Java Data Structure

**ArrayList time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(n)* |  |
| *Read/Search any element* | *O(1)* | If you know the index then the complexity is O(1) |
| *Update* | *O(n)* |  |
| *Delete at beginning* | *O(n)* |  |
| *Delete in middle* | *O(n)* |  |
| *Delete at end* | *O(n)* |  |
| *Add at beginning* | *O(n)* |  |
| *Add in middle* | *O(n)* |  |
| *Add at end* | *O(n)* |  |

**LinkedList time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(n)* |  |
| *Update* | *O(n)* |  |
| *Delete at beginning* | *O(1)* |  |
| *Delete in middle* | *O(n)* |  |
| *Delete at end* | *O(n)* |  |
| *Add at beginning* | *O(1)* |  |
| *Add in middle* | *O(n)* |  |
| *Add at end* | *O(n)* |  |
|  |  |  |

**HashMap time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(1)* |  |
| *Update* | *O(1)* |  |
| *Delete* | *O(1)* |  |
| *Add* | *O(1)* |  |
|  |  |  |

**LinkedHashMap time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(1)* |  |
| *Update* | *O(1)* |  |
| *Delete* | *O(1)* |  |
| *Add at beginning* | *O(1)* |  |
| *Add in middle* | *O(n)* |  |
| *Add at end* | *O(n)* |  |

**TreeMap time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(log n)* |  |
| *Update* | *O(log n)* |  |
| *Delete* | *O(log n)* |  |
| *Add* | *O(log n)* |  |

**HashSet time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(1)* |  |
| *Update* | *O(1)* |  |
| *Delete* | *O(1)* |  |
| *Add* | *O(1)* |  |
|  |  |  |
|  |  |  |

**LinkedHashSet time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(1)* |  |
| *Update* | *O(1)* |  |
| *Delete* | *O(1)* |  |
| *Add at beginning* | *O(1)* |  |
| *Add in middle* | *O(n)* |  |
| *Add at end* | *O(n)* |  |

**TreeSet time complexity**

|  |  |  |
| --- | --- | --- |
| Description | Complexity | Comments |
| *Read/Search any element* | *O(log n)* |  |
| *Update* | *O(log n)* |  |
| *Delete* | *O(log n)* |  |
| *Add* | *O(log n)* |  |
|  |  |  |
|  |  |  |

## Stack:

* **Push**: **O(1)**
* **Pop**: **O(1)**
* **Top**: **O(1)**
* **Search** (Something like lookup, as a special operation): **O(n)** (I guess so)

## Queue/Deque/Circular Queue:

* **Insert**: **O(1)**
* **Remove**: **O(1)**
* **Size**: **O(1)**

## Red-Black Tree:

* **Insert, delete and search**: Average case: **O(log n)**, Worst Case: **O(log n)**

## Heap/PriorityQueue (min/max):

* **Find Min/Find Max**: **O(1)**
* **Insert**: **O(log n)**
* **Delete Min/Delete Max**: **O(log n)**
* **Extract Min/Extract Max**: **O(log n)**
* **Lookup, Delete** (if at all provided): **O(n)**, we will have to scan all the elements as they are not ordered like BST

**What is Red-Black Tree Algorithm**

1. A node is either red or black.
2. The root is black. This rule is sometimes omitted. Since the root can always be changed from red to black, but not necessarily vice versa, this rule has little effect on analysis.
3. All leaves (NIL) are black.
4. If a node is red, then both its children are black.
5. Every [path](https://en.wikipedia.org/wiki/Path_(graph_theory)) from a given node to any of its descendant NIL nodes contains the same number of black nodes. The uniform number of black nodes in the paths from root to leaves is called the **black-height** of the red–black tree.[[14]](https://en.wikipedia.org/wiki/Red%E2%80%93black_tree#cite_note-14)

Collection Interfaces

**Interface Deque<E>**

A linear collection that supports element insertion and removal at both ends. The name *deque* is short for "double ended queue" and is usually pronounced "deck". Most Deque implementations place no fixed limits on the number of elements they may contain, but this interface supports capacity-restricted deques as well as those with no fixed size limit.

This interface defines methods to access the elements at both ends of the deque. Methods are provided to insert, remove, and examine the element. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation). The latter form of the insert operation is designed specifically for use with capacity-restricted Deque implementations; in most implementations, insert operations cannot fail.

The twelve methods described above are summarized in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **First Element (Head)** | | **Last Element (Tail)** | |
|  | *Throws exception* | *Special value* | *Throws exception* | *Special value* |
| **Insert** | [addFirst(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#addFirst(E)) | [offerFirst(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#offerFirst(E)) | [addLast(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#addLast(E)) | [offerLast(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#offerLast(E)) |
| **Remove** | [removeFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeFirst()) | [pollFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#pollFirst()) | [removeLast()](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeLast()) | [pollLast()](file:///E:\dev\java7\docs\api\java\util\Deque.html#pollLast()) |
| **Examine** | [getFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#getFirst()) | [peekFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#peekFirst()) | [getLast()](file:///E:\dev\java7\docs\api\java\util\Deque.html#getLast()) | [peekLast()](file:///E:\dev\java7\docs\api\java\util\Deque.html#peekLast()) |

This interface extends the [Queue](file:///E:\dev\java7\docs\api\java\util\Queue.html) interface. When a deque is used as a queue, FIFO (First-In-First-Out) behavior results. Elements are added at the end of the deque and removed from the beginning. The methods inherited from the Queue interface are precisely equivalent to Deque methods as indicated in the following table:

|  |  |
| --- | --- |
| Queue Method | Equivalent Deque Method |
| [add(e)](file:///E:\dev\java7\docs\api\java\util\Queue.html#add(E)) | [addLast(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#addLast(E)) |
| [offer(e)](file:///E:\dev\java7\docs\api\java\util\Queue.html#offer(E)) | [offerLast(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#offerLast(E)) |
| [remove()](file:///E:\dev\java7\docs\api\java\util\Queue.html#remove()) | [removeFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeFirst()) |
| [poll()](file:///E:\dev\java7\docs\api\java\util\Queue.html#poll()) | [pollFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#pollFirst()) |
| [element()](file:///E:\dev\java7\docs\api\java\util\Queue.html#element()) | [getFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#getFirst()) |
| [peek()](file:///E:\dev\java7\docs\api\java\util\Queue.html#peek()) | [peekFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#peek()) |

Deques can also be used as LIFO (Last-In-First-Out) stacks. This interface should be used in preference to the legacy [Stack](file:///E:\dev\java7\docs\api\java\util\Stack.html) class. When a deque is used as a stack, elements are pushed and popped from the beginning of the deque. Stack methods are precisely equivalent to Deque methods as indicated in the table below:

|  |  |
| --- | --- |
| Stack Method | Equivalent Deque Method |
| [push(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#push(E)) | [addFirst(e)](file:///E:\dev\java7\docs\api\java\util\Deque.html#addFirst(E)) |
| [pop()](file:///E:\dev\java7\docs\api\java\util\Deque.html#pop()) | [removeFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeFirst()) |
| [peek()](file:///E:\dev\java7\docs\api\java\util\Deque.html#peek()) | [peekFirst()](file:///E:\dev\java7\docs\api\java\util\Deque.html#peekFirst()) |

Note that the [peek](file:///E:\dev\java7\docs\api\java\util\Deque.html#peek()) method works equally well when a deque is used as a queue or a stack; in either case, elements are drawn from the beginning of the deque. This interface provides two methods to remove interior elements, [removeFirstOccurrence](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeFirstOccurrence(java.lang.Object)) and [removeLastOccurrence](file:///E:\dev\java7\docs\api\java\util\Deque.html#removeLastOccurrence(java.lang.Object)). Unlike the [List](file:///E:\dev\java7\docs\api\java\util\List.html) interface, this interface does not provide support for indexed access to elements. While Deque implementations are not strictly required to prohibit the insertion of null elements, they are strongly encouraged to do so. Users of any Deque implementations that do allow null elements are strongly encouraged *not* to take advantage of the ability to insert nulls. This is so because null is used as a special return value by various methods to indicated that the deque is empty. Deque implementations generally do not define element-based versions of the equals and hashCode methods, but instead inherit the identity-based versions from class Object.

**All Known Implementing Classes:**

**Normal Classes**

[**ArrayDeque**](file:///E:\dev\java7\docs\api\java\util\ArrayDeque.html)**,**[**LinkedList**](file:///E:\dev\java7\docs\api\java\util\LinkedList.html)

**Concurrent Classes**

[**ConcurrentLinkedDeque**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html)**,**[**LinkedBlockingDeque**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)

## Interface ListIterator<E>

An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list. A ListIterator has no current element; its *cursor position* always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next(). An iterator for a list of length n hasn+1 possible cursor positions, as illustrated by the carets (^) below:

Element(0) Element(1) Element(2) ... Element(n-1)

cursor positions: ^ ^ ^ ^ ^

Note that the [remove()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#remove()) and [set(Object)](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#set(E)) methods are *not* defined in terms of the cursor position; they are defined to operate on the last element returned by a call to [next()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#next()) or [previous()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previous()).

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| void | [add](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#add(E))([E](file:///E:\dev\java7\docs\api\java\util\ListIterator.html) e)  Inserts the specified element into the list (optional operation). |
| boolean | [hasNext](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#hasNext())()  Returns true if this list iterator has more elements when traversing the list in the forward direction. |
| boolean | [hasPrevious](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#hasPrevious())()  Returns true if this list iterator has more elements when traversing the list in the reverse direction. |
| [E](file:///E:\dev\java7\docs\api\java\util\ListIterator.html) | [next](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#next())()  Returns the next element in the list and advances the cursor position. |
| int | [nextIndex](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#nextIndex())()  Returns the index of the element that would be returned by a subsequent call to [next()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#next()). |
| [E](file:///E:\dev\java7\docs\api\java\util\ListIterator.html) | [previous](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previous())()  Returns the previous element in the list and moves the cursor position backwards. |
| int | [previousIndex](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previousIndex())()  Returns the index of the element that would be returned by a subsequent call to [previous()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previous()). |
| void | [remove](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#remove())()  Removes from the list the last element that was returned by [next()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#next()) or [previous()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previous()) (optional operation). |
| void | [set](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#set(E))([E](file:///E:\dev\java7\docs\api\java\util\ListIterator.html) e)  Replaces the last element returned by [next()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#next()) or [previous()](file:///E:\dev\java7\docs\api\java\util\ListIterator.html#previous()) with the specified element (optional operation). |

## Interface List<E>

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and hashCode methods. Declarations for other inherited methods are also included here for convenience.

The List interface provides four methods for positional (indexed) access to list elements. Lists (like Java arrays) are zero based. Note that these operations may execute in time proportional to the index value for some implementations (the LinkedList class, for example). Thus, iterating over the elements in a list is typically preferable to indexing through it if the caller does not know the implementation.

The List interface provides a special iterator, called a ListIterator, that allows element insertion and replacement, and bidirectional access in addition to the normal operations that the Iteratorinterface provides. A method is provided to obtain a list iterator that starts at a specified position in the list.

The List interface provides two methods to search for a specified object. From a performance standpoint, these methods should be used with caution. In many implementations they will perform costly linear searches.

The List interface provides two methods to efficiently insert and remove multiple elements at an arbitrary point in the list.

Note: While it is permissible for lists to contain themselves as elements, extreme caution is advised: the equals and hashCode methods are no longer well defined on such a list.

Some list implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically NullPointerException or ClassCastException. Attempting to query the presence of an ineligible element may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible element whose completion would not result in the insertion of an ineligible element into the list may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

**All Known Implementing Classes:**

[**AbstractList**](file:///E:\dev\java7\docs\api\java\util\AbstractList.html)**,**[**AbstractSequentialList**](file:///E:\dev\java7\docs\api\java\util\AbstractSequentialList.html)**,**[**AttributeList**](file:///E:\dev\java7\docs\api\javax\management\AttributeList.html)**,**[**RoleList**](file:///E:\dev\java7\docs\api\javax\management\relation\RoleList.html)**,**[**RoleUnresolvedList**](file:///E:\dev\java7\docs\api\javax\management\relation\RoleUnresolvedList.html)

**Normal Classes**

[**ArrayList**](file:///E:\dev\java7\docs\api\java\util\ArrayList.html) **,** [**LinkedList**](file:///E:\dev\java7\docs\api\java\util\LinkedList.html) **,** [**Stack**](file:///E:\dev\java7\docs\api\java\util\Stack.html)**,**[**Vector**](file:///E:\dev\java7\docs\api\java\util\Vector.html)

**Concurrent Classes**

[**CopyOnWriteArrayList**](file:///E:\dev\java7\docs\api\java\util\concurrent\CopyOnWriteArrayList.html)

## Interface Map<K,V>

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.

The Map interface provides three *collection views*, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The *order* of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the TreeMap class, make specific guarantees as to their order; others, like the HashMap class, do not.

Note: great care must be exercised if mutable objects are used as map keys. The behavior of a map is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is a key in the map. A special case of this prohibition is that it is not permissible for a map to contain itself as a key. While it is permissible for a map to contain itself as a value, extreme caution is advised: the equals and hashCode methods are no longer well defined on such a map.

All general-purpose map implementation classes should provide two "standard" constructors: a void (no arguments) constructor which creates an empty map, and a constructor with a single argument of type Map, which creates a new map with the same key-value mappings as its argument. In effect, the latter constructor allows the user to copy any map, producing an equivalent map of the desired class. There is no way to enforce this recommendation (as interfaces cannot contain constructors) but all of the general-purpose map implementations in the JDK comply.

The "destructive" methods contained in this interface, that is, the methods that modify the map on which they operate, are specified to throw UnsupportedOperationException if this map does not support the operation. If this is the case, these methods may, but are not required to, throw an UnsupportedOperationException if the invocation would have no effect on the map. For example, invoking the [putAll(Map)](file:///E:\dev\java7\docs\api\java\util\Map.html#putAll(java.util.Map)) method on an unmodifiable map may, but is not required to, throw the exception if the map whose mappings are to be "superimposed" is empty.

Some map implementations have restrictions on the keys and values they may contain. For example, some implementations prohibit null keys and values, and some have restrictions on the types of their keys. Attempting to insert an ineligible key or value throws an unchecked exception, typically NullPointerException or ClassCastException. Attempting to query the presence of an ineligible key or value may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible key or value whose completion would not result in the insertion of an ineligible element into the map may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

Many methods in Collections Framework interfaces are defined in terms of the [equals](file:///E:\dev\java7\docs\api\java\lang\Object.html#equals(java.lang.Object)) method. For example, the specification for the [containsKey(Object key)](file:///E:\dev\java7\docs\api\java\util\Map.html#containsKey(java.lang.Object)) method says: "returns true if and only if this map contains a mapping for a key k such that (key==null ? k==null : key.equals(k))." This specification should *not* be construed to imply that invoking Map.containsKey with a non-null argument key will cause key.equals(k) to be invoked for any key k. Implementations are free to implement optimizations whereby the equals invocation is avoided, for example, by first comparing the hash codes of the two keys. (The [Object.hashCode()](file:///E:\dev\java7\docs\api\java\lang\Object.html#hashCode()) specification guarantees that two objects with unequal hash codes cannot be equal.) More generally, implementations of the various Collections Framework interfaces are free to take advantage of the specified behavior of underlying [Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) methods wherever the implementor deems it appropriate.

**All Known Implementing Classes:**

[AbstractMap](file:///E:\dev\java7\docs\api\java\util\AbstractMap.html), [Attributes](file:///E:\dev\java7\docs\api\java\util\jar\Attributes.html), [AuthProvider](file:///E:\dev\java7\docs\api\java\security\AuthProvider.html), [PrinterStateReasons](file:///E:\dev\java7\docs\api\javax\print\attribute\standard\PrinterStateReasons.html), , [Provider](file:///E:\dev\java7\docs\api\java\security\Provider.html),[RenderingHints](file:///E:\dev\java7\docs\api\java\awt\RenderingHints.html), [SimpleBindings](file:///E:\dev\java7\docs\api\javax\script\SimpleBindings.html), [TabularDataSupport](file:///E:\dev\java7\docs\api\javax\management\openmbean\TabularDataSupport.html), , [UIDefaults](file:///E:\dev\java7\docs\api\javax\swing\UIDefaults.html),

**Normal Classes**

[**EnumMap**](file:///E:\dev\java7\docs\api\java\util\EnumMap.html)**,**[**HashMap**](file:///E:\dev\java7\docs\api\java\util\HashMap.html)**,**[**Hashtable**](file:///E:\dev\java7\docs\api\java\util\Hashtable.html)**,**[**IdentityHashMap**](file:///E:\dev\java7\docs\api\java\util\IdentityHashMap.html)**,**[**LinkedHashMap**](file:///E:\dev\java7\docs\api\java\util\LinkedHashMap.html) **,** [**TreeMap**](file:///E:\dev\java7\docs\api\java\util\TreeMap.html) **,** [**WeakHashMap**](file:///E:\dev\java7\docs\api\java\util\WeakHashMap.html) **,** [**Properties**](file:///E:\dev\java7\docs\api\java\util\Properties.html)

**Concurrent Classes**

[**ConcurrentHashMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)**,**[**ConcurrentSkipListMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| void | [clear](file:///E:\dev\java7\docs\api\java\util\Map.html#clear())()  Removes all of the mappings from this map (optional operation). |
| boolean | [containsKey](file:///E:\dev\java7\docs\api\java\util\Map.html#containsKey(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Returns true if this map contains a mapping for the specified key. |
| boolean | [containsValue](file:///E:\dev\java7\docs\api\java\util\Map.html#containsValue(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Returns true if this map maps one or more keys to the specified value. |
| [Set](file:///E:\dev\java7\docs\api\java\util\Set.html)<[Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\Map.html),[V](file:///E:\dev\java7\docs\api\java\util\Map.html)>> | [entrySet](file:///E:\dev\java7\docs\api\java\util\Map.html#entrySet())()  Returns a [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the mappings contained in this map. |
| boolean | [equals](file:///E:\dev\java7\docs\api\java\util\Map.html#equals(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Compares the specified object with this map for equality. |
| [V](file:///E:\dev\java7\docs\api\java\util\Map.html) | [get](file:///E:\dev\java7\docs\api\java\util\Map.html#get(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| int | [hashCode](file:///E:\dev\java7\docs\api\java\util\Map.html#hashCode())()  Returns the hash code value for this map. |
| boolean | [isEmpty](file:///E:\dev\java7\docs\api\java\util\Map.html#isEmpty())()  Returns true if this map contains no key-value mappings. |
| [Set](file:///E:\dev\java7\docs\api\java\util\Set.html)<[K](file:///E:\dev\java7\docs\api\java\util\Map.html)> | [keySet](file:///E:\dev\java7\docs\api\java\util\Map.html#keySet())()  Returns a [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the keys contained in this map. |
| [V](file:///E:\dev\java7\docs\api\java\util\Map.html) | [put](file:///E:\dev\java7\docs\api\java\util\Map.html#put(K,%20V))([K](file:///E:\dev\java7\docs\api\java\util\Map.html) key, [V](file:///E:\dev\java7\docs\api\java\util\Map.html) value)  Associates the specified value with the specified key in this map (optional operation). |
| void | [putAll](file:///E:\dev\java7\docs\api\java\util\Map.html#putAll(java.util.Map))([Map](file:///E:\dev\java7\docs\api\java\util\Map.html)<? extends [K](file:///E:\dev\java7\docs\api\java\util\Map.html),? extends [V](file:///E:\dev\java7\docs\api\java\util\Map.html)> m)  Copies all of the mappings from the specified map to this map (optional operation). |
| [V](file:///E:\dev\java7\docs\api\java\util\Map.html) | [remove](file:///E:\dev\java7\docs\api\java\util\Map.html#remove(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Removes the mapping for a key from this map if it is present (optional operation). |
| int | [size](file:///E:\dev\java7\docs\api\java\util\Map.html#size())()  Returns the number of key-value mappings in this map. |
| [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<[V](file:///E:\dev\java7\docs\api\java\util\Map.html)> | [values](file:///E:\dev\java7\docs\api\java\util\Map.html#values())()  Returns a [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) view of the values contained in this map. |

## Interface NavigableMap<K,V>

A [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) extended with navigation methods returning the closest matches for given search targets. Methods lowerEntry, floorEntry, ceilingEntry, and higherEntry return Map.Entry objects associated with keys respectively less than, less than or equal, greater than or equal, and greater than a given key, returning null if there is no such key. Similarly, methods lowerKey, floorKey,ceilingKey, and higherKey return only the associated keys. All of these methods are designed for locating, not traversing entries.

A NavigableMap may be accessed and traversed in either ascending or descending key order. The descendingMap method returns a view of the map with the senses of all relational and directional methods inverted. The performance of ascending operations and views is likely to be faster than that of descending ones. Methods subMap, headMap, and tailMap differ from the like-named SortedMapmethods in accepting additional arguments describing whether lower and upper bounds are inclusive versus exclusive. Submaps of any NavigableMap must implement the NavigableMap interface.

This interface additionally defines methods firstEntry, pollFirstEntry, lastEntry, and pollLastEntry that return and/or remove the least and greatest mappings, if any exist, else returning null.

Implementations of entry-returning methods are expected to return Map.Entry pairs representing snapshots of mappings at the time they were produced, and thus generally do *not* support the optionalEntry.setValue method. Note however that it is possible to change mappings in the associated map using method put.

Methods [subMap(K, K)](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#subMap(K,%20K)), [headMap(K)](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#headMap(K)), and [tailMap(K)](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#tailMap(K)) are specified to return SortedMap to allow existing implementations of SortedMap to be compatibly retrofitted to implement NavigableMap, but extensions and implementations of this interface are encouraged to override these methods to return NavigableMap. Similarly, [SortedMap.keySet()](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#keySet()) can be overriden to return NavigableSet.

**All Known Implementing Classes:**

**Normal Classes**

[**TreeMap**](file:///E:\dev\java7\docs\api\java\util\TreeMap.html)

**Concurrent Classes**

[**ConcurrentSkipListMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [ceilingEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#ceilingEntry(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such key. |
| [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) | [ceilingKey](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#ceilingKey(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns the least key greater than or equal to the given key, or null if there is no such key. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [descendingKeySet](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#descendingKeySet())()  Returns a reverse order [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) view of the keys contained in this map. |
| [NavigableMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [descendingMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#descendingMap())()  Returns a reverse order view of the mappings contained in this map. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [firstEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#firstEntry())()  Returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [floorEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#floorEntry(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key. |
| [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) | [floorKey](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#floorKey(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns the greatest key less than or equal to the given key, or null if there is no such key. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [headMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#headMap(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) toKey)  Returns a view of the portion of this map whose keys are strictly less than toKey. |
| [NavigableMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [headMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#headMap(K,%20boolean))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) toKey, boolean inclusive)  Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [higherEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#higherEntry(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key. |
| [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) | [higherKey](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#higherKey(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns the least key strictly greater than the given key, or null if there is no such key. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [lastEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#lastEntry())()  Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [lowerEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#lowerEntry(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key. |
| [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) | [lowerKey](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#lowerKey(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) key)  Returns the greatest key strictly less than the given key, or null if there is no such key. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [navigableKeySet](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#navigableKeySet())()  Returns a [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) view of the keys contained in this map. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [pollFirstEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#pollFirstEntry())()  Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| [Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [pollLastEntry](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#pollLastEntry())()  Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| [NavigableMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [subMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#subMap(K,%20boolean,%20K,%20boolean))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) fromKey, boolean fromInclusive, [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) toKey, boolean toInclusive)  Returns a view of the portion of this map whose keys range from fromKey to toKey. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [subMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#subMap(K,%20K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) fromKey, [K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) toKey)  Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [tailMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#tailMap(K))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) fromKey)  Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| [NavigableMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html),[V](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html)> | [tailMap](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html#tailMap(K,%20boolean))([K](file:///E:\dev\java7\docs\api\java\util\NavigableMap.html) fromKey, boolean inclusive)  Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey. |

## Interface NavigableSet<E>

A [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) extended with navigation methods reporting closest matches for given search targets. Methods lower, floor, ceiling, and higher return elements respectively less than, less than or equal, greater than or equal, and greater than a given element, returning null if there is no such element. A NavigableSet may be accessed and traversed in either ascending or descending order. ThedescendingSet method returns a view of the set with the senses of all relational and directional methods inverted. The performance of ascending operations and views is likely to be faster than that of descending ones. This interface additionally defines methods pollFirst and pollLast that return and remove the lowest and highest element, if one exists, else returning null. Methods subSet,headSet, and tailSet differ from the like-named SortedSet methods in accepting additional arguments describing whether lower and upper bounds are inclusive versus exclusive. Subsets of anyNavigableSet must implement the NavigableSet interface.

The return values of navigation methods may be ambiguous in implementations that permit null elements. However, even in this case the result can be disambiguated by checking contains(null). To avoid such issues, implementations of this interface are encouraged to *not* permit insertion of null elements. (Note that sorted sets of [Comparable](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) elements intrinsically do not permit null.)

Methods [subSet(E, E)](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#subSet(E,%20E)), [headSet(E)](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#headSet(E)), and [tailSet(E)](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#tailSet(E)) are specified to return SortedSet to allow existing implementations of SortedSet to be compatibly retrofitted to implement NavigableSet, but extensions and implementations of this interface are encouraged to override these methods to return NavigableSet.

**All Known Implementing Classes:**

**Normal Classes**

[**TreeSet**](file:///E:\dev\java7\docs\api\java\util\TreeSet.html)

**Concurrent Classes**

[**ConcurrentSkipListSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [ceiling](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#ceiling(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) e)  Returns the least element in this set greater than or equal to the given element, or null if there is no such element. |
| [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [descendingIterator](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#descendingIterator())()  Returns an iterator over the elements in this set, in descending order. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [descendingSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#descendingSet())()  Returns a reverse order view of the elements contained in this set. |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [floor](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#floor(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) e)  Returns the greatest element in this set less than or equal to the given element, or null if there is no such element. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [headSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#headSet(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) toElement)  Returns a view of the portion of this set whose elements are strictly less than toElement. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [headSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#headSet(E,%20boolean))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) toElement, boolean inclusive)  Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement. |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [higher](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#higher(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) e)  Returns the least element in this set strictly greater than the given element, or null if there is no such element. |
| [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [iterator](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#iterator())()  Returns an iterator over the elements in this set, in ascending order. |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [lower](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#lower(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) e)  Returns the greatest element in this set strictly less than the given element, or null if there is no such element. |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [pollFirst](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#pollFirst())()  Retrieves and removes the first (lowest) element, or returns null if this set is empty. |
| [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) | [pollLast](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#pollLast())()  Retrieves and removes the last (highest) element, or returns null if this set is empty. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [subSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#subSet(E,%20boolean,%20E,%20boolean))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) fromElement, boolean fromInclusive, [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) toElement, boolean toInclusive)  Returns a view of the portion of this set whose elements range from fromElement to toElement. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [subSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#subSet(E,%20E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) fromElement, [E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) toElement)  Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [tailSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#tailSet(E))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) fromElement)  Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |
| [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)> | [tailSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html#tailSet(E,%20boolean))([E](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) fromElement, boolean inclusive)  Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement. |

## Interface Queue<E>

A collection designed for holding elements prior to processing. Besides basic [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) operations, queues provide additional insertion, extraction, and inspection operations. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation). The latter form of the insert operation is designed specifically for use with capacity-restricted Queue implementations; in most implementations, insert operations cannot fail.

Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner. Among the exceptions are priority queues, which order elements according to a supplied comparator, or the elements' natural ordering, and LIFO queues (or stacks) which order the elements LIFO (last-in-first-out). Whatever the ordering used, the head of the queue is that element which would be removed by a call to [remove()](file:///E:\dev\java7\docs\api\java\util\Queue.html#remove()) or [poll()](file:///E:\dev\java7\docs\api\java\util\Queue.html#poll()). In a FIFO queue, all new elements are inserted at the tail of the queue. Other kinds of queues may use different placement rules. Every Queue implementation must specify its ordering properties.

The [offer](file:///E:\dev\java7\docs\api\java\util\Queue.html#offer(E)) method inserts an element if possible, otherwise returning false. This differs from the [Collection.add](file:///E:\dev\java7\docs\api\java\util\Collection.html#add(E)) method, which can fail to add an element only by throwing an unchecked exception. The offer method is designed for use when failure is a normal, rather than exceptional occurrence, for example, in fixed-capacity (or "bounded") queues.

The [remove()](file:///E:\dev\java7\docs\api\java\util\Queue.html#remove()) and [poll()](file:///E:\dev\java7\docs\api\java\util\Queue.html#poll()) methods remove and return the head of the queue. Exactly which element is removed from the queue is a function of the queue's ordering policy, which differs from implementation to implementation. The remove() and poll() methods differ only in their behavior when the queue is empty: the remove() method throws an exception, while the poll() method returnsnull.

The [element()](file:///E:\dev\java7\docs\api\java\util\Queue.html#element()) and [peek()](file:///E:\dev\java7\docs\api\java\util\Queue.html#peek()) methods return, but do not remove, the head of the queue.

The Queue interface does not define the *blocking queue methods*, which are common in concurrent programming. These methods, which wait for elements to appear or for space to become available, are defined in the [BlockingQueue](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html) interface, which extends this interface.

