CS 2334
Project 1: Reading Data from Files

September 10, 2015

Due: 1:29 pm on Sept 23, 2015

Introduction

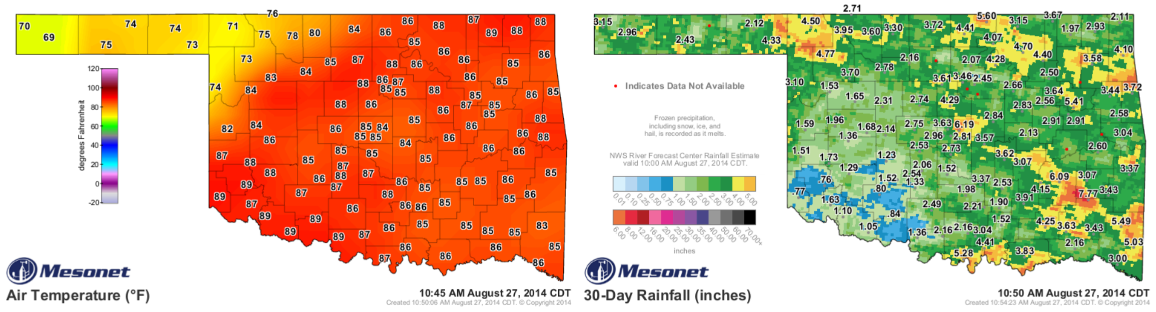
The Oklahoma Mesonet[1] is a network of weather observation stations that is unique to the State of Oklahoma. This network provides a variety of weather observations for every county in Oklahoma every five minutes. Having been in existence for 20 years, it is an invaluable resource for understanding our weather and climate. The data for our projects this semester will be derived from this set of observation stations.

For this project, you will focus on a single station (Norman) and only temperature and rainfall data. Later projects will expand to other stations and additional weather data. The data that we provide to you is a comma separated file (CSV format) with the daily summary information for Norman.

We have provided 5 years of temperature and rainfall data for Norman and your job is to read in the data files, parse the data, create appropriate objects from this data, and summarize the data using maximum, minimum, and average mathematical functions. You will also continue to expand your use of unit tests beyond the lab and ensure that your parsing and mathematical functions are correct. More details are below in the Project Components section.

Note: due to unforeseen circumstances, such as extended power outages, sometimes data is unavailable for a day at a particular station. These situations are represented using values of -900 and below. Make sure you don't accidentally use these values in your statistical summaries.

[1] `http://www.mesonet.org`


Mesonet
Air Temperature (°F)
degrees Fahrenheit
10:45 AM August 27, 2014 CDT
Indicates Data Not Available
Frozen precipitation, including snow, ice, and hail, is recorded as it melts.
NWS River Forecast Center Rainfall Estimate valid 10:00 AM August 27, 2014 CDT.
inches
Mesonet
30-Day Rainfall (inches)
10:50 AM August 27, 2014 CDT

(a) Temperatures on August 27, 2014 (b) Rainfall for August, 2014

Figure 1: Example Mesonet Data

Learning Objectives

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

1. Parse structured data from a file
2. Create objects using data parsed from a file
3. Use mathematical transformations on Strings
4. Implement mathematical functions in code
5. Employ unit testing to ensure that different pieces of your code are functioning properly
6. Provide proper documentation in Javadoc format

Proper Academic Conduct

This lab 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

- 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 project1 implementation into your eclipse workspace:
`http://www.cs.ou.edu/~fagg/classes/cs2334/projects/project1/project1.zip`

Project Components

As we begin to develop large programs, it becomes harder to keep all of the details in mind at once. A key skill to success in computer science is learning how to chop big problems into small, manageable ones. In part, this involves process (separating design from implementation and from testing), but it also involves cutting the implementation into logical pieces that are clearly independent and have simple interfaces between them. This list of project components will serve to guide you in the design and development of your project.

1. Develop and use a proper design using the Unified Modeling Language (UML)
    - UML will help you to organize your program. For many of you, this is the largest program that you have written. The UML design process will help us see what the individual classes must do and represent, and how the set of classes fits together. The UML diagram for this project is given in the next section. In future projects, you will be developing your own UML diagrams.
2. Use proper documentation and formatting (Javadoc and in-line documentation)

- This is important for debugging and for communication with your project partner and your future, possibly sleep deprived, self. You may re-use your project code in future projects this semester, so don't make it obfuscated.
- Use the same documentation standards that we established for Lab 1.

3. Create a class called **Observation**

   - This immutable class contains a single observation value (a double called *value*) and a boolean flag called *valid*.
   - This class contains a single constructor. On construction, if the assigned value is valid (greater than -900), then *valid* is set to *true*. Otherwise, this flag is set to false.
   - This class must contain an appropriate set of getters (using the standard names).
   - This class must contain an appropriate *toString()* method that will return a string containing the value if it is valid and "invalid" if it is not valid.

