Java 2D Graphics, The Color Constructors and Transparency

by Richard G. Baldwin
baldwin@austin.cc.tx.us

Java Programming, Lecture Notes # 324

March 21, 2000

- Introduction
- What is the New Color Class?
- Sample Program
- Summary
- Complete Program Listing

Introduction

In an earlier lesson, I explained that the **Graphics2D** class extends the **Graphics** class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

Beginning with JDK 1.2, **Graphics2D** is the fundamental class for rendering two-dimensional shapes, text and images.

You also need to understand some other classes and interfaces

I also explained that without understanding the behavior of other classes and interfaces, it is not possible to fully understand the inner workings of the **Graphics2D** class.

Throughout this series of lessons, I have been providing you with information and sample programs designed to help you understand the various classes and interfaces that are necessary for an understanding of the **Graphics2D** class.

Two ways to achieve transparency

There are at least two different ways to achieve transparency in Java 2D. One approach is to use new constructors for the **Color** class that allow you to create solid colors with a specified degree of transparency. I will discuss that approach in this lesson.

A more general approach

A second, and possibly more general approach is to make use of an object that implements the **Composite** interface, passing a reference to that object to the **setComposite()** method of the **Graphics2D** class.

Earlier lessons explained the use of the **Composite** interface for solid colors as well as for color gradients

What is the New Color Class?

Here is part of what Sun has to say about the new **Color** class supported by Java2D.

"A class to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.

Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor. When constructing a Color with an explicit alpha or getting the color/alpha components of a Color, the color components are never premultiplied by the alpha component.
...
Eventually this class should supersede java.awt.Color, but for now it is defined to be part of the java.java2d package, so we can generate documentation for a single package for review."

Overloaded constructors

This class provides several overloaded constructors that allow you to provide an explicit alpha.

Sun's description of one of those constructors follows. This is the constructor that will be used in the sample program in this lesson.

public Color(
float r, float g,float b,float a)

Creates an sRGB color with the specified red, green, blue, and alpha values in the range (0.0 - 1.0). The actual color used in rendering will

depend on finding the best match given the color space available for a given output device.

Parameters:

- r - the red component
- g - the green component
- b - the blue component
- a - the alpha component

How is transparancy determined?

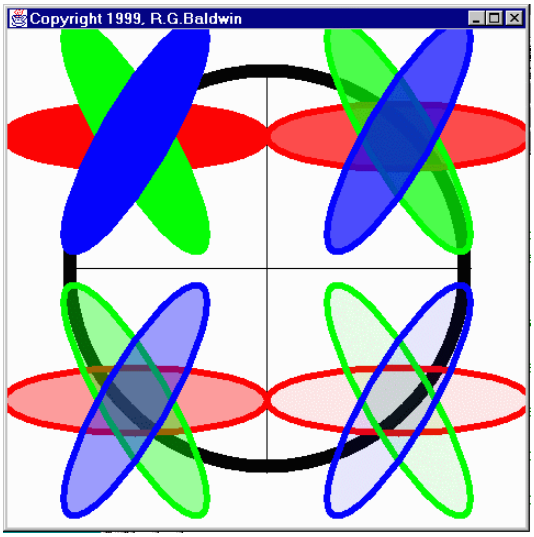
The value of alpha determines transparency with a value of **1.0f** being opaque, and **0.0f** being completely transparent.

Sample Program

This program is named **Composite03**. You will need to compile and execute the program so that you can view its output while reading the discussion. Otherwise, without being able to view the output, the discussion will probably mean very little to you.

A screen shot of the output

In case you are unable to compile and execute the program, a screen shot of the output follows. Note, however, that this screen shot was reduced to about seventy-percent of its original size in pixels, so some of the detail has been lost.



The GUI is a Frame object

The program draws a four-inch by four-inch **Frame** on the screen. It translates the origin to the center of the **Frame**. Then it draws a pair of X and Y-axes centered on the new origin.

A large circle

After drawing the X and Y-axes, the program draws a circle with a thick border centered on the origin. This circle is used later to provide visual cues relative to transparency.

Transparent ellipses

After the large circle is drawn, three ellipses are drawn on top of one another in each quadrant. Each ellipse has a common center, and is rotated by sixty degrees relative to the ellipse beneath it. The color and transparency of each ellipse is established using the constructor described above.

Red on the bottom

A red ellipse is on the bottom of each stack and a blue ellipse is on the top of each stack. A green ellipse is sandwiched between the other two.

Different transparency values

The different ellipses are given various transparency values in the different quadrants to illustrate the effect of the alpha parameter of the **setComposite()** method.

