Project #3

Computer Science 2334

Spring 2008

User Request:

"Create a Graphical User Interface and import/export software for a Zip Code Database."

Milestones:

1	Import zip code data from a text file into the database. Export the zip code database to a text file. A sample text file "zips.txt" will be provided.	(5 points)
2	Implement Serializable for the zip code database class.	(5 points)
3	Utilize Object Serialization to save and load the zip code database to and from a binary file.	(15 points)
4	Implement a simple Graphical User Interface using Swing that allows for selecting among the program's functions, in addition to the specific functions described in points 5 – 8.	(15 points)
5	Use a JFileChooser dialog to allow the user to choose the file used when saving and loading the database. Additionally, use a JFileChooser dialog to allow the user to choose the file used when importing and exporting the database.	(5 points)
6	Use JList objects to allow the user to select one or more zip codes to be mapped from among the zip codes imported or loaded from data files. This will be done through JList objects that allow users to select at the state, city, and zip code level.	(10 points)
7	Implement code to write JavaScript for generating maps using the Google Maps API.	(10 points)
8	Plot the selected zip codes using the code from step 7.	(5 points)
➲	Develop and use a proper design.	(15 points)
➲	Use proper documentation and formatting.	(15 points)

Description:

Your previous project used console-based input to determine the files with which to work and presented output to the user in the form of text. Console-based input and text-based output are quite appropriate for some programs. However, graphical interfaces are more appropriate for other programs. This project will be a program of the latter type. Here you will develop a full-fledged Graphical User Interface (GUI) for dealing with zip code data, including using the GUI for selecting files, filtering the zip code data, and displaying the data in a graphical format. This program will support importing, exporting, loading, saving, selecting, and displaying zip code data. For this program you **may** reuse some of the classes that you developed for Project #2, although you are not required to do so.

Importing/Exporting. For this project, "importing" results means reading files that have the same format used in the sample text file. Loading/Saving. Besides importing/exporting data from/to text files, your code will be able to load and save data in its own "native" format. This feature will be accomplished through the use of object serialization.

File Selection. The user should be able to use the GUI to choose which file is to be used when importing/exporting and loading/saving data. Your program will handle this using a JFileChooser dialog.

Data Selection. Once data is imported or loaded into your program, the user should have the option of selecting a subset of the data. This will be done via the GUI, which will first allow the user to select a single state, then select a single city from that state, then select one or more zip codes from that city. Your program will handle this using a JList for each selection.

Mapping/Displaying. Finally, your program will support displaying information for the selected data set. To do this, your program will generate JavaScript to provide data to a Google Maps server that will create maps and send them back to a browser launched by your program. A Java component will be provided to you that will launch a browser to display the map that is returned. The API documentation and source code for this component will be posted on the class web pages.

Design Issues:

The basic design of the GUI will be posted on the class web page. If you wish to modify the interface, you will need the permission of the instructor and will need to present a design document similar in form to (though differing in content from) the one that will be posted. If you choose to change the GUI design, think about how it should be laid out such that the interface is logical and it is easy to see the data you're interested in. Also think about making the buttons and/or menus logical and consistent. What would make the program easy for the user to use and understand without a lot of experimentation?

The zip code database should be have a hierarchical structure. The zip code database object should contain a list of state objects. Each of the state objects could contain a list of city objects. Each of the city objects should contain a list of zip code objects.

This is the largest project we've had so far. (Don't worry; they won't get much bigger than this one.) So, make sure to start early and budget your time well. Once you have a good design, you can write a part at a time and test it before moving on to the next part. Don't expect to be able to finish the project if you put it off until the last minute; on the other hand, if you use your time well, you should have plenty of it.

Object Serialization:

In lab, you did an exercise where you stored a list of objects to a data file using

ObjectOutputStream and then read them back into the program using ObjectInputStream. You are to use this method for storing objects into a file to store the zip code database. The program should allow the user to choose not to save the database when they close the program; however, it should warn the user if there is modified data that is unsaved.

File Chooser Dialogs:

The Java API includes a class named JFileChooser that provides a dialog to allow the user to choose a file with which the program will work. The API documentation for JFileChooser can be found at http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JFileChooser.html. The following code shows how to create a JFileChooser.

```
     JFileChooser chooser = new JFileChooser( "." );
     // Below, frame is an instance of JFrame that is the main window
     // of the program.
     int returnVal = chooser.showOpenDialog(frame);
     if(returnVal == JFileChooser.APPROVE_OPTION)
     {
          System.out.println("You chose to open this file: " +
                    chooser.getSelectedFile().getPath());
     }
```

You should examine the API specifications for the showOpenDialog() and showSaveDialog() methods of JFileChooser which can be found at the following URLs:

http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JFileChooser.html#showOpenDialog(java.awt.Component)

http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JFileChooser.html#showSaveDialog(java.awt.Component).

Note that you must call getSelectedFile() before you can call getPath() to retrieve the actual string representation of the path and name of the file the user chose.

Due Dates and Notes:

1.No third-party GUI packages may be used. Only standard Java classes and packages are allowed on projects. **Using a non-standard class will result in the program not compiling on the graders' computers, and a non-compiling program will receive a grade of zero.**

2.Make sure to start early and budget your time well. Once you've got a good design, you can write a part at a time and test it before moving on to the next part. For example, you can test the import/export code by importing a file and exporting the data to a second file without changing it. If the two files are exactly the same (you can easily check using a utility such as diff on UNIX or UNIX-like systems), then the import and export code is working properly. Code re-use from Project #2 may be applicable here as well, and can speed your development. **If you do not understand the design you developed in lab, it is your responsibility to attend office hours early in the project to get help with your design.**

3.Your revised design and detailed Javadoc documentation are due on Thursday, March 13th. Submit your revised UML design on engineering paper or a hardcopy produced using UML layout software, a hardcopy of the Javadoc documentation, and a hardcopy of the stubbed source code at the beginning of your assigned lab session. Submit the project archive following the steps given in the Submission Instructions by 9:00pm.

4.The final version of the project is due on Thursday, March 27th. Submit your final UML design on engineering paper or a hardcopy produced using UML layout software, a hardcopy of the Javadoc documentation, and a hardcopy of the source code at the beginning of your assigned lab session. Submit the project archive following the steps given in the Submission Instructions by 9:00pm.

5.You are not allowed to use the StringTokenizer class. Instead you must use String.split() and a regular expression that specifies the delimiters you wish to use to "tokenize" or split each line of the file.

6.You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

7.As noted in the syllabus, you are required to work on this programming assignment in a group of at least two people. It is your responsibility to find other group members and work with them. The group should turn in only one (1) hard copy and one (1) electronic copy of the assignment. Both the electronic and hard copies should contain the names and student ID numbers of all group members. If your group composition changes during the course of working on this assignment (for example, a group of five splits into a group of two and a separate group of three), this must be clearly indicated in your write-up, including the names and student ID numbers of everyone involved and details of when the change occurred and who accomplished what before and after the change.

8.Each group member is required to contribute equally to each project, as far as is possible. You must thoroughly document which group members were involved in each part of the project. For example, if you have three functions in your program and one function was written by group member one, the second was written by group member two, and the third was written jointly and equally by group members three and four, both your write-up and the comments in your code must clearly indicate this division of labor.