Translucent Window in Java 7

Published on January 17, 2012

Let us define three terms - transparent, translucent, and opaque, before we discuss how to use a translucent window in Swing. If something is transparent, you can see through it. For example, clear water is transparent. If something is opaque, you cannot see through it. For example, a concrete wall is opaque. If something is translucent, you can see through it, but not clearly. If something is translucent, it partially allows light to pass through. For example, a plastic curtain is translucent. The terms, “transparent” and “opaque”, describe two opposite states, whereas the term “translucent” describes a state between “transparent” and “opaque”.

Java 7 lets you define the degree of translucency of a window, e.g., a `JFrame`. A 90% translucent window is 10% opaque. The degree of translucency for a window can be defined using the alpha value of the color component for a pixel. You can define the alpha value of a color using the constructors of the `Color` class as shown below.

```
Color(int red, int green, int blue, int alpha)
Color(float red, float green, float blue, float alpha)
```

The value for the `alpha` argument is specified between 0 and 255, when the color components are specified in terms of `int` values. For the `float` type arguments, its value is between 0.0 and 1.0. The `alpha` value of 0 or 0.0 means transparent (100% translucent or 0% opaque). The `alpha` value of 255 or 1.0 means opaque (0% translucent or not transparent).

Java 7 supports three forms of translucency in a window, which are represented by the following three constants in the `WindowTranslucency` enum.

- `PERPIXEL_TRANSPARENT`: In this form of translucency, a pixel in a window is either opaque or transparent. That is, the alpha value for a pixel is either 0.0 or 1.0.
- `TRANSLUCENT`: In this form of translucency, all pixels in a window have the same translucency, which can be defined by an alpha value between 0.0 and 1.0.
- `PERPIXEL_TRANSLUCENT`: In this form of translucency, each pixel in a window can have its own alpha value between 0.0 and 1.0. This form lets you define the translucency in a window on a per pixel basis.

Not all platforms support all the three forms of translucency in a window. You must check for the supported forms of translucency in your program before using them. Otherwise, your code may throw an `UnsupportedOperationException`. The `isWindowTranslucencySupported()` method of the `GraphicsDevice` class lets you check the forms of translucency that are supported

on a platform. Listing 1 demonstrates how to check for translucency support on a platform. The code in this listing is short and self-explanatory.

TIP
Using a translucent window on a platform is dependent on the form of translucency supported by the platform. Before using a specific form of translucency, you must check if that form of translucency is supported on the platform. Otherwise, your program may run fine on one platform and generate errors on another. This check is omitted in the examples to keep the code shorter.

Listing 1: Checking for a translucency support on a platform

```
// TranslucencySupport.java
package com.jdojo.chapter1;

import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import static java.awt.GraphicsDevice.WindowTranslucency.*;

public class TranslucencySupport {
    public static void main(String[] args) {
        GraphicsEnvironment graphicsEnv =
                GraphicsEnvironment.getLocalGraphicsEnvironment();

        GraphicsDevice graphicsDevice =
                graphicsEnv.getDefaultScreenDevice();

        // Print the translucency supported by the platform
        boolean isSupported =
                graphicsDevice.isWindowTranslucencySupported(
                                 PERPIXEL_TRANSPARENT);
        System.out.println("PERPIXEL_TRANSPARENT supported:" +
                           isSupported);

        isSupported =
            graphicsDevice.isWindowTranslucencySupported(TRANSLUCENT);
        System.out.println("TRANSLUCENT supported:" + isSupported);

        isSupported = graphicsDevice.isWindowTranslucencySupported(
                                      PERPIXEL_TRANSLUCENT);
        System.out.println("PERPIXEL_TRANSLUCENT supported:" +
                           isSupported);
    }
}
```

Let us see a uniform translucent `JFrame` in action. You can set the translucency of a `JFrame` using the `setOpacity(float opacity)` method. The value for the specified `opacity` must be

between 0.0f and 1.0f. Before you call this method on a window, the following three conditions must be met.

