CS6312
Lab 11

The Thermometer

Objectives:

- Use standard JavaFX Graphics classes to draw a simple image

Overview:

At this point in the course, we've pretty much hit the bulk of the conceptual material. Of course there's more stuff to learn, but I like to take the end of the second programming course to allow you to apply what you've already learned in a new setting. Specifically, this week we'll turn to drawing simple graphics. In many senses this won't really be a standard 'Graphical User Interface (GUI)' application because it won't accept any user input, but will instead just draw a simple image in a window.

In order to get this drawing accomplished, you'll need to spend some time with JavaFX and the Java API learning about new classes that you probably haven't encountered before. The good news is that the textbook will tell you which classes you'll need, but it will be up to you to read about them and figure out how to get them to do what you want them to.

Additionally, because you will be drawing an image on the screen, you might find it useful to print a sheet of graph paper and get your handheld calculator ready so that you can do a little math and plotting of your image. My suggestion – whether you are drawing an image or writing a console application – is to plan your work out before you sit down to the computer. Then once you have your plan in hand, work through the exercise one small piece at a time. This exercise isn't terrible hard conceptually, but you may find yourself a touch frustrated at times, simply because of a combination of the way graphics are laid out and your plans (or lack thereof). I found myself a little frustrated a few times and had to walk away from things to take a little break, before I could come back and get it running properly.

Also, I want to try to make things easier for you and not require you to key in data, so I've set this one up to read in data from a file. This way you can just click Run and have your image drawn. Additionally, it will give the TA's an easier way to make sure that they are testing your programs with appropriate values. As a consequence, you're also going to be asked to write and test a small Model class.

What to do:

1. Start a new Eclipse Java project with the name `Lab11`. This project should contain the following packages:

   ```
   edu.westga.cs6312.graphics.controller
   edu.westga.cs6312.graphics.model
   edu.westga.cs6312.graphics.view
   ```

   Don't forget to include appropriately named testing packages too.

2. The `model` package should define:
   A class named `Thermometer`. This class will be used to model the thermometer to be drawn. This class will define:

   Instance Variables (only):
   - The minimum temperature (int)
   - The maximum temperature (int)
   - The current temperature (int)

   Methods:
   - A 3-parameter constructor that accepts the values to be used to initialize the instance variables
   - 3 different getter methods, one for each of the instance variables

3. Be sure that you have appropriate JUnit test classes with appropriate tests for the `Thermometer` class.

4. The `controller` package should have a class named `GraphicsDriver`. This class will have a `main` method to launch the application.

   It will also define a `start` method and the '5 things' needed to do the things necessary to get an appropriate GUI created and displayed. Be sure that your `start` method creates an instance of `DrawingGUI` and then uses it to call `getThermometerPane` to be used in the `Scene`.

5. The `view` package should have a class named `DrawingGUI`. This class will define:

   Instance Variables (only):
   - An object of type `Thermometer`
   - An object of type `ThermometerPane`

   Methods (only):
   - A constructor that gets the data read in from the temperature.txt file and then creates the `Thermometer`
   - A getter method for the `ThermometerPane`
   - A private method named `readData` that will open a file named 'temperature.txt' (this file name will be hard-coded into your program) that resides in the default folder for an Eclipse Project (just what we've been doing all along). This file will contain exactly three pieces of data, one per line as follows:

     Minimum temperature
     Current temperature
     Maximum temperature

     Be sure to appropriately handle any situations where there are problems with the file and/or with invalid data inside the file.

   - A private method named `createThermometerPane` that will create and save the `ThermometerPane` corresponding to the data in the file.

6. The view package will also have a class named `ThermometerPane`, which will be a subclass of `Pane` and will contain the code to actually draw the thermometer. This class will define:

Instance Variables (only):

- An object of type `Thermometer`

Methods:

- A 1-parameter constructor that accepts a `Thermometer` object to be stored as the instance variable. This object contains the data that we will be drawing. You will use this constructor to set up the appropriate size of the Pane and then have the thermometer drawn.

  As another hint, it is recommended that you continue to practice incremental development, as well as writing short, cohesive methods that accomplish one task. This thermometer is made of a bunch of different pieces (described below). You will find it helpful to do just one piece at a time until the whole thing is built.

