Frames, Panes and Panels

A guide to Java GUI development in Netbeans

Phil Longenecker

June 2018

Frames, Panes and Panels

A guide to Java GUI development in Netbeans

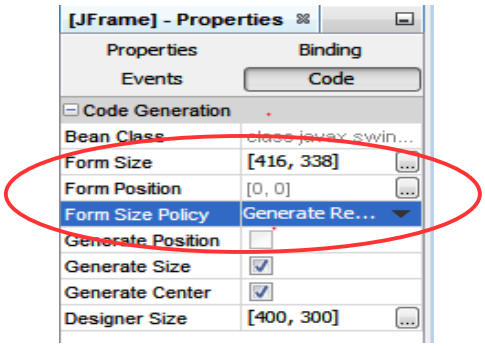
by Phil Longenecker
philofjava@gmail.com

Table of Contents

Introduction.

I will assume you are fairly adept at navigating in Netbeans. If you're still new to it, I suggest you check out Netbeans home on the web and learn about using the various windows, text editor and mouse actions. For this lesson, I will omit the obvious, i.e.: click OK, press enter, highlight line, etc. Have fun!

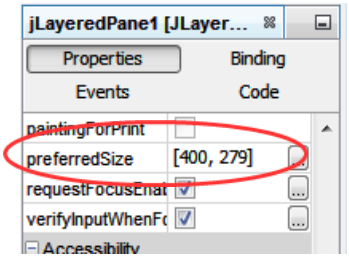
The allure of Java is the idea that using a Virtual Machine, code can be written once and run anywhere. The joke is, Java is write once, debug everywhere. It's really not that bad, although coders tend to write things that best run under only one OS, the OS they have and are used to. A Java GUI designed on Widows does not necessarily work the same on Linux. When using Netbeans, the default sizes for controls and the fonts within (labels, Menu Bar, etc.) can vary from PC to PC. I'm not talking about the fonts and sizes and colors used in the text editor window. Those can be set with Tools→Options→Fonts&Colors. That said, you'll find within this guide sizing numbers that come from the PC I used to create the examples you will build. The first thing you should do when you add a JFrame to a new project is find the Form Size numbers in the JFrame Properties Code window. Click



on the entry for Form Size Policy and change it to Generate Resize Code. Write down the Form Size numbers. If you know you want to start with a larger JFrame, change the Designer Size at the same time, before adding anything to the JFrame. The Frame will grow to the new size and the Form Size will change, too. Then change it back to Generate pack(). This is a good time to tick Generate Center, so your app will appear in the center of your display instead of the upper left corner.

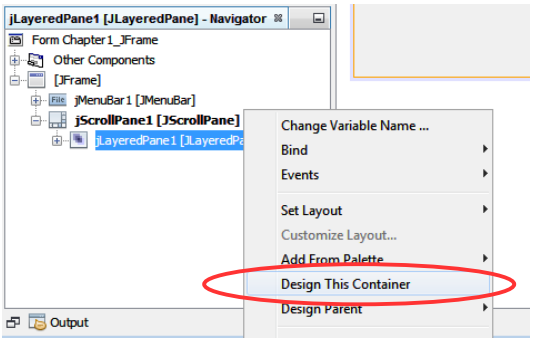
For a Windows7® PC with a screen resolution using 96 DPI and 'text and other items' set at 100% (the way I set my display for this guide), those numbers are 416, 338, which relate to horizontal and vertical sizes, respectfully. If you have your display set for 125%, you can get 418, 345 instead. So, designing for the display it will run on, is important. You will need the Form Size numbers for many of the projects throughout this guide. The next thing to keep in mind, is the size of panes and panels when

a Menu Bar is used in the JFrame. For instance, dragging a jScrollPane to use the entire available space in a frame with a Menu Bar (something you'll do) will result in a pane with a smaller vertical size than the Form Size. With these screen settings, for a Form Size of 400 by 300, you'll get a Menu Bar that's 21 vertical and you end up with jScrollPane height of 279 (300 less 21). Add a jLayeredPane and it will fill all of that jScrollPane and automatically assume the available size, 400, 279. This is in the Properties window as preferredSize.



Sometimes you'll directly set preferredSize in the Properties window but you'll also use your numbers when writing some code. In the Properties window, Layout Horizontal and Vertical Sizes in Properties will usually be set to 'Default'. Without following this practice, your application will start out with wrong sizes that can lead to scroll bars present when not needed. Code that re-sizes frames, panes and panels may have scroll bars covering controls you want displayed; you and your users will have to take the mouse and enlarge the frame every time it's used. Chapter 10 covers different screen resolutions for PC's with different size monitors, but ignore this difference for now.

Whenever I say "Select jLayeredPane1 in the Navigator window" or such, right click in the Navigator window on the container and select "Design This Container" before doing anything else. This assures that whatever you are about to do or add, goes on the desired container. If selected, that



container's name will show as bolded in the Navigator. Sometimes you can just double click on it, but that does not always work. I've gotten in the habit of right clicking, just to be sure I'm on the right container.

Let's get started on your first project.

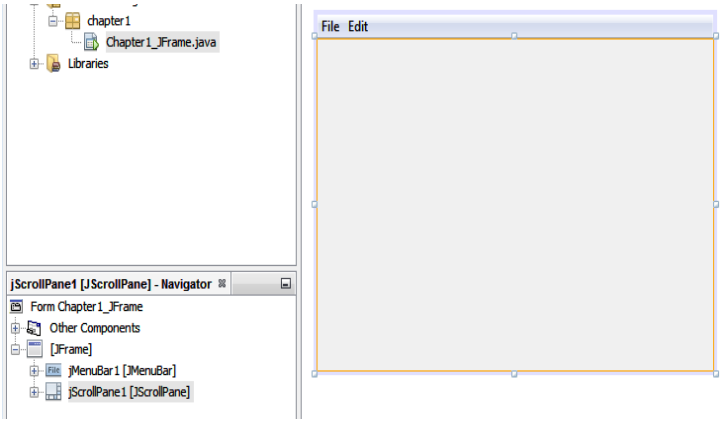
Ch. 1 Create a base form

For this guide, many of our projects will start with the same base form consisting of a JFrame, jMenuBar, jScrollPane and a jLayeredPane. Each project will be named for the chapter you are working on, with a Source Package and the JFrame named for the chapter, also. The containers and controls will use either the default name or a descriptive name. Descriptive names will usually be used when an Event is given to the control, that way it's a bit easier to recognize in the Source editor when writing code. In subsequent chapters, you'll copy this base form or other project forms and rename the copy, saving repetitions in construction.

Each chapter will have the steps grouped and numbered, in order, so follow along as shown.

1. Start a New Project in Netbeans. Select Java Application. If you don't have a specific folder already for the projects in this guide, click on the Browse button and create one for the Project Location. Name the folder whatever you want. Name the project 'Chapter1'. Uncheck 'Create Main Class'. The Project will be created and it will get listed in the Projects Window.
2. Right click on the project and select New, Java Package. Name the package 'chapter1'.
3. Right click on the Source Package just created in the Projects window and select New, JFrame Form. Name the Class Name 'Chapter1_JFrame'.
4. The GUI designer will show a new, blank JFrame form. Click on Code in the Properties window and note the Form Size as explained in my introduction and check the box next to Generate Center.
5. This step is not necessary, just makes the GUI easier to work with and gives MS style scroll bars. Click on the Source tab in the Designer window, go to the line below /* Set the Nimbus look and Feel */ and expand the code by clicking on the + box on the left. Change "Nimbus".equals… to "Windows".equals… Netbeans default Look and Feel should be set to Windows. To check that, go to Tools > Options > Appearance > Look and Feel. Windows will be the default if you have installed Netbeans on a Windows® PC.
6. Moving on, click on the Design tab in the editor window. Select the JFrame in the Navigator window. In the Properties window on the right, title this 'Chapter 1'. Right click on JFrame in the Navigator window and select Add From Palette, Swing Menus, Menu Bar.

7. Select the JFrame in the Navigator window again. Right click on it and select Add From Palette, Swing Containers, Scroll Pane. Click on Properties in the Properties window. Click on border-XPFillBorder. A selection of borders will pop up. Select No Border (that's my personal preference). Click on the corners of the jScrollPane1 outline and drag to the corners of the JFrame without going past the Menu Bar or the designer outline.



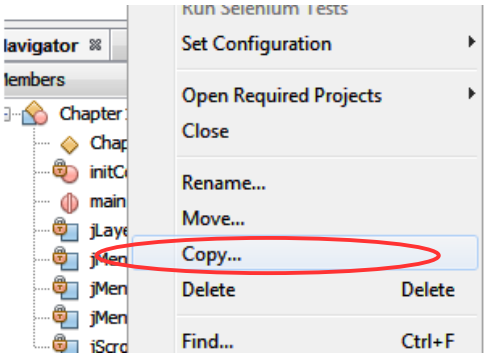
You'll get a 400×300 JFrame with a Menu Bar and a jScrollPane that fills the rest of the frame. Set the jScrollPane1 Layout Horizontal Size and Vertical Size to Default in the Properties window.

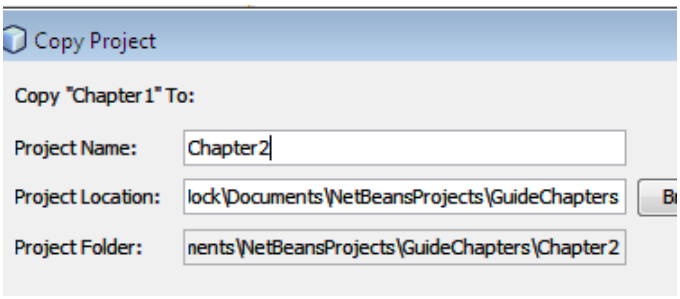
8. Select the jScrollPane1 in the Navigator window. Right click on it and select Add From Palette, Swing Containers, Layered Pane. The jLayeredPane1 will automatically fill the jScrollPane and show in the Navigator window below and inside jScrollPane1. In the Properties window, note the preferredSize setting, like I showed you in the introduction.
9. Run the project. You will get a pop up to select the main class. 'chapter1. Chapter1_JFrame' should be highlighted. Click on OK. After a few seconds, your new Java Application should show up in the center of the screen. When you drag the window (JFrame) smaller, scroll bars should appear. Drag it larger, and the grayish background should fill the window as you go.

This is your Base Form you can use for the next project.

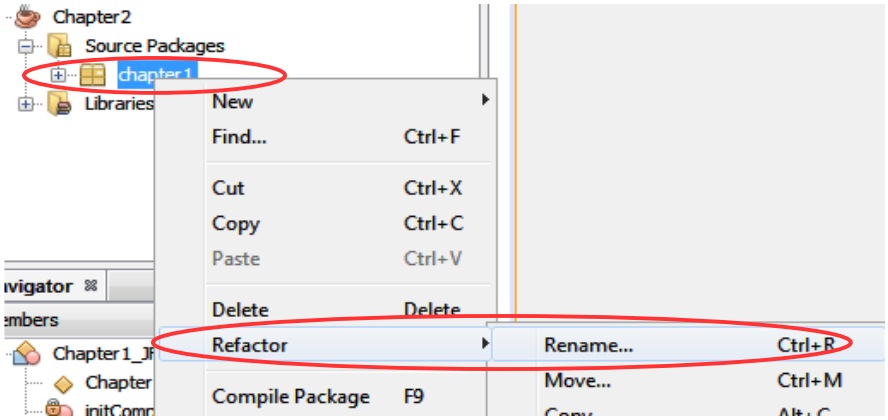
At the beginning of each chapter (Project), you may be asked to copy a project using the instructions in the two paragraphs below.

10. Copy your Project named Chapter1. Right click on "Chapter1" in the Projects and Files window, and select "Copy". Change the Project name to "Chapterx" where x=the new chapter. In the following examples, I copied Chapter1 to Chapter2.





11. A new Project, Chapter2, will appear in the Projects and Files window, and will be highlighted. Open the Source Packages tree and right click on ‘chapter1’, and select Refactor → Rename. For this example, use “chapter2” for the New Name and click the Refactor button.

