Being somewhat new to the Java language I'm trying to familiarize myself with all the ways (or at least the non-pathological ones) that one might iterate through a list (or perhaps other collections) and the advantages or disadvantages of each.

Given a List<E> list object, I know of the following ways to loop through all elements:

Basic [for](http://docs.oracle.com/javase/specs/jls/se7/html/jls-14.html#jls-14.14.1) [loop](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html) (of course, there're equivalent while / do while loops as well)

// Not recommended (see below)!

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

E element = list.get(i);

// 1 - can call methods of element

// 2 - can use i to make index-based calls to methods of list

// ...

}

Note: As @amarseillan pointed out, this form is a poor choice for iterating over Lists because the actual implementation of the get method may not be as efficient as when using an Iterator. For example, LinkedList implementations must traverse all of the elements preceding i to get the i-th element. In the above example there's no way for the List implementation to "save its place" to make future iterations more efficient. For an ArrayList it doesn't really matter because the complexity/cost of get is constant time (O(1)) whereas for a LinkedList is it proportional to the size of the list (O(n)). For more information about the computational complexity of the built-in Collections implementations, check out [this question](http://stackoverflow.com/questions/559839/big-o-summary-for-java-collections-framework-implementations).

Enhanced [for loop](http://docs.oracle.com/javase/specs/jls/se7/html/jls-14.html#jls-14.14.2) (nicely explained [in this question](http://stackoverflow.com/questions/85190/how-does-the-java-for-each-loop-work))

for (E element : list) {

// 1 - can call methods of element

// ...

}

[Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html)

for (Iterator<E> iter = list.iterator(); iter.hasNext(); ) {

E element = iter.next();

// 1 - can call methods of element

// 2 - can use iter.remove() to remove the current element from the list

// ...

}

**EDIT:** Added ListIterator

[ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html)

for (ListIterator<E> iter = list.listIterator(); iter.hasNext(); ) {

E element = iter.next();

// 1 - can call methods of element

// 2 - can use iter.remove() to remove the current element from the list

// 3 - can use iter.add(...) to insert a new element into the list

// between element and iter->next()

// 4 - can use iter.set(...) to replace the current element

// ...

}

**EDIT:** Added "functional-style" solution (thanks Dave Newton)

[Functional Java](http://functionaljava.org/)

list.map({E e => e++ } // can apply a transformation function for e

What other ways are there, if any?

I feel like this has got to be a duplicate, but I haven't been able to find what I'm looking for, so I apologize for this question potentially being redundant. (BTW, my interest does not stem at all from a desire to [optimize performance](http://stackoverflow.com/questions/2113216/which-is-more-efficient-a-for-each-loop-or-an-iterator); I just want to know what forms are available to me as a developer.)

**EDIT:** Moved ListIterationExample.java to a suggested answe