CS 2334
Project 3: Java Collections Framework

October 19, 2015

Due: 1:29 pm on Monday, Nov 2, 2015

Introduction

For the last two projects, you have been using data that are well-structured. In particular, you could assume ahead of time that you knew which stations were included in the data set and, for each station, you could assume that you knew which data elements were being recorded. In this project, we will break both of these assumptions. At run time, your program will load a pair of configuration files that will inform it of 1) the set of stations that are included in the data set, and 2) the set of measurements that are made at each station. Given this information, your program will create the data structures necessary to load in the data for the set of stations and to compute statistics over the individual measurements.

Your final product will:

1. Load in files that describe the set of measures taken (the variables) at the stations, the set of stations, and the data.
2. Query the user for a Mesonet station (single station only).
3. Query the user for a statistic name (single statistic only).
4. Report the minimum, maximum and average of that statistic over all data for that station.

Learning Objectives

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

1. Make an interactive menu for a user and handle errors in input
2. Make use of **HashMaps** and **TreeMaps** to flexibly store data in a structure that is efficient to access
3. Compute statistics over the stored data in a manner that does not rely on *a priori* knowledge of the specifics of the data
4. Continue to exercise good coding practices for Javadoc and for unit testing

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.

Refactoring

We are refactoring our project 2 code to implement this new project. The large changes to the problem include:

- All of the data are contained within a single file. The data loading process will be managed by the **StationList** class. The **DataSet** and **YearlyData** classes will provide an *add()* method that will add a new day to a DataSet or YearlyData object.
- Your code will not know *a priori* which variables are being measured by the stations. This information will be provided in a file that will describe the variableId, variable name, units and a textual description.
- Your code will not know *a priori* the list of stations that will be in the data file. This information will be provided in a station configuration file that includes the stationId, station name and city.

- We will not make substantial use of the **ArrayList** class. Instead, we will rely on the **HashMap** and **TreeMap** classes. This will allow us to be very flexible in the data that we store and efficient in our access of that data.
- The minimum, maximum and average statistics **will not** be computed as the data are loaded. Instead, these will be computed based on specific user queries.

Strategies for Success

- The UML specification constitutes the interface that we will rely on during our testing. Do not make changes to this interface.
- 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

Import the existing project3 implementation into your eclipse workspace:
http://www.cs.ou.edu/~fagg/classes/cs2334/projects/project2/project3-initial.zip

Example Interactions

Below are several examples of our implementation of **UserQuery** class interacting with a user. Your implementation should behave in the same way. Keep in mind that we will be testing many other cases when we evaluate your code.

```
#####################################################
Station ID   Name                      City
----------   ----                      ----
ACME         Acme                      Rush Springs
ADAX         Ada                       Ada
ALTU         Altus                     Altus
ALV2         Alva                      Alva
ALVA         Alva                      Alva
ANT2         Antlers                   Antlers
ANTL         Antlers                   Antlers
APAC         Apache                    Apache
ARD2         Ardmore                   Ardmore
ARDM         Ardmore                   Ardmore
ARNE         Arnett                    Arnett
BBOW         Broken Bow                Broken Bow
BEAV         Beaver                    Beaver
BEEX         Bee                       Tishomingo
BESS         Bessie                    Bessie
BIXB         Bixby                     Bixby
BLAC         Blackwell                 Blackwell
BOIS         Boise City                Boise City
BOWL         Bowlegs                   Bowlegs
BREC         Breckinridge              Breckinridge
BRIS         Bristow                   Bristow
BROK         Broken Bow                Broken Bow
BUFF         Buffalo                   Buffalo
BURB         Burbank                   Burbank
BURN         Burneyville               Burneyville
BUTL         Butler                    Butler
BYAR         Byars                     Byars
CALV         Calvin                    Calvin
CAMA         Camargo                   Camargo
CARL         Lake Carl Blackwell       Orlando
CATO         Catoosa                   Catoosa
CENT         Centrahoma                Centrahoma
CHAN         Chandler                  Sparks
CHER         Cherokee                  Cherokee
CHEY         Cheyenne                  Cheyenne
CHIC         Chickasha                 Chickasha
CLAR         Claremore                 Claremore
CLAY         Clayton                   Clayton
CLOU         Cloudy                    Cloudy
CLRM         Claremore                 Claremore
COOK         Cookson                   Marble City
COPA         Copan                     Copan
DURA         Durant                    Durant
ELRE         El Reno                   El Reno
ERIC         Erick                     Erick
EUFA         Eufaula                   Eufaula
FAIR         Fairview                  Fairview
FITT         Fittstown                 Fittstown
FORA         Foraker                   Foraker
FREE         Freedom                   Freedom
FTCB         Fort Cobb                 Fort Cobb
GOOD         Goodwell                  Goodwell
GRA2         Grandfield                Grandfield
GRAN         Grandfield                Grandfield
```

