Collection(I)

* **boolean add(E e)**
* **boolean contains(Object o)** //Returns true if this collection contains the specified element.
* **Iterator<E> iterator()** //Returns an iterator over the elements in this collection.
* **boolean remove(Object o)** //Removes a single instance of the specified element from this collection
* **int size()** //Returns the number of elements in this collection.
* **object[] toArray()** //Returns an array containing all of the elements in this collection.

# **|---List**

| |

| |----**ArrayList**

* **boolean add (E e)** // Appends the specified element to the end of this list.
* **void add(int index, E element)** //Inserts at the specified position and Shifts current and subsequent elements to the right (adds one to their indices)
* **int indexOf(Object o) /**/index of the first occurrence of the element in this list, or -1 if this list does not contain the element.
* **E get(int index)** //Returns the element at the specified position in this list.
* **ListIterator listIterator()**
* **E set(int index, E element**) //Replaces the element at the specified position .Returns the previous element
* **E remove(int index)** //Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices).
* **boolean remove(Object o) //**Removes the first occurrence of the specified element from this list, if it is present.

| |----**LinkedList**

* **Boolean add(E e) //**Appends the specified element to the end of this list.
* **void** **add(int index, E element) //**Inserts the specified element at the specified position in this list.
* **void addFirst(E e) //**Inserts the specified element at the beginning of this list.
* **void addLast(E e)** //Appends the specified element to the end of this list.
* **E element() //**Retrieves, but does not remove, the head (first element) of this list.
* **Iterator<E> descendingIterator() //**Returns an iterator over the elements in this deque in reverse sequential order.
* **E getFirst()** // Returns the first element in this list.
* **E getLast() //** Returns the last element in this list
* **Boolean offer(E e) //**Adds the specified element as the tail (last element) of this list.
* **E peek() //Retrieves, but does not remove, the head (first element) of this list.**
* **E poll() //Retrieves and removes the head (first element) of this list.**

| |----**Vector---Stack**

* **Boolean empty() //**Tests if this stack is empty.
* **E peek() //**Looks at the object at the top of this stack without removing it from the stack.
* **E pop() //**Removes the object at the top of this stack and returns that object as the value of this function.
* **E push(E item) //**Pushes an item onto the top of this stack.
* **Int search(Object o) //**Returns the 1-based position where an object is on this stack.

|

|

# **|----Set**

| |

| |----**HashSet----LinkedHashSet**

* **Same as Collection Class Methods**

| |----TreeSet

* **E first() //**Returns the first (lowest) element currently in this set.
* **E last() //**Returns the last (highest) element currently in this set**.**

|

|

|----Queue

* **boolean add(E e) //**Inserts the specified element into this queue, returning true upon success throws IllegalStateException if no space is available
* **boolean offer(E e)** //this method is generally preferable to add(E), which can fail to insert an element only by throwing an exception.
* **E element()** //Retrieves, but does not remove, the head of this queue.
* **boolean offer(E e)** //Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
* **E peek()** //Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
* **E poll() //Retrieves and removes the head of this queue, or returns null if this queue is empty.**
* **E remove() //Retrieves and removes the head of this queue.**

| |

| |----PriorityQueue(I)(Heap Representation in Java)

* **boolean offer(E e) //**Inserts the specified element into this priority queue.
* **E peek() //**Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
* **E poll()//** Retrieves and removes the head of this queue, or returns null if this queue is empty.

| |----BlockingQueue

**Add() internally just call offer() method and does nothing extra.**

# **|---Map**

* **boolean containsKey(Object key) //**Returns true if this map contains a mapping for the specified key.
* **boolean containsValue(Object value) /**/Returns true if this map maps one or more keys to the specified value**.**
* **boolean [isEmpty](https://docs.oracle.com/javase/7/docs/api/java/util/Map.html" \l "isEmpty())() //**Returns true if this map contains no key-value mappings.
* **int size()** //Returns the number of key-value mappings in this map**.**
* **Set<K> keySet() //**Returns a Set view of the keys contained in this map.
* **V put**(**K key, V value**) //Associates the specified value with the specified key in this map.
* **V remove(Object key) //**Removes the mapping for a key from this map if it is present.

| |

| |----**HashMap**

* **Boolean** [**isEmpty**](https://docs.oracle.com/javase/7/docs/api/java/util/Map.html#isEmpty())**() //**Returns true if this map contains no key-value mappings.

| |

| |----**TreeMap**