CSE 11 Winter 2016 **START EARLY!**
Program Assignment #4 (100 points)
Due: 27 Feb 2016 at 1159pm (2359, Pacific Standard Time)

PROGRAM #4 : Histogram

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

The program focuses on using java Swing and Exception Handling. You will be starting with an existing program, first understanding how it works, and then design a new program to achieve different results (using as much of the existing code as makes sense). The goal is for you to become more familiar with a non-trivial Swing program, then modify it to achieve different results.

The supplied class was taken from an online post about how to create a logical grid so that each x,y coordinate in the grid is really a rectangular box. The program can only make a grid of a specific size, using hard-coded constants, and other items. The first thing you need to do is *understand* how this program works. You need to figure out how the classes (all coded as helper classes) work with each other, how the method `paintComponent()` is utilized, how the grid is created, and more.

BlockGrid – As supplied for your homework

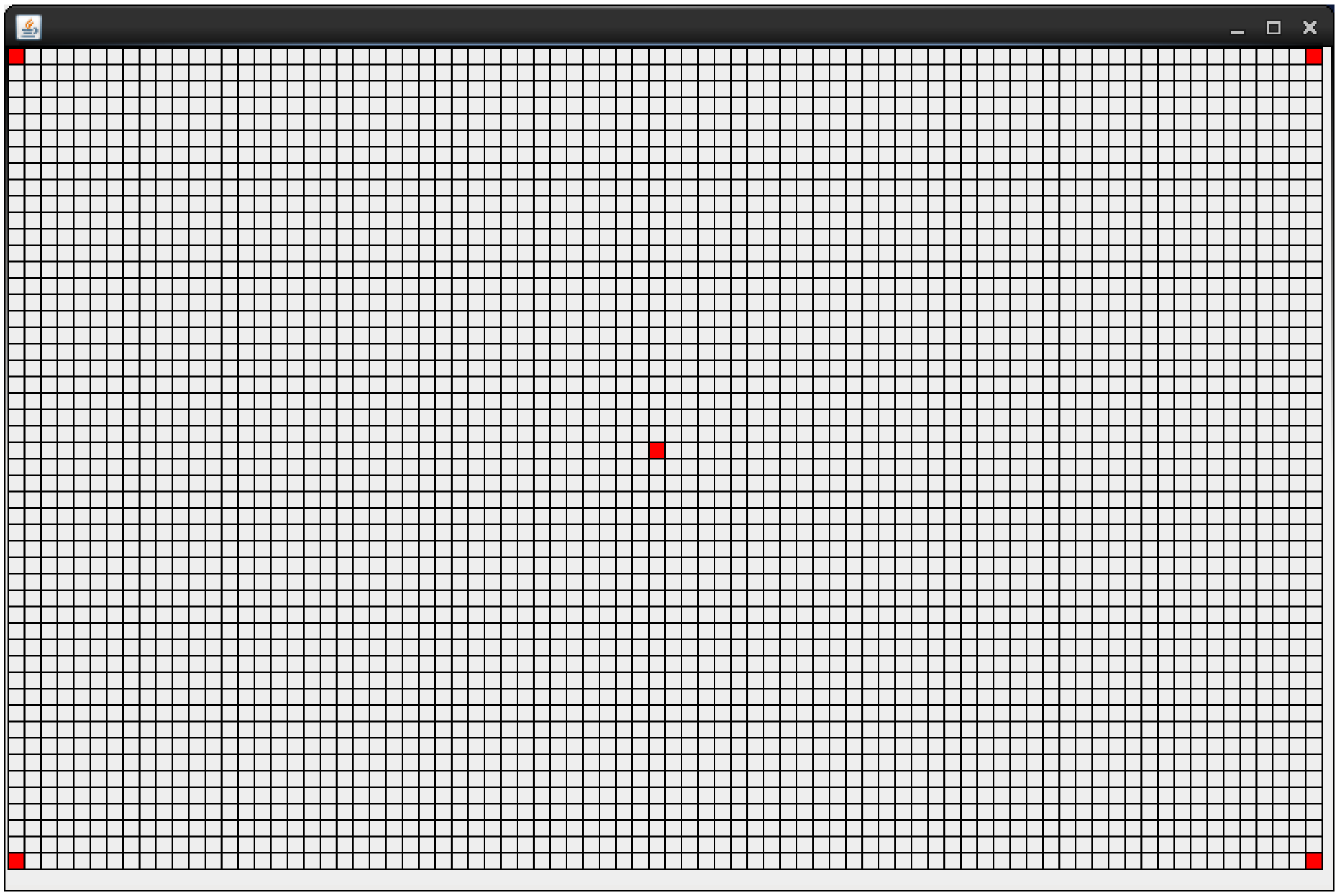

Figure 1 - BlockGrid. As Supplied for the homework

Figure 1 shows BlockGrid. In a nutshell, what you are looking at

- A grid of 10x10 pixel cells
- The grid is 800 pixels wide and 500 pixels tall
- The grid is logically 80 x 50
- Each corner of the grid is filled with a red-colored cell
- The "center" of the grid is also a box colored red.

If you look at the implementation, you will notice that the grid and cells are inside of java Swing JPanel object.

Your Program: Histogram

Basic Idea:
The basic idea of your actual program is to read a file of non-negative integers and treat them as elements of an array. You are then to plot each value as the height of a bar in histogram. For example, given an input file called `testinput` that has seven integers

```
    5 10 15 20 5 10 15
```

It should create the following graphical output when the command is entered

