Lab 7 - patterns - solution

Simon Rogers

2nd March 2015

Task 1

This task was trickier than I intended. To make a solution that will work for both Integers and Strings is hard (see later). However, to see a version that can just use, say, Integers, see `FindMax/ItArray.java` and `FindMax/Finder.java`.

To make it work with any type we have to use generics. In particular we modify (see `Iterator2/`) `ItArray` to take a generic type `T` (note that this is beyond this course - you don't need to know this, but it's very useful):

```
import java.util.Iterator;

public class ItArray<T> implements Iterator<T> {
    // Inside my iterable array, I'll have a standard array
    public T[] array;
    // Start at position zero
    int pos = 0;
    public ItArray(T[] array) {
        // Copies a reference to the array
        this.array = array;
    }
    // hasNext returns true if there are any items left
    public boolean hasNext() {
        if(pos<array.length) {
            return true;
        } else {
            return false;
        }
    }
    // return the next item (and increment the position)
    public T next() {
        return array[pos++];
    }
    // Remove it optional - you don't have to implement it but
    // you do have to define it. If you don't implement it, have it
    // throw an exception.
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

when we create an `ItArray` object now, we must pass a type. e.g. `ItArray<Integer> ia = new ItArray<Integer>()`. We must also modify the `findBiggest` method to make it a generic method:

```
public static <T extends Comparable<T>> T findBiggest(Iterator<T> b) {
        // Start by setting the biggest to the first one
        T biggest = b.next();
        int pos = 1;
        while(b.hasNext()) {
            T current = b.next();
            if(biggest.compareTo(current)<0) {
                biggest = current;
            }
            pos++;
        }
        return biggest;
    }
```

Note two things. Firstly in the method declaration we declare a type `T`. This is determined at runtime depending on the type passed to the method. Secondly, the standard way of typing a method would be:

```
public static <T> T findBiggest(Iterator<T> b) {
```

(The second `T` is the return type. They don't have to be the same). This says that this method works with objects of type `T`. It suggests that `T` can be any type *but* this would not compile. Our method uses the `compareTo` method of `T` so we need to tell Java that we only want this method to be used by objects that `extend compareTo`. Hence the additional bit in the declaration.

A full working solution that works for Strings and Integers (and any object that has `compareTo` defined) can be found in `Iterator2/`.

Task 2

The second task is very similar to the example in the notes and a working solution can be found in `CourseComposite/`. Part of the question asks about how to have hierarchies of hierarchies (i.e. blocks of blocks). The key is in the type of the `ArrayList` in the composite object.