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

The AWT Package, Placing Components in Containers, GridLayout

Java Programming, Lecture Notes # 118, Revised 02/21/98.

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.

JDK 1.1 was formally released on February 18, 1997. This lesson was originally written on March 7, 1997 using the software and documentation in the JDK 1.1 download package. It has been updated on various occasions since then.

Introduction

This is one in a series of lessons that concentrate on the package **java.awt** where most of the functionality exists for providing the user interface to your application or applet.

In this lesson, we will learn how to use the **GridLayout** class for creating a layout manager.

Perhaps more important, we will take another important step in learning about the User Interface. In particular, we will learn how to create composite user-interface objects by building up a combination of smaller objects.

In particular, we will create a top-level user-interface object that is a **Frame** object containing two **Panel** objects where each of the **Panel** objects contains several **Button** objects.

This is one of the processes that you can use to create the types of complex User Interface objects that you are accustomed to seeing in typical Graphical User Interfaces.

GridLayout Manager

In earlier lessons, we have learned about absolute positioning and sizing of components in a container, have learned about the **BorderLayout** manager, and have learned about the **FlowLayout** manager.

The **GridLayout** manager lays out components in a grid of rows and columns.

If the container is resized by the user at runtime, the system attempts to maintain the proper row/column layout within overall size constraints. Maintaining overall row/column format takes priority over minimum size constraints and, for example, the components can shrink to such a small size that labels become unreadable.

The **GridLayout** class has no *Fields*, has three *Constructors*, and has more than a dozen *methods*.

In most cases, you will probably only be concerned with the *Constructors* unless you need to dynamically modify the layout at runtime.

The three constructors are:

- **GridLayout()** --Creates a grid layout with a default of one column per component, in a single row.
- **GridLayout(int, int)** -- Creates a grid layout with the specified rows and columns.
- **GridLayout(int, int, int, int)** -- Creates a grid layout with the specified rows, columns, horizontal gap, and vertical gap.

By this point, you should have no difficulty with the terminology used in the description of these three constructors.

Some previous lessons have contained two sample programs; one very simple program, and another program that was not quite so simple.

This lesson contains only one sample program and it is considerably more complex than any of the sample programs in previous lessons on layout managers. This complexity doesn't derive from layout management as such. Rather, the complexity derives from the fact that we are now going to begin creating top-level user-interface objects that are composites of two or more lower-level objects.

Also, rather than to pass up on the opportunity to improve our event-handling skills, this sample program also processes **ActionListener** events on two buttons.

When one of the buttons is clicked, the **GridLayout** display format is set to three rows by two columns. When the other button is clicked, the **GridLayout** display format changes to two rows by three columns. Thus, this sample program also illustrates dynamic modification of the layout manager at runtime.

Sample Program

This program illustrates use of the **GridLayout** manager. Equally important, this program illustrates building a fairly complex user-interface object through the composition of other objects. Finally, the program illustrates the process of dynamically modifying a layout at runtime.

Discussion

This program is designed to be compiled and run under JDK 1.1

The top-level user interface consists of a **Frame** object. Two **Panel** objects are placed on the **Frame** object using the default **BorderLayout** manager for the **Frame** object.

One of the **Panel** objects contains six **Button** objects which are placed there using a **GridLayout** manager. These buttons are placed on the **Panel** object for illustration of layout only, and are not functional (they have *no registered listener* objects).

The buttons are initially placed on this panel in a grid pattern consisting of two rows and three columns. This is accomplished by passing the appropriate parameters to the constructor for the **GridLayout** object used to establish the layout manager for the **Panel** object.

The other **Panel** object contains two **Button** objects labeled **3x2** and **2x3**.

These two buttons are placed on the **Panel** object using the default **FlowLayout** manager. These buttons are functional because they have **ActionListener** objects registered on them.

When the user clicks the button labeled **3x2**, the buttons on the other **Panel** object are rearranged into three rows and two columns.