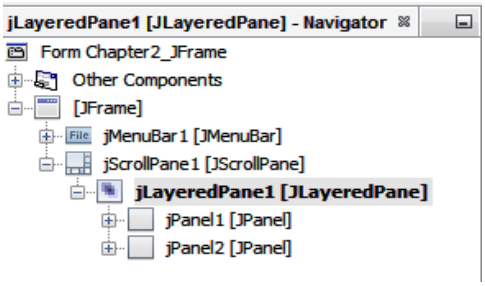


Expand the chapter2 package tree and right click on ‘Chapter1_JFrame’ and Refactor > Rename to ‘Chapter2_JFrame’. Open Chapter2_JFrame and change the title to Chapter 2 in the Properties window.

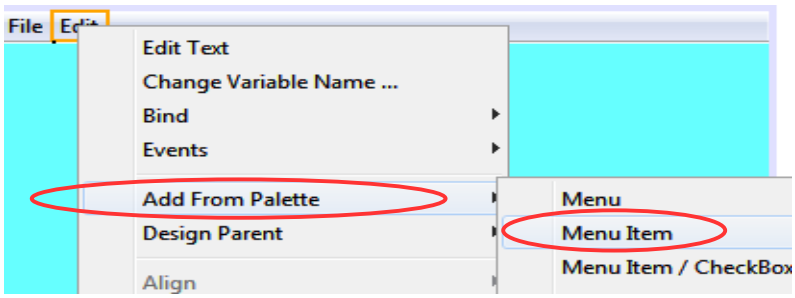
Ch. 2 Multiple full window jPanels with scroll bars

You need a desktop application with one window (JFrame) that contains multiple full-size panels. The window opens without scroll bars visible. The window is re-sizable, and scroll bars appear when the window is dragged smaller than the initial size. When the window is dragged larger, scroll bars disappear and the visible panel expands to fit the window. A jMenuBar with jMenuItems are used to switch views between the different Panels.

1. Copy the Project named Chapter1, title and rename it Chapter2. Right click on the JFrame in the Projects and Files window and select Open. Expand the JFrame tree in the Navigator window.
2. Expand the jScrollPane1 in the Navigator window. Select jLayeredPane1, right click on it and select Add From Palette, Swing Containers, Panel. So you can see it better, click on 'background' in the Properties window and select some pleasing color. Drag the corners of the Panel so it fills the designer, but no larger. Add another Panel. Make sure you're adding the second panel to jLayeredPane1, not jPanel1. You will now have jPanel1 and jPanel2 under the jLayeredPane1 in the Navigator window.



3. Now, let's write some code. But first, we want a place to put it. Select the JFrame in the Navigator window. In the editor window, select Design, if needed. Right click on 'Edit' in the Menu Bar, select Add From Palette, Menu Item.



Right click on the new item and select ‘Change Variable Name’ and change the name of ‘jMenuItem1’ to ‘jMenuItem_ShowPanel1’. Change the text of the Item to ‘Show Panel 1’. Right click on it and select Events, Action, actionPerformed. The editor window will change to the Source tab. Go back to the Design tab and create another Menu Item like you just did but name it ’Panel2’ instead of ‘Panel1’. You’ll now have both events listed in the Source tab, ready for editing. In the jMenuItem_ShowPanel1ActionPerformed event, add the following:

```
this.jPanel1.setVisible(true);
this.jPanel2.setVisible(false);
```

Copy these two lines to the jMenuItem_ShowPanel2ActionPerformed event and swap (true) and (false). Above the ShowPanel1 event, create a new method, ‘private void moreInit()’ and paste the two lines there, too. Go way up to top of the source code to the ‘public Chapter2_JFrame()’ constructor and after the line ‘initComponents();’ add a line with ‘moreInit();’.

Here’s what you should end up with in the Source window:

```
public Chapter2_JFrame() {
  initComponents();
  moreInit();
}

/**
 * This method is called from within the constructor to i
 * WARNING: Do NOT modify this code. The content of this r
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Co
private void initComponents() {...109 lines }// </editor-fol

private void moreInit() {
  this.jPanel1.setVisible(true);
  this.jPanel2.setVisible(false);
}
private void jMenuItem_ShowPanel1ActionPerformed(jav
  this.jPanel1.setVisible(true);
  this.jPanel2.setVisible(false);
  // TODO add your handling code here:
}

private void jMenuItem_ShowPanel2ActionPerformed(jav
  this.jPanel1.setVisible(false);
  this.jPanel2.setVisible(true);
  // TODO add your handling code here:
}
```

This will make sure that when the program starts only jPanel1 shows. In case you were wondering, you can have jPanel2 show at start by swapping the ‘true’ and ‘false’ settings in the moreInit() method.

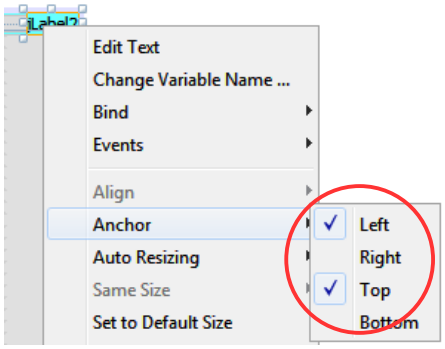
4. Run your program. JPanel1 should show on start. jPanel2 will show when you use the jMenuBar and click on Edit → Show Panel 2, and jPanel1 will no longer show. If you shrink the window(JFrame) scroll bars will appear and when you enlarge it, the scroll bars will disappear while the panel will grow to fill the window. If it doesn't check the Properties, Layout, Horizontal and Vertical Resizable are checked in the jPanels and jScrollPane1.
5. You can add more panels the same way you just did for jPanel2. Here's a hint on how to add more panels. Select jLayeredPane1 then Add From Palette another Panel, as usual. Give the new panel a different background color (something dark) so you can see it re-size. To make it fill jLayeredPane1, drag the lower right corner of the new panel to the lower right corner of the designer outline. You should be able to see a "shadow" of the lower right edge of the new panel as it is dragged to the edge. Stop dragging it before going past the Designer outline. Now drag the upper left corner of the new panel to upper left corner of the designer and you'll see it fill the designer with the color you chose as it meets the edge.

This is the best way to add overlaid panels, the editor does not make the other panels invisible as you drag the new panel out to the edges of the designer, and you won't be able to see where you're dragging it unless it's done this way. When you add the first few panels, the new panel will obliterate the showing panel. After that, the showing panel and other panels stay visible, thus the trick about looking for the shadow. You may want to practice this a couple of times to get used to seeing that shadow as you drag it out to the right corner. Add a visibility switch for any additional panels in their jMenuItem_ShowPanel(x)ActionPerformed methods and the moreInit() method. The more the merrier, as they say.

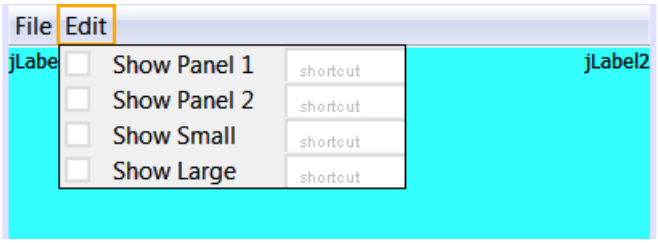
Ch. 3 Expanding JFrame

You want a project like you just made to have a window (JFrame) that you can expand with Menu click. Scroll bars will work as they do in Chapter 2, at launch, and after you change the size of the window by clicking on the Edit → jMenuItem. We'll add some labels and make them stay in the same position.

1. Copy the Project named Chapter2, Refactor and rename it Chapter3. Click on the JFrame in the Projects and Files window and select Open.
2. Select jPanel1. Add a label from the Palette to each of the corners of the panel. Right click on each of the labels and set their Anchor to Left, Top.



3. Select jPanel2. Add four labels to the corners of jPanel2. Set their Anchors to Left, Top. Run your project, all labels should remain in their positions no matter the size of the window.
4. Select the JFrame and add two more jMenuItems to the Edit menu, using the same method as you did in chapter 2.3, rename and relabel them to Show Small and Show Large respectively (format the variable names like you did in chapter 2).



Add actionPerformed events for both. You will end up at the Source editor window, ready to add code to these events.

Add the following code to the Show Small actionPerformed event:

```
this.jLayeredPane1.setPreferredSize(new Dimension(400, 279));
this.setSize(416, 338);
```

and add to the actionPerformed Show Large event:

```
this.jLayeredPane1.setPreferredSize(new Dimension(600, 279));
this.setSize(616, 338);
```

The added Source code should look something like this:

```
private void jMenuItem_ShowSmallActionPerformed(java.awt.event.A
  this.jLayeredPane1.setPreferredSize(new Dimension(400, 279));
  this.setSize(416, 338);    // TODO add your handling code here:
}

private void jMenuItem_ShowLargeActionPerformed(java.awt.event.
  this.jLayeredPane1.setPreferredSize(new Dimension(600, 279));
  this.setSize(616, 338);    // TODO add your handling code here:
}
```

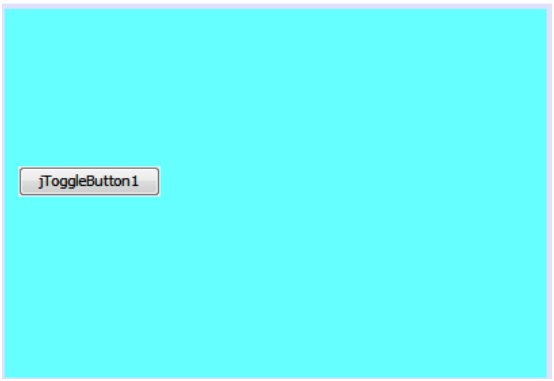
You'll have to import java.awt.Dimension. Right click while the cursor is on the line with 'Dimension', select 'Fix Imports'. For Show Small, 416, 338 are the numbers I got when I changed the Form Size Policy and I wrote them down to use here. In Show Large, 616, 338 is what I want to expand to, although the window will only be 600×300. The 279 size comes from the jScrollPane and jLayeredPane1 properties. Use the numbers you have in your project! This is auto generated to take in account the size of the Menu Bar added to the JFrame. You'll need the numbers for your project, that you took note of at the beginning of this lesson.

5. Run your application. You should have a normal sized window at start, with labels in the corners, that will have scroll bars when dragged smaller and be able to select which panel you want to see. When you select the menu Edit → Show Large, the window will expand, without scroll bars, and the labels will remain in their original place. Dragging the window smaller will make the scroll bars appear. Click on Show Small and the window will revert to the initial size.

Ch. 4 Expanding hidden jPanel

You want a JFrame that will expand to reveal another jPanel alongside the currently viewed jPanel. You will set this up with a button that will toggle the change. This jPanel can be used for settings or configurations, it appears only when called for.

1. Copy the Project named Chapter1, Refactor and rename it Chapter4. Change the JFrame title. Right click on the JFrame in the Projects and Files window and select Open.
2. Select jLayeredPane1, add a panel, give it a green color and drag it to fill only the right half of jLayeredPane1. In the Properties of jPanel1, the preferredSize should be 200, 279 (or 200, and whatever your project has for vertical size) and the Layout H & V Size should both be 'Default'. Layout, Horizontal Resizable should be unchecked.
3. Add another Panel to jLayeredPane1, make it full sized, give it a blue color. The Layout H & V Size should both be 'Default'. Vertical and Horizontal Resizable should be checked.
4. Select jPanel2 and add a Toggle Button to the left half of the panel. Right click on it and select Anchor, Left, Top. In the Properties window, check 'opaque' so we can change the outline color of the button later. Add a actionPerformed Event to jToggleButton1.



