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

The AWT Package, Arranging Components in Containers, BorderLayout

Java Programming, Lecture Notes # 114, 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 6, 1997 using the software and documentation in the JDK 1.1 download package.

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.

BorderLayout

In a previous lesson, we learned how to create a **Frame** object and place **Button** and **Label** components in the frame using *absolute coordinates*. We also learned that there can be some cross-platform compatibility problems with such an approach.

This lesson looks at the first of several different approaches designed to place components in a container while minimizing cross-platform compatibility problems. This is accomplished using *layout manager classes*. In this lesson, we will concentrate on the **BorderLayout** class.

With the **BorderLayout** class, you can place up to five different components in a container. If you place all five components in the container, four of the components will be positioned along the edges of the container and the fifth will be in the center.

You specify the position of each component when you **add** the component to the container using the strings "**North**", "**South**", "**East**", "**West**" and "**Center**".

For example, the following three statements would

- create a new **Panel** container,
- specify that components are to be placed in the container using a **BorderLayout** manager, and
- place a single **Button** object at the bottom ("**South**") position in the container.

.

```
//create a Panel container object
Panel p = new Panel();

//specify use of the BorderLayout manager
p.setLayout(new BorderLayout());

//add a button at the bottom
p.add(new Button("Okay"), "South");
```

BorderLayout interprets the absence of a string location specification the same as "**Center**".

The "**North**", "**South**", "**East**" and "**West**" components get sized and positioned according to their *preferred sizes* and the constraints of the container's size. The "**Center**" component gets any space left over.

Regarding the *preferred size* mentioned above, according to Exploring Java by Niemeyer & Peck:

> "Every component has two important pieces of information used by the layout manager in placing and sizing it: a preferred size and a minimum size. ... For example, a plain **Button** object can be resized to any proportions. However, the button's designer can provide a preferred size for a good-looking button. The layout manager might use this size when there are no other constraints, or it might ignore it, depending on its scheme. Now, if we give the button a label, the button may need a minimum size in order to display itself properly. The layout manager might show more respect for the button's minimum size and guarantee that it has at least that much

space."

According to Java AWT Reference by John Zukowski:

"The **getPreferredSize()** method returns the **Dimension** (width and height) for the preferred size of the components within the container. The container determines its preferred size by calling the **preferredLayoutSize()** method of the current **LayoutManager**, which says how much space the layout manager needs to arrange the components. If you override this method, you are overriding the default preferred size."

For example, the following code fragment is taken from a subsequent lesson on Java Beans. A Java Bean is a reusable component having certain specific characteristics. In this case, the **getPreferredSize()** method of the class that defines the Java Bean has been overridden to establish the preferred size for the Java Bean object to be a square, fifty pixels on each side.

```
  //This method defines the preferred display size of the
  // bean object.
  public synchronized Dimension getPreferredSize(){
    return new Dimension(50,50);
  }//end getPreferredSize()
```

Our first example program will illustrate how the **BorderLayout** manager adjusts the size of a button to accommodate the minimum size needed to display a long label on the button.

The most straightforward way to specify a layout manager is to invoke the **setLayout()** method on the **Container** object passing an object of a class that *implements* the **LayoutManager** interface as a parameter.

BorderLayout is a class that extends **Object** and implements **LayoutManager2**. (If you are curious about the minimal difference between **LayoutManager** and **LayoutManager2**, it would be a good exercise to look it up in the JDK 1.1 documentation.)

The documentation defines the **setLayout()** method as follows:

```
public void setLayout(LayoutManager mgr)
      Parameters: mgr - the specified layout manager
```

BorderLayout has the following constructors:

```
BorderLayout() -- Constructs a new BorderLayout with no gaps between
components.
```

```
BorderLayout(int, int) -- Constructs a BorderLayout with the specified gaps.
```

In addition, the **BorderLayout** class has numerous methods as shown in the following list. We will look at a couple of these methods in the example programs in this lesson. The term *Deprecated* in the following list indicates an old method from JDK 1.0 that has been replaced by another method in JDK 1.1 and should no longer be used. Apparently the interpretation of *Deprecated* is that the method is still being supported, but won't be supported in some future release of the JDK.

