# CS 2334
# Project 4: Graphical User Interfaces

November 1, 2016

**Due: 1:29:00 pm on Wednesday, Nov 16, 2016**

## 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 (multiple data load requests will be allowed).

3. Allow the user to select a station, a variable of interest and a set of years of interest.

4. Report detailed information about the station and the statistic, as well as 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 the 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. Note that it is very challenging to write JUnit tests for GUIs – we do not expect you to provide these here. However, we do expect that you will provide unit tests for the "back end" of your code and that you will test your GUI in person.

- 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

- This description and supporting materials are available at: http://cs.ou.edu/~fagg/classes/cs2334/projects/project4

- We will be providing parts of our project 3 implementation on Canvas.

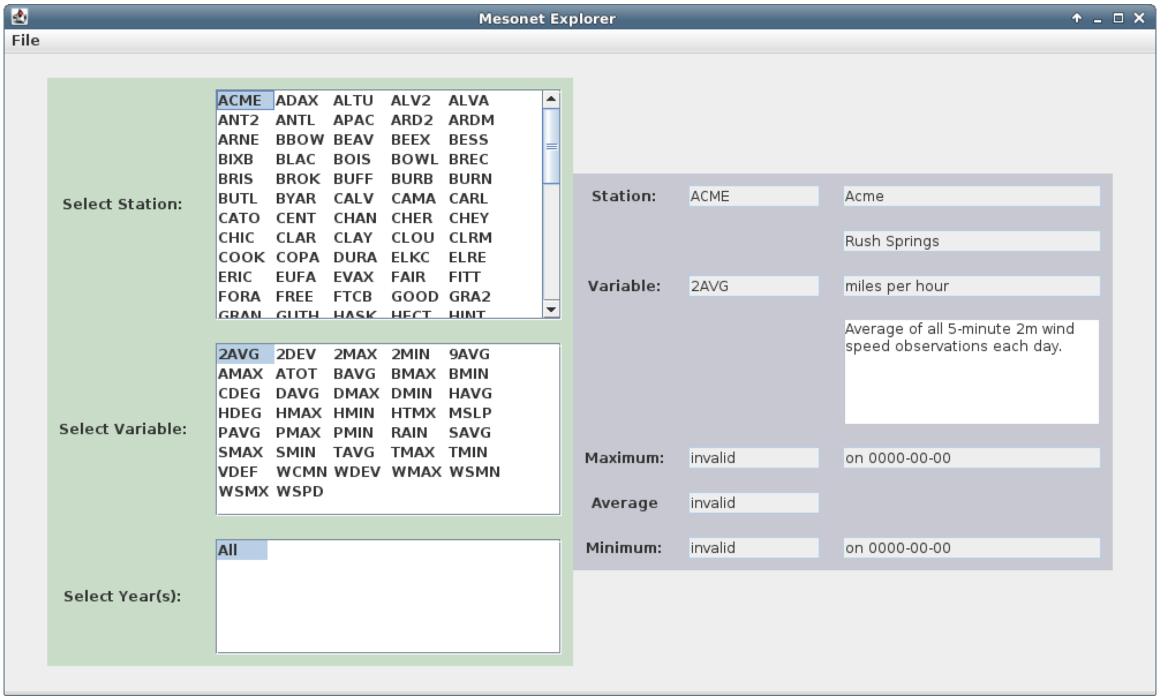
- In Eclipse, copy your *project3* folder to a new *project4* project. Within this project, your data should be located in the *data* directory (folder). The data will be the same as for the last project.

- Download project4.zip from the project directory. This zip file contains a partial implementation of the **WeatherFrame** class and new versions of the *geoinfo.csv* and *DataTranslation.csv* files. Copy the former into your *src* directory; and the data files into your *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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT

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.

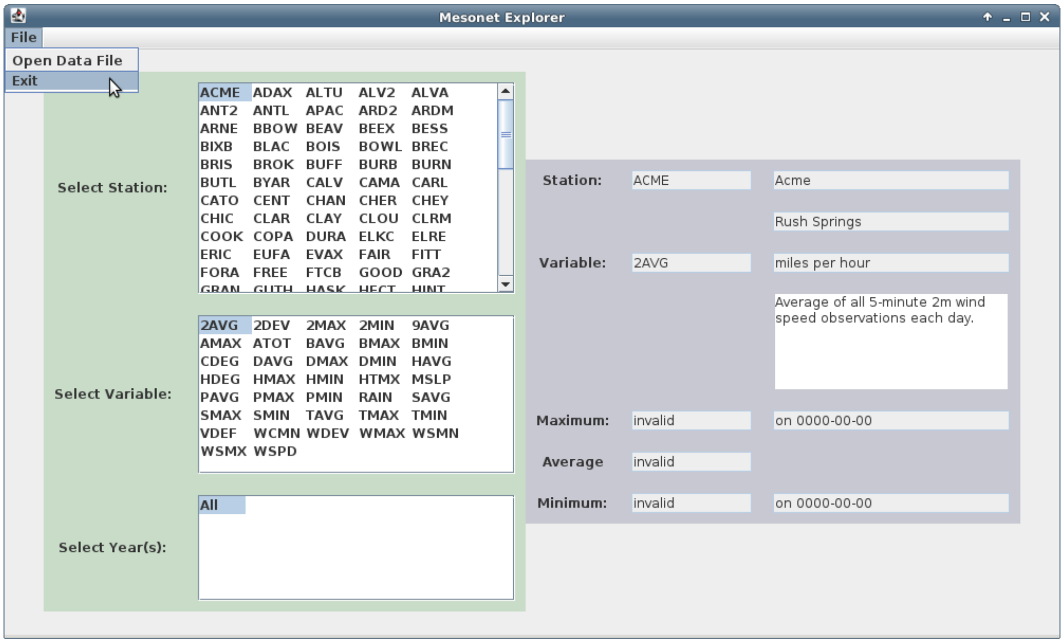
Maximum: invalid on 0000-00-00

Average invalid

Minimum: invalid on 0000-00-00

- A file menu is presented in the upper-left corner of the window.
- The green area contains three list interfaces that allow 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:



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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT
```

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

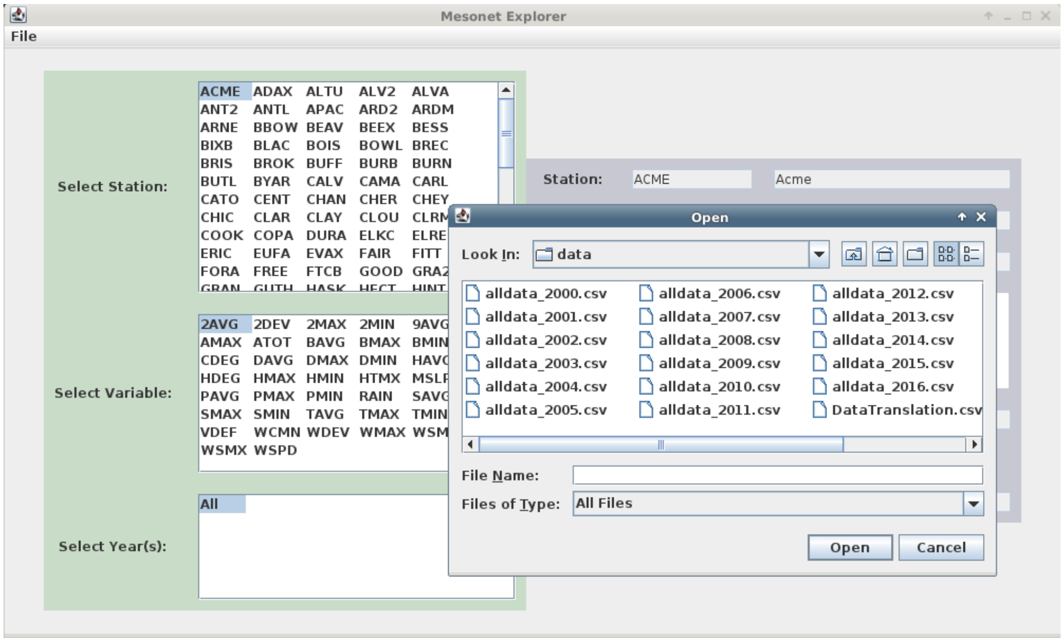
Average of all 5-minute 2m wind speed observations each day.

Maximum: invalid on 0000-00-00

Average invalid

Minimum: invalid on 0000-00-00

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:



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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT
```

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 WSM
WSMX WSPD
```

Select Year(s): All

Station: ACME Acme