5. Add the following code to the jToggleButton1ActionPerformed event in the Source editor:

```
if(jToggleButton1.isSelected()){
   jToggleButton1.setBackground(Color.blue);
   jToggleButton1.setText("Contract <--");
   this.jLayeredPane1.setPreferredSize(new Dimension(600,279));
   this.setSize(616, 338);
   this.jPanel1.setVisible(true);
   this.jPanel1.setLocation(416, 0);
}else{
   this.jPanel1.setVisible(false);
   jToggleButton1.setBackground(Color.green);
   jToggleButton1.setText("Expand -->");
   jToggleButton1.setSelected(false);
   this.jLayeredPane1.setPreferredSize(new Dimension(400,279));
   this.setSize(416, 338);
}
```

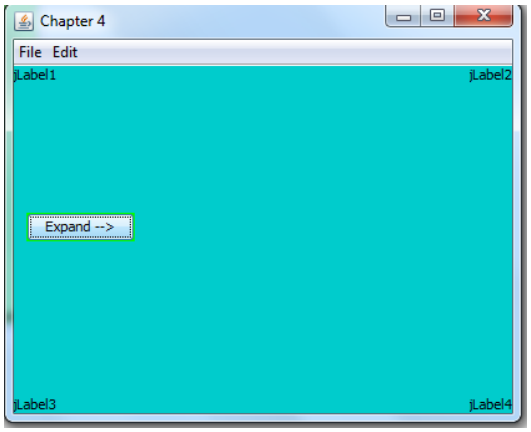
   Place the cursor on any line from above, right click and select Fix Imports. These numbers give your application a window that's 400×300 or 600×300, depending on whether the jToggleButton1 is selected (clicked on) or not. Again, use the numbers from the start of your project.

6. In the Source editor, create a method called 'moreInit' like you did in Chapter2. Add the following code:

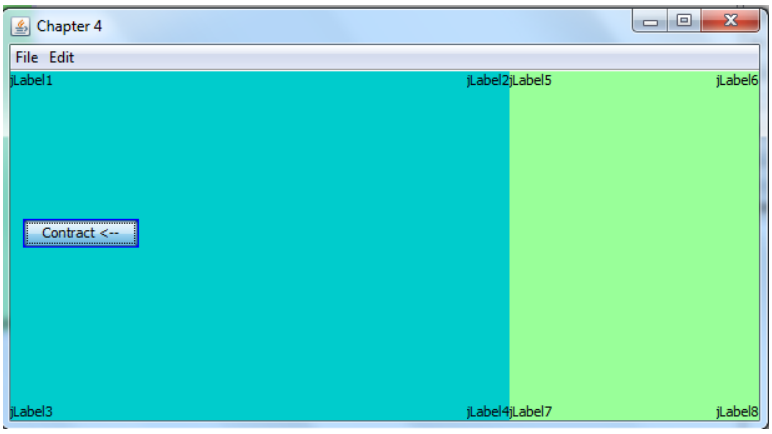
```
this.jLayeredPane1.setPreferredSize(new Dimension(400,279));
this.setSize(416, 338);
this.jPanel1.setVisible(false);
this.jPanel2.setVisible(true);
jToggleButton1.setBackground(Color.green);
jToggleButton1.setText("Expand -->");
```

7. Go way up to top of the source code to the 'public Chapter4_JFrame()' constructor and after the line 'initComponents();' add a line with 'moreInit();'. This will set the initial size and view to a 400×300 window, with only jPanel2 visible. We will not make Menu Items for showing either jPanel in this Project.

8. Run your project. At start, jPanel2 will show, scroll bars appear when the window is dragged smaller. The toggle button with have the text ‘Expand –>’ and have a green outline.



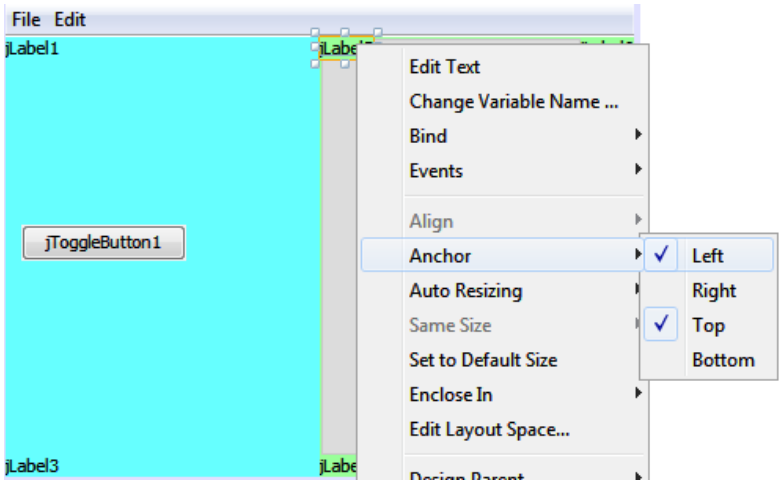
Clicking the Expand button it have a blue outline and make the window larger, with jPanel1 added at the right.



Again, scroll bars will appear when the window is dragged smaller. JPanel1 will reduce to invisible if you drag it small enough. Drag the window wider, and jPanel2 will expand, but jPanel1 will not become larger than its original size, and will always be at the right of the window. If you set jPanel1 Layout Vertical Resizable checked, it will fill from top to bottom of the window whatever size you drag the window. Click the button again and the window will go back to the initial size and jPanel1 will not be visible. Dragging the window wider will expand jPanel2 but jPanel1 does not appear.

9. You can prove the sizing by adding labels to the corners of the panels.

   Set Anchors on the labels to keep them relative to each other, like you did in chapter 3.4. Add labels to both jPanels.



   Remember to right click on the jPanel in the Navigator widow and select Design This Container then add labels from the palette. You should see jPanel1 and its designer appear at half the size of jPanel2. Do not change the designer size.

Ch. 5 Adding a hidden jPanel after the project is built

What if you already have a project, and get asked to (or told to) add a hidden configuration panel? There's a trick to make this work, and it's the only way I can find to do it, unless you build the project again, from scratch. But we'll just copy our Chapter 2 project so we have a plain, two panel application to start with. Name and title this copy Chapter5, with the usual refactoring. Check the Designer Size!

1. Color jPanel1 yellow. Color jPanel2 blue. Run the project, you should be able to switch panels and scroll bars should work as usual.
2. Select jLayeredPane1 and create a third panel at half size on the right half of the pane. It should have a size of 200, 279. Color it green. If you're having trouble with creating the third jPanel, see Ch 2.5 for a hint.
3. Go to the Source in the editor and change the lines of code to the moreInit() method:

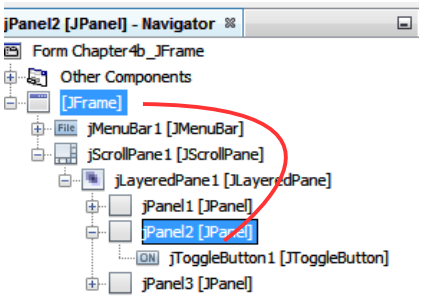
```
this.jLayeredPane1.setPreferredSize(new Dimension(400,279));
this.setSize(416, 338);
this.jPanel1.setVisible(true);
this.jPanel2.setVisible(false);
this.jPanel3.setVisible(false);
jToggleButton1.setBackground(Color.green);
jToggleButton1.setText("Expand -->");
```

   Ignore the errors for now.
4. Select jPanel2 and add a Toggle Button. Set it to 'opaque' and create an Action Event for it. Put the following code in the Event on the Source page:

```
if(jToggleButton1.isSelected()){
   jToggleButton1.setBackground(Color.blue);
   jToggleButton1.setText("Contract <--");
   this.jLayeredPane1.setPreferredSize(new Dimension(600,279));
   this.setSize(616, 338);
   this.jPanel3.setVisible(true);
   this.jPanel3.setLocation(416, 0);
}else{
   this.jPanel3.setVisible(false);
   jToggleButton1.setBackground(Color.green);
   jToggleButton1.setText("Expand -->");
   jToggleButton1.setSelected(false);
   this.jLayeredPane1.setPreferredSize(new Dimension(400,279));
   this.setSize(416, 338);
}
```

Add the Imports for Color and Dimension when prompted. This pretty much the same as you did in Chapter 4.

5. Run the project. Select the menu item to Show Panel 2. Click on the toggle button. The panel expands, but Panel 3 doesn't show on the right like you expect. Exit the application and return to the Design editor.
6. Select jLayeredPane1. You must Select jLayeredPane1 first. In the Navigator window, use your



mouse and drag jPanel2 from jLayeredPane1 to the JFrame as shown. Then drag it back into the jLayeredPane1. Back to its old position below jPanel1. Select jLayeredPane1 then click on jPanel2. Drag the corners of jPanel2 in the designer to fill jLayeredPane1 again. Make sure you don't exceed the designer size. Make sure you do this exactly as I've written to do.

7. Run your project. When it starts, only jPanel1 will show, scroll bars will appear when the window is dragged smaller. Use the menu to change to Show panel 2. The toggle button will be there and have the text 'Expand –>' with a green outline. Clicking the button will change the button outline to blue and make the window larger, with jPanel3 added at the right. Click the toggle button to close the additional panel on the right.

The trick to this project is in paragraph 6. You must drag the panel with the jToggleButton out of and back into jLayeredPane1. That's because of the way Netbeans constructs the JFrame. You can see this in the grayed out area of the method initComponents() in the JFrame Source tab. In the section of the jLayeredPane1, the jPanels are placed differently than what was done before dragging jPanel2 out and back in. At first, the code for jLayeredPane1 Group Layout has this:

```
    jLayeredPane1Layout.setHorizontalGroup(
      jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 400, Short.MAX_VALUE)
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jLayeredPane1Layout.createSequentialGroup()
          .addGap(0, 200, Short.MAX_VALUE)
          .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
    );
    jLayeredPane1Layout.setVerticalGroup(
      jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 279, Short.MAX_VALUE)
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel3, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
```

After you dragged jPanel2 out to the JFrame, then back into the jLayeredPane1, it changed:

```
    jLayeredPane1Layout.setHorizontalGroup(
      jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 400, Short.MAX_VALUE)
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jLayeredPane1Layout.createSequentialGroup()
          .addGap(0, 200, Short.MAX_VALUE)
          .addComponent(jPanel3, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    jLayeredPane1Layout.setVerticalGroup(
      jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 279, Short.MAX_VALUE)
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel3, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      .addGroup(jLayeredPane1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
```

The differences are highlighted in green (before the drag), and red (after the drag). After the drag, jPanel2 is added last, after jPanel1 and jPanel3. You can't change the order in the Source view. The lines are grayed out. You could open the Chapter5_JFame.java in a text editor like NotePad++ and make the changes. But this way is quicker.

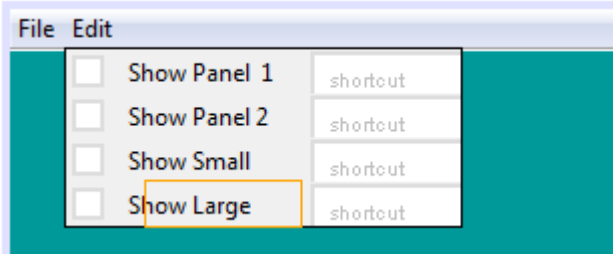
You could add the toggle button to jPanel1 instead of jPanel2. In that case, you have to drag jPanel1 out and in to make this work. You have to use the toggle button to hide the 'configuration' panel. You could add code to make a menu item do it that. You could add code to make the toggle button request a password before the hidden panel is shown. Passing an argument when the application is launched could determine if the toggle button is visible or not. There's going to be a few ways to protect that hidden panel from showing.

Add labels to the corners of each panel. Set the Label anchors to Top, Left. You can add additional panels to this project in the usual manner. The hidden panel will still only be accessible from the toggle button. Just remember to add code to handle the new panel's visibility in the moreInit() method, and any Show Panel(x) Menu Item Action Event.

Ch. 6 Combine previous styles

So what if you want the styles of the applications you just made combined in one application? That's easy enough to do.

1. Copy the project from Chapter 5, title and rename it Chapter 6 in the usual manner. If you didn't add labels in Chapter 5 do so now, the way you did in Chapter 4.
2. Add two more jMenuItems to the Edit menu. Rename and relabel them to Show Small and Show Large respectively (format the names like you did in chapter 1).



