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, Absolute Coordinates

Java Programming, Lecture Notes # 112, 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 5, 1997 using the software and documentation in the JDK 1.1 download package. The lesson has been updated several times since it was originally written.

Introduction

This is one in a series of lessons that concentrates on the package **java.awt** which contains most of the functionality for providing the user interface to your Java application or applet.

We can place components in containers using absolute position and size coordinates, or we can use any of several different layout managers to do the job.

Using layout managers is considered by many to be the safer approach because this approach is designed to automatically compensate for differences in screen resolution and component size/shape between platforms.

The different layout managers are defined in classes having the following names:

- **BorderLayout**
- **CardLayout**

- **FlowLayout**
- **GridLayout**
- **GridBagLayout**

In addition to the layout manager classes, there is a class named **GridBagConstraints** that is used to work with the **GridBagLayout** class as well as a class named **Insets** that is used to work with other layout manager classes.

In addition to using the standard layout manager classes, we can create our own custom layout manager.

This lesson will discuss the placement of components into a container using absolute coordinates. Subsequent lessons will discuss the other possibilities listed above.

Absolute Positioning and Size

The **Component** class of JDK 1.1 provides the methods **setBounds(int,int,int,int)** and **setBounds(Rectangle)** that allow you to specify the location and size of a component in absolute coordinates measured in pixels.

A description of these two methods, as extracted from the JDK 1.1.3 documentation follows:

```
public void setBounds(int x,
                      int y,
                      int width,
                      int height)
```

Moves and resizes this component. The new location of the top-left

corner is specified by `x` and `y`, and the new size is specified by `width` and `height`.

Parameters:

`x` - The new *x-coordinate* of this component.
`y` - The new *y-coordinate* of this component.
`width` - The new `width` of this component.

`height` - The new `height` of this component.

.

```
public void setBounds(Rectangle r)
```

Moves and resizes this component to conform to the new bounding

rectangle `r`. This component's new position is specified

by `r.x` and `r.y`, and its new size is specified by `r.width` and `r.height`
Parameters:

`r - The new bounding rectangle for this component.`

As you can see from the above, the location and size of the component are specified by the location and size of a bounding rectangle that can in turn be specified by four integer parameters, or can be specified as an object of class **Rectangle**..

Many authors will tell you that specifying the location and size of your components in absolute coordinates will make it more difficult to achieve satisfactory results on a variety of platforms and displays. Regardless of that, you probably need to know how to do it, and that is the subject of this lesson.

The following sample program illustrates the positioning and sizing of components on an absolute coordinate basis in JDK 1.1. Note that this program uses several new methods that did not exist in JDK 1.0.

Sample Program

The following application, named **Layout01.java** places a **Button** object and a **Label** object in a **Frame** object using absolute coordinates in pixels. The program is designed for simplicity and therefore does not contain any event handlers. With no event handlers, the *Close* button on the Frame object is not operational, so you will have to terminate the program using some other method.

Discussion

The program shows how to use the new methods in JDK 1.1 to set the size and position of components in a **Frame** object using absolute coordinates.

The program also illustrates some of the potential problems associated with the use of absolute coordinates.

A **Button** and a yellow **Label** are placed in a **Frame** object. Note that the specified location and size of the components may cause the two objects to overlap. This is one of the potential problems with the use of absolute coordinates. Note also that the initial height of the **Frame** object may clip off the bottom of the **Label**. Of course, problems of this sort could be avoided, on one platform at least, by careful design.

If the **Frame** is resized by the user, the **Button** and the **Label** remain fixed in size and position. The **Frame** can be resized to the point where the two components are no longer visible. Hopefully, a user would know not to do that.

Also, the specified size of the **Label** may not be sufficient to display the entire initial text string placed in the label. Again, this could be avoided, on one platform, through more careful design.

Getting it right on one platform, however, doesn't guarantee that it will be right on other platforms as well.

The program was tested using JDK 1.1 running under Win95.

Interesting Code Fragments

The following code fragments from the program are particularly interesting.

Create and Size the Button Object

The following two statements

```
    Button myButton = new Button("Button");
    myButton.setBounds(new Rectangle(25,50,100,75));
```

result in the following actions.

- Create a **Button** object with the caption "Button"
- Specify that the button is to be located at x = 25 and y = 50, measured in pixels from the upper left-hand corner of the containing object. Positive y is measured going down the screen. The reference point on the button is the upper left-hand corner of the button.
- Specify that the button is to be 100 pixels wide and 75 pixels high

Note that at the point in the program where these two statements are executed, the containing object hasn't yet been identified. Thus, **setBounds()** provides absolute coordinate information for whatever container the **Button** is eventually placed in.

