CS 2334
Project 4: Graphical User Interfaces

November 2, 2015

Due: 1:29 pm on Wednesday, Nov 18, 2015

Introduction

For the last three projects, you have been focused on reading data from files and constructing large, efficient representations from the data. For this project, we will focus on presenting these data to a user, enabling the user to explore the statistics associated with specific stations, variables and years.

Your implementation from project 3 will continue to serve as the basis for data loading and representation (with minimal changes). What you will add is a graphical user interface that interacts with the user.

Your final product will:

1. Load in files that describe the set of measures taken (the variables) at the stations, and the set of stations.
2. Allow the user to specify a data file to load.
3. Allow the user to select a station, a variable of interest and a set of years of interest.
4. Report the minimum, maximum and average of the selected statistic over the range of years that has been specified.

Learning Objectives

By the end of this project, you should be able to:

1. Create a menu that is attached to a frame.
2. Make use of JLists that present a set of options to a user and allow the user to select one or more of these options
3. Create a set of components that display textual data to a user
4. Create the listeners necessary to allow the GUI to respond to user input
5. Continue to exercise good coding practices for Javadoc and for testing

Note that this project relies heavily on your reading of the Java API documentation, and even examples. We have tried to provide you with a good set of hints, but, fundamentally, you have to pull the details out of the documentation.

Proper Academic Conduct

This project is to be done in the groups of two that we have assigned. You are to work together to design the data structures and solution, and to implement and test this design. You will turn in a single copy of your solution. Do not look at or discuss solutions with anyone other than the instructor, TAs or your assigned team. Do not copy or look at specific solutions from the net.

Strategies for Success

- The UML is a guide to the new classes and methods that you will implement.
- When you are implementing a class or a method, focus on just what that class/method should be doing. Try your best to put the larger problem out of your mind.
- We encourage you to work closely with your other team member, meeting in person when possible.
- Start this project early. In most cases, it cannot be completed in a day or two.

- Implement and test your project components incrementally. Don't wait until your entire implementation is done to start the testing process.
- Write your documentation as you go. Don't wait until the end of the implementation process to add documentation. It is often a good strategy to write your documentation **before** you begin your implementation.

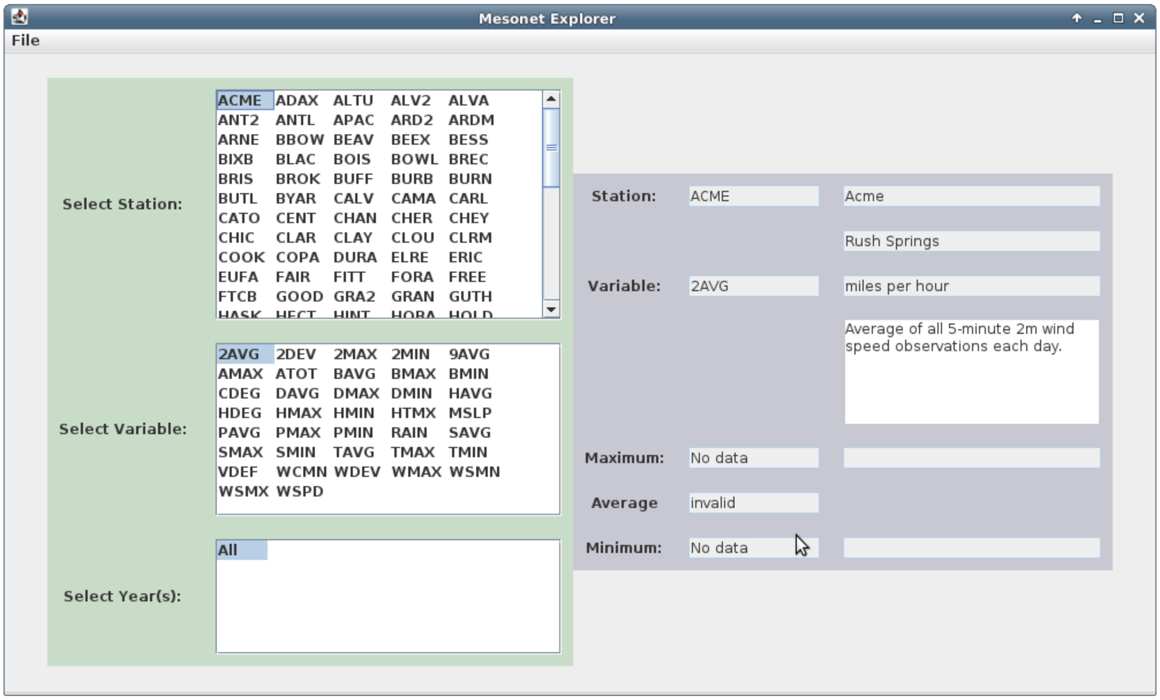
Preparation

- We will be providing parts of our project 3 implementation. However, this will only become available after the final deadline for project 3.
- Create a *project4* project. Within this project, create a *data* directory (folder). Copy the data that we provided in project 3 to this data directory.

Example Interactions

Below is a set of screen-shots for our implementation. Your implementation may have a different look. However, it must have the essential functionality, as described in the next section.

When your program starts up, it will immediately load the station and variable configuration files, but will not load a data file. Given the loaded information, here is the initial state of the interface:



Select Station:

ACME ADAX ALTU ALV2 ALVA
ANT2 ANTL APAC ARD2 ARDM
ARNE BBOW BEAV BEEX BESS
BIXB BLAC BOIS BOWL BREC
BRIS BROK BUFF BURB BURN
BUTL BYAR CALV CAMA CARL
CATO CENT CHAN CHER CHEY
CHIC CLAR CLAY CLOU CLRM
COOK COPA DURA ELRE ERIC
EUFA FAIR FITT FORA FREE
FTCB GOOD GRA2 GRAN GUTH
HASK HECT HINT HOBA HOLD

Select Variable:

2AVG 2DEV 2MAX 2MIN 9AVG
AMAX ATOT BAVG BMAX BMIN
CDEG DAVG DMAX DMIN HAVG
HDEG HMAX HMIN HTMX MSLP
PAVG PMAX PMIN RAIN SAVG
SMAX SMIN TAVG TMAX TMIN
VDEF WCMN WDEV WMAX WSMN
WSMX WSPD

Select Year(s):

All

Station: ACME Acme

Rush Springs

Variable: 2AVG miles per hour

Average of all 5-minute 2m wind speed observations each day.

