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

The AWT Package, Graphics - Getting Started

Java Programming, Lecture Notes # 160, Revised 02/03/98.

Preface

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

Introduction

This lesson is the first in a series of several lessons which will concentrate primarily on the use of the **Graphics** class to render shapes, fonts, and images on the screen.

Before getting into the technical details, we will look at some relatively simple but interesting programs that illustrate the rendering of shapes and fonts. We will defer the rendering of images to a subsequent lesson. This lesson will explain the sample program in a general sense. Subsequent lessons will fill in the technical details of the **Graphics** class.

Sample Program

This program illustrates the use of the **Canvas** class and several of the methods of the **Graphics** class.

The program uses simple drawing techniques to create two fake buttons that appear (on some systems at least) to be green 3D button objects.

Note that these 3D buttons are not constructed using the **Button** class or the **draw3DRect()** method of the Graphics class. Rather, they are rendered using a simple **drawRect()** method of the **Graphics** class (which draws a 2D rectangle). The 3D effect results from a visual artifact of the drawing process.

When the program starts, two green fake buttons with captions of *"Small"* and *"Large"* are displayed in a **Frame** object along with a real **Button** that has a caption of *"Button"*.

The green buttons initially appears to protrude slightly from the surface of the **Frame** object in a manner similar to the real **Button** object.

When you point to a green button and press the left mouse button, the green button appears to be depressed into the surface of the **Frame** object in a manner similar to a real **Button** object.

When you release the mouse button, the green button appears to pop back out and protrude from the surface of the **Frame** object.

Since this is an optical illusion, some observers may not experience the illusion in which case they will simply see green rectangles with text and lines on them.

The 3D effect is an artifact of the manner in which the rectangle is drawn onto an underlying **Canvas** object.

The fake buttons are instances of a class that subclasses the **Canvas** class. The program subclasses **Canvas** to override the **paint()** method in order to set the size and color of a **Canvas** object, to draw a rectangle which is partially on and partially off of the **Canvas** object, and to draw a caption on the **Canvas** object. The fact that the rectangle is only partially on the **Canvas** object causes the **Canvas** object to appear to protrude out of the screen, or to be depressed into the screen which produces the 3D effect.

The rectangle is offset either up and to the left relative to the **Canvas** object or down and to the right. In either case, two lines on the border of the rectangle are lost because they hang off the edge of the **Canvas** object. The remaining two lines appear to be shadows and this causes the 3D effect. When the two visible lines are on the right and bottom of the **Canvas** object, the button appears to protrude out of the screen. When the two visible lines are on the left and top of the **Canvas** object, the button appears to be depressed into the screen.

When you click on one of the green buttons, some text is displayed on the standard output device to confirm that the click occurred. Some text is also displayed when you click on the real **Button** object.

It may be significant to note that the fake green buttons are much more responsive than the real **Button** when running under Win95 on a machine with a 133MHz Pentium. This can be demonstrated by clicking repeatedly on one or the other as fast as you can click. The real **Button** appears to miss some of the clicks while the fake green buttons appear to respond properly to every click. At least the fake green buttons respond to a much higher percentage of clicks than the real **Button**.

These results were produced using JDK 1.1.3, running under Windows 95.

Two improved versions of this program are presented at the end of this lesson. The first improved version uses lines instead of rectangles to achieve the shadow effect. Double lines are used to enhance the shadow effect.

In addition, the caption on the button is caused to move slightly when the button is pushed. This also enhances the 3D effect by giving the illusion of motion to the button.

The second improved version draws highlights in addition to shadows on the button. In these two cases, a lighter or darker shade of the background color of the button (instead of black) is used to produce the effect. This produces a much more convincing illusion of a 3D button than the previous two versions of the program.

Interesting Code Fragments

The first interesting code fragment is the constructor for the class that extends **Canvas** and overrides **paint()** along with the set of instance variables that define the state of the object.

The instance variable named **pushed** is used in the overridden **paint()** method to determine how to render the fake button. This instance variable is initialized to *false* and later modified by a mouse listener object. When **pushed** is *false*, the fake button appears to protrude out of the screen, and when it is *true*, the fake button appears to be depressed into the screen.