Drawing Specifications:

- The thermometer's "bulb" (the bottom of the thermometer where all the Mercury is located)
  - The center of the bulb should be in the middle of the Pane's width and located at 2/3 of the Pane's height
  - To keep things as simple as possible, just make the bulb's radius 50 pixels (always)

I found that in order to get the next or 'rectangle' of the thermometer to draw correctly, I needed to have one in the background (all red) and one in the foreground (with the black covering up the red proportionally so that the appropriate amount of Mercury (red) shows through) as follows:

- The thermometer's "background fill" (red outline in image, though the top almost looks purple because the "black fill rectangle" is drawn over it)
  - The x value should be such that this rectangle is centered in the Pane
  - To make it visually more appealing, make sure that the top of the rectangle is 1/10 the height of the Pane (so that there is always a proportional amount of space above the top of the thermometer)
  - We want to be sure that the background goes all the way into the bulb, therefore we'll make this rectangle's height such that it extends all the way down to the center of the bulb

- The thermometer's "black fill rectangle" (black outline in image)
  - The x, y, and width values will be the same as the background rectangle.
  - The height of the rectangle will be proportional, according to the current temperature value. For example, if the data has a low of 0, a high of 100, and a current temperature of 50, then this rectangle will be half the height of the background rectangle so that only half of the column is filled with Mercury

- The temperature lines
  - The maximum temperature line should be at the top of the "background fill" rectangle
  - The minimum temperature line should be at the very top of the bulb. This is drawn in blue in the image below. You can feel free to make yours blue or black in your submission
  - The current temperature line should be at a y location that is an appropriate ratio between the maximum and minimum temperature lines. For example, if the maximum is 100, the minimum is 0, and the current is 75, then the line should be 3/4 of the way between 0 and 100.

- The thermometer's temperature labels
  - The text should be centered vertically in reference to its corresponding line
  - The text values should be displayed an appropriate distance away from the rectangle
  - The current temperature should be displayed to the left of the rectangle
  - The maximum and minimum temperatures should be displayed to the right of the rectangle.

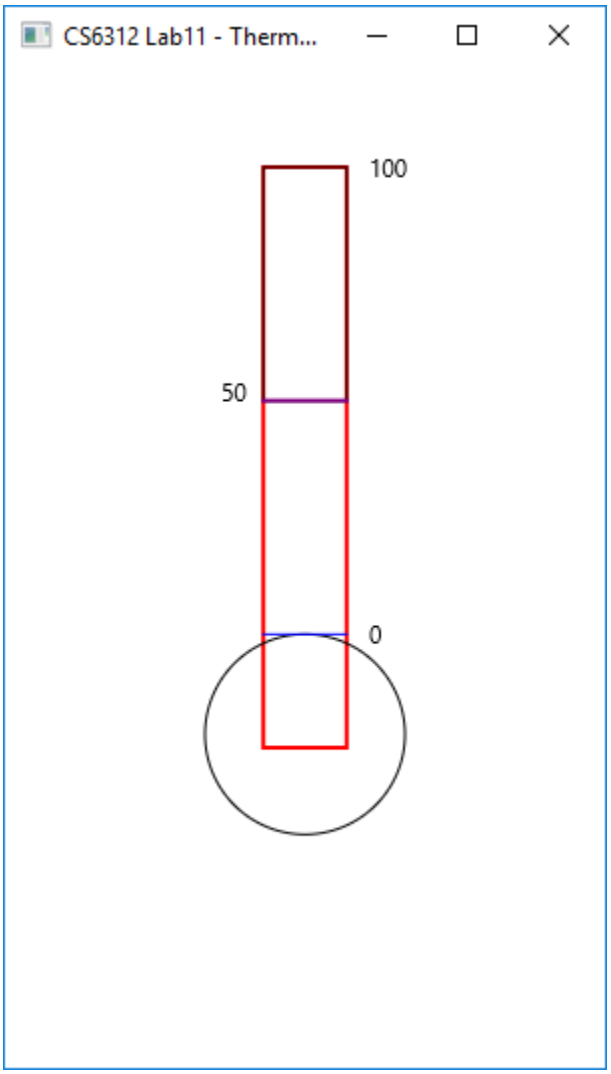
Hint:

- Be sure to use appropriate shapes the in the `javafx.scene.shape` package.
- Make use of `bind()` so that the appropriate property of each shape is properly bound to the `Pane`'s height and/or width