Maximum: No data

Average invalid

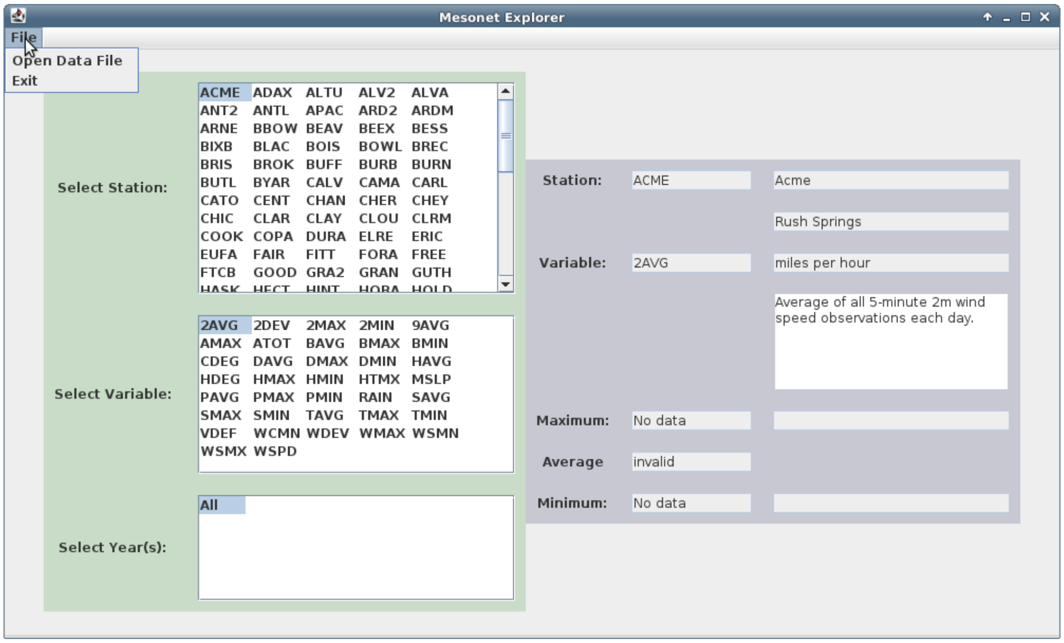
Minimum: No data

- A file menu is presented in the upper-left corner of the window.

- The green area contains three list interfaces that allows the user to select a stationId, a variable and one or more years. Only one station and variable may be selected at any one time. However, any combination of years can be selected.

- The dark gray area displays the selected station (ID, Name and City), the selected variable (ID, Units and Description), and the maximum, average and minimum for the selected station, variable and years. For the minimum and maximum values, the dates of the minimum and maximum are also shown.

When the file menu is selected, the full menu opens:



```
ACME ADAX ALTU ALV2 ALVA
ANT2 ANTL APAC ARD2 ARDM
ARNE BBOW BEAV BEEX BESS
BIXB BLAC BOIS BOWL BREC
BRIS BROK BUFF BURB BURN
BUTL BYAR CALV CAMA CARL
CATO CENT CHAN CHER CHEY
CHIC CLAR CLAY CLOU CLRM
COOK COPA DURA ELRE ERIC
EUFA FAIR FITT FORA FREE
FTCB GOOD GRA2 GRAN GUTH
HASK HECT HINT HORA HOLD
```

```
2AVG 2DEV 2MAX 2MIN 9AVG
AMAX ATOT BAVG BMAX BMIN
CDEG DAVG DMAX DMIN HAVG
HDEG HMAX HMIN HTMX MSLP
PAVG PMAX PMIN RAIN SAVG
SMAX SMIN TAVG TMAX TMIN
VDEF WCMN WDEV WMAX WSMN
WSMX WSPD
```

All

Station: ACME Acme

Rush Springs

Variable: 2AVG miles per hour

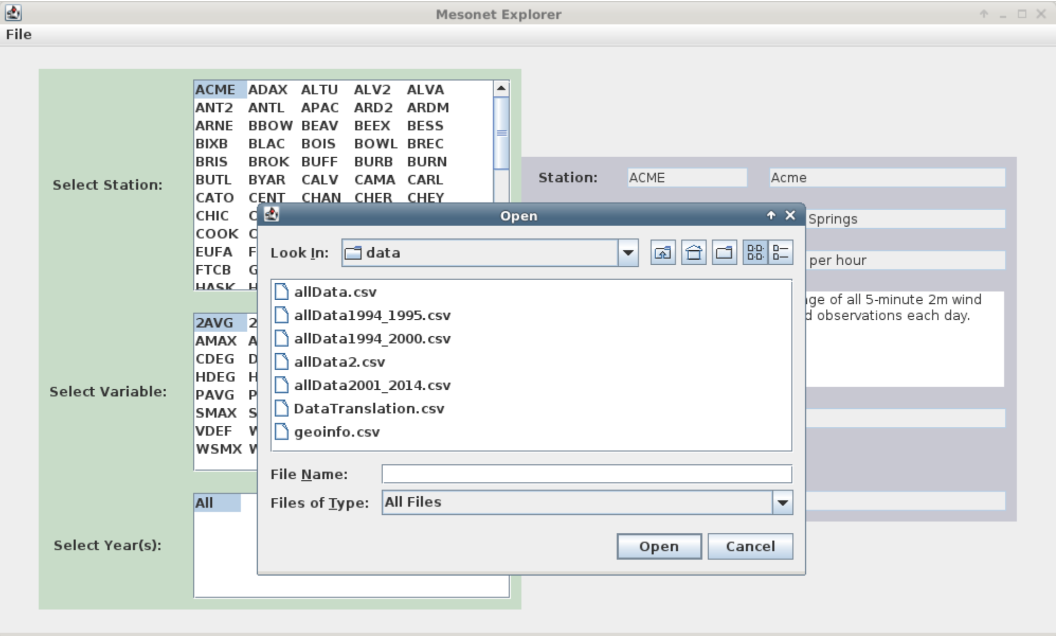
Average of all 5-minute 2m wind speed observations each day.