```
alldata_2000.csv   alldata_2006.csv   alldata_2012.csv
alldata_2001.csv   alldata_2007.csv   alldata_2013.csv
alldata_2002.csv   alldata_2008.csv   alldata_2014.csv
alldata_2003.csv   alldata_2009.csv   alldata_2015.csv
alldata_2004.csv   alldata_2010.csv   alldata_2016.csv
alldata_2005.csv   alldata_2011.csv   DataTranslation.csv
```

File Name:

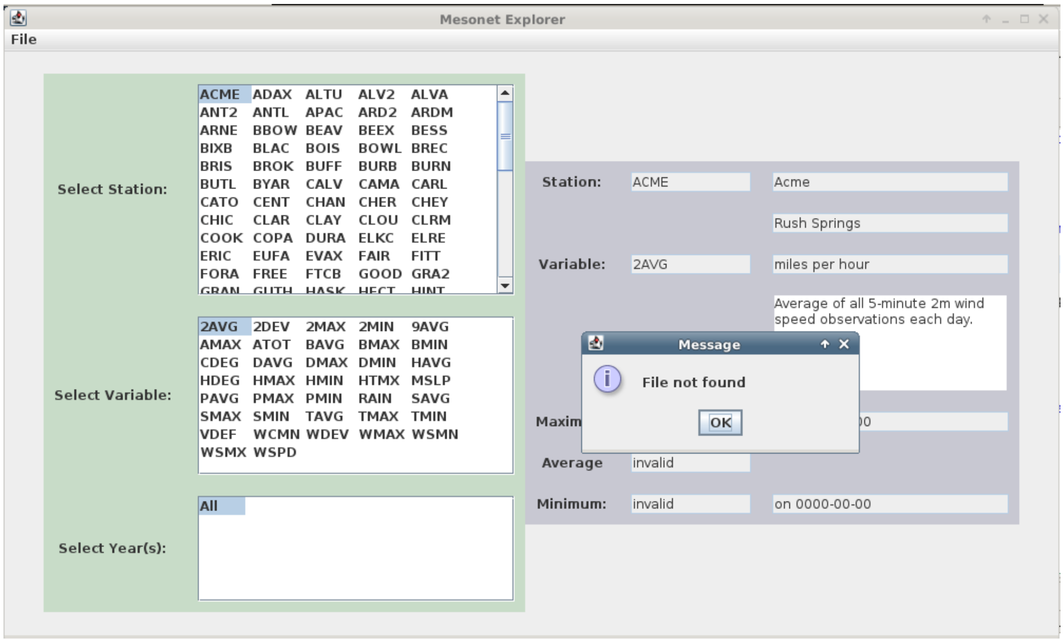
Files of Type: All Files

- 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:



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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT

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.

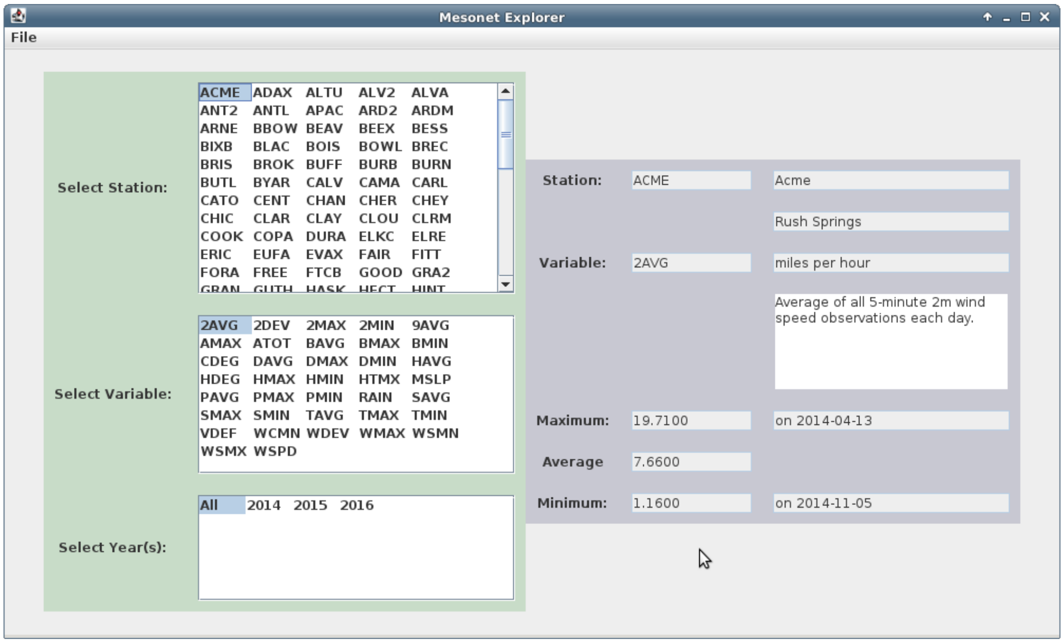
Maxim 00

Average invalid

Minimum: invalid on 0000-00-00

File not found

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



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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT

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 2014 2015 2016

Station: ACME Acme

Rush Springs

Variable: 2AVG miles per hour

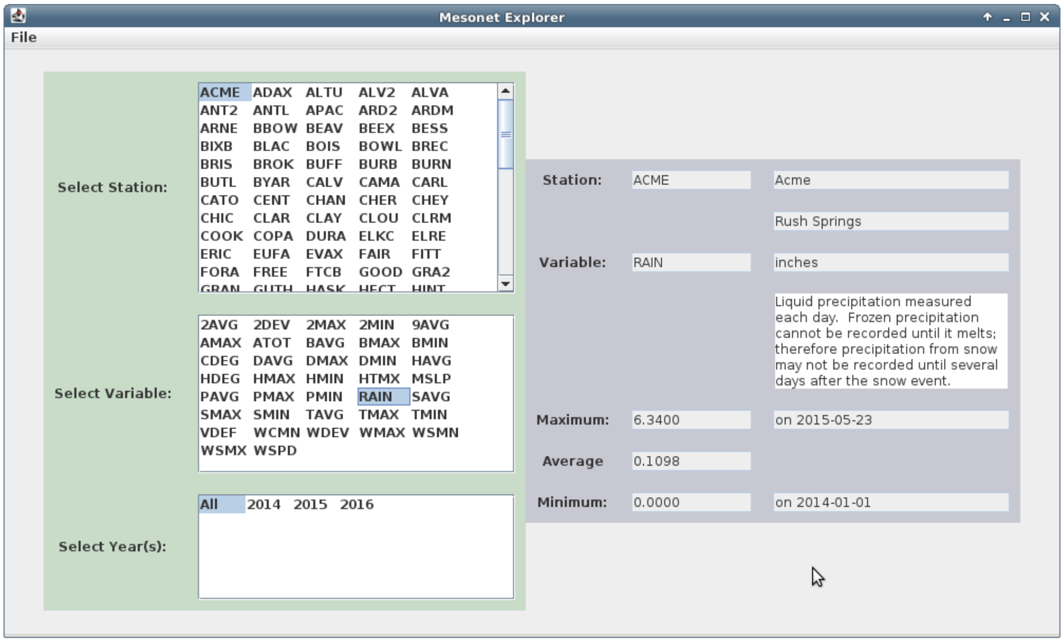
Average of all 5-minute 2m wind speed observations each day.

Maximum: 19.7100 on 2014-04-13

Average 7.6600

Minimum: 1.1600 on 2014-11-05

Another example:



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 ELKC ELRE
ERIC EUFA EVAX FAIR FITT
FORA FREE FTCB GOOD GRA2
GRAN GUTH HASK HECT HINT

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 2014 2015 2016

Station: ACME Acme

Rush Springs

Variable: RAIN inches

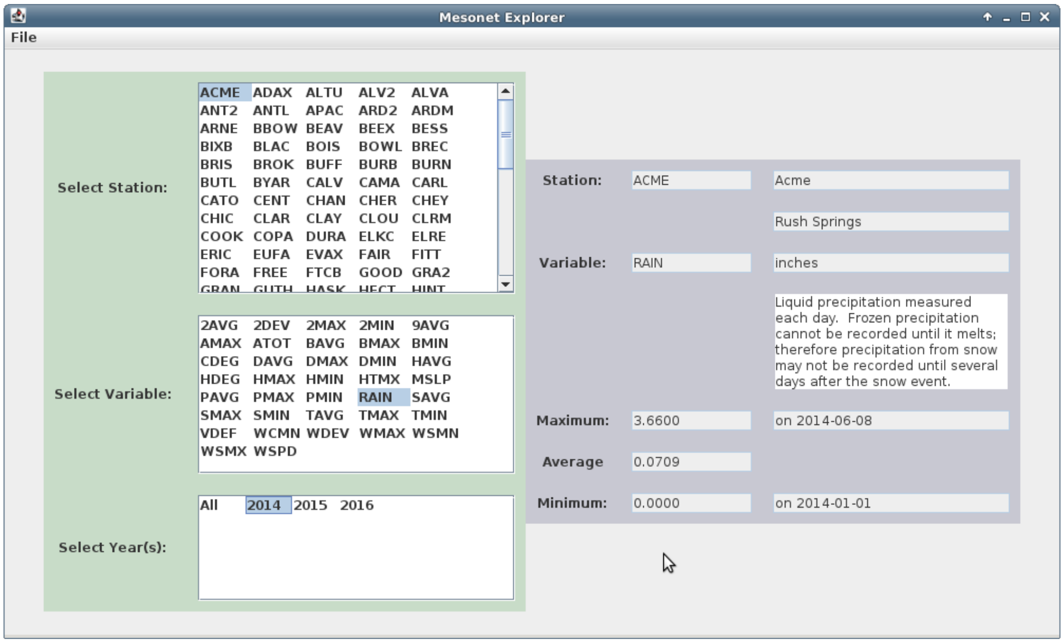
Liquid precipitation measured each day. Frozen precipitation cannot be recorded until it melts; therefore precipitation from snow may not be recorded until several days after the snow event.

Maximum: 6.3400 on 2015-05-23

Average 0.1098

Minimum: 0.0000 on 2014-01-01

Specific years can be selected:



Station: ACME Acme

Rush Springs

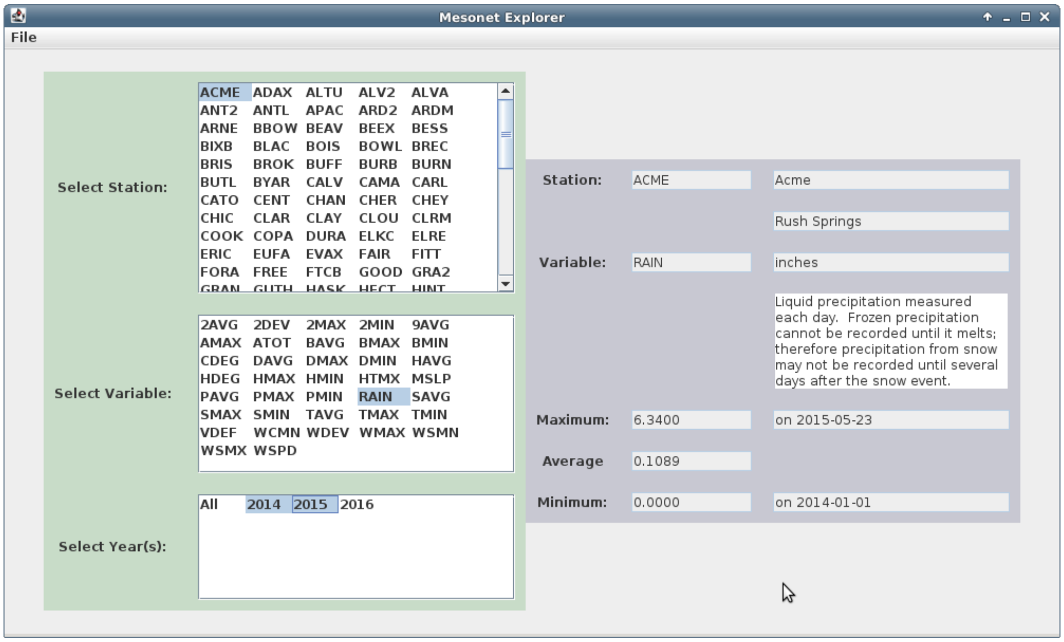
Variable: RAIN inches

Liquid precipitation measured each day. Frozen precipitation cannot be recorded until it melts; therefore precipitation from snow may not be recorded until several days after the snow event.

Maximum: 3.6600 on 2014-06-08

Average 0.0709

Minimum: 0.0000 on 2014-01-01



Station: ACME Acme

Rush Springs

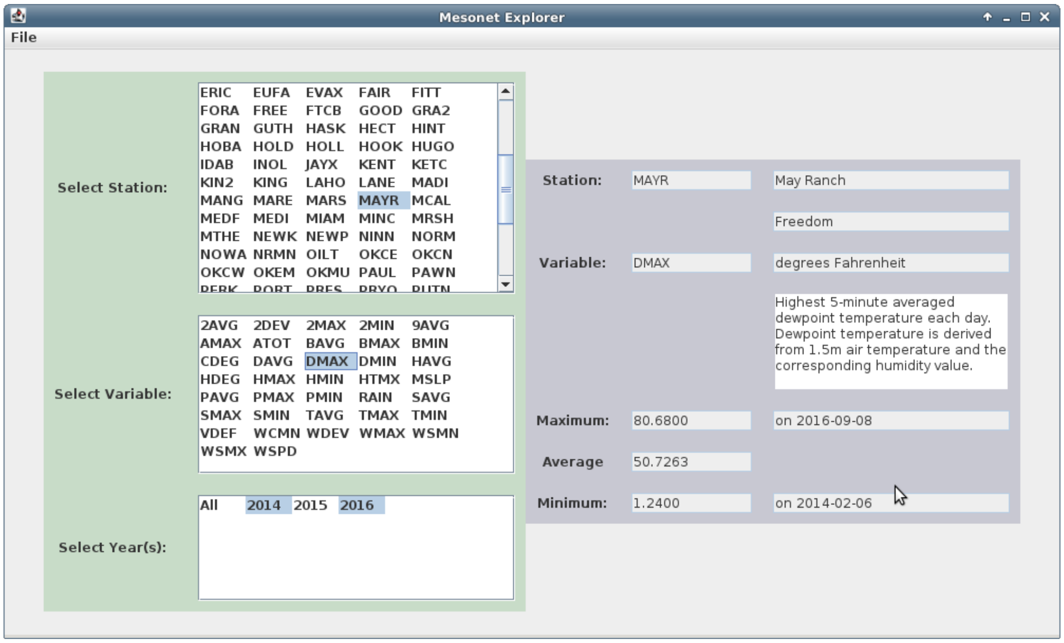
Variable: RAIN inches

Liquid precipitation measured each day. Frozen precipitation cannot be recorded until it melts; therefore precipitation from snow may not be recorded until several days after the snow event.

Maximum: 6.3400 on 2015-05-23

Average 0.1089

Minimum: 0.0000 on 2014-01-01



Select Station: MAYR

Select Variable: DMAX

Select Year(s): 2014 2016

Station: MAYR May Ranch

Freedom

Variable: DMAX degrees Fahrenheit

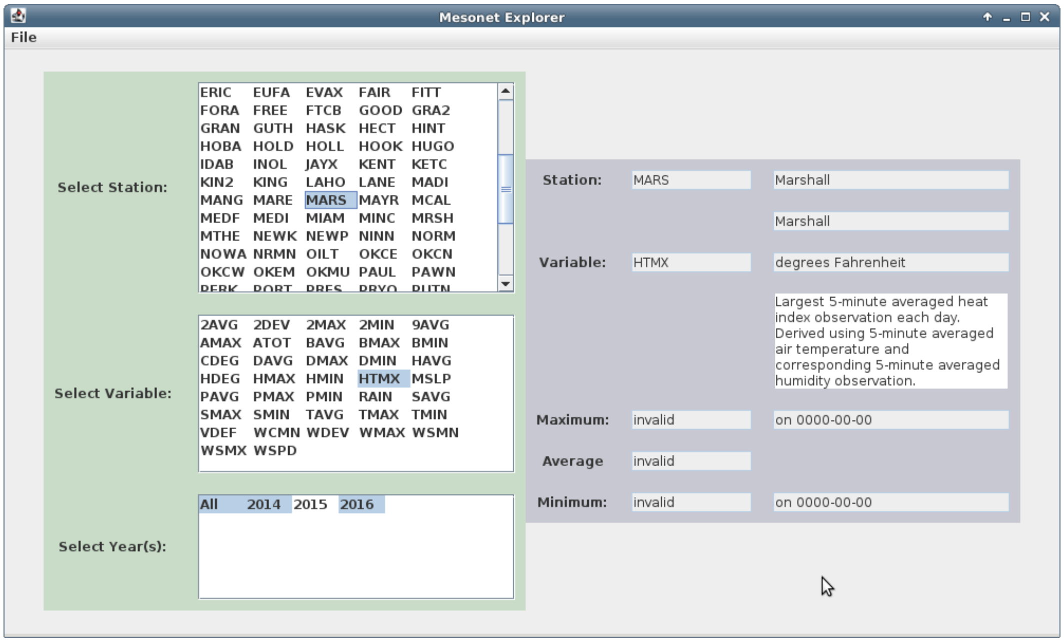
Highest 5-minute averaged dewpoint temperature each day. Dewpoint temperature is derived from 1.5m air temperature and the corresponding humidity value.

Maximum: 80.6800 on 2016-09-08

Average 50.7263

Minimum: 1.2400 on 2014-02-06

A few other examples:



Select Station: MARS

Select Variable: HTMX

Select Year(s): All 2014 2016

Station: MARS Marshall

Marshall

Variable: HTMX degrees Fahrenheit

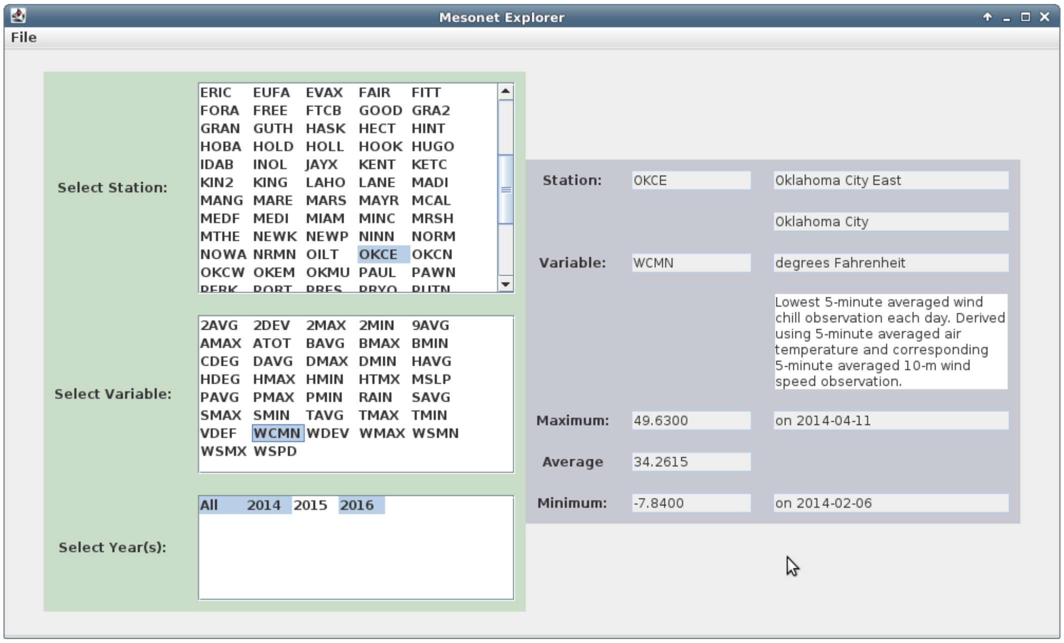
Largest 5-minute averaged heat index observation each day. Derived using 5-minute averaged air temperature and corresponding 5-minute averaged humidity observation.

Maximum: invalid on 0000-00-00

Average invalid

Minimum: invalid on 0000-00-00

Note: the 0000-00-00 occurs because 2016 does not contain data for the months of November and December. Hence, the min/max for December is an invalid DataDay, which reports 0000-00-00 as its date. Because of our implementation of isLessThan and isGreaterThan, this invalid day overrides all of months that contain data.



