Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

Swing, Understanding getContentPane() and other JFrame Layers

Java Programming, Lecture Notes # 87, Revised 07/01/98.

Preface

Students in Prof. Baldwin's **Intermediate Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

Introduction

Previous lessons involving Swing told you that when you add a component to a **JFrame** object, unlike with the **AWT**, you must use statements similar to the following:

```
myJFrameObject.getContentPane().add(myChildComponent);
```

As you have probably suspected, there is a lot more to it than just inserting **getContentPane()** between the object reference and the **add()** method.

The purpose of this lesson is to help you understand why you need to use **getContentPane()** or some similar method call to add components to a **JFrame** object, remove components from a **JFrame** object, or set the layout for a **JFrame** object.

Note that this also applies to container objects of type **JInternalFrame** and **JDialog** as well.

General Discussion

When programming with the **AWT**, you can place a **Panel** object in a **Frame** object and place other components on the **Panel** object if you wish, or you can simply place other components directly on the viewable area of the **Frame** object.

The *viewable* area of Frame or a **JFrame** is the *bounds* minus the *insets*. Insets are used to account for the space covered by the borders and the values of the insets on all four sides are available by invoking the method named **getInsets()** on the object.

You cannot place components directly on the viewable area of a **JFrame** object.

An object of type **JRootPane** is automatically placed in the **JFrame** completely covering the viewable area of the **JFrame** object.

One way or another, if you want to place components in the **JFrame**, you must deal with the **JRootPane** object.

One way to deal with it, and this is the way that you will find recommended in many current discussions on the subject, is simply ignore the technical details and insert a call to the **getContentPane()** method between the reference to the **JFrame** and the **add()** method.

That is not my approach. My approach is to first understand why you need to do that, an in those cases where that is the appropriate thing to do, go ahead and do it.

In those cases where invoking **getContentPane()** is not the most appropriate thing to do, do the thing that is most appropriate.

Note: You will find much discussion in this lesson about some components being on top of other components. These discussions apply only to Swing lightweight components. Remember that heavyweight components are always on top of lightweight (Swing) components regardless of what the text in this lesson may say.

Synopsis

JRootPane is a container.

The instance of **JRootPane** that is automatically placed in a **JFrame** object contains at least two other objects.

1. An object of type **JLayeredPane** that we will refer to simply as the ***layeredPane***.
2. An object of type **JPanel** that we will refer to simply as the ***glassPane***.

(There is also an optional menu bar object that comes in here somewhere, but since it is not essential for understanding what is going on at this level, I have elected to defer the discussion until the discussion on Swing menus.)

The **layeredPane** and the **glassPane** also fill the viewable area of the **JFrame** object. The **glassPane** is on top of the **layeredPane**, and as the name would imply, it is normally transparent.

Components on the **layeredPane** are visible through the **glassPane**, and mouse events are capable of penetrating the **glassPane** and reaching components on the **layeredPane** as long as the **glassPane** is transparent.

The **layeredPane** contains another component of type **JPanel** that we will refer to as the ***contentPane***. This is the same pane that we are dealing with when we invoke the **getContentPane()** method on the **JFrame** object.

The **layeredPane** has a very interesting behavior.

In the same sense that we can add components to the **contentPane**, we can also add components to the **layeredPane**.

When we add components to the **contentPane**, they are all added at the same layer and components added first are painted on top of components added later.

However, when we add components to the **layeredPane**, we can specify the layer number that we want the component to be drawn on.

Components drawn on layers with high (algebraically signed) numbers are painted on top of components on layers with smaller (algebraically signed) numbers. (Large negative numbers are smaller than small positive numbers in this case with the smallest allowable layer number being -29,999..)

In addition, a variety of methods are available to move components from one layer to another at runtime. This gives us the ability to not only control the order in which components are painted initially, but also to modify that order later at runtime.

This layering concept is a much more powerful approach than simply inserting **getContentPane()** between the reference to the **JFrame** object and the invocation of the **add()** method. (Layering has a long history of beneficial results in computer graphics such as Computer Aided Design and game programming.)

So, where does the **contentPane** object that belongs to the **layeredPane** fit into this?

