Project 5

Computer Science 2334
Spring 2014

This project is individual work. Each student must complete this assignment independently.

User Request:

"Expand GeoGrapher by adding neighborhood search and display features and exception handling."

Milestones:

1	Create appropriate classes, variables, and/or methods to deal with the concept of geographic neighborhoods, as described below.	*10 points*
2	Create appropriate classes, variables, and/or methods to recursively search a geographic region's neighborhood for nearby geographic regions, as described below.	*25 points*
3	Create appropriate classes, complete with variables and methods, for the new views described below.	*15 points*
4	Revise as necessary the classes, including variables and methods, of the existing model, controller, and views to allow them to correctly interact with the new methods and views.	*10 points*
5	Improve the robustness of your code by adding exception handling as described below.	*15 points*
►	Develop and use a proper design.	*15 points*
►	Use proper documentation and formatting.	*10 points*

Description:

An important skill in software design is extending the work you have done previously. For this project you will build on Project 4 in order to add new functionality related to the concept of geographic "neighborhoods" and improving the exception handling for I/O.

Geographic Coordinates:

For this project, we are going to refer to any geographic region for which latitude, longitude, and elevation are given as having *geographic coordinates* or just *coordinates* and to the region itself as a *coordinate region*.

Geographic Neighborhoods:

For this project, we are going to define a region's geographic neighborhood using latitude and/or longitude. For any coordinate region c, its *latitude neighborhood* N_{lat} of breadth b is the set of all coordinate regions s for which the absolute value of the difference of latitudes between c and s is less than b. That is:

$$N_{lat}(c, b) = \{s : |lat(c) - lat(s)| < b\}$$

Correspondingly, for any coordinate region c, its *longitude neighborhood* N_{long} of length l is the set of all coordinate regions s for which the absolute value of the difference of longitudes between c and s is less than l. That is:

$$N_{long}(c, l) = \{s : |long(c) - long(s)| < l\}$$

Similarly, for any coordinate region c, its *latlong neighborhood* $N_{latlong}$ of breadth b and length l is the set

of all coordinate regions *s* for which the absolute value of the difference of latitudes between *c* and *s* is less than *b* and the absolute value of the difference in longitudes between *c* and *s* is less than *l*. That is:

$$N_{latlong}(c, b, l) = \{s : (|lat(c) - lat(s)| < b) \wedge (|long(c) - long(s)| < l)\}$$

Additionally, we are going to define a region's recursive latitude, longitude, and latlong neighborhoods. For any coordinate region *c*, its *recursive latitude neighborhood* RN_{lat} of breadth *b* is the set of all coordinate regions *s* for which the absolute value of the difference of latitudes between *c* and *s* is less than *b* together with the set of all coordinate regions *t* for which the absolute value of the difference of latitudes between *s* and *t* is less than *b* for some coordinate region *s* in the recursive latitude neighborhood of *c* with breadth *b*. That is:

$$RN_{lat}(c, b) = \{s : |lat(c) - lat(s)| < b\} \cup \{t : (|lat(s) - lat(t)| < b) \wedge (s \in RN_{lat}(c, b))\}$$

A region's *recursive longitude neighborhood* RN_{long} and *recursive latlong neighborhood* $RN_{latlong}$ are defined analogously.

Models, Views, and Controllers:

As with Project 4, this project will be organized around the model, view, controller (MVC) design pattern which gives us a way to organize code involving graphical user interfaces (GUIs). In particular, you will create one or more models to hold the application data and extend them to act as sources for communicating with views, multiple views to display and manipulate different aspects of the data, and one or more controllers to moderate between user gestures and the model(s) and views.

The models, views, and controllers described in Project 4 all need to be present and functioning in Project 5. See Project 4 for a description of those elements. For this program you could reuse some or all of the classes that you developed for your Project 4, although you are not required to do so. If your Project 4 was designed and implemented well, you should be able to use those classes with little or no modification. If your project 4 had significant design or implementation problems, you will need to significantly change that code as well as added new code for the new functionality. Note that if you did poorly on Project 4, this gives you a chance to redo your Project 4 and potentially gain credit for points you missed previously.

In addition to the Project 4 features, Project 5 will add new features related to geographic neighborhoods. In particular, six new views will be added, and the menus of the Selection View will be modified.