Station: OKCE Oklahoma City East

Oklahoma City

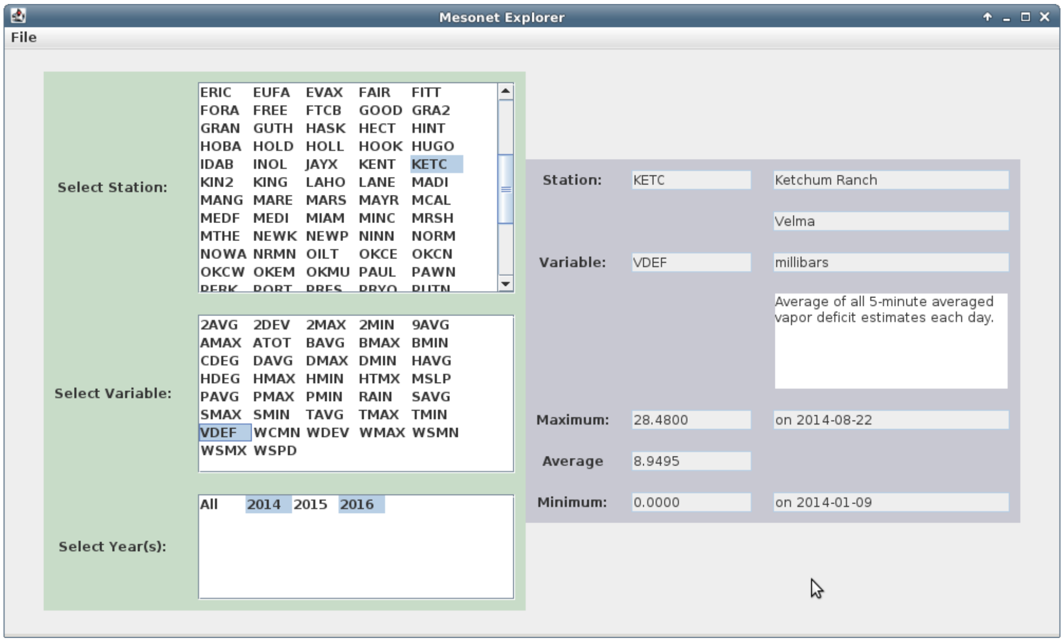
Variable: WCMN degrees Fahrenheit

Lowest 5-minute averaged wind chill observation each day. Derived using 5-minute averaged air temperature and corresponding 5-minute averaged 10-m wind speed observation.

Maximum: 49.6300 on 2014-04-11

Average 34.2615

Minimum: -7.8400 on 2014-02-06



Station: KETC Ketchum Ranch

Velma

Variable: VDEF millibars

Average of all 5-minute averaged vapor deficit estimates each day.

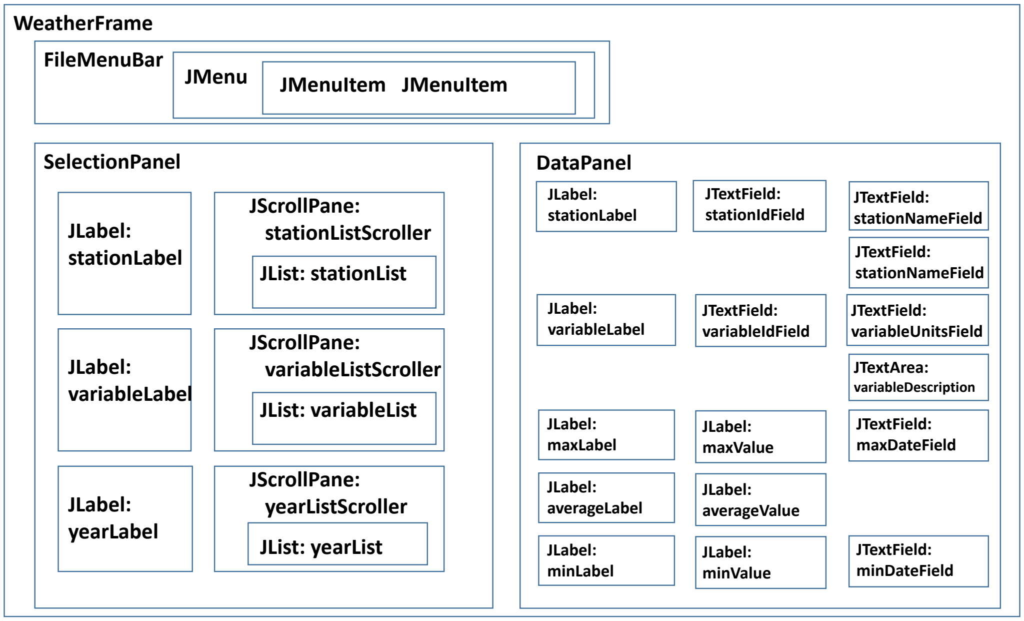
Maximum: 28.4800 on 2014-08-22

Average 8.9495

Minimum: 0.0000 on 2014-01-09

# GUI Layout

Below is a sketch of our GUI layout. Here, we are describing the key GUI components and their approximate layout. Implicit in the way we have drawn things is also a *containment* relationship. Some of the relevant instance variables are also listed.



The **WeatherFrame** contains three main components: a **FileMenuBar**, a **SelectionPanel** and a **DataPanel**.

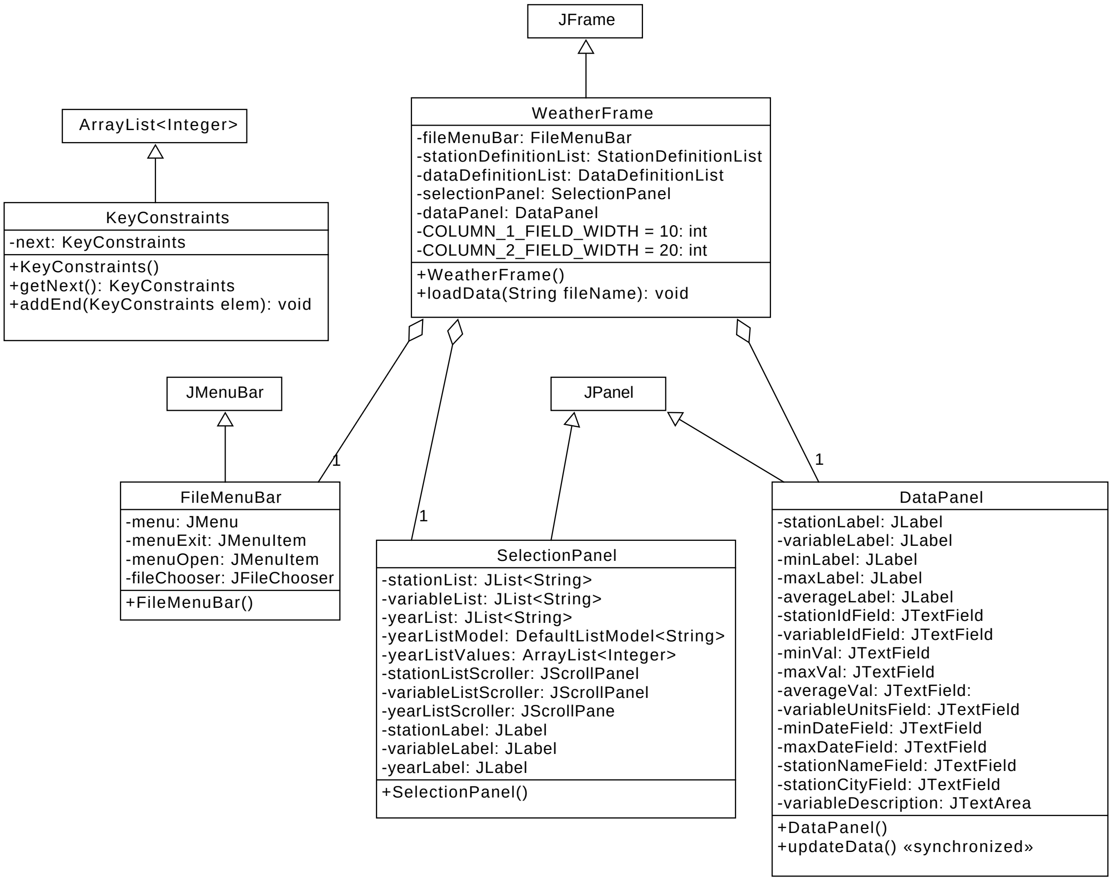
The **FileMenuBar** contains a single **JMenu**, which, in turn, contains two **JMenuItems**.

The **SelectionPanel** contains a grid of sub-components: the rows correspond to the station, variable and the years to be selected. The first column contains the labels, while the second column contains a set of **JLists** that we will use for selection. Note that each **JList** is contained within a **JScrollPane**. These scroll panes allow us to have a **JList** than will always fit within the allotted space. Should a list be too large, the **JScrollPane** will automatically show a scroll bar on the right hand side of the list.

The **DataPanel** presents information according to what the user has selected. A set of labels are presented in the first column. The remaining columns are either **JTextField** or **JTextArea** objects and are used to display specific **Strings.**