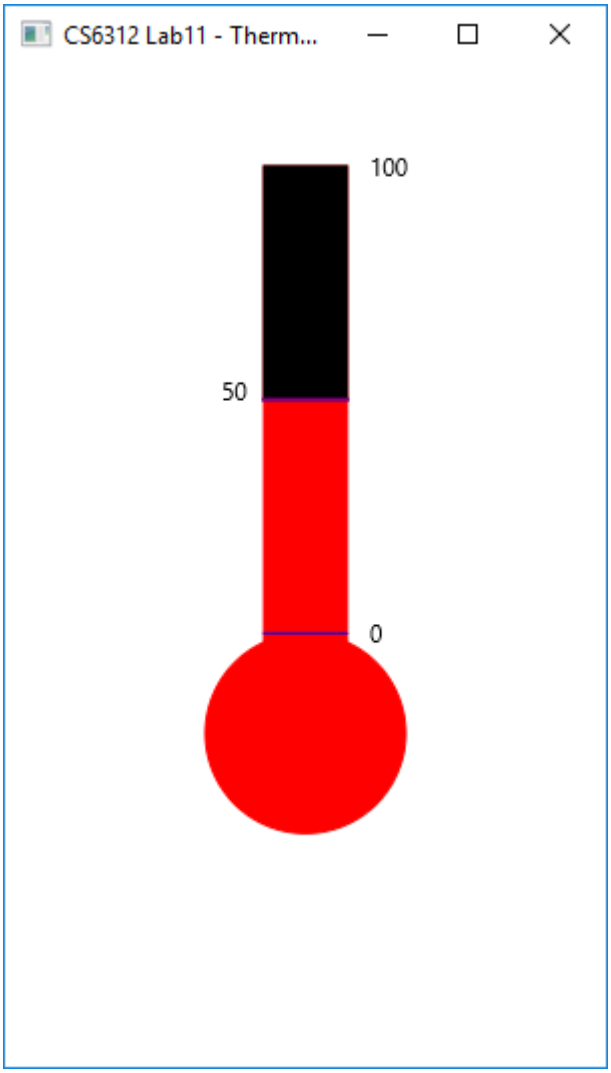
Reminders of things we teach here:

- Planning: You might consider drawing things out with graph paper and use hard-coded values to begin with. Then worry about binding them so that they are resized appropriately.
  - The math on this is easy, but there are details to keep up with
  - Far and away the one that took me the longest was doing the 'current temperature' line and label. Again, it's just math, but writing things out in Java took some time to get it correct (at least it did for me)
- Please start early. Doing this will provide the opportunity for you to get really frustrated, walk away, and still have time to come back and get things fixed so that it works the way you want.
- Please type out the code examples in the text before doing anything else. We said back when we started that in an ideal world you would read the chapter 'like a novel' first. Then go back and read it again, but this time type in the code snippets from the text to see how they work. Then take the Quiz. Then start on the lab. I can promise you that typing in (not copy/paste) the code from the text is going to take you a long way this week.

Sample Images:



Outline Image
(for your drawing reference -
Do NOT draw this)



Application Image
(this is what yours should look like)

I have also included a video on Moodle that shows what this application looks like when I resize the window. Sometimes seeing how the layout is done when window is resized will help to give you a better idea of (a) how things work and (b) if your drawing is correct. Remember, we don't want to use hard-coded, static values here for the drawing. Instead the drawing should resize with the window.

Grading:

A quick note here about grading – this exercise is intended to give you experience with the graphical objects and give you practice reading the API. My experience has been that it's pretty easy to spend LOTS of time trying to get images to look 'perfect'. That is not what this exercise is about. Therefore please know that when we're grading, if your image is off by a pixel or two, that's completely fine. Now if it's off by 10 pixels or if the window is resized and the 'off by 1 pixel' increases over and over to the point where the shapes aren't touching each other anymore – then that's a problem. As always, please use your best judgment. We're not trying to make things difficult for you, but we do want you to produce an application that makes an appropriate image.

Submitting:

When you are finished, make one final commit to your repository, be sure that all of the class files are saved, close Eclipse, and Zip the folder holding your Eclipse project using 7-Zip. Give your file the name 6312*YourLastName*Lab11.zip. Upload the Zip file to the appropriate link in Moodle.

It's probably worth taking a moment to download the file you uploaded to Moodle and double-checking to be sure that it contains everything you are required to have (including the repository).