials/" \l "1) 2. [LinkedList](https://beginnersbook.com/java-collections-tutorials/#2) 3. [Vector](https://beginnersbook.com/java-collections-tutorials/#3) 4. [HashSet](https://beginnersbook.com/java-collections-tutorials/#4)  
5. [LinkedHashSet](https://beginnersbook.com/java-collections-tutorials/" \l "5) 6. [TreeSet](https://beginnersbook.com/java-collections-tutorials/" \l "6) 7. [HashMap](https://beginnersbook.com/java-collections-tutorials/#7) 8. [TreeMap](https://beginnersbook.com/java-collections-tutorials/" \l "8)  
9. [LinkedHashMap](https://beginnersbook.com/java-collections-tutorials/" \l "9) 10. [Hashtable](https://beginnersbook.com/java-collections-tutorials/" \l "10)

**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.



[**Linked List**:](https://www.geeksforgeeks.org/linked-list-in-java/) Linked List class is an implementation of the [Linked List 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**.



Java LinkedList class uses a **doubly linked list** to store the elements. It provides a linked-list data structure. It inherits the **AbstractList** class and implements **List** and **Deque** *interfaces*.

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to occur.
* Java LinkedList class can be used as a list, stack or queue.

**Example program on Linked List**

import java.util.\*;

public class LinkedList1{

public static void main(String args[])

{

LinkedList<String> al=new LinkedList<String>( );

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

Iterator<String> itr=al.iterator();

while(itr.hasNext())

System.out.println(itr.next());

//Removing specific element from arraylist

              al.remove("Vijay");

              System.out.println("After invoking remove(object) method: "+al);

         //Removing element on the basis of specific position

              al.remove(0);

              System.out.println("After invoking remove(index) method: "+al);

}

}

**Java Sets**

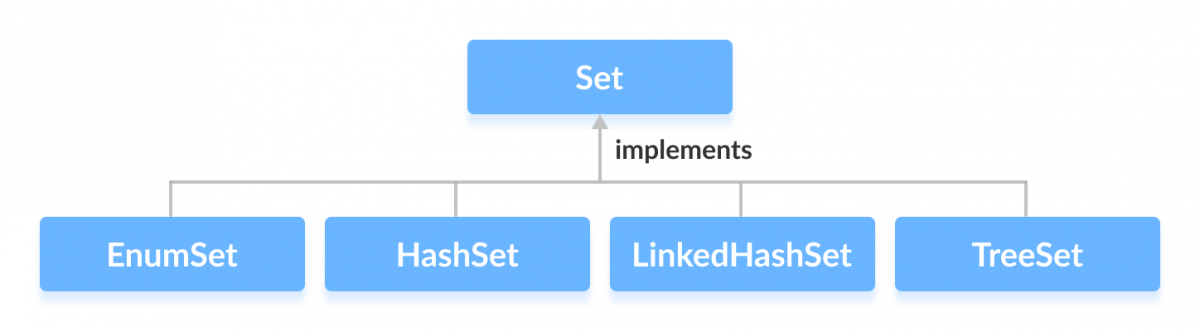
The Set interface of the Java Collections framework provides the features of the mathematical set in Java. It extends the Collection interface.

Unlike the List interface, sets cannot contain duplicate elements.

## Classes that implement Set

Since Set is an interface, we cannot create objects from it.In order to use functionalities of the Set interface, we can use these classes:

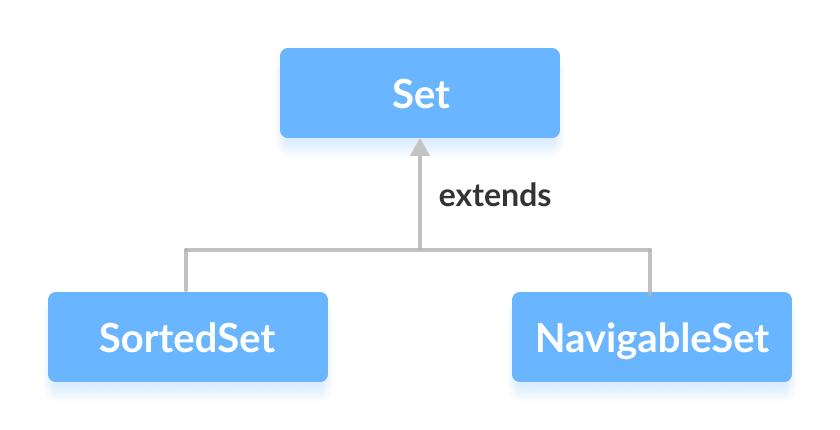
* [HashSet](https://www.programiz.com/java-programming/hashset)
* [LinkedHashSet](https://www.programiz.com/java-programming/linkedhashset)
* [EnumSet](https://www.programiz.com/java-programming/enumset)
* [TreeSet](https://www.programiz.com/java-programming/treeset)



## Interfaces that extend Set

The set interface is also extended by these sub interfaces:

* [SortedSet](https://www.programiz.com/java-programming/sortedset)
* [NavigableSet](https://www.programiz.com/java-programming/navigableset)



**Set Operations**

The Java Set interface allows us to perform basic mathematical set operations like union, intersection, and subset.

Union - to get the union of two sets x and y, we can use x.addAll(y)

Intersection - to get the intersection of two sets x and y, we can use x.retainAll(y)

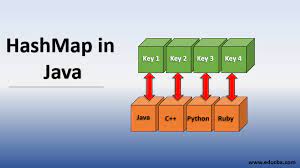
Subset - to check if x is a subset of y, we can use y.containsAll(x)

**HashSet**

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.

**HashMap**

**Hashing :**It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.



**HashMap :** HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of **Key and Value**. HashMap contains an array of the nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value.

There are four fields in HashMap.

int hash

K key

V value

Node next

Hashing

Hashing is a process of converting an object into integer form by using the method hashCode(). Its necessary to write hashCode() method properly for better performance of HashMap

**hashCode( ) :** is used to get the hash Code of an object. hashCode() method of object class returns the memory reference of object in integer form. Definition of hashCode() method is public native hashCode(). It indicates the implementation of hashCode() is native because there is not any direct method in java to fetch the reference of object. It is possible to provide your own implementation of hashCode().

In HashMap, hashCode() is used to calculate the bucket and therefore calculate the index.

**equals( )** : equals method is used to check that 2 objects are equal or not. This method is provided by Object class. You can override this in your class to provide your own implementation.

HashMap uses equals() to compare the key whether are equal or not. If equals() method return true, they are equal otherwise not equal.

**Buckets :**A bucket is one element of HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case link list structure is used to connect the nodes. Buckets are different in capacity. A relation between bucket and capacity is as follows:

capacity = number of buckets \* load factor

A single bucket can have more than one nodes, it depends on hashCode() method. The better your hashCode() method is, the better your buckets will be utilized.

//HashMap Example program

import java.util.Map;

import java.util.HashMap;

class Main {

public static void main(String[] args) {

// Creating a map using the HashMap

Map<String, Integer> numbers = new HashMap<>();

// Insert elements to the map

numbers.put("One", 1);

numbers.put("Two", 2);

System.out.println("Map: " + numbers);

// Access keys of the map

System.out.println("Keys: " + numbers.keySet());

// Access values of the map

System.out.println("Values: " + numbers.values());

// Access entries of the map

System.out.println("Entries: " + numbers.entrySet());

// Remove Elements from the map

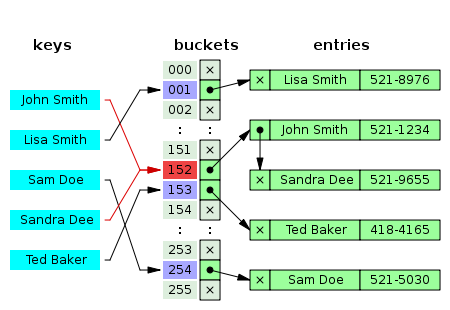
int value = numbers.remove("Two");

System.out.println("Removed Value: " + value);

}

}

**Hash Table:** a hash table is a data structure that implements an associative array abstract data type, a structure that can map keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found.