# UML Design

You will adopt your implementation from project 3 with minimal changes (detailed below). Below are the new classes that you will be implementing/modifying for this project.


JFrame
ArrayList<Integer>
WeatherFrame
-fileMenuBar: FileMenuBar
-stationDefinitionList: StationDefinitionList
-dataDefinitionList: DataDefinitionList
-selectionPanel: SelectionPanel
-dataPanel: DataPanel
-COLUMN_1_FIELD_WIDTH = 10: int
-COLUMN_2_FIELD_WIDTH = 20: int
+WeatherFrame()
+loadData(String fileName): void
KeyConstraints
-next: KeyConstraints
+KeyConstraints()
+getNext(): KeyConstraints
+addEnd(KeyConstraints elem): void
JMenuBar
JPanel
1
1
1
FileMenuBar
-menu: JMenu
-menuExit: JMenuItem
-menuOpen: JMenuItem
-fileChooser: JFileChooser
+FileMenuBar()
SelectionPanel
-stationList: JList<String>
-variableList: JList<String>
-yearList: JList<String>
-yearListModel: DefaultListModel<String>
-yearListValues: ArrayList<Integer>
-stationListScroller: JScrollPanel
-variableListScroller: JScrollPanel
-yearListScroller: JScrollPane
-stationLabel: JLabel
-variableLabel: JLabel
-yearLabel: JLabel
+SelectionPanel()
DataPanel
-stationLabel: JLabel
-variableLabel: JLabel
-minLabel: JLabel
-maxLabel: JLabel
-averageLabel: JLabel
-stationIdField: JTextField
-variableIdField: JTextField
-minVal: JTextField
-maxVal: JTextField
-averageVal: JTextField:
-variableUnitsField: JTextField
-minDateField: JTextField
-maxDateField: JTextField
-stationNameField: JTextField
-stationCityField: JTextField
-variableDescription: JTextArea
+DataPanel()
+updateData() «synchronized»