Maximum: No data

Average invalid

Minimum: No data

If *Exit* is selected, then the program exits (by calling System.exit(0)).
If *Open Data File* is selected, then a file chooser is opened:



Look In: data

allData.csv
allData1994_1995.csv
allData1994_2000.csv
allData2.csv
allData2001_2014.csv
DataTranslation.csv
geoinfo.csv

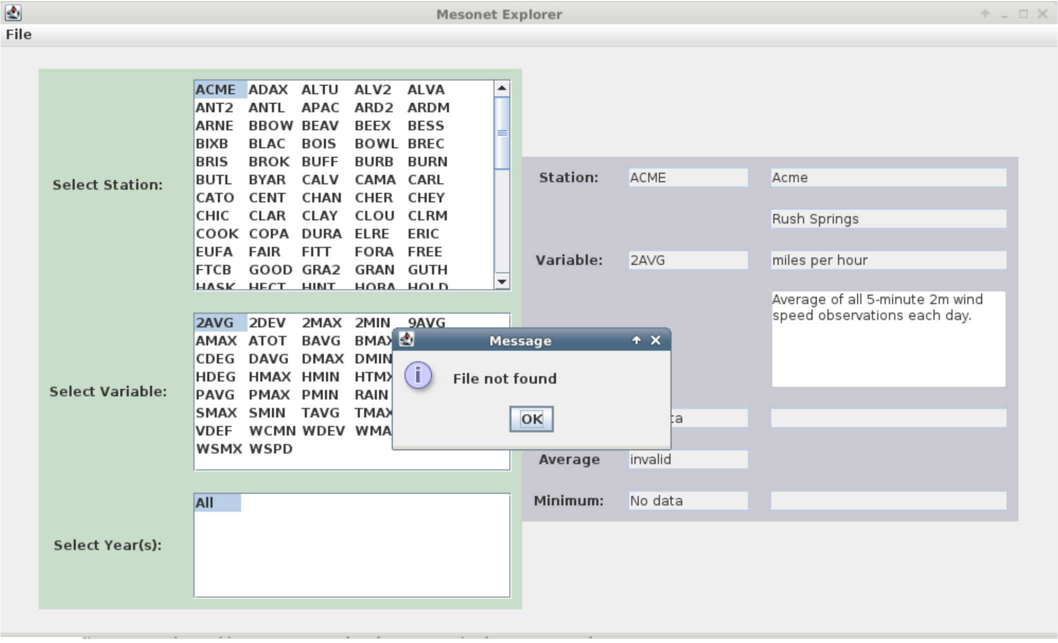
File Name:

Files of Type: All Files

Station: ACME Acme

- If any of the *allData* files are selected, then your program will begin to load the data. While the data are loading, the cursor changes to an animated clock to indicate that your program is busy. This can be accomplished by setting the Frame's cursor to: **Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR)**.
- If a file is specified that does not exist, your program should open an error window. This can be accomplished using **JOptionPane.showMessageDialog()**
- If an Exception is thrown while loading the file, then your program should also open an error window.

Here is one example of an error window:



Station: ACME Acme

Rush Springs

Variable: 2AVG miles per hour

Average of all 5-minute 2m wind speed observations each day.

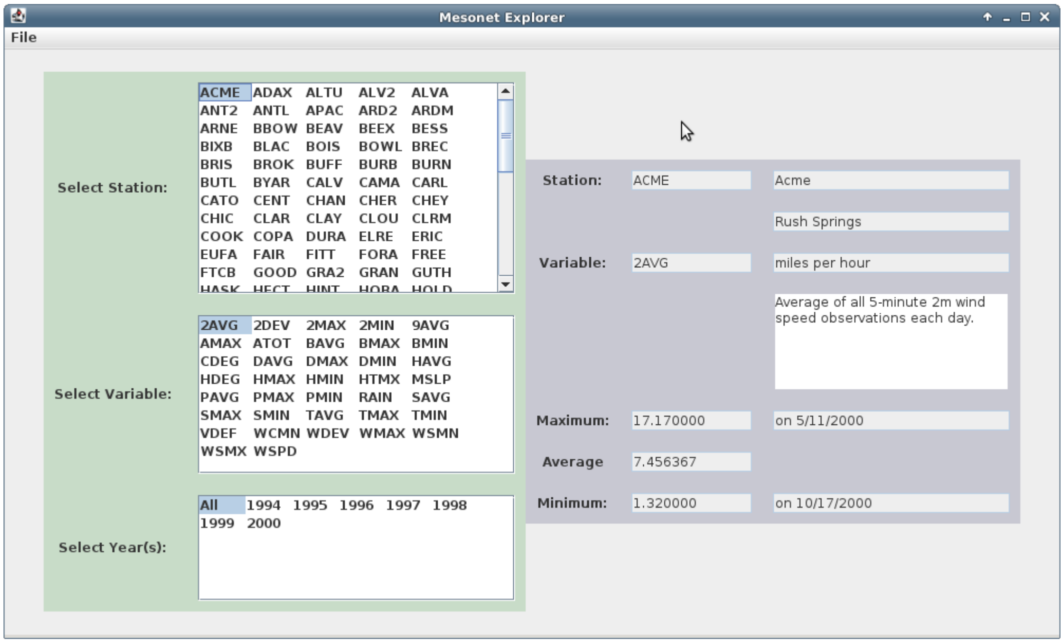
a

Average invalid

Minimum: No data

File not found

After loading, your program will display statistics about the selected station and variable for all years:



Station: ACME Acme

Rush Springs

Variable: 2AVG miles per hour

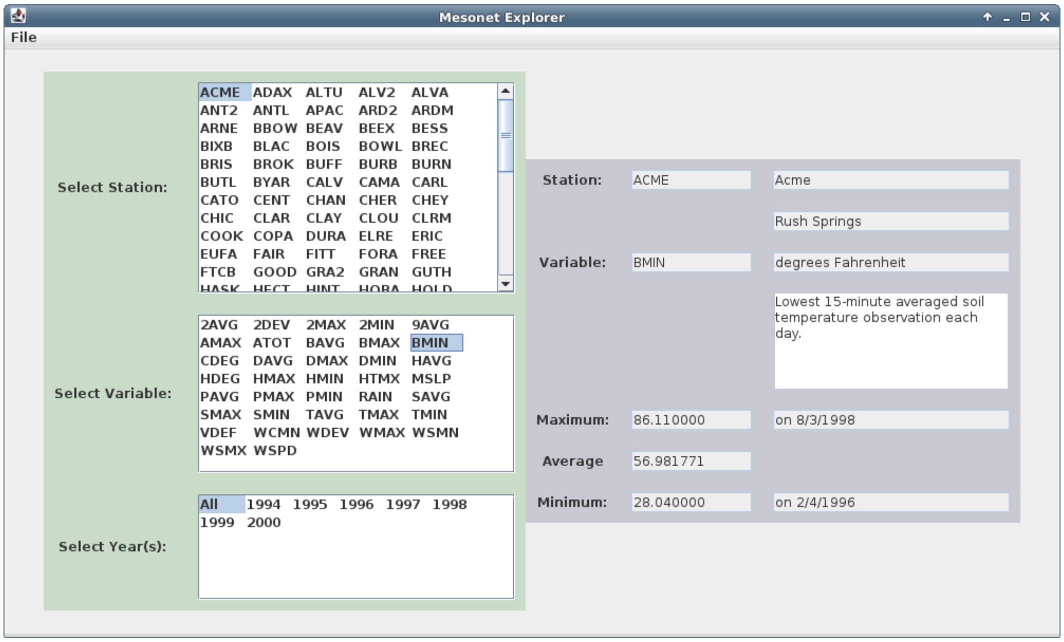
Average of all 5-minute 2m wind speed observations each day.

Maximum: 17.170000 on 5/11/2000

Average 7.456367

Minimum: 1.320000 on 10/17/2000

Another example:



Station: ACME Acme

Rush Springs

Variable: BMIN degrees Fahrenheit

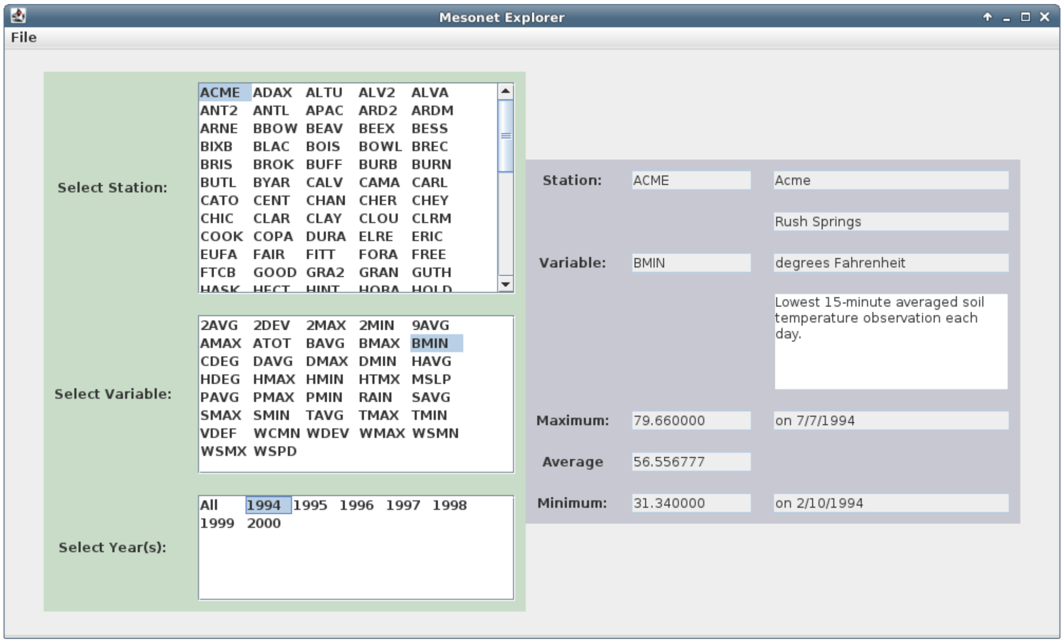
Lowest 15-minute averaged soil temperature observation each day.

Maximum: 86.110000 on 8/3/1998

Average 56.981771

Minimum: 28.040000 on 2/4/1996

Specific years can be selected:



Station: ACME Acme

Rush Springs

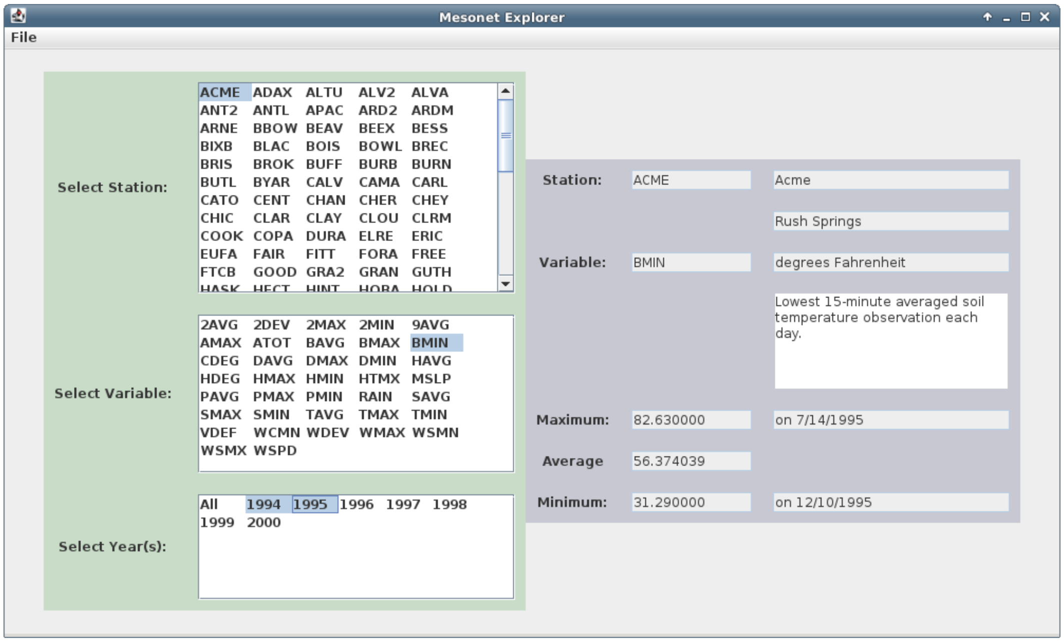
Variable: BMIN degrees Fahrenheit

Lowest 15-minute averaged soil temperature observation each day.