**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 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 remove(Object o): It removes the specified Object o from the Set.

**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);

}

}

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

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

**Difference between List and Set**

A list can contain duplicate elements whereas Set contains unique elements only.

**Linked HashSet**

The LinkedHashSet 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. When cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

LinkedHashSet<E> hs = new LinkedHashSet<E>();

Example program

import java.util.\*;

public class LinkedHashSetEx

{

public static void main(String[] args)

{

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

// Adding element to LinkedHashSet

linkedset.add("A");

linkedset.add("B");

linkedset.add("C");

linkedset.add("D");

// This will not add new element as A already exists

linkedset.add("A");

linkedset.add("E");

System.out.println("Size of LinkedHashSet = " + linkedset.size());

System.out.println("Original LinkedHashSet:" + linkedset);

System.out.println("Removing D from LinkedHashSet: " + linkedset.remove("D"));

System.out.println("Trying to Remove Z which is not "+ "present: " + linkedset.remove("Z"));

System.out.println("Checking if A is present=" + linkedset.contains("A"));

System.out.println("Updated LinkedHashSet: " + linkedset);

}

}

# Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

However, there are many differences between ArrayList and LinkedList classes that are given below.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| **4) ArrayList is better for storing and accessing data.** | **LinkedList is better for manipulating data.** |

**List Vs Set**

List and Set both are interfaces. They both extends Collection interface.

The differences between List and Set interfaces in java.

1) List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.

Set is an unordered collection, it doesn’t maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).

2) List allows duplicates while Set doesn’t allow duplicate elements.

All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.

3) List implementations: ArrayList, LinkedList etc.

Set implementations: HashSet, LinkedHashSet, TreeSet etc.

4) List allows any number of null values.

Set can have only a single null value at most.

5) ListIterator can be used to traverse a List in both the directions(forward and backward)

However it cannot be used to traverse a Set. We can use Iterator (It works with List too) to traverse a Set.

6) List interface has one legacy class called Vector whereas Set interface does not have any legacy class.

**When to use Set and When to use List?**

The usage is purely depends on the requirement:

If the requirement is to have only unique values then Set is your best bet as any implementation of Set maintains unique values only.

If there is a need to maintain the insertion order irrespective of the duplicity then List is a best option. Both the implementations of List interface – ArrayList and LinkedList sorts the elements in their insertion order.

**List Example**

import java.util.List;

import java.util.ArrayList;

import java.util.LinkedList;

public class ListExample {

public static void main(String[] args) {

List<String> al = new ArrayList<String>();

al.add("Chaitanya");

al.add("Rahul");

al.add("Ajeet");

System.out.println("ArrayList Elements: ");

System.out.print(al);

List<String> ll = new LinkedList<String>();

ll.add("Kevin");

ll.add("Peter");

ll.add("Kate");

System.out.println("\nLinkedList Elements: ");

System.out.print(ll);

}

}

**Output:**

ArrayList Elements:

[Chaitanya, Rahul, Ajeet]

LinkedList Elements:

[Kevin, Peter, Kate]

**Set Example**

import java.util.Set;

import java.util.HashSet;

import java.util.TreeSet;

public class SetExample {

public static void main(String args[]) {

int count[] = {11, 22, 33, 44, 55};

Set<Integer> hset = new HashSet<Integer>();

try{

for(int i = 0; i<4; i++)

hset.add(count[i]);

System.out.println(hset);

TreeSet<Integer> treeset = new TreeSet<Integer>(hset);

System.out.println("The sorted list is:");

System.out.println(treeset);

}

catch(Exception e){

e.printStackTrace();

}

}

}

Output:

[33, 22, 11, 44]

The sorted list is:

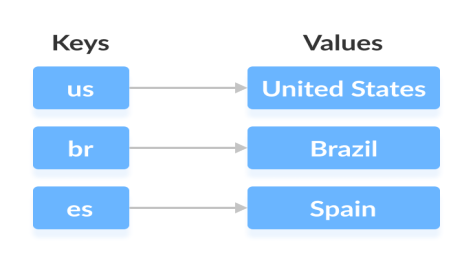
[11, 22, 33, 44]

**Map interface**

The Map interface of the Java collections framework provides the functionality of the map data structure.