These new views are described below. These are all considered Display Views, in the terminology of Project 4, and should follow the standard behaviors described there for Display Views. Note that all of the Display Views should persist until closed by the user or until the data they display has been cleared from the system. While they are open, they will not interfere with accessing other views, whether Input Views or other Display Views. If the user modifies data in the model(s) while a Display View is open, that Display View should update itself if the data relevant to it has been changed.

Geographic Neighborhood List View:

The *Geographic Neighborhood List View* will be a simple list of all of the coordinate regions within a given geographic neighborhood. This view will be tied to a new menu item in a new menu in the menu bar. The new menu will be labeled "Neighborhoods" and the new menu item will be labeled "Neighborhood List." When this menu item is selected, the user will be asked to select from a list of coordinate regions in the system. Then the user will be asked to enter a breadth and length for the neighborhood to view. If the user only enters a breadth but not a length, then the *Geographic Neighborhood List View* will display a list corresponding to the latitude neighborhood specified by the

user. If the user only enters a length but not a breadth, then the *Geographic Neighborhood List View* will display a list corresponding to the longitude neighborhood specified by the user. If the user enters both a breadth and a length, then the *Geographic Neighborhood List View* will display a list corresponding to the latlong neighborhood specified by the user.

Geographic Neighborhood Check View:

The *Geographic Neighborhood Check View* will give the user a way to easily check whether two coordinate regions are within some geographic neighborhood of one another. This view will be tied to a new menu item in the new "Neighborhood" menu described above under *Geographic Neighborhood List View*. The new menu item will be labeled "Geographic Neighborhood Check." When this menu item is selected, the user will be asked to select two coordinate regions from a list of coordinate regions in the system. Then the user will be asked to enter a breadth and length for the neighborhood to check. As with the *Geographic Neighborhood List View*, the type of neighborhood (latitude, longitude, or latlong) will be determined by whether the user enters a breadth, and length, or both (respectively). The *Geographic Neighborhood Check View* will then display a message to the user corresponding to whether the two selected coordinate regions are within the geographic neighborhood specified by the user.

Geographic Neighborhood Map View:

The *Geographic Neighborhood Map View* will ask for the same input from the user as the *Geographic Neighborhood List View*. However, instead of listing the coordinate regions within the geographic region, it will map them. This map will be similar to the maps in the map views from Project 4 but will plot on the map all of the coordinate regions within the geographic neighborhood and will, in addition, plot lines showing the borders of the geographic neighborhood. This view will be tied to a new menu item in the Graph Menu from Project 4. This menu item will be labeled "Geographic Neighborhood."

Recursive Views:

For each Display View described above, there will be a corresponding recursive version labeled "Recursive _____" (where the blank corresponds to the label for the non-recursive version). In the recursive version, GeoGrapher will use the appropriate recursive geographic neighborhood based for the menu item chosen and data entered. Note that, for the *Recursive Geographic Neighborhood Map View*, GeoGrapher will draw borders around each neighborhood.

Exception Handling:

Because I/O errors are likely to occur without warning, all of your I/O routines should be thoroughly wrapped with exception handling routines. Note that this does not mean simply re-throwing all possible I/O exceptions. Instead, you should consider the possible I/O exceptions individually and write appropriate exception handling routines to provide for graceful handing of different exception types. For example, a `FileNotFoundException` could be handled by asking the user to select an alternate file or select not to do I/O at this time.

How to Complete this Project:

1 Create figures, on engineering paper or using drawing software, to show approximately what each view will contain. These figures do not need to exactly match the appearance of the final windows but should contain all major components and show their basic layout.

2 During the lab session and in the week following, you should work ~~with your partner(s)~~ to determine the classes, variables, and methods needed for this project and their relationship to one another. This will be your preliminary design for your software.

2.1 Make a list of the nouns you find in the project description that relate to items of interest to the "customer." Mark these nouns as either more important or less important. More important nouns describing the items of interest to the "customer" should probably be incorporated into your project as classes and objects of those classes. Less important nouns should probably be incorporated as variables of the classes/objects just described. This list will be turned in with your preliminary and final designs long with your other design documents.

2.2 Make a list of the verbs you find in the project description that relate to items of interest to the "customer." Verbs describing behaviors of the desired objects and the systems as a whole should probably be incorporated into your project as methods. This list will be turned in with your preliminary and final designs long with your other design documents.

