Java Notes

Iterators

The List and Set collections provide *iterators*, which are objects that allow going over all the elements of a collection in sequence. The `java.util.Iterator<E>` interface provides for one-way traversal and `java.util.ListIterator<E>` provides two-way traversal. `Iterator<E>` is a replacement for the older `Enumeration` class which was used before collections were added to Java.

Creating an Iterator

Iterators are created by calling the *iterator()* or *listIterator()* method of a List, Set, or other data collection with iterators.

Iterator Methods

Iterator defines three methods, one of which is optional.

Result	Method	Description
`b =`	*it*.**hasNext**()	`true` if there are more elements for the iterator.
`obj =`	*it*.**next**()	Returns the next object. If a generic list is being accessed, the iterator will return something of the list's type. Pre-generic Java iterators always returned type `Object`, so a downcast was usually required.
	it.remove()	Removes the most recent element that was returned by `next`. Not all collections support `delete`. An *UnsupportedOperationException* will be thrown if the collection does not support *remove()*.

Example with Java 5 generics

An iterator might be used as follows.

```
ArrayList<String> alist = new ArrayList<String>();
// . . . Add Strings to alist

for (Iterator<String> it = alist.iterator(); it.hasNext(); ) {
    String s = it.next();  // No downcasting required.
    System.out.println(s);
}
```

Example as above but with enhanced Java 5 for loop

```
for (String s : alist) {
    System.out.println(s);
}
```

Example pre Java 5, with explicit iterator and downcasting

An iterator might be used as follows, wi.

```
ArrayList alist = new ArrayList(); // This holds type Object.
// . . . Add Strings to alist

for (Iterator it = alist.iterator(); it.hasNext(); ) {
    String s = (String)it.next();  // Downcasting is required pre Java 5.
    System.out.println(s);
}
```

ListIterator methods

`ListIterator` is implemented only by the classes that implement the `List` interface (`ArrayList`, `LinkedList`, and `Vector`). ListIterator provides the following.

Result	Method	Description
Forward iteration		
`b =`	*it*.**hasNext**()	`true` if there is a next element in the collection.
`obj =`	*it*.**next**()	Returns the next object.
Backward iteration		
`b =`	*it*.**hasPrevious**()	`true` if there is a previous element.
`obj =`	*it*.**previous**()	Returns the previous element.
Getting the index of an element		
`i =`	*it*.**nextIndex**()	Returns index of element that would be returned by subsequent call to *next()*.
`i =`	*it*.**previousIndex**()	Returns index of element that would be returned by subsequent call to *previous()*.
Optional modification methods. UnsupportedOperationException thrown if unsupported.		
	it.add(*obj*)	Inserts *obj* in collection before the next element to be returned by `next()` and after an element that would be returned by `previous()`.
	it.set()	Replaces the most recent element that was returned by `next` or `previous()`.
	it.remove()	Removes the most recent element that was returned by `next()` or `previous()`.

BAD BAD BAD

Q: What does this loop do? Note mixing of iterator with index.

```
ArrayList<String> alist = new ArrayList<String>();
// . . . Add Strings to alist

int i = 0;
for (Iterator<String> it = alist.iterator(); it.hasNext(); ) {
    System.out.println(alist.get(i++));
}
```

A: It throws an exception when it goes beyond the end.

After `hasNext()` returns true, the only way to advance the iterator is to call `next()`. But the element is retrived with `get()`, so the iterator is never advanced. `hasNext()` will continue to always be true (ie, there is a first element), and eventually the `get()` will request something beyond the end of the ArrayList. Use either the iterator scheme.

```
for (Iterator<String> it = alist.iterator(); it.hasNext(); ) {
    System.out.println(it.next());
}
```

Or the indexing scheme, but don't mix them.

```
for (int i=0; i < alist.size(); i++) {
    System.out.println(alist.get(i));
}
```