GUTH	Guthrie	Guthrie
HASK	Haskell	Haskell
HECT	Hectorville	Hectorville
HINT	Hinton	Hinton
HOBA	Hobart	Hobart
HOLD	Holdenville	Holdenville
HOLL	Hollis	Gould
HOOK	Hooker	Hooker
HUGO	Hugo	Hugo
IDAB	Idabel	Idabel
INOL	Inola	Inola
JAYX	Jay	Jay
KENT	Kenton	Kenton
KETC	Ketchum Ranch	Velma
KIN2	Kingfisher	Kingfisher
KING	Kingfisher	Kingfisher
LAHO	Lahoma	Lahoma
LANE	Lane	Lane
MADI	Madill	Lebanon
MANG	Mangum	Mangum
MARE	Marena	Coyle
MARS	Marshall	Marshall
MAYR	May Ranch	Freedom
MCAL	McAlester	McAlester
MEDF	Medford	Medford
MEDI	Medicine Park	Medicine Park
MIAM	Miami	Miami
MINC	Minco	Minco
MRSH	Marshall	Marshall
MTHE	Mt Herman	Smithville
NEWK	Newkirk	Newkirk
NEWP	Newport	Ardmore
NINN	Ninnekah	Ninnekah
NORM	Norman	Norman
NOWA	Nowata	Delaware
NRMN	Norman	Norman
OILT	Oilton	Oilton
OKCE	Oklahoma City East	Oklahoma City
OKCN	Oklahoma City North	Oklahoma City
OKCW	Oklahoma City West	Oklahoma City
OKEM	Okemah	Okemah
OKMU	Okmulgee	Morris
PAUL	Pauls Valley	Pauls Valley
PAWN	Pawnee	Pawnee
PERK	Perkins	Perkins
PORT	Porter	Clarksville
PRES	Preston	Preston
PRYO	Pryor	Adair
PUTN	Putnam	Putnam
REDR	Red Rock	Red Rock
RETR	Retrop	Willow
RING	Ringling	Ringling
SALL	Sallisaw	Sallisaw
SEIL	Seiling	Seiling
SHAW	Shawnee	Shawnee
SKIA	Skiatook	Skiatook
SLAP	Slapout	Slapout
SPEN	Spencer	Spencer

```
STIG        Stigler              Stigler
STIL        Stillwater           Stillwater
STUA        Stuart               Stuart
SULP        Sulphur              Sulphur
TAHL        Tahlequah            Tahlequah
TALA        Talala               Talala
TALI        Talihina             Talihina
TIPT        Tipton               Tipton
TISH        Tishomingo           Tishomingo
TULL        Tullahassee          Tullahassee
TULN        Tulsa                Tulsa
VANO        Vanoss               Vanoss
VINI        Vinita               Centralia
WAL2        Walters              Walters
WALT        Walters              Walters
WASH        Washington           Washington
WATO        Watonga              Watonga
WAUR        Waurika              Waurika
WEAT        Weatherford          Weatherford
WEBB        Webbers Falls        Webbers Falls
WEBR        Webbers Falls        Webbers Falls
WEST        Westville            Westville
WILB        Wilburton            Wilburton
WIST        Wister               Wister
WOOD        Woodward             Woodward
WYNO        Wynona               Wynona
Please choose a station (or "END" to quit):
SHAW
Variable ID Name                                              Units
----------- ----                                              -----
2AVG        Average Wind Speed at 2m                          miles per hour
2DEV        Standard Deviation of Wind Speed at 2m            miles per hour
2MAX        Maximum 2m Wind Speed                             miles per hour
2MIN        Minimum 2m Wind Speed                             miles per hour
9AVG        Average Air Temperature at 9m                     degrees ↩
    Fahrenheit
AMAX        Maximum Solar Radiation                           Watts per square↩
     meter