Components on the **contentPane** are painted behind all components that may be added directly to the **layeredPane**. In fact, the layer position for the **contentPane** is effectively layer number -30,000. The smallest layer number that can be used to place a component directly on a layer is -

29,999 (we will place one there in our sample program later).

More Detailed Discussion

As mentioned earlier, a **JRootPane** contains a **glassPane** (**JPanel**) and a **layeredPane** (**JLayeredPane**). The **layeredPane** contains an optional **menuBar** and a **contentPane** (**JPanel**). The menuBar component is optional and may or may exist at any time. The **layeredPane**, **contentPane**, and **glassPane** will always be available.

Unfortunately, the syntax for dealing with these panes is somewhat different from what you are accustomed to. For example, the typical syntax for adding a component to a container would look something like the following:

```
parentContainer.add(childObject);
```

The proper syntax for dealing with these panes is more like one of the following:

```
myJFrameObject.getContentPane().add(myChildComponent);

myJFrameObject.getLayeredPane().add(
                    myChildComponent, new Integer(5));
```

The first statement above adds a component to the **contentPane**. The second statement adds a component to layer number 5 of the **layeredPane**.

The same concepts apply when removing components, setting layout managers, etc.

The **contentPane** has a **BorderLayout** manager by default. The **layeredPane** has no layout manager (null) by default as you will see in the sample program that follows later in this lesson.

If a **JMenuBar** component is set on the **JRootPane**, it is positioned along the upper edge of the frame. The **contentPane** is adjusted in location and size to fill the remaining area.

If you examine the documentation for **JLayeredPane**, you will see that the class has the following fields which are all public static final **Integer** symbolic constants.

DEFAULT_LAYER - Object defining the Default layer. Equivalent to new Integer(0).

PALETTE_LAYER - Object defining the Palette layer. Equivalent to new Integer(100).

MODAL_LAYER - Object defining the Modal layer. Equivalent to new Integer(200).

POPUP_LAYER - Object defining the Popup layer. Equivalent to new Integer(300).

DRAG_LAYER - Object defining the Drag layer. Equivalent to new Integer(400).

FRAME_CONTENT_LAYER - Object defining the Frame Content layer. This layer is normally only used to position the **contentPane** and menuBar components of **JFrame**. Equivalent to new Integer(-30000).

The **JMenuBar** and the **contentPane** are added to the **layeredPane** component at the following layer (-30000):

JLayeredPane.FRAME_CONTENT_LAYER

As mentioned earlier, the **layeredPane** object is an instance of the **JLayeredPane** class. The purpose of this object is to be the parent of all children of the **JRootPane**.

Also, as mentioned earlier, this object provides the ability to add components at several layers. This is very useful when working with popup menus, dialog boxes, during dragging, or for any other situation in which you might want to separate graphic objects onto different layers for display purposes.

Also as alluded to earlier, the **glassPane** is always added as the first child of the **JRootPane**. This causes the **glassPane** to always be on the top of the stack. By default, the **glassPane** is not visible and is transparent. Thus, mouse events can normally penetrate the **glassPane** and impinge upon the components below it.

It is also possible to draw components on the **glassPane** (although I haven't been able to purposely do it as of 5/12/98). When this is done, those components shadow the components below them preventing mouse events from impinging on the components down below.

Components drawn on the **glassPane** will always be above all other lightweight components in the stacking order of components (but not over top of heavyweight components). Thus, the **glassPane** can be used to assure that such components as popup menus and tool tips are always on top of the other components on the screen.

As mentioned earlier, by default, the **glassPane** is not visible. Developers should use setVisible on the **glassPane** to control when the **glassPane** displays over the other children.

The custom, layout manager used by **JRootPane** insures that the following is true:

- The **glassPane**, if present, fills the entire viewable area of the **JRootPane** (bounds - insets).
- The **layeredPane** fills the entire viewable area of the **JRootPane**. (bounds - insets)
- The **menuBar** is positioned at the upper edge of the **layeredPane**().
- The **contentPane** fills the entire viewable area, minus the **MenuBar**, if present.