- **addLayoutComponent(Component, Object)** -- Adds the specified component to the layout, using the specified constraint object.
- **addLayoutComponent(String, Component)** -- Replaced by addLayoutComponent(Component, Object). *Deprecated*.
- **getHgap()** -- Returns the horizontal gap between components.
- **getLayoutAlignmentX(Container)** -- Returns the alignment along the x axis.
- **getLayoutAlignmentY(Container)** -- Returns the alignment along the y axis.
- **getVgap()** -- Returns the vertical gap between components.
- **invalidateLayout(Container)** -- Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
- **layoutContainer(Container)** -- Lays out the specified container.
- **maximumLayoutSize(Container)** -- Returns the maximum dimensions for this layout given the components in the specified target container.
- **minimumLayoutSize(Container)** -- Returns the minimum dimensions needed to layout the components contained in the specified target container.
- **preferredLayoutSize(Container)** -- Returns the preferred dimensions for this layout given the components in the specified target container.
- **removeLayoutComponent(Component)** -- Removes the specified component from the layout.
- **setHgap(int)** -- Sets the horizontal gap between components.
- **setVgap(int)** -- Sets the vertical gap between components.
- **toString()** -- Returns the String representation of this BorderLayout's values.

We will take a look at two different example programs in this lesson. The first one will be designed for extreme simplicity while the second one will provide additional functionality.

Plain Vanilla Sample Program

This program was designed to be extremely simple. It contains only the code necessary to create a "Plain Vanilla" visual object by placing five non-functional Button objects on a Frame object using the default **BorderLayout** manager. (The buttons are non-functional because there are no event listener objects registered for them.)

One of the buttons is given a long label to force the layout manager to allocate space according to the minimum size requirement of that particular button.

Discussion of First Program

If you compile and execute this program, you will see that you can resize the **Frame** object, and that within limits, the components will automatically be resized to accommodate the overall size of the frame.

You will also note that not all of the possible problems are eliminated. It is still possible to make the frame sufficiently small that

- first the labels begin to be truncated,
- then the center button disappears completely, and
- finally the buttons on the side can be made to disappear.

However, this visual component is much better able to accommodate resizing than was the case in an earlier lesson that contained components placed and sized on an absolute coordinate basis.

Another thing to notice is the way in which the larger minimum size requirement of the button on the "East" (caused by a longer label) is accommodated automatically with the extra space needed being taken from the button in the "Center".

Interesting Code Fragments in First Program

The following code fragment was extracted from the constructor of the the GUI object.

```
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
    myFrame.add(new Button("South"),"South");
    myFrame.add(new Button("West"),"West");
    myFrame.add(new Button("North"),"North");
    myFrame.add(new Button(
                          "East with a long label"),"East");
    myFrame.add(new Button("Center"),"Center");
    myFrame.setSize(250,150);
    myFrame.setVisible(true);
```

These statements create a new **Frame** object and add five *anonymous* **Button** objects to the frame specifying the location of each object using the strings: "South", "West", etc.

Then the size of the frame is established, and frame and its components are made visible.

The label on the button in the "East" position was made longer than the others to cause the minimum size of that button to be larger than the minimum size of each of the other buttons.

Program Listing of First Program

A complete listing of the program follows.

```
/*File Layout02.java Copyright 1997, R.G.Baldwin
```

```
Revised 10/27/97 to better fit within the space.
This program is designed to be compiled and run under
JDK 1.1

This program creates a "Plain Vanilla" visual object by
placing five non-functional Button objects on a Frame
object using the default BorderLayout manager.

One of the buttons is given a long label in order to force
the layout manager to allocate space according to the
minimum size requirement of that particular button.

The program was tested using JDK 1.1 running under Win95.
*/
//=======================================================//

import java.awt.*;
import java.awt.event.*;
//=======================================================//
public class Layout02 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Layout02
//=======================================================//

class GUI {
  public GUI(){//constructor
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
    myFrame.add(new Button("South"),"South");
    myFrame.add(new Button("West"),"West");
    myFrame.add(new Button("North"),"North");
    myFrame.add(new Button(
                         "East with a long label"),"East");
    myFrame.add(new Button("Center"),"Center");
    myFrame.setSize(250,150);
    myFrame.setVisible(true);
  }//end constructor
}//end class GUI definition
//=======================================================//
```