**Working of Map**

In Java, elements of Map are stored in key/value pairs. Keys are unique values associated with individual Values. A map cannot contain duplicate keys. And, each key is associated with a single value.



We can access and modify values using the keys associated with them. In the above diagram, we have values: United States, Brazil, and Spain. And we have corresponding keys: us, br, and es.

Now, we can access those values using their corresponding keys.

**Note:** The Map interface maintains 3 different sets:

* The set of keys
* The set of values
* The set of key/value associations (mapping).

Hence we can access keys, values, and associations individually.

**Classes that implement Map**

Since Map is an interface, we cannot create objects from it.

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

* HashMap
* EnumMap
* LinkedHashMap
* WeakHashMap
* TreeMap

These classes are defined in the collections framework and implement the Map interface.

## Interfaces that extend Map



**Characteristics of a Map Interface**

A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value like the HashMap and LinkedHashMap, but some do not like the TreeMap.

The order of a map depends on the specific implementations. For example, TreeMap and LinkedHashMap have predictable order, while HashMap does not.

There are two interfaces for implementing Map in java. They are, Map and SortedMap, and three classes: HashMap, TreeMap and LinkedHashMap.

**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 examples are:

1. A map of error codes and their descriptions.
2. *A map of zip codes and cities.*
3. A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
4. A map of classes and students. Each class (key) is associated with a list of students (value).

**Creating Map Objects**

Since Map is an interface, objects cannot be created of the type map. We always need a class which extends this map in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Map. This type-safe map can be defined as:

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

Map hm = new HashMap ();

**Java Iterator**

An Iterator is an object that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

To use an Iterator, you must import it from the java.util package.

Example program

// Import the ArrayList class and the Iterator class

import java.util.ArrayList;

import java.util.Iterator;

public class IteratorEx {

public static void main(String[] args) {

// Make a collection

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

// Get the iterator

Iterator<String> it = cars.iterator();

// Print the first item

System.out.println(it.next());

while(it.hasNext()) {

System.out.println(it.next());

}

}

}

**List Iterator**

The listIterator() method of java.util.ArrayList class is used to return a list iterator over the elements in this list (in proper sequence). The returned list iterator is fail-fast.

Syntax:

public ListIterator listIterator()

Return Value: This method returns a list iterator over the elements in this list (in proper sequence).

**Methods of ListIterator**

1) void add(E e): Inserts the specified element into the list (optional operation).

2) boolean hasNext(): Returns true if this list iterator has more elements when traversing the list in the forward direction.

3) boolean hasPrevious(): Returns true if this list iterator has more elements when traversing the list in the reverse direction.

4) E next(): Returns the next element in the list and advances the cursor position.

5) int nextIndex(): Returns the index of the element that would be returned by a subsequent call to next().

6) E previous(): Returns the previous element in the list and moves the cursor position backwards.

7) int previousIndex(): Returns the index of the element that would be returned by a subsequent call to previous().

8) void remove(): Removes from the list the last element that was returned by next() or previous() (optional operation).

9) void set(E e): Replaces the last element returned by next() or previous() with the specified element (optional operation).

**List Iterator example program**

import java.util.ArrayList;

import java.util.List;

import java.util.ListIterator;

public class ListIteratorEx {

public static void main(String[] args) {

ListIterator<String> litr = null;

List<String> names = new ArrayList<String>();

names.add("Ram");

names.add("Raja");

names.add("Praveen");

names.add("Tulasi");

names.add("Kiran");

//Obtaining list iterator

litr=names.listIterator();

System.out.println("Traversing the list in forward direction:");

while(litr.hasNext()){

System.out.println(litr.next());

}

System.out.println("\nTraversing the list in backward direction:");

while(litr.hasPrevious()){

System.out.println(litr.previous());

}

} }