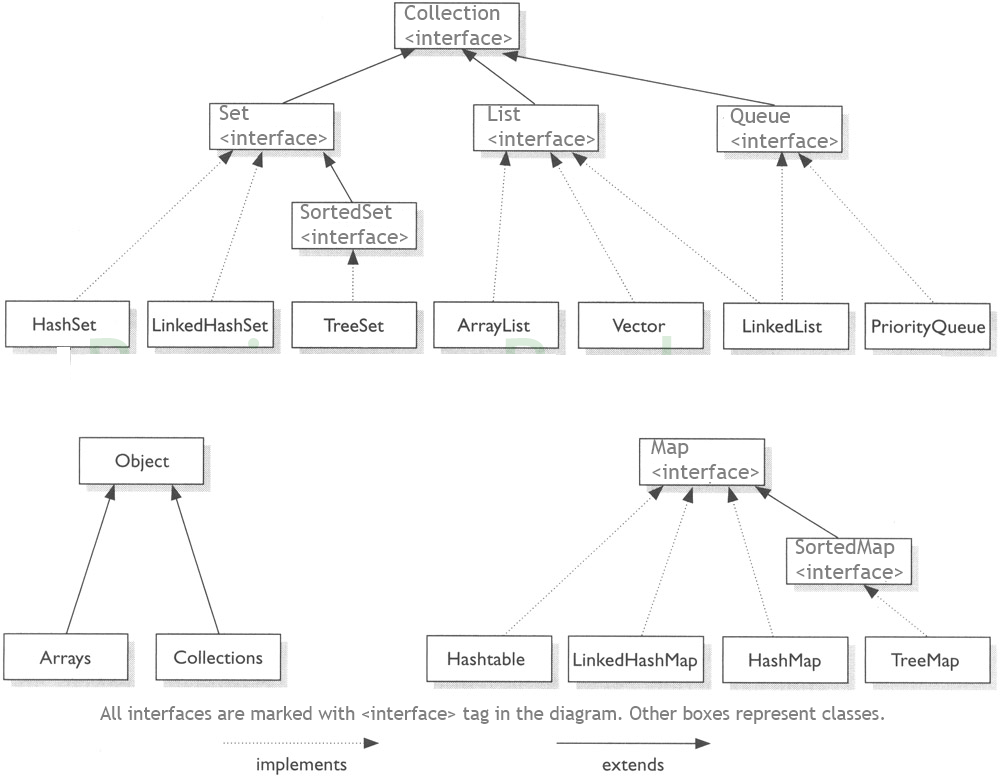
**Collections**

****

**ArrayList**

1) add( Object o): This method adds an object o to the arraylist.

obj.add("hello");

This statement would add a string hello in the arraylist at last position.

2) add(int index, Object o): It adds the object o to the array list at the given index.

obj.add(2, "bye");

It will add the string bye to the 2nd index (3rd position as the array list starts with index 0) of array list.

3) remove(Object o): Removes the object o from the ArrayList.

obj.remove("Chaitanya");

This statement will remove the string “Chaitanya” from the ArrayList.

4) remove(int index): Removes element from a given index.

obj.remove(3);

It would remove the element of index 3 (4th element of the list – List starts with o).

5) set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.

obj.set(2, "Tom");

It would replace the 3rd element (index =2 is 3rd element) with the value Tom.

6) int indexOf(Object o): Gives the index of the object o. If the element is not found in the list then this method returns the value -1.

int pos = obj.indexOf("Tom");

This would give the index (position) of the string Tom in the list.

7) Object get(int index): It returns the object of list which is present at the specified index.

String str= obj.get(2);

Function get would return the string stored at 3rd position (index 2) and would be assigned to the string “str”. We have stored the returned value in string variable because in our example we have defined the ArrayList is of String type. If you are having integer array list then the returned value should be stored in an integer variable.

8) int size(): It gives the size of the ArrayList – Number of elements of the list.

int numberofitems = obj.size();

9) boolean contains(Object o): It checks whether the given object o is present in the array list if its there then it returns true else it returns false.

obj.contains("Steve");

It would return true if the string “Steve” is present in the list else we would get false.

10) clear(): It is used for removing all the elements of the array list in one go. The below code will remove all the elements of ArrayList whose object is obj.

