

### Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

|  |  |  |
| --- | --- | --- |
| No. | Method | Description |
| 1 | public boolean add(Object element) | is used to insert an element in this collection. |
| 2 | public boolean addAll(Collection c) | is used to insert the specified collection elements in the invoking collection. |
| 3 | public boolean remove(Object element) | is used to delete an element from this collection. |
| 4 | public boolean removeAll(Collection c) | is used to delete all the elements of specified collection from the invoking collection. |
| 5 | public boolean retainAll(Collection c) | is used to delete all the elements of invoking collection except the specified collection. |
| 6 | public int size() | return the total number of elements in the collection. |
| 7 | public void clear() | removes the total no of element from the collection. |
| 8 | public boolean contains(Object element) | is used to search an element. |
| 9 | public boolean containsAll(Collection c) | is used to search the specified collection in this collection. |
| 10 | public Iterator iterator() | returns an iterator. |
| 11 | public Object[] toArray() | converts collection into array. |
| 12 | public boolean isEmpty() | checks if collection is empty. |
| 13 | public boolean equals(Object element) | matches two collection. |
| 14 | public int hashCode() | returns the hashcode number for collection. |

### Iterator interface

|  |
| --- |
| Iterator interface provides the facility of iterating the elements in forward direction only. |

#### **Methods of Iterator interface**

There are only three methods in the Iterator interface. They are:

1. public boolean hasNext() it returns true if iterator has more elements.
2. public object next() it returns the element and moves the cursor pointer to the next element.
3. public void remove() it removes the last elements returned by the iterator. It is rarely used.

# Java ArrayList class

* Java ArrayList class uses a dynamic array for storing the elements.It extends AbstractList class and implements List interface.
* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

### Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

1. ArrayList al=**new** ArrayList();//creating old non-generic arraylist

Let's see the new generic example of creating java collection.

1. ArrayList<String> al=**new** ArrayList<String>();//creating new generic arraylist

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

For more information of java generics,

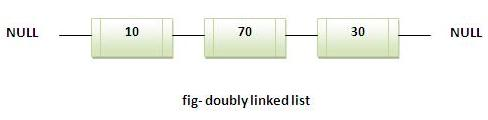
### Two ways to iterate the elements of collection in java

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

# Java LinkedList class

* Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
* 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 be occurred.
* Java LinkedList class can be used as list, stack or queue.



# Difference between ArrayList and LinkedList

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

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

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| 1) ArrayList internally uses dynamic array to store the elements. | LinkedList internally uses doubly linked list to store the elements. |
| 2) Manipulation with ArrayList is slow because it internally uses 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 doubly linked list so no bit shifting is required in memory. |
| 3) 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. |

# Java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

### Commonly used methods of List Interface:

1. public void add(int index,Object element);
2. public boolean addAll(int index,Collection c);
3. public object get(int Index position);
4. public object set(int index,Object element);
5. public object remove(int index);
6. public ListIterator listIterator();
7. public ListIterator listIterator(int i);

## Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

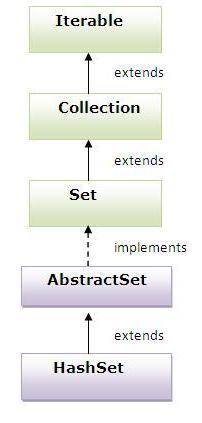
### Commonly used methods of ListIterator Interface:

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

# Java HashSet class

* uses hashtable to store the elements.It extends AbstractSet class and implements Set interface.
* contains unique elements only.

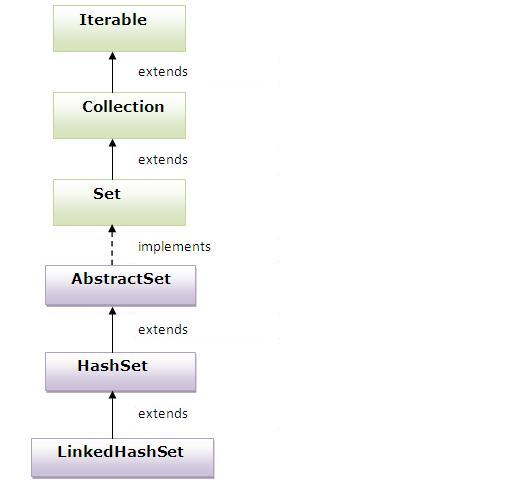
## Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.