The **setBounds()** method is a method of the **Component** class that is inherited by all subclasses of **Component** including **Button**.

The method specifies that the component is to be located and sized according to a bounding rectangle whose sides are parallel to the **x** and **y** axes.

As shown earlier, there are two overloaded versions of **setBounds()**. One version allows the location and size of the box to specified directly as four integer parameters (x, y, width, height).

The other version, which is the version used above, requires the location and size of the box to be specified by an object of type **Rectangle** that is passed as a parameter to the method. This is the most flexible approach because the **Rectangle** class has about seven different constructors that can be used to create the bounding box.

Create and Size the Label Object

The next interesting code fragment creates a **Label** component with a yellow background, again using the **setBounds()** method to specify its location and size. This code places the upper left-hand corner of the **Label** object at a coordinate position of 100 by 100 pixels, and causes its width to be 100 pixels and its height to be 75 pixels.

```
    Label myLabel = new Label(
                      "Copyright 1997, R.G.Baldwin");
    myLabel.setBounds(new Rectangle(100,100,100,75));
    myLabel.setBackground(Color.yellow);
```

The **setBackground(Color c)** method is also a method of the **Component** class that is inherited by subclasses of **Component**. This method requires an object of the **Color** class as a parameter. There are many different ways to create the **Color** object, the simplest of which is simply to refer to one of the *class constants* that are defined in the **Color** class using syntax such as

```
public final static Color yellow
```

Recall that variables declared as **static** in a class can be accessed using the name of the class and the name of the variable. Making the variable **final** causes it to be a constant.

About thirteen different colors can be specified in this way. If you need a color not included in this list, you can use one of the overloaded constructors to specify your color using specified contributions of red, green, and blue. The **Color** class provides many different methods for dealing with colors.

Create the Frame Object

The next interesting code fragment creates a **Frame** object with the title "Copyright 1997, R.G.Baldwin" **Frame** is the type of object that users of the *Windows* operating system would probably consider to be a typical "Window". It is *resizable*, has a *minimize* button, a *maximize* button, a *control box* button, and a *close* button.

The default layout for a frame is **BorderLayout** which we will be studying later.

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

In this particular example, we don't want to accept the default layout. Rather, we want to be able to specify the location and size of our components in absolute coordinates . Therefore, we will override the default layout with a specification of a **null** layout as a parameter to the **setLayout()** method.

The **setLayout()** method is a method of the **Container** class, of which the **Frame** class is a subclass. The **setLayout()** method requires an object of the **LayoutManager** class or **null** (indicating no layout manager) as a parameter.

Add the Components to the Frame

The next interesting code fragment causes the two previously defined components to become part of the composite visual object through use of the **add()** method of the class **Container**. This method is inherited by the **Frame** class which is a subclass of **Container**.

As of 3/6/97, there are five overloaded versions of the **add()** method. We are using the version that requires an object of type **Component** as a parameter. Since our **Button** and **Label** objects are subclasses of the **Component** class, they qualify as parameters allowing us to use the following code:

```
    myFrame.add(myButton);
    myFrame.add(myLabel);
```

Size the Frame and Make It Visible

Once we reach this point, all that is left to cause our GUI to appear on the screen at program startup is to establish the size of the **Frame** object in pixels, and to make it *visible*. We accomplish this with the following two statements:

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

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

Program Listing

The complete listing of the program with additional comments follows.

```
/*File Layout01.java Copyright 1997, R.G.Baldwin
Revised 10/27/97 to improve the layout on the page.

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

The program shows how to use the new methods in JDK 1.1 to
set the size and position of components in a Frame object
using absolute coordinates.

The program also illustrates some of the problems
associated with the use of absolute coordinates.

A button and a yellow label are placed in a Frame object.
Note that the two objects may overlap, and that the initial
height of the Frame object may clip off the bottom of the
Label.

Note also that if the Frame is resized by the user, the
Button and the Label remain fixed in size and position.
The Frame can be resized to the point where the two
components are no longer visible.
```

```
Note also that the specified size of the Label may not be
sufficient to display all of the initial text placed in the
label.

For simplicity, no event handler is provided for the Button
and no event handler is provided for the "Close" button on
the Frame. Therefore, it is necessary to terminate the
program using some other method.

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

import java.awt.*;
import java.awt.event.*;
//=========================================================//
public class Layout01 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class Layout01
//=========================================================//

//The following class is used to instantiate a graphical
// user interface object.
class GUI {
  public GUI(){//constructor
    //Create a Button object with the specified caption and
    // the specified size and location within its container
    // (in pixels).
    Button myButton = new Button("Button");
    //Arguments are x,y,width,height
    myButton.setBounds(new Rectangle(25,50,100,75));

    //Create a Label object with the specified initial text
    // and the specified size and location within its
    // container (in pixels).  Make it yellow so that its
    // outline will be visible.
    Label myLabel = new Label(
                        "Copyright 1997, R.G.Baldwin");
    //Arguments are x,y,width,height
    myLabel.setBounds(new Rectangle(100,100,100,75));
    myLabel.setBackground(Color.yellow);

    //Create a Frame object with the specified title, and
    // with no layout manager so that size and location of
    // components shown above will be effective.
    Frame myFrame = new Frame(
                        "Copyright 1997, R.G.Baldwin");
    //Note the following null argument.
    myFrame.setLayout(null);

    //Add the two components to the Frame, set its size in
    // pixels, and make it visible.
    myFrame.add(myButton);
    myFrame.add(myLabel);
    myFrame.setSize(250,150);
    myFrame.setVisible(true);
  }//end constructor
}//end class GUI definition
//=========================================================//
```

