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

# The AWT Package, The Canvas Component

Java Programming, Lesson # 146, 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 series of lessons is concentrating on the package **java.awt** where most of the functionality exists for providing the user interface to your application or applet.

We learned earlier how to handle events and how to use layout managers.These two topics form the basis for the design and implementation of a Graphical User Interface (GUI).

Now we are learning about the various components that we can combine with layout and event handling to produce an effective Graphical User Interface.

The available components are defined by classes in the package **java.awt**. Our approach is to group those classes into categories and study the material on a category basis. As of this writing, the remaining categories are:

- **The Canvas Class**
- Graphics - Shapes, Fonts, Images
- The PrintJob Class
- The Toolkit Class

This lesson will concentrate on the **Canvas** class. As things develop, it may be necessary to modify these groupings.

## Canvas

The inheritance hierarchy for the **Canvas** class is as shown below.

```
java.lang.Object
        |
        +----java.awt.Component
                |
                +----java.awt.Canvas
```

According to the JavaSoft documentation:

"A **Canvas** component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.

An application must subclass the **Canvas** class in order to get useful functionality such as creating a custom component. The paint method must be overridden in order to perform custom graphics on the canvas."

According to Campione and Walrath:

"The **Canvas** class exists to be subclassed. It does nothing on its own; it merely provides a way for you to implement a custom Component. For example, Canvases are useful as display areas for images and custom graphics, whether or not you wish to handle events that occur within the display area."

An important caution provided by Campione and Walrath follows:

"When implementing a **Canvas** subclass, take care to implement the **minimumSize()** and **preferredSize()** methods to properly reflect your canvas's size. Otherwise, depending on the layout your canvas's container uses, your canvas could end up too small -- perhaps even invisible."

This particular caution wasn't important in the main sample program that follows because the layout manager being used (BorderLayout) ignores the two size parameters mentioned above and fills the available space in the *Center* of the layout with the **Canvas** object. However, there is another sample program at the end of this lesson that does deal with the size of the canvas. You should become familiar with both of these programs.

In summary, a **Canvas** object emulates its namesake in the art world. It provides a workspace on which you can draw or paint.

However, unlike its namesake in the art world, it also provides a surface on which you can recognize events.

## Canvas Class

The **Canvas** class is very simple consisting of a single constructor with no arguments and two methods.

The two methods of the class are as follows:

| | |
|---|---|
| **addNotify()** | Creates the peer of the canvas. |
| **paint(Graphics)** | This method is called to repaint this canvas. |

## Sample Program

This program is designed to illustrate the use of the **Canvas** class.

It also illustrates the ability to instantiate **listener** objects that can manipulate the **source** objects on which they are registered without the requirement to pass references to those source objects when the listener objects are instantiated.

Thus, no parameterized constructors are used in the instantiation of listener objects in this program.

When the program first appears on the screen, four non-functional**Button** objects and a green **Canvas** object appear in a **Frame** object. The objects are separated by horizontal and vertical gaps of thirty pixels. The **Button** objects are provided simply to cause the **Frame** object to contain something other than the **Canvas** object.

The four **Button** objects are placed at the borders of the **Frame** object using the **BorderLayout** manager. The **Canvas** object is placed in the *Center* of the **Frame** object. As mentioned earlier, the **BorderLayout** manager fills all the available space with the object in the *Center* and therefore, the preferredSizeand minimumSize parameters weren't a consideration in this program.

When you click on the green **Canvas** object, the coordinates of the mouse pointer are displayed on that object. If you have studied earlier lessons in this tutorial, you will recognize this program to be very similar to an earlier program where the coordinates of a mouse click were displayed in the client area of a **Frame** object. The main difference is that in the earlier program, the **Frame** object itself was the **source** of the **mouse** events, and in this program, a **Canvas** object that is placed in a **Frame** object is the **source** for **mouse** events.

No listener objects were registered for the **Frame** object or for any of the **Button** objects. Therefore, clicking in the gaps or clicking on the buttons has no effect on the program.

Clicking on the close button on the **Frame** object terminates the program and returns control to the operating system.

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

## Interesting Code Fragments

As mentioned above, this program is very similar to a program in a previous lesson which displays the coordinates of mouse clicks in the client area of a **Frame** object.

The first interesting code fragment in this version of the program is the code in which we subclass the **Canvas** class in order to override the **Paint** method, and also to cause the object to be green when it is first instantiated.

```
class MyCanvas extends Canvas{
  int clickX;
  int clickY;

  public MyCanvas(){//constructor
    this.setBackground(Color.green);
  }//end constructor

  //Override the paint() method.
  public void paint(Graphics g){
    g.drawString(
              "" + clickX + ", " + clickY, clickX, clickY);
  }//end paint()
}//end class MyCanvas
```

The next interesting code fragment is the code used to create a **BorderLayout** manager,with horizontal and vertical gaps, for the **Frame** object. Although the default layout manager for **Frame** is **BorderLayout,** gaps are not included in the default. If gaps are desired, it is necessary to use a named layout manager on which to invoke the methods to set the gaps.

```
    //Create a border layout with gaps.
    BorderLayout myLayout = new BorderLayout();
    myLayout.setVgap(30);
    myLayout.setHgap(30);
    this.setLayout(myLayout);//Apply layout to the Frame object
```