# Class Design Outline

Your project 3 code will largely stay the same. The key difference is that we want to be able to compute statistics in some cases over a subset of the available keys. For example, in this project, we would like to compute the maximum day for a statistic over years 2001, 2002, 2003, but exclude 2004 and 2005, even though all five years have been loaded. We will represent these "constrained queries" using a list of keys that we want to include in a given statistic computation.

These constraints will be represented using an instance of the **KeyConstraints** class. For this project, we will only constrain the years within a **DataSet**. However, in the next project we will also constrain the months within a **DataYear** and the days within a **DataMonth**. In order to represent these relationships in an efficient manner, the **KeyConstraints** class implements a linked list of constraints: the **KeyConstraints** over the years will point to a next **KeyConstraints** over the months. Likewise, the **KeyConstraints** over the months, will point to a **KeyConstraints** over the days.

Here are the key changes to your project 3 code:

- All **getStatisticAverage(), getStatisticMinDay()** and **getStatisticMaxDay()** methods will now take an additional **KeyConstraints** parameter. This parameter will be added to the end of the parameter list for each method implementation.

- **DataDay** will accept this parameter and ignore it.

- The **MultiStatisticsAbstract** class will use the **KeyConstraints** object to constrain which keys are searched in the iteration process. Furthermore, when this class calls the getStatisticXX() method on its sub-objects, it will pass **constraints.next()**. Note that if a constraint reference is ever **null**, then the getStatisticXX() method should use all of the available keys, as you are already doing in project 3.

- The **StationDefinition** and **StationDefinitionList** classes will simply pass the constraint down to the next level.

- The **DataYear** class:

  - Add the following property:

```
        private static TreeSet<Integer> yearList;
```

  - Initialize this property outside of your constructor.
  - Every time a day is added to the year, add the day's year to this set.
  - Provide the following accessor:

```
        public static ArrayList<Integer> getYearList()
        {
          return new ArrayList<Integer>(yearList);
        }
```

Below are the implementation notes for our Graphical User Interface:

- For this project, we are using **GridBagLayout** as the layout manager for our frames and panels.
- **WeatherFrame**: this class is-a **JFrame** and is the primary window of the interface.
  - Complete the implementation of the constructor
  - Complete the implementation of **loadData()**
- **FileMenuBar** is an inner class of **WeatherFrame** that is-a **JMenuBar**.
  - Complete the menu creation process
  - Complete the implementation of the open menu listener
- **SelectionPanel** is 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 selection type.
  - Complete the creation of the JLists
  - Implement the layout of the components
- **DataPanel** is an inner class that is-a **JPanel** that displays the selected information and the associated statistics.
  - Complete the creation of the **JTextFields**
  - Implement the layout of the components
  - Complete the implementation of the **updateData()** method. Note that this method is declared as being *synchronized*. While this keyword is beyond the scope of this class, you should leave it in place.

# 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. For example, 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** have **ActionListener**s attached to them to implement the functionality of selecting a menu item.

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

  **new File("./data")**

- **JLists** present a list of items to the user and allow 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 any change in what is selected. A change can be either the deselection of an item or the selection of an item (note that most "clicks" involve a sequence of deselection followed by selection). The currently selected element (if we assume that there is only one) can be read from the **JList** using **getSelectedValue()**.

  If a **JList** allows you to select more than one item, you can access the list of indices (in the presented list) using **getSelectedIndices()**.

  When the items are not known *a priori* or will change with time (as is the case with our 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 or cleared from. Every time this list changes, the **DefaultListModel** will automatically inform the **JList** that the list has changed, which, in turn, will cause the display to be updated. To attach a **ListModel** to a **JList**, you include a reference to the model in your call to the **JList** constructor.

- Each **JList** is placed 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. If the information fits, then the scroll bar is automatically hidden.

- **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. And, the text presented in the field can be selected and copied by a user through the use of mouse operations.

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

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

- Depending on server load, real-time grading of submissions may be halted at any time. Our priority is to let groups submit solutions in a timely fashion. If we do halt online grading, we will attempt to reenable it at a time where the load on the server is low. This means that you should not expect feedback on solutions that are submitted near to the deadline.

# 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 (five primary classes plus four JUnit test classes) and that all of your documented methods have the necessary documentation.

# Submission Instructions

- All required components (source code and compiled documentation) are due at 1:29:00 pm on Wednesday, November 16 (i.e, before class begins)

- Submit your project to Web-Cat using one of the two procedures documented in the Lab 2 specification.

# 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 Web-Cat server.

- Any time following the submission, you may do the code review with the instructor or one of 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 Canvas). 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). Note that schedule code review time **may not** be used for help with a lab or a project
  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 deadline, you have the option of going back to make changes and resubmitting (by the deadline). If you do this, you may need to return for another code review, as determined by the grader conducting the current code review

- The code review must be completed by Monday, November 21st to receive credit for the project

## Notes

## 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:

**Correctness/Testing: 45 points**

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against our set of tests. These unit tests will not be visible to you, but the Web-Cat server will inform you as to how many tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests).

**Style/Coding: 20 points**

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

**Design/Readability: 35 points**

This element will be assessed by a grader during the code review. Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names
- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

If you do not submit compiled Javadoc for your project, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

**Bonus: up to 5 points**

You will earn one bonus point for every twelve hours that your assignment is submitted early.

**Penalties: up to 100 points**

You will lose five points for every twelve hours that your assignment is submitted late (up to 48 hours). Submissions will not be accepted more than 48 hours after the deadline.