Project #3 – Revision 1

Computer Science 2334

Fall 2007

User Request:

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

Milestones:

1. Create import software that can read the output files from Project 2 to populate a hurricane database for Project 3. Create export software that can produce output files of the same format as those used in Project 2. *(5 points)*

2. Implement *Serializable* for all data classes. *(5 points)*

3. Utilize Object *Serialization* to store and retrieve the database of cities and storms 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 to read/store items from/to when saving/loading the database. Additionally, use *JFileChooser* to allow the user to choose the file used to import/export items ~~to/from~~ from/to. *(5 points)*

6. Use *JList* objects to allow the user to select a zip code to be mapped from among the zip codes imported or loaded from results files, to select a distance from among the distances associated with the selected zip code, and to select one or more storms from among the storms associated with a zip and distance pair. *(10 points)*

7. Implement code to write ~~XML~~ JavaScript for generating maps using the ~~Yahoo!~~ Google Maps API. *(10 points)*

8. Integrate a map component into your Graphical User Interface that is used to display the location of a zip code and its associated storm(s). *(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. Project #3 will be a program of the latter type. Here you will develop a full-fledged Graphical User Interface (GUI) for dealing with zip code and storm data, including using the GUI for selecting files, selecting among the data items found in the files, and displaying the data in a graphical format. This program will support importing, exporting, loading, saving, pruning, and displaying results of the type created by your code in Project #2. For this program you **may** *reuse* some of the classes that you developed for Project #2.

Importing/Exporting. For Project #3, "importing" results means reading files that use the output format specified in Project #2. Similarly, "exporting" means creating files following that same format.

Loading/Saving. Besides importing/exporting data from/to Project #2 formats, your Project #3 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 what files are to be used when importing/exporting and loading/saving data from/to a data file. Your program will handle this using a *JFileChooser* dialog.

Data Pruning. Once data is imported or loaded into your program, the user should have the option of pruning it (removing data items from its internal memory, although not the file unless the user subsequently exports or saves it). This will be done via the GUI, which will first allow the user to select a zip code from among those loaded, then select a distance from among those associated with a given zip code (if there is more than one), then select one or more storms from among those associated with the selected zip code (if there is more than one). Your program will handle this using a *JList* for each selection.

Mapping/Displaying. Finally, your program will support displaying information about a selected zip code, distance, and selected associated storms. To do this, your program will generate ~~XML~~ JavaScript to provide data to a ~~Yahoo!~~ Google Maps server that will create maps and send them back to a browser launched by your program. A ~~Swing~~ 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 webpage. 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?

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. This may seem overwhelmingly large, but 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. Don't expect to be able to finish the project if you put it off until the last minute, but 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 and then read them back into the program using *ObjectOutputStream* and *ObjectInputStream*. You are to use this method for storing objects into a file to store the database of hurricanes and cities. 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 that the program will work with. 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*. This is sample code that is intended to show you how to use *JFileChooser* and should not be copied directly into your program. Instead of copying this code, you should analyze and modify it to fit the requirements of this project.

```
import javax.swing.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 are found at
http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JFileChooser.html#showOpenDialog(java.awt.Component)
and
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.

Extra Credit:

You will be allowed to extend the basic features required for this project to earn extra credit towards your project grade in the course. An addendum to this handout will be posted on the class website that discusses extra credit options for the project.

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 as the course syllabus specifies, 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 Wednesday, October 17th.*** Submit a hard copy of your revised UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of both the stub code and the JavaDoc documentation generated from the comments in your stub code at the ***beginning of class***. Finally, submit a soft copy of your stub code using the submit tool on *codd.cs.ou.edu* by ***9:00pm***. This submission counts as part of the design and documentation portions of the project grade.
   The commands for submitting your stub code on *codd.cs.ou.edu* are:
   ```
   cd design3
   /opt/cs2334/bin/submit cs2334-010 project3-design <.java filenames>
   ```
   where `<.java filenames>` is a list of the .java files you are submitting.

3. The final version of the project is due on ***Wednesday, October 31st.*** Submit your final UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of your source code, the JavaDoc documentation generated from the comments in your code,

and COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt. Finally, submit a soft copy of your source code files and the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files using the submit tool on *codd.cs.ou.edu* by ***5:00pm***.

The commands for submitting your final project on *codd.cs.ou.edu* are:

```
cd project3
/opt/cs2334/bin/submit cs2334-010 project3-final <.java filenames> <other files>
```

where `<.java filenames>` is a list of the .java files you are submitting and `<other files>` are the other files you are required to submit. Make sure that the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files are present in the directory along with all of your .java source files.

4. The *Get Out of Jail Free* due date for this project is ***Friday, November 2nd***. Submit your final UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of your source code, the JavaDoc documentation generated from the comments in your code, and COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt. Finally, submit a soft copy of your source code files and the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files using the submit tool on *codd.cs.ou.edu* by ***9:00pm***.

   The commands for submitting your final project on *codd.cs.ou.edu* using Get Out of Jail Free are:

   ```
   cd project3
   /opt/cs2334/bin/submit cs2334-010 project3-goojf <.java file names> <other files>
   ```

   where `<.java filenames>` is a list of the .java files you are submitting and `<other files>` are the other files you are required to submit. Make sure that the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files are present in the directory along with all of your .java source files.

5. 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.

6. 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.

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.