Objective  To gain experience using GUI components and listeners.

The Scenario  You will write a program that manipulates text through a JSlider, a couple of JLabels, a pop-up menu (JComboBox), a JTextField, and a JButton. A picture of the screen can be seen below:

A running version of the program is provided on the handouts web page.

As usual, the center of the screen is the canvas. At the “South” end of the screen is a JPanel that holds a JTextField where the user can write some text to be displayed on the canvas, and a subpanel that contains two components used to select a font and remove text from the screen. Below that subpanel is a JSlider used to control the font size, and below that is a label showing the current font size. The main panel should use a GridLayout with four rows and one column as its layout manager. The subpanel uses a simple FlowLayout. The components included in the subpanel are a JButton that is used to remove the last item written on the screen and a pop-up menu (JComboBox) that allows the user to choose from among several fonts (we used “Courier”, “Sand”, “Zapfino”, and “Times”).

When the user clicks anywhere on the canvas, your program should display the text showing in the JTextField in the selected font and font size at the place where the user clicked. When the
user adjusts the JSlider or selects a new font with the JComboBox menu, the last text placed on
the canvas should change its size or font accordingly. Changing the text in the JTextField has no
impact on items displayed on the canvas.

If the JButton is pressed, the last text placed on the canvas should be removed. (You may think of
this as a one-level undo. Pressing the button a second or third time will not remove additional items.)

How to Proceed

First, download a copy of the starter folder from the handouts page and import it into Eclipse.
Before beginning be sure that you have your GUI cheat sheet in front of you (you can find it on the
CS134 web site if you don’t have it with you). As a quick reminder, remember that the basic steps in
displaying components:

1. Construct an instance of the component.
2. Initialize it if necessary (like adding items to a JComboBox).
3. Add the component to a JPanel or to the content pane of the window controller.

To react to events generated by the user interacting with a GUI component, remember these steps:

1. Add the window controller as a listener to the component, calling a method like addActionListener.
2. Declare that the window controller class implements the listener interface for that component,
such as ActionListener.
3. Implement the method that is called when the user interacts with the component, such as
actionPerformed.

Note that the statements

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

appear at the top of our starter Java file. These lines inform Java that your program will need access
to the Java libraries that support GUI components and events (including event listeners).
When you run your program from Eclipse, set the width to 340 and the height to 420.

Here is a suggestion on how to decompose the development of this program into simpler steps.

1. Include code in your begin method to add a JPanel at the “South” end of the program window.
   This JPanel should use a GridLayout having 4 rows and 1 column. Place a JTextField in the
   panel. Make it so that if the user clicks anywhere in the canvas then whatever is showing in the
   JTextField is displayed as a Text item on the canvas. Make sure that successive clicks on the
   canvas insert new Text items on the screen (showing whatever is currently in the JTextField).

   You will want a variable associated with the Text item most recently displayed so that you can
   change its font or font size later. You will associate this name with the current text item in your
   onMouseClick method.

   Since your program does not react to the user typing in the JTextField, it is not necessary to
   associate a listener with this component.

2. Now that you can display text items, start working on adding the components that will let you
   modify them.

   Start with the JButton.

   Add a JButton to the panel so that it contains the words “Remove last text”. In order to al-
   low your WindowController extension to respond when the button is pressed, add the phrase
   “implements ActionListener” to the header of the class. Call the button’s addActionListener
method in your `begin method` to inform the button that your `WindowController` wants to be notified when the button is pressed and add an `actionPerformed` method to your class that performs the appropriate action (removing the text from the canvas) when the button is pressed. Make sure your program behaves reasonably if the user clicks the button before actually putting any text on the canvas.

3. At this point, there are only two GUI components in the panel you have placed in the south quadrant of your display: the `JTextField` and the `JButton`. Since this panel uses a 4x1 `GridLayout`, there is no room left in the second row to add extra components. To make it possible to add extra components in that row of this `GridLayout JPanel`, you should create a new subpanel to hold the `JButton` you already created and the `JComboBox` component you will create in the next step. Then, add the existing `JButton` to the subpanel rather than to the original panel. After this is done, test the program again. It should work the same except the `JButton` will no longer be stretched to fill the entire width of the window.

4. The next step is to create the `JComboBox` menu that will allow you to choose the font in which the `Text` item is displayed.

See the GUI cheat sheet for the constructor and the method used to add choices to the `JComboBox` menu. Then add the `JComboBox` menu to the main panel and tell it that your program will be the listener. The `JComboBox` requires an `ActionListener` just like `JButton` does, so you do not need to add an “implements” clause. You do need to modify your `actionPerformed` method to react to menu selections made by the user. Use the `getSource()` method on the event parameter to find out which GUI component the user interacted with. `getSource()` will return either the `JButton` object or the `JComboBox` object. Depending on which value `getSource()` returns, you should either remove the last text from the canvas or change its font.

To change the font, you can call the method

```
setFont(String fontName)
```

of the `Text` class to change the font used by an existing `Text` object.

5. The last control we want you to add is the `JSlider` to control the font size. As shown on the first page of the handout, we also want you to place a `JLabel` component under the `JSlider` to display the current font size.

Start by adding a `JSlider` to your main panel just as you added the `JTextField` and subpanel. Place it after the subpanel. Look at the GUI cheat sheet to determine what type of listener a `JSlider` needs. (When you have a class that implements several interfaces, you simply separate the interfaces with commas. Do NOT include the keyword `implements` more than once in the class header!) Define the appropriate listener method so that when the bubble in the scrollbar is moved, the size of the text changes. The possible font sizes should range from 10 to 48.

6. The `JLabel` beneath the `JSlider` should always display the current value represented by the `JSlider`. As shown on the first page of the handout, we want you to display this value preceded by the string “Current Font Size: ”. This can be done as usual using the concatenation operator (+) in constructing the string to be inserted as the parameter of the `setText` method of the `JLabel`.

7. Finally, make sure that when the user clicks on the canvas the new `Text` item drawn has all the characteristics selected for font and font size. Avoid duplicating the code used to set the font and font size by creating a private method that you can call from several places.

Be careful that your program does not crash if you manipulate the controls after removing the last text item from the canvas or before adding the first text item to the canvas.
Extra Credit

If you are interested in doing some extra credit work on this project, here is a suggestion. Add the capability of changing the color of your text. There are two ways of doing this, one fairly simple, while the other is more complicated. We'll explain the simple one first.

1. Add a JComboBox menu with the names of colors (e.g., black, red, blue, green, yellow). Make “this” a listener for that choice item. Update your actionPerformed method to change the color of the text when the user uses the color menu.

2. A more flexible way to control the text color would be to use a set of three sliders. You could do this by adding three more JSliders to your display.

Submitting Your Work

When your work is complete, you should deposit the folder containing your java file into the appropriate dropoff folder. Make sure the folder name is your last name followed “Lab6”. Also make sure that your class includes a comment containing your name.

Also, before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.

NOTE: For this lab, Monday labs are due Tuesday at 11 PM and Tuesday labs are due Wednesday at 11 PM, although it is likely that you will complete this during your lab period.

Grading Guidelines

<table>
<thead>
<tr>
<th>Value</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Style (5 pts total)</strong></td>
</tr>
<tr>
<td>1 pts.</td>
<td>Descriptive comments</td>
</tr>
<tr>
<td>1 pts.</td>
<td>Good names and formatting</td>
</tr>
<tr>
<td>1 pts.</td>
<td>Good use of constants</td>
</tr>
<tr>
<td>1 pts.</td>
<td>Good use of private and local variables</td>
</tr>
<tr>
<td>1 pts.</td>
<td>Good use of private methods</td>
</tr>
<tr>
<td></td>
<td><strong>Correctness (5 pts total)</strong></td>
</tr>
<tr>
<td>1 pt.</td>
<td>Displaying appropriate text at click point</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Changing the font</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Removing the last text</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Changing the font size (and showing its size in the label)</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Avoiding null pointer errors</td>
</tr>
</tbody>
</table>