CSCI 135 Programming Exam #1
Fundamentals of Computer Science I
Fall 2012

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. A time series is a set of data points where the points have a time order. For example, the opening price of a stock every day for a month. You are going to build a library of static methods that can calculate various things about a sequence of floating-point time series data. You will then build a client program that reads time series data from standard input and graphically displays the data.

To get started, create an empty Eclipse project and extract the contents of this zip file into your project directory: http://katie.mtech.edu/classes/csci135/timeseries.zip

Part 1: TimeSeries. This class should provide a library of static methods that computes useful things about a set of time series floating-point data. The time series is stored in a `double` array. You need to add the implementation of all the following methods in the public API:

```
public class TimeSeries
```

```
 double min(double [] d)                  // Find the minimum value in the time series
 double max(double [] d)                  // Find the maximum value in the time series
 double mean(double [] d)                 // Find the mean of the time series
 double smooth(double [] d, int i, int w) // Smoothed mean around the ith point
boolean isPeak(double [] d, int i)        // Check if the ith point is a local maximum
```

We have provided a stub version of `TimeSeries.java` that includes more extensive comments describing what each method should do. In particular, your methods need to be *defensive*; they should avoid crashing if the client asks for silly things (e.g. taking the mean of an empty array). On bad input, the methods that return a `double` should return `Double.NaN` (not a number). On bad input, the method returning a `boolean` should return `false`.

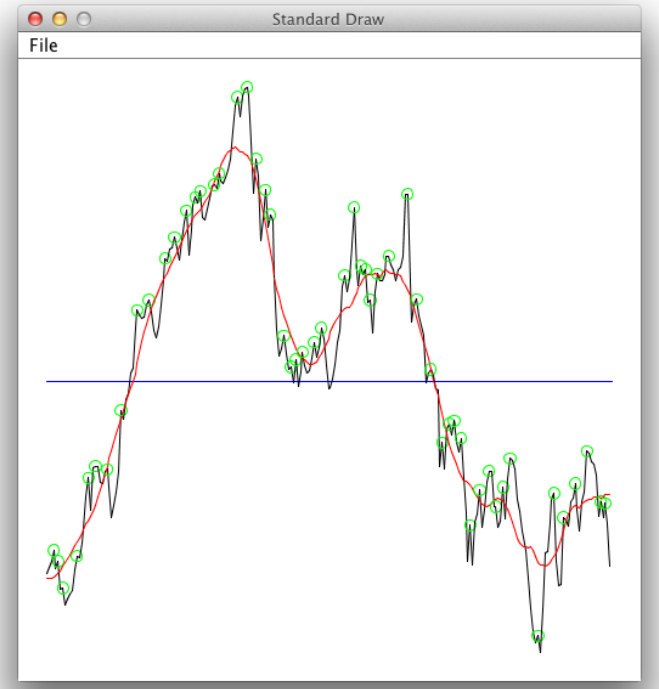
We have provided a test `main()` method in the stub version of `TimeSeries.java`. Here is our test output:

```
% java TimeSeries
min  = 2.0
max  = 5.0
mean = 3.1166666666666667

smooth(1,1) = 2.9
smooth(3,1) = 3.0666666666666664
smooth(4,2) = 3.55

d[0] = 2.0, false
d[1] = 2.5, false
d[2] = 4.2, true
d[3] = 2.0, false
d[4] = 3.0, false
d[5] = 5.0, false
```

Part 2: GraphSeries. You need to create a program `GraphSeries.java`. Its goal is to read in a time series from standard input and produce a graph like this:



The program reads the time series data from standard input. The input file starts with an integer specifying the number of data points followed by the floating-point data values. Here is the file `small6.txt`:

```
6
2.0
2.5
4.2
2.0
3.0
5.0
```

The program should meet the following requirements:

- Takes a single optional integer command-line argument specifying a non-negative smoothing width. If no argument is provided, a default width of 2 should be used.
- Print to the console the number of data points, the range of values, and the mean value. The minimum, maximum and mean values should be rounded to two decimal places. Example output:

  ```
  N=6 [2.00, 5.00] mean=3.12
  ```
- The points in the time series data should be drawn as connected black lines.
- The smoothed average should appear as connected red lines.
- The overall mean of the data should appear as a horizontal blue line located at the mean value.
- The points which are local maximums should be shown with a small (but visible) green circle. For full credit, the visible size of the green circles should be the same regardless of which particular data series is being graphed.
- Aside from `StdDraw's` small white buffer region along the canvas edges, the graph should fill the vertical and horizontal space of the window.