The next interesting code fragment instantiates the **Canvas** object from the class named **MyCanvas** that extends **Canvas** and adds it to the center position of the **Frame** object. The **MyCanvas** object is not instantiated as an anonymous object. Rather, it is instantiated as a named object so that a mouse listener object can be registered on it later.

```
     MyCanvas myCanvasObject = new MyCanvas();
     this.add(myCanvasObject,"Center");
```

This is followed by some code that you have seen many times in the past to add some buttons to the **Frame** object and make the composite of all the objects visible. There is also some code to register a **WindowListener** object to terminate the program when the user clicks the close button on the **Frame** object. That code is not shown here.

The next interesting code fragment instantiates and registers a Listener object which will process mouse events to determine and display the coordinates when the user presses the mouse button on the **MyCanvas** object.

Note that the Listener object is instantiated anonymously and no reference to the **MyCanvas** object is passed to the constructor for the **Listener** object. Therefore, the listener object must identify the component on which to display coordinate information from within its own code. We will see later that the identification is based on the **MouseEvent** object that is passed to the event listener when the event occurs.

```
    myCanvasObject.addMouseListener(new MouseProc());
```

The final interesting code fragment is the definition of the listener class that displays the coordinates of the mouse pointer when the left mouse button is pressed on an object for which an object of the class is registered.

This version uses the **getComponent()** method on the incoming **MouseEvent** object to identify the component that was the **source** of the event. This method returns a reference to an object of type **Component** which must be downcast to type **MyCanvas** before it can be used to access the instance variables of an object of type **MyCanvas**.

```
class MouseProc extends MouseAdapter{
  //Override the mousePressed method
  public void mousePressed(MouseEvent e){
    //Get x and y coordinates of the mouse pointer and
    // store in the instance variables of the MyCanvas
    // object.
    ((MyCanvas)e.getComponent()).clickX = e.getX();
    ((MyCanvas)e.getComponent()).clickY = e.getY();
    //display coordinate information
    e.getComponent().repaint();

  }//end mousePressed()
}//end class MouseProc
```

The remaining code in this program is code that you have seen many times in the past and therefore will not be highlighted in this section.

## Program Listing

This section contains a complete listing of the program.

```
/*File Canvas01.java Copyright 1997, R.G.Baldwin
Illustrates the use of the Canvas class.

Also illustrates the ability to instantiate listener
objects that can manipulate the source objects on which
they are registered without the requirement to pass
references to those source objects when the listener
objects are instantiated.

No parameterized constructors are used in the
instantiation of listener objects in this program.

When the program first appears on the screen, four non-
functional buttons and a green Canvas object appear in a
Frame object.  The objects are separated by horizontal and
vertical gaps of thirty pixels.

The four buttons are placed at the borders of the Frame
object using the BorderLayout manager.  The Canvas object
is placed in the Center of the Frame object.

When you click on the green Canvas object, the coordinates
of the mouse pointer are displayed.

Clicking in the gaps or clicking on the buttons has no
effect on the program.

Clicking on the close button on the Frame object
terminates the program and returns control to the
operating system.

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

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

//Subclass Canvas in order to override the paint method
// and to make it green when instantiated.
class MyCanvas extends Canvas{
  int clickX;
  int clickY;

  public MyCanvas(){//constructor
    this.setBackground(Color.green);
  }//end constructor

  //Override the paint() method.
  public void paint(Graphics g){
    g.drawString(
              "" + clickX + ", " + clickY, clickX, clickY);
  }//end paint()
}//end class MyCanvas
```

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

class Canvas01 extends Frame{//controlling class
  public static void main(String[] args){
    //instantiate an object of this type
    new Canvas01();
  }//end main

  public Canvas01(){//constructor

    //Create a border layout with gaps.
    BorderLayout myLayout = new BorderLayout();
    myLayout.setVgap(30);
    myLayout.setHgap(30);

    this.setLayout(myLayout);//Apply layout to the Frame
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,300);

    //Instantiate a green customized Canvas object
    MyCanvas myCanvasObject = new MyCanvas();

    //Add the MyCanvas object to the center of the
    // Frame object.
    this.add(myCanvasObject,"Center");

    //Add four nonfunctional buttons to the borders
    // of the Frame object

    this.add(new Button("North"),"North");
    this.add(new Button("South"),"South");
    this.add(new Button("East"),"East");
    this.add(new Button("West"),"West");

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

    //Instantiate and register Listener object which will
    // terminate the program when the user closes the
    // Frame.
    WProc1 winProcCmd1 = new WProc1();
    this.addWindowListener(winProcCmd1);

    //Instantiate and register Listener object which will
    // process mouse events to determine and display the
    // coordinates when the user presses the mouse button
    // on the MyCanvas object.