The constructor simply accepts parameters for the **width** and **height** of the fake button along with a **caption** to display on the button and saves those parameters in a set of instance variables.

```
class FakeButtonClass extends Canvas{
  boolean pushed = false;//indicates button pushed or not
  int width; //button width in pixels
  int height;//button height in pixels
  String caption;//caption for button

  public FakeButtonClass(//constructor
                      int width,int height,String caption){
    this.width = width;
    this.height = height;
    this.caption = caption;
    this.setBackground(Color.green);
    this.setSize(this.width,this.height);
  }//end constructor

  //...
}//end class FakeButtonClass
```

We will discuss the overridden **paint()** method in two steps. The first discussion centers on the rendering of the 3D button. The second discussion centers on the rendering of the **caption** in the object.

As you can see in the follow code fragment, the code for rendering the 3D button is very simple. A determination is made as to which edges of the rectangle should hang off the **Canvas** object

using the value of the **pushed** instance variable. Then the **drawRect()** method is used to draw a rectangle on the **Canvas** object with the same dimensions as the **Canvas** object. The first two parameters of the **drawRect()** method specify the coordinates of the upper left-hand corner of the rectangle in its container (the **Canvas** object).

For the "depressed" case, the rectangle is offset down and to the right by one pixel causing the right and bottom border lines of the rectangle to be lost.

For the "protruding" case, the rectangle is offset up and to the left by one pixel causing the left and top border lines of the rectangle to be lost.

```
class FakeButtonClass extends Canvas{
  //...

  public void paint(Graphics g){
    if(pushed) g.drawRect(1,1,width,height);
    else g.drawRect(-1,-1,width,height);

    //...
  }//end paint()
}//end class FakeButtonClass
```

The next step in the discussion of the overridden **paint()** method centers around locating the **caption** within the object. The goal is to have it approximately centered in the object. This is slightly more complicated than drawing the rectangle. We will settle for causing the **caption** to be centered horizontally and slightly above the center vertically.

We will accomplish the positioning using an object of type **FontMetrics**. Such an object can be caused to contain information about the font being used in a particular **Graphics** object. We will have more to say about this in a subsequent lesson. For the time being, suffice it to say that we can obtain a **FontMetrics** object containing information about the font in our **Graphics** object by invoking the **getFontMetrics()** method on that object. Having the **FontMetrics** object, we can extract information about the font by invoking methods on the **FontMetrics** object.

Horizontal placement is easy. We determine the width in pixels of our **caption** by invoking the **stringWidth()** method on the **FontMetrics** object and passing the **caption** as a parameter. Then we do a little arithmetic to determine the horizontal coordinate where we should render the **caption**.

Vertical placement is a little more difficult. The **getHeight()** method returns a combination of several parameters that describe the font:

- **leading** (space between lines)
- **ascent** (size of the upper-case characters above the baseline)
- **descent** (size of the part that hangs down below the baseline, as in a lower-case "y")

For ease of programming, I simply adjusted the baseline down from the vertical center of the object by an amount equal to one-fourth of the value returned by **getHeight()** and it seemed to work pretty well for the default font that I was using.

```
class FakeButtonClass extends Canvas{
  //...
  String caption;//caption for button

  public void paint(Graphics g){
    //...

    int fontHeight = g.getFontMetrics().getHeight();
    int stringWidth =
                   g.getFontMetrics().stringWidth(caption);
    g.drawString(caption,
          (width-stringWidth)/2,(height/2)+(fontHeight/4));
  }//end paint()
}//end class FakeButtonClass
```

The next interesting code fragment contains statements from the constructor of our GUI object to instantiate two objects of the **FakeButtonClass** which are subsequently added to a **Frame** object using a **FlowLayout** manager.

```
    FakeButtonClass firstFakeButton =
                        new FakeButtonClass(40,20,"Small");
    FakeButtonClass secondFakeButton =
                        new FakeButtonClass(80,40,"Large");
```

The next interesting code fragment instantiates a mouse listener object and registers it to listen for mouse events on the two **FakeButton** objects.

```
    MouseListenerClass myMouseListener =
                                  new MouseListenerClass();
    firstFakeButton.addMouseListener(myMouseListener);
    secondFakeButton.addMouseListener(myMouseListener);
```

Next, we see the code in the overridden **mousePressed()** method in the mouse listener object which is invoked whenever a **mouse** event occurs on one of the **FakeButton** objects. This method is invoked on the downstroke of the mouse button and causes the appearance of the **FakeButton** object to become depressed into the screen by setting the **pushed** instance variable of the object to *true* and calling **repaint()** on the object.

