Assignment 6: Iterators and Arrays

In what follows, "develop a class" means

1. to decide on names for the class and its fields;
2. to give examples of Java declarations that create instances of the class;
3. to draw a class diagram (but this need not be turned in);
4. to write the Java code for the class, including its fields and constructor, and including Java comments that describe the purpose of each field.

In what follows, "develop a method" means

1. to give examples of Java expressions that call the method, together with Java comments that specify the expected results of those calls;
2. to write a Java comment that explains the purpose of the method;
3. to write Java code for the method, including its header and body;
4. to convert the examples into executable tests.

Note that you are *not* required to use templates for this assignment.

Problem 1

1. Develop a class `Address` with fields that represent a city and a zip code. The city shall be represented by a `String`; the zip code shall be represented by an `int`. The `Address` class shall include a method `makeString` that takes no arguments and returns a `String` that describes this instance of the `Address` class, including the class name and the values of its fields.
2. Develop a class `Person` with fields that represent the name, address, and age of a person. The `Person` class shall include a method `makeString` that takes no arguments and returns a `String` that describes this instance of the `Person` class, including the class name and the values of its fields.
3. Copy the Java code for the following interfaces and classes from Lab6: `IObj2Bool`, `IRange`, `ALoObj`, `MTLoObj`, `ConsLoObj`, `ListRange`.

4. Write Java code that constructs example lists, including one of at least five `Persons` and one of at least five `Addresses`.

5. Define an interface `IComparator` that contains a method `compare` that takes two `Objects` as arguments and produces an integer result that is negative if the first argument should be regarded as less than the second, zero if the two arguments should be regarded as equal, or positive if the first argument should be regarded as greater than the second. (This is the same as the standard Java interface `java.util.Comparator` documented at `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Comparator.html`.)

6. Develop classes that implement `IComparator` and perform the following comparisons:

   - Compare two addresses by zip code, in the usual numerical order.
   - Compare two addresses by the name of the city, in the usual lexicographic (dictionary) order.
   - Compare two persons by age, in the usual numerical order.
   - Compare two persons by their names, in the usual lexicographic (dictionary) order.

7. Develop a `minimum` method in a `TestClass` that, given an `IRange` and an `IComparator` as arguments, returns the smallest object generated by the `IRange`, where "smallest" is as defined by the `IComparator`. You should assume that the `IRange` will always generate at least one object.

8. Give examples that use the `minimum` method with `IRanges` that generate the elements of your examples of lists of persons and addresses. Your examples should use all four `IComparators` you developed earlier.

9. Develop a `sort` method in the `TestClass` that, given an `IRange` and an `IComparator`, returns a list of the objects generated by the `IRange` in the order determined by the `IComparator`.

10. Develop an `ArrayRange` class that implements `IRange`. The constructor for this class shall take an array of objects as its lone argument, and instances of the class shall generate the elements of the array in order starting with element 0. (You will see this in class and in the next lab.)

11. Write Java code that creates example arrays of addresses and of persons. Test the `minimum` and `sort` methods on your example arrays, using `ArrayRange`.

12. To build an arbitrary collection of data, we desire an interface that allows us to add new items into a mutable collection. Write Java code for an interface `ICollection`, which contains the following methods:

    (a) `add`, which takes an `Object` as argument and changes the state of the collection by adding the object to the collection; and

    (b) `iterator`, which returns a new `IRange` object that generates the current elements of this collection in some order.

13. Develop a class `ListCollection` that implements the `ICollection` interface by using an `ALoObj` to hold the objects that have been added to the collection. Give examples that create instances of `ListCollection` and use those instances with the `minimum` and `sort` methods in `TestClass`.

14. Develop a class `ArrayCollection` that implements the `ICollection` interface by using an array to hold the objects that have been added to the collection. The maximum number of objects that can be added to the collection shall be specified by an `int` argument that is passed to the constructor for the `ArrayCollection` class. If an attempt to is made to add more objects than were specified when the `ArrayCollection` was created, then that attempt shall be ignored, and the state of the `ArrayCollection` shall remain unchanged.