```
    % java Histogram 300 200 testinput
```

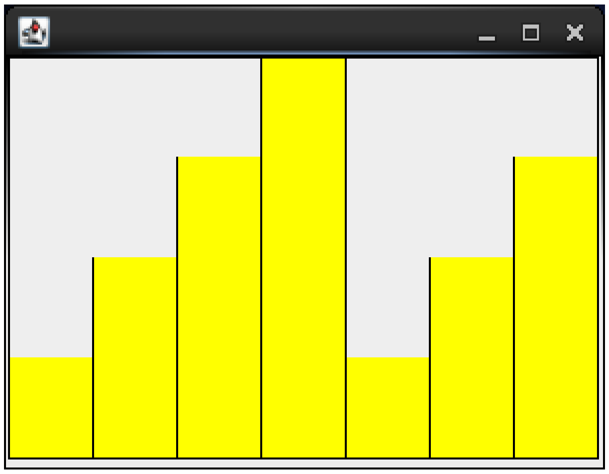

Figure 2 - Example output with 7 integers. The chart is approximately 300x200 pixels (not including the window borders and decoration)

The first two numbers of the command are the width and height of the histogram (in pixels). The third argument is the name of the input file that contains only non-negative integers.

- Each bar in the histogram has identical width
- Each bar is colored yellow
- The largest value is full height
- All other values are of proportional height with respect to the largest value. For

example, the values 5 are ¼ the height of the largest value (20)
- There are black lines between the bars and black framed rectangle around the histogram

Your final program will need to gracefully handle a variety of possible error conditions. The end of this assignment has a complete (and fairly extensive) list of errors that your final program should handle

Suggested Development Plan

This is really your first complex program and attempting to solve all of it at once is a mistake. First, copy BlockGrid.java to Histogram.java, rename the public class and recompile. Make sure that works properly.

Step I – Generalize your grid.

Modification 1: Make Histogram generate any M x N logical grid, where the cells are K x K pixels. This modified version then fills in the corners and "center" pixel with red squares. The actual grid will be (M*K) x (N*K) pixels. Since the final Histogram program must process up to three command-line arguments, use this to your advantage. E.g., this version of Histogram might take command-line arguments as follows

```
$ java Histogram <width> <height> <pixels>
```

The parameters are as follows
- width = width of grid in pixels
- height = height of grid in pixels
- pixels = size in pixels of each grid square

At this stage, it reproduces the five squares in BlockGrid but is now variable size because of the new parameters.