Note that the **component** object returned from **getComponent()** must be downcast before it can be used to access the **pushed** instance variable.

```
  public void mousePressed(MouseEvent e){
    ((FakeButtonClass)e.getComponent()).pushed = true;
    e.getComponent().repaint();
  }//end mousePressed()
```

Finally, we see the code in the overridden **mouseReleased()** method in the mouse listener object. This method is invoked on the upstroke of the mouse button and causes the appearance of the **FakeButton** object to appear to protrude from the screen by setting the **pushed** instance variable of the object to *false* and calling **repaint()** on the object.

This method also displays a message on the screen to confirm that the event has occurred.

```
  public void mouseReleased(MouseEvent e){
    //Display a message
          System.out.println("mouseReleased");

    ((FakeButtonClass)e.getComponent()).pushed = false;
    e.getComponent().repaint();
  }//end mouseReleased()
```

.

Program Listing

This section contains a complete listing of the program.

```
/*File Shapes02.java.java Copyright 1997, R.G.Baldwin
This program illustrates the use of the Canvas class and
several of the methods of the Graphics class.

The program creates two fake buttons that appear (on some
systems at least) to be green 3D button objects.

Note that the green 3D buttons are not constructed using
the Button class or the draw3DRect() method of the Graphics
class.

When the program starts, two green fake buttons with
captions of "Small" and "Large" along with a real Button
with a caption of "Button" appear in a Frame object.

The green buttons appears to protrude slightly from the
surface of the Frame object in a manner similar to the
real Button object.

When you point to a green button with the mouse and
press the left mouse button, the green button appears to
become depressed into the surface of the Frame object in
a manner similar to a real Button object.

When you release the mouse button, the green button
appears to pop back out of the surface of the Frame
object.

This is accomplished using a simple drawRect() method.
The draw3DRect() method is not used in this case.

The 3D effect is an artifact of the manner in which the
rectangle is drawn onto an underlying Canvas object. The
reason that the 3D effect appears is explained in the
comments.

When you click on a green button, some text is displayed
on the standard output device to confirm that the click
```

```
occurred.

Some text is also displayed when you click on the real
Button object.

It may be significant to note that the fake green buttons
are much more responsive than the real Button when running
under Win95 on a machine with a 133MHz Pentium.  This can
be demonstrated by clicking repeatedly on one or the
other as fast as you can click. The real Button appears to
miss some of the clicks while the fake green buttons
appear to respond properly to every click.  At least the
fake green buttons responds to a much higher percentage of
clicks than the real Button.

These results were produced using JDK 1.1.3, running under
Windows 95.
*/
//=======================================================//
import java.awt.*;
import java.awt.event.*;

//=======================================================//
//This class is used to create a fake button object.

//Subclass Canvas to override the paint method in order
// to set the size and color of a Canvas object, to draw
// a rectangle which is partially on and partially off
// of the Canvas object, and to draw a caption on the
// Canvas object.

//The fact that the rectangle is only partially on the
// Canvas object causes the Canvas object to appear to
// protrude out of the screen, or to be depressed into the
// screen which produces the 3D effect.

//The rectangle is offset either up and to the left
// relative to the Canvas object or down and to the right.
// In either case, two lines on the border of the
// rectangle are lost.  The remaining two appear to be
// shadows and this causes the 3D effect.

class FakeButtonClass extends Canvas{
  boolean pushed = false;//indicates button pushed or not
  int width; //button width in pixels
  int height;//button height in pixels
  String caption;//caption for button

  public FakeButtonClass(//constructor
                     int width,int height,String caption){