If you replace the **LayoutManager** of the **JRootPane**, you are responsible for managing all of these views.

So now we know that the **JRootPane** object contains an object of type **JLayeredPane**.

While **JLayeredPane** manages it's list of children like **Container**, it also allows for the definition of a several layers within itself.

Children in the same layer are managed exactly like the normal **Container** object. However, children in higher layers display above the children in lower layers. Each layer has a distinct integer number.

Apparently each Swing **Component** has a layer attribute (but I haven't been able to find out anything about it other than as described below).

There are at least three ways to set the layer attribute on a **Component**:

1. Passing an Integer object with a literal integer value during the add call:

    layeredPane.add(child, new Integer(10));
2. Passing one of the symbolic constants mentioned earlier during the add call:
    layeredPane.add(child, JLayeredPane.DEFAULT_LAYER)
3. Calling the following set method on the **JLayeredPane** that will be the parent of the component and passing the name of the component and an **int** that specifies the layer number:
    layeredPaneParent.setLayer(child, 10)

In this third case, the layer should be set before adding the child to the parent.

Layers with higher numbers display on top of layers with lower numbers.

Higher and lower in this case includes the algebraic sign of the layer. For example, a large negative number is a lower layer than a small positive number.

These layers are simply a logical construct and **LayoutManagers** will affect all child components without regard for layer settings. Therefore, to take advantage of this layering capability, you may need to design your own layout manager or use absolute layout (null layout manager).

We will see some examples in the sample program that follows.

Sample Program

The primary purpose of this program is to illustrate the use of both the **contentPane** and the **layeredPane**.

The program places a **JFrame** object on the screen as the primary GUI.

A red **JTextField** and a **JLabel** are placed on the **contentPane** on the **JFrame** object. (Note that the **JLabel** is rendered as gray in the metal L&F.)

The **JTextField** is placed in the Center of the **JFrame** using the default border layout. The **JLabel** is placed in the South position on the **JFrame** object.

A green **JButton** and a yellow **JButton** are placed on the **layeredPane**. Apparently the default layout for the **layeredPane** is absolute or null. The **JButton** objects are purposely placed so as to partially overlap. Both buttons appear on top of the red **JTextField** object on the **contentPane**. (Components on layered panes are always on top of components on the **contentPane**).

The green **JButton** is initially placed at layer position +1 on the layered pane and the yellow **JButton** is initially placed at layer position -29999 which is the limit in the negative direction.

Action listeners are registered on the two buttons. When the top button is clicked, the action is to swap the layer positions of the two buttons causing the one on the bottom to move to the top.

Several lines of code are included to investigate the parent child relationships of the various panes. The output from this part of the program follows. Note that line breaks were manually inserted here to force the material to fit in this format.

```
Root pane is: class com.sun.java.swing.JRootPane
Parent of root pane is SwingPane01
     [frame0,0,0,0x0,invalid,hidden,
     layout=java.awt.BorderLayout,resizable,title=]

Glass pane is: class com.sun.java.swing.JPanel
Parent of glass pane is com.sun.java.swing.JRootPane
     [,0,0,0x0,invalid,
     layout=com.sun.java.swing.JRootPane$RootLayout]

Layered pane is: class com.sun.java.swing.JLayeredPane
Parent of layered pane is com.sun.java.swing.JRootPane
     [,0,0,0x0,invalid,
     layout=com.sun.java.swing.JRootPane$RootLayout]
```

```
Content pane is: class com.sun.java.swing.JPanel
Parent of content pane is com.sun.java.swing.JLayeredPane
     [null.layeredPane,0,0,0x0,invalid]

An interpretation of the above is:
  The root pane is a child of the JFrame object.
  The glass pane is a child of the root pane.
  The layered pane is a child of the root pane.
  The content pane is a child of the layered pane.

  The root pane is of type JRootPane.
  The glass pane is of type JPanel.
  The layered pane is of type JLayeredPane.
  The content pane is of type JPanel.
```

The JavaSoft documentation indicates that the **contentPane** is placed at an equivalent layer position of -30000 in the **layeredPane**.