3. Add actionPerformed events for both new menu items.

   Add the following code to the Show Small actionPerformed event:

```
this.jLayeredPane1.setPreferredSize(new Dimension(400, 279));
this.setSize(416, 338);
```

   and add to the actionPerformed Show Large event:

```
this.jLayeredPane1.setPreferredSize(new Dimension(600, 279));
this.setSize(616, 338);
```

   Your Source code should end up looking something like this:

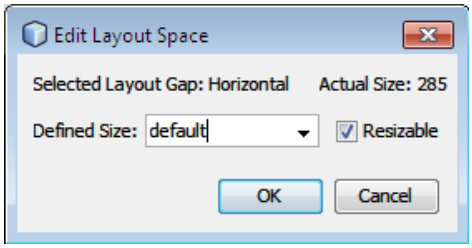
```
private void jMenuItem_ShowSmallActionPerformed(java.awt.event.A
    this.jLayeredPane1.setPreferredSize(new Dimension(400, 279));
    this.setSize(416, 338);    // TODO add your handling code here:
}

private void jMenuItem_ShowLargeActionPerformed(java.awt.event.
    this.jLayeredPane1.setPreferredSize(new Dimension(600, 279));
    this.setSize(616, 338);    // TODO add your handling code here:
}
```

4. For Show Small, 416, 338 are the numbers I got when I changed the Form Size Policy and I wrote them down to use here. In Show Large, 616, 338 is what I want to expand to, although the window will only be 600×300. The 279 size comes from the jScrollPane and jLayeredPane1 properties. You'll need the numbers from your computer, that you took note of at the beginning of this guide. This is the same as the project in Chapter 3.
5. Run your application. You should have a normal sized window at start, with labels in the corners, that will have scroll bars when dragged smaller and be able to select which panel you want to see. When you select the Menu -> Show Large, the window will expand, without scroll bars, and the labels will remain in their original place. Dragging the window smaller will make the scroll bars appear. Click on Show Small and the window will revert to the initial size.

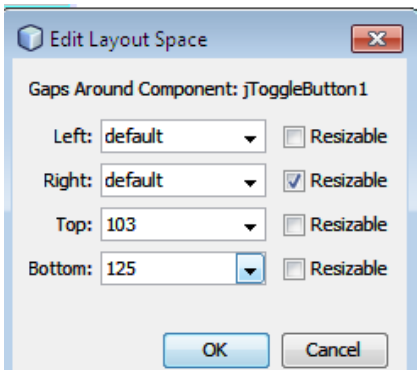
In this case, when you click on the 'Expand' toggle button, the hidden panel appears, but if you then use the Show Small Menu Item, the hidden panel retains its size while jPanel2 (the one with the button) shrinks horizontally. The jPanel2 labels on the right side are covered up. The only way to restore jPanel2 is to drag it, use the toggle button to hide the hidden panel, or use the Show large Menu Item. You could add code to do this when Show Small is used, just copy the code from the toggle button Action Event to close it and put it in the Show Small Action Event code.

Ch. 7 Control placement

This Chapter can be hard. You'll be creating problems and I'll show you how to correct them. Don't feel bad if it takes you a couple of tries to get it right. You may want the controls on your panels to stay in the same location relative to each other when you drag or re-size the window (JFrame). When Netbeans creates a container like the JFrame, jScrollPane, jLayeredPane or jPanels, it sets the Layout to Free Design. That's the name Netbeans uses for Java's Group Design. With this layout manager you can place Swing Controls like labels, text fields, etc., where you want on the panel and control the placement and/or movement of these with the Anchor and Edit Layout Space functions found by right clicking on the Swing Control. Anchor, as you've used in previous chapters, sets which direction the control adheres to. It may be another Control or a 'gap' which can be horizontal or vertical. A gap will become 'fixed' if it's in the direction of the Anchor that's used on the Control. If the Control's Anchor is not in the direction of a gap, the gap will be resizable by default. A gap does not have the Anchor function, its' right, left, top, and bottom choices will be grayed out. A gap will have the Edit Layout Space available. When you select this, you'll get a pop-up where you can set Defined size and check Resizable. The gap can be set to nothing with the Defined Size of 0.

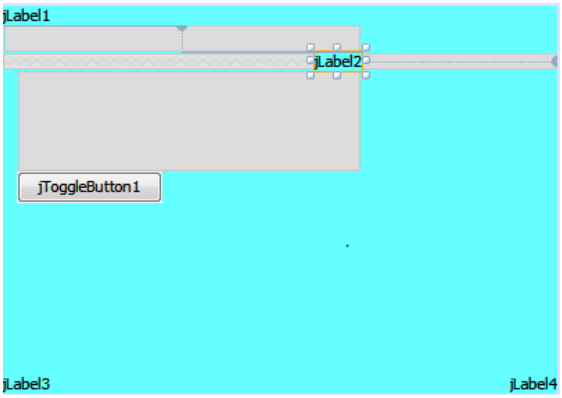


When you select Edit Layout Space on a Control (like a label), you get a pop-up with choices for setting gap sizes for Left, Right, Top and Bottom, each with check boxes for Resizable.



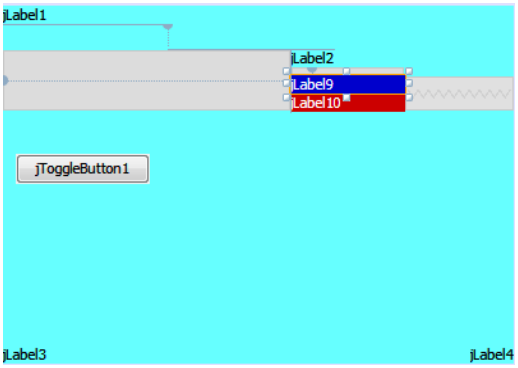
This is a way to set the Anchors on a Control with more options and control over the sizing. This becomes useful when trying to stack 2 labels without any gap between them. This a lot of information all at once, but what we'll do next is place labels that do and don't re-position when the window (JFrame) is dragged and place two labels vertically without any gap.

1. Copy the Project named Chapter4, title and rename it Chapter7. Open the JFrame for editing. If you didn't add labels to the corners of both jPanels in chapter 4.9, do it now.
2. Select jPanel2 and move jLabel2 like shown below, closer to the middle of the panel horizontally and a little lower vertically. Set its Anchor to Right, Top.

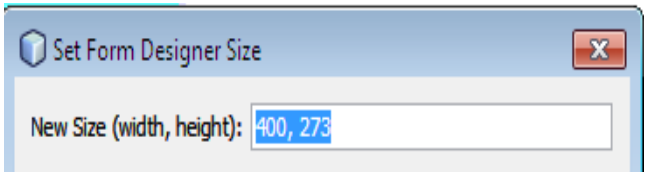


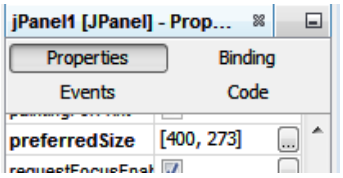
3. Run your application. Drag the window to the right far enough and you will see jLabel2 re-position to the right of jLabel4, which should not move. If not, check the Anchor on jLabel4 to make sure it's set for Left, Top. If any other control moves, check its Anchor and re-position jLabel2 a little, away from the other controls alignment. I'll explain how alignment works as you go along in this chapter.
4. Expand the window by clicking jToggleButton1 ("Expand"). JPanel1 will appear and overlay jLabel2. Dragging the window either way will keep jLabel2 in the same relative position to the right of the window. Click the button again, jPanel1 goes away and jLabel2 moves to its original position when the application started to run, which is still relative to the right side of the window.

5. Select jPanel2, add another 2 labels (jLabel 9 and 10) below jLabel2. You'll notice a broken line appears to help you align to the left of jLabel2. A series of horizontal broken lines will appear as you move jLabel9 below jLabel2. Position jLabel9 on the first horizontal line below and at the vertical line to the left of jLabel2. Place jLabel10 on the first horizontal line below jLabel9 and at the line at the left of jLabel9. If you need to, set jLabel2 a little further up on jPanel2 so jLabel10 is not placed to the right of jToggleButton1. Set both these new labels to opaque, then pick different background colors for each, so you can see the effect of stacking the labels. You may want to change the foreground color to white. Expand these labels horizontally so they're larger than jLabel2, but the same as each other.
6. Right click on both jLabel9 and 10 and set Anchor Left, Top. Right click on jLabel9 and select Edit Layout Space. Set Bottom to 0, uncheck Resizable if needed.



The Design window will show jLabel 9 and jLabel10 without any gap between them. But, as soon as you set Bottom to '0', the designer and preferred sizes can change. Position the cursor on the edge of the Designer outline and double click to open Set Form Designer Size.



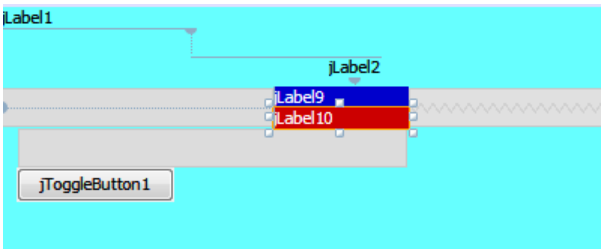


Change the numbers to your Project sizes (400, 279 in my case). Make sure you're doing this only on jPanel2 (in this case), not on another container. Change jPanel2 preferredSize in the Properties window back (also 400,279), if needed. Check that jPanel1 Layout Horizontal and Vertical Size is set to 'Default. The JFrame and jLayeredPane1 may have Designer Size changes along with jScrollPane1; they may need correcting too. The labels at the bottom of the panel probably have moved, also. Reposition them and check Anchors for all the controls.

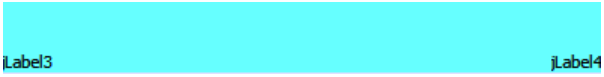
Tip- Move jLabel4 first, this may take two tries, check its' anchors then move jLabel3. What a mess, but this is only a demonstration.

7. Run your application. JLabel2 now is anchored to the left. Dragging the window keeps jLabel2 in the same position relative to the left of the window. Because the three labels were placed aligned to the each other, they all have the same anchor. Close it and go back to Design.
8. Hold down the shift key and click on jLabel9 and jLabel10. With both selected, drag them to a position like shown below. Change the positions of the new labels so they do not align with either side of jLabel2 or the button. Place the labels to the left of the left edge of jLabel2.
9. Set jLabel2 Anchor to Right, Top. Set both jLabel9 and jLabel10 anchors to Left, Top. Confirm JToggleButton1, jLabel3 and jLabel4 have an Anchor of Left, Top. If needed, drag jLabel2 or jToggleButton1 slightly away to disrupt the alignments enough to the correct Anchors. Check designer and container sizes again and make corrections.

   The controls anchors and gaps, Designer Size and anything else can change as you edit jPanel2.



Notice the gaps to the left and right of jLabel9 & 10 goes all the way to the edges of jPanel2.



10. Run your application. If you got everything right, when you drag the window you'll see jLabel2 moves to the right again, while the new labels stay in the same position to the left. When you use the Expand button, jLabel2 should be overwritten. If not, go back and try to get all controls and the Designer set correctly. It's may take a few tries, but it can be done. Why? Well...

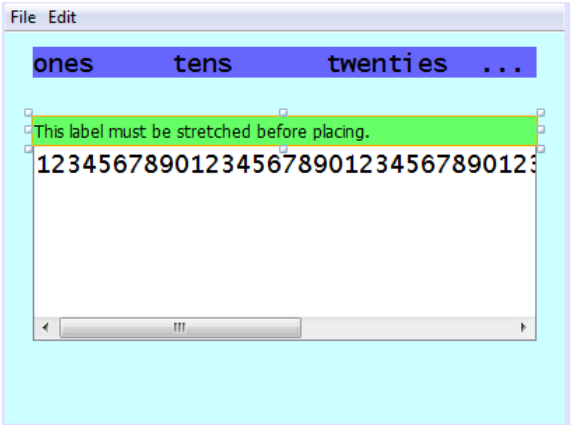
This project shows you how the broken line guides are used to align Controls like labels, and makes their anchors the same. To get independent positioning to the fullest, the Controls cannot be aligned. Generally speaking though, good English design will have Controls set to Anchor Left, Top and you align similar function controls when you place them. As you get creative you will have to be careful of alignment, lest you need to reset previously placed controls. Try placing more controls and getting them to stay in the positions you want on the panel no matter how the window is re-sized.