    this.width = width;
    this.height = height;
    this.caption = caption;
    this.setBackground(Color.green);
    this.setSize(this.width,this.height);
  }//end constructor
```

```
  public void paint(Graphics g){
    //Determine offset and draw rectangle on Canvas object
    if(pushed) g.drawRect(1,1,width,height);
    else g.drawRect(-1,-1,width,height);

    //Draw the caption centered horizontally and slightly
    // above center vertically. For vertical placement,
    // use one-fourth the sum of leading, ascent, and
    // descent parameters of the font to calculate the
    // baseline position for the characters.
    int fontHeight = g.getFontMetrics().getHeight();
    int stringWidth =
                   g.getFontMetrics().stringWidth(caption);
    g.drawString(caption,
          (width-stringWidth)/2,(height/2)+(fontHeight/4));
  }//end paint()
}//end class FakeButtonClass

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

class Shapes02 extends Frame{//controlling class
  public static void main(String[] args){
    //instantiate an object of this type
    new Shapes02();
  }//end main

  public Shapes02(){//constructor
    this.setLayout(new FlowLayout());
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,150);

    //Instantiate two fake button objects of different
    // sizes and a real Button object
    FakeButtonClass firstFakeButton =
                        new FakeButtonClass(40,20,"Small");
    FakeButtonClass secondFakeButton =
                        new FakeButtonClass(80,40,"Large");
    Button myRealButton = new Button("Button");

    //Add the buttons to the Frame object
    this.add(firstFakeButton);
    this.add(myRealButton);
    this.add(secondFakeButton);

    this.setVisible(true);//make it all visible

    //Instantiate a mouse listener object and register it
    // to listen for mouse events on the two fake buttons
    MouseListenerClass myMouseListener =
                                    new MouseListenerClass();
    firstFakeButton.addMouseListener(myMouseListener);
    secondFakeButton.addMouseListener(myMouseListener);

    //Instantiate and register an action listener object
    // on the real Button object.
```

```
    myRealButton.addActionListener(
                               new MyActionListenerClass());

    //Instantiate and register a Listener object which will
    // terminate the program when the user closes the
    // Frame.
    WindowListenerClass myWindowListener =
                                  new WindowListenerClass();
    this.addWindowListener(myWindowListener);
  }//end constructor
}//end class Shapes02 definition
//=======================================================//

//This class is used to instantiate a listener object that
// listens for mouse events on the fake buttons and causes
// the visual rendering to switch between a protruding
// image and a "pushed" image. Also a message is displayed
// when the button is clicked.
class MouseListenerClass extends MouseAdapter{

  //Override the mousePressed() method
  public void mousePressed(MouseEvent e){
    //Set the "pushed" variable to true and repaint
    ((FakeButtonClass)e.getComponent()).pushed = true;
    e.getComponent().repaint();
  }//end mousePressed()

  //Override the mouseReleased() method
  public void mouseReleased(MouseEvent e){
    //Display a message
          System.out.println("mouseReleased");

    //Set the "pushed" variable to false and repaint
    ((FakeButtonClass)e.getComponent()).pushed = false;
    e.getComponent().repaint();
  }//end mouseReleased()
}//end class MouseListenerClass
//=======================================================//

//This class is used to instantiate a listener object that
// listens for action events on the real button and
// displays a message when it is clicked.
class MyActionListenerClass implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println("actionPerformed");//announce
  }//end actionPerformed()
}//end class MyActionListenerClass

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

//The following listener is used to terminate the program
// when the user closes the frame.
class WindowListenerClass extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
```

```
}//end class WindowListenerClass
//=======================================================//
```

