# Project #4

*Computer Science 2334*
*Spring 2014*

## *User Request:*

*"Create a geographic data system **with a Graphical User Interface**."*

## *Milestones:*

1. Create appropriate classes, complete with data fields and methods, to handle application data on geographic regions as described below under Model. *15 points*
2. Add a class, complete with data fields and methods, to the model to allow the model to correctly interact with the controller and the views. *10 points*
3. Create appropriate classes, complete with data fields and methods, for the views described below. *25 points*
4. Create an appropriate class, complete with data fields and methods, for the controller as described below. *25 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 rework Project #3 in order to handle information pertaining to geographic regions and allow users to view and manipulate that information graphically. 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 a single model to hold the data, several views to display and manipulate different aspects of the data, and one controller to moderate between user gestures and the model and views. For this program you may reuse some of the classes that you developed for your previous projects, although you are not required to do so. Note that much of the code you write for this program could be reused in more complex applications.

Model:

You will create a model class called "**RegionModel**." Models in the version of the MVC design pattern we have used in class contain data and methods for the application objects being modeled (which are all related to geographic regions in this project) as well as data and methods to allow the model to interact with views. Your model class will follow this version of the MVC design pattern.

For the application objects, you will have classes to represent the following kinds of geographic regions: continents, countries, cities, places of interest, and points of interest. Note that you have represented most of these application objects in previous projects, although you will need to modify your classes for those objects at least slightly in order to satisfy the requirements of this project. (You may, of course, modify your classes substantially for this project, if doing so would substantially improve your project.)

One difference between the geographic regions as described in previous projects and this project is that a place of interest may be specified as belonging to a continent, rather than one or more countries. So, for example, one could specify the Transantarctic Mountains as a place of interest on the continent of

Antarctica. Similarly, a user could choose to list the Rocky Mountains as belonging to either North America or to Canada and the United States. Also, a small place of interest could be specified as belonging to a city, rather to a continent or to one or more countries.

Going even smaller, a new type of application object you need to represent in this project is call a "point of interest." A *point of interest* is a geographic feature that has an area less than ½ square mile (i.e., that would round off to 0 square miles), so it is too small to meet our definition of a place of interest. Example points of interest would be buildings of various sorts, monuments, and small parks. For each point of interest, your system will keep track of its name; a brief description of the its type (e.g,. "museum" or "city park"); its latitude, longitude, and elevation; and the region in which it is located. The region in which it is located may be any of the existing region types – continent, country, city, or place of interest – or an existing point of interest (e.g., a monument in a city park).

As you are designing or revising the classes for these application objects, please note that it will be useful in this project (and in Project 5) to go from objects of one type to another and back again using references without having to search through lists or hashmaps. So, for example, you should consider not only having a list of countries within each continent but also a reference from each country back to its continent. With points of interest, you'll want to consider having a list of points of interest within each region class and allowing a point of interest to refer to an enclosing region of any region classes.

The data for this model will enter the system through three alternate ways: (1) It may be read in from a text file, which we will refer to as *importing* the data. (2) It may be read in using object I/O, which will will refer to as *loading* the data. (3) It may be entered by a person using the input views described below, which we will refer as *entering* the data.

Whether data is imported or entered, it must meet the following sanity checks: (1) A continent cannot contain counties with a combined area (or population) larger than the area (or population) of the continent itself. (2) A country cannot contain cities with a combined area (or population) larger than the area (or population) of the country itself. (3) A place of interest cannot have an area larger than the region(s) to which it belongs. (4) A point of interest cannot have a latitude (or longitude or elevation) different from the latitude (or longitude or elevation) of a city or point of interest to which it belongs. If invalid data is found during import or entry, the user will be informed of the error and the model will not be updated.

To interact with views, the **RegionModel** class will have variables and methods akin to those from the examples we have seen in MVC lectures and labs. In particular, when new information is added to the model by importing, loading, or entering new data, any relevant display views should be notified so that they may update themselves to reflect the new data.

Views:

Producing views of information can be very useful to users. Therefore your program will create and maintain several views of the data, as described below.

*Selection View:*

The *Selection View* will be displayed as soon as the program is started. There will be a title in the top bar that says "GeoGrapher." In addition, there will be a menu bar with a file menu, a graph menu, and, in the content pane of the window, five vertical lists displayed side by side each with its own title above it and set of buttons below it.

Content Pane:

Within the content pane of the selection view, the five vertical lists will be organized from left to right. The first list will be titled “Continents,” the second list will be titled “Countries,” the third list will be titled “Cities,” the fourth list will be titled “Places of Interest,” and the fifth list will be entitled “Points of Interest.” Below each list will be a set of buttons labeled “Add, “Edit,” and “Delete.”