ATOT        Total Solar Radiation                             mega Joules per ↩
    square meter
BAVG        Average Temperature Under Bare Soil at 10cm       degrees ↩
    Fahrenheit
BMAX        Maximum Temperature Bare Soil at 10cm             degrees ↩
    Fahrenheit
BMIN        Minimum Temperature Under Native Vegetation at 10cm  degrees ↩
    Fahrenheit
CDEG        Cooling Degree Days                               degrees ↩
    Fahrenheit
DAVG        Average Dewpoint Temperature                      degrees ↩
    Fahrenheit
DMAX        Maximum Dewpoint Temperature                      degrees ↩
    Fahrenheit
DMIN        Minimum Dewpoint Temperature                      degrees ↩
    Fahrenheit
HAVG        Average Humidity                                  percent
HDEG        Heating Degree Days                               degrees ↩
    Fahrenheit
HMAX        Maximum Humidity                                  percent
```

```
HMIN         Minimum Humidity                                           percent
HTMX         Maximum Heat Index Temperature                             degrees ↩
    Fahrenheit
MSLP         Mean Sea Level Pressure                                    inches of ↩
    mercury
PAVG         Average Station Pressure                                   inches of ↩
    mercury
PMAX         Maximum Station Pressure                                   inches of ↩
    mercury
PMIN         Minimum Station Pressure                                   inches of ↩
    mercury
RAIN         Rain                                                       inches
SAVG         Average Temperature Under Native Vegetation at 10cm        degrees ↩
    Fahrenheit
SMAX         Maximum Temperature Under Native Vegetation at 10cm        degrees ↩
    Fahrenheit
SMIN         Minimum Temperature Under Native Vegetation at 10cm        degrees ↩
    Fahrenheit
TAVG         Average Air Temperature                                    degrees ↩
    Fahrenheit
TMAX         Maximum Daily Air Temperature                              degrees ↩
    Fahrenheit
TMIN         Minimum Daily Air Temperature                              degrees ↩
    Fahrenheit
VDEF         Average Daily Vapor Deficit                                millibars
WCMN         Minimum Wind Chill Index Temperature                       degrees ↩
    Fahrenheit
WDEV         Standard Deviation of Wind Speed at 10m                    miles per hour
WMAX         Maximum Wind Gust                                          miles per hour
WSMN         Minimum Wind Speed                                         miles per hour
WSMX         Maximum Wind Speed                                         miles per hour
WSPD         Average Wind Speed                                         miles per hour
Please choose a variable (or "END" to quit):
DAVG
Station: SHAW, Shawnee, Shawnee
Variable: DAVG, Average Dewpoint Temperature (degrees Fahrenheit)
Average: 49.87511010823314 degrees Fahrenheit
Maximum: 75.85 degrees Fahrenheit on 6/27/1999 at SHAW
Minimum: 0.65 degrees Fahrenheit on 1/3/1999 at SHAW
Hit <ENTER> to continue
Done..
```