Queue implementations generally do not allow insertion of null elements, although some implementations, such as [LinkedList](file:///E:\dev\java7\docs\api\java\util\LinkedList.html), do not prohibit insertion of null. Even in the implementations that permit it, null should not be inserted into a Queue, as null is also used as a special return value by the poll method to indicate that the queue contains no elements. Queue implementations generally do not define element-based versions of methods equals and hashCode but instead inherit the identity based versions from class Object, because element-based equality is not always well-defined for queues with the same elements but different ordering properties.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\Queue.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) e)  Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) | [**element**](file:///E:\dev\java7\docs\api\java\util\Queue.html#element())()  Retrieves, but does not remove, the head of this queue. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\Queue.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) e)  Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\Queue.html#peek())()  Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\Queue.html#poll())()  Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\Queue.html) | [**remove**](file:///E:\dev\java7\docs\api\java\util\Queue.html#remove())()  Retrieves and removes the head of this queue. |

**All Known Implementing Classes:**

[AbstractQueue](file:///E:\dev\java7\docs\api\java\util\AbstractQueue.html)

**Normal Classes**

[**ArrayDeque**](file:///E:\dev\java7\docs\api\java\util\ArrayDeque.html) **,** [**PriorityQueue**](file:///E:\dev\java7\docs\api\java\util\PriorityQueue.html)**,** [**LinkedList**](file:///E:\dev\java7\docs\api\java\util\LinkedList.html)

**Concurrent Classes**

[**ArrayBlockingQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)**,** [**ConcurrentLinkedDeque**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html)**,**[**ConcurrentLinkedQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html)**,**[**DelayQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\DelayQueue.html)**,**[**LinkedBlockingDeque**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)**,**[**LinkedBlockingQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html)**,** [**LinkedTransferQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) **,** [**PriorityBlockingQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\PriorityBlockingQueue.html)**,**[**SynchronousQueue**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html)

## Interface RandomAccess

Marker interface used by List implementations to indicate that they support fast (generally constant time) random access. The primary purpose of this interface is to allow generic algorithms to alter their behavior to provide good performance when applied to either random or sequential access lists. The best algorithms for manipulating random access lists (such as ArrayList) can produce quadratic behavior when applied to sequential access lists (such as LinkedList). Generic list algorithms are encouraged to check whether the given list is an instanceof this interface before applying an algorithm that would provide poor performance if it were applied to a sequential access list, and to alter their behavior if necessary to guarantee acceptable performance.

It is recognized that the distinction between random and sequential access is often fuzzy. For example, some List implementations provide asymptotically linear access times if they get huge, but constant access times in practice. Such a List implementation should generally implement this interface. As a rule of thumb, a List implementation should implement this interface if, for typical instances of the class, this loop:

for (int i=0, n=list.size(); i < n; i++)

list.get(i);

runs faster than this loop:

for (Iterator i=list.iterator(); i.hasNext(); )

i.next();

**All Known Implementing Classes:**

[**ArrayList**](file:///E:\dev\java7\docs\api\java\util\ArrayList.html)**,**[**AttributeList**](file:///E:\dev\java7\docs\api\javax\management\AttributeList.html)**,**[**CopyOnWriteArrayList**](file:///E:\dev\java7\docs\api\java\util\concurrent\CopyOnWriteArrayList.html)**,**[**RoleList**](file:///E:\dev\java7\docs\api\javax\management\relation\RoleList.html)**,**[**RoleUnresolvedList**](file:///E:\dev\java7\docs\api\javax\management\relation\RoleUnresolvedList.html)**,**[**Stack**](file:///E:\dev\java7\docs\api\java\util\Stack.html)**,**[**Vector**](file:///E:\dev\java7\docs\api\java\util\Vector.html)

**How it is used, helpful for MarkerInterface questions**

**Refer to the class AbstractList.java**

**public** List<E> subList(**int** fromIndex, **int** toIndex) {  
 **return** (**this instanceof** RandomAccess ?  
 **new** RandomAccessSubList<>(**this**, fromIndex, toIndex) :  
 **new** SubList<>(**this**, fromIndex, toIndex));  
}

[**subList**](file:///E:\dev\java7\docs\api\java\util\AbstractList.html#subList(int,%20int))(int fromIndex, int toIndex)

Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.

**It is used only in case of Stack, Vector , ArrayList and CopyOnWriteArrayList**

## Interface Set<E>

A collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. As implied by its name, this interface models the mathematical *set* abstraction.

The Set interface places additional stipulations, beyond those inherited from the Collection interface, on the contracts of all constructors and on the contracts of the add, equals and hashCode methods. Declarations for other inherited methods are also included here for convenience. (The specifications accompanying these declarations have been tailored to the Set interface, but they do not contain any additional stipulations.)

The additional stipulation on constructors is, not surprisingly, that all constructors must create a set that contains no duplicate elements (as defined above).

Note: Great care must be exercised if mutable objects are used as set elements. The behavior of a set is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is an element in the set. A special case of this prohibition is that it is not permissible for a set to contain itself as an element.

Some set implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically NullPointerException or ClassCastException. Attempting to query the presence of an ineligible element may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible element whose completion would not result in the insertion of an ineligible element into the set may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

**All Known Implementing Classes:**

[AbstractSet](file:///E:\dev\java7\docs\api\java\util\AbstractSet.html), [JobStateReasons](file:///E:\dev\java7\docs\api\javax\print\attribute\standard\JobStateReasons.html),

**Normal Classes**

[**EnumSet**](file:///E:\dev\java7\docs\api\java\util\EnumSet.html)**,**[**HashSet**](file:///E:\dev\java7\docs\api\java\util\HashSet.html) **,** [**LinkedHashSet**](file:///E:\dev\java7\docs\api\java\util\LinkedHashSet.html)**,**[**TreeSet**](file:///E:\dev\java7\docs\api\java\util\TreeSet.html)

**Concurrent Classes**

[**ConcurrentSkipListSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)**,**[**CopyOnWriteArraySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\CopyOnWriteArraySet.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [add](file:///E:\dev\java7\docs\api\java\util\Set.html#add(E))([E](file:///E:\dev\java7\docs\api\java\util\Set.html) e)  Adds the specified element to this set if it is not already present (optional operation). |
| boolean | [addAll](file:///E:\dev\java7\docs\api\java\util\Set.html#addAll(java.util.Collection))([Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? extends [E](file:///E:\dev\java7\docs\api\java\util\Set.html)> c)  Adds all of the elements in the specified collection to this set if they're not already present (optional operation). |
| void | [clear](file:///E:\dev\java7\docs\api\java\util\Set.html#clear())()  Removes all of the elements from this set (optional operation). |
| boolean | [contains](file:///E:\dev\java7\docs\api\java\util\Set.html#contains(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this set contains the specified element. |
| boolean | [containsAll](file:///E:\dev\java7\docs\api\java\util\Set.html#containsAll(java.util.Collection))([Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Returns true if this set contains all of the elements of the specified collection. |
| boolean | [equals](file:///E:\dev\java7\docs\api\java\util\Set.html#equals(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Compares the specified object with this set for equality. |
| int | [hashCode](file:///E:\dev\java7\docs\api\java\util\Set.html#hashCode())()  Returns the hash code value for this set. |
| boolean | [isEmpty](file:///E:\dev\java7\docs\api\java\util\Set.html#isEmpty())()  Returns true if this set contains no elements. |
| [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[E](file:///E:\dev\java7\docs\api\java\util\Set.html)> | [iterator](file:///E:\dev\java7\docs\api\java\util\Set.html#iterator())()  Returns an iterator over the elements in this set. |
| boolean | [remove](file:///E:\dev\java7\docs\api\java\util\Set.html#remove(java.lang.Object))([Object](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the specified element from this set if it is present (optional operation). |
| boolean | [removeAll](file:///E:\dev\java7\docs\api\java\util\Set.html#removeAll(java.util.Collection))([Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Removes from this set all of its elements that are contained in the specified collection (optional operation). |
| boolean | [retainAll](file:///E:\dev\java7\docs\api\java\util\Set.html#retainAll(java.util.Collection))([Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Retains only the elements in this set that are contained in the specified collection (optional operation). |
| int | [size](file:///E:\dev\java7\docs\api\java\util\Set.html#size())()  Returns the number of elements in this set (its cardinality). |
| [Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [toArray](file:///E:\dev\java7\docs\api\java\util\Set.html#toArray())()  Returns an array containing all of the elements in this set. |
| <T> T[] | [toArray](file:///E:\dev\java7\docs\api\java\util\Set.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array. |

## Interface SortedMap<K,V>

A [Map](file:///E:\dev\java7\docs\api\java\util\Map.html) that further provides a *total ordering* on its keys. The map is ordered according to the [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its keys, or by a [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html) typically provided at sorted map creation time. This order is reflected when iterating over the sorted map's collection views (returned by the entrySet, keySet and values methods). Several additional operations are provided to take advantage of the ordering. (This interface is the map analogue of [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html).)

All keys inserted into a sorted map must implement the Comparable interface (or be accepted by the specified comparator). Furthermore, all such keys must be *mutually comparable*: k1.compareTo(k2)(or comparator.compare(k1, k2)) must not throw a ClassCastException for any keys k1 and k2 in the sorted map. Attempts to violate this restriction will cause the offending method or constructor invocation to throw a ClassCastException.

Note that the ordering maintained by a sorted map (whether or not an explicit comparator is provided) must be *consistent with equals* if the sorted map is to correctly implement the Map interface. (See theComparable interface or Comparator interface for a precise definition of *consistent with equals*.) This is so because the Map interface is defined in terms of the equals operation, but a sorted map performs all key comparisons using its compareTo (or compare) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a tree map *is* well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the Map interface.

All general-purpose sorted map implementation classes should provide four "standard" constructors. It is not possible to enforce this recommendation though as required constructors cannot be specified by interfaces. The expected "standard" constructors for all sorted map implementations are:

1. A void (no arguments) constructor, which creates an empty sorted map sorted according to the natural ordering of its keys.
2. A constructor with a single argument of type Comparator, which creates an empty sorted map sorted according to the specified comparator.
3. A constructor with a single argument of type Map, which creates a new map with the same key-value mappings as its argument, sorted according to the keys' natural ordering.
4. A constructor with a single argument of type SortedMap, which creates a new sorted map with the same key-value mappings and the same ordering as the input sorted map.

**Note**: several methods return submaps with restricted key ranges. Such ranges are *half-open*, that is, they include their low endpoint but not their high endpoint (where applicable). If you need a *closed range* (which includes both endpoints), and the key type allows for calculation of the successor of a given key, merely request the subrange from lowEndpoint to successor(highEndpoint). For example, suppose that m is a map whose keys are strings. The following idiom obtains a view containing all of the key-value mappings in m whose keys are between low and high, inclusive:

SortedMap<String, V> sub = m.subMap(low, high+"\0");

A similar technique can be used to generate an *open range* (which contains neither endpoint). The following idiom obtains a view containing all of the key-value mappings in m whose keys are betweenlow and high, exclusive:

SortedMap<String, V> sub = m.subMap(low+"\0", high);

**All Known Implementing Classes**

**Normal Classes**

[**TreeMap**](file:///E:\dev\java7\docs\api\java\util\TreeMap.html)

**Concurrent Classes**

[**ConcurrentSkipListMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html)<? super [K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [comparator](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#comparator())()  Returns the comparator used to order the keys in this map, or null if this map uses the [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its keys. |
| [Set](file:///E:\dev\java7\docs\api\java\util\Set.html)<[Map.Entry](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html),[V](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)>> | [entrySet](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#entrySet())()  Returns a [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the mappings contained in this map. |
| [K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) | [firstKey](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#firstKey())()  Returns the first (lowest) key currently in this map. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html),[V](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [headMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#headMap(K))([K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) toKey)  Returns a view of the portion of this map whose keys are strictly less than toKey. |
| [Set](file:///E:\dev\java7\docs\api\java\util\Set.html)<[K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [keySet](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#keySet())()  Returns a [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the keys contained in this map. |
| [K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) | [lastKey](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#lastKey())()  Returns the last (highest) key currently in this map. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html),[V](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [subMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#subMap(K,%20K))([K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) fromKey, [K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) toKey)  Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)<[K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html),[V](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [tailMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#tailMap(K))([K](file:///E:\dev\java7\docs\api\java\util\SortedMap.html) fromKey)  Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<[V](file:///E:\dev\java7\docs\api\java\util\SortedMap.html)> | [values](file:///E:\dev\java7\docs\api\java\util\SortedMap.html#values())()  Returns a [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) view of the values contained in this map. |

## Interface SortedSet<E>

A [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) that further provides a *total ordering* on its elements. The elements are ordered using their [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html), or by a [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html) typically provided at sorted set creation time. The set's iterator will traverse the set in ascending element order. Several additional operations are provided to take advantage of the ordering. (This interface is the set analogue of [SortedMap](file:///E:\dev\java7\docs\api\java\util\SortedMap.html).)

All elements inserted into a sorted set must implement the Comparable interface (or be accepted by the specified comparator). Furthermore, all such elements must be *mutually comparable*:e1.compareTo(e2) (or comparator.compare(e1, e2)) must not throw a ClassCastException for any elements e1 and e2 in the sorted set. Attempts to violate this restriction will cause the offending method or constructor invocation to throw a ClassCastException.

Note that the ordering maintained by a sorted set (whether or not an explicit comparator is provided) must be *consistent with equals* if the sorted set is to correctly implement the Set interface. (See theComparable interface or Comparator interface for a precise definition of *consistent with equals*.) This is so because the Set interface is defined in terms of the equals operation, but a sorted set performs all element comparisons using its compareTo (or compare) method, so two elements that are deemed equal by this method are, from the standpoint of the sorted set, equal. The behavior of a sorted set *is*well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the Set interface.

All general-purpose sorted set implementation classes should provide four "standard" constructors: 1) A void (no arguments) constructor, which creates an empty sorted set sorted according to the natural ordering of its elements. 2) A constructor with a single argument of type Comparator, which creates an empty sorted set sorted according to the specified comparator. 3) A constructor with a single argument of type Collection, which creates a new sorted set with the same elements as its argument, sorted according to the natural ordering of the elements. 4) A constructor with a single argument of type SortedSet, which creates a new sorted set with the same elements and the same ordering as the input sorted set. There is no way to enforce this recommendation, as interfaces cannot contain constructors.

