Today’s Goals

We begin with `ArrayIntList` & `LinkedIntList`.

Our goals are:
- To make an interface that captures the behaviors of an “IntList”
- To write a client search function in both of these classes
- To learn what iterators are (and why they might be useful!)
- To re-implement a better version of search using iterators

IntList Interface

```java
1 public interface IntList {
2   void add(int value);
3   int get(int index);
4   void remove(int index);
5   void set(int index, int value);
6   int size();
7   String toString();
8 }
```

Then, to make `ArrayIntList` and `LinkedIntList` actually use it:

```java
1 public class ArrayIntList implements IntList {
2   // ...
3 }
4 public class LinkedIntList implements IntList {
5   // ...
6 }
```

Now, let’s make a client search function for `IntList`:

```java
1 /** Returns true if value can be found in list and false otherwise. */
2 public boolean search(IntList list, int value) {
3   for (int i = 0; i < list.size(); i++) {
4     if (list.get(i) == value) {
5       return true;
6     }
7   }
8   return false;
9 }
```

Consider the following:

```java
1 IntList arrayList = new ArrayIntList();
2 IntList linkedList = new LinkedIntList();
3 */ Add 1000000 elements to each list... */
4 search(arrayList, 9);
5 search(linkedList, 9);
```

What is the complexity of the two method calls?
- In `ArrayIntList`, get is an $O(1)$ operation
- In `LinkedIntList`, get is an $O(n)$ operation
So, $O(n)$ and $O(n^2)$, respectively.
How do for-each loops work? 5

How does Java KNOW the ordering?

If you were implementing a for-each loop for a type T, what would you need to be able to do with the elements in that data structure?

We would need to be able to provide them one after one...

Java calls this idea an "Iterator".

Iterator
The Iterator interface allows us to tell Java how to order the elements of a data structure:

```java
public interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void remove();
}
```

This says, "to be an Iterator, classes must define hasNext, next, and remove".

This is a lot like how we use a Scanner!!

foreach Loop Warning! 7

You Can’t Remove In A foreach Loop!

```java
Set<String> set = new TreeSet<String>();
set.add("hello");
set.add("world");
for (String s : set) {
    if (s.startsWith("h")) {
        set.remove(s);
    }
}
```

Exception in thread "main" java.util.ConcurrentModificationException
- at java.util.TreeMap$PrivateEntryIterator.nextEntry(TreeMap.java:1115)
- at java.util.TreeMap$KeyIterator.next(TreeMap.java:1169)
- at Client.main(Client.java:12)

ConcurrentModificationException
A ConcurrentModificationException happens when you try to edit a structure that you are looping through in a foreach loop. You should not try to remove inside a foreach loop! It will fail!

Using an Iterator 6

Using a Scanner
```java
Scanner input = new Scanner(...);
while (input.hasNext()) {
    System.out.println(input.next());
}
```

Using an Iterator
```java
List<Integer> list = new ArrayList<Integer>();
... ...
Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

You’ve actually been using iterators without knowing it:

Java uses the iterator() method to power for-each loops!

Using Iterators to make search better 8

```java
/** Returns true if value can be found in list and false otherwise. */
public boolean search(IntList list, int value) {
    Iterator<Integer> it = list.iterator();
    while (it.hasNext()) {
        int next = it.next();
        if (next == value) {
            return true;
        }
    }
    return false;
}
```

Now, they’re both $O(n)$.

Implementing numberOfOdds in multiple ways 9

As a client...
- with loops
- with iterators

As an implementor...
- with loops
- with iterators

See code for solutions.