Project 4

Computer Science 2334
Fall 2008

User Request:

"Create a Word Frequency Display System
with Import/Export and Load/Save Features and a Graphical User Interface."

Milestones:

1. Create a dictionary to store words and information about them, as done in Project 3, and add to that dictionary information on the frequency of certain features of the word list it contains. In particular, the dictionary will keep a tally of the number of words it contains of different scores, lengths, and languages. *5 points*
2. Create an MVC model as exemplified by the **CircleModel** class from your textbook and as discussed in class. The class for this model will be called "**DictionaryModel**." This model will contain (1) the variables and methods needed to keep track of the dictionary, and (2) the variables and methods necessary to keep track of the views that will listen for changes to the dictionary. *10 points*
3. Create an MVC view associated with the score frequency information in the model to display a histogram of scores, as you did in Project 3. The class for this view will be known as "**ScoreView**." *5 points*
4. Add functionality to the **ScoreView** class to dynamically resize the histogram as the user resizes its enclosing pane. *5 points*
5. Create an MVC view associated with the *word length* frequency information in the model to display a histogram of *word lengths*. The class for this view will be known as "**LengthView**." This histogram should also dynamically resize. *5 points*
6. Create an MVC view associated with the *language* frequency information in the model to display a histogram of *languages*. The class for this view will be known as "**LanguageView**." This histogram should also dynamically resize. *5 points*
7. Implement a simple Graphical User Interface "control panel" using Swing that allows for selecting among the program's functions (load, save, import, export, display views, and filter). The GUI will use buttons, check boxes, and text boxes. *15 points*
8. Use a **JFileChooser** dialog to allow the user to choose the file used when loading, saving, importing, or exporting the dictionary. *5 points*
9. Create an MVC controller called "**DictionaryController**" associated with the model. When the user clicks a button, the controller will tell the model to load, save, import, or export itself. When the user marks a check box, the controller will tell a corresponding view to display itself. When the user enters a filter value into a text boxes, the controller will tell the model to remove from itself words with scores outside the range specified. *15 points*

- ► Develop and use a proper design. *15 points*
- ► Use proper documentation and formatting. *15 points*

Description:

An important skill in software design is extending the work you have done in a previous project. For this project you will rework Project 3, adding a graphical user interface organized around the MVC paradigm. The MVC paradigm gives us a way to organize our code involving graphical data displays and/or user interfaces, particularly GUIs. For this assignment, you will be creating a GUI to view and manipulate data using MVC. In particular, you will be creating a single model to hold the data, three views to display various 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.

For this project, as with your previous projects, you will put together several techniques and concepts learned in CS 1323 and some new techniques to make a new application. This application will graphically display information about a large database of words which we will call a "dictionary" even though it lacks definitions. Note that much of the code you write for this program could be reused in more complex applications, such as a spell-checker or an actual dictionary (with definitions).

Model:
You will create a model class called "**DictionaryModel**." Models in the particular version of the MVC paradigm shown in the Circle example from your textbook contain data and methods for the application objects being modeled (in this case these will be words) as well as data and methods to allow the model to interact with views. Your model class will follow this version of the MVC paradigm.

For the application objects, you will create a dictionary, similar to the one from Project 3. This dictionary will additionally contain information on the frequency of word scores, lengths, and languages.

To interact with views, the **DictionaryModel** class will have variables and methods akin to those from the **CircleModel** class in your textbook. In particular, when words are read into or filtered from the dictionary in the model, the view objects should be notified.

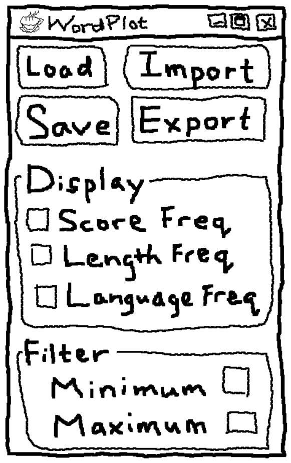
Views:
Producing views of information can be very useful to users. Therefore your program will create and maintain three views of the data: one histogram for word score frequency, one or word length frequency, and one for language frequency called "**ScoreView**," "**LengthView**," and "**LanguageView**," respectively. Each view will be given its own window, which should be a **JFrame**. Most other details of the histogram views are up to you (whether vertical or horizontal, width of the bars, etc.). To make things easy, you may assume that the scores will only range from 1 to 100. However, you must allow the user to resize the window in which each histogram is displayed and ensure that each histogram dynamically resizes along with its associated window in order to make good use of the size of the window.

Controller:
You will create a controller class called "**DictionaryController**" to handle the task of asking the model to update itself in response to user input and selecting views. In particular, the controller will be responsible for the following:

1. When the user asks to load/import (or save/export) a dictionary, the **DictionaryController** will tell the model to read in (write out) a user-specified file.
2. When the user asks to filter the dictionary based on score, the **DictionaryController** will tell the model to remove words with scores beyond the user-specified range.
3. When the user asks to see a view, the **DictionaryController** will tell the corresponding view to display itself.

GUI:

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. This project will be a program of the latter type. This program will support importing, exporting, loading, saving, filtering, and displaying dictionary data. Here you will develop a full-fledged Graphical User Interface (GUI) for dealing with word data, including using the GUI for selecting files, filtering the words, and displaying the data in a graphical format.

