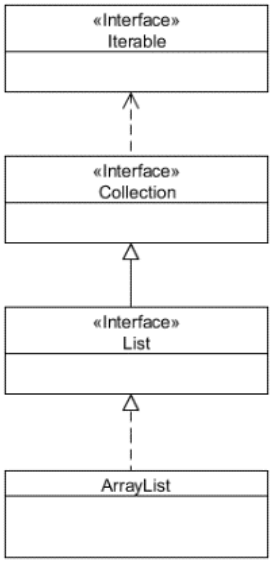
Kyle McClintick
Jared Hoffman
Ben Stevens

Lists and Iterators



A List is an interface in Java, which may have several different implementations. One of the most important implementations is ArrayList, which is a class that implements the behavior of the List interface using arrays as the data structure. It is an ordered collection. The user has complete control over the values of each index's value and can access and search the list with ease. To remove an element from a list at position n, one would have to use the remove method, passing it n+1 as the only parameter. As an interface, one would never create a list interface.

ListIterator is an interface that List provides. It is located within a List, and allows the element insertion and replacement as well as bidirectional access within the list in addition to the normal operations that its superinterface Iterator provides. The user can collect the iterator's current position in the list. A ListIterator has no current element though, 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(). Additionally, the remove() method can be used to remove elements from a list. An iterator for a list of length n has n+1 possible cursor positions. For example, the following list:

```
    [0]   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]   [10]
^     ^     ^     ^     ^     ^     ^     ^     ^     ^     ^     ^
```

With each ^ representing a possible cursor location, note how there are 11 indexes and 12 possible cursor locations.

Enhanced for loops also use Iterators. The Iterator in an enhanced for loop will go through each element the List being analyzed has, using the next() method in ListIterator to return the value of the next element. For instance:

```
ArrayList<Fruit> fruits = new ArrayList<Fruit>();

for(Fruit f : fruits) {f.doSomething();}
```

Would traverse each element, using next() to store the value of each element as a temporary field f until the next iteration of the loop upon which f would be given the value of a new element. The above statement is also equivalent to:

```
ArrayList<Fruit> fruits = new ArrayList<Fruit>();

for(Iterator i=fruits.iterator(); i.hasNext()){i.next().doSomething();}
```

This notation would be slower for very large lists, as for some data structures get(i) is an O(n) operations which can cause a for loop to become an $O(n^2)$ operation, and this does not happen in an enhanced for loop; it remains O(n).

One key difference between Iterators and ListIterators is that a ListIterators allows you to traverse through the list by going forwards or backwards, while Iterators can only check elements that are in front.

Sources

http://stackoverflow.com/questions/10978126/what-is-the-difference-between-iterator-and-listiterator

http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html