Note: several of the above lines have been wrapped to the next line. The arrows are not part of the program output.

```
#########################################################
Station ID   Name                      City
----------   ----                      ----
ACME         Acme                      Rush Springs
:
:
WYNO         Wynona                    Wynona
Please choose a station (or "END" to quit):
TULS
Please choose a station (or "END" to quit):
TULN
Variable ID  Name                                                  Units
-----------  ----                                                  -----
2AVG         Average Wind Speed at 2m                              miles per hour
:
:
WSPD         Average Wind Speed                                    miles per hour
Please choose a variable (or "END" to quit):
WMAX
Station: TULN, Tulsa, Tulsa
Variable: WMAX, Maximum Wind Gust (miles per hour)
Average: invalid
Maximum: invalid
Minimum: invalid
Hit <ENTER> to continue
Done..
```

```
#########################################################
Station ID   Name                      City
----------   ----                      ----
ACME         Acme                      Rush Springs
:
:
WYNO         Wynona                    Wynona
Please choose a station (or "END" to quit):
tala
Variable ID  Name                                                  Units
-----------  ----                                                  -----
2AVG         Average Wind Speed at 2m                              miles per hour
:
:
WSPD         Average Wind Speed                                    miles per hour
Please choose a variable (or "END" to quit):
htmx
Station: TALA, Talala, Talala
Variable: HTMX, Maximum Heat Index Temperature (degrees Fahrenheit)
Average: invalid
There are no valid measurements.
Hit <ENTER> to continue
Done..
```

```
##########################################################
Station ID   Name                       City
-----------  ----                       ----
ACME         Acme                       Rush Springs
ADAX         Ada                        Ada
:
:
WYNO         Wynona                     Wynona
Please choose a station (or "END" to quit):
foo
Please choose a station (or "END" to quit):
56
Please choose a station (or "END" to quit):

Please choose a station (or "END" to quit):
Sulp
Variable ID  Name                                                     Units
-----------  ----                                                     -----
2AVG         Average Wind Speed at 2m                                 miles per hour
:
:
WSPD         Average Wind Speed                                       miles per hour
Please choose a variable (or "END" to quit):
bar
Please choose a variable (or "END" to quit):
baz
Please choose a variable (or "END" to quit):
45 34 jafd
Please choose a variable (or "END" to quit):

Please choose a variable (or "END" to quit):
BAvG
Station: SULP, Sulphur, Sulphur
Variable: BAVG, Average Temperature Under Bare Soil at 10cm (degrees Fahrenheit)
Average: 66.41428053442884 degrees Fahrenheit
Maximum: 96.42 degrees Fahrenheit on 7/13/1995 at SULP
Minimum: 32.89 degrees Fahrenheit on 1/8/1996 at SULP
Hit <ENTER> to continue
Done..
```