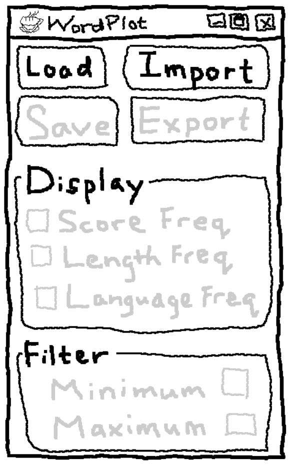


The basic Graphical User Interface for Project 4 will have four buttons, three check boxes, and two text boxes. The four buttons will be load, save, import, and export. The three check boxes will be for turning on (checked) and off (unchecked) the three views (score, length, and language). The two text boxes will be for entering the minimum and maximum scores to be used for word filtering. A rough sketch of this is shown in the figure to the right. (The figure is intentionally rough so that you will not attempt to copy it exactly.)

This main window will be augmented by additional windows that will pop up to help with file selection and to display the three views. In particular, a **JFileChooser** window will pop up to help with file selection for loading, saving, importing, or exporting; a **JFrame** will pop up to display each view.

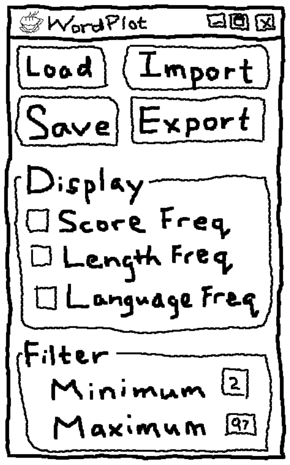
Loading/Saving and Importing/Exporting:

Your software will read and write data files, both text and object data, in the same way as in Project 3. They will use the same data file formats as in Project 3 although there will be no query file or score output file. The only other difference will be that these I/O operations will be driven by user selections through the GUI.



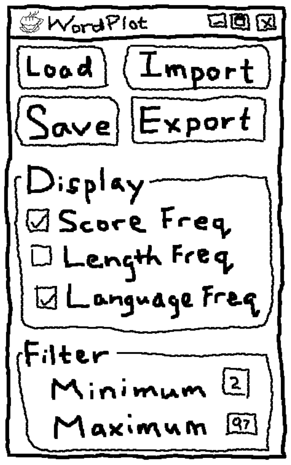
When your program is run initially, it should be clear from looking at the GUI that only load and import are possible. To make this clear, the save and export buttons will be displayed differently (e.g., with the letters "grayed out"). Likewise, until dictionary is loaded or imported, the display view check boxes and filter text boxes will be similarly displayed in such a way that it should be clear that there is nothing that can be done with respect to them. See, for example, the figure at right.

When the load or import button is selected, a **JFileChooser** window will pop up to help with file selection. Once a dictionary has been loaded or imported, the other two buttons (save and export) will become active, as shown at right. These will provide the user with the two corresponding options. (1) The user may choose to save the dictionary, using object output. If the user makes this choice, an appropriate dialog will pop up to allow the user to specify a file name and to browse directories to determine where to save the file. (2) The user may choose to export the dictionary using text output. Again, if the user makes this choice, an appropriate dialog will pop up to handle file name and directory selection.



Additionally, once a dictionary has been loaded or imported, the display view check boxes will become active starting in the unchecked state and the minimum and maximum filter text boxes will become active starting with the minimum and maximum scores from the file as their respective default values.

Displaying Views:
Your program should create all views on startup but make them initially invisible. Only when the corresponding check box is checked will each view become visible. For example, when the boxes are checked as in the figure at the right, the score and language histograms should be visible but the word length histogram should be invisible.



Each time the dictionary contents are changed, your program will update the three views. This includes when the first file is loaded or imported (changing the dictionary from an empty dictionary to one containing words), when a replacement file is loaded or imported (changing from the dictionary previously in memory to a new dictionary), and when the words are filtered (removing words from the dictionary, see below).

Filtering:
As with Project 3, the user will have the option of filtering the words in the dictionary based on score. To enter filter score values, the user will type data into one of the text boxes and hit return. When the user enters into the minimum text box a value that is greater than the minimum score of the current dictionary, the dictionary will remove all words with lower scores than the value just entered. Similarly, when the user enters into the maximum text box a value that is less than the maximum score of the current dictionary, the dictionary will remove all words with higher scores than the value just entered.

Note that filtering takes place as soon as more restrictive values are entered into a text box and return is pressed. This will immediately remove words from the dictionary in memory. This means that all views should immediately be updated to reflect the changed model. It is not necessary to save or export the dictionary for the filtering to take place. To get back filtered words, one must read (import or load) a dictionary from a file.

Implementation Issues:

This is the largest project we've had so far so make sure to start early and budget your time well. Once you have 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; on the other hand, if you use your time well, you should have plenty of it.

Due Dates and Notes:

Your revised design and detailed Javadoc documentation are due on **Wednesday, November 6th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your revised UML design on *engineering paper* or a hardcopy using UML layout software, a hardcopy of the index page of your Javadoc documentation, and a hardcopy of the stubbed source code at the **beginning of lab on Thursday, November 7th**.

The final version of the project is due on **Wednesday, November 13th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your final UML design on *engineering paper* or a hardcopy using UML layout software, a hardcopy of the index page of your Javadoc documentation and a hardcopy of the source code at the **beginning of lab on Thursday, November 14th**.

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. 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. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.