For example, `java Histogram 600 400 10` should create a logical 40x60 (rows X columns) grid and paint five red squares at logical locations (0,0), (39,0), (0,59),(39,59), and (20,30).

Then test with different parameters (for example, 10 evenly divided both width and height), for example 600 400 11. Decide for yourself if this should result in 36 x 54, or 37 x 55 or something else. This is an interim stage of development so either answer is "OK"– but it will require you to much better understand the existing code and how it works.

Step 2 - Make Histograms instead of Grids.
Now that you have made Histogram a bit more general, you probably want to now have it draw histograms instead of a grid. First, look at the `paintComponent` method and how it decides exactly what to draw. Notice that is uses the (x,y) components of a Point (java.awt.Point) Object as the (X,Y) coordinate in the logical grid.

What if you thought about these instead of as representing (X,height)? If you also removed the horizontal lines, you would be much closer to creating histograms. Now, suppose you modified your program to draw a test Histogram so that a test output could look like the following diagrams. (Note these are only making 3 calls to the `fillCell` method instead of 5.

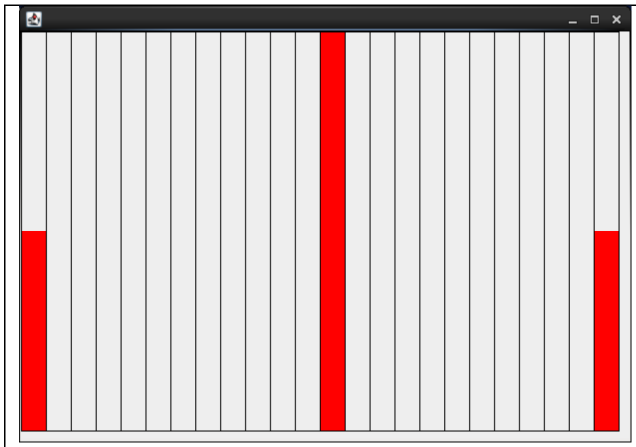

Figure 3 - Histogram 600 400 25

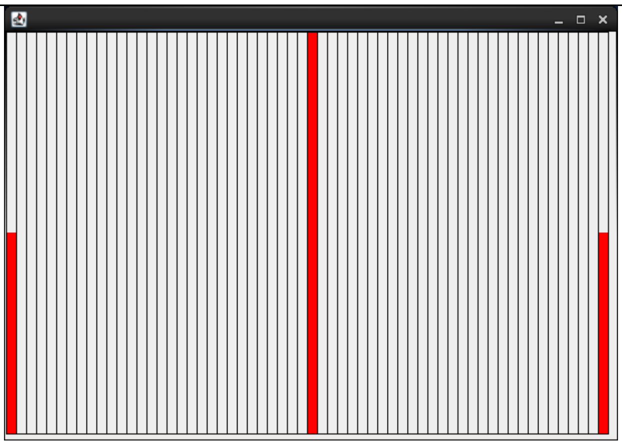

Figure 4 Histogram 600 400 10

As the next logical step of this stage of development, use a Random number generator to create random heights for each Histogram bar. For example, output *might* look like the following at this stage. (The Random numbers are in place of reading a file)