Maximum: 79.660000 on 7/7/1994

Average 56.556777

Minimum: 31.340000 on 2/10/1994



Station: ACME Acme

Rush Springs

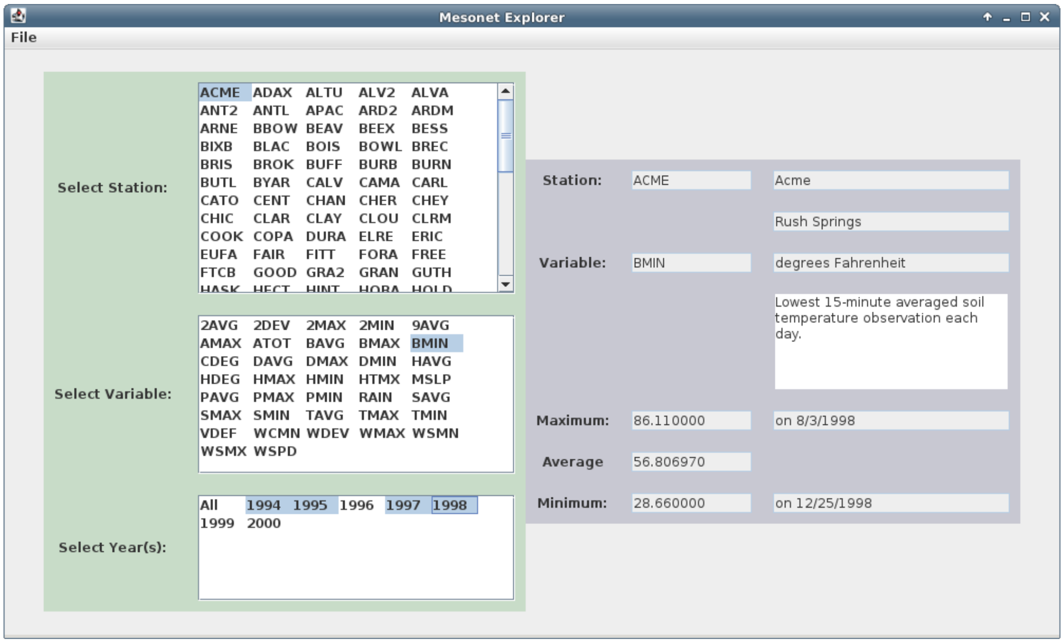
Variable: BMIN degrees Fahrenheit

Lowest 15-minute averaged soil temperature observation each day.

Maximum: 82.630000 on 7/14/1995

Average 56.374039

Minimum: 31.290000 on 12/10/1995



Select Station:

ACME ADAX ALTU ALV2 ALVA
ANT2 ANTL APAC ARD2 ARDM
ARNE BBOW BEAV BEEX BESS
BIXB BLAC BOIS BOWL BREC
BRIS BROK BUFF BURB BURN
BUTL BYAR CALV CAMA CARL
CATO CENT CHAN CHER CHEY
CHIC CLAR CLAY CLOU CLRM
COOK COPA DURA ELRE ERIC
EUFA FAIR FITT FORA FREE
FTCB GOOD GRA2 GRAN GUTH
HASK HECT HINT HOBA HOLD

Select Variable:

2AVG 2DEV 2MAX 2MIN 9AVG
AMAX ATOT BAVG BMAX BMIN
CDEG DAVG DMAX DMIN HAVG
HDEG HMAX HMIN HTMX MSLP
PAVG PMAX PMIN RAIN SAVG
SMAX SMIN TAVG TMAX TMIN
VDEF WCMN WDEV WMAX WSMN
WSMX WSPD

Select Year(s):

All 1994 1995 1996 1997 1998
1999 2000

Station: ACME Acme

Rush Springs

Variable: BMIN degrees Fahrenheit

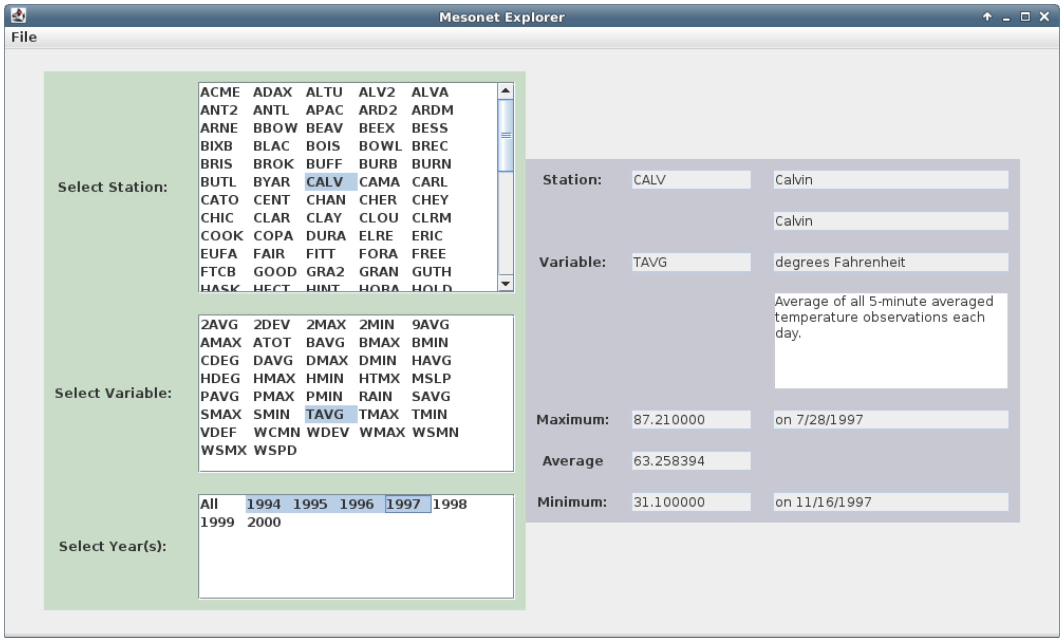
Lowest 15-minute averaged soil temperature observation each day.