# Java LinkedHashSet class:

* contains unique elements only like HashSet. It extends HashSet class and implements Set interface.
* maintains insertion order.

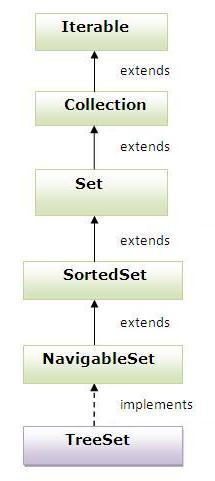
Hierarchy of LinkedHashSet class:



# Java TreeSet class

* contains unique elements only like HashSet. The TreeSet class implements NavigableSet interface that extends the SortedSet interface.
* maintains ascending order.

## Hierarchy of TreeSet class:



# Java Vector class

Vector implements a dynamic array. It is similar to ArrayList, but with two differences:

* Vector is synchronized.
* Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Java Enumeration Interface

The Enumeration interface defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects.

This legacy interface has been superceded by Iterator. Although not deprecated, Enumeration is considered obsolete for new code. However, it is used by several methods defined by the legacy classes such as Vector and Properties, is used by several other API classes, and is currently in widespread use in application code.

The methods declared by Enumeration are summarized in the following table:

|  |  |
| --- | --- |
| SN | Methods with Description |
| 1 | **boolean hasMoreElements( )**  When implemented, it must return true while there are still more elements to extract, and false when all the elements have been enumerated. |
| 2 | **Object nextElement( )**  This returns the next object in the enumeration as a generic Object reference. |

The following table summarizes the principal classes in Java collections framework for quick reference:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Principal collection class** | **Base class** | **Base interfaces** | **Allow duplicate elements?** | **Ordered?** | **Sorted?** | **Thread-safe?** |
| **ArrayList**<E> | *AbstractList<E>* | *List<E>* | **Yes** | **Yes** | **No** | **No** |
| **LinkedList**<E> | *AbstractSequentialList<E>* | *List<E>;*  *Deque<E>* | **Yes** | **Yes** | **No** | **No** |
| **Vector**<E> | *AbstractList<E>* | *List<E>* | **Yes** | **Yes** | **No** | **Yes** |
| **HashSet**<E> | *AbstractSet<E>* | *Set<E>* | **No** | **No** | **No** | **No** |
| **LinkedHashSet**<E> | *HashSet<E>* | *Set<E>* | **No** | **Yes** | **No** | **No** |
| **TreeSet**<E> | *AbstractSet<E>* | *Set<E>;*  *NavigableSet<E>;*  *SortedSet<E>* | **No** | **Yes** | **Yes** | **No** |
| **HashMap**<K, V> | *AbstractMap<K, V>* | *Map<K, V>* | **No** | **No** | **No** | **No** |
| **LinkedHashMap**<K, V> | *HashMap<K, V>* | *Map<K, V>* | **No** | **Yes** | **No** | **No** |
| **Hashtable**<K, V> | *Dictionary<K, V>* | *Map<K, V>* | **No** | **No** | **No** | **Yes** |
| **TreeMap**<K, V> | *AbstractMap<K, V>* | *Map<K, V>;*  *NavigableMap<K, V>;*  *SortedMap<K, V>* | **No** | **Yes** | **Yes** | **No** |

From this table we can conclude the following characteristics of the main collections in Java Collection Frameworks:

* + All lists allow duplicate elements which are ordered by index.
  + All sets and maps do not allow duplicate elements.
  + All list elements are not sorted.
  + Generally, sets and maps do not sort its elements, except TreeSet and TreeMap – which sort elements by natural order or by a comparator.
  + Generally, elements within sets and maps are not ordered, except for:
    - * LinkedHashSet and LinkedHashMap have elements ordered by insertion order.
      * TreeSet and TreeMap have elements ordered by natural order or by a comparator.
  + There are only two collections are thread-safe: Vector and Hastable. The rest is not thread-safe.