.

Review

Q - Without viewing the solution that follow, write a Java application that replicates the functionality of the program named Shapes02 but uses the **drawLine()** method instead of the **drawRect()** method. Use double-width lines to emphasize the shadow effect. Also draw the caption with a slight shift when the button is pushed to create the optical illusion that the button has moved.

A - See the following solution

```
/*File SampProg141.java.java Copyright 1997, R.G.Baldwin
From lesson 160.

Without viewing the solution that follows, write a Java
application that replicates the functionality of the
application named Shapes02.java but uses the drawLine()
method in place of the drawRect() method.

Use double width lines to emphasize the shadow effect.

Also draw the caption with a slight shift when the button
is pushed to give the visual impression of motion.

This program illustrates the use of the Canvas class and
several of the methods of the Graphics class.

The program creates two fake buttons that appear (on some
systems at least) to be green 3D button objects.

When the program starts, two green fake buttons with
captions of "Small" and "Large" along with a real Button
with a caption of "Button" appear in a Frame object.

The green buttons appears to protrude slightly from the
surface of the Frame object in a manner similar to the
real Button object.

When you point to a green button with the mouse and
press the left mouse button, the green button appears to
become depressed into the surface of the Frame object in
a manner similar to a real Button object.

When you release the mouse button, the green button
appears to pop back out of the surface of the Frame
object.

The 3D effect is an artifact of the manner in which the
lines and the text are drawn onto an underlying Canvas
```

```
object. The reason that the 3D effect appears is explained
in the comments.

It may be significant to note that the fake green buttons
are much more responsive than the real Button when running
under Win95 on a machine with a 133MHz Pentium.  This can
be demonstrated by clicking repeatedly on one or the
other as fast as you can click. The real Button appears to
miss some of the clicks while the fake green buttons
appear to respond properly to every click.  At least the
fake green buttons responds to a much higher percentage of
clicks than the real Button.

These results were produced using JDK 1.1.3, running under
Windows 95.
*/
//=======================================================//
import java.awt.*;
import java.awt.event.*;

//=======================================================//
//This class is used to create a fake button object.

//Subclass Canvas to override the paint method in order
// to set the size and color of a Canvas object, to draw
// a series of lines on the Canvas object, and to draw a
// caption on the Canvas object.

//The lines are drawn so as to suggest shadows on the
// right and bottom of the button when it is not pushed,
// and to suggest shadows on the right and top of the
// button when it is pushed.

//The caption is drawn so as to move slightly down and to
// the right when the button is pushed to give the illusion
// of motion.

//The combination of the shadow effect and the motion
// effect produce the illusion of a 3D button being
// pressed into the screen.

class FakeButtonClass extends Canvas{
  boolean pushed = false;//indicates button pushed or not
  int width; //button width in pixels
  int height;//button height in pixels
  String caption;//caption for button

  public FakeButtonClass(//constructor
                    int width,int height,String caption){
    this.width = width;
    this.height = height;
    this.caption = caption;
    this.setBackground(Color.green);
    this.setSize(this.width,this.height);
  }//end constructor
```

```
  public void paint(Graphics g){
    //Draw the caption centered horizontally and slightly
    // above center vertically. For vertical placement,
    // use one-fourth the sum of leading, ascent, and
    // descent parameters of the font to calculate the
    // baseline position for the characters.
    int fontHeight = g.getFontMetrics().getHeight();
    int stringWidth =
                   g.getFontMetrics().stringWidth(caption);

    //Determine the state and draw corresponding lines on
    // Canvas object.
    //Also draw the caption and shift it slightly to give
    // the impression of motion when the button is pushed.
    if(pushed){
      g.drawLine(0,0,width,0);
      g.drawLine(0,1,width,1);

      g.drawLine(0,0,0,height);
      g.drawLine(1,0,1,height);

      //Note the one-pixel x and y offset in the following
      // statement to give the impression that the button
      // has moved.
      g.drawString(caption, 1+(width-stringWidth)/2,
                            1+((height/2)+(fontHeight/4)));
    }//end if
    else{
      //Note that a line drawn at height or width is off
      // the edge of the object and not visible.
      g.drawLine(0,height-1,width-1,height-1);
      g.drawLine(0,height-2,width-2,height-2);

      g.drawLine(width-1,0,width-1,height-1);
      g.drawLine(width-2,0,width-2,height-2);

      //Note that there is no motion offset in the
      // following statement.
      g.drawString(caption,
         (width-stringWidth)/2,(height/2)+(fontHeight/4));
    }//end else
  }//end paint()
}//end class FakeButtonClass

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

class SampProg141 extends Frame{//controlling class
  public static void main(String[] args){
    //instantiate an object of this type
    new SampProg141();
  }//end main

  public SampProg141(){//constructor
    this.setLayout(new FlowLayout());
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,150);
```

```
    //Instantiate two fake button objects of different
    // sizes and a real Button object
    FakeButtonClass firstFakeButton =
                        new FakeButtonClass(40,20,"Small");
    FakeButtonClass secondFakeButton =
                        new FakeButtonClass(80,40,"Large");
    Button myRealButton = new Button("Button");

    //Add the buttons to the Frame object
    this.add(firstFakeButton);
    this.add(myRealButton);
    this.add(secondFakeButton);

    this.setVisible(true);//make it all visible

    //Instantiate a mouse listener object and register it
    // to listen for mouse events on the two fake buttons
    MouseListenerClass myMouseListener =
                                  new MouseListenerClass();
    firstFakeButton.addMouseListener(myMouseListener);
    secondFakeButton.addMouseListener(myMouseListener);

    //Instantiate and register a Listener object which will
    // terminate the program when the user closes the
    // Frame.
    WindowListenerClass myWindowListener =
                                 new WindowListenerClass();
    this.addWindowListener(myWindowListener);
  }//end constructor
}//end class SampProg141 definition
//=========================================================//

//This class is used to instantiate a listener object that
// listens for mouse events on the fake buttons and causes
// the visual rendering to switch between a protruding
// image and a "pushed" image. Also a message is displayed
// when the button is clicked.
class MouseListenerClass extends MouseAdapter{

  //Override the mousePressed() method
  public void mousePressed(MouseEvent e){
    //Set the "pushed" variable to true and repaint
    ((FakeButtonClass)e.getComponent()).pushed = true;
    e.getComponent().repaint();
  }//end mousePressed()

  //Override the mouseReleased() method
  public void mouseReleased(MouseEvent e){
    //Set the "pushed" variable to false and repaint
    ((FakeButtonClass)e.getComponent()).pushed = false;
    e.getComponent().repaint();
  }//end mouseReleased()
}//end class MouseListenerClass

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