An opaque border

Each ellipse is given an opaque border, which makes it easier to visually discern the stacking order of the transparent versions of the ellipses.

Transparency by quadrant

Here is the transparency given to each of the ellipses in the different quadrants.

```
TRANSPARENCY
Upper-left quadrant
No transparency

Upper-right quadrant
All three ellipses are 30-
percent transparent

Lower-left quadrant
All three ellipses are 60-
percent transparent

Lower-right quadrant
```

```
All three ellipses are 90-
percent transparent
```

None are opaque

Unlike previous lessons with similar sample programs, none of the ellipses are opaque in all four quadrants. As a result, the large black circle shows through all three ellipses in all quadrants except the upper-left quadrant.

Upper-left quadrant

All three ellipses are opaque in the upper-left quadrant, so nothing shows through.

Other ellipses become transparent

All three ellipses are made progressively more transparent as you move through the other three quadrants. As a result, you can "see through" all three ellipses. In other words, you can see the geometric figures that lie beneath them (the other ellipses and the large black circle).

Similar to previous lesson

The material in this lesson is very similar to previous lessons except for the use of the **Color** constructor as an alternate to the **setComposite()** method to achieve transparency. Therefore, I am not going to discuss the output of this program in detail.

Illustrates rotation and translation

As mentioned in the earlier lesson, this lesson also provides a good illustration of the benefits of rotation and translation. The task of rotating the ellipses relative to each other and the task of translating them into the various quadrants was made much easier (even possible) through the use of the **AffineTransform** to rotate and translate the ellipses.

The normal caveat regarding inches

This discussion of dimensions in inches on the screen depends on the method named **getScreenResolution()** returning the correct value. However, the **getScreenResolution()** method always seems to return 120 on my computer regardless of the actual screen resolution settings.

Will discuss in fragments

I will briefly discuss this program in fragments. The controlling class and the constructor for the GUI class are essentially the same as you have seen in several previous lessons, so, I won't repeat that discussion here. You can view that material in the complete listing of the program at the end of the lesson.

All of the interesting action takes place in the overridden **paint()** method, so I will begin the discussion there.

Overridden paint() method

The beginning portions of the overridden **paint()** method should be familiar to you by now as well. So, I am going to let the comments in Figure 1 speak for themselves.

```
public void paint(Graphics g){
  //Downcast the Graphics object to a
  // Graphics2D object
  Graphics2D g2 = (Graphics2D)g;

  //Scale device space to produce inches on
  // the screen based on actual screen
  // resolution.
  g2.scale((double)res/72,(double)res/72);

  //Translate origin to center of Frame
  g2.translate((hSize/2)*ds,(vSize/2)*ds);

  //Draw x-axis
  g2.draw(new Line2D.Double(
                    -1.5*ds,0.0,1.5*ds,0.0));
  //Draw y-axis
  g2.draw(new Line2D.Double(
                    0.0,-1.5*ds,0.0,1.5*ds));

  //Draw a big circle underneath all of the
  // ellipses.
  g2.setStroke(new BasicStroke(0.1f*ds));
  Ellipse2D.Double bigCircle =
           new Ellipse2D.Double(
               -1.5*ds,-1.5*ds,3.0*ds,3.0*ds);
  g2.draw(bigCircle);

  //Declare a reference variable for the ellipses
  Ellipse2D.Double theEllipse;
```

Figure 1

Setting the Stroke

Figure 2 sets the **Stroke** to 0.05 inches. This will cause each ellipse drawn following this fragment to have a border of that width.

```
g2.setStroke(new BasicStroke(0.05f*ds));
```

Figure 2

I discussed the use of **setStroke()** and **BasicStroke** in an earlier lesson, so I won't discuss it further here.

Translation

Figure 3 translates the origin to the center of what was previously the upper-left quadrant. After this statement is executed, any geometric figure that is drawn centered on the origin will actually be rendered in the center of what was previously the upper-left quadrant.

```
g2.translate(-1.0*ds,-1.0*ds);
```

Figure 3

This should also be "old stuff" to you by now.

Opaque red-to-green ellipse outline

Figure 4 draws an opaque outline of a red ellipse using the **Stroke** object instantiated earlier. This ellipse is centered on the new origin.

```
theEllipse = new Ellipse2D.Double(
          -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
 //draw nontransparent outline
 g2.setPaint(Color.red);
 g2.draw(theEllipse);
```

Figure 4

The code has been covered in previous lessons, so I won't discuss it further here.