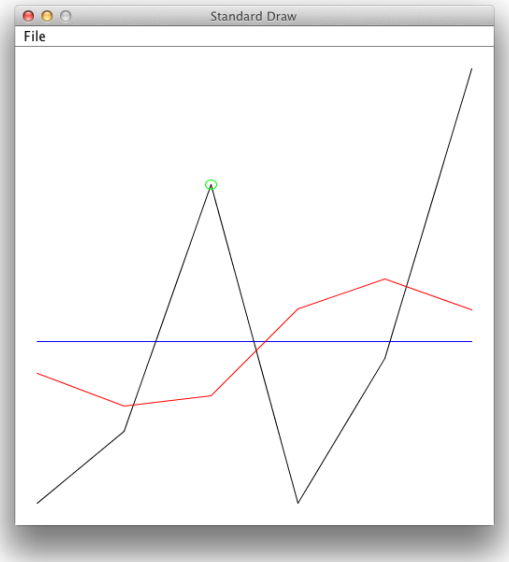
You should obviously use the methods in your `TimeSeries` library to help graph the data. Depending on the details of your implementation, you will also need a subset of the following `StdDraw` methods:

```
public class StdDraw
```

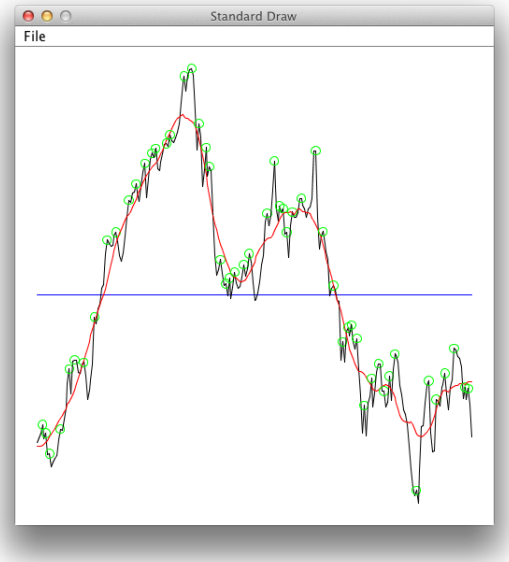
```
void setXscale(double min, double max)
void setYscale(double min, double max)
void setPenColor(Color color)
void line(double x0, double y0, double x1, double y1)
void circle(double x, double y, double r)
void ellipse(double x, double y, double semiMajorAxis, double semiMinorAxis)
```

Here are some more example runs on our provided data files:

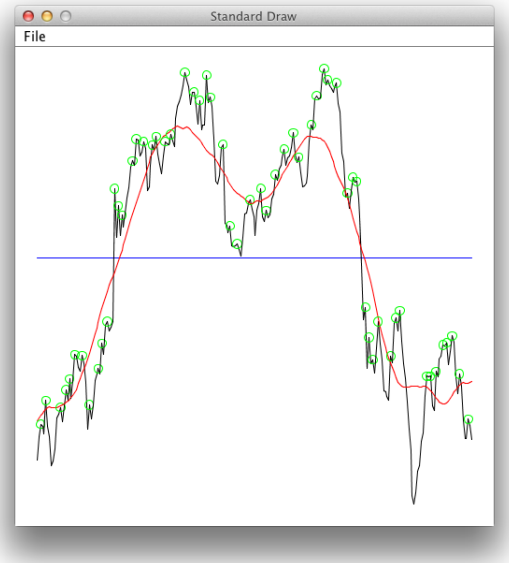
% java GraphSeries < small6.txt
N=6 [2.00, 5.00] mean=3.12



% java GraphSeries 10 < goog.txt
N=243 [438.34, 627.84] mean=529.08



% java GraphSeries 20 < msft.txt
N=242 [23.09, 31.32] mean=27.74



% java GraphSeries 50 < aapl.txt
N=241 [164.50, 277.75] mean=221.34