The **glassPane** is something of a mystery to me at this point in time (5/12/98). I was unable to draw on the **glassPane**. An attempt to place either a **JButton** object or a **JToolTip** object on the **glassPane** was rejected by the compiler with error messages that the **JButton** and the **JToolTip** could not be converted to type **JPopupMenu**. I don't know if this is proper behavior or a bug.

The program was tested using JDK 1.1.6 and Swing 1.0.1 under Win95.

Interesting Code Fragments

We will begin with the import statements that highlight the requirement to import the **Swing** packages.

```
import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;
```

As you can see in the next fragment, the controlling class in this program is an object of type **JFrame** because it extends **JFrame**. This fragment also declares some instance variables that are required later in the **ActionEvent** handler to swap the two buttons between layers.

I have also included the **main()** method in this fragment simply to provide continuity.

```
class SwingPane01 extends JFrame{//subclass JFrame
  JLayeredPane theLayeredPane;
  JButton greenButton;
  JButton yellowButton;
```

```
  public static void main(String[] args){
    new SwingPane01();
  }//end main
```

The next fragment shows the beginning of the constructor where much of the work in this program is accomplished. We begin by getting a reference to the **layeredPane** to make it more convenient to work with it later. The width and height values for the **JFrame** object are set in this fragment as well.

```
  SwingPane01(){//constructor
    theLayeredPane = this.getLayeredPane();
    int frameWidth = 300;
    int frameHeight = 200;
```

The next fragment shows the code that is used to discover and display the types of the different panes along with the parent-child relationships in the hierarchy of panes. This code is all pretty intuitive. You should be able to surmise what it is doing simply from the names of the methods being invoked.

```
    System.out.println("Root pane is: " +
                            this.getRootPane().getClass());
    System.out.println("Parent of root pane is " +
                    this.getRootPane().getParent() + "\n");

    System.out.println("Glass pane is: " +
                           this.getGlassPane().getClass());
    System.out.println("Parent of glassPane is " +
                   this.getGlassPane().getParent() + "\n");

    System.out.println("Layered pane is: " +
                         this.getLayeredPane().getClass());
    System.out.println("Parent of layeredPane is " +
                this.getLayeredPane().getParent() + "\n");

    System.out.println("Content pane is: " +
                         this.getContentPane().getClass());
    System.out.println("Parent of contentPane is " +
                this.getContentPane().getParent() + "\n");
```

The next fragment instantiates a **JLabel** object and a **JTextField** object and places them on the **contentPane**. The default layout manager for the **contentPane** is **BorderLayout**. This code places the red **JTextField** in the **Center** position of the **contentPane** and places the **JLabel** in the **South** position.

For the *metal* L&F, the **JLabel** displays as gray and appears to be part of the border at the bottom of the **JFrame** object.

The *South* **JLabel** provides instructions regarding the use of the **JButton** objects that will also be placed on the **JFrame** object.

```
    JLabel theLabel = new JLabel(
         "  Click buttons to swap their layer positions.");
    this.getContentPane().add(theLabel,"South");

    JTextField redTextField = new JTextField(
                         "    redTextField on contentPane");
    redTextField.setBackground(Color.red);
    this.getContentPane().add(redTextField,"Center");
```

The next fragment instantiates a green **JButton** object and places it on the **layeredPane** at layer number one (1). Note that the **setBounds()** method is used to establish the location and size of the button in absolute pixel coordinates. Recall that I said earlier that apparently the default layout manager for the **layeredPane** is null. Otherwise, it would have been necessary for me to set it to null before placing this object on an absolute location and size basis.