Class Design Outline

Because your program will not know at compile time what the set of variables will be, we must fundamentally change the way that we are representing that data. Specifically, variables will be identified using a String *variableId*. These IDs will then be used to look-up the assoicated Observation value, as well compute the minimum, maximum and average of a variable. Likewise, your program will not know ahead of time what the set of stations will be. This set will be loaded at run time.

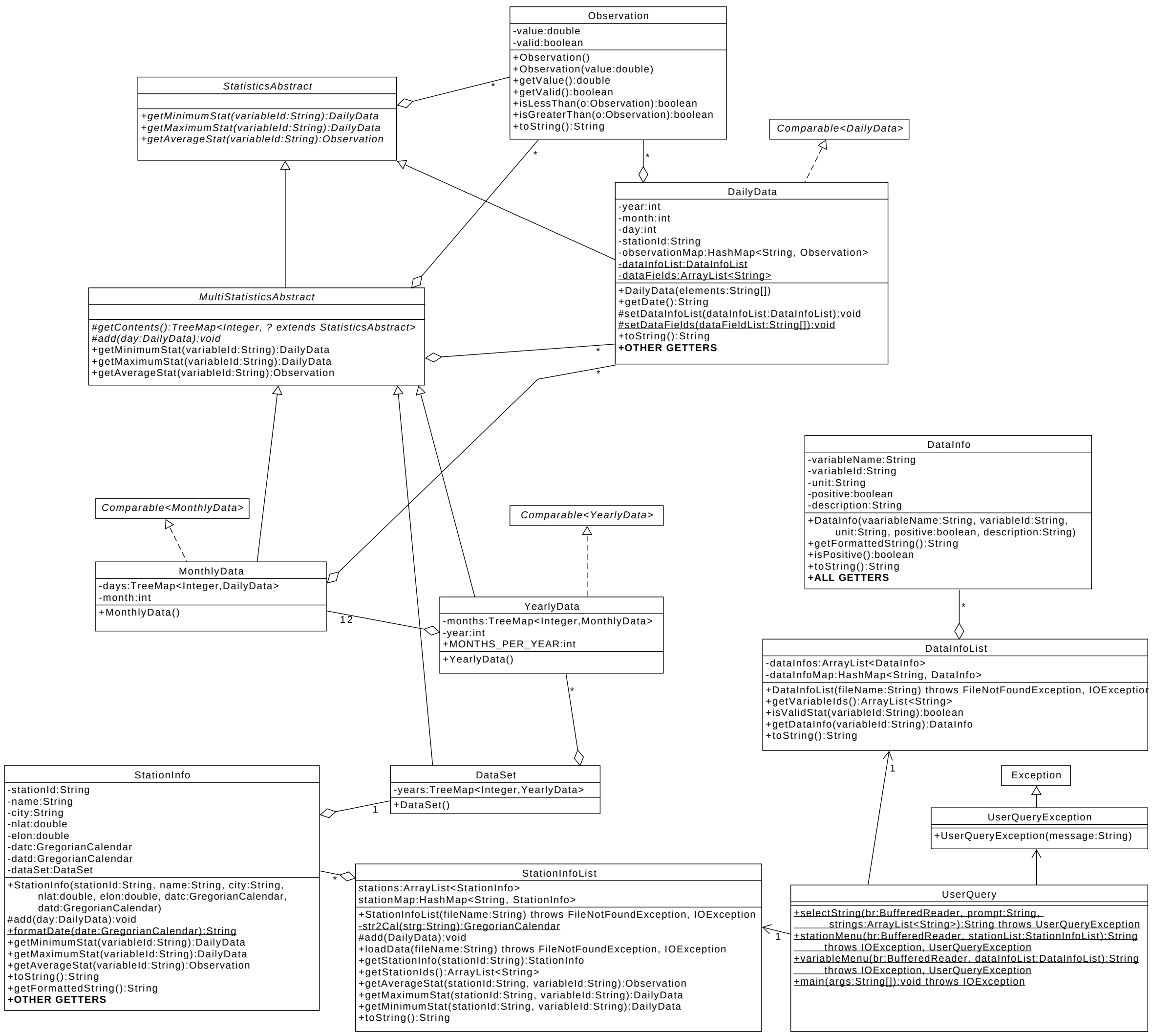
- The **DataInfo** class will represent the information about a single variable, including its ID, a text description and the variables's units.
- The **DataInfoList** class will represent all of the possible variables that are stored for a single station. The constructor will load a configuration file (DataTranslation.csv); each line of this file encodes a single variable.
- The **StationInfo** class will represent the information about a station, including its ID, name and location, as well as a text description of the station and dates that the station started and stopped recording data.
- The **StationInfoList** class will represent the information about all of the stations. The constructor will load a configuration file (geoinfo.csv); each line of this file encodes one station.
- Many classes implement an *add(DailyData day)* method. In all cases, this is about adding a new day to the data structure. In each class, the method must decide how to handle this day. For the case of a YearlyData object, it must decide which month to add the day to and then add the day to that month.
- Many classes implement a *getAverageStat(String stationId)* method that returns an **Observation**. The **MultiStatisticsAbstract** class performs this computation across all of the contents of *this* object (e.g., over all of the months contained within *this* year). For other classes, implementation of this method is a matter of looking up the correct value (in the case of a **DailyData** object) or of asking a sub-object what the average statistic is.
- *getMinimumStat()* and *getMaximumStat()* are the same as above.
- The **MultiStatisticAbstract** child classes all represent their sub-objects using a **TreeMap**. Keys for the **TreeMap** are Integers (e.g., corresponding to the year in the case of a **DataSet** object). We use **TreeMap** because we

want to preserve the order of the keys when we perform searches for the minimum and maximum values of a variable. The implementation required by the **Comparable**`<T>` interface determines the ordering of these keys.