Ch. 8 A case for Absolute Layout

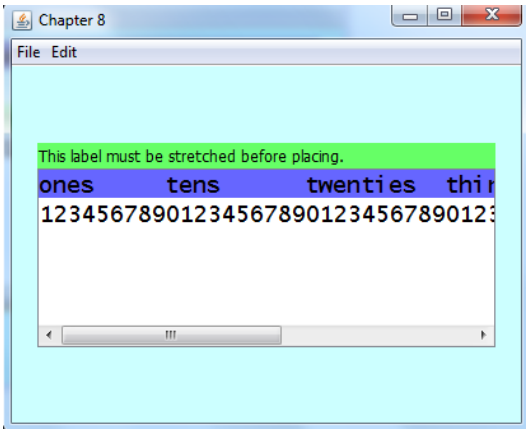
In all the projects you've done so far, you used Free Design for the layout manager. Absolute Layout will accomplish many of the same as Free Design, but can be much more trickier to do. Locating controls like labels in respect to other controls is straight forward with Free Design using the Anchor and Edit Layout Space. But, there is one case where I use Absolute Layout. When I want a Column Header on a list, and I want it to scroll horizontally with the contents of the list. I used this in an app that shows a long string for each list entry, and I wanted to save display space, so I'm good with the list scrolling. Each of the list entries consists of a formatted group of items. The font used in the list is monospaced so the items always line up vertically, so having a header makes sense to id the items as the list is scrolled down. I could have used a jTable, but I don't care for the dotted lines that outline a cell when clicked on, and the string of items is also used elsewhere in the app.

1. Copy the Chapter1 project, refactor and rename it Chapter 8.
2. Select the jLayeredPane1 and add a jPanel. Make it fill the jLayeredPane completely.
3. Select jPanel1 and choose Set Layout. Change the layout from Free Design to Absolute Layout. It may take some time for Netbeans to apply the change.
4. Add two labels then a list to jPanel1. It's important to have the two jLabels placed above the jList in the Navigator view. Use your mouse and move them if necessary. Stretch them out to almost fill the width of jPanel1, making the three the same width. You will not get alignment lines, just eyeball the width. Check 'opaque' on both labels and pick a background color.
5. Set the fonts for jLabel1 and jList1 to something like Lucida Sans Typewriter, Plain, 18 to have a monospaced font that's large. Re-size the jList smaller if needed.
6. For the text in jList1, choose 'model' in Properties and open the editor with the ellipses icon. Replace all the text the following line of text:
   1234567890123456789012345678901234567890123456789012345678901234567890
   Replace the text of jLabel1 and enter this the following line keeping the spaces intact:
   ones tens twenties thirties forties fifties
   There are three spaces at the end of the line.
7. Type something in jLabel2. Set the font and size whatever you want. Using the mouse, stretch the height of the label, one setting should do, then drag the entire label so it rests adjacent to the top of the jList. Usually, the jLabel will not rest on top of the list. If you check its Properties →

Layout → Height it is created as Preferred. Changing the height will allow you to place it exactly where you want it in Absolute Layout. You should have a designer that looks like this:



8. Right click on the jList and choose Customize Code. Near the end of the code, just before jScrollPane2.setViewportView(jList1); add the following:
   jScrollPane2.setColumnHeaderView(jLabel1);
9. Run your project. You should get a window that looks like this where jList1 will have a scrollbar at the bottom to move horizontally, and the header will scroll with it. The column names in jLabel1 will line up with the string of numbers in jList1 as it scrolls.



The trick here is to make jLabel1 the same width as jList1, and fill it with text equal to the length of the longest string of text in jList1. Using a monospaced font makes this pretty straight forward, but jLabel1 can have a proportional font, and different size, as long as it has the same width of text as the longest string in the list. It will require some trial and error to get the header text spaced where it lines up with the list text if you do that. In this example I used the monospaced fonts to make it easier to create.

Ch. 9 Resizing after building

So you successfully built an application, but now you've decided it too small. There's much more you want/need in at least one of the panels and no room to place anything. You don't want to re-build the whole project, (but you will copy one for this chapter), so let's see how to get bigger panels and make more room.

1. Copy the Project named Chapter3, title and refactor it Chapter9. Open the JFrame for editing.
2. Select the JFrame. Select code in the Properties window. Change the Designer Size from 400, 300 to 500, 400. The JFrame becomes larger in the Design window. Click on jLayeredPane1 in Navigator and look at preferredSize in the Properties window. The number there will have changed from 400, 279 to 500, 379.
3. Click on either jPanel and you'll see the preferredSize has not changed. Select jPanel1 in the Navigator window. The Designer Size is the new, larger size, the background color of jPanel1 fills it, but the labels are in the same relative positions, measured from left and top. In the Properties window, under Layout, the Horizontal Size and Vertical Size items show Default.
4. Select jPanel2 in the Navigator window. What happened? The Designer Size became smaller (the original size) and jPanel2 appears as it used to. Select jPanel1. It, too, has re-sized to the original, smaller Designer Size. Now Select jLayeredPane1 and the larger Designer Size comes back. Select jPanel2 again and the Designer Size is still larger, with jPanel2 showing. Are you confused yet? You're not the only one. Remember, jLayeredPane1 has the new, larger preferredSize, but the jPanels still have the original preferredSize. What this shows you is the parent container has precedence over the child container for many (if not all) things the application does. Netbeans Matisse GUI editor follows this logic. So, for the most part, whatever has precedence, its' configuration sets the stage.

5. Go to the Source window and let's make a couple of size changes to the Show Small and the Show Large actionPerformed events. Change the code lines of the Show Small event to:

```
this.jLayeredPane1.setPreferredSize(new Dimension(500, 279));
this.setSize(516, 338);
```

and change the code lines of the Show Large event to:

```
this.jLayeredPane1.setPreferredSize(new Dimension(700, 279));
this.setSize(716, 338);
```

6. Run the project. A larger window initializes, the background color of jPanel1 fills it, but the labels haven't moved to the corners. Drag the edges to re-size the window and the scroll bars act the same as before we started to change this application. Don't bother to click on any Menu item. Close the application and go back to Design.
7. Select jPanel1. In the Properties window, change the preferredSize to 500, 379. Select jPanel2 and do the same. After doing the Design This Container action again on both, you'll see the designer size change to the larger size. However, the labels haven't moved. Run the application, and that's what you'll see. The window starts out 500×400. Scroll bars behave as they used to. Drag the window larger. Use Show Large or Show Small. The window then contracts *vertically* to the old size. It should, we haven't finished changing the code for the Show actionPerformed events. Again you see that the parent container sets the minimum size before scroll bars appear due to the setPreferredSize for the jLayeredPane code in the events. But, what about the size of the jScrollPane? Well, if you check it out in its Properties window, the Layout Horizontal and Vertical Sizes are set to Default. The Default is the size of the JFrame, (the parent container of jScrollPane1) minus the Menu Bar, which is 500, 379. When the actionPerformed events change the JFrame (this.setSize() code in the jMenuItem_ShowPanel*x*ActionPerformed), then the Default will be the new size of 'this', the JFrame. Setting the jLayeredPane1 to the expected size of the jScrollPane1 (in this case, the new Default size) makes the Scroll Bars appear only when the window is dragged smaller.
8. Close your app and go back to Design. Select both of the jPanels. Move the labels on both panels to the new corner positions, check the anchors for Left, Top as they will change when they are moved. Run the application and the labels appear at the corners of the new bigger window, until you use the Edit menu to Show Small or Large. If you drag the window after using the Menu Items, the labels appear when the window gets big enough vertically.

9. Change the Show Small and Large events to the larger vertical sizes:

```
this.jLayeredPane1.setPreferredSize(new Dimension(500, 379));
this.setSize(516, 438);
```

and:

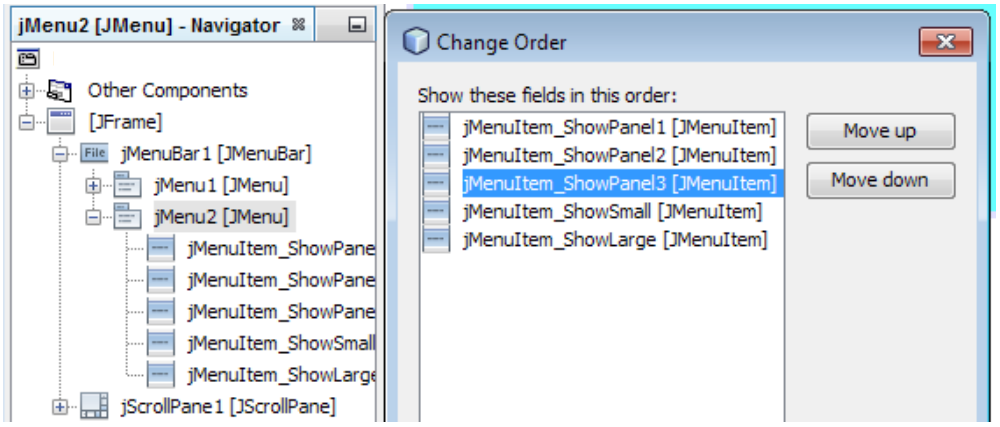
```
this.jLayeredPane1.setPreferredSize(new Dimension(700, 379));
this.setSize(716, 438);
```

then add the 2 lines of code from the Show Small event to the moreInit() method, to make the sure the application starts out at the Small size.

```
private void moreInit() {
  this.jLayeredPane1.setPreferredSize(new Dimension(500,379));
  this.setSize(516, 438);
  this.jPanel1.setVisible(true);
  this.jPanel2.setVisible(false);
 }
```

Run the application and all is better again, the window, panels and labels work as they did before, just at different sizes. You now have more room on each panel to add more Controls.

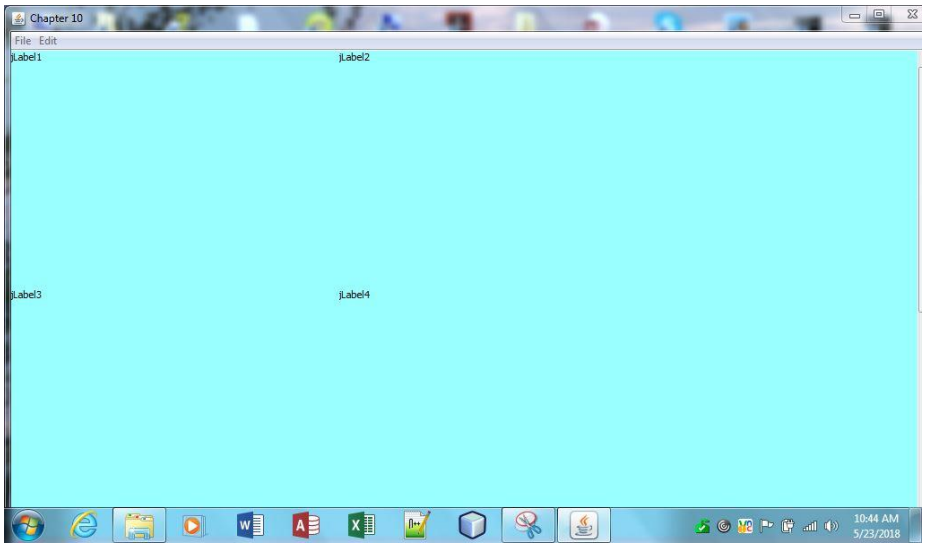
10. Add a third panel and labels to the project. See chapter 2.5 for a hint on how to add more panels. BTW, if you add another panel, remember to add code to make it visible(true) or (false) in your moreInit() method, and any time you change panel visibility in any Menu Item, etc. Because all panels overlay, you must control which panel(s) are to be visible and all other panels that are not to be visible. Another thing, if you right click on jMenu2, you can choose Change Order… and move the Action Event for Show Panel 3 so it appears below Show Panel 2.



Ch. 10 Adjusting for different screen resolutions

There are situations where the user has a screen that is smaller than what your application needs at launch. Luckily, there is a way to test the size of the screen your application is running on, and a way to adjust the Form Size at runtime. Obviously, if the application starts smaller, not all the controls will show. In this case, you want the scroll bars to appear so the user can navigate the entire application.

1. Copy the Project named Chapter3, title and rename it Chapter10. Open the JFrame for editing.
2. Select the JFrame. Select code in the Properties window. Change the Designer Size from 400, 300 to 1280, 1024. The JFrame becomes larger in the Design window. Select jScrollPane and make sure the Layout H & V Sizes are 'Default'. Click on jLayeredPane1 in Navigator and look at preferredSize in the Properties window. The number there will have changed from 400, 279 to 1280, 1003.
3. Select jPanel1. In the Properties window, change the preferredSize to 1280. 1003. Select jPanel2 and do the same.
4. Change to the Source view. Copy the code from your Show Small Event to the beginning of the moreInit() event. Change the Dimension of the jLayeredPane1 to 1280, 1003 and the size of the frame to 'this.setSize(1296, 1062);'. Run the project on a large enough monitor and you'll see a much taller window, with the labels in their original places. If you Clean and Build this on a normal size display, then run the jar file on a netbook with a 1024x600 screen and run it, you'll see this:



It's so large, that the vertical scroll bar barely appears. You have to shorten the application by dragging down the top edge then move the entire window up and sideways far enough to show enough of the whole thing on the screen. If you don't have a smaller monitor/screen, see if your regular monitor can be set to 1024x600. When you run the jar file at the lower resolution, you should see the same effect. What you need is the app to measure the display the it's running on, then calculate the difference of the application size and adjust the JFrame size.

5. Let's add some code to try to fix this. This gets done in the moreInit method and Show Large actionPerformed event. Change the moreInit method code to what is shown below:

```
  private void moreInit() {
    Dimension totalScreenSize = Toolkit.getDefaultToolkit().getScreenSize();
    screenInsetsSize = Toolkit.getDefaultToolkit().getScreenInsets(getGraphicsConfiguration());
    availableWidth = (int) (totalScreenSize.getWidth() - screenInsetsSize.left - screenInsetsSize.right);
    availableHeight = (int) (totalScreenSize.getHeight() - screenInsetsSize.top - screenInsetsSize.bottom);
    widthOfVerticalScrollbar = (int) jScrollPane1.getVerticalScrollBar().getPreferredSize().getWidth();
//    System.out.println("width of vertical scrollbar = " + widthOfVerticalScrollbar);
    heightOfHorizontalScrollbar = (int) jScrollPane1.getHorizontalScrollBar().getPreferredSize().getHeight();
//    System.out.println("height of horizontal scrollbar = " + heightOfHorizontalScrollbar);
    this.jPanel1.setVisible(true);
    this.jPanel2.setVisible(false);
    // code below adjusts for a smaller display.
//    System.out.println("task bar height is " + (screenInsetsSize.bottom - screenInsetsSize.top));
//    System.out.println("avail width is " + availableWidth + "  avail height is " + availableHeight);
//    System.out.println("Orig Form Size height = 1062");
//    widthDiff = (availableWidth - 1296);
//    heightDiff = (availableHeight - 1062);
//    System.out.println("widthDiff " + widthDiff + ", heightDiff "+ heightDiff);
    this.jMenuItem_ShowLarge.doClick(); // start the Form to fit the screen
  } // close moreInit method
```

Ignore the errors for now.

This time, you should include the green comment lines. They're handy if there's problems.

6. You need to add some new variables to the end of the Variables declaration at the bottom of the Source window after the ' // End of variables declaration' line:

```
  private int availableHeight = 0;
  private int availableWidth = 0;
  private int heightDiff = 0;
  private int widthDiff = 0;
  private int widthOfVerticalScrollbar = 0;
  private int heightOfHorizontalScrollbar = 0;
  private Insets screenInsetsSize;
  private String thisTitle = "";
  private boolean screenWidthOK;
  private boolean screenHeightOK;
  private int correctedWidth;
  private int correctedHeight;
```

7. Then you'll need to fix imports. Add an import for Point and StrictMath. Imports should look like:

```
import java.awt.Dimension;
import java.awt.Insets;
import java.awt.Toolkit;
import java.awt.Point;
import static java.lang.StrictMath.abs;
```

Go back to the Design tab, select each jPanel and move the labels on both panels to the new corner positions, check the anchors for Left, Top as they may change when they are moved. At this point, you've added code to the moreInit() method to get the size and availability of the display your project is running on. As it builds, it'll call the ShowLargeActionPerformed action event to make the project Form and JFrame fit that availability. The ShowLarge and ShowSmall action events will contain the code to make the project Form and JFrame resize accordingly, and you'll use hard coded sizes you get from the JFrame Form Size property and the preferredSize properties of jLayeredPane1.

8. Go to the Source window and let's make a lot of changes to the ShowLargeActionPerformed event. Edit the code so it looks like this:

```
  private void jMenuItem_ShowLargeActionPerformed(java.awt.event.ActionEvent evt) {
    this.jLayeredPane1.setPreferredSize(new Dimension(1280, 1003));
    screenWidthOK =! ((1296 + widthOfVerticalScrollbar) >= availableWidth);
    screenHeightOK =! ((1062 + heightOfHorizontalScrollbar) >= availableHeight);
    System.out.println("large width " + screenWidthOK +", large height " + screenHeightOK);
    if(screenWidthOK && screenHeightOK) {
      System.out.println("large width and height OK");
      correctedWidth = 1296; // original large size
      correctedHeight = 1062; // original large size
    }else {
      System.out.println("large width and height not OK");
      if(!screenWidthOK) {
        correctedWidth = availableWidth;
      } else {
        correctedWidth = 1296 + widthOfVerticalScrollbar; // because height must not be OK
      } // close if(!screenWidthOK)
      if(!screenHeightOK) {
        correctedHeight = availableHeight;
      } else {
        correctedHeight = 1062 +heightOfHorizontalScrollbar;
      } //close if(!screenHeightOK)
    } // close  if(screenWidthOK && screenHeightOK)
    System.out.println("original large form width " + 1296 +", original large form height " + 1062);
    System.out.println("corrected large width " + correctedWidth +", corrected large height " +
correctedHeight);
    this.setSize(correctedWidth, correctedHeight);
    widthDiff = (availableWidth - 1296);
    heightDiff = (availableHeight - 1062);
    System.out.println("Large widthDiff " + widthDiff + ", Large heightDiff "+ heightDiff);
    thisTitle = ("Form width is " + abs(widthDiff) + ((widthDiff > -1) ? " less" :" more") +" than avail width,
height is " + abs(heightDiff) + ((heightDiff > -1 ) ? " less" : " more") + "  than avail height.");
    this.setTitle(thisTitle);
// set the Form starting location
    int x,y;
    x=(int) ((availableWidth-this.getWidth())/2);
    y=(int)((availableHeight-this.getHeight())/2);
    if(this.getWidth()>=availableWidth) {x=0;}  // if too wide, start at left
    if(this.getHeight()>=availableHeight) {y=0;}  // if too, start at top
    this.setLocation (new Point(x + screenInsetsSize.left, y + screenInsetsSize.top));  // avoid task bar
    this.setTitle(thisTitle +  " Starting location is " + x + ", " + y);
    this.jLayeredPane1.setLocation(0, 0);
    this.jScrollPane1.getVerticalScrollBar().setValue(0);
    this.jScrollPane1.getHorizontalScrollBar().setValue(0);
  }
```

9. Copy the code in the ShowLargeActionPerformed event and paste it in the ShowSmallActionPerformed event, and edit the sizes so it looks like this:

```
  private void jMenuItem_ShowSmallActionPerformed(java.awt.event.ActionEvent evt) {
    this.jLayeredPane1.setPreferredSize(new Dimension(1280, 1003));
    // make the Form 600 less wide by 400 less high
    screenWidthOK =! (((1296 - 600) + widthOfVerticalScrollbar) >= availableWidth); // small size
    screenHeightOK =! (((1062 - 400) + heightOfHorizontalScrollbar) >= availableHeight); // small size
    System.out.println("small width " + screenWidthOK +", small height " + screenHeightOK);
    if(screenWidthOK && screenHeightOK) {
      System.out.println("small width and height OK");
      correctedWidth = (1296 - 600); // small size
      correctedHeight = (1062 - 400); // small size
    }else {
      System.out.println("small width and height not OK");
      if(!screenWidthOK) {
        correctedWidth = availableWidth;
      } else {
        correctedWidth = (1296 - 600) + widthOfVerticalScrollbar; // because height must not be OK
      } // close if(!screenWidthOK)
      if(!screenHeightOK) {
        correctedHeight = availableHeight;
      } else {
        correctedHeight = (1062 - 400) +heightOfHorizontalScrollbar;
      } //close if(!screenHeightOK)
    } // close  if(screenWidthOK && screenHeightOK)
    System.out.println("small form width " + (1296 - 600) +", small form height " + (1062 - 400));
    System.out.println("corrected small width " + correctedWidth +", corrected small height " +
      correctedHeight);
    this.setSize(correctedWidth, correctedHeight);
    widthDiff = (availableWidth - (1296 - 600));
    heightDiff = (availableHeight - (1062 - 400));
    System.out.println("Small widthDiff " + widthDiff + ", Small heightDiff "+ heightDiff);
    thisTitle = ("Form width is " + abs(widthDiff) + ((widthDiff > -1) ? " less" :" more") +" than avail width,
height is " + abs(heightDiff) + ((heightDiff > -1 ) ? " less" : " more") + " than avail height.");
    this.setTitle(thisTitle);
// set the Form starting location
    int x,y;
    x=(int) ((availableWidth-this.getWidth())/2);
    y=(int)((availableHeight-this.getHeight())/2);
    if(this.getWidth()>=availableWidth) {x=0;}  // if too wide, start at left
    if(this.getHeight()>=availableHeight) {y=0;}  // if too, start at top
    this.setLocation (new Point(x + screenInsetsSize.left, y + screenInsetsSize.top));  // avoid task bar
    this.setTitle(thisTitle +  " Starting location is " + x + ", " + y);
    this.jLayeredPane1.setLocation(0, 0);
    this.jScrollPane1.getVerticalScrollBar().setValue(0);
    this.jScrollPane1.getHorizontalScrollBar().setValue(0);
  }
```

Edit the ShowPanel action events for both panels like so:

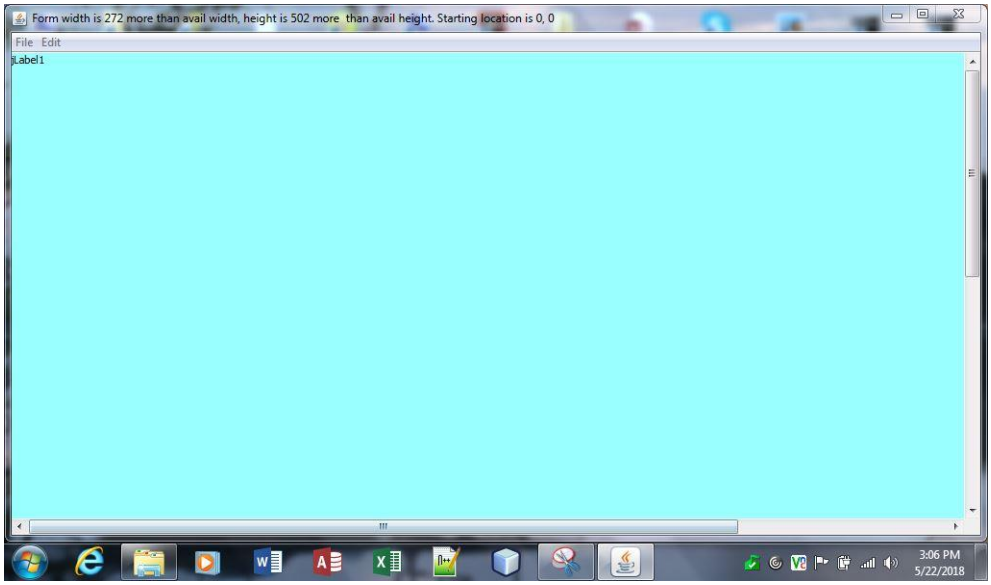
```
private void jMenuItem_ShowPanel1ActionPerformed(java.awt.event.ActionEvent evt) {
  this.jLayeredPane1.setLocation(0, 0);
  this.jScrollPane1.getVerticalScrollBar().setValue(0);
  this.jScrollPane1.getHorizontalScrollBar().setValue(0);
  this.jPanel1.setVisible(true);
  this.jPanel2.setVisible(false);
 }
```

and:

```
private void jMenuItem_ShowPanel2ActionPerformed(java.awt.event.ActionEvent evt) {
  this.jLayeredPane1.setLocation(0, 0);
  this.jScrollPane1.getVerticalScrollBar().setValue(0);
  this.jScrollPane1.getHorizontalScrollBar().setValue(0);
  this.jPanel1.setVisible(false);
  this.jPanel2.setVisible(true);
 }
```

Run the project and you'll get a full window (JFrame) within the available space on your monitor. Use the mouse to shrink the window. Scroll down or across so the other corner labels appear, then use the menu item to show the other panel. The view changes so the panel is shown with its top, left corner at the top, left of the window and the scroll bars reposition to their default positions. Show Small reduces the window and Show Large sets the window back to its original size. Using any menu item will reposition the panel to its top, left.