- The platform must support the `TRANSLUCENT` translucency. You can use the logic from Listing 1 to check if the `TRANSLUCENT` translucency is supported by the platform.
- The window must be undecorated. You can make a `JFrame` or `JDialog` undecorated by calling the `setUndecorated(false)` method on them.
- The window must not be in full-screen mode. You can put a window in full-screen mode using the `setFullScreenWindow(Window w)` method of the `GraphicsDevice` class.

If not all of the above-mentioned conditions are met, setting the opacity of a window other than 1.0f throws an `IllegalComponentStateException`. Listing 2 demonstrates how to use a uniform translucent `JFrame`. The following two statements in the `initFrame()` method in the listing is of interest to get a translucent `JFrame`. The first statement makes sure that the frame is undecorated, and the second one sets the translucency of the frame in terms of opacity.

```
// Make sure the frame is undecorated
this.setUndecorated(true);

// Set 40% opacity. That is, 60% translucency.
this.setOpacity(0.40f);
```

When you run this program, you can see the contents on your screen through the `JFrame` display area. A Close button is added to the frame to close the frame at runtime.

Listing 2: Using a uniform translucent JFrame

```
// UniformTranslucentFrame.java
package com.jdojo.chapter1;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class UniformTranslucentFrame extends JFrame {
     private JButton closeButton = new JButton("Close");

     public UniformTranslucentFrame(String title) {
          super(title);
          initFrame();
     }

     public void initFrame() {
          this.setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
        // Make sure the frame is undecorated
        this.setUndecorated(true);

        // Set 40% opacity. That is, 60% translucency.
        this.setOpacity(0.40f);

        // Set its size
        this.setSize(200, 200);

        // Center it on the screen
        this.setLocationRelativeTo(null);

        // Add a button to close the window
        this.add(closeButton, BorderLayout.SOUTH);

        closeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        UniformTranslucentFrame frame =
                new UniformTranslucentFrame("Translucent Frame");
        frame.setVisible(true);
    }
}
```

Let us see a per-pixel translucent `JFrame` in action. We will create a gradient effect (fading effect) inside a `JPanel` by setting the alpha value for its background color different for different pixels in its display area. You can get a per-pixel translucency in different ways. The easiest way to see it in action is to use a `JPanel` with a background color, which has its alpha component set to a desired translucency. The following snippet of code illustrates this method.

```
// Create a frame and set its properties
JFrame frame = new JFrame();
frame.setUndecorated(true);
frame.setBounds(0, 0, 200, 200);

// Set the background color of the frame to all zero
// so that the per-pixel translucency works
frame.setBackground(new Color(0, 0, 0, 0));

// Create a blue JPanel with 128 alpha component
JPanel panel = new JPanel();
int alpha = 128;
```

```
Color bgColor = new Color(0, 0, 255, alpha);
panel.setBackground(bgColor);

// Add the JPanel to the frame and display it
frame.add(panel);
frame.setVisible(true);
```

Two things are different in the above snippet of code. First, it sets the background color of the frame with all color components set to zero to achieve the per-pixel translucency. Second, it sets the background color of the `JPanel`, which has an alpha component, to 128. You can add another `JPanel`, with a different alpha component for its background color, to the `JFrame`. This will give you two areas on the `JFrame`, whose pixels use different translucency.

You can achieve a little fancier result if you use an object of the `GradientPaint` class to paint your `JPanel`. A `GradientPaint` object fills a `Shape` with a linear gradient pattern. It requires you to specify two points, p1 and p2, and a color for each point, c1 and c2. The color on the connecting line between p1 and p2 will proportionally change from c1 to c2.

Listing 3 contains the code for a custom `JPanel`, which uses a `GradientPaint` object to paint its area. The background color for the `JPanel` is specified in its constructor. We have overridden its `paintComponent()` to provide the custom painting effect. The gradient color pattern is provided by `Graphics2D`. The method checks if we have a `Graphics2D` object. Our starting point, p1, is the upper left corner of the `JPanel`. The color for the starting point, c1, is the same as the one passed in the constructor. We use 255 as its alpha component. The second point, p2, is the upper right corner of the `JPanel`, with the same color that uses a zero alpha component. This will give the `JPanel` a gradient effect - from opaque at the left edge to gradually turning transparent at the right edge. You can experiment by changing the two points and the alpha component values for them to get a different gradient pattern. It sets the `GradientPaint` object as the `Paint` object for the `Graphics2D` object and calls the `fillRect()` method to paint the area of the `JPanel`.