Similarly, when the user clicks the button labeled **2x3**, the buttons on the other **Panel** object are rearranged into two rows and three columns.

A **windowClosing()** event listener object is instantiated and registered on the **Frame** object to terminate the program when the frame is closed.

The program was tested using JDK 1.1 running under Win95.

Interesting Code Fragments

Because of its complexity, this program contains a number of interesting code fragments.

As mentioned earlier, the top-level user-interface object is the composite of several lower-level objects. The general arrangement of the code is designed to construct the components from the

inside out ending with the construction through composition of the final top-level interface object.

We begin by instantiating two **Button** objects which will later become functional. These objects cannot be instantiated anonymously because we will be registering an **ActionListener** object on each of them and we will need a variable referencing each object in order to accomplish the registration.

```
    Button button7 = new Button("3x2");
    Button button8= new Button("2x3");
```

Next, we will instantiate a **GridLayout** object that we will use to establish the layout manager for one of the **Panel** objects. This also cannot be an anonymous object because we will later need access to it in order to modify the row/column arrangement of components in the **Panel** object.

```
GridLayout myGridLayout = new GridLayout(2,3);//row, col
```

Next, we will instantiate the first of two **Panel** objects that will later be combined onto a Frame object. We will use the above instantiated **GridLayout** object to establish the layout manager for the **Panel** object.

Following that, we will use a **for** loop to place six anonymous**Button** objects on the **Panel**. In this program, it is OK for these buttons to be anonymous because we don't intend to register any type of event listeners on the buttons. Normally, however, you would want to register one or more event listeners on a **Button** object, so you normally would **not** make it anonymous.

Note that the caption for each button is being constructed by concatenating the value of **cnt** to the string **Button**.

Note also that the reference variable **panel1** is defined earlier as an instance variable of the **GUI** class because it needs to be accessed later by the **Action** listener.

```
    panel1 = new Panel();
    panel1.setLayout(myGridLayout);

    for(int cnt = 0; cnt < 6; cnt++)
      panel1.add(new Button("Button" + cnt));
```

Next, we will instantiate the second **Panel** object that will later be combined onto the **Frame** object. Once we instantiate it, we will place the two **Button** objects previously instantiated onto the **Panel** object.

```
    Panel panel2 = new Panel();
    panel2.add(button7);
    panel2.add(button8);
```

The next step is to instantiate a **Frame** object which will serve as our top-level user-interface object. Once we instantiate it, we will place the two **Panel** objects on it using the default **BorderLayout** manager.

```
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
    myFrame.add(panel1,"North");
    myFrame.add(panel2,"South");
```

At this point, the physical construction of our top-level user-interface object is complete. The next step is to instantiate and register anonymous **ActionListener** objects which will be used to process action events on the two **Button** objects.

```
    button7.addActionListener(
       new A3x2ActionListener(myGridLayout,myFrame,this));
    button8.addActionListener(
       new A2x3ActionListener(myGridLayout,myFrame,this));
```

The **ActionListener** objects are not greatly different from those that we have seen in previous lessons on layout managers, except this time we are not requiring buttons to share a single listener. In this case, each of the **Button** objects has its own registered **ActionListener** object.

As before, the code in the event handler makes use of methods of the layout manager class to effect the desired action upon receipt of an event. In this case, an event on one **Button** object causes the components to be arranged in a grid with three rows and two columns and an event on the other **Button** object causes the components to be arranged in a grid with two rows and three columns. Since the code for the two event handlers is so similar, only the code for the body of one of the event handlers is shown below.

Note that the reference variables used in this code fragment are passed in as parameters when the **Action** listener object is constructed.

```
  public void actionPerformed(ActionEvent e){
    myGridLayoutObject.setRows(3);
    myGridLayoutObject.setColumns(2);
    myGuiObject.panel1.setLayout(myGridLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
```

A complete listing of the program is provided in the next section.

