Lab 10 Requirements

Create a new Eclipse workspace named "**`Lab10_1234567890`**" on the desktop of your computer (replace **`1234567890`** with your student ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**`Question1`**", "**`Question2`**", etc. If you do not remember how to create a workspace or projects, read the "*Introduction to Eclipse*" document which is on iSpace. Answer all the questions below. At the end of the lab, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**`Lab10_1234567890.zip`**" (replace **`1234567890`** with your student ID number). Upload the ZIP file on iSpace.

Question 1

Create a **`MyFrame`** class that extends **`JFrame`**. In the constructor of **`MyFrame`**, add all the necessary code to create a window of size 400 x 300 (check your lecture notes if you forgot how to do this). Then in the **`main`** method of the **`Start`** class, create an anonymous class that implements the **`Runnable`** interface with a **`run`** method that creates a **`MyFrame`** object, and use the **`javax.swing.SwingUtilities.invokeLater`** method to run that code on the event dispatch thread.

Check that your program works correctly.

In the constructor of the **`MyFrame`** class, comment out the call to the **`setTitle`** method. Run the program. What happens? Then undo the commenting out.

Comment out the call to the **`setSize`** method. Run the program. What happens? Then undo the commenting out.

Comment out the call to the **`setDefaultCloseOperation`** method. Run the program and close the window. Is the program still running or not? (Use Eclipe's **`Window`** → **`Show View`** → **`Console`** menu to check whether the program is running or not: the program is running if the red square button is visible in the console.) Then undo the commenting out.

Comment out the call to the **`setVisible`** method. Run the program. What happens? Then undo the commenting out.

In a line before the call to the **`setVisible`** method, add a call to the **`setLocationRelativeTo`** method of the frame and give it an argument of **`null`**. Run the program. What happens?

In the **`run`** method of the anonymous class inside the **`Start`** class, add a second **`MyFrame`** object. Run the program. What happens? (Use the mouse to move the frame to check what happens!) What happens when you close one of the two frames? Then undo the change in the **`run`** method.

Question 2

In the constructor of the **`MyFrame`** class, add two buttons, each with different text. Run the program. What happens?

Note: make sure you add the buttons before calling **`setVisible`**, otherwise the buttons might not be visible in the frame!

Note: be careful to use the **`JButton`** class from the Swing GUI library, not the **`Button`** class from the old AWT GUI library!

Note: clicking on the buttons does nothing. This is because no action is currently associated with the buttons. You will learn later how to do that, for now we are only concerned with the placement of components in a frame.

In a line before the call to the **setVisible** method, add a flow layout manager to the frame. The flow layout manager should left-justify rows of components, use a horizontal gap of 20 pixels between components, and a vertical gap of 40 pixels between rows of components. What happens now when you run the program?

Instead of adding to the frame two different buttons, add twice the *same* button object. Run the program. What happens? Then undo the change.

Question 3

Create the following components in the constructor of **MyFrame** and then add all these new components to the frame in a line before the call to the **setVisible** method:

```
JLabel l = new JLabel("Enter your name: ");
JTextField tx = new JTextField("Type Text Here");
JCheckBox cb = new JCheckBox("I agree");
JRadioButton rb = new JRadioButton("Yes");
JComboBox<String> cb1 = new JComboBox<String>(new String[]{"Red", "Green", "Blue"});
JComboBox<Integer> cb2 = new JComboBox<Integer>(new Integer[]{2, 7, -3, 24});
```

Add imports to your program as necessary. Then run the program to see what the different components do (click everywhere to test things!) Also use the mouse to resize the frame and check what happens with the components.

Question 4

Change your code to use a grid layout manager instead of using a flow layout manager. Set the number of rows to 5 and the number of columns to 5. What happens when you run the program? Does the program use the number of rows that you specified? Does the program use the number of columns that you specified?

Use the mouse to resize the frame. Does the layout change?

Set the number of rows to zero in the grid layout manager and try again. What happens?

Change your code to use a border layout manager instead of using a grid layout manager. A border layout manager defines 5 areas in the frame: **BorderLayout.*PAGE_START*** (top), **BorderLayout.*PAGE_END*** (bottom), **BorderLayout.*LINE_START*** (left), **BorderLayout.*LINE_END*** (right), and **BorderLayout.*CENTER***. Therefore, after changing your code to use the border layout manager, you also need to modify all the method calls to the **add** method of the frame to specify as second argument of the method in which of the five areas each component should be added. If you do not specify a specific area in which to add a component then Java will add the component in the **BorderLayout.*CENTER*** area by default.

What happens when you add more than one component to one of the five areas of the frame? What happens when one of the five areas does not contain any component?

Question 5

So far all the different components have been simply placed in the frame without regard for aesthetics or for the logical connection between some of the components. To fix this problem, use three **JPanel** to organize all the different components into three separate groups inside the frame:

- The first panel must use a border layout manager and must contain the two buttons, with one button in each of the two **BorderLayout.*LINE_START*** (left) and **BorderLayout.*LINE_END*** (right) zones of the panel.