4. Implement unit tests for the **Observation** class

5. Create a class called **DailyData** that will hold the daily data for a station.

   - Examine one of the CSV files that we have provided in the project (see the *data* folder)
   - This immutable class contains instance variables for the year, month and day (all ints that are named accordingly)
   - This class contains a String for the station ID (called *stationID*)
   - This class contains *Observations* for the maximum, minimum and average temperature, and for the total rain fall (called *temperatureMax, temperatureMin, temperatureAverage, rainFall*)
   - This class contains a single constructor
   - This class must include an appropriate set of getters (using the standard names)
   - This class must contain an appropriate *toString()* method.

6. Implement unit tests for the **DailyData** class

7. Create a class called **MonthlyData** that will represent an entire month's of observations.

   - This immutable class will include an instance variable of type *ArrayList<DailyData>* called *days*
   - This class will also include instance variables (all doubles) called *temperatureMax, temperatureMin, temperatureAverage, rainMax, rainMin,* and *rainAverage*
   - This class will include instance variables that will represent the year and month (both are ints)
   - The only constructor will take as input the name of a file (a String), and then will:
     - Read in the data for the individual days and add these days to the ArrayList. You must use a *BufferedReader* for this task
     - Using a pair of private helper methods, this constructor will fill in the minimum, maximum and average temperature and rain fall instance variables. Make sure to ignore invalid values during the computation of these statistics.
   - You may assume that all months have at least one valid day's worth of data
   - This class must include an appropriate set of getters (using the standard names)
   - This class must contain an appropriate *toString()* method.

8. Implement unit tests for the **MonthlyData** class.

   - Make a file with data that you know is correct (just use a small number of days) and verify that the max/min and averages that are computed properly

9. Create a class called **DataSet** that will represent data from a set of months

   - This class must include the following instance variables:
     - An ArrayList of **MonthlyData**, called *months*

    - Primitive doubles that represent the minimum, maximum and average rainfall and temperature across all of the months, called *rainMin, rainMax, rainAverage, temperatureMin, temperatureMax, temperatureAverage*
    - For each minimum and maximum, a reference to the *MonthlyData* that contains the minimum/maximum value. These instance variables are called *rainMinMonth, rainMaxMonth, temperatureMinMonth, temperatureMaxMonth*
  - The constructor for this class must:
    - Take as input an array of Strings (one for each data file)
    - Load in all of the months into the ArrayList
    - Call private helper methods that compute the minimum, maximum and average temperature and rainfall (one private method for each of temperature and rainfall). Note that averages are computed over the average temperature/rainfall for each month (not the average of all the days contained within all of the months)

10. Implement unit tests for the **DataSet** class

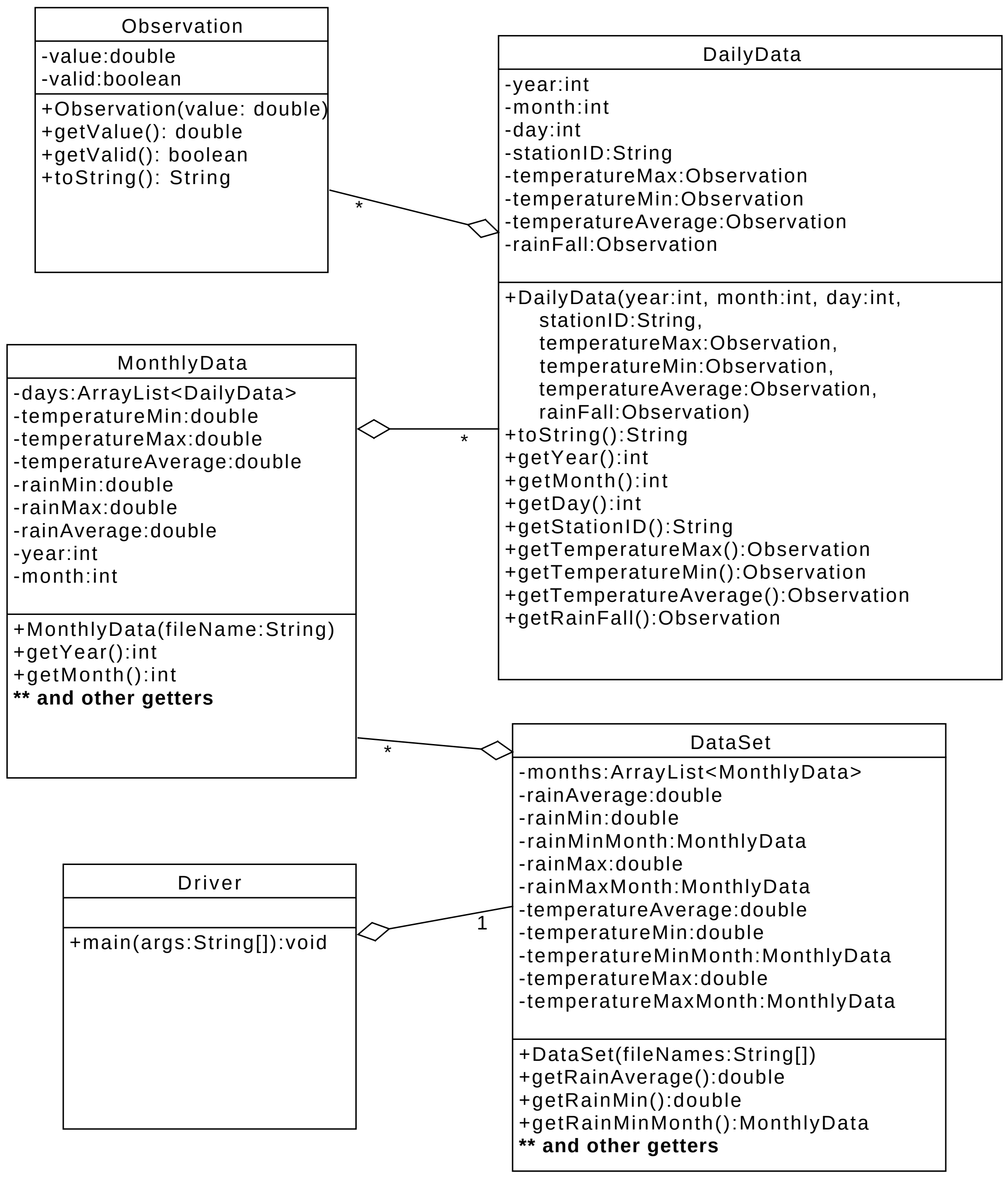
    - Make two files with data that you know is correct (just do a small number of days for each) and verify that the max/min and averages that are computed properly