Program Listing

This section contains a complete listing of the program with a large number of explanatory comments.

```
/*File Layout06.java Copyright 1997, R.G.Baldwin
Revised 10/29/97 to correct a logic error in the
earlier version.

This program is designed to be compiled and run under
JDK 1.1

This program illustrates use of the GridLayout manager.
Equally important, this program illustrates  building a
fairly complex user interface object through the
composition of subunits. Finally, the program illustrates
the process of dynamically modifying a layout at runtime.

The top-level user interface consists of a Frame object.
Two Panel objects are placed on the Frame object using the
default BorderLayout manager.

One of the Panel objects contains six Button objects which
are placed there using a GridLayout manager.  These buttons
are placed on the Panel object using a GridLayout manager
for illustration of layout only, and they are not
functional (they have no registered listener objects).

The buttons are initially placed on this panel in a grid
pattern consisting of two rows and three columns.  This is
accomplished by passing the appropriate parameters to the
constructor for the GridLayout object used to establish the
layout manager for the panel.

The other Panel object contains two Button objects labeled
3x2 and 2x3.  These buttons are placed on the panel using
the default FlowLayout manager. These buttons are
functional.  When the user clicks the button labeled 3x2,
the buttons on the other Panel object are arranged into
three rows and two columns.  Similarly, when the user clicks
the button labeled 2x3, the buttons on the other Panel
object are arranged into two rows and three columns.

A windowClosing() event listener object is instantiated and
registered on the frame to terminate the program when the
frame is closed.

The program was tested using JDK 1.1.3 running under Win95.
*/
//=======================================================//

import java.awt.*;
import java.awt.event.*;
//=======================================================//
public class Layout06 {
```

```
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Layout06
//=========================================================//

class GUI {
  Panel panel1;

  public GUI(){//constructor
    //Instantiate two button objects that will later
    // become functional
    Button button7 = new Button("3x2");
    Button button8= new Button("2x3");

    //Instantiate a layout manager object to be used with
    // a Panel object
    GridLayout myGridLayout = new GridLayout(2,3);//row,col

    //Instantiate the first of two Panel objects that will
    // be combined onto a Frame object.
    panel1 = new Panel();
    //Specify the GridLayout manager for the Panel object
    panel1.setLayout(myGridLayout);
    //Place six Button objects on the Panel with labels
    // as shown
    for(int cnt = 0; cnt < 6; cnt++)
      panel1.add(new Button("Button" + cnt));

    //Instantiate the second Panel object using default
    // FlowLayout and place two Button objects on it.
    // These buttons will become functional later when
    // ActionListener objects are registered on them.
    Panel panel2 = new Panel();
    panel2.add(button7);
    panel2.add(button8);

    //Instantiate a Frame object which will become the
    // top-level user-interface object.
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");

    //IMPORTANT Add the two previously prepared Panel