```
//The following listener is used to terminate the program
// when the user closes the frame.
class WindowListenerClass extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WindowListenerClass
//=======================================================//
```

Q - Without viewing the solution that follows, write a Java application that replicates the functionality of the application named SampProg141.java with the following differences:

Make the background color of the fake button to be a parameter in the constructor for the class.

Cause the background color of your fake buttons to be the same color as the background color of their container. For example, an RGB color of 0,128,0 is an appropriate color for this purpose.

Cause the highlighted edge of your fake button to be a brighter shade of the background color of the button.

Cause the shadow edge of your fake button to be a darker shade of the background color of the button. This produces a much more convincing illusion of a 3D button than previous versions where only the shadow was drawn and it was drawn in black.

A - See solution below.

```
/*File SampProg142.java.java Copyright 1997, R.G.Baldwin
From lesson 160.

Without viewing the solution that follows, write a Java
application that replicates the functionality of the
application named SampProg141.java with the following
differences

Make the background color of the fake button to be a
parameter in the constructor for the class.

Cause the background color of your fake buttons to be the
same color as the background color of their container.
For example, an RGB color of 0,128,0 is an appropriate
color for this purpose.

Cause the highlighted edge of your fake button to be a
brighter shade of the background color of the button.
Cause the shadow edge of your fake button to be a darker
shade of the background color of the button. This produces
a much more convincing illusion of a 3D button than
previous versions where only the shadow was drawn and it
was drawn in black.

The program creates two fake buttons that appear (on some
```

```
systems at least) to be green 3D button objects.

When the program starts, two green fake buttons with
captions of "Small" and "Large" along with a real Button
with a caption of "Button" appear in a Frame object.

The green buttons appears to protrude slightly from the
surface of the Frame object in a manner similar to the
real Button object.

When you point to a green button with the mouse and
press the left mouse button, the green button appears to
become depressed into the surface of the Frame object in
a manner similar to a real Button object.

When you release the mouse button, the green button
appears to pop back out of the surface of the Frame
object.

As in previous versions of this application, the fake green
buttons are much more responsive than the real Button when
running under Win95 on a machine with a 133MHz Pentium.

These results were produced using JDK 1.1.3, running under
Windows 95.
*/
//=======================================================//
import java.awt.*;
import java.awt.event.*;

//=======================================================//
//This class is used to create a fake button object.

//Subclass Canvas to override the paint method in order
// to set the size and color of a Canvas object, to draw
// a series of lines on the Canvas object, and to draw a
// caption on the Canvas object.

//The lines are drawn so as to suggest shadows and
// highlights on the edges of a rectangular button object.

//The caption is drawn so as to move slightly down and to
// the right when the button is pushed to give the illusion
// of motion.

//The highlights and the shadows and the motion effect
// combine to produce the illusion of a 3D button being
// pressed into the screen and released to pop out.

class FakeButtonClass extends Canvas{
  boolean pushed = false;//indicates button pushed or not
  int width; //button width in pixels
  int height;//button height in pixels
  String caption;//caption for button

  public FakeButtonClass(//constructor
```

```
          int width,int height,String caption,Color color){
    this.width = width;
    this.height = height;
    this.caption = caption;
    this.setBackground(color);
    this.setSize(this.width,this.height);
  }//end constructor

  public void paint(Graphics g){
    int fontHeight = g.getFontMetrics().getHeight();
    int stringWidth =
                   g.getFontMetrics().stringWidth(caption);