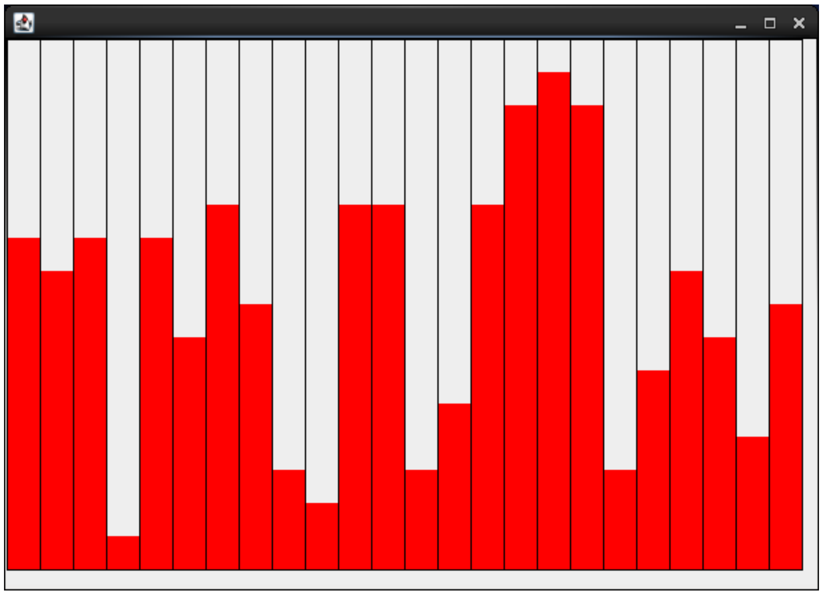

Figure 5 Histogram modified to display Random heights at each logical X coordinate (600 400 25 were used as the arguments for this example)

At this point, you are almost drawing histograms correctly. Your program just needs to adjust the vertical black lines to be the "correct" height.

Step 3 – Read the Input File

The next step would be to Read a file of non-negative integers and use those values as the heights of each histogram bar.

Modify how your Histogram program processes command-line arguments to be as follows

```
          $ java Histogram <width> <height> <filename>
```

Your program should read all the values in the filename, and calculate how wide each bar should be in pixels (based on the number of values). For example, if your window is 600x400 and your files has 22 values, then each bar should be about (600/22 = 27) pixels wide. You will also need to compute the maximum value of all the integers read in. When you graph the maximum value, its height should be the height typed in on the command line (400 in this example). All other values should have their heights in proportion (see the Figure 2). Also change the color of the bars to yellow.

At this point, you should be able to reproduce Figure 2 with your own program.

Your program is now in the state where ***the common case*** works. Common case means that there are no input errors, no format errors in the input file, neither too few nor too many values in the file or other possible issues

Step 4 – Handling Errors and Other Requirements

Below is a list of requirements. If a "Reason Statement" is given, then that statement should go after "Reason:" in the usage statement. You may add other information to the reason (for example, including the error string (using `getMessage()` of the `Exception` class). If you add more information, please keep it on the same line.

I. Errors that must be handled by your program

Requirement 1. When an error is detected, your program should output to *stderr* (the standard error stream) a two-line statement of the following form:

```
Usage: Histogram [ width height] filename
Reason: a reason for the usage statement
```

Requirement 2. Your program should take either one (1) or three (3) arguments. If one argument is given, the single argument is interpreted as a file name. If three arguments are given, the first is interpreted as the width of histogram, the second as the height, and the third is the filename.
Reason statement: Improper number of arguments

Requirement 3. Bars must show at least 2 yellow pixels in width (a vertical black line also consumes a pixel). That is, when graphed, at least yellow pixels are visible
Reason statement: Too many bins for pixel width

Requirement 4. All elements in the input file are integers. If anything other than an integer is present in the file, then the usage statement should be given

Reason statement: Non-integer in file

Requirement 5. All integers in the input file are non-negative

Reason Statement: Value is negative

Requirement 6. Program must handle any file system errors. including file does not exist, and incorrect permissions

Reason statement: File system error

Requirement 7. Program must handle empty files

Reason statement: No values to display

Requirement 8. Program must handle obviously incorrect grids (e.g. with non-positive width or height)

Reason statement: Invalid Dimensions

Requirement 9. Program must handle when width or height given on command line is not an integer

Reason statement: Invalid Integer

Requirement 10. If more than one error is possible given specific inputs (eg. invalid grid and invalid data), only ONE of the errors should be given (the first one your program finds)

II. Functional Requirements of your Program

Requirement 1. The JFrame that holds the bars can be no larger the (width + 2) x (height + 2) pixels. This allows you to treat the width x height as the area for the histogram. The extra pixels allows you to frame the histogram in a black-colored rectangle.

