# The Collection classes in Java

Java collection framework consists of various classes that are used to store objects. These classes on the top implements the Collection interface. Some of the classes provide full implementations that can be used as it is. Others are abstract classes, which provides skeletal implementations that can be used as a starting point for creating concrete collections.

- **Java Collection Framework Classes**

This table contains abstract and non-abstract classes that implements collection interface.

The standard collection classes are:

| Class | Description |
|---|---|
| AbstractCollection | Implements most of the Collection interface. |
| AbstractList | Extends AbstractCollection and implements most of the List interface. |
| AbstractQueue | Extends AbstractCollection and implements parts of the Queue interface. |
| AbstractSequentialList | Extends AbstractList for use by a collection that uses sequential rather than random access of its elements. |
| LinkedList | Implements a linked list by extending AbstractSequentialList |
| ArrayList | Implements a dynamic array by extending AbstractList |

| | |
|---|---|
| ArrayDeque | Implements a dynamic double-ended queue by extending AbstractCollection and implementing the Deque interface(Added by Java SE 6). |
| AbstractSet | Extends AbstractCollection and implements most of the Set interface. |
| EnumSet | Extends AbstractSet for use with enum elements. |
| HashSet | Extends AbstractSet for use with a hash table. |
| LinkedHashSet | Extends HashSet to allow insertion-order iterations. |
| PriorityQueue | Extends AbstractQueue to support a priority-based queue. |
| TreeSet | Implements a set stored in a tree. Extends AbstractSet. |

**Note:**

To use any Collection class in your program, you need to import java.util package.

Whenever you print any Collection class, it gets printed inside the square brackets [] with its elements.

- **ArrayList class**

This class provides implementation of an array-based data structure that is used to store elements in linear order. This class implements List interface and an abstract AbstractList class. It creates a dynamic array that grows based on the element's strength.

Simple array has fixed size i.e it can store fixed number of elements but sometimes you may not know beforehand about the number of elements that you are going to store in your array. In such situations, We can use an ArrayList, which is an array whose size can increase or decrease dynamically.

1. ArrayList class extends AbstractList class and implements the List interface.

2. ArrayList supports dynamic array that can grow as needed.

3. It can contain Duplicate elements and it also maintains the insertion order.

4. Manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

5. ArrayLists are not synchronized.

6. ArrayList allows random access because it works on the index basis.

- **ArrayList Constructors**

ArrayList class has three constructors that can be used to create Arraylist either from empty or elements of other collection.

ArrayList()  // It creates an empty ArrayList

ArrayList( Collection C ) // It creates an ArrayList that is initialized with elements of the Collection C

ArrayList( int capacity ) // It creates an ArrayList that has the specified initial capacity.

**Methods of ArrayList:**

| Method | Description |
|---|---|
| add(int index, Object element) | This method is used to insert a specific element at a specific position index in a list. |
| add(Object o) | This method is used to append a specific element to the end of a list. |
| addAll(Collection C) | This method is used to append all the elements from a specific collection to the end of the mentioned list, in such an order that the values are returned by the specified collection's iterator. |
| addAll(int index, Collection C) | Used to insert all of the elements starting at the specified position from a specific collection into the mentioned list. |
| clear() | This method is used to remove all the elements from any list. |
| clone() | This method is used to return a shallow copy of an ArrayList in Java. |
| contains? (Object o) | Returns true if this list contains the specified element. |
| ensureCapacity?(int minCapacity) | Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument. |
| forEach?(Consumer<? super E> action) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |
| get?(int index) | Returns the element at the specified position in this list. |
| indexOf(Object O) | The index the first occurrence of a specific element is either returned or -1 in case the element is not in the list. |

| Method | Description |
| --- | --- |
| isEmpty?() | Returns true if this list contains no elements. |
| lastIndexOf(Object O) | The index of the last occurrence of a specific element is either returned or -1 in case the element is not in the list. |
| listIterator?() | Returns a list iterator over the elements in this list (in proper sequence). |
| listIterator?(int index) | Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list. |
| remove?(int index) | Removes the element at the specified position in this list. |
| remove? (Object o) | Removes the first occurrence of the specified element from this list, if it is present. |
| removeAll?(Collection c) | Removes from this list all of its elements that are contained in the specified collection. |
| removeIf?(Predicate filter) | Removes all of the elements of this collection that satisfy the given predicate. |
| removeRange?(int fromIndex, int toIndex) | Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| retainAll?(Collection<?> c) | Retains only the elements in this list that are contained in the specified collection. |
| set?(int index, E element) | Replaces the element at the specified position in this list with the specified element. |
| size?() | Returns the number of elements in this list. |

| Method | Description |
|---|---|
| spliterator?() | Creates a late-binding and fail-fast Spliterator over the elements in this list. |
| subList?(int fromIndex, int toIndex) | Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. |
| toArray() | This method is used to return an array containing all of the elements in the list in the correct order. |
| toArray(Object[] O) | It is also used to return an array containing all of the elements in this list in the correct order same as the previous method. |
| trimToSize() | This method is used to trim the capacity of the instance of the ArrayList to the list's current size. |

**Example of ArrayList:**

```
import java.util.*;
class Main{
    public static void main(String[] args){
        ArrayList <String> al = new ArrayList<>();
        System.out.println(al);             // prints empty ArrayList
        al.add("Java");       // adding elements in ArrayList
        al.add("C");
        al.add("C++");
        al.add("Python");
        al.add("Java");
        System.out.println(al);     // printing ArrayList using it's object
        String str1 = al.get(2);            // get element of 2nd index
```

```
            System.out.println("Index 2 : " +str1);
            String str2 = al.remove(2);         // removes of 2nd index
            System.out.println("Removed : " +str2);
            System.out.println("Updated List: " +al);
            int num = al.indexOf("Java");     // gets index of First occurrence of
element
            System.out.println("FirstIndex of Java: "+ num);
            int num1 = al.lastIndexOf("Java");        // gets index of First
occurrence of element
            System.out.println("LastIndex of Java: "+ num1);
            al.set(3,"HTML");   // replace 3rd index element with new element
            System.out.println("Updated List: " +al);
        }
}
```

- **LinkedList class**

Java LinkedList class provides implementation of linked-list data structure. It used doubly linked list to store the elements.