Note: several methods return subsets with restricted ranges. Such ranges are *half-open*, that is, they include their low endpoint but not their high endpoint (where applicable). If you need a *closed range*(which includes both endpoints), and the element type allows for calculation of the successor of a given value, merely request the subrange from lowEndpoint to successor(highEndpoint). For example, suppose that s is a sorted set of strings. The following idiom obtains a view containing all of the strings in s from low to high, inclusive:

SortedSet<String> sub = s.subSet(low, high+"\0");

A similar technique can be used to generate an *open range* (which contains neither endpoint). The following idiom obtains a view containing all of the Strings in s from low to high, exclusive:

SortedSet<String> sub = s.subSet(low+"\0", high);

**All Known Implementing Classes:**

**Normal Classes**

[**TreeSet**](file:///E:\dev\java7\docs\api\java\util\TreeSet.html)

**Concurrent Classes**

[**ConcurrentSkipListSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html)<? super [E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)> | [comparator](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#comparator())()  Returns the comparator used to order the elements in this set, or null if this set uses the [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its elements. |
| [E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) | [first](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#first())()  Returns the first (lowest) element currently in this set. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)> | [headSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#headSet(E))([E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) toElement)  Returns a view of the portion of this set whose elements are strictly less than toElement. |
| [E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) | [last](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#last())()  Returns the last (highest) element currently in this set. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)> | [subSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#subSet(E,%20E))([E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) fromElement, [E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) toElement)  Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| [SortedSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)<[E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html)> | [tailSet](file:///E:\dev\java7\docs\api\java\util\SortedSet.html#tailSet(E))([E](file:///E:\dev\java7\docs\api\java\util\SortedSet.html) fromElement)  Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |

Important Collection Classes and Implementation

**Class ArrayList<E>**

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.AbstractCollection](file:///E:\dev\java7\docs\api\java\util\AbstractCollection.html)<E>

[java.util.AbstractList](file:///E:\dev\java7\docs\api\java\util\AbstractList.html)<E>

java.util.ArrayList<E>

* **All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Iterable](file:///E:\dev\java7\docs\api\java\lang\Iterable.html)<E>, [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<E>, [List](file:///E:\dev\java7\docs\api\java\util\List.html)<E>, [RandomAccess](file:///E:\dev\java7\docs\api\java\util\RandomAccess.html)

## Class LinkedList<E>

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.AbstractCollection](file:///E:\dev\java7\docs\api\java\util\AbstractCollection.html)<E>

[java.util.AbstractList](file:///E:\dev\java7\docs\api\java\util\AbstractList.html)<E>

[java.util.AbstractSequentialList](file:///E:\dev\java7\docs\api\java\util\AbstractSequentialList.html)<E>

java.util.LinkedList<E>

* **Type Parameters:**

E - the type of elements held in this collection

**All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Iterable](file:///E:\dev\java7\docs\api\java\lang\Iterable.html)<E>, [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<E>, [Deque](file:///E:\dev\java7\docs\api\java\util\Deque.html)<E>, [List](file:///E:\dev\java7\docs\api\java\util\List.html)<E>, [Queue](file:///E:\dev\java7\docs\api\java\util\Queue.html)<E>

**Differences between ArrayList and LinkedList**

|  |  |
| --- | --- |
| 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.  LinkedList can be used Stack,Queue,Deque |
| 4) ArrayList is better for storing and accessing data. | LinkedList is better for manipulating data. |
| **Search :** performance of O(1) | **Search :** performance is O(n) |
| **Remove :** O(n) in worst case (while removing first element) and O(1) in best case (While removing last element) | **Remove:** O(1) first element |
| **Add:** O(n) in worst case, adding element | **Add:** O(1) , adding element |
| **Memory Overhead**: ArrayList maintains indexes and element data | **Memory Overhead:** LinkedList maintains element data and two pointers for neighbor nodes hence the memory consumption is high in LinkedList comparatively. |

See below the usage of LinkedList as Stack,Queue

*//Used as Stack***public static void** useLinkedListAsStack() {  
  
 LinkedList<String> ll = **new** LinkedList<String>();  
 **for** (**int** i = 0; i < 10; i++) {  
 ll.push(**"T-"** + i);  
 }  
 **while** (ll.size() != 0) {  
 System.***out***.println(**"ll.pop();------>"** + ll.pop());  
 }  
 System.***out***.println(**"ll.size()------>"** + ll.size());  
}  
  
*//As Queue***public static void** useLinkedListAsQueue() {  
  
 LinkedList<String> ll = **new** LinkedList<String>();  
 **for** (**int** i = 0; i < 10; i++) {  
 ll.offer(**"T-"** + i);  
 }  
 **while** (ll.size() != 0) {  
 System.***out***.println(**"ll.pop();------>"** + ll.poll());  
 }  
 System.***out***.println(**"ll.size()------>"** + ll.size());  
}

## Class HashSet<E>

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.AbstractCollection](file:///E:\dev\java7\docs\api\java\util\AbstractCollection.html)<E>

[java.util.AbstractSet](file:///E:\dev\java7\docs\api\java\util\AbstractSet.html)<E>

java.util.HashSet<E>

* **Type Parameters:**

E - the type of elements maintained by this set

**All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Iterable](file:///E:\dev\java7\docs\api\java\lang\Iterable.html)<E>, [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<E>, [Set](file:///E:\dev\java7\docs\api\java\util\Set.html)<E>

## Class Hashtable<K,V>

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.Dictionary](file:///E:\dev\java7\docs\api\java\util\Dictionary.html)<K,V>

java.util.Hashtable<K,V>

* **All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Map](file:///E:\dev\java7\docs\api\java\util\Map.html)<K,V>

## Class HashMap<K,V>

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.AbstractMap](file:///E:\dev\java7\docs\api\java\util\AbstractMap.html)<K,V>

java.util.HashMap<K,V>

* **Type Parameters:**

K - the type of keys maintained by this map

V - the type of mapped values

**All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Map](file:///E:\dev\java7\docs\api\java\util\Map.html)<K,V>

**ArrayDeque**

Resizable-array implementation of the [Deque](http://docs.oracle.com/javase/7/docs/api/java/util/Deque.html) interface. Array deques have no capacity restrictions; they grow as necessary to support usage. They are not thread-safe; in the absence of external synchronization, they do not support concurrent access by multiple threads. Null elements are prohibited. This class is likely to be faster than [Stack](http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html) when used as a stack, and faster than [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html)when used as a queue.

Class methods

|  |  |
| --- | --- |
| S.N. | Method & Description |
| 1 | [boolean add(E e)](http://www.tutorialspoint.com/java/util/arraydeque_add.htm)  This method inserts the specified element at the end of this deque. |
| 2 | [void addFirst(E e)](http://www.tutorialspoint.com/java/util/arraydeque_addfirst.htm)  This method inserts the specified element at the front of this deque. |
| 3 | [void addLast(E e)](http://www.tutorialspoint.com/java/util/arraydeque_addlast.htm)  This method inserts the specified element at the end of this deque. |
| 4 | [void clear()](http://www.tutorialspoint.com/java/util/arraydeque_clear.htm)  This method removes all of the elements from this deque. |
| 5 | [ArrayDeque<E> clone()](http://www.tutorialspoint.com/java/util/arraydeque_clone.htm)  This method returns a copy of this deque. |
| 6 | [boolean contains(Object o)](http://www.tutorialspoint.com/java/util/arraydeque_contains.htm)  This method returns true if this deque contains the specified element. |
| 7 | [Iterator<E> descendingIterator()](http://www.tutorialspoint.com/java/util/arraydeque_descendingiterator.htm)  This method returns an iterator over the elements in this deque in reverse sequential order. |
| 8 | [E element()](http://www.tutorialspoint.com/java/util/arraydeque_element.htm)  This method retrieves, but does not remove, the head of the queue represented by this deque. |
| 9 | [E getFirst()](http://www.tutorialspoint.com/java/util/arraydeque_getfirst.htm)  This method retrieves, but does not remove, the first element of this deque. |
| 10 | [E getLast()](http://www.tutorialspoint.com/java/util/arraydeque_getlast.htm)  This method retrieves, but does not remove, the last element of this deque. |
| 11 | [boolean isEmpty()](http://www.tutorialspoint.com/java/util/arraydeque_isempty.htm)  This method returns true if this deque contains no elements. |
| 12 | [Iterator<E> iterator()](http://www.tutorialspoint.com/java/util/arraydeque_iterator.htm)  This method returns an iterator over the elements in this deque. |
| 13 | [boolean offer(E e)](http://www.tutorialspoint.com/java/util/arraydeque_offer.htm)  This method inserts the specified element at the end of this deque. |
| 14 | [boolean offerFirst(E e)](http://www.tutorialspoint.com/java/util/arraydeque_offerfirst.htm)  This method inserts the specified element at the front of this deque. |
| 15 | [boolean offerLast(E e)](http://www.tutorialspoint.com/java/util/arraydeque_offerlast.htm)  This method inserts the specified element at the end of this deque. |
| 16 | [E peek()](http://www.tutorialspoint.com/java/util/arraydeque_peek.htm)  This method retrieves, but does not remove, the head of the queue represented by this deque, or returns null if this deque is empty. |
| 17 | [E peekFirst()](http://www.tutorialspoint.com/java/util/arraydeque_peekfirst.htm)  This method retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty. |
| 18 | [E peekLast()](http://www.tutorialspoint.com/java/util/arraydeque_peeklast.htm)  This method retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty. |
| 19 | [E poll()](http://www.tutorialspoint.com/java/util/arraydeque_poll.htm)  This method retrieves and removes the head of the queue represented by this deque, or returns null if this deque is empty. |
| 20 | [E pollFirst()](http://www.tutorialspoint.com/java/util/arraydeque_pollfirst.htm)  This method retrieves and removes the first element of this deque, or returns null if this deque is empty. |
| 21 | [E pollLast()](http://www.tutorialspoint.com/java/util/arraydeque_polllast.htm)  This method retrieves and removes the last element of this deque, or returns null if this deque is empty. |
| 22 | [E pop()](http://www.tutorialspoint.com/java/util/arraydeque_pop.htm)  This method pops an element from the stack represented by this deque. |
| 23 | [void push(E e)](http://www.tutorialspoint.com/java/util/arraydeque_push.htm)  This method pushes an element onto the stack represented by this deque. |
| 24 | [E remove()](http://www.tutorialspoint.com/java/util/arraydeque_remove.htm)  This method retrieves and removes the head of the queue represented by this deque. |
| 25 | [boolean remove(Object o)](http://www.tutorialspoint.com/java/util/arraydeque_remove_object.htm)  This method removes a single instance of the specified element from this deque. |
| 26 | [E removeFirst()](http://www.tutorialspoint.com/java/util/arraydeque_removefirst.htm)  This method retrieves and removes the first element of this deque. |
| 27 | [boolean removeFirstOccurrence(Object o)](http://www.tutorialspoint.com/java/util/arraydeque_removefirstoccurrence.htm)  This method removes the first occurrence of the specified element in this deque. |
| 28 | [E removeLast()](http://www.tutorialspoint.com/java/util/arraydeque_removelast.htm)  This method retrieves and removes the last element of this deque. |
| 29 | [boolean removeLastOccurrence(Object o)](http://www.tutorialspoint.com/java/util/arraydeque_removelastoccurrence_object.htm)  This method removes the last occurrence of the specified element in this deque. |
| 30 | [int size()](http://www.tutorialspoint.com/java/util/arraydeque_size.htm)  This method returns the number of elements in this deque. |
| 31 | [object[] toArray()](http://www.tutorialspoint.com/java/util/arraydeque_toarray.htm)  This method returns an array containing all of the elements in this deque in proper sequence. |

## Deque

A linear collection that supports element insertion and removal at both ends. The name *deque* is short for "double ended queue" and is usually pronounced "deck". Most Deque implementations place no fixed limits on the number of elements they may contain, but this interface supports capacity-restricted deques as well as those with no fixed size limit.

This interface defines methods to access the elements at both ends of the deque. Methods are provided to insert, remove, and examine the element. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either null or false, depending on the operation). The latter form of the insert operation is designed specifically for use with capacity-restricted Deque implementations; in most implementations, insert operations cannot fail.

Concurrent Version of Deque is **ConcurrentLinkedDeque**.

|  |  |
| --- | --- |
| Single Threaded | Concurrent |
| **Deque** | **ConcurrentLinkedDeque** |
| **ArrayDeque** | **ConcurrentLinkedDeque** |

|  |  |
| --- | --- |
| Single Threaded | Concurrent |
| **Queue** | [**LinkedBlockingQueue**](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedBlockingQueue.html) |
|  |  |

**Single Threaded and Concurrent Collections in Java**

|  |  |  |
| --- | --- | --- |
|  | **Single threaded** | **Concurrent** |
| Lists | ArrayList – generic array-based  LinkedList – do not use  Vector – deprecated | CopyOnWriteArrayList – seldom updated, often traversed |
| Queues / deques | ArrayDeque – generic array-based  Stack – deprecated  PriorityQueue – sorted retrieval operations | ArrayBlockingQueue – bounded blocking queue  ConcurrentLinkedDeque / ConcurrentLinkedQueue – unbounded linked queue (CAS)  DelayQueue – queue with delays on each element  LinkedBlockingDeque / LinkedBlockingQueue – optionally bounded linked queue (locks)  LinkedTransferQueue – may transfer elements w/o storing  PriorityBlockingQueue – concurrent PriorityQueue  SynchronousQueue – Exchanger with Queue interface |
| Maps | HashMap – generic map  EnumMap – enum keys  Hashtable – deprecated  IdentityHashMap – keys compared with ==  LinkedHashMap – keeps insertion order  TreeMap – sorted keys  WeakHashMap – useful for caches | ConcurrentHashMap – generic concurrent map  ConcurrentSkipListMap – sorted concurrent map |
| Sets | HashSet – generic set  EnumSet – set of enums  BitSet – set of bits/dense integers  LinkedHashSet – keeps insertion order  TreeSet – sorted set | ConcurrentSkipListSet – sorted concurrent set  CopyOnWriteArraySet – seldom updated, often traversed |

## Class PriorityQueue<E>

An unbounded priority [queue](http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html) based on a priority heap. The elements of the priority queue are ordered according to their [natural ordering](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html), or by a [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) provided at queue construction time, depending on which constructor is used. A priority queue does not permit null elements. A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in ClassCastException).

The *head* of this queue is the *least* element with respect to the specified ordering. If multiple elements are tied for least value, the head is one of those elements -- ties are broken arbitrarily. The queue retrieval operations poll, remove, peek, and element access the element at the head of the queue.

