ListIterators

Mason Vail
Boise State University Computer Science

ListIterator Interface

Lists in Java are expected to support a special version of *Iterator* called a *ListIterator*.

As a child interface of *Iterator*, *ListIterator* has the methods defined in *Iterator* and can be assigned polymorphically to *Iterator* references. It “is-a” *Iterator*. In addition, a *ListIterator* has the ability to start at any valid index position in its *List* when its indexed constructor is used. It can move to previous positions as well as next positions. It can insert new elements and replace elements as well as remove them. It can tell you the index of its next and previous elements.

ListIterator Purpose

The primary advantage of using a *ListIterator* is efficient navigation through a *List* and modification of a *List* in place. It is especially useful for a double-linked list implementation where avoiding O(n) indexed methods is essential for efficiency.

Because a *ListIterator* does not lose its place, its add(), remove(), and set() methods are O(1) at the *ListIterator*'s current position. Moving a *ListIterator* to an arbitrary position in the list is is still a O(n) process, but each individual move and each individual change is O(1).

ListIterator Constructors

- public **ListIterator()** - Default constructor initializes the ListIterator such that a call to next() would return the first element of the list. Equivalent to constructor ListIterator(0).
- public **ListIterator(int startingIndex)** - Indexed constructor initializes the ListIterator such that a call to next() would return the element at the given index.

ListIterator Methods: Navigation

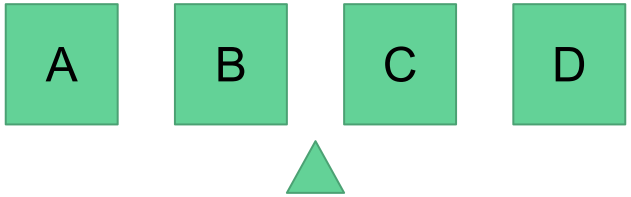
- public boolean **hasNext()** - Returns true if the Iterator has a next element. Returns false if there is no next element.
- public T **next()** - Returns the next element. Throws a NoSuchElementException if there is no next element.
- public int **nextIndex()** - Returns the index of the element that would be next.
- public boolean **hasPrevious()** - Returns true if the Iterator has a previous element. Returns false if there is no previous element.
- public T **previous()** - Returns the previous element. Throws a NoSuchElementException if there is no previous element.
- public int **previousIndex()** - Returns the index of the element that would be previous.

ListIterator Methods: Modification

- public void **remove()** - Removes the element most recently returned by either next() or previous(). Throws an IllegalStateException if not preceded by a call to next() or previous(). Cannot follow a call to add(), set(), or remove().
- public void **set(T element)** - Replaces the most recently returned element with the given element. Throws an IllegalStateException if not preceded by a call to next() or previous(). Cannot follow a call to add() or remove().
- public void **add(T element)** - Inserts the given element into the list in front of the *ListIterator*'s current position.

ListIterator: Indexed Constructor

User’s abstract perspective of a list



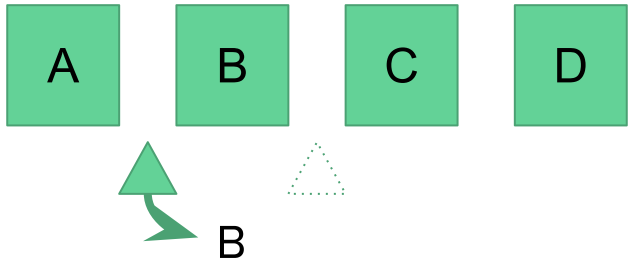
Using indexed constructor ListIterator(2) initializes this ListIterator before element C, the element at index 2.

For this list, the indexed constructor could be used to initialize a ListIterator anywhere from the front of the list (index 0) to the end of the list (index 4).

Regardless of where a ListIterator is initialized by the indexed constructor, no call has been made by the user to next() or previous(), and a call to remove() or set() would throw an IllegalStateException().

ListIterator: previous()

User’s abstract perspective of a list

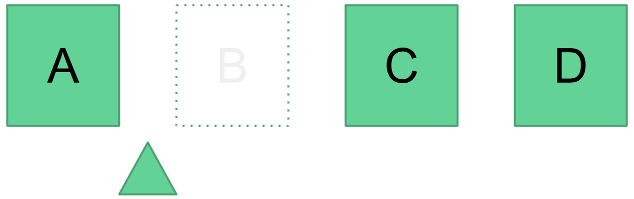


When previous() is called, the ListIterator moves to the position between A and B and returns B, the element it just passed over.

At this point, remove() would be expected to remove element B from the list and set() would replace element B.

ListIterator: remove() after previous()

User's abstract perspective of a list



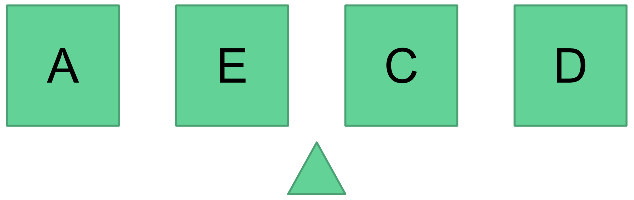
When remove() is called, element B, the element last returned by the ListIterator, is removed from the list. The ListIterator is between elements A and C.

A call to previous() would return A. A call to next() would return C.

A call to remove() or set() would be expected to throw an IllegalStateException.

ListIterator: add()

User's abstract perspective of a list



If add(E) is called, element E is inserted in front of the ListIterator's current position. The ListIterator is now between elements E and C.

A call to remove() or set() would be expected to throw an IllegalStateException.

A call to previous() would return the newly added E. A call to next() would still return C.

“Fail-fast” and ConcurrentModificationException

ListIterator, like *Iterator*, promises to reliably represent the contents of its *List* and is expected to fail-fast if any change occurs in the *List* that didn’t involve the *ListIterator*’s own add(), remove(), or set() methods.

As soon as a *ListIterator* detects an unknown change to the list, calling any of its methods should result in a ConcurrentModificationException.

ListIterators

Mason Vail
Boise State University Computer Science