Review

Q - Without viewing the following solution, upgrade the program named Layout01 to cause it to terminate when the close button is pressed on the Frame object.

While continuing to use absolute coordinate and sizing, modify the program to ensure that the components don't overlap, there is a safety margin of at least 25 pixels between the components and the edge of the client area in the Frame object, and that the Frame object is sufficiently large to contain the two components.

A - See the following program.

```
/*File SampProg134.java Copyright 1997, R.G.Baldwin
Without viewing the following solution, upgrade the
program named Layout01 to cause it to terminate when the
close button is pressed on the Frame object.

While continuing to use absolute coordinate and sizing,
modify the program to ensure that the components don't
overlap, there is a safety margin of at least 25 pixels
between the components and the edge of the Frame object,
and that the Frame object is sufficiently large to
contain the two components.

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

A button and a yellow label are placed in a Frame object.

Note that with JDK 1.1.3, if the top margin is set to zero,
the button is partially covered by the banner at the top
of the Frame object.  Thus, the vertical coordinate of the
white portion of the client area in the Frame object does
not begin at zero.  Rather, the zero coordinate is
somewhere underneath the banner. Experimentation indicates
that the banner at the top covers about the first twenty
pixels of the white client area of the Frame object.

Note that if the Frame is resized by the user, the
Button and the Label remain fixed in size and position.
The Frame can be resized to the point where the two
components are no longer visible.

Note also that the specified size of the Label may not be
```

```
sufficient to display all of the initial text placed in the
label.

For simplicity, no event handler is provided for the
Button.

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

import java.awt.*;
import java.awt.event.*;
//=======================================================//
public class SampProg134 {
  public static void main(String[] args){
    //instantiate a Graphical User Interface object
    GUI gui = new GUI();
  }//end main
}//end class SampProg134
//=======================================================//

//The following class is used to instantiate a graphical
// user interface object.
class GUI {
  int leftMargin = 25;//specify the margins
  int topMargin = 45;
  int rightMargin = 25;
  int bottomMargin = 25;

  int buttonWidth = 100;//specify button width and height
  int buttonHeight = 75;

  int labelWidth = 100;//specify label width and height
  int labelHeight = 75;

  public GUI(){//constructor
    //Create a Button object with the specified caption and
    // the specified size and location within its container
    // (in pixels).
    Button myButton = new Button("Button");
    //Arguments are x,y,width,height
    myButton.setBounds(new Rectangle(
           leftMargin,topMargin,buttonWidth,buttonHeight));

    //Create a Label object with the specified initial text
    // and the specified size and location within its
    // container (in pixels).  Make it yellow so that its
    // outline will be visible. Base its location on the
    // location and the size of the Button object.
    Label myLabel = new Label(
                            "Copyright 1997, R.G.Baldwin");
    //Arguments are x,y,width,height
    myLabel.setBounds(new Rectangle(
         leftMargin + buttonWidth,topMargin + buttonHeight,
                                 labelWidth,labelHeight));
    myLabel.setBackground(Color.yellow);
    //Create a Frame object with the specified title, and
    // with no layout manager so that size and location of
    // components shown above will be effective.
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");
    //Note the following null argument.
    myFrame.setLayout(null);

    //Add the two components to the Frame, set its size in
    // pixels, and make it visible.
    myFrame.add(myButton);
    myFrame.add(myLabel);
    myFrame.setSize(
       leftMargin + buttonWidth + labelWidth + rightMargin,
                             topMargin + buttonHeight +
                               labelHeight + bottomMargin);
    myFrame.setVisible(true);

    myFrame.addWindowListener(new WindowClose());
  }//end constructor
}//end class GUI definition
//=========================================================//

//Class used to instantiate window listener objects.
class WindowClose extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(1);
  }//end windowClosing()
}//end class WindowClose
```

-end-