Project 5

Computer Science 2334
Spring 2011

User Request:

"Expand on the Census Information System by Adding Apportionment and Exception Handling."

Milestones:

1	Create appropriate classes, complete with variables and methods, for a recursive method to compute apportionment as described below.	*15 points*
2	Create appropriate classes, complete with variables and methods, for an iterative method to compute apportionment as described below.	*15 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 model and controller to allow them to correctly interact with the new methods and views.	*15 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 apportionment of Representatives in the US House of Representatives based on state population. In particular, you will add both recursive and iterative methods for calculating apportionment and two new views to allow users to view nationwide population and apportionment data graphically. You will also add new views to view population by income bracket at the state and national levels, as well as the county level. You will also add exception handling to your code to make it more robust. In addition, you will have the opportunity to revise your code from Project 4 to ensure that it is implemented correctly.

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. For this program you should 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.

Apportionment:

Apportionment of US Representatives is based on population. However, the calculation is somewhat

more complex than one might imagine. This is because there is a minimum number of Representatives for each state (one) and because "rounding" fractional representation one way or another can have a significant effect on a state's representation, so simply dividing the number of representatives by the population percentage of the state is not used. Rather than spell out here the method used, you are required to research the method yourself. (The amount of research should be minimal. When last I checked, Wikipedia did a sufficient job of expressing this information.) Once you have researched the apportionment method, you will need to create both a recursive and an iterative method for calculating apportionment. Note that these methods should return identical apportionments. Note also that you will need to include the information source(s) you selected in your code comments, Javadoc, and milestones.

Models, Views, and Controllers:

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. In addition, four new views will be added and the menu structure of the Selection View will be modified. These new and changed views are described below.

State Income View:

The *State Income View* will be similar to the Pie Chart View from Project 4 but will graph income data at the state level, rather than at the county level. Since this data is not given directly in the data sources, your program will need to calculate the numbers at the state level by summing the numbers at the county level. (Note that it might be a good idea to rename Pie Chart View to "County Income View" to avoid confusion with the State Income View or other views using pie charts as described below. For this, consider using the refactoring capabilities of Eclipse.)

National Income View:

The *National Income View* will be similar to the State Income View described above but will graph income data at the national level, rather than at the state level. Since this data is not given directly in the data sources, your program will need to calculate the numbers at the national level by summing the numbers at the state level.

National-State Population View:

The *National-State Population View* will be similar to the *National Income View* in that it will display a pie chart. However, this pie chart will not be of income brackets but of state populations. That is, there will be up to 50 slices in the pie, one for each state, and the width of each slice will be based on that state's population. Again, since this data is not given directly in the data sources, your program will need to calculate it from the numbers you have available.

Apportionment View:

The *Apportionment View* will be similar to the *National-State Population View* in that it will display a pie chart. However, this pie chart will not be of state populations but of state representation in the US House of Representatives based on the apportionment calculations done elsewhere in your program. That is, there will be up to 50 slices in the pie, one for each state, and the width of each slice will be based on that state's representation in the US House. Note that if representation were calculated as a direct proportion of population, this pie chart would have slices exactly as wide as those of the *National-State Population View*. However, because representation is based on a more complex formula, comparing this graph with the *National-State Population View* will allow users to see the difference between direct proportions and the method actually in use.

Selection View:

The Selection View will remain largely unchanged, although its menu structure will change in two ways.

First, to request an apportionment calculation, a new menu item with two submenu items will be added to the Data menu of the Selection View. The new menu item will be labeled "Apportion" and the submenu items will be labeled "Recursive" and "Iterative."

The second change to the Selection View menu structure has to do with graphing the data. For this you will add a "Graph" menu to the menu bar. The Graph menu will contain menu items labeled "Selected County Income," "Selected State Income," and "National Income," "National Population by State," and "Apportionment." Relatedly, you will remove the "Graph" menu item from the Data menu, as its function has been subsumed by the Selected County Income menu item of the Graph menu.

The functions of all the new and changed menu and submenu items should be inferable from their names. These menu and submenu items, like others in your program, should only be active when meaningful. In your milestones, you should explain under which conditions each will be active.

Persistence of Display Views:

Note that all of the graph views will persist until closed by the user. While they are open, they will not interfere with accessing other views, whether data entry views or other display views. If the user modifies data in the model(s) while a graph view is open, that graph view should update itself if the data relevant to it has been changed.

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 Be sure to look for nouns in the project description. 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.

2.2 Be sure to look for verbs in the project description. Verbs describing behaviors of the desired objects and the systems as a whole should probably be incorporated into your project as methods.

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. 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 the classes until the design is completed.

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 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

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

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

8 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 5 and 6.

9 Test your program and fix any bugs.

10 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.

11 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:

An electronic copy of your revised design including **view figures**, UML, stub code, and detailed

Javadoc is due on **Wednesday, April 27th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your **view figures on *engineering paper* or hardcopies made using drawing software**, revised UML design on *engineering paper* or a hardcopy made using UML layout software, a hardcopy of your stub source code, and a hardcopy of your cover page at the **beginning of lab on Thursday, April 28th**.

An electronic copy of the final version of the project is due on **Wednesday, May 4th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. 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 5th**.

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), 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*.

Also, you must ensure that there is no personally identifying information (such as name or student ID number) anywhere within the project materials themselves, including in page headers and footers, except on the cover sheet.