The next sample program is slightly more realistic in that it includes functional buttons and makes use of some of the methods and parameters of the **BorderLayout** Class

Second Sample Program

This program has more substance that the previous one in this lesson, although it still doesn't do anything particularly useful. It is designed simply to illustrate additional aspects of the use of the **BorderLayout** class as a layout manager.

Discussion

Five buttons are added to a frame using a **BorderLayout** object as the layout manager with a three-pixel gap between components in both the horizontal and vertical direction.

An action listener object is instantiated and registered to listen for action events on all five of the buttons, with all five buttons sharing the same action event handler.

The behavior of the action event handler is to increase the spacing between components by five pixels whenever any of the buttons is pressed. This is accomplished by

- increasing the **Vgap** and **Hgap** attributes of the **BorderLayout** object,
- setting the layout manager of the frame to the modified **BorderLayout** object, and
- validating the frame.

The validation step is required in order for the change to take effect and become visible.

If you click the buttons repeatedly, three of the buttons will shrink entirely out of sight. They can be made to reappear by physically enlarging the frame.

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 running under Win95.

Interesting Code Fragments

The three statements in the following code fragment

- instantiate a **Frame** object,
- instantiate a **BorderLayout** object with a gap of three pixels between components in both the horizontal and vertical directions, and
- establish this **BorderLayout** object as the layout manager for the **Frame** object.

.

```
    Frame myFrame = new Frame(
                        "Copyright 1997, R.G.Baldwin");

    BorderLayout myBorderLayout = new BorderLayout(3,3);

    myFrame.setLayout(myBorderLayout);
```

The following statement is typical of those used to instantiate five Button objects.

```
    Button button1 = new Button("South");
```

The following statement is typical of those used to add all five Button objects to the Frame object.

```
   myFrame.add(button1,"South");
```

The following statements are typical of those used to instantiate an action listener object and register it to listen for action events on all five buttons.

```
    MyActionListener myActionListener =
        new MyActionListener(myBorderLayout,myFrame);
    button1.addActionListener(myActionListener);
```

The following code in the **actionPerformed()** event handler uses the **BorderLayout** methods **getHgap(), setHgap(), getVgap(), and setVgap()** to modify the vertical and horizontal gap attributes of the **BorderLayout** object.

Then the modified **BorderLayout()** object is used in conjunction with **setLayout()** to cause the modified **BorderLayout** object to become the layout manager for the **Frame** object. (Note that the layout manager for the **Frame** object is being modified at runtime.)

Then the **validate()** method is used to force the **Frame** object to readjust the size and locations of its components and to display the modified version of itself.

```
  public void actionPerformed(ActionEvent e){
    myBorderLayoutObject.setHgap(
                      myBorderLayoutObject.getHgap() + 5 );
    myBorderLayoutObject.setVgap(
                      myBorderLayoutObject.getVgap() + 5 );
    myFrameObject.setLayout(myBorderLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
```

.

