Supervisor Request:

“Create GUI data selection and mapping software for a zip code database using the MVC paradigm.”

Milestones:

1. Create a hierarchical database of places, as you had in Project 3, to store all of the information from the input file. This will be called the “full database.” (5 points)

2. Create a second database called the “mapping database” to hold place information for places to be mapped. You will determine an appropriate structure for this database. (5 points)

3. 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 “PlaceModel.” This model will contain (1) the variables and methods needed to keep track of the two databases, and (2) the variables and methods necessary to keep track of views that will listen for changes to the databases. (10 points)

4. Create an MVC view associated with the full database in the model using the JTree class. The class for this view will be known as “TreeView.” Besides representing the full database in a tree, this view will have a button labeled “Add.” (10 points)

5. Create an MVC view associated with the mapping database in the model using the JTable class. The class for this view will be known as “TableView.” Besides representing the mapping database in a table, this view will have a button labeled “Map.” (10 points)

6. Create a “pseudo-view” associated with the mapping database in the model using the mapper code provided to you for Project 3. The class for this pseudo-view will be known as “MapView.” (10 points)

7. Create an MVC controller associated with the model. When the user clicks the “Add” button in a TreeView object, the controller will tell the mapping database to add to itself the zip objects from the full database that are selected in the TreeView object. When the user clicks on the “Map” button in a TableView object, the controller will tell a MapView to display itself. (10 points)

8. Catch and handle I/O exceptions. (10 points)

9. Develop and use a proper design. (15 points)

10. Use proper documentation and formatting. (15 points)

Description:

An import skill in software design is extending the work you have done in a previous project. For this project you will rework Project 3, using the MVC paradigm and some different Swing classes.
Overview:
The MVC paradigm gives us a way to organize our code involving 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 the data and accept user input, and one controller to moderate between the user gestures and the model and views. Your code will also catch and handle exceptions that may occur when reading from a file.

Model:
You will create a model class called `PlaceModel`. 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 states, cities, zips, and locations) 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 two databases in this project. One database will be static and the other will be dynamic.

Static Database:
The static database will be called the “full database.” When your program is run, it will take a command-line argument specifying the name of the data file to read in to populate the full database. If no filename is specified on the command line, it will display a dialog message informing the user that a command line argument needs to be supplied, and will exit when the dialog is closed. If a filename is specified but an exception is thrown when your program tries to open and/or read the file (for example, if the file does not exist or your program does not have permission to read it), your program will catch the exception, provide the user with an informative message in a dialog, then exit when the dialog is closed. If the filename is specified and the file is opened and read without exceptions, then it will populate the static database. The contents of this database will not be changed once it has been populated. You should ensure that no modifications are made to this database by making it immutable.

Dynamic Database:
The dynamic database will be called the “mapping database.” This database will initially be empty and each time the controller sends it a message to add selected data (see below under “Controller”), it will add objects to itself that will be copies of parts of the full database.

Interaction Variables and Methods:
To interact with views, the `PlaceModel` class will have variables and methods akin to those from the `CircleModel` class in your textbook. In particular, when objects are added to the mapping database in the model, the `TableView` object should be notified.

Views:
As listed under milestones, Project 4 will have two views of the model, `TreeView` and `TableView`, and one “pseudo-view” `MapView`.

The Tree View of the Full Database:
The `TreeView` class specifies a view of the static, full database. As such, this view will always be of the same data (because the data will not change once the database is populated). This view will be created and displayed as soon as the full database is populated. However, while the data in the view will not change, the view itself may change as the parts of the tree are expanded, collapsed, and/or highlighted. The hierarchical
nature of the full database should be reflected in the tree structure displayed in the **TreeView**. The database itself will be the root, the states the first level of the tree, the cities the second level, and the zip codes will be the leaves.