    // objects to the Frame object to create the composite
    // user-interface object.
    myFrame.add(panel1,"North");
    myFrame.add(panel2,"South");

    myFrame.setSize(250,150);
    myFrame.setVisible(true);

    //Instantiate action listener objects and register on
    // button7 & button8
    button7.addActionListener(
        new A3x2ActionListener(myGridLayout,myFrame,this));
```

```
    button8.addActionListener(
        new A2x3ActionListener(myGridLayout,myFrame,this));

    //Instantiate and register a window listener to
    // terminate the program when the Frame is closed.
    myFrame.addWindowListener(new Terminate());
  }//end constructor
}//end class GUI definition
//=========================================================//

//The next two classes are ActionListener classes.  One
// object of each is instantiated and registered on the two
// active buttons respectively.  The purpose of these event
// handlers is to modify the GridLayout manager for one of
// the Panel objects that make up the composite
// user-interface object.  The first of these two classes
// sets the grid to 3 rows by 2 columns.  The other class
// sets the grid to 2 rows by 3 columns.

//=========================================================//
class A3x2ActionListener implements ActionListener{
  GridLayout myGridLayoutObject;
  Frame myFrameObject;
  GUI myGuiObject;

  //constructor
  A3x2ActionListener(GridLayout layoutObject,
                           Frame inFrame,GUI inGuiObject){
    myGridLayoutObject = layoutObject;
    myFrameObject = inFrame;
    myGuiObject = inGuiObject;
  }//end constructor

  //When an action event occurs, set the rows to 3 and the
  // columns to 2 in the GridLayout object.  Then set the
  // layout manager for the frame to be the newly-modified
  // GridLayout object.  Then validate the frame to ensure
  // a valid layout so that the new visual will
  // take effect.
  public void actionPerformed(ActionEvent e){
    myGridLayoutObject.setRows(3);
    myGridLayoutObject.setColumns(2);
    myGuiObject.panel1.setLayout(myGridLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class A3x2ActionListener
//=========================================================//

class A2x3ActionListener implements ActionListener{
  GridLayout myGridLayoutObject;
  Frame myFrameObject;
  GUI myGuiObject;

  //constructor
  A2x3ActionListener(GridLayout layoutObject,
                           Frame inFrame,GUI inGuiObject){
```

```
    myGridLayoutObject = layoutObject;
    myFrameObject = inFrame;
    myGuiObject = inGuiObject;
  }//end constructor

  //When an action event occurs, set the rows to 2 and the
  // columns to 3 in the GridLayout object.  Then set the
  // layout manager for the frame to be the newly-modified
  // GridLayout object.  Then validate the frame to ensure
  // a valid layout so that the new visual will
  // take effect.
  public void actionPerformed(ActionEvent e){
    myGridLayoutObject.setRows(2);
    myGridLayoutObject.setColumns(3);
    myGuiObject.panel1.setLayout(myGridLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class A3x2ActionListener
//=======================================================//

class Terminate extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    //terminate the program when the window is closed
    System.exit(0);
  }//end windowClosing
}//end class Terminate
//=======================================================//
```