A priority queue is unbounded, but has an internal *capacity* governing the size of an array used to store the elements on the queue. It is always at least as large as the queue size. As elements are added to a priority queue, its capacity grows automatically. The details of the growth policy are not specified. This class and its iterator implement all of the *optional* methods of the [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) and [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) interfaces. The Iterator provided in method [iterator()](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html#iterator()) is *not* guaranteed to traverse the elements of the priority queue in any particular order. If you need ordered traversal, consider using Arrays.sort(pq.toArray()). **Note that this implementation is not synchronized.** Multiple threads should not access a PriorityQueue instance concurrently if any of the threads modifies the queue. Instead, use the thread-safe[PriorityBlockingQueue](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/PriorityBlockingQueue.html) class. Implementation note: this implementation provides O(log(n)) time for the enqueing and dequeing methods (offer, poll, remove() and add); linear time for the remove(Object) and contains(Object)methods; and constant time for the retrieval methods (peek, element, and size).

|  |  |
| --- | --- |
| Single Threaded | Concurrent |
| **PriorityQueue** | [**PriorityBlockingQueue**](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/PriorityBlockingQueue.html) |

The **java.util.PriorityQueue** class is an unbounded priority queue based on a priority heap.Following are the important points about PriorityQueue:

* The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.
* A priority queue does not permit null elements.
* A priority queue relying on natural ordering also does not permit insertion of non-comparable objects.
* Class methods

|  |  |
| --- | --- |
| S.N. | Method & Description |
| 1 | [boolean add(E e)](http://www.tutorialspoint.com/java/util/priorityqueue_add.htm)  This method inserts the specified element into this priority queue. |
| 2 | [void clear()](http://www.tutorialspoint.com/java/util/priorityqueue_clear.htm)  This method removes all of the elements from this priority queue. |
| 3 | [Comparator<? super E> comparator()](http://www.tutorialspoint.com/java/util/priorityqueue_super.htm)  This method returns the comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements. |
| 4 | [boolean contains(Object o)](http://www.tutorialspoint.com/java/util/priorityqueue_contains.htm)  This method returns true if this queue contains the specified element. |
| 5 | [Iterator<E> iterator()](http://www.tutorialspoint.com/java/util/priorityqueue_iterator.htm)  This method returns an iterator over the elements in this queue. |
| 6 | [boolean offer(E e)](http://www.tutorialspoint.com/java/util/priorityqueue_offer.htm)  This method inserts the specified element into this priority queue. |
| 7 | [E peek()](http://www.tutorialspoint.com/java/util/priorityqueue_peek.htm)  This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| 8 | [E poll()](http://www.tutorialspoint.com/java/util/priorityqueue_poll.htm)  This method retrieves and removes the head of this queue, or returns null if this queue is empty. |
| 9 | [boolean remove(Object o)](http://www.tutorialspoint.com/java/util/priorityqueue_remove.htm)  This method removes a single instance of the specified element from this queue, if it is present. |
| 10 | [int size()](http://www.tutorialspoint.com/java/util/priorityqueue_size.htm)  This method returns the number of elements in this collection. |
| 11 | [Object[] toArray()](http://www.tutorialspoint.com/java/util/priorityqueue_toarray_object.htm)  This method returns an array containing all of the elements in this queue. |
| 12 | [<T> T[] toArray(T[] a)](http://www.tutorialspoint.com/java/util/priorityqueue_toarray.htm)  This method returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array. |

## Class Stack<E>

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top. When a stack is first created, it contains no items. A more complete and consistent set of LIFO stack operations is provided by the [Deque](http://docs.oracle.com/javase/7/docs/api/java/util/Deque.html) interface and its implementations, which should be used in preference to this class. For example:

## Class Stack<E>

[java.lang.Object](file:///E:\dev\java7\docs\api\java\lang\Object.html)

[java.util.AbstractCollection](file:///E:\dev\java7\docs\api\java\util\AbstractCollection.html)<E>

[java.util.AbstractList](file:///E:\dev\java7\docs\api\java\util\AbstractList.html)<E>

[java.util.Vector](file:///E:\dev\java7\docs\api\java\util\Vector.html)<E>

java.util.Stack<E>

* **All Implemented Interfaces:**

[Serializable](file:///E:\dev\java7\docs\api\java\io\Serializable.html), [Cloneable](file:///E:\dev\java7\docs\api\java\lang\Cloneable.html), [Iterable](file:///E:\dev\java7\docs\api\java\lang\Iterable.html)<E>, [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html)<E>, [List](file:///E:\dev\java7\docs\api\java\util\List.html)<E>, [RandomAccess](file:///E:\dev\java7\docs\api\java\util\RandomAccess.html)

Deque<Integer> stack = new ArrayDeque<Integer>();

Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| **1** | [**boolean empty()**](http://www.tutorialspoint.com/java/util/stack_empty.htm)  **This method tests if this stack is empty.** |
| **2** | [**E peek()**](http://www.tutorialspoint.com/java/util/stack_peek.htm)  **This method looks at the object at the top of this stack without removing it from the stack.** |
| **3** | [**E pop()**](http://www.tutorialspoint.com/java/util/stack_pop.htm)  **This method removes the object at the top of this stack and returns that object as the value of this function.** |
| **4** | [**E push(E item)**](http://www.tutorialspoint.com/java/util/stack_push.htm)  **This method pushes an item onto the top of this stack.** |
| **5** | [**int search(Object o)**](http://www.tutorialspoint.com/java/util/stack_search.htm)  **This method returns the 1-based position where an object is on this stack.** |

|  |  |
| --- | --- |
| Single Threaded | Concurrent |
| **LinkedList** | [**ConcurrentLinkedQueue**](http://java.sun.com/javase/6/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html)**or**[**LinkedBlockingDeque**](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/LinkedBlockingDeque.html) |

[**LinkedBlockingDeque**](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/LinkedBlockingDeque.html) **is advisable**

**Ordered Collection -** Iteration order for above implementations:

* HashSet - *undefined*
* HashMap - *undefined*
* LinkedHashSet - insertion order
* LinkedHashMap - insertion order of keys (by default), or 'access order'
* ArrayList - insertion order
* LinkedList - insertion order
* TreeSet - ascending order, according to Comparable / Comparator
* TreeMap - ascending order of keys, according to Comparable / Comparator

Concurrent Collections

## Class ArrayBlockingQueue<E>

A bounded [blocking queue](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html) backed by an array. This queue orders elements FIFO (first-in-first-out). The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.

This is a classic "bounded buffer", in which a fixed-sized array holds elements inserted by producers and extracted by consumers. Once created, the capacity cannot be changed. Attempts to put an element into a full queue will result in the operation blocking; attempts to take an element from an empty queue will similarly block.

This class supports an optional fairness policy for ordering waiting producer and consumer threads. By default, this ordering is not guaranteed. However, a queue constructed with fairness set to truegrants threads access in FIFO order. Fairness generally decreases throughput but reduces variability and avoids starvation.

This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

Example is given below

**package** com.blockingqueue;  
  
**import** java.util.concurrent.BlockingQueue;  
  
**public class** Producer **implements** Runnable{  
   
 **private** BlockingQueue **queue**;  
   
 **public** Producer(BlockingQueue queue) {  
 **this**.**queue** = queue;  
 }  
  
 @Override  
 **public void** run() {  
   
 *// We are adding elements using offer() in order to check if  
 // it actually managed to insert them.* **for** (**int** i = 0; i < 8; i++) {  
 System.***out***.println(**"Trying to add to queue: String "** + i +  
 **" and the result was "** + **queue**.offer(**"String "** + i));  
   
 **try** {   
 Thread.*sleep*(500);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

**package** com.blockingqueue;  
  
**import** java.util.concurrent.BlockingQueue;  
  
**public class** Consumer **implements** Runnable{  
   
 **private** BlockingQueue **queue**;  
   
 **public** Consumer(BlockingQueue queue) {  
 **this**.**queue** = queue;  
 }  
  
 @Override  
 **public void** run() {  
   
 *// As long as there are empty positions in our array,  
 // we want to check what's going on.* **while** (**queue**.remainingCapacity() > 0) {  
 System.***out***.println(**"Queue size: "** + **queue**.size() +  
 **", remaining capacity: "** + **queue**.remainingCapacity());  
   
 **try** {  
 Thread.*sleep*(500);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

**package** com.blockingqueue;  
  
**import** java.util.concurrent.ArrayBlockingQueue;  
**import** java.util.concurrent.BlockingQueue;  
  
**public class** ArrayBlockingQueueExample {  
  
 **public static void** main(String[] args) {  
 *// Let's create a blocking queue that can hold at most 5 elements.* BlockingQueue queue = **new** ArrayBlockingQueue<>(5);  
   
 *// The two threads will access the same queue, in order  
 // to test its blocking capabilities.* Thread producer = **new** Thread(**new** Producer(queue));  
 Thread consumer = **new** Thread(**new** Consumer(queue));  
   
 producer.start();  
 consumer.start();  
 }  
}

**Methods**

|  |  |
| --- | --- |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html) e)  Inserts the specified element at the tail of this queue if it is possible to do so immediately without exceeding the queue's capacity, returning trueupon success and throwing an IllegalStateException if this queue is full. |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#clear())**()**  **Atomically removes all of the elements from this queue.** |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#contains(java.lang.Object))**(**[**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)**o)**  **Returns true if this queue contains the specified element.** |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#drainTo(java.util.Collection))**(**[**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)**<? super**[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)**> c)**  **Removes all available elements from this queue and adds them to the given collection.** |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#drainTo(java.util.Collection,%20int))**(**[**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)**<? super**[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)**> c, int maxElements)**  **Removes at most the given number of available elements from this queue and adds them to the given collection.** |
| [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[E](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#iterator())**()**  **Returns an iterator over the elements in this queue in proper sequence.** |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#offer(E))**(**[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)**e)**  **Inserts the specified element at the tail of this queue if it is possible to do so immediately without exceeding the queue's capacity, returning trueupon success and false if this queue is full.** |
| [E](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#peek())**()**  **Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.** |
| [E](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#poll())**()**  **Retrieves and removes the head of this queue, or returns null if this queue is empty.** |
| void | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#put(E))**(**[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html)**e)**  **Inserts the specified element at the tail of this queue, waiting for space to become available if the queue is full.** |
| int | [**remainingCapacity**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#remainingCapacity())**()**  **Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.** |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#remove(java.lang.Object))**(**[**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)**o)**  **Removes a single instance of the specified element from this queue, if it is present.** |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#size())**()**  **Returns the number of elements in this queue.** |
| [E](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html) | [**take**](file:///E:\dev\java7\docs\api\java\util\concurrent\ArrayBlockingQueue.html#take())**()**  **Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.** |

## Class ConcurrentHashMap<K,V>

A hash table supporting full concurrency of retrievals and adjustable expected concurrency for updates. This class obeys the same functional specification as [Hashtable](file:///E:\dev\java7\docs\api\java\util\Hashtable.html), and includes versions of methods corresponding to each method of Hashtable. However, even though all operations are thread-safe, retrieval operations do *not* entail locking, and there is *not* any support for locking the entire table in a way that prevents all access. This class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details.

Retrieval operations (including get) generally do not block, so may overlap with update operations (including put and remove). Retrievals reflect the results of the most recently *completed* update operations holding upon their onset. For aggregate operations such as putAll and clear, concurrent retrievals may reflect insertion or removal of only some entries. Similarly, Iterators and Enumerations return elements reflecting the state of the hash table at some point at or since the creation of the iterator/enumeration. They do *not* throw [ConcurrentModificationException](file:///E:\dev\java7\docs\api\java\util\ConcurrentModificationException.html). However, iterators are designed to be used by only one thread at a time.

The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default 16), which is used as a hint for internal sizing. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. Because placement in hash tables is essentially random, the actual concurrency will vary. Ideally, you should choose a value to accommodate as many threads as will ever concurrently modify the table. Using a significantly higher value than you need can waste space and time, and a significantly lower value can lead to thread contention. But overestimates and underestimates within an order of magnitude do not usually have much noticeable impact. A value of one is appropriate when it is known that only one thread will modify and all others will only read. Also, resizing this or any other kind of hash table is a relatively slow operation, so, when possible, it is a good idea to provide estimates of expected table sizes in constructors. This class and its views and iterators implement all of the *optional* methods of the [Map](file:///E:\dev\java7\docs\api\java\util\Map.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

Like [Hashtable](file:///E:\dev\java7\docs\api\java\util\Hashtable.html) but unlike [HashMap](file:///E:\dev\java7\docs\api\java\util\HashMap.html), this class does *not* allow null to be used as a key or value.

**Methods**

|  |  |
| --- | --- |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#clear())()  Removes all of the mappings from this map. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Legacy method testing if some key maps into the specified value in this table. |
| boolean | [**containsKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#containsKey(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Tests if the specified object is a key in this table. |
| boolean | [**containsValue**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#containsValue(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Returns true if this map maps one or more keys to the specified value. |
| [**Enumeration**](file:///E:\dev\java7\docs\api\java\util\Enumeration.html)<[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)> | [**elements**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#elements())()  Returns an enumeration of the values in this table. |
| [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html)<[**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)>> | [**entrySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#entrySet())()  Returns a [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the mappings contained in this map. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) | [**get**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#get(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#isEmpty())()  Returns true if this map contains no key-value mappings. |
| [**Enumeration**](file:///E:\dev\java7\docs\api\java\util\Enumeration.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)> | [**keys**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#keys())()  Returns an enumeration of the keys in this table. |
| [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)> | [**keySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#keySet())()  Returns a [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the keys contained in this map. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#put(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) value)  Maps the specified key to the specified value in this table. |
| void | [**putAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#putAll(java.util.Map))([**Map**](file:///E:\dev\java7\docs\api\java\util\Map.html)<? extends [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html),? extends [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)> m)  Copies all of the mappings from the specified map to this one. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) | [**putIfAbsent**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#putIfAbsent(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) value)  If the specified key is not already associated with a value, associate it with the given value. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Removes the key (and its corresponding value) from this map. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#remove(java.lang.Object,%20java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key, [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Removes the entry for a key only if currently mapped to a given value. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) | [**replace**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#replace(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) value)  Replaces the entry for a key only if currently mapped to some value. |
| boolean | [**replace**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#replace(K,%20V,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) oldValue, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html) newValue)  Replaces the entry for a key only if currently mapped to a given value. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#size())()  Returns the number of key-value mappings in this map. |
| [**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html)> | [**values**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentHashMap.html#values())()  Returns a [**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html) view of the values contained in this map. |

## Class ConcurrentLinkedQueue<E>

An unbounded thread-safe [queue](file:///E:\dev\java7\docs\api\java\util\Queue.html) based on linked nodes. This queue orders elements FIFO (first-in-first-out). The head of the queue is that element that has been on the queue the longest time. The tailof the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. A ConcurrentLinkedQueue is an appropriate choice when many threads will share access to a common collection. Like most other concurrent collection implementations, this class does not permit the use of null elements.

This implementation employs an efficient "wait-free" algorithm based on one described in [Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms](http://www.cs.rochester.edu/u/michael/PODC96.html) by Maged M. Michael and Michael L. Scott.

Iterators are *weakly consistent*, returning elements reflecting the state of the queue at some point at or since the creation of the iterator. They do not throw [ConcurrentModificationException](file:///E:\dev\java7\docs\api\java\util\ConcurrentModificationException.html), and may proceed concurrently with other operations. Elements contained in the queue since the creation of the iterator will be returned exactly once.