- The second panel must use a flow layout manager with centered justification and must contain the **JLabel** and the **JTextField** components.
- The third panel must use a 2 x 2 grid layout manager for the **JCheckBox**, **JRadioButton**, and the two **JComboBox** components.

The frame itself must use a border layout manager and the three panels must appear in the **BorderLayout.*PAGE_START*** (top), **BorderLayout.*CENTER***, and **BorderLayout.*PAGE_END*** (bottom) zones of the frame. Use the **setBackground** method of each **JPanel** to give a different color to each panel. This will help you see where the panels are inside the frame.

Run the program and see what happens when you resize the window to make it very small or very big.

Question 6

Create a **MyPanel** class that extends **JPanel**. Override the **protected void paintComponent(Graphics g)** method inherited from **JPanel**, and, inside your new **paintComponent** method, use the **drawString** method of the **Graphics** object **g** to draw the string "**hello**" at coordinates (20, 80). Do not forget to call the **paintComponent** method of the superclass **JPanel** to clean the panel before drawing anything on it.

In the constructor of **MyFrame**, remove all the components (including all the panels) and all the layout managers (including the layout manager of the frame itself), then add a **MyPanel** object to the frame in a line before the call to the **setVisible** method.

Run the program and check that you correctly see the string "**hello**".

Note: if you do not remove the layout manager for the frame itself then the layout manager might shrink the size of the MyPanel until it is invisible inside the frame! If you remove the layout manager for the frame itself then the frame automatically uses its default layout manager which then expends the size of the MyPanel to be the full size of the frame.

Note: drawing a string inside the **MyPanel** using the **drawString** method of the **Graphics** object **g** is different from adding a **JLabel** component to the **MyPanel**. A **JLabel** object is a Swing component that can be added (and later removed) to the panel (or to the frame) to display a string, while drawing a string does not involve adding any component: when drawing a string, it is the pixels of the **MyPanel** itself which are directly modified.

To illustrate the fact that the **paintComponent** method of **MyPanel** is automatically called by the Java system every time the frame content needs to be redrawn, replace the call to the **drawString** method in the **paintComponent** method with the following method call, which draws the string "**hello**" at a random position inside the panel:

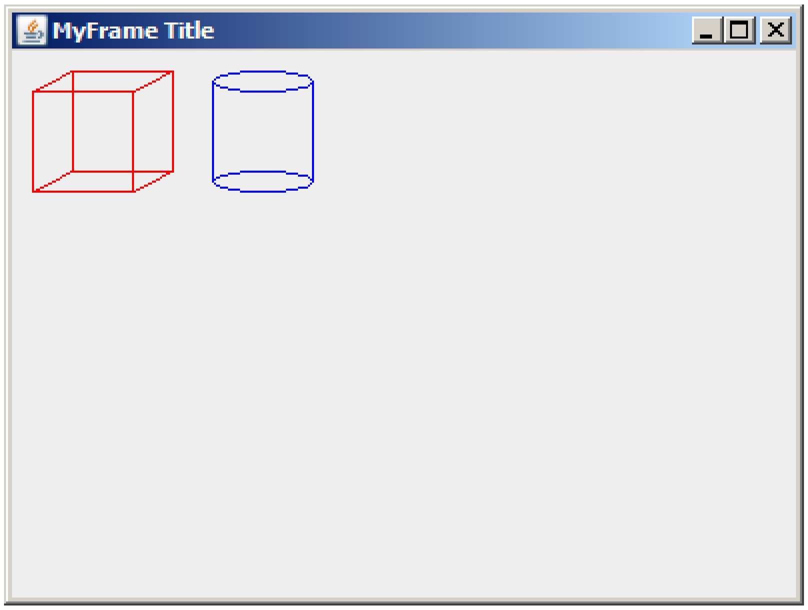
```
g.drawString("hello",
             (int)Math.round(Math.random() * this.getWidth()),
             (int)Math.round(Math.random() * this.getHeight()));
```

Run the program and slightly resize the window several times to see what happens. Also use the leftmost button in the top right corner of the window to minimize the window, then unminimize it. Also hide the window under another window (using for example the window of a web browser), then unhide it.

Note: how many times the **paintComponent** method is called and when it is called depends partly on which operating system you are using. You might get slightly different results on different operating systems.

Question 7

Remove the **drawString** method call from the **paintComponent** method of **MyPanel**, then use the **setColor**, **drawRect**, **drawOval**, and **drawLine** methods of the **Graphics** object g to draw a red 3D cube and a blue 3D cylinder, like this:



Hint: remember that the point with coordinates (0, 0) is in the top left corner of the panel, and that the horizontal x coordinate increases from left to right but that the vertical y coordinate increases from top to bottom!