11. Create a class called **Driver** that contains your **main** method. This Driver will:

    - Create a **DataSet** instance from all of available monthly data files
    - Report the minimum, maximum and average temperature and rainfall across the months. For the minimum and maximum values, this method will also report the month and year in which the minimum and maximum occurred. In the case of ties, then the first one to be found should be reported

UML Design

Below is a nearly-complete UML diagram for our key classes.


Observation
-value:double
-valid:boolean
+Observation(value: double)
+getValue(): double
+getValid(): boolean
+toString(): String
*
DailyData
-year:int
-month:int
-day:int
-stationID:String
-temperatureMax:Observation
-temperatureMin:Observation
-temperatureAverage:Observation
-rainFall:Observation
+DailyData(year:int, month:int, day:int, stationID:String, temperatureMax:Observation, temperatureMin:Observation, temperatureAverage:Observation, rainFall:Observation)
+toString():String
+getYear():int
+getMonth():int
+getDay():int
+getStationID():String
+getTemperatureMax():Observation
+getTemperatureMin():Observation
+getTemperatureAverage():Observation
+getRainFall():Observation
MonthlyData
-days:ArrayList<DailyData>
-temperatureMin:double
-temperatureMax:double
-temperatureAverage:double
-rainMin:double
-rainMax:double
-rainAverage:double
-year:int
-month:int
+MonthlyData(fileName:String)
+getYear():int
+getMonth():int
** and other getters
*
*
DataSet
-months:ArrayList<MonthlyData>
-rainAverage:double
-rainMin:double
-rainMinMonth:MonthlyData
-rainMax:double
-rainMaxMonth:MonthlyData
-temperatureAverage:double
-temperatureMin:double
-temperatureMinMonth:MonthlyData
-temperatureMax:double
-temperatureMaxMonth:MonthlyData
+DataSet(fileNames:String[])
+getRainAverage():double
+getRainMin():double
+getRainMinMonth():MonthlyData
** and other getters
Driver
+main(args:String[]):void
1

Final Steps

1. Generate Javadoc using Eclipse for all of your classes.
2. Open the *project1/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 Wednesday, September 23rd (i.e, before class begins).
- Prepare your submission file by creating a project1.zip file. This file must include your entire project, including: src, and doc
- Submit your zip file to the project1 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 10-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 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 Friday, October 2nd to receive credit for the project.

Notes

Some classes/methods will raise one or more of the following exceptions: *IOException, NumberFormatException, FileNotFoundException*. It is OK in this project if you deal with this by having your methods *throw* these exceptions, as well. Note that multiple methods will need to do this, including your main() method.

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

(15) The program passes all unit tests (these are unit tests that we provide).

(10) The program fails one test.

(5) The program fails two tests.

(0) The program fails three or more tests.

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: 5 points

(5) 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.

Inline Documentation: 10 points

(10) Every method contains appropriate inline documentation.

(7) 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: 10 points

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

(5) 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.