

I have created a class in /home/bvz/gui/hw/hw2 called template.java that creates a basic JComponent object, adds it to a JFrame, and makes the JFrame visible. You can modify this class to fit your needs for the following problems. I have not over-specified the problems by telling you what the exact sizes of all the objects and windows are. Unless I specify precise sizes for windows or objects, use your best judgement.

1) Rewrite your thermometer application from hw1 so that it incorporates the following changes:

1) the model uses the observer pattern described in class and the model supports the following methods:

- i) void setValue(int degrees)
- ii) int getMinValue()
- iii) int getMaxValue()
- iv) int getValue()

b) use an Area object to create the filled bulb and stem portion of the thermometer in the original thermometer view

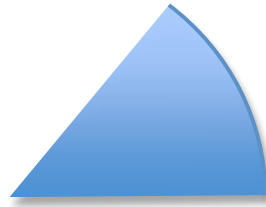
c) add the following two views, each in its own window:

a) A gauge with a chorded half oval and an arrow line that goes from the minimum value (right side of the gauge) to the maximum value (left side of the gauge). It should resemble the gauge shown below. Note that the arrow is filled. Select appropriate values for the width and length of the arrow and use whatever color, line style, and width you prefer. It's okay to use a semi-circle. Should you decide to use an oval, it's easiest to make the angle of the arrowline be proportional to the ratio of the current value to the temperature range.



b) A filled pie arc whose arc is equal to the percentage of the current value as a function of the maximum value. For example, if the minimum temperature is 32 and the maximum temperature is 212, then the arc below represents a temperature of roughly 57 degrees, because 57 degrees is roughly 1/8 of the way between 32 and 212, and a 45 degree

arc is roughly 1/8 of the way between 0 and 360. Fill the arc with the color blue.



In `/home/bvz/gui/hw/hw2` you will find a class named *TemperatureController*. This class creates a dialog box that you can use to control the temperature gauge. Your main method should create an instance of this class after creating instances of the model and the views. The constructor for *TemperatureController* takes two arguments, which are a pointer to the model object and a pointer to the *JFrame* that contains the thermometer gauge. The dialog box will pop up to the right of the window containing the temperature gauge. When you manipulate the slider in this dialog box, *TemperatureController* updates the model. Your model should in turn notify the three views, which should change in response to changes in the temperature.

I have used a rough heuristic to calculate the correct size for the slider, based on the minimum and maximum value of your thermometer. You may need to resize the dialog box if it is too small to accommodate all the labels on the slider.

- 2) Write a program named *DisplayImage* that displays an image and centers a text caption horizontally beneath the image that describes the image.
- 3) Take the same image from problem 2 and use it to fill a rectangle with three tiles of the image. The rectangle should be a 210x50 rectangle. Use the *TexturePaint* class to create the fill style and use the *Image* class's *getScaledInstance* method to get a scaled image of the appropriate size so that three tiles will fit in the rectangle. Name the program *DisplayTile*.
- 4) Design your own composite Java object using Java's *Area* class. Your object must use 3 of the 4 methods described in class (union, intersect, set difference, exclusive-or). Display your object in a program named *Composite*.
- 5) **(CS594 students only; CS460 students can get 20 points of extra credit by doing this problem).** Write a plotting application named *Function.Cube* that plots the function ax^3+bx^2+cx+d for the values of x from -20 to +20. Show the X and Y axes as horizontal and vertical lines with a dashed line. Use a *GeneralPath* object for plotting the function and color it green with a line thickness of 2. Scale the y values so that the function fits completely within

the window. The window should be re-sizeable by the user. The x-axis should be appropriately aligned between the minimum and maximum y-value of the function. For example, if the minimum y-value is -100 and the maximum y-value is 200, then the x-axis should appear 1/3 of the way above the bottom. Do not worry about the degenerate case where $a=b=c=0$ and the function degenerates to a horizontal line. We will not test this case.

Write the application using a separate model and graphical view. The model should support the following methods:

- a. `void setA(double newA)`
- b. `void setB(double newB)`
- c. `void setC(double newC)`
- d. `void setD(double newD)`
- e. `double getA()`
- f. `double getB()`
- g. `double getC()`
- h. `double getD()`

In `/home/bvz/gui/hw/hw2` you can find a class named *FunctionController.java* that creates a dialog box that allows you to manipulate the values of a, b, c, and d. After you create your model and view classes, you will need to create an instance of this class in your main procedure and pass it an instance of your model object and a pointer to the JFrame containing your function drawing. The dialog window will pop up next to the function window.

Here is what a sample screen shot of your application might look like, along with the function controller window:

