**Assignment 7 Turn in Sheet Name: Tyler Quayle**

Lab Questions (**Total 50 Pts. + 22 Extra**)

Big Java, Late Objects / Java for Everyone, 2e

**Chapter Number: 11 Advanced User Interfaces (each (sub) question 2 points)**

1.1) If you execute the code listed below, you should see a frame that contains a single red panel. Modify the code so that it creates five additional panels of the same size, but with different colors: white, blue, green, yellow, and orange. Since we have not specified a layout manager for the main panel, the default FlowLayout manager is used to control the placement of panels. What happens if you resize the frame to be narrower and longer?

import javax.swing.JFrame;

import javax.swing.JPanel;

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.Color;

import java.awt.Dimension;

public class PanelRunner

{

/\*\*

Creates and displays the application frame.

\*/

public static void main(String[] args)

{

JFrame frame = new JFrame("Color Panels");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(1200, 1000);

JPanel panel = new JPanel();

panel.setPreferredSize(new Dimension(1200, 1000));

JPanel subpanel1 = new JPanel();

subpanel1.setBackground(Color.RED);

subpanel1.setPreferredSize(new Dimension(200, 1000));

panel.add(subpanel1);

frame.add(panel);

frame.setVisible(true);

}

}

1.2) Create PanelRunner2 by modifying PanelRunner1. Set the preferred panel sizes to 200 by 200 for each of the subpanels. Use the BorderLayout manager to control panel placement so that the subpanels are configured as below:

Red

White Blue Green Yellow

Orange

You will need to create another subpanel to hold the blue and green panels. Try running the program and making the frame narrower. Does the green panel disappear? Why? Fix the problem by using a GridLayout manager for the center panel.

1.3) Create PanelRunner3 by modifying PanelRunner2. Set the preferred panel sizes to 200 by 200 for each of the subpanels. Use the GridLayout manager to control panel placement so that the six subpanels are configured into three rows and two columns. Add the subpanels so that the final arrangement appears as below:

Red White

Blue Green

Yellow Orange

What effect does narrowing or enlarging the frame have on the subpanels?

**Enlarging the frame will increase size of the panel to fill the allotted space. Shrinking the frame below (PANELS\_IN\_ROW \* SIZE) will make the panels on the right disappear if the frame drops below the preferred size. (IE 2 panels of 200 wide cannot exist in a 200 pixel frame. They do not ‘share’)**

2) Create a class called ComboPanel that extends JPanel. The class constructor should add three JComboBox objects to the panel and initialize each of them with Integer objects with values that range from 0 to 255. Each JComboBox will allow the user to select an integer for the three components (red, green, and blue) that make up a Color object. For example, invoking new Color(255, 215, 0) creates a yellowish color object.

Create listener objects for each JComboBox that set the background color of the panel when the user selects an integer. Use the current values supplied by the JComboBox objects to determine the background color.

Use the following code to test your class.

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JComboBox;

public class ComboPanelRunner

{

/\*\* Creates and displays the application frame. \*/ public static void main(String[] args)

{

JFrame frame = new JFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(400, 500);

ComboPanel cp = new ComboPanel();

frame.add(cp);

frame.setVisible(true);

}

}

3) Create a class called ColorFrame that extends JFrame. Create a “Colors” menu with three menu items: Red, Green, and Blue. The ColorFrame class also contains a JPanel object that changes to the appropriate color when the user selects a menu item. Use the following code as a demo class.

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JComboBox;

public class ColorPanelRunner

{

/\*\* Creates and displays the application frame. \*/ public static void main(String[] args)

{

ColorFrame frame = new ColorFrame();

frame.setSize(400, 400);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

4) Use the Java API to research the Swing class called JList. Create a simple demo application that builds and displays a JList in a frame.

5.1) Problems 5.1- 5.4 in this lab are based on the HelloViewer program:

/\*\* File HelloViewer.java \*/

import javax.swing.JFrame;

public class HelloViewer

{

public static void main(String[] args)

{

JFrame frame = new HelloFrame();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setTitle("HelloViewer");

frame.setVisible(true);

}

}

----------------------------

/\*\* File HelloFrame.java \*/

import javax.swing.JFrame;

import javax.swing.JLabel;

import java.awt.BorderLayout;

import java.awt.Font;

public class HelloFrame extends JFrame

{

private String message;

private JLabel label;

private static int FRAME\_WIDTH = 300;

private static int FRAME\_HEIGHT = 300;

private static int DEFAULT\_SIZE = 20;

public HelloFrame()

{

message = "Hello, World!";

label = new JLabel(message);

label.setFont(new Font("Serif", Font.PLAIN, DEFAULT\_SIZE));

add(label, BorderLayout.CENTER);

setSize(FRAME\_WIDTH, FRAME\_HEIGHT);

}

}

In this problem, you need to add two buttons to the bottom of the frame, like this:

Follow these steps:

1. Make two instance variables of the class JButton:

private JButton smallerButton;

private JButton largerButton;

2. Place the following two lines inside the constructor to initialize the variables with JButton objects:

smallerButton = new JButton("Smaller");

largerButton = new JButton("Larger");

3. Place the buttons into a panel:

JPanel southPanel = new JPanel();

southPanel.add(smallerButton); southPanel.add(largerButton);

4. Add the south panel to the frame:

add(southPanel, BorderLayout.SOUTH);

5. In the frame constructor, call the method createSouthPanel:

createSouthPanel();

Make these changes in your program. Run the program to confirm that the buttons appear.

What happens when you click the buttons? Why?

**Nothing happens, as there is no action listeners the buttons are just static items with no purpose as of right now;**

What is the code of your HelloFrame class?

5.2) Next, you need to connect the buttons to actions. For each button, carry out these steps:

1. Create a class that implements the ActionListener interface and override the actionPerformed method.

2. Make an object of that class.

3. Install that object as the action listener of the button.

When the "Larger" button is clicked, we want to increase the font size by 25 percent. When the "Smaller" button is clicked, we want to decrease it. To avoid integer-rounding errors, keep a floating-point instance variable fontSizeFactor that is initialized with 1. Clicking on the buttons multiplies by 1.25 or 0.8 (because 0.8 = 1/ 1.25).

private double fontSizeFactor = 1;

Here is an appropriate action listener for the "Larger" button.

class LargerFontListener implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

fontSizeFactor = 1.25 \* fontSizeFactor;

label.setFont(new Font("Serif", Font.PLAIN,

(int) (DEFAULT\_SIZE \* fontSizeFactor)));

label.repaint();

}

}

This listener must be added to one of the buttons. As you can see, the code for setting up the buttons is getting quite complex. Introduce a new method createSouthPanel that contains all button initialization code. Place the listener class inside the createSouthPanel method, because the actionPerformed method needs to access the label and fontSizeFactor variables.

To connect it to the "Larger" button, you need to create an object of this class and set it as an action listener of the button. Place the following instructions into the frame constructor:

ActionListener largerListener = new LargerFontListener();

largerButton.addActionListener(largerListener);

Add the code, compile the program, and run it. Don't add the code for the "Smaller" button yet.

Click on both buttons several times. What happens?

**Since the action listener for smaller button has yet to be implemented, nothing happens when you push on smaller. However the text will grow by 25% for every time you push the larger button;**

5.3) Repeat the steps above for the "Smaller" button. Make an action listener class and add an instance of that class as an action listener for the smallerButton. Run your program. Both buttons should work.

What is the code for the createSouthPanel method?

5.4) In the preceding program, the buttons were placed on the bottom of the frame, next to each other. Change your program so that they are placed on the right of the frame, one above the other.

What is the code for the HelloFrame class now?

6.1) In this series of lab exercises, we will use a menu instead of a set of buttons to increase and decrease the font size.

As with buttons, you need to solve two unrelated issues:

How to place the menu items.

How to connect actions to the menu items.

In Java, there is a three-level hierarchy for menus:

1. A *menu bar* is attached to a window.

2. The menu bar contains *menus*, rectangular panels with menu strings inside them.

3. Menus contain *submenus* and *menu items.*

Only menu items have actions associated with them.

Here are the instructions to build the menu for this program.

JMenuBar menuBar = new JMenuBar();

setJMenuBar(menuBar);

JMenu fontMenu = new JMenu("Font");

menuBar.add(fontMenu);

JMenuItem largerItem = new JMenuItem("Larger");

fontMenu.add(largerItem);

JMenuItem smallerItem = new JMenuItem("Smaller");

fontMenu.add(smallerItem);

Start again with the HelloViewer program.

Create a method called createMenu, and place these instructions inside this method.

What is your createMenu method?

What happens when you select a menu item?

**Again, no listener attached to the menu buttons, so no actions are linked to the buttons. They are clickable since the inherit that from the JMenu objects. But nothing will happen when you click on them;**

6.2) Next, you need to attach actions to the menu items. This process is *identical* to attaching actions to buttons. Simply reuse the code from the Lab 11.5.2.

Attach the actions to the two menu items and run your program. Check that selecting the menu items respectively increases and decreases the font size.

What is the complete source code for your frame class now?

7.1) In this exercise, you will see how to offer a user a selection among multiple choices. You can place all choices into a *combo box*.



To build such a combo box, you add items to it:

final JComboBox comboBox = new JComboBox();

comboBox.addItem("Small");

comboBox.addItem("Medium");

comboBox.addItem("Large");

comboBox.addItem("Extra Large");

Then place the combo box in the south end of the frame.

add(comboBox, BorderLayout.SOUTH);

Add a createSouthPanel method and paste the lines indicated above inside the method. Try it out. Don't forget to call the createSouthPanel method from within the constructor. Run the program and compare it against the figure.

The program looks different. Why?

**I don’t understand the question, or maybe the question seems too simple. It looks different because we added a combo box. Commonly referred to as a drop down menu;**

7.2) The border layout grows all items to their maximum size. To avoid this, enclose the combo box inside a panel, even though it is a single item.

JPanel southPanel = new JPanel();

southPanel.add(comboBox);

add(southPanel, BorderLayout.SOUTH);

Try it out. Add the combo box to the frame of the HelloViewer program. Compile and run the program. Pull down the item list of the combo box to see the items.

What happens if you select one of the items?

**Nothing happens, no listener attached to these combo box items;**

7.3) To activate the combo box, you need to attach an action listener. The actionPerformed method needs to determine which item the user selected.

class ComboListener implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

String item = (String) comboBox.getSelectedItem();

. . .

}

}

You need to cast the return value of the getSelectedItem method to a String because it is possible to put other objects, such as icons, into a combo box.

Now you simply set the correct font size, depending on the item string.

int messageSize = DEFAULT\_SIZE;

if (item.equals("Small")) { messageSize = SMALL\_SIZE; }

else if (item.equals("Medium")) { messageSize = MEDIUM\_SIZE; }

else if (item.equals("Large")) { messageSize = LARGE\_SIZE; }

else if (item.equals("Extra Large")) { messageSize = EXTRA\_LARGE\_SIZE; }

label.setFont(new Font("Serif", Font.PLAIN, messageSize));

label.repaint();

Define the size constants as follows:

public static final int SMALL\_SIZE = 12;

public static final int MEDIUM\_SIZE = 18;

public static final int LARGE\_SIZE = 24;

public static final int EXTRA\_LARGE\_SIZE = 36;

Add the combo box action listener and try out your program. When you select an item from the combo box, the message should be displayed in the appropriate size.

What is the complete code for your createSouthPanel method?

8.1) Radio buttons are another way to offer the user a selection among multiple alternatives:



These buttons are called radio buttons because they work like the channel buttons on a radio. When you select a radio button, the previously selected radio button is turned off.

However, radio buttons take up more screen "real estate" than combo boxes and therefore are best used when you need to offer users only a small selection.

Here are the instructions to construct two radio buttons:

helloButton = new JRadioButton("Hello");

goodbyeButton = new JRadioButton("Goodbye");

Lay out the buttons on the screen:

JPanel southPanel = new JPanel();

southPanel.add(helloButton);

southPanel.add(goodbyeButton);

add(southPanel, BorderLayout.SOUTH);

Add the two radio buttons to the frame of the HelloViewer program. Add a createSouthPanel method, just like you did in the previous problems. Run the program. What happens when you click the radio buttons? Why is this behavior wrong?

**As radio buttons, there should only be 1 active button. As of right now you may select both buttons at once. Which is precisely what radio buttons are designed not to do;**

8.2) To get the “radio button” effect, where all other buttons are turned off when one of the buttons is clicked, you need to place the buttons in a button group:

ButtonGroup group = new ButtonGroup();

group.add(helloButton);

group.add(goodbyeButton);

Finally, you want to turn the first button on:

helloButton.setSelected(true);

Now what happens when you click the buttons?

**The radio buttons behave as they are intended. With one button turning off as the other is turned on.**

8.3) Now change the message text when the user clicks a radio button. To assemble the correct text, simply check which button is currently selected.

class MessageListener implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

String message = "";

if (helloButton.isSelected()) { message = "Hello"; }

else if (goodbyeButton.isSelected()) { message = "Goodbye"; }

message = message + ", World!";

label.setText(message);

}

}

Note that you can attach the *same* action object to both buttons.

Add the action listener and attach it to the radio buttons. Run the program. Observe how clicking the buttons changes the message text.

What is the complete source code for your createSouthPanel method?

8.4) Check boxes are used to allow users to select a single logical alternative, typically of the form "do X" or "do nothing". Here, a check box is used to allow the user to specify whether the word "cruel" should be added to the message text:

Follow these steps to add the check box to the program.

1. Construct the check box.

cruelCheckBox = new JCheckBox("Cruel");

2. Attach the *same* action listener to the check box and the radio buttons.

cruelCheckBox.addActionListener(listener);

3. Add the check box to the south panel.

southPanel.add(cruelCheckBox);

4. In the actionPerformed method of the MessageListener class, add the action that should occur when the box is checked.

if (cruelCheckBox.isSelected()) . . .

Run the program. Observe how clicking the check box changes the message text.

What is the complete source code for your createSouthPanel method now?

9.1) A timer notifies a listener at regular time intervals. The time interval is given in milliseconds. The listener must implement the ActionListener interface.

For example, the call

Timer t = new Timer(1000, listener);

t.start();

causes the timer to call the actionPerformed method once per second.

To see the timer at work, install a listener that simply prints out the current time. To print out the current time, call

import java.util.Date;

. . .

System.out.println(new Date());

Supply a class CurrentTimePrinter that implements the ActionListener interface and whose actionPerformed method prints the current time.

What is the code for the CurrentTimePrinter class?

9.2) Now put together a demo program that prints the current time once each second.

Construct a CurrentTimePrinter, construct and start a timer, and put up a message dialog so that the program user can quit the program.

JOptionPane.showMessageDialog(null, "Quit?");

System.exit(0);

Enter your TimerDemo program here.

9.3) The program in Lab 11.9.2 keeps printing the time, once per second. In this exercise, you will modify the program so that it stops the timer after 15 seconds, restarts it after another 15 seconds, stops it after a further 15 seconds, and so on.

Of course, we will use a second timer for this purpose. Here is the implementation:

class TimerToggler implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

if (t.isRunning()) { t.stop(); }

else { t.restart(); }

}

}

ActionListener listener2 = new TimerToggler();

Timer t2 = new Timer(15000, listener2);

t2.start();

Add the TimerToggler class and the code that defines t2 to the TimerDemo program after the instruction for starting t and before displaying the option pane. Compile the code.

What compiler error do you get? Why? What can you do to avoid it?

**I get no errors whatsoever no matter how long I run this. I also found this question on another university Java lab. It was from when Java was still 2.5.3 being the latest. I’m assuming the earlier versions of java had some sort of problem with final vs non-final. But I can’t replicate the errors.**

9.4) Fix your TimerDemo program so that the first timer is declared as final. Methods of inner classes can access only final local variables of the enclosing method.

Your program should now compile and run. Execute it for approximately one minute. What output do you get?

9.5) It is a nuisance that the user must stop the program by clicking a button. Solve that problem by automatically stopping the program after two minutes (or 120,000 milliseconds). Simply make a timer listener that calls System.exit(0) in the actionPerformed method of its action listener.

Write the code for the third timer.

9.6) Now add the code for your third timer to the program and remove the last two lines of main, that is, the message dialog display and the call to System.exit(0).

Compile your program and run it.

Does it work as expected? If not, why not?

**There is nothing to ‘catch’ the program. It ‘falls’ thru all the code. Need something for the compiler to output while the timers run their code**

9.7) The program exited immediately when exiting main, so the timers never ran.

To keep main alive, you'll need to add the message dialog back in. Simply add the line

JOptionPane.showMessageDialog(null, "Please wait");

to the end of main.

Now the program will run for two minutes, and then exit. What output do you get?

**BUILD SUCCESSFUL (total time: 2 minutes 1 second)**

10.1) A user-interface component such as a button can detect five types of mouse events. What are these events?

**MOUSE\_PRESSED, MOUSE\_RELEASED, MOUSE\_CLICKED, MOUSE\_MOVED, MOUSE\_DRAGGED**

10.2) Th exercise focuses on "entered" and "exited" mouse events. Write a mouse listener class that prints "Mouse Entered" and "Mouse Exited" to System.out when the mouse has entered or exited the frame. Do nothing for the other three mouse event types. Call your class EnterExitListener. Remember that you need to implement the MouseListener interface.

Paste the code of your listener class below.

10.3) Write a MouseListenerDemo program that demonstrates your listener. You will need to add the listener to the frame using the addMouseListener method.

10.4) The MouseListenerDemo program is a little dull. Make it more interesting. Rather than printing a message, change the background color of the frame. In the mouseEntered method, add the line

frame.setBackground(Color.BLUE);

and in the mouseExited method, add the line

frame.setBackground(Color.RED);

Make sure that your EnterExitListener is an *inner class* of the main method.

Compile and run your test program. When the mouse enters the frame, the background color should change to blue. When the mouse leaves the frame, the background should turn red.

Enter the code for your MouseTester program here.

10.5) Why must the EnterExitListener be an inner class?

*Hint:* If you are unsure, make EnterExitListener into a regular class, compile, and study the error message.

**It cannot be a static class since a static class is a subset of the class and not of the object**

11.1) Write a component whose paintComponent method draws four lines:

1. From the point p to the top left corner

2. From the point p to the bottom left corner

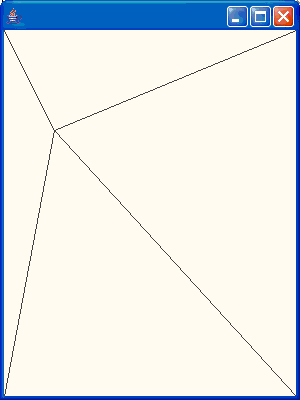
3. From the point p to the top right corner

4. From the point p to the bottom right corner

Let p be the point with the coordinates (50, 100).

Call your component FourLineComponent.

*Hint:* Call getWidth() and getHeight() to get the x- and y-coordinates of the corner points.

If added to a frame and displayed, the component should look like this:

Enter the code for your component here.

11.2) Now write a viewer program that creates a frame, adds the FourLineComponent to the frame and displays it. Call it FourLineComponentViewer.

11.3) Now enhance the program so that the point p can be changed by a mouse click.

Add a method to the component that allows setting the point, so that it can be changed during the program execution. If you did not have x and y instance variables in the component, you will need to modify the paintComponent method too, so that the lines all have one end located at this point.

Paste the code of your modified FourLineComponent class here.

11.4) Write a MousePressListener class that implements the MouseListener interface. Its mousePressed method sets the center point to the *x*- and *y*-values of the mouse event.

Make sure that the mousePressed method calls the repaint method on the component, after changing the point. You can use the getX and getY methods to find the position where the user clicked:

event.getX()

Add a MousePressListener as mouse listener to the viewer program.

Run your program. Now you should be able to click the mouse and have the four lines meet at the point where you just clicked.

Enter the code for your program here.

11.5) Why is it important to call repaint in the mouse press listener? What would happen if you didn't call repaint?

Try it out. Comment out the call to repaint, recompile, and try your program again. What happens?

**If we don’t repaint the listeners will still capture events (as with mine I system displayed mouse coordinates) however if will never show the updates;**