9 Abstracting Traversals and Algorithms.

In this problem set you will work with several predefined classes that represent cities and lists of cities, as well as files that provide infrastructure for the tests and for dealing with user input.

The focus of your work is on learning to use abstractions in building reusable program components.

Part 1: Iterators, Loops, User input

Start by downloading the file **HW9part1.zip** and making an Eclipse project **HW9part1** that contains these files. Add **jpt.jar** as a *Variable* to your project. Run the project, to make sure you have all pieces in place. The main method is in the class *Interactions*.

The classes you will work with are the following:

- **class** *City* represents name, state, latitude, longitude, and a zip code for one city
- *AListOfCities* and its subclasses represent lists of cities
- **interface** *Traversal* to represent an iterator
- **class** *Examples* is the class that holds examples of data and the tests for all your methods
- **class** *Interactions* is the class that facilitates user interactions and allows you to explore the behavior of parts of your program
- **class** *Algorithms* contains methods that implement loops, such as our *orMap* and filter as well as other algorithms, such as sorting algorithms.
- **interface** *ISelect* and **interface** *IObj2Obj* represent function objects consumed by the loop methods.
- **interface** *ISame* is our standard interface for implementing the usual extensional equality comparison of objects

9.1 Problem

For the given classes *City*, *AListOfCities* (and its subclasses), and the interface *Traversal* design the following classes and methods:

1. Modify the classes *AListOfCitiest* and its subclasses to implement the *Traversal* iterator.
2. In the **class** *Algorithms* design the method *buildList* that consumes an *Traversal* iterator and produces a list of cities. In one of your tests for this method make a copy of a list of cities and compare the results. (You may need to reverse the result.)
3. Explore the use of the method *buildList* in the **class** *Interactions* by using the *InGuiTraversal*, *InConsoleTraversal* and *InFileBufferedTraversal* iterators to provide the sources of data.
4. Design the methods *andMap* and filter in the *Algorithms* class.
5. In the *Examples* class include the following additional tests for the filter, orMap, and andMap methods:
   - produce a list of all cities in a given state
   - find out whether there are any cities in a given state in some list of cities
   - are all cities in some list in a given state
   - is there a city with the given name in this list
   - produce a list of all cities with the given name from some list of cities
   - do all cities in some list have the given name ∎
6. Design *insertionSort* in the *Algorithms* class that consumes a *Traversal* and a *Comparator*.
7. In the *Examples* class include the following additional tests for the *insertionSort* methods:
   - sort the list of cities by latitude
   - sort the list of cities by city and state
   - sort the list of cities by state and city

   Remember, you need to design a *Comparator* for each case.

9.2 Problem

The goal here is to start thinking about systematic design of tests.

Read the code for the **class** *TraversalTestHarness*. Design a test suite for it.

Write a pragraph or two of documentation that describes what kinds of tests can/cannot be done using this test harness. ▮

Part 2: Understanding Equality of Lists.

9.3 Problem

The goal here is to understand the difficulties in making copies of lists and comparing them for equality.

The file CopyTests.java defines three different ways of copying lists of cities.

- Use each of the methods to perform the following series of tasks:
  - Create *list1* of six cites and *list2* that is the result of copying *list1*.
  - Sort *list2*. Print both lists.
  - Start afresh - with *list2* being a new copy of *list1*, then change the name of one city in *list2*. Again, print the resulting lists.
- Design examples/tests for each of these methods.
- Design the method that tests the equality of two lists according to the corresponding copy method. .It produces true when comparing a list with its copy and produces false if the other list could not be produced by this copy method.

▮

9.4 Writing Problem

Writing assignments are separate from the rest of the assignment for the week. You should work on this assignment alone, and submit your work individually on the due date for the rest of the homework. The answer should be about two paragraphs long – not to exceed half a page or 300 words.

Look at the last problem in this homework. Think of when each type of copying of lists is appropriate. Describe briefly the scenaria where each of the copying would be appropriate and why. ▮