1. LinkedList class extends AbstractSequentialList and implements List, Deque and Queue interfaces.

2. It can be used as List, stack or Queue as it implements all the related interfaces.

3. It is dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.

4. It can contain duplicate elements and it is not synchronized.

5. Reverse Traversing is difficult in linked list.

6. In LinkedList, manipulation is fast because no shifting needs to be occurred.

- **LinkedList Constructors**

LinkedList class has two constructors.

LinkedList() // It creates an empty LinkedList

LinkedList( Collection c) // It creates a LinkedList that is initialized with elements of the Collection c

**Methods of LinkedList:**

| Method | Description |
|---|---|
| add(int index, E element) | This method Inserts the specified element at the specified position in this list. |
| add(E e) | This method Appends the specified element to the end of this list. |
| addAll(int index, Collection<E> c) | This method Inserts all of the elements in the specified collection into this list, starting at the specified position. |
| addAll(Collection<E> c) | 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) | This method Inserts the specified element at the beginning of this list. |
| addLast(E e) | This method Appends the specified element to the end of this list. |
| clear() | This method removes all of the elements from this list. |
| clone() | This method returns a shallow copy of this LinkedList. |

| Method | Description |
| --- | --- |
| contains(Object o) | This method returns true if this list contains the specified element. |
| descendingIterator() | This method returns an iterator over the elements in this deque in reverse sequential order. |
| element() | This method retrieves but does not remove, the head (first element) of this list. |
| get(int index) | This method returns the element at the specified position in this list. |
| getFirst() | This method returns the first element in this list. |
| getLast() | This method returns the last element in this list. |
| indexOf(Object o) | 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) | 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) | 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) | This method Adds the specified element as the tail (last element) of this list. |
| offerFirst(E e) | This method Inserts the specified element at the front of this list. |

| Method | Description |
| --- | --- |
| offerLast(E e) | This method Inserts the specified element at the end of this list. |
| peek() | This method retrieves but does not remove, the head (first element) of this list. |
| peekFirst() | This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty. |
| peekLast() | This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty. |
| poll() | This method retrieves and removes the head (first element) of this list. |
| pollFirst() | This method retrieves and removes the first element of this list, or returns null if this list is empty. |
| pollLast() | This method retrieves and removes the last element of this list, or returns null if this list is empty. |
| pop() | This method Pops an element from the stack represented by this list. |
| push(E e) | This method pushes an element onto the stack represented by this list. |
| remove() | This method retrieves and removes the head (first element) of this list. |
| remove(int index) | This method removes the element at the specified position in this list. |

| Method | Description |
| --- | --- |
| remove(Object o) | This method removes the first occurrence of the specified element from this list if it is present. |
| removeFirst() | This method removes and returns the first element from this list. |
| removeFirstOccurrence(Object o) | This method removes the first occurrence of the specified element in this list (when traversing the list from head to tail). |
| removeLast() | This method removes and returns the last element from this list. |
| removeLastOccurrence(Object o) | 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) | This method replaces the element at the specified position in this list with the specified element. |
| size() | This method returns the number of elements in this list. |
| spliterator() | This method creates a late-binding and fail-fast Spliterator over the elements in this list. |
| toArray() | This method returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| toArray(T[] a) | 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. |

| Method | Description |
| --- | --- |
| toString() | 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. |

**Example of LinkedList:**

Let's take an example to create a linked-list and add elements using add and other methods as well. See the below example.

```
import java.util.* ;

class Demo {
  public static void main(String[] args) {
    LinkedList< String> ll = new LinkedList< String>();
    ll.add("a");
    ll.add("b");
    ll.add("c");
    ll.addLast("z");
    ll.addFirst("A");
    System.out.println(ll);
  }
}
```

Output

[A, a, b,c, z]

- **Difference between ArrayList and Linked List**

ArrayList and LinkedList are the Collection classes, and both of them implements the List interface. The ArrayList class creates the list which is

internally stored in a dynamic array that grows or shrinks in size as the elements are added or deleted from it. LinkedList also creates the list which is internally stored in a DoublyLinked List. Both the classes are used to store the elements in the list, but the major difference between both the classes is that ArrayList allows random access to the elements in the list as it operates on an index-based data structure. On the other hand, the LinkedList does not allow random access as it does not have indexes to access elements directly, it has to traverse the list to retrieve or access an element from the list.

Some more differences:

ArrayList extends AbstractList class whereas LinkedList extends AbstractSequentialList.

AbstractList implements List interface, thus it can behave as a list only whereas LinkedList implements List, Deque and Queue interface, thus it can behave as a Queue and List both.

In a list, access to elements is faster in ArrayList as random access is also possible. Access to LinkedList elements is slower as it follows sequential access only.

In a list, manipulation of elements is slower in ArrayList whereas it is faster in LinkedList.