Fill the ellipse with an opaque solid red color

Figure 5 contains the material that is new to this lesson. This fragment instantiates a new **Color** object with an alpha value of **1.0f**, which is the alpha value for opaque as described above.

```
g2.setPaint(new Color(
    1.0f,0.0f,0.0f,1.0f));//red, not transparent
g2.fill(theEllipse);
```

Figure 5

The parameters to the constructor also specify that the color will be pure red with no contribution from either green or blue.

The **fill()** method is then used to fill the ellipse according to the new **Color** object that is passed to the **setPaint()** method before invoking **fill()**. Except for the use of the **Color** constructor that includes an alpha parameter, there is nothing new here.

Skip to upper-right quadrant

Because much of the code in this lesson is very similar to code that was explained in previous lessons, I am going to skip ahead to the code that establishes the fill colors and transparency values for the ellipses in the upper-right quadrant. (You can view all of the code in the complete listing of the program at the end of the lesson.)

Set fill color and transparency

In addition, for brevity, I am going to delete some of the code having to do with that quadrant. Figure 6 shows only the code required to establish the fill colors and transparency values for each of the ellipses in that quadrant.

```
g2.setPaint(new Color(
   1.0f,0.0f,0.0f,0.7f));//red, 30% transparent
g2.fill(theEllipse);

    //...

g2.setPaint(new Color(
  0.0f,1.0f,0.0f,0.7f));//green,30% transparent
g2.fill(theEllipse);

    //...

g2.setPaint(new Color(
   0.0f,0.0f,1.0f,0.7f));//blue, 30% transparent
g2.fill(theEllipse);
```

Figure 6

Percent opaque versus percent transparent

You may prefer to think of the transparency specified by an alpha value of **0.7f** as representing 70-percent opaque instead of 30-percent transparent.

Summary

In this lesson, I have shown you how to use one of the new constructors in the **Color** class to establish the fill colors and transparency values of several ellipses.

Note that for solid colors, this approach is an alternative to the use of the setComposite**()** method of the **Grapics2D** class (along with the **AlphaComposite** class) to control the manner in which new pixel values are composited with existing pixel values.