- The **DailyData** class will no longer explicitly represent variables for every possible **Observation**. Instead, this class will use a HashMap to map a variableId to an instance of an Observation for that variable.
- Likewise, any class that extends the **MultiStatisticsAbstract** class will not explicitly represent specific variables. Instead, the variableId will be used to query value of the variable and to compute the minimum, maximum and average of the variable.
- **DataSet** will contain all of the years associated with a specific station.

Below is a complete UML diagram for our key classes.
Notes:

- If an abstract method is defined in an abstract class or an interface, and it is not listed elsewhere in the diagram, then the method must be implemented by the concrete class.



Project Components

We provide an initial implementation for most classes. Please start from these implementations. Where it is useful, it is fine to copy implementations from project 2.

1. Implement your **DataInfo** and **DataInfoList** classes. Use the *DataTranslation.csv* file as a guide to how these should be implemented.
2. Implement the **DataInfoListTest** class.
3. Implement your **StationInfo** and **StationList** classes. Use the *geoinfo.csv* file as a guide to how these should be implemented.
4. Implement the **StationListTest** class.
5. Copy your working **Observation** class (assuming that it is correctly implemented).
6. Keep, and possibly update, the unit tests for the **Observation** class within the **ObservationTest** class.
7. Update your **StatisticsAbstract** class according to the UML.
8. Update your **DailyData** implementation to match the requirements of the UML.
9. Update your **DailyDataTest** class to test the new implementation of the class.
10. Update the **MultiStatisticsAbstract** to match the UML specification.
11. Modify your classes called **MonthlyData**, **YearlyData** and **DataSet** classes.
12. Create a unit test that loads the variable list, station list and data, then carefully tests the minimum, maximum and average statistics computations for multiple stations and variables.
13. Copy your **UserQueryException** class from project 2.
14. Update your **UserQuery** class that contains your **main** method. This method must:
    - Load the variable list, station list and data.

- Query the user for the desired station
- Query the user for the desired variable
- Report the average, maximum and minimum statistics for that station's variable
- Repeat the queries until the user enters "END"

Assumptions

You may make the following simplifying assumptions:

1. All stations in the data set will be included in the geoinfo.csv file. The columns in this file are pre-defined (i.e., they won't change).

2. All files exist and are properly-formatted csv files.

3. The data file will include the following columns: YEAR, MONTH, DAY, STID.

4. All variable values in the data set file can be interpreted as doubles.

Notes

1. The first line of the data file determines the names of each of the fields in this file (each column is a field). While most of these columns are variables that are measured by the stations, not all are. In your DailyData objects, you should only include fields that correspond to variables (as determined by the *isValidStat()* method.

2. *getMinimumStat()* and *getMaximumStat()* may return null if no DailyData objects are associated with the station.

Final Steps

1. Generate Javadoc using Eclipse for all of your classes.
2. Open the *project3/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 pm on Monday, November, 2nd (i.e, before class begins).
- Prepare your submission file by creating a project3.zip file. This file must include your entire project, including: src, and doc
- Submit your zip file to the project3 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, November 16th to receive credit for the project.

References

- The Java API: `https://docs.oracle.com/javase/8/docs/api/`
- The Oklahoma Mesonet: `http://www.mesonet.org`
- The API of the *Assert* class can be found at:
  `http://junit.sourceforge.net/javadoc/org/junit/Assert.html`
- JUnit tutorial in Eclipse:
  `https://dzone.com/articles/junit-tutorial-beginners`

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.

Unit Tests: 10 points

- (10) A complete set of unit tests has been implemented.
- (7) One key unit test is missing.
- (4) Two key unit tests are missing.
- (0) Three or more key unit tests 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.