# Generic Types

Lab 2 next week will use inheritance features of Java that we don't touch on in the CS 150 class. You should read some of the Weiss text about subclasses, abstract classes, interfaces, and generic types:

Sections 4.2 and 4.4

These are the main things we'll be talking about in the next few classes.

The Java Collections collect data, but what kind of data? Python doesn't care about types, so it will let you have a list where one element is an integer, the next is an object of class Person, the next is a boolean, and so forth. Java does care about types.

All of the objects in a Java collection have to have the same type. But what type?

Java's solution (Java 5 in 2004) to this is to allow classes to *parameterize* types. For example, in Lab 2 you will implement a class called MyArrayList. Here is the start of this class declaration:

```
public class MyArrayList<E> {
        E [ ] data;
        int size;
        public MyArrayList( ) {
                size = 0;
                data = new E[2];
                ....
```

A specific list might have type

MyArrayList<String>

We will construct a new array list of Strings with

MyArrayList<String> L = new MyArrayList<String>();

Note that the constructor we call is

new MyArrayList<String>( )

though in the class declaration the constructor is defined as

public MyArrayList( )

Look again at the class declaration:

```
public class MyArrayList<E> {
        E [ ] data;
        int size;
        public MyArrayList( ) {
                size = 0;
                data = new E[2];
                ....
```

E is used as a type throughout this class declaration. Of course, each instance of E refers to the same type.

We could also have classes that use several type parameters:

```
public class Pair<A, B> {
        A first;
        B second;
        public A getFirst() {
                return first;
        }
        ....
```

The actual types put in place of the type parameters need to be *reference types* -- classes or arrays. Primitive types, such as int, boolean, and float are not allowed. Fortunately, Java provides *wrapper* classes for each of the primitive types. For example, Integer is a Java class that holds a single int value. Java even automatically wraps and unwraps primitive types.

For example, suppose you want to make an ArrayList of ints. The declaration is

```
ArrayList<Integer> L = new ArrayList<Integer>();
```

We could then call the add method for this list to put a value into L with

```
L.add(23);
```

Java automatically wraps 23 into an Integer to fit into this list, as though you had written L.add( new Integer(23) );

Similarly, you can say

```
int x = L.get(0);
```

even though L.get( ) technically returns an Integer, not an int.