`Set<ElmtType>` Interface

The classes that implement this interface are: **`TreeSet`** and **`HashSet.`**

Selected methods:

`boolean contains(ElmtType elmt)`	Returns true iff `elmt` is in the set
`int size()`	Returns number of elements in the set
`boolean add(ElmtType elmt)`	Ensures that `elmt` is in the set. Returns true iff the set changed as a result of this call
`boolean remove(ElmtType elmt)`	Removes `elmt` from the set. Returns true iff the set changed as a result of this call
`boolean isEmpty()`	Returns true iff the set contains no elements.
`Iterator<ElmtType> iterator()`	Returns an iterator over the elements in the set.

`Queue<ElmtType>` Interface

Selected methods:

`Queue<ElmtType> = new LinkedList<ElmtType>();`
: Creates an empty queue.

`boolean isEmpty()`
: Tests if this queue is empty.

`ElmtType peek()`
: Looks at the front element in the queue without removing it. PRE: `!isEmpty()`

`void add(ElmtType elmt)`
: Adds an element to the end of the queue.

`ElmtType remove()`
: Removes the front element and returns it. PRE: !isEmpty()

[More other side]

Reminder of some `LinkedList` and `ListIterator` operations by example:

```
LinkedList<Integer> list = new LinkedList<Integer>();
                        // create empty linked list that can hold ints

list.add(3);            // add a value to the end of the linked list

list.addLast(17);       // does the same thing as add

int num = list.getLast();         // returns last element in the list
                                  // PRE: !isEmpty()

int num = list.removeLast();
                    // removes last element in the list and returns it
                    // PRE: !isEmpty();

// Note: addFirst, getFirst, removeFirst: like the "Last" versions,
//                                    but at the beginning of the list

int num = list.get(i);   // gets the element at position i.
                         // (elements are numbered as in an array)

int howMany = list.size());      // number of elements in list

boolean empty = list.isEmpty();  // true iff list has no elements


ListIterator<Integer> iter = list.listIterator();
           // return list iterator positioned before the first element

boolean done = iter.hasNext();
              // returns true iff iterator is not after last element

int num = iter.next();        // returns element after iter position
                              // and advances iter past the element
                              // PRE: hasNext()

iter.remove();
          // removes element returned by last call to next or previous.
          // this call can only be made once per call to next or previous.

iter.add(32);              // adds element before the iterator position

iter.set(44);          // replaces the last element returned by a call
                       // to next or previous with the value given

ListIterator<Integer> iter2 = list.listIterator(list.size());
             // return list iterator positioned after the last element

boolean done = iter2.hasPrevious();
                   // returns true if iter is not before first element

int num = iter2.previous();  // returns element before iter2 position
            // and iter2 moves to position before the element returned
            // (in this ex previous() returns last element in the list)
                          // PRE: hasPrevious()

ListIterator<Integer> iter3 = list.listIterator(k);
            // return list iterator positioned before element k.
            // So, an initial call to next() returns element k;
            // an initial call to previous() instead returns element k-1
            // (elements are numbered as in an array)
```

Note: Illegal to use LinkedList mutators on a list you are iterating over.

[More other side]