    //Determine the state and draw corresponding lines on
    // the Canvas object.
    //Use lines which are darker and lighter than the
    // background color of the object.
    //Also draw the caption and shift it slightly to give
    // the impression of motion when the button is pushed.
    if(pushed){
      //draw the shadow as dark green
      g.setColor(this.getBackground().darker());
      g.drawLine(0,0,width,0);
      g.drawLine(0,1,width,1);
      g.drawLine(0,0,0,height);
      g.drawLine(1,0,1,height);

      //draw the highlight as light green
      g.setColor(this.getBackground().brighter());
      g.drawLine(0,height-1,width-1,height-1);
      g.drawLine(0,height-2,width-2,height-2);
      g.drawLine(width-1,0,width-1,height-1);
      g.drawLine(width-2,0,width-2,height-2);

      //Draw the caption in black, centered horizontally
      // and slightly above center vertically.

      //Note the one-pixel x and y offset in the following
      // statement to give the impression that the button
      // has moved.
      g.setColor(Color.black);
      g.drawString(caption, 1+(width-stringWidth)/2,
                            1+((height/2)+(fontHeight/4)));
    }//end if
    else{
      //Draw the shadow in dark green
      //Note that a line drawn at height or width is off
      // the edge of the object and not visible.
      g.setColor(this.getBackground().darker());
      g.drawLine(0,height-1,width-1,height-1);
      g.drawLine(0,height-2,width-2,height-2);
      g.drawLine(width-1,0,width-1,height-1);
      g.drawLine(width-2,0,width-2,height-2);

      //Draw the highlight as light green
      g.setColor(this.getBackground().brighter());
```

```
      g.drawLine(0,0,width,0);
      g.drawLine(0,1,width,1);
      g.drawLine(0,0,0,height);
      g.drawLine(1,0,1,height);

      //Draw the caption in black, centered horizontally
      // and slightly above center vertically.
      //Note that there is no motion offset in the
      // following statement.
      g.setColor(Color.black);
      g.drawString(caption,
         (width-stringWidth)/2,(height/2)+(fontHeight/4));
    }//end else
  }//end paint()
}//end class FakeButtonClass

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

class SampProg142 extends Frame{//controlling class
  public static void main(String[] args){
    //instantiate an object of this type
    new SampProg142();
  }//end main

  public SampProg142(){//constructor
    this.setLayout(new FlowLayout());
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,150);
    this.setBackground(new Color(0,128,0));

    //Instantiate two fake button objects of different
    // sizes and a real Button object
    FakeButtonClass firstFakeButton =
                    new FakeButtonClass(
                        40,20,"Small",this.getBackground());
    FakeButtonClass secondFakeButton =
                    new FakeButtonClass(
                        80,40,"Large",this.getBackground());
    Button myRealButton = new Button("Button");

    //Add the buttons to the Frame object
    this.add(firstFakeButton);
    this.add(myRealButton);
    this.add(secondFakeButton);

    this.setVisible(true);//make it all visible

    //Instantiate a mouse listener object and register it
    // to listen for mouse events on the two fake buttons
    MouseListenerClass myMouseListener =
                                    new MouseListenerClass();
    firstFakeButton.addMouseListener(myMouseListener);
    secondFakeButton.addMouseListener(myMouseListener);

    //Instantiate and register a Listener object which will
    // terminate the program when the user closes the
```

```
    // Frame.
    WindowListenerClass myWindowListener =
                                  new WindowListenerClass();
    this.addWindowListener(myWindowListener);
  }//end constructor
}//end class SampProg142 definition
//=========================================================//

//This class is used to instantiate a listener object that
// listens for mouse events on the fake buttons and causes
// the visual rendering to switch between a protruding
// image and a "pushed" image. Also a message is displayed
// when the button is clicked.
class MouseListenerClass extends MouseAdapter{

  //Override the mousePressed() method
  public void mousePressed(MouseEvent e){
    //Set the "pushed" variable to true and repaint
    ((FakeButtonClass)e.getComponent()).pushed = true;
    e.getComponent().repaint();
  }//end mousePressed()

  //Override the mouseReleased() method
  public void mouseReleased(MouseEvent e){
    //Set the "pushed" variable to false and repaint
    ((FakeButtonClass)e.getComponent()).pushed = false;
    e.getComponent().repaint();
  }//end mouseReleased()
}//end class MouseListenerClass

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

//The following listener is used to terminate the program
// when the user closes the frame.
class WindowListenerClass extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WindowListenerClass
//=========================================================//
```

-end-