2.3 Be sure to use UML class diagrams as tools to help you with the design process.

3 Once you have completed your UML design, create Java "stub code" for the classes and methods specified in your design. Stub code is the translation of UML class diagrams into code. It will contain code elements for class, variable, and method names; relationships between classes such as inheritance, composition, and aggregation as appropriate; variable types; and method signatures. Stub code does not, however, contain method bodies (except for return statements for methods that return values or object references – these should return placeholders such as `null`). Because we are using object-oriented design, in which the inner workings of one class are largely irrelevant to the classes with which it interfaces (that is, we are using encapsulation), we do not need to complete the implementation of any class until the design is complete.

4 Add comments to your stubbed code as specified in the documentation requirements posted on the class website. Run your commented stubbed code through Javadoc as described in the Lab 2 slides. This will create a set of HTML files in a directory named "docs" under your project directory.

5 Create unit tests using JUnit for all of the non-trivial units (methods) specified in your design. There should be at least one test per non-trivial unit and units with many possible categories of input and output should test each category. (For example, if you have a method that takes an argument of type `int` and behaves differently based on the value of that `int`, you might consider testing it with a large positive `int`, and small positive `int`, zero, a small negative `int`, and a large negative `int` as these are all likely to test different aspects of the method.)

6 At the end of the first week (see ***Due Dates and Notes***, below), you will turn in your preliminary design documents including your view figures, which the TA will grade and return to you with helpful feedback on your preliminary design. Please note: You are encouraged to work with the instructor and the TAs during their office hours during the design week to get feedback throughout the design process as needed.

Final Design and Completed Project

7 Using feedback from the instructor and TAs as well as your own (continually improving) understanding of OO design, revise your preliminary UML design.

8 Make corresponding changes to your stub code, including its comments.

9 Make corresponding changes to your unit tests.

10 Implement the design you have developed by coding each method you have defined. If you find that your design does not allow for the implementation of all methods, repeat steps 5 and 6.

11 Test each unit as it is implemented and fix any bugs.

12 Test the overall program and fix any bugs.

13 Once you have completed the project and are ready to submit it for grading, create a new set of Javadoc files using Eclipse and inspect them to make sure your final design is properly documented in your source code.

14 Submit all parts of your completed project. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

Extra Credit Features:

You may extend this project with more features for an extra 5 points of credit. For example, you could think of ways to present the user with helpful information about the program, such as a context-dependent help system.

To receive the full five points of extra credit, your extended features must be novel (unique) and must involve effort in the design of the extra features and their integration into the project and the actual coding of the features. Also, you must indicate on your final UML design which portions of the design support the extra feature(s); and you must include a write-up of the feature(s) in your milestones.txt file. The write-up must indicate what each feature is, how it works, how it is unique, and the write-up must cite any outside resources used.

Due Dates and Notes:

The electronic copy of your revised design including view figures, Noun/Verb List, UML, stub code, and detailed Javadoc is due on **Wednesday, April 23rd**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**. Submit your view figures on *engineering paper* or hardcopies made using drawing software, Noun/Verb List, revised UML design on *engineering paper* or a hardcopy made using UML layout software, and a hardcopy of your cover page at the **beginning of lab on Thursday, April 24th**.

The electronic copy of the final version of the project is due on **Wednesday, April 30th**. Submit the project archive following the steps given in the submission instructions **by 10:00pm**. Submit your final view figures on *engineering paper* or hardcopies made using drawing software, final UML design on *engineering paper* or a hardcopy made using UML layout software, a hardcopy of the cover page for your project, and a hardcopy of the milestones.txt file at the **beginning of lab on Thursday, May 1st**.

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.

When zipping your project (or design) for submission, be sure to follow the instructions carefully. In particular, *before* zipping the project be sure to

- **place additional files (such as UML diagrams, cover page, and milestones files) within the "docs" directory inside your Eclipse folder for the given project and be sure that Eclipse sees these files (look in the Package Explorer and hit Refresh if necessary), and**
- **rename the project folder to your 4x4. Note that renaming the project folder to your 4x4 *before* zipping is not the same thing as naming the zip file with your 4x4. The latter is fine; the former is *mandatory*.**