Complete Program Listing

A complete listing of the program is provided in Figure 7.

```
/*Composite03.java 12/12/99
 Copyright 1999, R.G.Baldwin

Illustrates use of the constructors of the Color class
to achieve transparency with solid-fill colors.

Similar to Composite01 except that Composite01
uses the  AlphaComposite class to achieve
transparency.

Draws a 4-inch by 4-inch Frame on the screen.

Translates the orgin to the center of the Frame.

Draws a pair of X and Y-axes centered on the new
origin.

Draws a big circle centered on the origin underneath
all of the ellipses.

Uses rotation and translation to fill three ellipses in
each of the four quadrants.  The ellipses intersect at
their center.  Each is rotated by 60 degrees relative
to the one below it.  The order is:
  Red ellipse on the bottom
  Green ellipse in the middle
  Blue ellipse on the top

Each ellipse has a non-transparent outline,
approximately 0.05 inches in width.

TRANSPARENCY
Upper-left quadrant
No transparency

Upper-right quadrant
All three ellipses are 30-percent transparent

Lower-left quadrant
All three ellipses are 60-percent transparent

Lower-right quadrant
All three ellipses are 90-percent transparent

Whether the dimensions in inches come out right
or not depends on whether the method
getScreenResolution() returns the correct resolution
for your screen.

Tested using JDK 1.2.2 under WinNT Workstation 4.0
**************************************/
import java.awt.geom.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

class Composite03{
```

```
  publicstaticvoid main(String[] args){
    GUI guiObj = new GUI();
  }//end main
}//end controlling class Composite03

class GUI extends Frame{
  int res;//store screen resolution here
  staticfinalint ds = 72;//default scale, 72 units/inch
  staticfinalint hSize = 4;//horizonal size = 4 inches
  staticfinalint vSize = 4;//vertical size = 4 inches

  GUI(){//constructor
    //Get screen resolution
    res = Toolkit.getDefaultToolkit().
                                getScreenResolution();
    //Set Frame size
    this.setSize(hSize*res,vSize*res);
    this.setVisible(true);
    this.setTitle("Copyright 1999, R.G.Baldwin");

    //Window listener to terminate program.
    this.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e){
        System.exit(0);}});
  }//end constructor
  //-------------------------------------------------------//

  //Override the paint() method
  public void paint(Graphics g){
    //Downcast the Graphics object to a
    // Graphics2D object
    Graphics2D g2 = (Graphics2D)g;

    //Scale device space to produce inches on the
    // screen based on actual screen resolution.
    g2.scale((double)res/72,(double)res/72);

    //Translate origin to center of Frame
    g2.translate((hSize/2)*ds,(vSize/2)*ds);

    //Draw x-axis
    g2.draw(new Line2D.Double(
                                -1.5*ds,0.0,1.5*ds,0.0));
    //Draw y-axis
    g2.draw(new Line2D.Double(
                                0.0,-1.5*ds,0.0,1.5*ds));

    //Draw a big circle underneath all of the ellipses.
    g2.setStroke(new BasicStroke(0.1f*ds));
    Ellipse2D.Double bigCircle = new Ellipse2D.Double(
                       -1.5*ds,-1.5*ds,3.0*ds,3.0*ds);
    g2.draw(bigCircle);

    Ellipse2D.Double theEllipse;
    g2.setStroke(new BasicStroke(0.05f*ds));

    //Translate origin to upper-left quadrant
    g2.translate(-1.0*ds,-1.0*ds);

    //Red horizontal ellipse
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);

    g2.setPaint(Color.red);//nontransparent outline
    g2.draw(theEllipse);

    g2.setPaint(new Color(
            1.0f,0.0f,0.0f,1.0f));//red, not transparent
    g2.fill(theEllipse);
```

```
//Green ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                    -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees

g2.setPaint(Color.green);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(Color.green);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
       0.0f,1.0f,0.0f,1.0f));//green, not transparent
g2.fill(theEllipse);

//Blue ellipse at 120 degrees
theEllipse = new Ellipse2D.Double(
                    -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate((Math.PI/3.0));//rotate 60 more degrees

g2.setPaint(Color.blue);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
        0.0f,0.0f,1.0f,1.0f));//blue, not transparent
g2.fill(theEllipse);


//Translate origin to upper-right quadrant
g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
g2.translate(2.0*ds,0.0*ds);

//Red horizontal ellipse
theEllipse = new Ellipse2D.Double(
                    -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);

g2.setPaint(Color.red);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
         1.0f,0.0f,0.0f,0.7f));//red, 30% transparent
g2.fill(theEllipse);

//Green ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                    -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees

g2.setPaint(Color.green);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
       0.0f,1.0f,0.0f,0.7f));//green,30% transparent
g2.fill(theEllipse);

//Blue ellipse at 120 degrees
theEllipse = new Ellipse2D.Double(
                    -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate((Math.PI/3.0));//rotate 60 more degrees

g2.setPaint(Color.blue);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
        0.0f,0.0f,1.0f,0.7f));//blue, 30% transparent
g2.fill(theEllipse);


//Translate origin to lower-left quadrant
g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
```

```
g2.translate(-2.0*ds,2.0*ds);

//Red horizontal ellipse
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);

g2.setPaint(Color.red);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
        1.0f,0.0f,0.0f,0.4f));//red, 60% transparent
g2.fill(theEllipse);

//Green ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees

g2.setPaint(Color.green);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
       0.0f,1.0f,0.0f,0.4f));//green,60% transparent
g2.fill(theEllipse);

//Blue ellipse at 120 degrees
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate((Math.PI/3.0));//rotate 60 more degrees

g2.setPaint(Color.blue);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
        0.0f,0.0f,1.0f,0.4f));//blue, 60% transparent
g2.fill(theEllipse);


//Translate origin to lower-right quadrant
g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
g2.translate(2.0*ds,0.0*ds);

//Red horizontal ellipse
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);

g2.setPaint(Color.red);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
        1.0f,0.0f,0.0f,0.1f));//red, 90% transparent
g2.fill(theEllipse);

//Green ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees

g2.setPaint(Color.green);//nontransparent outline
g2.draw(theEllipse);

g2.setPaint(new Color(
       0.0f,1.0f,0.0f,0.1f));//green,90% transparent
g2.fill(theEllipse);

//Blue ellipse at 120 degrees
theEllipse = new Ellipse2D.Double(
                  -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate((Math.PI/3.0));//rotate 60 more degrees
```

```
    g2.setPaint(Color.blue);//nontransparent outline
    g2.draw(theEllipse);

    g2.setPaint(new Color(
            0.0f,0.0f,1.0f,0.1f));//blue, 90% transparent
    g2.fill(theEllipse);

  }//end overridden paint()

 }//end class GUI
 //================================//
```

Figure 7

About the author

Richard Baldwin *is a college professor and private consultant whose primary focus is a combination of Java and XML. In addition to the many platform-independent benefits of Java applications, he believes that a combination of Java and XML will become the primary driving force in the delivery of structured information on the Web.*

Richard has participated in numerous consulting projects involving Java, XML, or a combination of the two. He frequently provides onsite Java and/or XML training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Java Programming Tutorials, which has gained a worldwide following among experienced and aspiring Java programmers. He has also published articles on Java Programming in Java Pro magazine.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

-end-