Clean and Build the project, then copy and run on the other computer or change resolution on your monitor. There you'll see the scroll bars like the screen shot below.



Your application has the same view and function as on the normal size monitor, with the addition of scroll bars so you can pan the view to see the other labels. Use the Show Small and Show Large Menu Items, and the view acts the same on both monitors. The Show Small and Large events act the same as they did in the project from Chapter 3, preserving the ability to view all the controls, but changing the window width and height while maintaining the window within the available screen space.

Because there is a lot happening, I'll explain the code changes for you.

In the moreInit() method, the code gets the total screen (display) size.

Next it gets the values of the Insets (the task bars).

Next there are two lines to calculate the available space on the screen by subtracting any/all of the insets from the width and height of total screen size. The results are the values in the variables availableWidth and availableHeight.

The code then gets the width of the vertical and the height of the horizontal scroll bars in jScrollPane1.

Two more lines to set the default view, show panel 1 and not panel 2.

There are some comments that can be useful if you have problems or just want to see the results of the code above.

The last thing to do in this method, is to call the jMenuItem_ShowLarge method with a 'doClick()'.

In the ShowLarge method, the code takes what it found for the available screen area and sizes the JFrame to fit.

First, jLayeredPane1 is sized to its preferredSize(1280, 1003).

Two Boolean variables are set as to whether the Form Size is larger than the available screen width and height. A comment line will output the result if you want it.

Next comes the heavy lifting. The two Booleans are checked in an if statement. If both are true, the variables correctedWidth and correctedHeight are set to the full size of the Form. These two variables will be used later.

If either of the two are false, both the width and height are checked. If either are false, the corrected variable is set to the available size, but if either fits, it's correctedWidth and/or correctedHeight are set to the full size plus the size of the scroll bar that it will have (because the other dimension will not fit or the logic would never get to this point).

Two lines of comments follow to output the results of the if block.

Finally, the Form is set to the correctedWidth and correctedHeight sizes. The Form will now fit within the available space on the screen.
Two integer variables are set, to be used for the title of the Form to show you what the Form Size is.
The title string is made with the tenary operator to use positive or negative numbers so it reads well.
The title string is applied.
The last few lines of code determine where the Form should appear of the display, centering it as much as possible. If the Form is too wide or tall, the location starts at zero, the far left or full top.
Then the Form title is appended to show the starting location.
Last, jLayeredPane1 and the scroll bars are set to default positions.

The Show panel events get a couple of new lines that set the panel to display at 0,0 in the application window, which is the upper left corner, and reset both scroll bars to 0, which is their original locations. If you drag the window, using the menu item to change panels will not change the window size, but makes the panel view at it's top, left and ready to scroll.
The Show Small event is basically a copy of the Show Large event, with changes to the size numbers only. The Show Small will reduce the JFrame width by 600 and the height by 400. Show Large resets the Form to its original size. Both will set the Form size within the available screen space.
This project is done with hard coded sizes to show you the changes as the code moves along. The next chapter will change this to variables that can be set by you or set by the existing values of the design.
This works on all my Windows7 PCs, since that's what I built this project on.

Ch. 11 Adjusting for different screen resolutions part 2

The last chapter shows how to adjust to different screens using hard coded values. In this chapter you'll change those hard coded values to variables, then the code will get the values for those variables. Show Panelx methods will also recalculate their sizes because you'll reduce the appearance of jPanel1 using the location and size of an added label with no text (so it doesn't show on the app). The app still has the ability to scroll when needed. All this is so you can design on a large editor, add controls, etc. to it later, and only show what's needed without problems simply by changing the position of this added label. This starts with the project from chapter 10, and a lot of code gets changed and a new method is made.

1. Copy chapter 10, rename, refactor and title it chapter 11. Add new variables to the end of the variables declarations:

```
private int formSizeWidth;
private int formSizeHeight;
private int jLP1PrefWidth;
private int jLP1PrefHeight;
private int cornerLabelX;
private int cornerLabelY;
```

add the following lines to beginning of the moreInit() method:

```
formSizeWidth =this.getWidth();
formSizeHeight = this.getHeight();
System.out.println("Form width, height " + formSizeWidth + ", " + formSizeHeight);
jLP1PrefWidth = (int) this.jLayeredPane1.getPreferredSize().getWidth();
jLP1PrefHeight = (int) this.jLayeredPane1.getPreferredSize().getHeight();
System.out.println("jLP1 width, height " + jLP1PrefWidth + ", " + jLP1PrefHeight);
```

In the jMenuItem_ShowSmallActionPerformed and jMenuItem_ShowLargeActionPerformed methods replace all '1296' with formSizeWidth, all '1062' with formSizeHeight. Run the project again, it's still the same. Replace all jLP1 preferredSize Dimensions with jLP1PrefWidth, jLP1PrefHeight. Run the project, Show Small has a different size than expected. Clean, build and run on netbook, same there. Change the variable declarations for private int formSizeWidth; to private int formSizeWidth = 1296; and private int formSizeHeight; to private int formSizeHeight = 1062; and private int jLP1PrefWidth; to private int jLP1PrefWidth = 1280; and private int jLP1PrefHeight; to private int jLP1PrefHeight = 1003;. These are the original sizes of the full Form.

Comment out the lines in moreInit() method that get these sizes.

```
    formSizeWidth =this.getWidth();
    formSizeHeight = this.getHeight();
    jLP1PrefWidth = (int) this.jLayeredPane1.getPreferredSize().getWidth();
    jLP1PrefHeight = (int) this.jLayeredPane1.getPreferredSize().getHeight();
```

Run the project, clean and build then run on netbook. Everything should be correct again. Change the Form and jLP1 sizes in the variable declarations and see what happens. Use the sizes from previous projects with JFrames of 400, 300 (416, 338 and 400, 279). You can stretch out the window to see the corner labels on jPanel1 and jPanel2 because those panels are still the original size of 1280, 1003. You just changed all the hard coded values to variables, had the app find the sizes and checked that Show Small isn't what you expected, because Generate pack() gives the wrong FormSize Height! If you left in the comments, the output window will show this. Let's fix this. Set the variable declarations back to have no assigned sizes. Keep those lines in moreInit() commented out.

2. Next, add the new method. This finds the sizes needed for display based on the location and size of labels on the jPanel. You'll add a new label on the jPanel1, and use the existing jLabel8 on jPanel2. Put this new method after the moreInit() method:

```
  private void findSizes() {
    int topInset = this.getInsets().top;
    int bottomInset = this.getInsets().bottom;
    int leftInset = this.getInsets().left;
    int rightInset = this.getInsets().right;
    int menuBarHeight = this.jMenuBar1.getHeight();
    jLP1PrefWidth  = cornerLabelX;
    jLP1PrefHeight = cornerLabelY;
    formSizeWidth = cornerLabelX + leftInset + rightInset;
    formSizeHeight = cornerLabelY + menuBarHeight + topInset + bottomInset;
    System.out.println("Insets: Top " + topInset + ", Bottom " +bottomInset + ", Left " + leftInset + ", Right "
+ rightInset);
    System.out.println("Corner Label Right Edge " + cornerLabelX + ", Bottom Edge " + cornerLabelY);
    System.out.println("Form Size Width " + formSizeWidth + ", Height " + formSizeHeight);
  }
```

As usual, I'll explain all this later.

3. Add the following to the constructor after the moreInit() line:

```
    findSizes();
```

4. Now let's do some graphics changes to jPanel1. Select jPanel1 and add some labels and a text field, a toggle button somewhere near the top left of the jPanel. Position isn't critical here, you're just making a mock-up of an app's panel. Add a text area below those controls. Set the Horizontal Size of the scrollpane for the text area to be 1000 using its Properties, Layout. Vertical Size can be 'Default'. Both resizable check box should be un-checked. Add another label just below the lower right of the text area. Change its name to 'jLabel_CornerPnl1'. Set the Horizontal Size and Vertical Size to 1 more than the default in Properties, Layout. The default sizes are the Preferred Size in Properties, in my case it was 40, 14. Delete the text ('jLabelx') for the label. Now nothing will show for that label when the app displays jPanel1. Select jPanel2 and change the variable name of jLabel8 to jLabel_CornerPnl2. Make sure the anchors for both labels are set to Left, Top.
5. In the jMenuItem_ShowPanel1ActionPerformed make the following the first lines of the event:

```
cornerLabelX = (int) this.jLabel_CornerPnl1.getLocation().getX() + this.jLabel_CornerPnl1.getWidth();
cornerLabelY = (int) this.jLabel_CornerPnl1.getLocation().getY() + this.jLabel_CornerPnl1.getHeight();
this.findSizes();
```

   put the same line at the beginning of jMenuItem_ShowPanel2ActionPerformed but edit it to

```
cornerLabelX = (int) this.jLabel_CornerPnl2.getLocation().getX() + this.jLabel_CornerPnl2.getWidth();
cornerLabelY = (int) this.jLabel_CornerPnl2.getLocation().getY() + this.jLabel_CornerPnl2.getHeight();
this.findSizes();
```

   Any last errors should clear now.

   Change the last line in moreInit() to

```
this.jMenuItem_ShowPanel1.doClick();
```

   Add the following as the last line to both action events:

```
this.jMenuItem_ShowLarge.doClick();
```

Did it work for you? If it did, the app launches with a smaller jPanel1 but all the new controls show. If your monitor/resolution is large enough, there'll be no scrollbars until you drag the window smaller. If you can drag it wider, the labels in the corners will show. You could just delete them. If you change the position of jLabel_CornerPnl1, the starting view changes. Use the Menu Item to Show Panel 2 and the window gets much larger, to the available screen size or jPanel2's original size. Shrink it and scroll bars appear like you'd expect from previous chapter projects. Show Small works here, too, but probably not too well when jPanel1 shows, because it's smaller already. In all cases, the size details are shown on the window title bar.

Now for the explanation.

Changes are made to all methods, action events and the constructor. A new method is created. jPanel1 has its graphics edited and a new jLabel added.

The constructor starts by calling initComponents() as usual, and moreInit() as has been done before. The moreInit() method works as explained in Chapter 10. But at the end, the call for the action event to Show Large is replaced with a call to the Show Panel 1 action event.

jMenuItem_ShowPanel1ActionPerformed as well as jMenuItem_ShowPanel2ActionPerformed start by getting the lower, right location of the jLabel (cornerLabelX, cornerLabelY) used to set the starting displayable size of the jPanel.

The new method, findSizes() runs next.

The findSizes() method gets all the necessary sizes for the requested jPanel including Form insets, Menu Bar height. The jLayeredPane1 is set to cornerLabelX and cornerLabelY. The Form sizes add the insets and menu bar size to cornerLabelX and cornerLabelY. Now the app has the reduced size of jPanelx that you want displayed.