Beware that, unlike in most collections, the size method is NOT a constant-time operation. Because of the asynchronous nature of these queues, determining the current number of elements requires a traversal of the elements, and so may report inaccurate results if this collection is modified during traversal. Additionally, the bulk operations addAll, removeAll, retainAll, containsAll, equals, andtoArray are not guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might view only some of the added elements.

This class and its iterator implement all of the optional methods of the [Queue](file:///E:\dev\java7\docs\api\java\util\Queue.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a ConcurrentLinkedQueue [*happen-before*](file:///E:\dev\java7\docs\api\java\util\concurrent\package-summary.html#MemoryVisibility) actions subsequent to the access or removal of that element from the ConcurrentLinkedQueue in another thread.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html) e)  Inserts the specified element at the tail of this queue. |
| boolean | [**addAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#addAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? extends [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html)> c)  Appends all of the elements in the specified collection to the end of this queue, in the order that they are returned by the specified collection's iterator. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this queue contains the specified element. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#isEmpty())()  Returns true if this queue contains no elements. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#iterator())()  Returns an iterator over the elements in this queue in proper sequence. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html) e)  Inserts the specified element at the tail of this queue. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#peek())()  Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#poll())()  Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes a single instance of the specified element from this queue, if it is present. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#size())()  Returns the number of elements in this queue. |
| [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#toArray())()  Returns an array containing all of the elements in this queue, in proper sequence. |
| <T> T[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedQueue.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this queue, in proper sequence; the runtime type of the returned array is that of the specified array. |

Example is given below.

**package** com.concurrentLinkedQueue;  
  
**import** java.util.concurrent.\*;  
   
**public class** ConcurrentLinkedQueueExample {  
 **public static void** main(String[] args) {  
 ConcurrentLinkedQueue<String> queue = **new** ConcurrentLinkedQueue<String>();  
 Thread producer = **new** Thread(**new** Producer(queue));  
 Thread consumer = **new** Thread(**new** Consumer(queue));  
 producer.start();  
 consumer.start();  
 }  
}  
   
*// the producer puts strings on the queue***class** Producer **implements** Runnable {  
   
 ConcurrentLinkedQueue<String> **queue**;  
 Producer(ConcurrentLinkedQueue<String> queue){  
 **this**.**queue** = queue;  
 }  
 **public void** run() {  
 System.***out***.println(**"Producer Started"**);  
 **try** {  
 **for** (**int** i = 1; i < 10; i++) {  
 **queue**.add(**"String"** + i);  
 System.***out***.println(**"Added: String"** + i);  
 Thread.*currentThread*().*sleep*(200);  
 }  
 } **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
}  
   
*// the consumer removes strings from the queue***class** Consumer **implements** Runnable {  
   
 ConcurrentLinkedQueue<String> **queue**;  
 Consumer(ConcurrentLinkedQueue<String> queue){  
 **this**.**queue** = queue;  
 }  
 **public void** run() {  
 String str;  
 System.***out***.println(**"Consumer Started"**);  
 **for** (**int** x = 0; x < 10; x++) {  
 **while** ((str = **queue**.poll()) != **null**) {  
 System.***out***.println(**"Removed: "** + str);  
 }  
 **try** {  
 Thread.*currentThread*().*sleep*(500);  
 } **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
}

## Class ConcurrentLinkedDeque<E>

An unbounded concurrent [deque](file:///E:\dev\java7\docs\api\java\util\Deque.html) based on linked nodes. Concurrent insertion, removal, and access operations execute safely across multiple threads. A ConcurrentLinkedDeque is an appropriate choice when many threads will share access to a common collection. Like most other concurrent collection implementations, this class does not permit the use of null elements.

Iterators are *weakly consistent*, returning elements reflecting the state of the deque at some point at or since the creation of the iterator. They do not throw [ConcurrentModificationException](file:///E:\dev\java7\docs\api\java\util\ConcurrentModificationException.html), and may proceed concurrently with other operations.

Beware that, unlike in most collections, the size method is NOT a constant-time operation. Because of the asynchronous nature of these deques, determining the current number of elements requires a traversal of the elements, and so may report inaccurate results if this collection is modified during traversal. Additionally, the bulk operations addAll, removeAll, retainAll, containsAll, equals, andtoArray are not guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might view only some of the added elements.

This class and its iterator implement all of the optional methods of the [Deque](file:///E:\dev\java7\docs\api\java\util\Deque.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a ConcurrentLinkedDeque [*happen-before*](file:///E:\dev\java7\docs\api\java\util\concurrent\package-summary.html#MemoryVisibility) actions subsequent to the access or removal of that element from the ConcurrentLinkedDeque in another thread.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the tail of this deque. |
| boolean | [**addAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#addAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? extends [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html)> c)  Appends all of the elements in the specified collection to the end of this deque, in the order that they are returned by the specified collection's iterator. |
| void | [**addFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#addFirst(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the front of this deque. |
| void | [**addLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#addLast(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the end of this deque. |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#clear())()  Removes all of the elements from this deque. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this deque contains at least one element e such that o.equals(e). |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html)> | [**descendingIterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#descendingIterator())()  Returns an iterator over the elements in this deque in reverse sequential order. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**element**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#element())()  Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque). |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**getFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#getFirst())()  Retrieves, but does not remove, the first element of this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**getLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#getLast())()  Retrieves, but does not remove, the last element of this deque. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#isEmpty())()  Returns true if this collection contains no elements. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#iterator())()  Returns an iterator over the elements in this deque in proper sequence. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the tail of this deque. |
| boolean | [**offerFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#offerFirst(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the front of this deque. |
| boolean | [**offerLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#offerLast(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Inserts the specified element at the end of this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#peek())()  Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque), or returns nullif this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**peekFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#peekFirst())()  Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**peekLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#peekLast())()  Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#poll())()  Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**pollFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#pollFirst())()  Retrieves and removes the first element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**pollLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#pollLast())()  Retrieves and removes the last element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**pop**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#pop())()  Pops an element from the stack represented by this deque. |
| void | [**push**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#push(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) e)  Pushes an element onto the stack represented by this deque (in other words, at the head of this deque) if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#remove())()  Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque). |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the first element e such that o.equals(e), if such an element exists in this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**removeFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#removeFirst())()  Retrieves and removes the first element of this deque. |
| boolean | [**removeFirstOccurrence**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#removeFirstOccurrence(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the first element e such that o.equals(e), if such an element exists in this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html) | [**removeLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#removeLast())()  Retrieves and removes the last element of this deque. |
| boolean | [**removeLastOccurrence**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#removeLastOccurrence(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the last element e such that o.equals(e), if such an element exists in this deque. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#size())()  Returns the number of elements in this deque. |
| [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#toArray())()  Returns an array containing all of the elements in this deque, in proper sequence (from first to last element). |
| <T> T[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentLinkedDeque.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this deque, in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array. |

Example is given below.

import java.util.concurrent.ConcurrentLinkedDeque;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class ConcurrentLinkedDequeDemo {

static ConcurrentLinkedDeque<String> cld=new ConcurrentLinkedDeque<String>();

public static void main(String[] args) {

ExecutorService exService = Executors.newFixedThreadPool(2);

ThreadOne elementAdd = new ConcurrentLinkedDequeDemo().new ThreadOne();

ThreadTwo elementGet = new ConcurrentLinkedDequeDemo().new ThreadTwo();

exService.execute(elementAdd);

exService.execute(elementGet);

exService.shutdown();

}

class ThreadOne implements Runnable{

@Override

public void run() {

for(int i=0;i<5;i++){

cld.add("A"+i);

}

}

}

class ThreadTwo implements Runnable{

@Override

public void run() {

for(int i=0;i<5;i++){

String s= cld.poll();

System.out.println("Element received is: "+s);

}

}

}

}

## Class ConcurrentSkipListMap<K,V>

A scalable concurrent [ConcurrentNavigableMap](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html) implementation. The map is sorted according to the [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its keys, or by a [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html) provided at map creation time, depending on which constructor is used.

This class implements a concurrent variant of [SkipLists](http://en.wikipedia.org/wiki/Skip_list) providing expected average *log(n)* time cost for the containsKey, get, put and remove operations and their variants. Insertion, removal, update, and access operations safely execute concurrently by multiple threads. Iterators are *weakly consistent*, returning elements reflecting the state of the map at some point at or since the creation of the iterator. They do not throw [ConcurrentModificationException](file:///E:\dev\java7\docs\api\java\util\ConcurrentModificationException.html), and may proceed concurrently with other operations. Ascending key ordered views and their iterators are faster than descending ones.

All Map.Entry pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do not support the Entry.setValue method. (Note however that it is possible to change mappings in the associated map using put, putIfAbsent, or replace, depending on exactly which effect you need.)

Beware that, unlike in most collections, the size method is not a constant-time operation. Because of the asynchronous nature of these maps, determining the current number of elements requires a traversal of the elements, and so may report inaccurate results if this collection is modified during traversal. Additionally, the bulk operations putAll, equals, toArray, containsValue, and clear are notguaranteed to be performed atomically. For example, an iterator operating concurrently with a putAll operation might view only some of the added elements.

This class and its views and iterators implement all of the optional methods of the [Map](file:///E:\dev\java7\docs\api\java\util\Map.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces. Like most other concurrent collections, this class does not permit the use of null keys or values because some null return values cannot be reliably distinguished from the absence of elements.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**ceilingEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#ceilingEntry(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns a key-value mapping associated with the least key greater than or equal to the given key, or null if there is no such entry. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**ceilingKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#ceilingKey(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns the least key greater than or equal to the given key, or null if there is no such key. |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#clear())()  Removes all of the mappings from this map. |
| [**ConcurrentSkipListMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**clone**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#clone())()  Returns a shallow copy of this ConcurrentSkipListMap instance. |
| [**Comparator**](file:///E:\dev\java7\docs\api\java\util\Comparator.html)<? super [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**comparator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#comparator())()  Returns the comparator used to order the keys in this map, or null if this map uses the [**natural ordering**](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its keys. |
| boolean | [**containsKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#containsKey(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Returns true if this map contains a mapping for the specified key. |
| boolean | [**containsValue**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#containsValue(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Returns true if this map maps one or more keys to the specified value. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**descendingKeySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#descendingKeySet())()  Returns a reverse order [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) view of the keys contained in this map. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**descendingMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#descendingMap())()  Returns a reverse order view of the mappings contained in this map. |
| [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html)<[**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)>> | [**entrySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#entrySet())()  Returns a [**Set**](file:///E:\dev\java7\docs\api\java\util\Set.html) view of the mappings contained in this map. |
| boolean | [**equals**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#equals(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Compares the specified object with this map for equality. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**firstEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#firstEntry())()  Returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**firstKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#firstKey())()  Returns the first (lowest) key currently in this map. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**floorEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#floorEntry(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns a key-value mapping associated with the greatest key less than or equal to the given key, or null if there is no such key. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**floorKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#floorKey(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns the greatest key less than or equal to the given key, or null if there is no such key. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**get**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#get(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**headMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#headMap(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) toKey)  Returns a view of the portion of this map whose keys are strictly less than toKey. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**headMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#headMap(K,%20boolean))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) toKey, boolean inclusive)  Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**higherEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#higherEntry(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns a key-value mapping associated with the least key strictly greater than the given key, or null if there is no such key. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**higherKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#higherKey(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns the least key strictly greater than the given key, or null if there is no such key. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#isEmpty())()  Returns true if this map contains no key-value mappings. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**keySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#keySet())()  Returns a [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) view of the keys contained in this map. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**lastEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#lastEntry())()  Returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**lastKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#lastKey())()  Returns the last (highest) key currently in this map. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**lowerEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#lowerEntry(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key. |
| [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**lowerKey**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#lowerKey(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key)  Returns the greatest key strictly less than the given key, or null if there is no such key. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**navigableKeySet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#navigableKeySet())()  Returns a [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) view of the keys contained in this map. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**pollFirstEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#pollFirstEntry())()  Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty. |
| [**Map.Entry**](file:///E:\dev\java7\docs\api\java\util\Map.Entry.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**pollLastEntry**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#pollLastEntry())()  Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#put(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) value)  Associates the specified value with the specified key in this map. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**putIfAbsent**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#putIfAbsent(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) value)  If the specified key is not already associated with a value, associate it with the given value. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key)  Removes the mapping for the specified key from this map if present. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#remove(java.lang.Object,%20java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) key, [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) value)  Removes the entry for a key only if currently mapped to a given value. |
| [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) | [**replace**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#replace(K,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) value)  Replaces the entry for a key only if currently mapped to some value. |
| boolean | [**replace**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#replace(K,%20V,%20V))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) key, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) oldValue, [**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) newValue)  Replaces the entry for a key only if currently mapped to a given value. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#size())()  Returns the number of key-value mappings in this map. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**subMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#subMap(K,%20boolean,%20K,%20boolean))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) fromKey, boolean fromInclusive, [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) toKey, boolean toInclusive)  Returns a view of the portion of this map whose keys range from fromKey to toKey. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**subMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#subMap(K,%20K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) fromKey, [**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) toKey)  Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**tailMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#tailMap(K))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) fromKey)  Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |
| [**ConcurrentNavigableMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentNavigableMap.html)<[**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html),[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**tailMap**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#tailMap(K,%20boolean))([**K**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html) fromKey, boolean inclusive)  Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey. |
| [**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<[**V**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html)> | [**values**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html#values())()  Returns a [**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html) view of the values contained in this map. |

Example is given below

import java.util.Iterator;

import java.util.NavigableSet;

import java.util.concurrent.ConcurrentNavigableMap;

import java.util.concurrent.ConcurrentSkipListMap;

public class ConcurrentSkipListMapExample {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

ConcurrentSkipListMap<String,String> concurrentSkipListMap = new ConcurrentSkipListMap<String,String>();

concurrentSkipListMap.put("1111", "Tom Smith");

concurrentSkipListMap.put("2222","David Jones");

concurrentSkipListMap.put("3333", "Jim Anderson");

concurrentSkipListMap.put("4444", "John Abraham");

concurrentSkipListMap.put("5555", "Brad Pitt");

System.out.println("The name associated with id 1111 is "+ concurrentSkipListMap.get("1111"));

NavigableSet navigableKeySet = concurrentSkipListMap.keySet();

System.out.println("The keys associated with this map are ");

for(Iterator iterator = navigableKeySet.iterator();iterator.hasNext();)

{

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

}

ConcurrentNavigableMap<String,String> subMap = concurrentSkipListMap.subMap("1111", "3333");

NavigableSet navigableSubKeySet = subMap.keySet();

System.out.println("The keys associated with the submap keys from 1111 to 3333 are");

for(Iterator subMapIterator = navigableSubKeySet.iterator(); subMapIterator.hasNext();)

{

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

}

ConcurrentNavigableMap<String,String> headerMap = concurrentSkipListMap.headMap("2222");

System.out.println("The keys associated with the headMap less than 2222");

NavigableSet navigableHeadMapKeySet = headerMap.keySet();

for(Iterator headMapIterator = navigableHeadMapKeySet.iterator(); headMapIterator.hasNext();)

{

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

}

ConcurrentNavigableMap<String,String> tailMap = concurrentSkipListMap.tailMap("1111");

System.out.println("Thekeys associated with the tailMap less than 1111");

NavigableSet navigableTailMapKeySet = tailMap.keySet();

for(Iterator tailMapIterator = navigableTailMapKeySet.iterator(); tailMapIterator.hasNext();)

{

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

}

}

}

## Class ConcurrentSkipListSet<E>

A scalable concurrent [NavigableSet](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html) implementation based on a [ConcurrentSkipListMap](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListMap.html). The elements of the set are kept sorted according to their [natural ordering](file:///E:\dev\java7\docs\api\java\lang\Comparable.html), or by a [Comparator](file:///E:\dev\java7\docs\api\java\util\Comparator.html) provided at set creation time, depending on which constructor is used.

