CS 2334
Project 1: Reading Data from Files

September 7, 2016

Due: 1:29 pm on Sept 21, 2016

Introduction

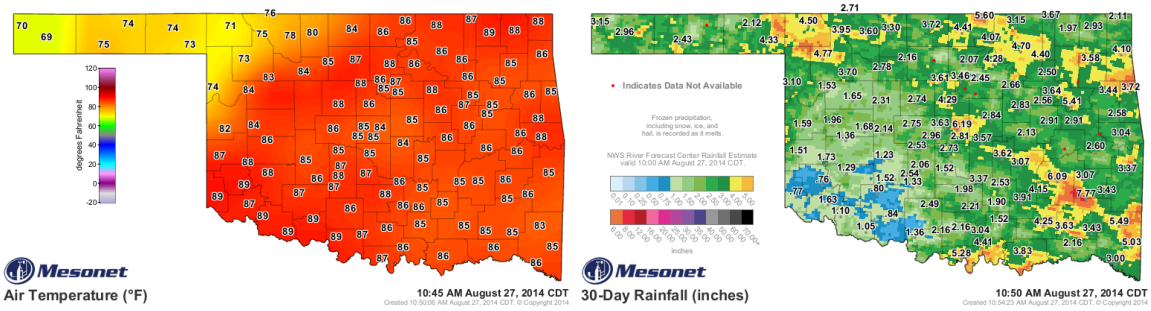
The Oklahoma Mesonet[1] is a network of weather observation stations that is unique to the State of Oklahoma. This network provides a variety of weather observations for every county in Oklahoma every five minutes. Having been in existence for 20 years, it is an invaluable resource for understanding 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 (Tishimingo) and only wind and solar radiation 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 Tishimingo.

We have provided several years of wind speed and solar radiation data for Tishimingo and your job is to read in the data files, parse the data, create appropriate objects from these 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 or sensor errors, sometimes the data is unavailable for a day at a particular station or for individual sensors. These situations are represented in the files using values of -900 and below. Make sure you don't accidentally use these values in your statistical summaries.

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



(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, and
6. provide proper documentation in Javadoc format.

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. Your group 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, as it cannot be completed in a single day.
- Implement and test your project components incrementally. Don't wait until your entire implementation is done to start the testing process. We suggest that you start with the lowest-level classes (*Sample*) and work your way up to highest-level (*DataMonth*).
- 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

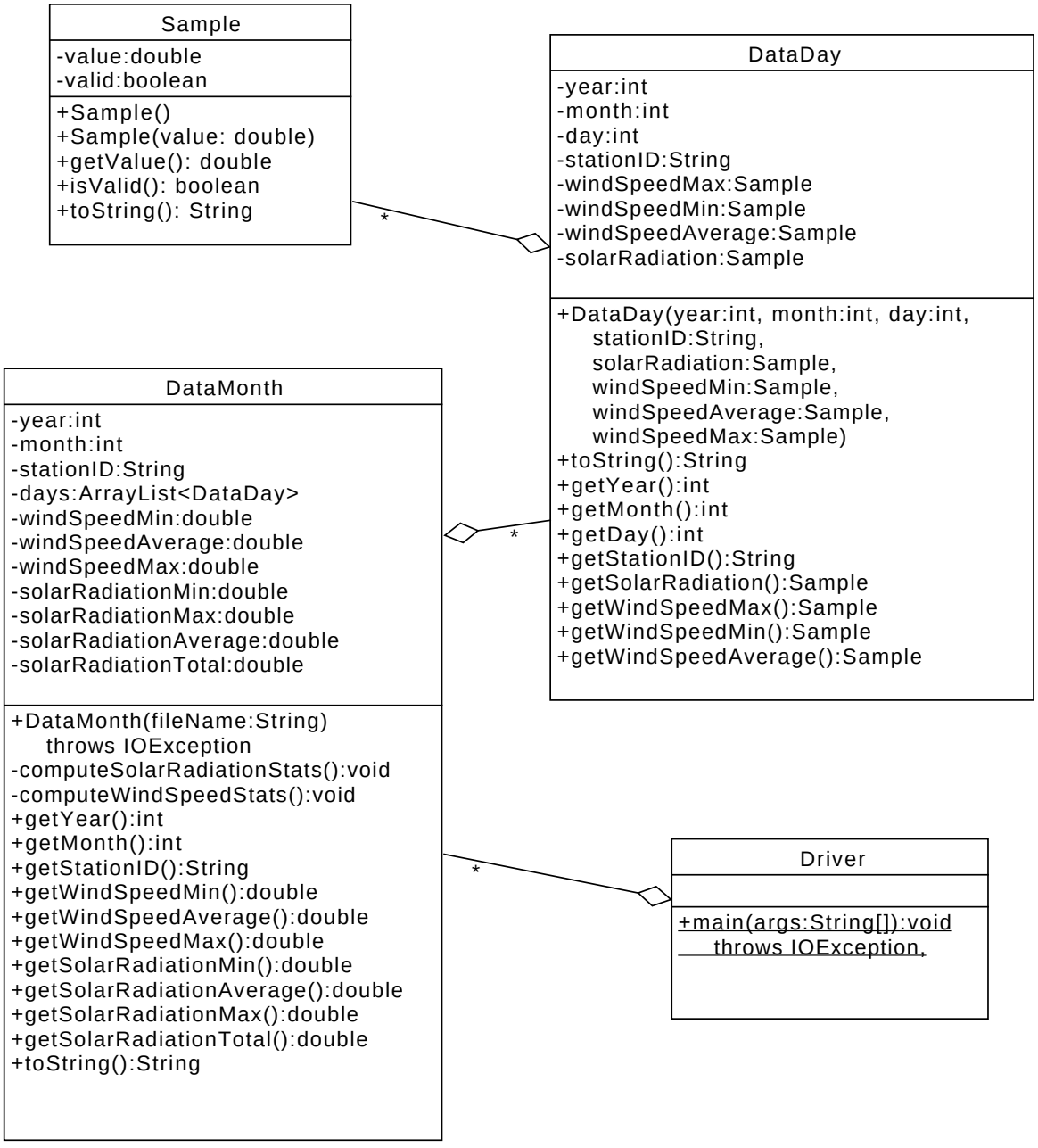
Import the existing project1 implementation into your eclipse workspace:

1. Download:
   http://www.cs.ou.edu/~fagg/classes/cs2334/projects/project1/project1.zip
2. *File* menu: *Import*
3. Select *General/Existing Projects into Workspace* and then click *Next*
4. *Select archive file*: browse to the project1.zip file
5. Click *Finish*
6. Once you create the new project, it will not initially know where to find the standard Java libraries (it varies depending on your configuration). In one of the provided Java files, find an undefined reference to a standard class (e.g., String) and mouse over it. Java will provide a list of ways to repair the problem:
   (a) Select *Fix project setup*
   (b) Select *Add library: JRE System Library*
   (c) Click *OK*

Project Design

As we begin to develop large programs, it becomes harder to keep all of the details in your mind at once. A key skill for success in computer science is learning how to chop big problems into small, manageable ones. In part, this involves the process that you use to solve the problem (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. We first summarize all of the key classes in the form of a UML diagram (next page), and then discuss each class in detail.

UML Design


Sample
-value:double
-valid:boolean
+Sample()
+Sample(value: double)
+getValue(): double
+isValid(): boolean
+toString(): String
*
DataDay
-year:int
-month:int
-day:int
-stationID:String
-windSpeedMax:Sample
-windSpeedMin:Sample
-windSpeedAverage:Sample
-solarRadiation:Sample
+DataDay(year:int, month:int, day:int,
stationID:String,
solarRadiation:Sample,
windSpeedMin:Sample,
windSpeedAverage:Sample,
windSpeedMax:Sample)
+toString():String
+getYear():int
+getMonth():int
+getDay():int
+getStationID():String
+getSolarRadiation():Sample
+getWindSpeedMax():Sample
+getWindSpeedMin():Sample
+getWindSpeedAverage():Sample
*
DataMonth
-year:int
-month:int
-stationID:String
-days:ArrayList<DataDay>
-windSpeedMin:double
-windSpeedAverage:double
-windSpeedMax:double
-solarRadiationMin:double
-solarRadiationMax:double
-solarRadiationAverage:double
-solarRadiationTotal:double
+DataMonth(fileName:String)
throws IOException
-computeSolarRadiationStats():void
-computeWindSpeedStats():void
+getYear():int
+getMonth():int
+getStationID():String
+getWindSpeedMin():double
+getWindSpeedAverage():double
+getWindSpeedMax():double
+getSolarRadiationMin():double
+getSolarRadiationAverage():double
+getSolarRadiationMax():double
+getSolarRadiationTotal():double
+toString():String
*
Driver
+main(args:String[]):void
throws IOException,

