9 Assignment

Etudes

1. The last exercise in the **Part 1** of the lab.
2. The last exercise in the **Part 2** of the lab.
3. The first and second exercise in the *Defining and Handling Exceptions* section of the **Part 3** of the lab.
4. The filter method in the *Defining and Handling Exceptions* section of the **Part 3** of the lab.
5. The *ArrayList and Java Libraries* section of the **Part 3** of the lab.

Main Assignment: 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 **HW9.zip** and making an Eclipse project **HW9** 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
- *AList<T>* and its subclasses represent lists of objects of the type *T*
- *interface Traversal<T>* 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<T extends ISame>* is designed to contain methods that process data generated by a *Traversal*. Examples of such methods are *orMap*, *filter*, and sorting algorithms.

- *interface ISelect<T>* and *interface IObj2Obj<T, S>* represent function objects consumed by the methods in the *Algorithms<T extends ISame>* class.

- *interface ISame* is our standard interface for implementing the usual extensional equality comparison of objects

9.1 Problem

Run the project and explore the buttons generated by the *Interactions* class. Notice, that the button labels correspond to the names of methods that have the following headers:

public void methodName()

The buttons allow you to execute one method at a time — within the context of your program. Try the *GUI* input, the console input, and the file input on small files.

9.2 Problem

For the given classes *City*, *AList<T>* (and its subclasses), and the interface *Traversal<T>* design the method *map* in the *Algorithms* class that consumes a *Traversal* and an instance of the class that implements the *IObj2Obj<T, S>* interface and produces a new *AList<S>*.

Use this method to produce a list of the names of all cities in the given *AList<City>*.

9.3 Problem

Design the class *ALT* that implements the *Traversal* interface and records its data in an *ArrayList*.

Note: We did this in class on Monday.

Use this method to produce a list of the names of all cities in the given *AList<City>*.

9.4 Problem: Extra Credit

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.

∎