Note that an **ActionListener** is registered on the **JButton** object. We will see the class that defines the behavior of the listener object later.

```
    greenButton = new JButton(
                             "greenButton on Layered Pane");
    greenButton.setBackground(Color.green);
    greenButton.setBounds(10,10,240,40);
    greenButton.addActionListener(new MyActionListener());
    theLayeredPane.add(greenButton,new Integer(1));
```

Next we instantiate a yellow **JButton** object and place it on the **layeredPane** at layer number -29999. This is the bottom-most layer of all possible layers in the **layeredPane**. The **contentPane** falls immediately below this layer.

```
    yellowButton = new JButton(
                            "yellowButton on Layered Pane");
    yellowButton.setBackground(Color.yellow);
    yellowButton.setBounds(40,20,240,40);
    yellowButton.addActionListener(new MyActionListener());
    theLayeredPane.add(yellowButton,new Integer(-29999));
```

After this, we set the title, size, visibility, etc., of the **JFrame** object and register an anonymous listener to terminate the program when the user closes the **JFrame**. And that ends the constructor.

```
    this.setTitle("Copyright 1998, R.G.Baldwin");
    this.setSize(frameWidth,frameHeight);
    this.setVisible(true);
    //=================================================//

    //Anonymous inner class to terminate program.
    this.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);}});//end addWindowListener

  }//end constructor
```

Finally, we see the **ActionListener** class from which listener objects are registered on the two **JButton** objects. This is a standard listener class that defines the **actionPerformed()** method.

The behavior of the **actionPerformed()** method in this case is to cause the layer positions of the two **JButton** objects to be swapped whenever the button on top is clicked. This causes the **JButton** object immediately below the one on the top to move to the top of the stack.

Note that this method invokes the **setLayer()** method of the **JLayeredPane** class to rearrange the positions of the two **JButton** objects at runtime.

```
  class MyActionListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
      if(e.getActionCommand().equals(
                          "greenButton on Layered Pane")){
        theLayeredPane.setLayer(greenButton,-29999);
        theLayeredPane.setLayer(yellowButton,1);
      }else{
        theLayeredPane.setLayer(greenButton,1);
        theLayeredPane.setLayer(yellowButton,-29999);
      }//end else
    }//end actionPerformed()
  }//end class MyActionListener
```

.

