# Iterator

An iterator is a way to iterate through a group of objects without having to worry about how to traverse that group. The goal is to not deal with the traversal implementation, but rather the abstract idea of "iteration", which is to view each item in the group exactly once.

The pre-Iterator way to iterate is to use indexes:

```
int myListMax = 3;

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

This loop prints each item in myList to the console. But wait, does it set the last element in myList? Does myList contain 3 elements, or does it hold elements in the indexed positions 0,1,2,3 ? If the latter is true, this loop will fail to set the last element in myList.

These kind of "off-by-one" errors are subtle bugs that can be hard to track down. What if we had a more complicted data structure than a list that required more complicated traversal to get to each object--we'd have to figure that out each time.

Instead, we delegate iteration to the "container" (anything that holds a group of objects can be called a container) and use the Iterator interface to get the objects out of the container. Thus, we have a single interface for all containers, and we are guaranteed that if the container implements Iterator correctly, then we will see every single object in the container exactly once.

Using an iterator looks like this (suppose myList contains Strings):

```
(List myList = new ArrayList();)
...

for (Iterator iter = myList.iterator() ; iter.hasNext() ; ) {
        String element = iter.next();
        System.out.println(element);
}
```

In the init part of the for loop we created a variable called "iter". It is of type Iterator, which is what myList.iterator() returns. Look up Iterator in the online Java 1.5 API and follow along. Iterator has two methods, hasNext() and next().

hasNext() returns true if there is another element in the iterator, and false if the iterator is empty (that is, if it has already shown you everything).

next() returns the next element in the Iterator.

Notice that the third part of the for () statement is empty. After the last ';' there is nothing but the closing ')' of the for statement. This is because calling iter.next() not only returns the next element, but also increments the iterator to point to the next element. You should only call iter.next() once per loop.

Iterators are so common in Java that there is syntactic sugar for dealing with them (introduced with

version 1.5):

```
for (String element : myList) {
        System.out.println(element);
}
```

does the exact same thing as the code above. You can only use this for classes that "implement" Iterable (and have a method called "iterator()"). Even if a class has a method that returns an iterator, you won't be able to use the syntactic sugar unless the class implements Iterable.