In Java, you have a set of template classes built in:

```
import java.util.ArrayList;        // specifies how the class is implemented (as an array)

import java.util.LinkedList;       // specifies how the class is implemented (as a doubly linked list)
```

(Skim http://www.tutorialspoint.com/java/java_collections.htm for an overview of this material.)

You are used to template classes, like the ItemType in C++. We supply the particular type in the <>, like in C++, but we're using Java's class-based Integer, Boolean, Double, and String here. In general, you can implement the same collection in multiple ways.

```
ArrayList<Integer> int_array_1 = new ArrayList<Integer>();        // array based

LinkedList<Integer> int_array_2 = new LinkedList<Integer>();      // doubly linked list
```

Java defines all of these Collections in an elaborate inheritance hierarchy that we won't cover here (the http://www.tutorialspoint.com/java/java_collections.htm link is a start if you want to know more).

But now that you know about arrays and lists and trees, you can see that these classes let you define more interesting data types, like arrays of arrays, or arrays of lists. It also makes it easy to define specialized data types, like stacks and queues, which are both variations on an array with extra rules. We'll also be using what Java calls maps (and Python calls dictionaries), which let you store data you don't remember and look it up by a key that you do remember. Think of an address book that lets you look up names (which are easy to remember) and then find addresses (which are harder to remember):

```
"Sherlock Holmes"      221 Baker St., London, England

"Gollum"               Horrible Dark Place, Somewhere
```

Given this class, you could have coded the polynomial homework in Java much faster, since you'd have a class with a linked list all built in that you could use; it would happily add or remove anything you liked. This week's lab is to get you a little practice in using these Java collections, in particular the array and map classes, which will be useful in defining a Graph class.

Java is superficially a lot like C++, but the memory model underneath it is quite different. For example, if you write a class, and then try to copy the objects it makes, your copies will tend to be shallow without extra work. Don't assign or initialize objects this way in Java without checking that your copies are deep. On the happier side, in Java, you never have to make a destructor; Java won't have memory leaks, even if you never delete an object, due to the garbage collection logic it uses (when a pointer to a chunk of memory is about to be lost, Java deletes the memory automatically). Java expects you to use its pre-developed classes, and also makes you plan for certain potential problems in advance, using exception handling. (This is actually one of Java's nicer features, and it's also possible to use it in C++, although I didn't make you handle exceptions in C++.) You'll see this when you read a file in the last homework.

We'll begin lecturing on this Wednesday, but for this week, you should be familiar with how these basic structures work; for the last homework, we'll combine a few of them to create a class for representing Graphs and finding paths from one vertex to another.