Requirement 2. The Histogram must be framed with a black rectangle drawn with lines that are one (1) pixel wide

Requirement 3. A single line, colored black, this is one pixel wide, separates each bar. The top of each bar does not have to be framed in black. Two black lines next to each other (each being 1 pixel wide to create a visual that is a 2-pixel-wide line is incorrect)

Requirement 4. The bars are colored yellow

Requirement 5. The largest value in the input file is full height. All other graphed values have pixel heights that are proportional to this maximum value

Requirement 6. If width and height are not given on the command line, you are to use the following defaults

- width = 600
- height = 400

Requirement 7. The window should be large enough to show the entire Histogram without resizing by the user. You will need to override the definition of `getPreferredSize` in the Grid class so that Swing can properly size your window for you.

Requirement 8. The main program should wait for the user to hit a key before exiting. You may use the same code as given in the example

Requirement 9. All of your classes and methods must be commented using Javadoc-style comments. Your name and email should go in as Javadoc style at the top of your program. See your previous assignments for examples.

Requirement 10. Your program should never end with a stack trace. Any errors should be gracefully handled with usage statements.

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR4" program that will turn in the file
Histogram.java

No other files will be turned in and they **must be named exactly as above.** BundlePR4 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11wa/public/bin/bundlePR4
```

The output of the turnin should be similar to what you have seen in your previous programming assignments

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

My Grid doesn't exactly fit in the JFrame, what should I do? When your Grid is added to the Window, the Window should be packed, using the pack() method defined in AWT. Your Grid class will need to override the definition of `getPreferredSize()`

What public methods can I add? Anything you want (methods and constructors). Don't use public class or instance variables.

Why are we doing this program? Roughly, this is preparation for your final program.

If the user just gives bad input, do I just print usage and reason statement? Yes. Make sure it is output to the standard error output.

Do I have to check for all kinds of crazy inputs? Programming with Exceptions can help make this a fairly easy task.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. .

I don't understand paintGraphics or how the original program really works, can you explain? `paintComponent()` is how AWT/Swing redraw a graphics screen. AWT decides when to ask your component to repaint itself (via paintComponent()). It might decide to this if you move the window on your screen, if it becomes uncovered via mouse click or any number of (events) reasons. paintComponent() can be called at anytime. In the sample code, the method repaint() is called. This puts a repaint() request onto Event Dispatch Queue (controlled by AWT) and then returns. When the Dispatch thread gets around to handling the repaint request, your paintComponent method will be invoked.

What does SwingUtiltilities.invokeLater() do? When you give invokeLater an object reference (say its called myGrObj), a request is placed onto the Event Dispatch Thread to invoke the run() method of myGrObj (only objects that implement Runnable are valid objects for invokeLater). At some point in time "later", the run() method of myGrObj will be called.

What IS the Event Dispatch Thread? Graphics programs have all kinds of events (mouse movement, keyboard, windows closing, resizing, etc). The Dispatch thread is in charge of informing Objects that a particular event has actually occurred. When an event occurs (say a mouse press), the event is put onto the event dispatch queue (We call this enqueing a request to the dispatch thread). Think of the queue as an inbox of work to-be-done. The Event Dispatch Thread takes an event off its dispatch queue (out of its inbox) and tells every Object that has registered interest in that particular event that the event (and its particulars) has occurred. Say it is a MousePress event, and three Objects have interest in the MousePress event, each of the three Object's appropriate handler is called. They are called one-at-a-time. When all of the handlers of all the Objects have completed for a particular event, the dispatch thread goes on to the next event in its dispatch queue. Note. This program is not defining any events to handle. (you will in Program 5)

How should I exit the program? Prompt in main() for the user to hit a key. Just like the sample program.

I have a question on input files. Are values separated by new lines or spaces or something else? They are separated by white space. If you are properly using Scanner, then programs with multiple lines of input or all values on a single line should work.