java.util.List<E>

» `ListIterator<E> listerOperator()`

returns a list iterator for visiting the elements of the list.

» `ListIterator<E> listerOperator(int index)`

returns a list iterator for visiting the elements of the list whose first call to `next` will return the element with the given index.

» `void add(int i, E element)`

adds an element at the specified position.

» `void addAll(int i, Collection<? extend E> elements)`

adds all elements from a collection to the specified position.

» `E remove(int i)`

removes and returns the element at the specified position.

» `E get(int i)`

gets the element at the specified position.

» `E set(int i)`

replaces the element at the specified position with a new element and returns the old element.

» `int indexOf(Object element)`

returns the position of the last occurence of an element equal to the specified element, or -1 if no matching element is found.

» `int lastIndexOf(Object element)`

returns the position of the last occurrence of an element equal to the specified element, or -1 if no matching element is found.

java.util.ListOperator<E>

» `void add(E newElement)`

adds an element before the current position.

» `void set(E newElement)`

replaces the last element visited by `next` or `previous` with a new element. Throws an `IllegalStateException` if the list structure was modified since the last call to `next` ore `previous`.

» `boolean hasPrevious()`

returns `true` if there is another element to visit when iterating backwards through the list.

» `E previous()`

returns the previous object. Throws a `NoSuchElementException` if the beginning of the list has been reached.

» `int nextIndex()`

java.util.ListOperator<E> (cont)

returns the index of the element that would be returned by the next call to `next`.

» `int previousIndex()`

returns the index of the element that would be returned by the next call to `previous`.

java.util.LinkedList<E>

» `LinkedList()`

constructs an empty linked list.

» `LinkedList(Collection <? extends E> elements)`

constructs a linked list and adds all elements from a collection.

» `void addFirst(E element)`

» `void addLast(E element)`

adds an element to the beginning or the end of the list.

» `E getFirst(E element)`

» `E getLast(E element)`

returns the element at the beginning or the end of the list.

» `E removeFirst(E element)`

» `E removeLast(E element)`

removes and returns the element at the beginning or the end of the list.

java.util.HashSet<E>

» `HashSet()`

constructs an empty hash set.

» `HashSet(Collection<? extends E> elements)`

constructs a hash set and adds all elements from a collection.

» `HashSet(int initialCapacity)`

constructs an empty hash set with the specified capacity (number of buckets).

» `HashSet(int initialCapacity, float loadFactor )`

constructs an empty hash set with the specified capacity and load factor (a number between 0.0 and 1.0 that determines at what percentage of fullness the hash table will be rehashed into a larger one).

java.lang.Object

» `int hashCode()`

returns a hash code for this object. A hash code can be any integer, positive or negative. The definitions of `equals` and `hashCode` must be compatible: if `x.equals(y)` is `true`, then `x.hashCode()` must be the same value as `y.hashCode()`.

java.util.TreeSet<E>

» `TreeSet()`

» `TreeSet(Comparator<? super E> comparator)`

constructs an empty tree set.

» `TreeSet(Collection<? extends E> elements)`

» `TreeSet(SortedSet<E> s)`

constructs a tree set and adds all elements from a collection or sorted set (int the latter case, using the same ordering).

java.util.SortedSet<E>

» `Comparator<? super E> comparator()`

returns the comparator used for sorting the elements, or `null` if the elements are compared with the `compareTo` method of the `Comparable` interface.

» `E first()`

» `E last()`

returns the smallest or largest element in the sorted set.

java.util.NavigableSet<E>

» `E higher(E value)`

» `E lower(E value)`

returns the least element `> value` or the largest element `< value`, or `null` if there is no such element.

» `E ceiling(E value)`

» `E floor(E value)`

returns the least element `>= value` or the largest element `<= value`, or `null` if there is no such element.

» `E pollFirst()`

» `E pollLast()`

removes and returns the smallest or largest element in this set, or `null` if the set is empty.

» `Iterator<E> descendingIterator()`

returns an iterator that traverses this set in descending direction.

java.util.Queue<E>

» `boolean add(E element)`

» `boolean offer(E element)`

adds the given element to the tail of this queue and returns `true`, provided the queue is not full. If the queue is full, the first method throws an `IllegalStateException`, whereas the second method returns `false`.

» `E remove()`

» `E poll()`

removes and returns the element at the head of this queue, provided the queue is not empty. If the queue is empty, the first method throws a `NoSuchElementException`, whereas the second method returns `null`.

» `E element()`

» `E peek()`

returns the element at the head of this queue without removing it, provided the queue is not empty. If the queue is empty, the first method throws a `NoSuchElementException`, whereas the second method returns `null`.