**LinkedList**

1) **boolean add(Object item)**: It adds the item at the end of the list.

llistobj.add("Hello");

It would add the string “Hello” at the end of the linked list.

2) **void add(int index, Object item)**: It adds an item at the given index of the the list.

llistobj.add(2, "bye");

This will add the string “bye” at the 3rd position( 2 index is 3rd position as index starts with 0).

3) **boolean addAll(Collection c)**: It adds all the elements of the specified collection c to the list. It throws NullPointerException if the specified collection is null. Consider the below example –

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

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

arraylist.add("String1");

arraylist.add("String2");

llistobj.addAll(arraylist);

This piece of code would add all the elements of ArrayList to the LinkedList.

4) **boolean addAll(int index, Collection c)**: It adds all the elements of collection c to the list starting from a give index in the list. It throws NullPointerException if the collection c is null and IndexOutOfBoundsException when the specified index is out of the range.

llistobj.add(5, arraylist);

It would add all the elements of the ArrayList to the LinkedList starting from position 6 (index 5).

5) **void addFirst(Object item)**: It adds the item (or element) at the first position in the list.

llistobj.addFirst("text");

It would add the string “text” at the beginning of the list.

6) **void addLast(Object item)**: It inserts the specified item at the end of the list.

llistobj.addLast("Chaitanya");

This statement will add a string “Chaitanya” at the end position of the linked list.

7) **void clear()**: It removes all the elements of a list.

llistobj.clear();

8) **Object clone()**: It returns the copy of the list.

For e.g. My linkedList has four items: text1, text2, text3 and text4.

Object str= llistobj.clone();

System.out.println(str);

Output: The output of above code would be:

[text1, text2, text3, text4]

9) **boolean contains(Object item)**: It checks whether the given item is present in the list or not. If the item is present then it returns true else false.

boolean var = llistobj.contains("TestString");

It will check whether the string “TestString” exist in the list or not.

10) **Object get(int index)**: It returns the item of the specified index from the list.

Object var = llistobj.get(2);

It will fetch the 3rd item from the list.

11) **Object getFirst()**: It fetches the first item from the list.

Object var = llistobj.getFirst();

12) **Object getLast()**: It fetches the last item from the list.

Object var= llistobj.getLast();

13) **int indexOf(Object item)**: It returns the index of the specified item.

llistobj.indexOf("bye");

14) **int lastIndexOf(Object item)**: It returns the index of last occurrence of the specified element.

int pos = llistobj.lastIndexOf("hello);

integer variable pos will be having the index of last occurrence of string “hello”.

15) **Object poll()**: It returns and removes the first item of the list.

Object o = llistobj.poll();

16) **Object pollFirst()**: same as poll() method. Removes the first item of the list.

Object o = llistobj.pollFirst();

17) **Object pollLast()**: It returns and removes the last element of the list.

Object o = llistobj.pollLast();

18) **Object remove()**: It removes the first element of the list.

llistobj.remove();

19) **Object remove(int index)**: It removes the item from the list which is present at the specified index.

llistobj.remove(4);

It will remove the 5th element from the list.

20) **Object remove(Object obj)**: It removes the specified object from the list.

llistobj.remove("Test Item");

21) **Object removeFirst()**: It removes the first item from the list.

llistobj.removeFirst();

22) **Object removeLast()**: It removes the last item of the list.

llistobj.removeLast();

23) **Object removeFirstOccurrence(Object item)**: It removes the first occurrence of the specified item.

llistobj.removeFirstOccurrence("text");

It will remove the first occurrence of the string “text” from the list.

24) **Object removeLastOccurrence(Object item)**: It removes the last occurrence of the given element.

llistobj.removeLastOccurrence("String1);

It will remove the last occurrence of string “String1”.

25) **Object set(int index, Object item)**: It updates the item of specified index with the give value.

llistobj.set(2, "Test");

It will update the 3rd element with the string “Test”.

26)**int size()**: It returns the number of elements of the list.

llistobj.size();

# HashSet Class in Java with example

BY CHAITANYA SINGH | FILED UNDER: [JAVA COLLECTIONS](https://beginnersbook.com/category/java-collections/)

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. However it can be synchronized explicitly like this: Set s = Collections.synchronizedSet(new HashSet(...));

**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 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);

}

}

Output:

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

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