Maximum: 86.110000 on 8/3/1998

Average 56.806970

Minimum: 28.660000 on 12/25/1998

A few other examples:



Select Station:

ACME ADAX ALTU ALV2 ALVA
ANT2 ANTL APAC ARD2 ARDM
ARNE BBOW BEAV BEEX BESS
BIXB BLAC BOIS BOWL BREC
BRIS BROK BUFF BURB BURN
BUTL BYAR CALV CAMA CARL
CATO CENT CHAN CHER CHEY
CHIC CLAR CLAY CLOU CLRM
COOK COPA DURA ELRE ERIC
EUFA FAIR FITT FORA FREE
FTCB GOOD GRA2 GRAN GUTH
HASK HECT HINT HOBA HOLD

Select Variable:

2AVG 2DEV 2MAX 2MIN 9AVG
AMAX ATOT BAVG BMAX BMIN
CDEG DAVG DMAX DMIN HAVG
HDEG HMAX HMIN HTMX MSLP
PAVG PMAX PMIN RAIN SAVG
SMAX SMIN TAVG TMAX TMIN
VDEF WCMN WDEV WMAX WSMN
WSMX WSPD

Select Year(s):

All 1994 1995 1996 1997 1998
1999 2000

Station: CALV Calvin

Calvin

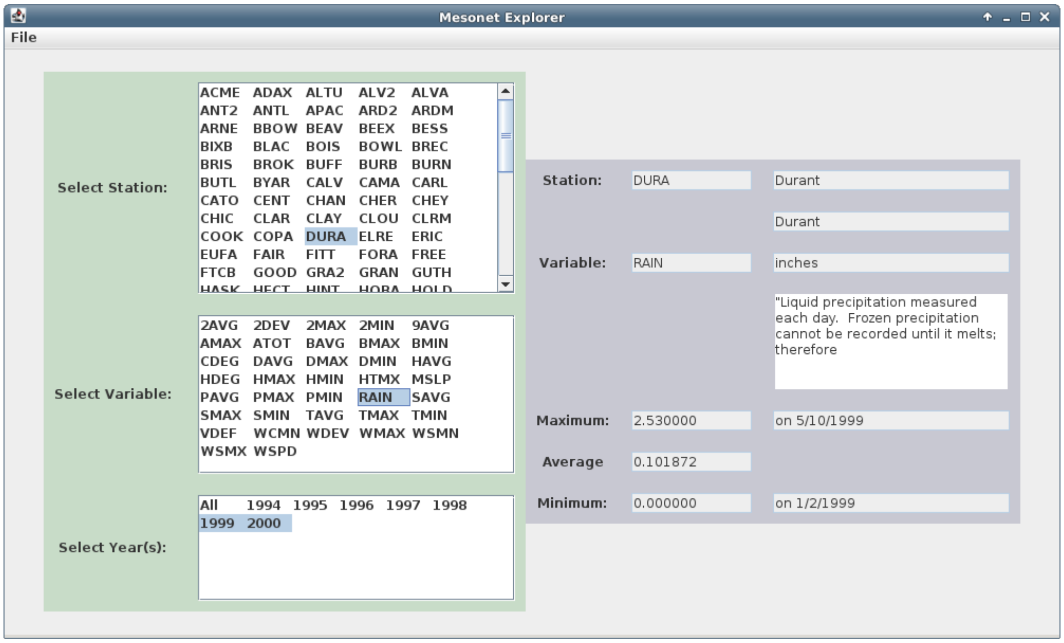
Variable: TAVG degrees Fahrenheit

Average of all 5-minute averaged temperature observations each day.

Maximum: 87.210000 on 7/28/1997

Average 63.258394

Minimum: 31.100000 on 11/16/1997



Station: DURA | Durant

Durant

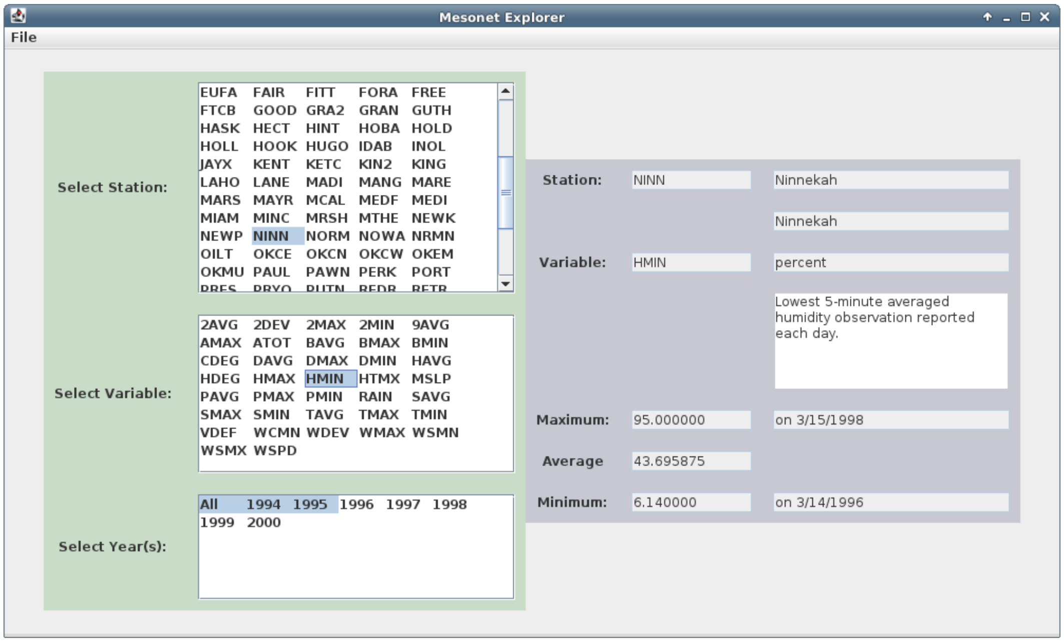
Variable: RAIN | inches

"Liquid precipitation measured each day. Frozen precipitation cannot be recorded until it melts; therefore