When findSizes() has run, the Show Panelx action event finishes with setting the starting location for jLayeredPane1 and the scroll bars to their defaults of 0, 0.

Next, ‘this.jMenuItem_ShowLarge.doClick()’ is called to make the starting Form window size for jPanelx like it does in the Chapter10 project using the sizes calculated in findSizes(). This action event positions the Form near center, or at the left/top edge depending on the available screen size found in moreInit(). It also sets the starting location for jLayeredPane1 and the scroll bars to their defaults of 0, 0.

Once the window is displayed, the user can use a Menu Item to change the view from jPanel1 to jPanel2, make the window smaller (jPanel1 is already smaller by choice). The visible window sizing info is shown in the window title bar at the top. Any change with a Menu Item will reposition the panel to the upper left of the window and reset the scroll bars, if any.

This project calculates the sizes and positions base on the design you made, not hard coded sizes. As a result, this project will display correctly on any platform. I checked it on Win XP, Win 10, Raspbian Stretch and XUbuntu using different display sizes and it ran as expected on all.

Ch. 12 Remembering the window size and position for each User

So far, you've learned how to use a jScrollPane to make sure your app always has every control available no matter what size the user has dragged the window. Also, you've made sure a large app runs on a smaller display, every control can be accessed. Next to do, saving and displaying the last known size and position of the window for your application. Users tend to like an app starting at the same size and position where they last used it. This project will use a file created for each user, located in a folder named for the user, under the app's parent folder. If the user is new, the defaults will be used to size and position the Form on the displays' available screen. For this project, we'll assume only jPanel1 as the last/first panel displayed. The title Bar will not necessarily have the correct size and position info when the app launches, so you can delete the code that displays that if it bugs you. After this project, you should be able to modify the code to save the user preference for which panel to start with, save each panels' size and position independently and make additional menu items to restore a panel to the default.

1. Copy the Chapter11 project, title and rename it Chapter12.
2. Select the JFrame, use the Source tab and add the following to the end of the variables declarations. Accept the imports when asked to.

```
private String userNamePath;
final static String SEP = FileSystems.getDefault().getSeparator();
Dimension thisSize;
Point thisLocation;
```

Add the following to the constructor, after the line findSizes();. Accept the imports here, also.

```
readUserInfo();
  this.addWindowListener( new WindowAdapter() {
      @Override
    public void windowClosing(WindowEvent we) {
      try {
        writeUserInfo();
      } catch(Exception e) {
        // do nothing here
      } // close try-catch
      System.exit(0);
    } // close WindowEvent
  }); // close addWindowListener
```

Add the following to the constructor after the line initComponents();:

```
   getLaunchDirectoryPath();
```

Ignore the errors for now.

Add the next three new methods after moreInit() method. Then right click and Fix Imports.

```
  void getLaunchDirectoryPath() {
    // get directory/path the jar file launched from, use for reading and writing config files
    Class className = this.getClass();
    File currentJavaJarFile = new
File(className.getProtectionDomain().getCodeSource().getLocation().getPath());
    String currentJavaJarFilePath = currentJavaJarFile.getAbsolutePath();
    jarPath = currentJavaJarFilePath.replace(currentJavaJarFile.getName(), "");
    try {
      jarPath = URLDecoder.decode(jarPath, "UTF-8");
    } catch (UnsupportedEncodingException ex) {
      Logger.getLogger(Chapter12_JFrame.class.getName()).log(Level.SEVERE, null, ex);
    }
    System.out.println("jarPath: " + jarPath);  // debugging only
  } // close getLaunchDirectoryPath method

  void readUserInfo() {
    Path userpath = Paths.get("");
    String getUser = System.getProperty("user.name");
    userNamePath = jarPath + getUser;
    File dir =  new File(userNamePath);
    this.jTextArea1.append(jarPath + getUser + this.SEP + "UserFormPrefs.sid");
    if(dir.exists()){
      String userDisplayPrefsFile= userNamePath + this.SEP + "UserFormPrefs.sid";
      try {
        FileInputStream fis = new FileInputStream(userDisplayPrefsFile);
        ObjectInputStream ois = new ObjectInputStream(fis);
        try {
          thisSize = (Dimension) ois.readObject();
          this.setSize(thisSize);
          thisLocation = (Point) ois.readObject();
          this.setLocation(thisLocation);
        } catch (ClassNotFoundException ex) {
          Logger.getLogger(Chapter12_JFrame.class.getName()).log(Level.SEVERE, null, ex);
        } // close try-catch FileInputStream fis
      } catch (IOException ex) {
        Logger.getLogger(Chapter12_JFrame.class.getName()).log(Level.SEVERE, null, ex);
      } // close try-catch ois.readObject()
    } // close if(dir.exists())
  } // close readUserInfo() method
```

```
void writeUserInfo() {
  try {
    Files.createDirectories(Paths.get(userNamePath));
  } catch (IOException ex) {
    Logger.getLogger(Chapter12_JFrame.class.getName()).log(Level.SEVERE, null, ex);
  } // close try-catch createDirectories
  try {
    FileOutputStream fos = new FileOutputStream(userNamePath + this.SEP + "UserFormPrefs.sid");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(this.getSize());
    oos.writeObject(this.getLocation());
    oos.flush();
    oos.close();
  }catch(IOException ex) {
  } // close try-catch write
} // close writeUserInfo() method
```

The position and size will be retrieved from a serialized file that's created when the application closes. It will save the last position and size prior to shutting down. If the file doesn't exist when the app launches, the default sizes are used to fit the available screen instead.

In a nutshell, when the app starts for the first time (or when it can't find the file with size and location settings saved previously), the available display size is found and the size of the JFrame that's determined as in Chapter 11 is used at start. The JFrame is re-sized if it's larger that the display can show, and the center position is calculated. If the app has run before, and the UserFormPrefs.sid file is found, the JFrame is sized and placed according to the saved settings. If you use any Menu Item, the Form is reset to the default size and position for that jPanel. BTW, the file extension 'sid' stands for Serial Information Data.

Now for the explanation:

The app starts and calls the getLaunchDirectoryPath() method to find the parent folder.

The moreInit method determines the available size on the display, same as Chapter11 does.

The jPanel1 size and positioned same as Chapter11 does.

The method readUserInfo() runs.

The file separator character(s), username and the filename are found from the OS.

A String for the parent folder and user is made. The app checks if that path/user folder exists.

If the folder and/or the file UserFormPrefs.sid is not found, the default size and position is used.

If the file UserFormPrefs.sid is found, the Form size/position can be set by the values found in the file.

The Form size and position is read into the app in the order it was written to the file.

You can assume that the Form fits the screen because the settings were saved the last time it ran, so, hopefully the user had made it a useable size at a usable location.
When the readUserInfo method is finished, a Window Listener is created to be used as the app closes.
The app is now running.
When you click the big red ‘X’ in the upper right corner of the window to close the app, the Window Listener calls the writeUserInfo method to get and save the current size and position of the Form.
The writeUserInfo method checks that a path to where the file should reside exists, if not, it creates the folder in the correct path.
A FileOutputStream and a ObjectOutputStream are created to make the file and write it.
The ObjectOutputStream gets two Objects, the Dimension of the Form and the Point the Form starts on the screen. These objects are written to the file in serialized fashion, since they are Objects.
When the writeUserInfo method completes, the Window Listener makes the application shut down with the call to System.exit(0).
You can make a menu item under File to exit. Make an Action Event for it and add this code:

```
this.dispatchEvent(new WindowEvent(this, WindowEvent.WINDOW_CLOSING));
```

Make this the same way you added other menu items.

The last two projects you made work well on different computers for different users. They are also good candidates for basic GUI forms for any new project. I have made several copies of the Chapter12 project using different Designer Sizes so I have smaller, average and larger GUI’s to start from. Chapter12 can be copied and renamed to make these additional projects. Here’s a good way to avoid sizing and typo errors, preserving the functionality of all the menu items.

1. Copy the Chapter12 project . Change the name. For a Designer size of 1000 by 1000, I suggest calling it ‘Designer1000x1000’. Refactor the JFrame and package and rename them the same. Change the title of the JFrame .
2. Delete the two jPanels.
3. Select jLayeredPane1 and change the preferredSize to 1000, 979 (21 less high for the Menu Bar).
4. Select jScrollPane1. The preferredSize to should have changed to 1000, 979. Check that the Layout Horizontal and Vertical Sizes are ‘default.

5. Select the JFrame. Set the Designer Size to 1000, 1000. Check the JFrame preferredSize changed to 1000, 1000 also.
6. Select jLayeredPane1 and add two panels. The should be named jPanel1 and jPanel2. Change the jPanels preferredSize in Properties to 1000, 979. Check the Resizable boxes here also.
7. Select jPanel1 and add a label somewhere inside the designer border, rename it 'jLabel_CornerPnl1'. Set it's anchors to Left and Top. Make it's Horizontal and Vertical sizes 1 more than the default preferred sizes. Leave the Resizable boxes un-checked. Now you can delete the text in the label. You have to rename the label to jLabel_CornerPnl1 because the code that got copied has references to jLabel_CornerPnl1.
8. Select jPanel2 and add a label to the lower right edge of the designer, rename it jLabel_CornerPnl2, set the anchors for Left and Top. Again, you have to rename the label because the code that got copied has references to jLabel_CornerPnl2.
9. Comment out the line that appends text to jTextArea1 in the readUserInfo() method.
10. If you want, change the first line in the jMenuItem_ShowSmallActionPerformed action event and add -600 and -400 to the Dimensions jLP1PrefWidth and jLP1PrefHeight respectively. This will remove scroll bars when a panel is made to Show Small.
11. The project should run as expected, with the different Designer, Frame and panes/panel sizes.

Now you have a basic GUI project named for the Designer Size it has. All the code from the Chapter12 project is there, no need to re-type or copy-paste into a new project. The size and location of the Form window will get saved in the jar file parent folder, as it did in the Chapter12 project. You can copy this for any new project that you want this Designer Size. Just copy this project and rename, refactor and retitle it. Move or replace the two jLabels as you require. I've made a few, because sometimes it's just easier to design on a smaller editor when the app doesn't require too many controls like labels, etc.

You made it to the end

What have you learned with this tutorial? Making a ‘base’ form that sets up your applications’ window to use scroll bars when needed, but not when the application starts, was covered in Chapter 1. At the end of that chapter, we went over how to copy the base form to start a new project. In Chapter 2, you added multiple panels that individually cover the full window. With Chapter 3, you made your application change sizes with the click of a button, or in this case, a click on a menu item. You also added labels to the corners and made the labels stay where you originally put them when the window is enlarged. Chapter 4 showed you how to construct a ‘hidden’ panel that’s added to the right edge of a panel that has a toggle button to display it. Chapter 5 introduced you to a trick that allows a ‘hidden’ panel to be added to an existing project. In Chapter 6, you combined it all in one application. Chapter 7 expanded on the placement of controls and how to control their relation to other controls. Chapter 8 gives a reason to use Absolute Layout to get a scrolling Column Header. Because jPanels are like garages, Chapter 9 showed you how to make the whole project larger without starting over from scratch. Up to this point, the projects have been mainly for apps you run on your computer, the one you make it on. But, before you know it, you need to make your app run on another pc, OS, or for another user. Chapter 10 goes into detail on how to have your code size up the display it’s running on, and adjust the size of the app if there’s not enough room to display all of the window. Chapter 11 takes Chapter10 further and uses variables in place of hard coded size and position numbers, and have the app to make the GUI fit the display by getting the necessary info from the JVM and the OS. Chapter12 is the same as Chapter11, but also has the application Form window start up in the place and at the size your user last used it. The projects in this guide have you building applications that act like web browsers or like other applications do. What you put in the application, and what it accomplishes is up to you. You now have the basics of the window your application will display as. The Show Large(Small) Menu Items used in these projects is probably useless in the real world, but one of these Menu Items could be adapted as a one-click way to put the window back to its original size. All of the projects you built are simple by design, so you can adapt them to whatever your project demands. My email is on the Table of Contents page, drop me a line if you want.