Initially, all lists will be empty and most buttons will be grayed out and inactive. The only initially active button will be the Add button below the continent list. The lists will be kept in sync with the underlying model information on geographic regions and the buttons will become active and inactive as appropriate. For example, once there is at least one continent, the Edit and Delete buttons below the continent list would be appropriate and should become active, but each Edit and Delete button below each other list should remain inactive until that list contains at least one entry. Similarly, some menu items in the file menu will initially be grayed out and inactive.

If the user clicks the Add button below the continent list, an add continent dialog will pop up to ask the user for the continent's name, population, and area. The controller will take the information entered into this dialog and pass it along to the model, asking the model to add the continent. However, note that the model will not allow duplicate continent names to be added and the user should be notified by the view if any such data entry error is made. Once there is at least one continent in the model, the Add button below the country list will become active (as well as the Edit and Delete buttons for continents, as mentioned previously).

If the user selects a continent on the continent list and clicks the Edit button below the continent list, then an edit dialog will pop up to allow the user to edit the existing data on the selected continent. This dialog will be similar to the add continent dialog, except that the fields will be pre-populated with the data currently found in the model.

If the user selects more than one continent from the continent list and clicks the Edit button below the continent list, then a series of edit dialogs will pop up, one for each selected continent.

If the user selects one or more continents on the continent list and clicks the Delete button below the continent list, then a dialog will pop up to confirm whether the user wants to delete the selected continent(s) and its (their) data. Note that if a continent is deleted from the system, all countries on that continent should be deleted from the system and, in turn, all cities in the deleted countries should be deleted from the system and each place of interest that is located in a deleted country should have that country deleted from its list of countries in which it is located. If a place of interest ever has its last associated country deleted, that place of interest should also be deleted. Also, any point of interest in a deleted continent, country, city, or place of interest should be deleted and any point of interest in a deleted point of interest should be deleted.

The country list and buttons for adding, editing, and deleting countries will behave similarly to the continent list and buttons with respect to their data with the exceptions that (1) there must be at least one continent in the system before countries may be added, (2) to specify the continent on which a country is located, GeoGrapher will provide the user with a list of continents in the system and the user will select one continent from that list, and (3) once the first country has been added to the system, the buttons for adding cities and places of interest will become active.

The city list and buttons for adding, editing, and deleting cities will behave similarly to the country list and buttons with respect to their data but note that the dialogs must also account for latitude, longitude, and elevation for cities. Also note that (1) there must be at least one continent and one country in the system before cities may be added, (2) to specify the country in which a city is located, GeoGrapher will

provide the user with a list of countries in the system and the user will select one country from that list, and (3) once the first city (or point of interest) has been added to the system, the map items in the graphing menu will become active.

The places of interest list and buttons for adding, editing, and deleting places of interest will behave similarly to the city list and buttons with respect to their data but note that the data fields differ. Also note that (1) there must be at least one continent in the system before places of interest may be added, (2) the user should be asked whether to add the place of interest to a continent, one or more countries, or a city, and (3) to specify the region in which a place of interest is located, GeoGrapher will provide the user with a list of the regions of the chosen type currently found in the system and the user will select from that list.

The points of interest list and buttons for adding, editing, and deleting points of interest will behave similarly to the places of interest list and buttons with respect to their data but note that the data fields differ. Also note that points of interest can be added to any region type.

File Menu:

The file menu will be marked "File" and have entries for "Load Geography," "Save Geography," "Import Geography," and "Export Geography." Initially, only Load Geography, and Import Geography will be active. Save Geography and Export Geography will be grayed out and inactive until at least one continent has been added through the GUI or an existing one has been read in using Load Geography or Import Geography. As with the buttons, the menu items should only be active when executing the action would be appropriate. (For example, it makes no sense for the user to save or export geographic data if none is present.)

If the user chooses Load Geography or Import Geography and there is unsaved data, your program will pop up a dialog box asking the user whether that data should be saved, exported, or discarded. The actions taken by your program in response to this save/export/discard dialog box should be to save, export, or discard this data, as specified by the user. Once there is no unsaved data in your system, your program will present the user with a file picker and allow him or her to select one or more files to read in via object input or text input, as appropriate. Once the data is read in, if it contains continents, the continent list will be populated with continents from the model and the Edit and Delete buttons for continents will become active, and similarly for the other types of regions. Note that, for text input files, GeoGrapher should ask for the name of the continent file first, and only move on to ask for the name of the country file once there is at least one continent in the system. It should approach cities, places of interest, and points of interest similarly.

Once geographic data is present in the system, the Save Geography and Export Geography menu items will become active. If the user chooses Save Geography or Export Geography, your program will present the user with a file picker and allow him or her to designate one or more files for writing via object output or text output, as appropriate.

