Introduction to Computer Programming (CS 1323)
Project 1

September 22, 2014

Introduction

The Oklahoma Mesonet is a network of 120 weather stations scattered across all counties in the state. Every five minutes, each of these stations report air temperature and relative humidity, rainfall barometric pressure, solar radiation and wind speed and direction. With information available at this fine spatial scale, meteorologists are able to study and make predictions about a range of weather phenomena, including: wind gusts, heat bursts, thunderstorms and dry lines.

In this project, we will simulate the processing of air temperature data from a single station. Your program will prompt a user to input a sequence of temperatures and compute their mean. Because the temperature sensors in our simulation occasionally produce erroneous reports, your program must re-prompt for temperatures that fall below -10^oC (i.e., only "correct" temperatures will be included in the mean). If the user provides a non-numeric input, then your program will report the mean of the entered temperatures and halt.

Objectives

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

1. write a method,
2. perform a simple computation with numerical values entered by a user, and
3. write java-qualified documentation.

Project Requirements

1. Write a sensor input method that takes as parameters a Scanner and a minimum allowable sensor value (a double). This method returns a double.

   This method must read a double as input from the Scanner. If this value is at or above the minimum value, then this value is returned by the method.

   If the value is below the minimum, then the method must prompt the user for another value. This is repeated until the minimum constraint is met.

   If the value is not a double (indicating that the user is done entering input), then the method must return **Double.NaN**. This is a special double value, Not-A-Number, that we will use here to indicate that there are no other values to be entered.

2. Write a main method that calls your sensor input method repeatedly until NaN is returned. When NaN is returned, your main method reports the mean of the entered values and halts.

Examples

Here are some example inputs and corresponding outputs from a properly executing program (note that user interaction output is not shown). You should carefully consider the cases with which you test your program. In particular, think about the key "boundary cases."

Example 1

Input:

```
10
14
16
```

Output:

```
13.333333333333334
```

Example 2

Input:

```
5
-10
3
5
```

Output:

```
0.75
```

Example 3

Input:

```
5
-5
-20
-15
6
```

Output:

```
2.0
```

Project Documentation

At the top of every java source file for this course, you must include a documentation block at the top of the file that includes the following:

```
/*
   Author: <your name>
   Date: <>
   Project: <number>

   <any  other important information>
*/
```

Note that the notation:

```
<some text>
```

indicates that you should provide some information and not actually write the less-than and greater than symbols.

Method-Level Documentation

Every method must include documentation above the method prototype using standard Javadoc markup. This documentation should be sufficient for another programmer to understand *what* the method does, but not *how* the method performs its task. For example, consider a method that will test whether a value is within a range and whose prototype is declared as follows:

```
public static boolean isInRange(double min, double max, double value)
```

The documentation for this method will be placed above the prototype in your java file and might look like this:

```
/**
   Indicate whether a value is within a range of values

   @param min Minimum value in the range
   @param max Maximum value in the range
   @param value The value being tested
   @return True if value is between min and max.  False if outside
```

```
this range.

*/
```

Note that this example includes all the required documentation elements:

- A short, intuitive description of what the method does (not how it does it),
- A list of the parameter names and a short description of the *meaning* of the parameter, and
- A short description of the return value.

Inline Documentation

Inside of each method, you must also include documentation that describes *how* a method is performing its task. This documentation should be detailed enough for another programmer to understand what you have done and to make modifications to your code. Typically, this documentation is preceded by "//" and occupies a line by itself ahead of the code that is being documented. While each line of code could be documented with its own documentation line, it is typically not necessary or appropriate. Instead, we typically use a single documentation line to capture what a small number of code lines is doing.

In addition, inline documentation should be done at a logical level and should not simply repeat what the line of code says.

Here is an example:

```
public static boolean isInRange(double min, double max, double value)
{
  // Check lower bound
  if(value < min)
     return false;

  // Check upper bound
  if(value > max)
     return false;

  // Within the boundaries
  return true;

}
```

Javadoc

Given the method-level documentation that you provide in your java file, Eclipse will automatically generate web pages that describe your code. Here is the procedure:

1. Select the project.
2. File Menu: Export.
3. Select Java/Java Doc; click Next.
4. Use the default settings. Click Finish.
5. You may be prompted to save one or more of your files. Click OK for these.

This will create a folder in your project folder called *doc*. Within this folder, Eclipse places the set of web pages about your project. In particular, for this project, doc/Project1.html will contain the top-level view of this project. You can use your favorite web browser to view the different methods in your project, as well as a "pretty" version of your method-level documentation.

Hints

Scanner.hasNextDouble() will return true if the next characters to be read by the Scanner constitute a double, and false if the next characters are not part of a double.

Double.isNan(val) will return true if *val* (a double) is NaN and false otherwise.

Project Submission

This project must be submitted no later than 2:00pm on Monday, September 22nd to the project 1 D2L dropbox. The file must be called *Project1.zip* and be generated in the same way as for Project 0.

Code Review

For the office hour sessions surrounding the project deadline, we will put together a "Doodle Poll" through which you can sign up for a 5-minute code review appointment. During this review, we will execute test examples and review the code with you. If you fail to show up for your reserved time, then you will forfeit your appointment and you must attend at a later time that is not already reserved by someone else.

If either the instructor or TA are free during office hours (or another mutually-agreed upon time), then we are happy to do code reviews.

We will only perform reviews on code that has already been submitted to the dropbox on D2L. Please prepare ahead of time for your code review by performing this submission step.

Code reviews must be completed no later than 11:00am on Thursday, September 25th. If you fail to do a code review, then you will receive a score of zero for the project.

Rubric

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

Implementation: 35 points

Program formatting: 15 points

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

(8) 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.

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

(0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

(10) All of the required methods are implemented.

(0) The required methods are not implemented.

Proper Execution: 30 points

Output: 15 points

(15) The program passes all tests.

(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: 35 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.

Method-Level Documentation: 10 points

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

(7) 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: 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.

(0) An incorrect zip file name is used.

References

- Mesonets: http://en.wikipedia.org/wiki/Mesonet
- Oklahoma Mesonet: http://en.wikipedia.org/wiki/Oklahoma_Mesonet