This implementation provides expected average *log(n)* time cost for the contains, add, and remove operations and their variants. Insertion, removal, and access operations safely execute concurrently by multiple threads. Iterators are *weakly consistent*, returning elements reflecting the state of the set at some point at or since the creation of the iterator. They do not throw[ConcurrentModificationException](file:///E:\dev\java7\docs\api\java\util\ConcurrentModificationException.html), and may proceed concurrently with other operations. Ascending ordered views and their iterators are faster than descending ones. Beware that, unlike in most collections, the size method is not a constant-time operation. Because of the asynchronous nature of these sets, determining the current number of elements requires a traversal of the elements, and so may report inaccurate results if this collection is modified during traversal. Additionally, the bulk operations addAll, removeAll, retainAll, containsAll, equals, andtoArray are not guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might view only some of the added elements. This class and its iterators implement all of the optional methods of the [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces. Like most other concurrent collection implementations, this class does not permit the use of nullelements, because null arguments and return values cannot be reliably distinguished from the absence of elements.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) e)  Adds the specified element to this set if it is not already present. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**ceiling**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#ceiling(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) e)  Returns the least element in this set greater than or equal to the given element, or null if there is no such element. |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#clear())()  Removes all of the elements from this set. |
| [**ConcurrentSkipListSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**clone**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#clone())()  Returns a shallow copy of this ConcurrentSkipListSet instance. |
| [**Comparator**](file:///E:\dev\java7\docs\api\java\util\Comparator.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**comparator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#comparator())()  Returns the comparator used to order the elements in this set, or null if this set uses the [**natural ordering**](file:///E:\dev\java7\docs\api\java\lang\Comparable.html) of its elements. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this set contains the specified element. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**descendingIterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#descendingIterator())()  Returns an iterator over the elements in this set in descending order. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**descendingSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#descendingSet())()  Returns a reverse order view of the elements contained in this set. |
| boolean | [**equals**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#equals(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Compares the specified object with this set for equality. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**first**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#first())()  Returns the first (lowest) element currently in this set. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**floor**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#floor(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) e)  Returns the greatest element in this set less than or equal to the given element, or null if there is no such element. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**headSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#headSet(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) toElement)  Returns a view of the portion of this set whose elements are strictly less than toElement. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**headSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#headSet(E,%20boolean))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) toElement, boolean inclusive)  Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**higher**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#higher(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) e)  Returns the least element in this set strictly greater than the given element, or null if there is no such element. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#isEmpty())()  Returns true if this set contains no elements. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#iterator())()  Returns an iterator over the elements in this set in ascending order. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**last**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#last())()  Returns the last (highest) element currently in this set. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**lower**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#lower(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) e)  Returns the greatest element in this set strictly less than the given element, or null if there is no such element. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**pollFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#pollFirst())()  Retrieves and removes the first (lowest) element, or returns null if this set is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) | [**pollLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#pollLast())()  Retrieves and removes the last (highest) element, or returns null if this set is empty. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the specified element from this set if it is present. |
| boolean | [**removeAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#removeAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Removes from this set all of its elements that are contained in the specified collection. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#size())()  Returns the number of elements in this set. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**subSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#subSet(E,%20boolean,%20E,%20boolean))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) fromElement, boolean fromInclusive, [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) toElement, boolean toInclusive)  Returns a view of the portion of this set whose elements range from fromElement to toElement. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**subSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#subSet(E,%20E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) fromElement, [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) toElement)  Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**tailSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#tailSet(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) fromElement)  Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |
| [**NavigableSet**](file:///E:\dev\java7\docs\api\java\util\NavigableSet.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html)> | [**tailSet**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html#tailSet(E,%20boolean))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\ConcurrentSkipListSet.html) fromElement, boolean inclusive)  Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement. |

## Class CopyOnWriteArrayList<E>

A thread-safe variant of [ArrayList](file:///E:\dev\java7\docs\api\java\util\ArrayList.html) in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array.

This is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don't want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException. The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported. These methods throw UnsupportedOperationException.

All elements are permitted, including null.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a CopyOnWriteArrayList [*happen-before*](file:///E:\dev\java7\docs\api\java\util\concurrent\package-summary.html#MemoryVisibility) actions subsequent to the access or removal of that element from the CopyOnWriteArrayList in another thread.

Code Snippet for the internals of CopyOnWriteArrayList.

public class CopyOnWriteArrayList<E>

implements List<E>, RandomAccess, Cloneable, java.io.Serializable {

private static final long serialVersionUID = 8673264195747942595L;

/\*\* The lock protecting all mutators \*/

**transient final ReentrantLock lock = new ReentrantLock();**

/\*\* The array, accessed only via getArray/setArray. \*/

**private volatile transient Object[] array;**

public boolean add(E e) {

**final ReentrantLock lock = this.lock;**

**lock.lock();**

try {

Object[] elements = getArray();

int len = elements.length;

Object[] newElements = Arrays.copyOf(elements, len + 1);

newElements[len] = e;

setArray(newElements);

return true;

} **finally {**

**lock.unlock();**

**}**

}

public E remove(int index) {

**final ReentrantLock lock = this.lock;**

**lock.lock();**

try {

Object[] elements = getArray();

int len = elements.length;

E oldValue = get(elements, index);

int numMoved = len - index - 1;

if (numMoved == 0)

setArray(Arrays.copyOf(elements, len - 1));

else {

Object[] newElements = new Object[len - 1];

System.arraycopy(elements, 0, newElements, 0, index);

System.arraycopy(elements, index + 1, newElements, index,

numMoved);

setArray(newElements);

}

return oldValue;

} **finally {**

**lock.unlock();**

**}**

}

## Class CopyOnWriteArraySet<E>

A [Set](file:///E:\dev\java7\docs\api\java\util\Set.html) that uses an internal [CopyOnWriteArrayList](file:///E:\dev\java7\docs\api\java\util\concurrent\CopyOnWriteArrayList.html) for all of its operations. Thus, it shares the same basic properties:

* It is best suited for applications in which set sizes generally stay small, read-only operations vastly outnumber mutative operations, and you need to prevent interference among threads during traversal.
* It is thread-safe.
* Mutative operations (add, set, remove, etc.) are expensive since they usually entail copying the entire underlying array.
* Iterators do not support the mutative remove operation.
* Traversal via iterators is fast and cannot encounter interference from other threads. Iterators rely on unchanging snapshots of the array at the time the iterators were constructed.

The internal code snippet for CopyOnWriteArraySet is given below.

**public class CopyOnWriteArraySet<E> extends AbstractSet<E>**

**implements java.io.Serializable {**

**private static final long serialVersionUID = 5457747651344034263L;**

**private final CopyOnWriteArrayList<E> al;**

**public boolean add(E e) {**

**return al.addIfAbsent(e);**

**}**

## Class DelayQueue<E extends [Delayed](file:///E:\dev\java7\docs\api\java\util\concurrent\Delayed.html)>

An unbounded [blocking queue](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html) of Delayed elements, in which an element can only be taken when its delay has expired. The head of the queue is that Delayed element whose delay expired furthest in the past. If no delay has expired there is no head and poll will return null. Expiration occurs when an element's getDelay(TimeUnit.NANOSECONDS) method returns a value less than or equal to zero. Even though unexpired elements cannot be removed using take or poll, they are otherwise treated as normal elements. For example, the size method returns the count of both expired and unexpired elements. This queue does not permit null elements.

This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

**Usage of DelayQueue**

<http://www.javacodegeeks.com/2012/04/using-delayed-queues-in-practice.html>

Often there are use cases when you have some kind of work or job queue and there is a need not to handle each work item or job immediately but with some delay. For example, if user clicks a button which triggers some work to be done, and one second later user realizes he / she was mistaken and job shouldn’t start at all. Or, f.e. there could be a use case when some work elements in a queue should be removed after some delay (expiration).

There are a lot of implementations out there, but one I would like to describe is using pure JDK concurrent framework classes: [DelayedQueue](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/DelayQueue.html)and [Delayed](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Delayed.html) interface.

Let me start with simple (and empty) interface which defines the work item. I am skipping the implementation details like properties and methods as those are not important.

package com.example.delayed;

public interface WorkItem {

// Some properties and methods here

}

The next class in our model will represent the postponed work item and implement [Delayed](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Delayed.html) interface. There are just few basic concepts to take into account: the delay itself and the actual time the respective work item has been submitted. This is how expiration would be calculated. So let’s do that by introducing **PostponedWorkItem** class.

package com.example.delayed;

import java.util.concurrent.Delayed;

import java.util.concurrent.TimeUnit;

public class PostponedWorkItem implements Delayed {

private final long origin;

private final long delay;

private final WorkItem workItem;

public PostponedWorkItem( final WorkItem workItem, final long delay ) {

this.origin = System.currentTimeMillis();

this.workItem = workItem;

this.delay = delay;

}

@Override

public long getDelay( TimeUnit unit ) {

return unit.convert( delay - ( System.currentTimeMillis() - origin ),

TimeUnit.MILLISECONDS );

}

@Override

public int compareTo( Delayed delayed ) {

if( delayed == this ) {

return 0;

}

if( delayed instanceof PostponedWorkItem ) {

long diff = delay - ( ( PostponedWorkItem )delayed ).delay;

return ( ( diff == 0 ) ? 0 : ( ( diff < 0 ) ? -1 : 1 ) );

}

long d = (getDelay( TimeUnit.MILLISECONDS)-delayed.getDelay(TimeUnit.MILLISECONDS));

return ( ( d == 0 ) ? 0 : ( ( d < 0 ) ? -1 : 1 ) );

}

}

As you can see, we create new instance of the class and save the current system time in internal **origin** property. The**getDelayed** method calculates the actual time left before work item gets expired. The **delay** is external setting which comes as constructor parameter. The mandatory implementation of [Comparable<Delayed>](http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Comparable.html) is required as [Delayed](http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Delayed.html) extends this interface.

Now, we are mostly done! To complete the example, let’s make sure that same work item won’t be submitted twice to the work queue by implementing **equals** and **hashCode** (implemenation is pretty trivial and should not require any comments).

public class PostponedWorkItem implements Delayed {

...

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ( ( workItem == null ) ? 0 : workItem.hashCode() );

return result;

}

@Override

public boolean equals( Object obj ) {

if( this == obj ) {

return true;

}

if( obj == null ) {

return false;

}

if( !( obj instanceof PostponedWorkItem ) ) {

return false;

}

final PostponedWorkItem other = ( PostponedWorkItem )obj;

if( workItem == null ) {

if( other.workItem != null ) {

return false;

}

} else if( !workItem.equals( other.workItem ) ) {

return false;

}

return true;

}

}

The last step is to introduce some kind of manager which will scheduled work items and periodically polls out expired ones: meet **WorkItemScheduler** class.

package com.example.delayed;

import java.util.ArrayList;

import java.util.Collection;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.DelayQueue;

public class WorkItemScheduler {

private final long delay = 2000; // 2 seconds

private final BlockingQueue< PostponedWorkItem > delayed =

new DelayQueue< PostponedWorkItem >();

public void addWorkItem( final WorkItem workItem ) {

final PostponedWorkItem postponed = new PostponedWorkItem( workItem, delay );

if( !delayed.contains( postponed )) {

delayed.offer( postponed );

}

}

public void process() {

final Collection< PostponedWorkItem > expired = new ArrayList< PostponedWorkItem >();

delayed.drainTo( expired );

for( final PostponedWorkItem postponed: expired ) {

// Do some real work here with postponed.getWorkItem()

}

}

}

Usage of **BlockingQueue** guarantees thread safety and high level of concurrency. The **process** method should be run periodically in order to drain work items queue. It could be annotated by @ **Scheduled** annotation from [Spring Framework](http://www.springsource.org/)or by EJB’s **@Schedule** annotation from [JEE 6](http://docs.oracle.com/javaee/6/api/javax/ejb/Schedule.html).

## Class LinkedBlockingDeque<E>

An optionally-bounded [blocking deque](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingDeque.html) based on linked nodes.

The optional capacity bound constructor argument serves as a way to prevent excessive expansion. The capacity, if unspecified, is equal to [Integer.MAX\_VALUE](file:///E:\dev\java7\docs\api\java\lang\Integer.html#MAX_VALUE). Linked nodes are dynamically created upon each insertion unless this would bring the deque above capacity.

Most operations run in constant time (ignoring time spent blocking). Exceptions include [remove](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#remove(java.lang.Object)), [removeFirstOccurrence](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeFirstOccurrence(java.lang.Object)), [removeLastOccurrence](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeLastOccurrence(java.lang.Object)), [contains](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#contains(java.lang.Object)), [iterator.remove()](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#iterator()), and the bulk operations, all of which run in linear time.