When a user highlights one or more (possibly non-contiguous) zip code leaves in the **TreeView** object, then clicks the “Add” button, the highlighted zip codes should be added to the mapping database. If the city with which the zip is associated is not already in the mapping database, a new city object will need to be created in the mapping database as well. Similarly, if the state object with which the new city object should be associated does not already exist in the mapping database, a new state object will need to be created and added to the mapping database.

**The Table View of the Mapping Database:**

Once the mapping database is non-empty, a table-based view of it should become visible. This view will be an instance of the **TableView** class. It will have columns for state, city, zip, latitude, and longitude. Each row will contain entries for all the data relevant to a single location – that is, each zip will go on its own row and all the columns will be filled for each row. The rows and columns of the table in the **TableView** object will be used for display only and will not accept any user input. Whenever the mapping database is updated, this view will receive notification of the change and will update itself to reflect the current state of the mapping database. The **TableView** object will also have a “Map” button to accept user input. When this button is pressed, a new view of the mapping database will be displayed that will show a Google map of all the zips in the mapping database.

**The Map Pseudo-View of the Mapping Database:**

When the “Map” button of the **TableView** is pressed, a new pseudo-view of the mapping database will be displayed that will show a Google map of all the zips in the mapping database. I am calling this a “pseudo-view” rather than a real view, since it will only represent a snapshot of the information in the model at the time the **MapView** is created; it will not continually update itself as the model changes. This is because we are launching an external application – a web browser – to display the Google map, which would make it difficult to keep it current. (Note that in many systems it would be possible to have our program interact with this external application to keep the view current. However, this is beyond the scope of this project, so we will be satisfied with having a pseudo-view for our maps.)

**Controller:**

You will create a controller class called “**PlaceController**” 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 highlights leaves in the **TreeView** and clicks on the “Add” button, the **PlaceController** will tell the mapping database to update itself by adding places corresponding to the highlighted leaves to itself.

2. Once the mapping database is no longer empty, the **PlaceController** will tell the **TableView** to display itself.

3. When the user hits “Map” in the **TableView**, the **PlaceController** will tell a new **MapView** to display itself.

**Extra Credit:**

There are several possibilities for extra credit for this assignment. One possibility is to design your system
to handle multiple view of each type (multiple TreeView and TableView objects) that all stay updated and allow for user input through their buttons. Another possibility is to allow for sorting of the places in the TableView, based on different fields of the data (state, city, zip, latitude, longitude). To determine how much extra credit will be given for any particular additional work, please check with Prof. Hougen.

**Due Dates and Notes:**

1. No third-party GUI packages may be used. Only standard Java classes and packages are allowed on projects. **Using a non-standard class will result in the program not compiling on the graders’ computers, and a non-compiling program will receive a grade of zero.**

2. Make sure to start early and budget your time well. Once you’ve got a good design, you can write a part at a time and test it before moving on to the next part. For example, you can test the TreeView code by importing the zips.txt file as you did in Project 3, selecting within the tree, then exporting the selected objects to a text file as you did in Project 3. Other code re-use from Project 3 may be applicable here as well, and can speed your development. **If you do not understand the design you developed in lab, it is your responsibility to attend office hours early in the project to get help with your design.**

3. Your revised design and detailed Javadoc documentation are due on Thursday, April 10th. Submit your revised UML design on engineering paper or a hardcopy produced using UML layout software, a hardcopy of the Javadoc documentation, and a hardcopy of the stubbed source code at the beginning of your assigned lab session. Submit the project archive following the steps given in the Submission Instructions by 9:00pm. **Note that the design points will be tripled for this assignment, to encourage you to spend more time and effort getting the design ready before you begin coding!**

4. The final version of the project is due on Thursday, April 17th. Submit your final UML design on engineering paper or a hardcopy produced using UML layout software, a hardcopy of the Javadoc documentation, and a hardcopy of the source code at the beginning of your assigned lab session. Submit the project archive following the steps given in the Submission Instructions by 9:00pm.

5. 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.

6. 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.

7. 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.