Classes and Other Components

1. 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

2. Create a class called **Sample**

    - This immutable class contains a single sample value (a double called *value*) and a Boolean flag (called *valid*). The flag indicates whether the sample is valid
    - The default constructor creates an invalid sample.
    - A second constructor accepts a single double value. On construction, if this value is valid (greater than -900), then the *valid* property is set to *true*. Otherwise, this property is set to false. We expect that a user of this class will only ask for the value of a Sample if the sample is known to be valid
    - This class contains a complete set of getters, using the standard names. Note that there are no setters
    - This class contains an appropriate *toString()* method that will return a string containing the value if the sample is valid (with exactly 4 digits after the decimal point) and "invalid" if the sample is not valid. Two examples:

```
1 invalid
2 98.3480
```

3. Implement unit tests for the **Sample** class. These tests should cover all possible cases

4. Create a class called **DataDay** 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 *Samples* for the maximum, minimum and average wind speed, and for the total solar radiation (called *windSpeedMax, windSpeedMin, windSpeedAverage, solarRadiation*)
- This class contains a single constructor. The order of the arguments is specified in the UML diagram
- This class must include an appropriate set of getters (using the standard names)
- This class must contain an appropriate *toString()* method. An example return value from this method is as follows:

```
2004-10-08, UPLAND: Wind = [0.1000, 13.2300, 26.8930], Solar Radiation = ↩
    15.5700
```

where the *Wind* values correspond to the minimum, average and maximum values, respectively.

Note that the character at the right-hand-side (the arrow) only indicates a line wrap in this document and is not included as part of the returned String

5. Implement unit tests for the **DataDay** class. These tests should cover your entire class

6. Create a class called **DataMonth** that will represent an entire month of samples

   - This immutable class will include an instance variable of type *ArrayList<DataDay>* called *days*
   - This class will also include instance variables (all doubles) called *windSpeedMax, windSpeedMin, windSpeedAverage, solarRadiationMax, solarRadiationMin, solarRadiationAverage* and *solarRadiationTotal*
   - This class will include instance variables that will represent the year and month (both are ints), and stationID (a String)
   - 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 project**. We will cover details of this in Lab 3.
  - Using a pair of private helper methods, this constructor will fill in the minimum, maximum and average wind speed and solar radiation instance variables, as well as the total solar radiation instance variable. Make sure to ignore invalid values during the computation of these statistics

- You may assume that each file contains at least one valid sample for each measure and contains only one year and one month for a single station. You may also assume that the CSV files are consistently formatted.
- This class must include the set of getters given in the UML diagram (using the standard names)
- This class must contain an appropriate *toString()* method. An example output from this method (for June, 2016) is as follows:

```
2016-06, TISH: Wind = [0.0000, 6.0417, 23.7100], Solar Radiation = ↩
    [4.6000, 23.3545, 30.7200, 677.2800]
```

  where the *Wind* values correspond to the minimum, average and maximum values, respectively, and where the *Solar Radiation* values correspond to the minimum, average, maximum and total, respectively.

7. Implement unit tests for the **DataMonth** class.

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

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

   - Create a **DataMonth** instance from a specified data file (this file name may be "hard coded")
   - Report the String from the month's toString() method

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 (four primary classes plus three Junit test classes) and that all of your methods have the necessary documentation

Submission Instructions

- All required components (source code and compiled documentation) are due at 1:29 pm on Wednesday, September 21st (i.e, before class begins)

- Submit your project to Web-Cat using one of the two procedures documented in the Lab 2 specification. As we get closer to the deadline, we will tell you which Web-Cat server to use

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 Wednesday, September 28th 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 (Eclipse will tell you where to do this). Later in the semester, we will cover the details of Exceptions.

The largest value that can be represented by a double is *Double.POSITIVE_INFINITY*, and the most negative value is *Double.NEGATIVE_INFINITY*.

In our UML diagram, we are being very prescriptive of the required object properties and methods (and their visibility). **Do not alter this design.**

Testing: the different "@test" methods will be executed in an arbitrary order (don't assume the implementation order). In some cases, however, you may wish to execute a method first that creates or loads in a data structure that is then used by multiple test methods.

1. Declare the shared data structure elements as *static* class variables of the test class

2. Initialize these data structures using a method that is declared as "@BeforeClass". Remember that you will need to import the BeforeClass class

(Eclipse will give you the right option if you mouse over the undefined BeforeClass reference.

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:

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.