Program Listing

This section contains a complete listing of the program.

```
/*File SwingPane01 Copyright 1998, R.G.Baldwin
The purpose of this program is to illustrate the use of
the contentPane and the layeredPane.

The program places a JFrame object on the screen as the
primary GUI.

A red JTextField and a JLabel are placed on the
contentPane on the JFrame object.  (The label is
actually rendered as gray in the metal L&F.)

The JTextField is placed in the Center of the JFrame using
the default border layout.  The JLabel is placed in the
South position on the JFrame object.

A green JButton and a yellow JButton are placed on the
layeredPane of the JFrame object.  Apparently the default
layout for the layeredPane is absolute or null.  The
JButton objects are purposely placed so as to partially
overlap.  Both buttons appear on top of the JTextField
object on the contentPane.

The green JButton is initially placed at layer position +1
on the layeredPane and the yellow JButton is initially
placed at layer position -29999 which is the limit in the
negative direction.

Action listeners are registered on the two buttons.  When
the top button is clicked, the action is to swap the layer
positions of the two buttons causing the other one to
move to the top layer position.

Several lines of code are included to investigate the
parent child relationships of the various panes.  The
output from this part of the program follows.  Note that
line breaks were manually inserted here to force the
material to fit in this format.


Root pane is: class com.sun.java.swing.JRootPane
Parent of root pane is SwingPane01
     [frame0,0,0,0x0,invalid,hidden,
     layout=java.awt.BorderLayout,resizable,title=]

Glass pane is: class com.sun.java.swing.JPanel
Parent of glassPane is com.sun.java.swing.JRootPane
     [,0,0,0x0,invalid,
     layout=com.sun.java.swing.JRootPane$RootLayout]

Layered pane is: class com.sun.java.swing.JLayeredPane
Parent of layeredPane is com.sun.java.swing.JRootPane
     [,0,0,0x0,invalid,
     layout=com.sun.java.swing.JRootPane$RootLayout]
```

```
Content pane is: class com.sun.java.swing.JPanel
Parent of contentPane is com.sun.java.swing.JLayeredPane
     [null.layeredPane,0,0,0x0,invalid]

An interpretation of the above is:
  The root pane is a child of the JFrame object.
  The glass page is a child of the root pane.
  The layeredPane is a child of the root pane.
  The contentPane is a child of the layeredPane.

  The root pane is of type JRootPane.
  The glassPane is of type JPanel.
  The layeredPane is of type JLayeredPane.
  The contentPane is of type JPanel.

The JavaSoft documentation indicates that the contentPane
is placed at layer position -30000 in the layeredPane.

The glassPane is something of a mystery at this point.
I was unable to draw upon the glassPane.  An attempt to
place either a JButton object or a JToolTip object on the
glass plane was rejected by the compiler with error
messages that the JButton and the JToolTip could not be
converted to type JPopupMenu.

Tested using JDK 1.1.6 and Swing 1.0.1 under Win95.
**********************************************************/

import java.awt.*;
import java.awt.event.*;
import com.sun.java.swing.*;

class SwingPane01 extends JFrame{//subclass JFrame
  JLayeredPane theLayeredPane;
  JButton greenButton;
  JButton yellowButton;

  public static void main(String[] args){
    new SwingPane01();
  }//end main
  //-------------------------------------------------------//

  SwingPane01(){//constructor
    //Get a ref to the layeredPane for later use.
    theLayeredPane = this.getLayeredPane();
    int frameWidth = 300;
    int frameHeight = 200;

    //Get and display types of different panes along with
    // parent-child hierarchy.
    System.out.println("Root pane is: " +
                            this.getRootPane().getClass());
    System.out.println("Parent of root pane is " +
                    this.getRootPane().getParent() + "\n");

    System.out.println("Glass pane is: " +
```

```
                          this.getGlassPane().getClass());
    System.out.println("Parent of glassPane is " +
                   this.getGlassPane().getParent() + "\n");

    System.out.println("Layered pane is: " +
                         this.getLayeredPane().getClass());
    System.out.println("Parent of layeredPane is " +
                 this.getLayeredPane().getParent() + "\n");

    System.out.println("Content pane is: " +
                         this.getContentPane().getClass());
    System.out.println("Parent of contentPane is " +
                 this.getContentPane().getParent() + "\n");


    //Put instructions in a JLabel on the contentPane.
    JLabel theLabel = new JLabel(
         "  Click buttons to swap their layer positions.");
    this.getContentPane().add(theLabel,"South");

    //Put a red JTextField in the Center of the JFrame on
    // the contentPane.
    JTextField redTextField = new JTextField(
                        "    redTextField on contentPane");
    redTextField.setBackground(Color.red);
    this.getContentPane().add(redTextField,"Center");

    //Put a green JButton on the layeredPane at a layer
    // position of +1.
    greenButton = new JButton(
                            "greenButton on Layered Pane");
    greenButton.setBackground(Color.green);
    greenButton.setBounds(10,10,240,40);
    greenButton.addActionListener(new MyActionListener());
    theLayeredPane.add(greenButton,new Integer(1));

    //Put a yellow JButton on the layeredPane at a layer
    // position of -29999.
    yellowButton = new JButton(
                           "yellowButton on Layered Pane");
    yellowButton.setBackground(Color.yellow);
    yellowButton.setBounds(40,20,240,40);
    yellowButton.addActionListener(new MyActionListener());
    theLayeredPane.add(yellowButton,new Integer(-29999));

    //Set title, size, and visibility of JFrame object.
    this.setTitle("Copyright 1998, R.G.Baldwin");
    this.setSize(frameWidth,frameHeight);
    this.setVisible(true);
    //================================================//
    //Anonymous inner class to terminate program.
    this.addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);}});//end addWindowListener

  }//end constructor
```

```
  //=======================================================//

  //Inner class for listener objects which swap the layer
  // positions of the two JButton objects when the one on
  // the top is clicked.
  class MyActionListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
      if(e.getActionCommand().equals(
                           "greenButton on Layered Pane")){
        theLayeredPane.setLayer(greenButton,-29999);
        theLayeredPane.setLayer(yellowButton,1);
      }else{
        theLayeredPane.setLayer(greenButton,1);
        theLayeredPane.setLayer(yellowButton,-29999);
      }//end else
    }//end actionPerformed()
  }//end class MyActionListener


}//end class SwingPane01
//=========================================================//
```

.

-end-