Project 4

Computer Science 2334
Spring 2011

User Request:

"Create a Census Information System with a Graphical User Interface."

Milestones:

1. Create appropriate classes, complete with variables and methods, to handle the application data on particular census information as described below under model. *15 points*
2. Add a class, complete with data 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 variables and methods, for the views described below. *25 points*
4. Create an appropriate class, complete with variables 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 the US Census 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 "**CensusModel**." 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 US Census data 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 create a class to represent a single Census. This Census will consist of a year (such as "2000") and a list of states[1]. In turn, each state will consist of a name[2] and a list of places (such as counties or urban areas). Each place will consist of a name (such as "Cleveland County") and a collection of data. In this assignment the collection of data for each place will be a count of the number of households in that place that fall into each of several different income brackets.

1. As with Project 3, we will use the term "state" throughout this assignment to refer to the 50 states of the USA, four of which are officially known as commonwealths, plus its territories and the District of Columbia.
2 Unlike Project 3, the state names in this assignment will actually be spelled out as names, rather than two-letter postal abbreviations.

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.

To interact with views, the **CensusModel** 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. It will initially consist of a title in the top bar that says "<Year> Census" where <Year> is replaced by the year of the Census (e.g., the top bar might say "2000 Census"), a menu bar with a file menu and a data menu, and, in the content pane of the window, two vertical lists displayed side by side each with its own title above it and set of buttons below it. The list on the left will be titled "State" and the list on the right will be titled "County." The buttons below the state list will be "New State," "Delete State," and "Delete All States." The buttons below the county list will be "New County," "Delete County," and "Delete All Counties." Initially, both lists will be empty and all buttons will be grayed out and inactive. The lists will be kept in sync with the underlying model information on states and counties, respectively, and the buttons will become active and inactive as appropriate. (That is, once there is a Census at all, the New State button would be appropriate and should become active, but the Delete State and Delete All States buttons should not be active unless the state list contains at least one entry. Similarly, once there is at least one state in the state list, when that state is selected its counties, if any, should appear in the county list and the New County button should become active. And so on.) Similarly, the data menu and some menu items in the file menu will initially be grayed out and inactive.

The file menu will have entries for "New Census," "Load Census," "Save Census," "Import Census," and "Export Census." Initially, only New Census, Load Census, and Import Census will be active. Save Census and Export Census will be grayed out and inactive until a new Census has been created using New Census or an existing one has been read in using Load Census or Import Census. 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 a Census if one has not been created, loaded, or imported.)

If the user selects New Census and there is no unsaved data in the system (i.e., this is the first time since your application was started that the user has selected New Census and the user has not loaded or imported an existing Census), your program will pop up a dialog box asking the user for a year and initialize the Census object to have that value. If, on the other hand, there is data in the system that has not yet been saved, it will pop up a dialog box asking the user whether that data should be saved, exported, or discarded. If the user chooses to save the data, your program will save it using object I/O. If the user chooses to export the data, your program will export it to a text file. If the user chooses to discard the data, the Census object should be cleared. Once there is no more unsaved census data in the system, your program will proceed to the dialog box asking the user for a year and initialize the Census object to have that value, as described previously.

If the user chooses Load Census or Import Census 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, as it did for the New Census option. The actions taken by your program in response to this save/export/discard dialog box should be the same as those for the New Census save/export/discard dialog. 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 an appropriate file to read in via object input or text input, as appropriate.

Once a new census has been created or an existing one read in from a file, the Save Census and Export Census menu items will become active. If the user chooses Save Census or Export Census, your program will present the user with a file picker and allow him or her to designate an appropriate file for writing via object output or text output, as appropriate.

Also once a new census has been created or an existing one read in from a file, the Add State button under the state list will become active. Additionally, if a Census is read in from a file and it contains states, the state list will be populated with states from the model and the Delete States and Delete All States buttons will likewise become active. If the user selects a state from the state list, the county list will be populated with all of the counties for that state that are in the model, the New County button will become active, and if the county list is not empty the Delete County and Delete All Counties buttons will become active as well.

If the user selects Add State, a dialog will pop up to ask the user for a new state name. The controller will take any name entered into this dialog and pass it along to the model, asking the model to add the state. (However, note that the model will not allow duplicate state names to be added and the user should be notified by the view if any such data entry error is made.)

If the user selects Delete State and a state is currently selected in the state list, then a dialog will pop up to confirm that the user wants to delete the selected state and its counties and data, if any. If the user selects Delete State but a state is not currently selected in the state list, then a dialog will pop up to ask the user for the name of the state to delete.

If the user selects Delete All States, then a dialog will pop up to confirm that the user wants to delete all states from the Census.

The county list and buttons will behave similarly to the state list and buttons with respect to their data with the exception that when a county is selected in the county list, the data menu will become active.

The data menu will contain menu items labeled “Graph,” “Enter/Modify,” and “Clear.” If the user selects Graph, the Pie Chart View (described below) will appear. If the user selects Enter/Modify, the Data Entry View (described below) will appear.

Pie Chart View:

If the user chooses Graph from the data menu, your program will open up a Pie Chart View for the place selected in the Selection View. The Pie Chart View have a label across the top that displays the county name and state. Below that on the left will be a pie chart that shows the percentage of the population of that county whose income falls into each of the income categories found in the data file. Each slice of the pie will be easily distinguished from either other slice by color(s) and/or pattern(s). To the right of the pie chart itself will be a key that shows which color pattern corresponds to each income category.

Data Entry View:

If the user chooses Enter/Modify, the *Data Entry View* will appear. As with the Pie Chart View, the Data Entry View have a label across the top that displays the county name and state. Below that will be a

table with two columns. The left column will list the income categories for the Census using labels. There will be 16 such categories: Less than $10;000, $10;000 to $14;999, $15;000 to $19;999, $20;000 to $24;999, $25;000 to $29;999, $30;000 to $34;999, $35;000 to $39;999, $40;000 to $44;999, $45;000 to $49;999, $50;000 to $59;999, $60;000 to $74;999, $75;000 to $99;999, $100;000 to $124;999, $125;000 to $149;999, $150;000 to $199;999, and $200;000 or more. The right column will have text fields into which the number of households in the corresponding income category can be entered. These will be kept in sync with the data from the model for the county and state named at the top of the view.

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 Pie Chart Views and/or Data Entry Views open at any given time. Each time the user selects Graph from the data menu a new Pie Chart View should open for the state and county currently selected in the Selection View, unless there is already a Pie Chart View open for that place. Similarly, each time the user selects Enter/Modify from the data menu a new Data Entry View should open for the state and county currently selected in the Selection View, unless there is already a Data Entry View open for that place. Each Pie Chart View and Data Entry View can be closed independently by the user by closing the window in which it resides. Additionally, if a county with which a Pie Chart View or Data Entry View is associated is 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 **CensusController**.

Text Input and Output

The format of the text file containing data can be inferred by looking at the sample text data file provided along with this assignment. 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 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 6th**. 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 7th**.

An electronic copy of the final version of the project is due on **Wednesday, April 20th**. 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, April 21st**.

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*.