Program Listing

A complete listing of the program with additional comments follows.

```
/*File Layout03.java Copyright 1997, R.G.Baldwin
Revised 10/26/97 to cause the source code to better fit in
the allocated space in the lesson.

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

This program has more substance that the previous one in
this lesson, although it doesn't do anything particularly
useful.

Five buttons are added to a frame using a BorderLayout
object as the layout manager with a three-pixel gap between
components in both the horizontal and vertical direction.

An action listener object is instantiated and registered to
```

```
listen for action events on all five of the buttons, with
all five buttons sharing the same event handler.

The behavior of the action event handler is to increase the
spacing between components whenever any of the buttons is
pressed.  This is accomplished by increasing the Vgap and
Hgap attributes of the BorderLayout object, setting the
layout manager of the frame to the modified BorderLayout
object, and validating the frame.  The validation step is
required in order for the change to become visible.

If you continue to click the buttons long enough, three of
the buttons will shrink entirely out of sight.  They can be
made to reappear by manually enlarging the frame.

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 Layout03 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Layout03
//=======================================================//

class GUI {
  public GUI(){//constructor
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
    //Instantiate a BorderLayout object with default Center
    // alignment and a Vgap and Hgap of 3 pixels.
    BorderLayout myBorderLayout = new BorderLayout(3,3);
    //Set the layout manager for the frame to be the
    // BorderLayout object.
    myFrame.setLayout(myBorderLayout);

    //Instantiate five Button objects
    Button button1 = new Button("South");
    Button button2 = new Button("West");
    Button button3 = new Button("North");
    Button button4 = new Button("East");
    Button button5 = new Button("Center");

    //Add the five Button objects to the Frame object in
    // the positions specified.
    myFrame.add(button1,"South");
    myFrame.add(button2,"West");
```

```
    myFrame.add(button3,"North");
    myFrame.add(button4,"East");
    myFrame.add(button5,"Center");

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

    //Instantiate an action listener object and register
    // it on all five buttons.
    MyActionListener myActionListener =
        new MyActionListener(myBorderLayout,myFrame);
    button1.addActionListener(myActionListener);
    button2.addActionListener(myActionListener);
    button3.addActionListener(myActionListener);
    button4.addActionListener(myActionListener);
    button5.addActionListener(myActionListener);

    //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
//=======================================================//

class MyActionListener implements ActionListener{
  BorderLayout myBorderLayoutObject;
  Frame myFrameObject;

  //constructor
  MyActionListener(
                  BorderLayout layoutObject,Frame inFrame){
    myBorderLayoutObject = layoutObject;//save references
    myFrameObject = inFrame;
  }//end constructor

  //When an action event occurs, increase the horizontal
  // and vertical gap between components in the
  // BorderLayout object.  Then set the layout manager for
  // the frame to be the newly-modified BorderLayout
  // object.  Then validate the frame to ensure a valid
  // layout so that the new visual will take effect.
  public void actionPerformed(ActionEvent e){
    myBorderLayoutObject.setHgap(
                      myBorderLayoutObject.getHgap() + 5 );
    myBorderLayoutObject.setVgap(
                      myBorderLayoutObject.getVgap() + 5 );
    myFrameObject.setLayout(myBorderLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class MyActionListener

//=======================================================//

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, rewrite the program named Layout03 with the following changes:

As in the program named Layout03, five buttons are added to a Frame object using a BorderLayout object as the layout manager with a one-pixel gap between components in both the horizontal and vertical direction. There are no captions on any of the buttons.

When the program starts, the buttons in the East and West positions are each approximately 100 pixels wide and the button in the Center is approximately 50 pixels wide.

Each time you click on any of the buttons, the horizontal gap between buttons increases by five pixels. However, the widths of the East and West buttons don't change appreciably. In other words, the increase in horizontal gap is accommodated almost entirely as a result of the Center button becoming narrower.

If you continue to click the buttons long enough, the Center button will shrink entirely out of sight but the other four buttons will continue to be visible with no apparent change in size. The Center button can be made to reappear by manually enlarging the Frame.

If you manually increase the width of the Frame, the widths of the East and West buttons don't change appreciably. Rather, the Center button increases in width to accommodate the change in the width of the Frame object.

If you start the program running and then manually decrease the width of the Frame object, the Center button decreases in width to accommodate the reduction in overall width. Eventually the Center button will shrink completely out of sight and then the East and West buttons will collide and begin to change width.

A - See the solution below.