Maximum: 2.530000 | on 5/10/1999

Average 0.101872

Minimum: 0.000000 | on 1/2/1999



Station: NINN | Ninnekah

Ninnekah

Variable: HMIN | percent

Lowest 5-minute averaged humidity observation reported each day.

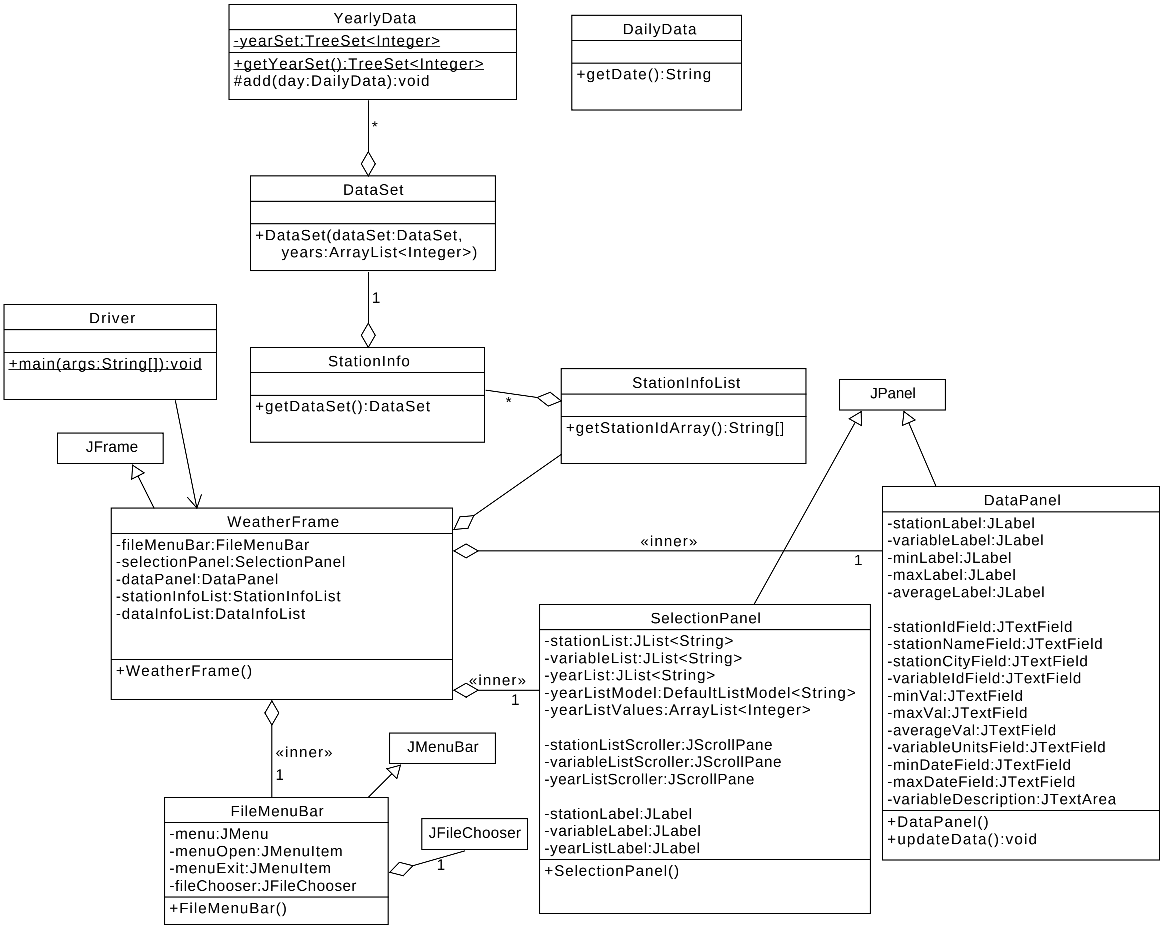
Maximum: 95.000000 | on 3/15/1998

Average 43.695875

Minimum: 6.140000 | on 3/14/1996

UML Design

Below is an outline of what has changed from project 3:



Class Design Outline

Your project 3 code will largely stay the same; changes are described below.

We outline our implementation of our GUI code. You may choose to follow this design, if you wish. However, you must include all of the essential elements, including: station, variable and year(s) selection; station display (including ID, Name and City); variable display (including ID, Units and Description); maximum, average and minimum statistics (including date of maximum and minimum).

- **DailyData**: change *getDate()* to only include month, day and year in the returned string.

- **YearlyData**: add a *yearSet* class variable that will be used to track the set of years for which data have been loaded. This will involve a small change to *add()*. Also, add a getter for this set.

- **DataSet**: add a new constructor that takes as input an existing **DataSet** and an array of years. This constructor creates a new **DataSet** instance that includes only those years that are in the original **DataSet** and the years array. **Do not clone the years for this new instance – just add the references to the appropriate years**.

- **StationInfo**: add a method that returns the station's **DataSet**.

- **StationInfoList**: add a method that returns an array of Strings that contain the stationIds that have been loaded.

- (optional) **WeatherFrame**: Create a new class that is-a **JFrame**. This is the primary window of the interface.

- (optional) **FileMenuBar**: Create an inner class to **WeatherFrame** that is-a **JMenuBar**. This class handles the creation of the menu and the user interaction.

- (optional) **SelectionPanel**: Create an inner class that is-a **JPanel** that presents the elements through which the user will select the station, variable and year(s). This class contains a **JList** for each of these.