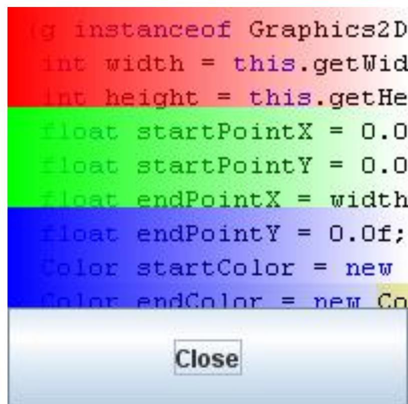
Listing 3: A custom JPanel with a gradient color effect using the per-pixel translucency

```
// TranslucentJPanel.java
package com.jdojo.chapter1;

import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import javax.swing.JPanel;

public class TranslucentJPanel extends JPanel {
     private int red = 240;
     private int green = 240;
     private int blue = 240;
```

```
    public TranslucentJPanel(Color bgColor) {
        this.red = bgColor.getRed();
        this.green = bgColor.getGreen();
        this.blue = bgColor.getBlue();
    }

    @Override
    protected void paintComponent(Graphics g) {
        if (g instanceof Graphics2D) {
            int width = this.getWidth();
            int height = this.getHeight();
            float startPointX = 0.0f;
            float startPointY = 0.0f;
            float endPointX = width;
            float endPointY = 0.0f;
            Color startColor = new Color(red, green, blue, 255);
            Color endColor = new Color(red, green, blue, 0);

            // Create a GradientPaint object
            Paint paint = new GradientPaint(startPointX, startPointY,
                                            startColor,
                                            endPointX, endPointY,
                                            endColor);

            Graphics2D g2D = (Graphics2D) g;
            g2D.setPaint(paint);
            g2D.fillRect(0, 0, width, height);
        }
    }
}
```

Listing 4 contains the code to see the per-pixel translucency in a `JFrame` in action. It adds three instances of the `TranslucentJPanel` class with the background color of red, green, and blue. A Close button is added to the frame to close it at runtime. Figure 1 shows the resulting screen. You will not be able to see the colors as the figure is printed in black-and-white. However, you can see the gradient effect (fading effect). Each panel is more translucent as it goes from left to right. The figure shows the part of the screen that was in the background of the `JFrame` at the time it was displayed.

Figure 1: Per-pixel translucency in action



Listing 4: Using per-pixel translucency in a JFrame

```
// PerPixelTranslucentFrame.java
package com.jdojo.chapter1;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class PerPixelTranslucentFrame extends JFrame {
     private JButton closeButton = new JButton("Close");

     public PerPixelTranslucentFrame(String title) {
          super(title);
          initFrame();
     }

     public void initFrame() {
          this.setDefaultCloseOperation(EXIT_ON_CLOSE);

          // Make sure the frame is undecorated
          this.setUndecorated(true);

          // Set the background color with all components as zero
          // so that per-pixel translucency is used
          this.setBackground(new Color(0,0,0,0));

          // Set its size
```

```
        this.setSize(200, 200);

        // Center it on the screen
        this.setLocationRelativeTo(null);

        this.getContentPane().setLayout(new GridLayout(0, 1));

        // Create and add three JPanel with different color gradients
        this.add(new TranslucentJPanel(Color.RED));
        this.add(new TranslucentJPanel(Color.GREEN));
        this.add(new TranslucentJPanel(Color.BLUE));

        // Add a button to close the window
        this.add(closeButton);
        closeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        PerPixelTranslucentFrame frame =
         new PerPixelTranslucentFrame("Per-Pixel Translucent Frame");
        frame.setVisible(true);
    }

}
```