```
/*File SampProg135.java Copyright 1997, R.G.Baldwin

Without viewing the solution that follows, rewrite the
program named Layout03 with the following changes:

As in the program named Layout03, five buttons are added to
a Frame object using a BorderLayout object as the layout
manager with a one-pixel gap between components in both the
horizontal and vertical direction.
```

```
There are no captions on any of the buttons.

When the program starts, the buttons in the East and West
positions are each approximately 100 pixels wide and the
button in the Center is approximately 50 pixels wide.

Each time you click on any of the buttons, the horizontal
gap between buttons increases by five pixels.  However,
the widths of the East and West buttons don't change
appreciably.  In other words, the increase in horizontal
gap is accommodated almost entirely as a result of the
Center button becoming narrower.

If you continue to click the buttons long enough, the
Center button will shrink entirely out of sight but the
other four buttons will continue to be visible with no
apparent change in size.  The Center button can be made to
reappear by manually enlarging the Frame.

If you manually increase the width of the Frame, the
widths of the East and West buttons don't change
appreciably.  Rather, the Center button increases in width
to accommodate the change in the width of the Frame
object.

If you start the program running and then manually
decrease the width of the Frame object, the Center button
decreases in width to accommodate the reduction in
overall width.  Eventually the Center button will shrink
completely out of sight and then the East and West buttons
will collide and begin to change width.

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 SampProg135 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class SampProg135
//========================================================//

class GUI {
  public GUI(){//constructor
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
```

```
    //Instantiate a BorderLayout object with a Vgap
    // and Hgap of 1 pixel.
    BorderLayout myBorderLayout = new BorderLayout(1,1);
    //Set the layout manager for the frame to be the
    // BorderLayout object.
    myFrame.setLayout(myBorderLayout);

    //Instantiate five button-like objects.  Note that two
    // of these objects are of type MyButton instead of
    // type Button.
    Button button1 = new Button();
    MyButton button2 = new MyButton();
    Button button3 = new Button();
    MyButton button4 = new MyButton();
    Button button5 = new Button();

    //Add the five button-like objects to the Frame object
    // in the positions specified.
    myFrame.add(button1,"South");
    myFrame.add(button2,"West");
    myFrame.add(button3,"North");
    myFrame.add(button4,"East");
    myFrame.add(button5,"Center");

    myFrame.setSize(250,150);//set size of frame
    myFrame.setVisible(true);

    //Instantiate an action listener object and register
    // it on all five buttons.
    MyActionListener myActionListener =
        new MyActionListener(myBorderLayout,myFrame);
    button1.addActionListener(myActionListener);
    button2.addActionListener(myActionListener);
    button3.addActionListener(myActionListener);
    button4.addActionListener(myActionListener);
    button5.addActionListener(myActionListener);

    //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
//=========================================================//

class MyActionListener implements ActionListener{
  BorderLayout myBorderLayoutObject;
  Frame myFrameObject;

  //constructor
  MyActionListener(
                  BorderLayout layoutObject,Frame inFrame){
    myBorderLayoutObject = layoutObject;//save references
    myFrameObject = inFrame;
  }//end constructor

  //When an action event occurs, increase the horizontal
```

```
  // gap between components in the BorderLayout object.
  // Then set the layout manager for the frame to be the
  // newly-modified BorderLayout object.  Then validate
  // the frame to ensure a valid layout so that the new
  // visual will take effect.
  public void actionPerformed(ActionEvent e){
    myBorderLayoutObject.setHgap(
                      myBorderLayoutObject.getHgap() + 5 );
    myFrameObject.setLayout(myBorderLayoutObject);
    myFrameObject.validate();
  }//end actionPerformed()
}//end class MyActionListener

//=========================================================//

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
//=========================================================//

//This class extends the standard Button class in order
// to make it possible to override the getPreferredSize()
// method.  The BorderLayout manager honors the width of
// the preferred size insofar as possible to maintain the
// specified preferred width.
class MyButton extends Button{

  public synchronized Dimension getPreferredSize(){
     return new Dimension(100,100);
  }//end getPreferredSize()

}//end class MyButton
```

-end-