Review

Q - Without viewing the solution that follows, write a Java application that meets the specifications provided in the comments at the beginning of the following program.

A - See the specifications and the solution below.

```
/*File SampProg137.java Copyright 1997, R.G.Baldwin

Write an application that meets the following
specifications:

The program begins with a Frame object on the screen.

The top half of the client area of the frame is yellow.

The bottom half of the client area of the frame is green.

The top half contains six buttons. All six buttons are the
same size,

The six buttons are arranged in two rows of three columns.

The yellow shows through the gap between the buttons.
```

```
The bottom half contains two buttons, one labeled "3x2"
and the other labeled "2x3".

When you click the 3x2 button, the six buttons in the top
half are rearranged into three rows of two columns.

When you click on the 2x3 button, the six buttons are
rearranged into two rows of three columns.

Regardless of the arrangement of the buttons in the top
half, the client area of the frame continues to be divided
into two equal halves.  The bottom half is green, and
the top half is yellow.

When you close the Frame object, the program terminates and
returns control to the operating system.

The program was tested using JDK 1.1.3 running under Win95.
*/
//=======================================================//

import java.awt.*;
import java.awt.event.*;
//=======================================================//
public class SampProg137 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class SampProg137
//=======================================================//

class GUI {
  Panel panel1;//The Action listener req a ref to panel1

  public GUI(){//constructor
    //Instantiate two button objects that will later
    // become functional
    Button button7 = new Button("3x2");
    Button button8= new Button("2x3");

    //Instantiate a layout manager object to be used with
    // a Panel object.  Make the gap 3 pixels.
    //Args: row,col,Hgap,Vgap
    GridLayout myGridLayout = new GridLayout(2,3,3,3);

    //Instantiate the first of two Panel objects that will
    // be combined onto a Frame object and make
    // it yellow.
    panel1 = new Panel();
    panel1.setBackground(Color.yellow);

    //Specify the GridLayout manager for the Panel object
    panel1.setLayout(myGridLayout);
```

```
    //Place six Button objects on the Panel with labels
    // as shown
    for(int cnt = 0; cnt < 6; cnt++)
      panel1.add(new Button("Button" + cnt));

    //Instantiate the second Panel object using default
    // FlowLayout and place two Button objects on it.
    // These buttons will become functional later when
    // ActionListener objects are registered on them.
    Panel panel2 = new Panel();
    panel2.setBackground(Color.green);
    panel2.add(button7);
    panel2.add(button8);

    //Instantiate a Frame object which will become the
    // top-level user-interface object.
    Frame myFrame = new Frame(
                           "Copyright 1997, R.G.Baldwin");
    //Note that the zero in the following argument list
    // allows for an many rows as are needed to accommodate
    // the data.
    myFrame.setLayout(new GridLayout(0,1));

    //Add the two previously prepared Panel objects to the
    // Frame object based on the GridLayout defined above
    // to create the composite user-interface object.
    myFrame.add(panel1);
    myFrame.add(panel2);

    myFrame.setSize(250,150);
    myFrame.setVisible(true);

    //Instantiate action listener objects and register on
    // button7 & button8
    button7.addActionListener(
        new A3x2ActionListener(myGridLayout,myFrame,this));
    button8.addActionListener(
        new A2x3ActionListener(myGridLayout,myFrame,this));

    //Instantiate and register a window listener to
    // terminate the program when the Frame is closed.
    myFrame.addWindowListener(new Terminate());
  }//end constructor
}//end class GUI definition
//=========================================================//

//The next two classes are ActionListener classes.  One
// object of each is instantiated and registered on the two
// active buttons respectively.  The purpose of these event
// handlers is to modify the GridLayout manager for one of
// the Panel objects that make up the composite
// user-interface object.  The first of these two classes
// sets the grid to 3 rows by 2 columns.  The other class
// sets the grid to 2 rows by 3 columns.

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

```
class A3x2ActionListener implements ActionListener{
  GridLayout myGridLayoutObject;
  Frame myFrameObject;
  GUI myGuiObject;

  //constructor
  A3x2ActionListener(GridLayout layoutObject,
                          Frame inFrame,GUI inGuiObject){
    myGridLayoutObject = layoutObject;
    myFrameObject = inFrame;
    myGuiObject = inGuiObject;
  }//end constructor

  //When an action event occurs, set the rows to 3 and the
  // columns to 2 in the GridLayout object.  Then set the
  // layout manager for the frame to be the newly-modified
  // GridLayout object.  Then validate the frame to ensure
  // a valid layout so that the new visual will
  // take effect.
  public void actionPerformed(ActionEvent e){
    myGridLayoutObject.setRows(3);
    myGridLayoutObject.setColumns(2);
    myGuiObject.panel1.setLayout(myGridLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class A3x2ActionListener
//=========================================================//

class A2x3ActionListener implements ActionListener{
  GridLayout myGridLayoutObject;
  Frame myFrameObject;
  GUI myGuiObject;

  //constructor
  A2x3ActionListener(GridLayout layoutObject,
                           Frame inFrame,GUI inGuiObject){
    myGridLayoutObject = layoutObject;
    myFrameObject = inFrame;
    myGuiObject = inGuiObject;
  }//end constructor

  //When an action event occurs, set the rows to 2 and the
  // columns to 3 in the GridLayout object.  Then set the
  // layout manager for the frame to be the newly-modified
  // GridLayout object.  Then validate the frame to ensure
  // a valid layout so that the new visual will
  // take effect.
  public void actionPerformed(ActionEvent e){
    myGridLayoutObject.setRows(2);
    myGridLayoutObject.setColumns(3);
    myGuiObject.panel1.setLayout(myGridLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class A3x2ActionListener
//=========================================================//
```

```
class Terminate extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    //terminate the program when the window is closed
    System.exit(0);
  }//end windowClosing
}//end class Terminate
//===========================================================//
```

-end-