- (optional) **DataPanel**: Create an inner class that is-a **JPanel** that displays the selected information and the associated statistics.

  This class also provides an *updateData()* method that takes the selection information from the **SelectionPanel** and updates the components of the **DataPanel**.

Note: *optional* here means that you can choose your own implementation. However, you must include this functionality somehow.

Notes

- Build your GUI incrementally. Focus on the "look and feel" of your GUI before you add functionality. Then, add functionality one piece at a time.

- The use of multiple classes to represent the GUI gives us the opportunity to logically partition the problem into smaller pieces. Because these pieces are largely independent of one-another, this allows us to keep the complexity down.

- By setting up all of these classes (but one) as **inner classes** of a larger frame class, this allows us to easily handle the dependencies between the various GUI classes. In particular, inner classes have the ability to access variables and methods of the outer class, even when they are private. In particular, an inner class can refer to the outer class instance using:

  WeatherFrame.this

  and, hence, access variables and call methods using:

  WeatherFrame.this.stationInfoList

  WeatherFrame.this.setCursor()

  In addition, one inner class can access pieces of another inner class. For example, the **SelectionPanel** instance can tell the **DataPanel** instance to update using:

  WeatherFrame.this.dataPanel.updateData()

- **JMenuItems** can have **ActionListener**s attached to them.

- You can create a reference to your data directory this way:

  new File("./data")

- **JLists** present a list of items to the user and allows the user to select one (or possibly more). See the reference section below for a useful link that talks about many options.

  When the items in the list are known *a priori* and won't change, the simple way to create a **JList** is to hand it an array of Strings – one for each item. You can then tell the **JList** to select the first item in the list automatically:

  setSelectedIndex(0)

A **SelectionListener** can then be added to respond to new selections. The currently selected element can be read from the **JList** using **getSelectedValue()**.

When the items are not known *a priori* or will change with time (as is the case with the list of years, which we won't know until we have loaded the data), we must use some form of **ListModel**. The **DefaultListModel** class is a **List** to which items can be added (this list can also be cleared). Every time this list changes, the **DefaultListModel** will automatically inform the **JList** that the list has changed, which will cause the display to be updated. Use the **ListModel** as the input to the **JList** constructor.

- I placed each **JList** inside of a **JScrollPane**. This tells the GUI to use a fixed size pane to present the information, but to provide scroll bars if the information is too large to display in the fixed area.

- **JTextField**s, by default, are about receiving text input from a user. However, they can be used as output-only components by setting their *editable* property to *false*. They are convenient for this because we can define their width in terms of the number of characters that they should hold.

- **GridBagLayout** works nice for this GUI.

- **JTextArea** will display multi-line text. I recommend the following configuration:

  setWrapStyleWord(true) and **setLineWrap(true)**

Final Steps

1. Generate Javadoc using Eclipse for all of your classes.

2. Open the *project4/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that all of your classes are listed and that all of your documented methods have the necessary documentation.

Submission Instructions

- All required components (source code and compiled documentation) are due at 1:29 pm on Wednesday, November, 18th (i.e, before class begins).

- Prepare your submission file by creating a project4.zip file. This file must include your entire project, including: src, and doc. **Do not include your data directory**

- Submit your zip file to the project4 folder on D2L.

Grading: Code Review

All groups must attend a code review session in order to receive a grade for your project. The procedure is as follows:

- Submit your project for grading to the D2L Dropbox, as described above.

- Any day following the submission, you may do the code review with the instructor or the TAs. For this, you have two options:

  1. Schedule a 15-minute time slot in which to do the code review. We will use Doodle to schedule these (a link will be posted on D2L). You must attend the code review during your scheduled time. Failure to do so will leave you only with option 2 (no rescheduling of code reviews is permitted).
  2. "Walk-in" during an unscheduled office hour time. However, priority will be given to those needing assistance in the labs and project.

- Both group members must be present for the code review.

- During the code review, we will discuss all aspects of the rubric, including:
  1. The results of the tests that we have executed against your code.
  2. The documentation that has been provided (all three levels of documentation will be examined).
  3. The implementation. Note that both group members must be able to answer questions about the entire solution that the group has produced.
- If you complete your code review before the submission deadline, you have the option of going back to make changes and resubmitting (by the deadline). If you do this, you will need to return for another code review.
- The code review must be completed by Monday, December 7th to receive credit for the project.

References

- The Java API: https://docs.oracle.com/javase/8/docs/api/
- JLists: https://docs.oracle.com/javase/tutorial/uiswing/components/list.html
- JFileChooser: https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html
- Menus: https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 45 points

Program formatting: 10 points

(10) The program is properly formatted (including indentation, curly brace and semicolon locations).

(5) There is one problem with program formatting.

(0) The program is not properly formatted.

Data types and method calls: 10 points

(10) The program is using proper data types and method calls.

(7) There is one error in data type or method call selection.

(4) There are two errors in data type or method call selection.

(0) There are three or more errors in data type and method call selection.

Required Methods: 15 points

(15) All of the required methods are implemented.

(10) One required method is not implemented

(5) Two required methods are not implemented.

(0) Two or more required methods are not implemented.

GUI Design: 10 points

(10) All required GUI elements are included.

(7) One key GUI element is missing

(4) Two key GUI elements are missing.

(0) Three or more key GUI elements are missing.

Proper Execution: 30 points

Output: 15 points

Two (2) points will be deducted for every test that your program fails (note that these are tests that we provide).

Execution: 15 points

(15) The program executes with no errors.

(8) The program executes, but there is one minor error.

(0) The program does not execute.

Documentation and Submission: 25 points

Project Documentation: 4 points

(4) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 9 points

(9) Every method contains all of the required documentation elements ahead of the method prototype.

(6) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements.

(0) The method documentation is missing more than two of the required documentation elements.

Inline Documentation: 9 points

(9) Every method contains appropriate inline documentation.

(6) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 3 points

(3) The correct zip file name is used and has the correct contents.

(2) The correct zip file name is used, but one required component is missing.

(0) An incorrect zip file name is used or more than one required component is missing.