Lists and Iterators

Sources:

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

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

Lists and *iterators* are two closely related interfaces. They provide a framework for storing data and retrieving data, respectively. Neither can be instantiated directly- rather, they are implemented by various classes.

More specifically, lists are known as *ordered collections* or *sequences*. They contain data in a linear format and access it with an index number, acting in a similar manner to static lists. As such, they are a type of *Collection*, which is the top-level interface for all data storage. Examples of Lists include *ArrayLists* and *Stacks*. Classes implementing a list require a significant number of methods, including **get()**, **isEmpty()**, **set()**, and **size()**. They also require the **iterator()** command, which produces an iterator.

Iterators provide a means to traverse a list. The Iterator interface is quite simple, requiring only three methods- **hasNext()**, **next()**, and **remove()**. When created, they sit at the very beginning of a list. Calling **next()** causes them to return the first element. If **remove()** is then called, the most recently returned element is dropped from the list. Note that the iterator will remain in the same relative position in the list, even if the remaining elements are shuffled forward to the front. The **hasNext()** method will return true if there are any further elements and false if the iterator has reached the end of the list. It is good practice to call this method before attempting to call **next().**

This is the core functionality of the Iterator- the *reliable* manipulation of lists. When used properly, they ensure that the correct elements are read and removed, reducing the chance of so-called "fencepost" errors in which the element next to the one intended to be manipulated is reached instead.

To use an Iterator, simply call the **iterator()** command from a List-implementing class. Note that it has yet to retrieve a value, so calling **remove()** would throw an *IllegalStateException*. As such, the list/iterator look like this to start (the brackets indicate the current location of the iterator):

< > 1 2 3 4 5

Calling **next()** advances the iterator by one.

<1> 2 3 4 5

Calling **remove()** then removes the most recently returned element.

< > 2 3 4 5

Note that calling **remove()** twice in a row will cause an exception, since the most recently returned element has already been removed.

Example:

```
import java.util.ArrayList;
import java.util.Iterator;

public class tests
{
	public static void main(String args[])
	{
		ArrayList<Integer> ints = new ArrayList<Integer>();
		ints.add(1);
		ints.add(2);
		ints.add(3);
		ints.add(4);
		ints.add(5);
		ints.add(6);
		Iterator it = ints.iterator();
		while(it.hasNext())
			System.out.print(it.next() + " ");
		System.out.println();
		it = ints.iterator(); //Note that a new iterator must be created
		it.next();
		it.remove();
		it.next();
		it.next();
		it.remove();
		it = ints.iterator();
		while(it.hasNext())
			System.out.print(it.next() + " ");
		System.out.println();
		it = ints.iterator();
		it.remove(); //Deliberately causing an exception

	}
}
```

Output:

```
1 2 3 4 5 6
2 4 5 6
Exception in thread "main" java.lang.IllegalStateException
	at java.util.ArrayList$Itr.remove(Unknown Source)
	at tests.tests.main(tests.java:32)
```