    // Note that the Listener object is instantiated
    // anonymously and no reference to the MyCanvas object
    // is passed to the constructor for the Listener
    // object.

    myCanvasObject.addMouseListener(new MouseProc());

  }//end constructor
}//end class Canvas01 definition
```

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

//This listener class monitors for mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed on the object for which it is
// registered.
class MouseProc extends MouseAdapter{
  //Override the mousePressed method
  public void mousePressed(MouseEvent e){
    //Get x and y coordinates of the mouse pointer and
    // store in the instance variables of the MyCanvas
    // object.  Note the requirement to cast the component
    // to the type of MyCanvas in order to access the
    // instance variables.
    ((MyCanvas)e.getComponent()).clickX = e.getX();
    ((MyCanvas)e.getComponent()).clickY = e.getY();
    //display coordinate information
    e.getComponent().repaint();

  }//end mousePressed()
}//end class MouseProc
//=========================================================//

//The following listener is used to terminate the program
// when the user closes the frame.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=========================================================//
```

.

## Review

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

A - See the specifications and the solution below.

```
/*File SampProg140.java Copyright 1997, R.G.Baldwin
Illustrates the use of the Canvas class along with the
requirement to establish the size of a Canvas object
for use with certain types of layout managers.

Also illustrates the ability to instantiate listener
objects that can manipulate the source objects on which
they are registered without the requirement to pass
references to those source objects when the listener
objects are instantiated.

No parameterized constructors are used in the
```

```
instantiation of listener objects in this program.

When the program first appears on the screen, two green
Canvas objects appear in a Frame object. The Frame object
is approximately 300 by 300 pixels in size.

One of the green Canvas objects is 100 pixels wide and 150
pixels in height.  The other green Canvas object is 50
pixels wide and 150 pixels in height.

The green Canvas objects  are separated by a horizontal gap
of 30 pixels when they are side-by-side and are separated
by a vertical gap of 30 pixels when they are arranged in
a column.

When you click on either of the green Canvas objects, the
coordinates of the mouse pointer are displayed near the
mouse pointer.

Clicking in the gaps between the Canvas objects, or
clicking in the other area outside the Canvas objects has
no effect on the program.

Clicking on the close button on the Frame object
terminates the program and returns control to the
operating system.

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

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

//Subclass Canvas in order to override the paint method
// and to set the size and color of the Canvas object
// when it is instantiated.
class MyCanvas extends Canvas{
  int clickX;
  int clickY;

  public MyCanvas(int width, int height){//constructor
    this.setBackground(Color.green);
    this.setSize(width,height);
  }//end constructor

  //Override the paint() method to display coordinate data
  public void paint(Graphics g){
    g.drawString(
              "" + clickX + ", " + clickY, clickX, clickY);
  }//end paint()
}//end class MyCanvas
//=========================================================//
```

```
class SampProg140 extends Frame{//controlling class
  public static void main(String[] args){
    //instantiate an object of this type
    new SampProg140();
  }//end main

  public SampProg140(){//constructor

    //Create a flow layout with gaps.
    FlowLayout myLayout = new FlowLayout();
    myLayout.setVgap(30);
    myLayout.setHgap(30);

    this.setLayout(myLayout);//Apply layout to the Frame
    this.setTitle("Copyright 1997, R.G.Baldwin");
    this.setSize(300,300);

    //Instantiate two green customized Canvas objects
    MyCanvas oneCanvasObject = new MyCanvas(100,150);
    MyCanvas anotherCanvasObject = new MyCanvas(50,100);

    //Add the MyCanvas objects to the Frame object.
    this.add(oneCanvasObject);
    this.add(anotherCanvasObject);

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

    //Instantiate and register Listener object which will
    // terminate the program when the user closes the
    // Frame.
    WProc1 winProcCmd1 = new WProc1();
    this.addWindowListener(winProcCmd1);

    //Instantiate and register Listener objects which will
    // process mouse events to determine and display the
    // coordinates when the user presses the mouse button
    // on either MyCanvas object.

    // Note that the Listener objects are instantiated
    // anonymously and no reference to the MyCanvas object
    // is passed to the constructors for the Listener
    // objects.

    oneCanvasObject.addMouseListener(new MouseProc());
    anotherCanvasObject.addMouseListener(new MouseProc());

  }//end constructor
}//end class SampProg140 definition
//=========================================================//

//This listener class responds to mouse presses and
// displays the coordinates of the mouse pointer when the
// mouse is pressed on the object for which it is
// registered.
class MouseProc extends MouseAdapter{
  //Override the mousePressed method
  public void mousePressed(MouseEvent e){
    //Get x and y coordinates of the mouse pointer and
    // store in the instance variables of the MyCanvas
    // object.  Note the requirement to cast the component
    // to the type of MyCanvas in order to access the
    // instance variables.
    ((MyCanvas)e.getComponent()).clickX = e.getX();
    ((MyCanvas)e.getComponent()).clickY = e.getY();
    //display coordinate information
    e.getComponent().repaint();

  }//end mousePressed()
}//end class MouseProc
//=========================================================//

//The following listener is used to terminate the program
// when the user closes the frame.
class WProc1 extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);
  }//end windowClosing()
}//end class WProc1
//=========================================================//
```

-end-