This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the end of this deque unless it would violate capacity restrictions. |
| void | [**addFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#addFirst(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the front of this deque if it is possible to do so immediately without violating capacity restrictions, throwing anIllegalStateException if no space is currently available. |
| void | [**addLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#addLast(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the end of this deque if it is possible to do so immediately without violating capacity restrictions, throwing anIllegalStateException if no space is currently available. |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#clear())()  Atomically removes all of the elements from this deque. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this deque contains the specified element. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)> | [**descendingIterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#descendingIterator())()  Returns an iterator over the elements in this deque in reverse sequential order. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#drainTo(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)> c)  Removes all available elements from this queue and adds them to the given collection. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#drainTo(java.util.Collection,%20int))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)> c, int maxElements)  Removes at most the given number of available elements from this queue and adds them to the given collection. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**element**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#element())()  Retrieves, but does not remove, the head of the queue represented by this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**getFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#getFirst())()  Retrieves, but does not remove, the first element of this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**getLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#getLast())()  Retrieves, but does not remove, the last element of this deque. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#iterator())()  Returns an iterator over the elements in this deque in proper sequence. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque) if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque), waiting up to the specified wait time if necessary for space to become available. |
| boolean | [**offerFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offerFirst(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the front of this deque if it is possible to do so immediately without violating capacity restrictions, returning trueupon success and false if no space is currently available. |
| boolean | [**offerFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offerFirst(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element at the front of this deque, waiting up to the specified wait time if necessary for space to become available. |
| boolean | [**offerLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offerLast(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the end of this deque if it is possible to do so immediately without violating capacity restrictions, returning trueupon success and false if no space is currently available. |
| boolean | [**offerLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#offerLast(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element at the end of this deque, waiting up to the specified wait time if necessary for space to become available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#peek())()  Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque), or returns nullif this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**peekFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#peekFirst())()  Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**peekLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#peekLast())()  Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#poll())()  Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), waiting up to the specified wait time if necessary for an element to become available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**pollFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#pollFirst())()  Retrieves and removes the first element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**pollFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#pollFirst(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the first element of this deque, waiting up to the specified wait time if necessary for an element to become available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**pollLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#pollLast())()  Retrieves and removes the last element of this deque, or returns null if this deque is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**pollLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#pollLast(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the last element of this deque, waiting up to the specified wait time if necessary for an element to become available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**pop**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#pop())()  Pops an element from the stack represented by this deque. |
| void | [**push**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#push(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Pushes an element onto the stack represented by this deque. |
| void | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#put(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque), waiting if necessary for space to become available. |
| void | [**putFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#putFirst(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the front of this deque, waiting if necessary for space to become available. |
| void | [**putLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#putLast(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) e)  Inserts the specified element at the end of this deque, waiting if necessary for space to become available. |
| int | [**remainingCapacity**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#remainingCapacity())()  Returns the number of additional elements that this deque can ideally (in the absence of memory or resource constraints) accept without blocking. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#remove())()  Retrieves and removes the head of the queue represented by this deque. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the first occurrence of the specified element from this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**removeFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeFirst())()  Retrieves and removes the first element of this deque. |
| boolean | [**removeFirstOccurrence**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeFirstOccurrence(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the first occurrence of the specified element from this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**removeLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeLast())()  Retrieves and removes the last element of this deque. |
| boolean | [**removeLastOccurrence**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#removeLastOccurrence(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes the last occurrence of the specified element from this deque. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#size())()  Returns the number of elements in this deque. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**take**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#take())()  Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), waiting if necessary until an element becomes available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**takeFirst**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#takeFirst())()  Retrieves and removes the first element of this deque, waiting if necessary until an element becomes available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html) | [**takeLast**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#takeLast())()  Retrieves and removes the last element of this deque, waiting if necessary until an element becomes available. |
| [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#toArray())()  Returns an array containing all of the elements in this deque, in proper sequence (from first to last element). |
| <T> T[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this deque, in proper sequence; the runtime type of the returned array is that of the specified array. |
| [**String**](file:///E:\dev\java7\docs\api\java\lang\String.html) | [**toString**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingDeque.html#toString())()  Returns a string representation of this collection. |

ConcurentLinkedDequeue is lock-free (see comments in source code) while LinkedBlockingQueue uses locking. That is the former is supposed to be more efficient

Example is given below.

LinkedBlockingDeque lbd=new LinkedBlockingDeque();

lbd.add("B");

lbd.add("A");

lbd.add("C");

lbd.add("D");

show(lbd.iterator(),"Data after adding.");

lbd.addFirst("E");

lbd.addLast("F");

show(lbd.iterator(),"After using addFirst and addLast");

List list=new ArrayList();

lbd.drainTo(list,2);

## Class LinkedBlockingQueue<E>

An optionally-bounded [blocking queue](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html) based on linked nodes. This queue orders elements FIFO (first-in-first-out). The head of the queue is that element that has been on the queue the longest time. The tail of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. Linked queues typically have higher throughput than array-based queues but less predictable performance in most concurrent applications. The optional capacity bound constructor argument serves as a way to prevent excessive queue expansion. The capacity, if unspecified, is equal to [Integer.MAX\_VALUE](file:///E:\dev\java7\docs\api\java\lang\Integer.html#MAX_VALUE). Linked nodes are dynamically created upon each insertion unless this would bring the queue above capacity. This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#clear())()  Atomically removes all of the elements from this queue. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this queue contains the specified element. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#drainTo(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html)> c)  Removes all available elements from this queue and adds them to the given collection. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#drainTo(java.util.Collection,%20int))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html)> c, int maxElements)  Removes at most the given number of available elements from this queue and adds them to the given collection. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#iterator())()  Returns an iterator over the elements in this queue in proper sequence. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) e)  Inserts the specified element at the tail of this queue if it is possible to do so immediately without exceeding the queue's capacity, returning trueupon success and false if this queue is full. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element at the tail of this queue, waiting if necessary up to the specified wait time for space to become available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#peek())()  Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#poll())()  Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available. |
| void | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#put(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) e)  Inserts the specified element at the tail of this queue, waiting if necessary for space to become available. |
| int | [**remainingCapacity**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#remainingCapacity())()  Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes a single instance of the specified element from this queue, if it is present. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#size())()  Returns the number of elements in this queue. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html) | [**take**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#take())()  Retrieves and removes the head of this queue, waiting if necessary until an element becomes available. |
| [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#toArray())()  Returns an array containing all of the elements in this queue, in proper sequence. |
| <T> T[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#toArray(T[]))(T[] a)  Returns an array containing all of the elements in this queue, in proper sequence; the runtime type of the returned array is that of the specified array. |
| [**String**](file:///E:\dev\java7\docs\api\java\lang\String.html) | [**toString**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedBlockingQueue.html#toString())()  Returns a string representation of this collection. |

## Class LinkedTransferQueue<E>

An unbounded [TransferQueue](file:///E:\dev\java7\docs\api\java\util\concurrent\TransferQueue.html) based on linked nodes. This queue orders elements FIFO (first-in-first-out) with respect to any given producer. The head of the queue is that element that has been on the queue the longest time for some producer. The tail of the queue is that element that has been on the queue the shortest time for some producer.

Beware that, unlike in most collections, the size method is NOT a constant-time operation. Because of the asynchronous nature of these queues, determining the current number of elements requires a traversal of the elements, and so may report inaccurate results if this collection is modified during traversal. Additionally, the bulk operations addAll, removeAll, retainAll, containsAll, equals, andtoArray are not guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might view only some of the added elements.

This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a LinkedTransferQueue [*happen-before*](file:///E:\dev\java7\docs\api\java\util\concurrent\package-summary.html#MemoryVisibility) actions subsequent to the access or removal of that element from the LinkedTransferQueue in another thread.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| boolean | [**add**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#add(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e)  Inserts the specified element at the tail of this queue. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Returns true if this queue contains the specified element. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#drainTo(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html)> c)  Removes all available elements from this queue and adds them to the given collection. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#drainTo(java.util.Collection,%20int))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html)> c, int maxElements)  Removes at most the given number of available elements from this queue and adds them to the given collection. |
| int | [**getWaitingConsumerCount**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#getWaitingConsumerCount())()  Returns an estimate of the number of consumers waiting to receive elements via [**BlockingQueue.take()**](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html#take()) or timed [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit)). |
| boolean | [**hasWaitingConsumer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#hasWaitingConsumer())()  Returns true if there is at least one consumer waiting to receive an element via [**BlockingQueue.take()**](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html#take()) or timed [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html#poll(long,%20java.util.concurrent.TimeUnit)). |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#isEmpty())()  Returns true if this queue contains no elements. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#iterator())()  Returns an iterator over the elements in this queue in proper sequence. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e)  Inserts the specified element at the tail of this queue. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element at the tail of this queue. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#peek())()  Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#poll())()  Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available. |
| void | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#put(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e)  Inserts the specified element at the tail of this queue. |
| int | [**remainingCapacity**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#remainingCapacity())()  Always returns Integer.MAX\_VALUE because a LinkedTransferQueue is not capacity constrained. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Removes a single instance of the specified element from this queue, if it is present. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#size())()  Returns the number of elements in this queue. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) | [**take**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#take())()  Retrieves and removes the head of this queue, waiting if necessary until an element becomes available. |
| void | [**transfer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#transfer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e)  Transfers the element to a consumer, waiting if necessary to do so. |
| boolean | [**tryTransfer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#tryTransfer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e)  Transfers the element to a waiting consumer immediately, if possible. |
| boolean | [**tryTransfer**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html#tryTransfer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\LinkedTransferQueue.html) e, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Transfers the element to a consumer if it is possible to do so before the timeout elapses. |

## Class SynchronousQueue<E>

A [blocking queue](file:///E:\dev\java7\docs\api\java\util\concurrent\BlockingQueue.html) in which each insert operation must wait for a corresponding remove operation by another thread, and vice versa. A synchronous queue does not have any internal capacity, not even a capacity of one. You cannot peek at a synchronous queue because an element is only present when you try to remove it; you cannot insert an element (using any method) unless another thread is trying to remove it; you cannot iterate as there is nothing to iterate. The head of the queue is the element that the first queued inserting thread is trying to add to the queue; if there is no such queued thread then no element is available for removal and poll() will return null. For purposes of other Collection methods (for example contains), a SynchronousQueue acts as an empty collection. This queue does not permit null elements. Synchronous queues are similar to rendezvous channels used in CSP and Ada. They are well suited for handoff designs, in which an object running in one thread must sync up with an object running in another thread in order to hand it some information, event, or task. This class supports an optional fairness policy for ordering waiting producer and consumer threads. By default, this ordering is not guaranteed. However, a queue constructed with fairness set to truegrants threads access in FIFO order. This class and its iterator implement all of the optional methods of the [Collection](file:///E:\dev\java7\docs\api\java\util\Collection.html) and [Iterator](file:///E:\dev\java7\docs\api\java\util\Iterator.html) interfaces.

### Method Summary

|  |  |
| --- | --- |
| Modifier and Type | Method and Description |
| void | [**clear**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#clear())()  Does nothing. |
| boolean | [**contains**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#contains(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Always returns false. |
| boolean | [**containsAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#containsAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Returns false unless the given collection is empty. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#drainTo(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html)> c)  Removes all available elements from this queue and adds them to the given collection. |
| int | [**drainTo**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#drainTo(java.util.Collection,%20int))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<? super [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html)> c, int maxElements)  Removes at most the given number of available elements from this queue and adds them to the given collection. |
| boolean | [**isEmpty**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#isEmpty())()  Always returns true. |
| [**Iterator**](file:///E:\dev\java7\docs\api\java\util\Iterator.html)<[**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html)> | [**iterator**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#iterator())()  Returns an empty iterator in which hasNext always returns false. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#offer(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) e)  Inserts the specified element into this queue, if another thread is waiting to receive it. |
| boolean | [**offer**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#offer(E,%20long,%20java.util.concurrent.TimeUnit))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) o, long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Inserts the specified element into this queue, waiting if necessary up to the specified wait time for another thread to receive it. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) | [**peek**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#peek())()  Always returns null. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#poll())()  Retrieves and removes the head of this queue, if another thread is currently making an element available. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) | [**poll**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#poll(long,%20java.util.concurrent.TimeUnit))(long timeout, [**TimeUnit**](file:///E:\dev\java7\docs\api\java\util\concurrent\TimeUnit.html) unit)  Retrieves and removes the head of this queue, waiting if necessary up to the specified wait time, for another thread to insert it. |
| void | [**put**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#put(E))([**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) o)  Adds the specified element to this queue, waiting if necessary for another thread to receive it. |
| int | [**remainingCapacity**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#remainingCapacity())()  Always returns zero. |
| boolean | [**remove**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#remove(java.lang.Object))([**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html) o)  Always returns false. |
| boolean | [**removeAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#removeAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Always returns false. |
| boolean | [**retainAll**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#retainAll(java.util.Collection))([**Collection**](file:///E:\dev\java7\docs\api\java\util\Collection.html)<?> c)  Always returns false. |
| int | [**size**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#size())()  Always returns zero. |
| [**E**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html) | [**take**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#take())()  Retrieves and removes the head of this queue, waiting if necessary for another thread to insert it. |
| [**Object**](file:///E:\dev\java7\docs\api\java\lang\Object.html)[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#toArray())()  Returns a zero-length array. |
| <T> T[] | [**toArray**](file:///E:\dev\java7\docs\api\java\util\concurrent\SynchronousQueue.html#toArray(T[]))(T[] a)  Sets the zeroeth element of the specified array to null (if the array has non-zero length) and returns it. |

**Why not to use synchronized keyword**

[**http://stackoverflow.com/questions/8521819/performance-of-synchronize-section-in-java**](http://stackoverflow.com/questions/8521819/performance-of-synchronize-section-in-java)

Single-threaded code will still run slower when using synchronized blocks. Obviously you will not have other threads stalled while waiting for other threads to finish, however you will have to deal with the other effects of synchronization, namely cache coherency. Synchronized blocks are not only used for **concurrency**, but also **visibility**. Every synchronized block is a memory barrier: the JVM is free to work on variables in registers, instead of main memory, on the assumption that multiple threads will not access that variable. Without synchronization blocks, this data could be stored in a CPU's cache and different threads on different CPUs would not see the same data. By using a synchronization block, you force the JVM to write this data to main memory for visibility to other threads. So even though you're free from lock contention, the JVM will still have to do housekeeping in flushing data to main memory. In addition, this has optimization constraints. The JVM is free to reorder instructions in order to provide optimization: consider a simple example:

foo++;

bar++;

versus:

foo++;

synchronized(obj)

{

bar++;

}

In the first example, the compiler is free to load foo and bar at the same time, then increment them both, then save them both. In the second example, the compiler **must** perform the load/add/save on foo, then perform the load/add/save on bar. Thus, synchronization may impact the ability of the JRE to optimize instructions.

There are 3 type of locking in HotSpot

1. **Fat**: JVM relies on OS mutexes to acquire lock.
2. **Thin**: JVM is using CAS algorithm.
3. **Biased**: CAS is rather expensive operation on some of the architecture. Biased locking - is special type of locking optimized for scenario when only one thread is working on object.

By default JVM uses **thin** locking. Later if JVM determines that there is no contention thin locking is converted to **biased** locking. Operation that changes type of the lock is rather expensive, hence JVM does not apply this optimization immediately. There is special JVM option -**XX:BiasedLockingStartupDelay=delay** which tells JVM when this kind of optimization should be applied.

Once biased, that thread can subsequently lock and unlock the object without resorting to expensive atomic instructions.

**What is happens-before relationship ?**

 In Java specifically, a **happens**-**before relationship** is a guarantee that memory written to by statement A is visible to statement B, that is, that statement A completes its write **before** statement B starts its read.