Return to Module Overview Page (https://onlinelearning.berkeley.edu/courses/665040/pages/module-six)

6.8: Generics

One of the limitations of the collections framework is that there is no type checking of its elements. What if you create a collection that takes only Double objects and you insert a String? This problem would not be caught until run time, which makes debugging more difficult. In the example below, the problem will not appear during compile time, but will throw a class cast exception during run time:

```
ArrayList alist = new ArrayList();
alist.add(new Double(12.3456));
alist.add("zero string");
Double dbl0 = (Double)alist.get(0);
Double dbl1 = (Double)alist.get(1);
```

Enter Generics. Generics, or parameterized types, allow you to create customized collections (classes, interfaces and methods) with a user specified type.

To create a customized collection, use the following format:

```
Collection<type>  collect = new Collection<type>();
```

If we rewrote our code above using Generics, here is what we will get:

```
ArrayList<Double> dbllist = new ArrayList<Double>();
dbllist.add(new Double(12.3456));
dbllist.add("zero string");
Double dbl2 = dbllist.get(0);
```

We now get compile time error checking (the code above will fail to compile due to the 3rd line). Comment out the third line, recompile and the program will work fine. In addition, we no longer need to do type casting when retrieving a value.

You can tell whether a class or interface accepts parameterized types if in the documentation its name is followed by angle brackets (<>). Practically all of the Collection and Map classes are generic.

There is more to generics than what we have presented here, including concepts such as Bounds, Wildcards and Generic Methods. They are advanced concepts that deserve more space than what we can devote here.