For each inactive button or menu item, if the user hovers the pointer over that component, a tool tip will appear to let the user know why the button is not currently active. (For example, "Cannot delete continent(s); empty continent list" or "Cannot delete continent(s); no continent(s) selected.")

Graph Menu and Corresponding Display Views:

The graph menu on the Selection View will be marked "Graph" and have entries for "Simple Bar Chart," "Stacked Bar Chart," and "Map."

Simple Bar Chart will be a sub-menu with entries for "Area" and "Population." In turn, Area and

Population will be sub-menus, each with menu items for continents, countries, cities, and places of interest. When a user chooses continents, countries, cities, or places of interest from one of these menus, GeoGrapher will display to the user a list of the appropriate region type and ask the user to select one or more regions for graphing. Once the user has selected regions from the list, a bar chart will be displayed that is similar to the ones described in Project 3 for "all continents," "all countries," "all cities," and "all places of interest," except that the user will be selecting particular regions of the given type for display, rather than necessarily displaying all of those regions.

Stacked Bar Chart will be similar to Simple Bar Chart in terms of sub-menus, except that the lowest level menu items will not be continents, countries, cities, and places of interest but will, instead, be "countries within continents," "cities within countries," "places of interest within continents," "places of interest within countries," and "places of interest within cities." If the user selects "countries within continents," GeoGrapher will display to the user a list of continents from which the user can choose one or more. For each continent the user chooses, GeoGrapher will display a list of countries within that continent that the user can choose. After the user has chosen one or more countries from each continent chosen, a bar chart will be displayed. This bar chart will be a stacked bar chart, containing one stacked bar, like those described in Project 3 for "all countries within a given continent" except that each stacked bar will only contain the countries selected by the user, rather than necessarily displaying all countries in that continent, and there will be one stacked bar for each continent selected with the stacked bars arranged from tallest stacked bar on the left to shortest stacked bar on the right. If the user selects "cities within countries," "places of interest within continents," "places of interest within countries," or "places of interest within cities," GeoGrapher will behave similarly albeit with the listed region types instead.

Map will have menu items for "cities within countries," "cities within continents," "cities worldwide," "points of interest within cities," "points of interest within countries," "points of interest within continents," and "points of interest worldwide." If the user selects "cities within countries," GeoGrapher will display to the user a list of countries from which the user can choose one or more. For each country that the user chooses, GeoGrapher will display display a list of cities from which the user can choose one or more. After the user has chosen one or more cities from each country chosen, GeoGrapher will display a map similar to those described in Project 3 for mapping or "all cities within a given country" except that the map will only contain those cities chosen by the user, rather than necessarily all cities within a country, and the cities may be spread over more than one country. If the user selects "cities within continents," "cities worldwide," "points of interest within cities," "points of interest within countries," "points of interest within continents," or "points of interest worldwide," GeoGrapher will behave similarly albeit with the listed region type(s) instead.

*Notes on Views:*

Note that there will be one Selection View for your program. It will be opened when your program runs. When the user closes this view, your program should exit. In contrast, there may be many Display Views open at any given time. Each time the user selects a graph menu item, a new Display View should open. Each Display View can be closed independently by the user by closing the window in which it resides. Additionally, if all of the regions with which a particular Display View is associated are deleted, the view should automatically close itself.

Controller

Besides at least one model and at least one view, every GUI-based program using the MVC design pattern needs to have at least one controller. The controller is responsible for connecting the model(s) to the view(s) so that appropriate actions are taken in response to user gestures and that appropriate representations of data are presented to users.

For this project, the appropriate actions for user gestures and appropriate representations of data are described above, under the individual views. The controller that you create for this project, then, will need to ensure that the actions are taken and views updated as described above. Call this controller **RegionController**.

### *Text Input and Output*

The format of the text files containing data will be the same as for Project 3. This format should be used for both importing and exporting data.

### *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 relationships 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 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. A good approach to the implementation of your project is to follow the project's milestones in the order they have been supplied. If you find that your design does not allow for the implementation of all methods, repeat steps 7, 8, and 9.

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 preliminary design (including **view figures**, Noun/Verb List, UML, stub code, detailed Javadoc documentation, and unit tests) is due on **Wednesday, April 2nd**. 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 3rd**.

The electronic copy of the final version of the project is due on **Wednesday, April 16th**. 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, April 17th**.

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.

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 on the cover sheet. 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 cover sheet, 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, your cover sheet must clearly indicate this division of labor. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.

**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 sheets, 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),**
- **compress all files into a .zip format. The formats .rar and .7z will no longer be accepted. Also, when submitting the initial design make sure that the UML is in one of the following formats: .png .jpg or .pdf. Custom formats such as .uml or .dia are NOT acceptable. If you are unsure how to export the file in that format, take a screen-shot of the diagram and attach that, and**
- **rename the project folder to the 4x4 of the team member submitting the project. 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*.**