

WELCOME
TO
JAVAPOLIS



Filthy Makeover

Chet Haase
Java Client Architect
Sun Microsystems



Learn how to use
Filthy Rich Clients
techniques to create applications
that are more compelling and
dynamic.

And more fun.

■ Chet

- is developer and architect at Sun
- writes a technical blog on Desktop Java related issues at <http://weblogs.java.net/blog/chet>

- is co-author of the book
Filthy Rich Clients



- speaks frequently on Desktop Java topics at conferences

But most importantly

- Chet has the microphone



Most desktop applications are boring.

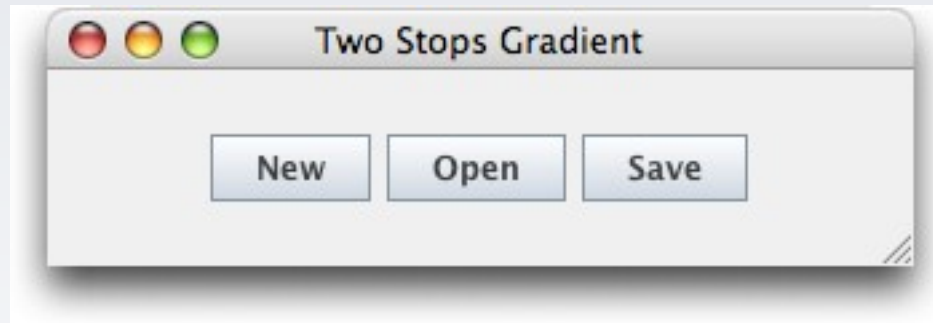
They don't have to be.

- Gradients
- Translucency
- Glass Pane
- Blur
- Shadow
- Animation
- Animated Transitions

- Gradients
- Translucency
- Glass Pane
- Blur
- Shadow
- Animation
- Animated Transitions

- Interpolation between colors
- Two-stop linear gradient
 - In original Java 2D API, J2SE 1.2
- Multi-stop linear gradient
 - Since Java SE 6
- Radial gradient
 - Since Java SE 6

- Interpolation between 2 colors



Two-Stop Linear Gradient: Sample Code

```
Color start, end;

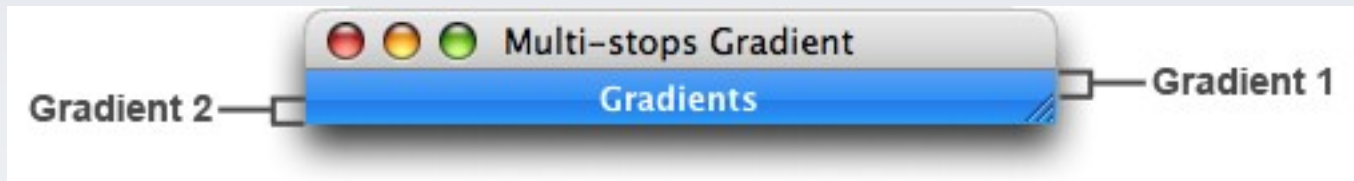
protected void paintComponent(Graphics g) {

    Graphics2D g2 = (Graphics2D) g.create();

    Paint p = new GradientPaint(
        0, 0, start, 0, getHeight(), end);
    g2.setPaint(p);
    g2.fillRect(0, 0, getWidth(), getHeight());

    g2.dispose();
    super.paintComponent(g);
}
```

- Interpolation between several colors
- Can be emulated with two-stop gradients



```
LinearGradientPaint p;  
p = new LinearGradientPaint(  
    0.0f, 0.0f, 0.0f, getHeight(),  
    new float[] { 0.0f, 0.5f, 0.501f, 1.0f },  
    new Color[] { new Color(0x63a5f7),  
                  new Color(0x3799f4),  
                  new Color(0x2d7eeb),  
                  new Color(0x30a5f9) } );
```

- Useful for lighting effects
 - Specular highlights
 - Shadows
- Available in Java SE 6
 - `java.awt.RadialGradientPaint`

DEMO

Gradients



- Cache gradients in images
- Use stretched images
 - Draw gradient in a 1-pixel wide image
 - Paint the image and stretch its width
 - Works with vertical and horizontal gradients only
- Use cyclic gradients
 - Avoids boundary tests
 - See Javadoc for appropriate constructors

- Gradients
- Composites
- Glass Pane
- Blur
- Shadow
- Animation
- Animated Transitions

- Java 2D draws primitives into a destination
 - Primitive == source
- A composite is a rule
 - Combines colors of source and destination
 - Ex.: “Copy only the blue value of source into destination”
- Common in graphics editing tools
 - Adobe Photoshop, The GIMP, ...
 - “Blending modes”

- One interface
 - `java.awt.Composite`
- One implementation
 - `java.awt.AlphaComposite`
- One usage
 - `Graphics2D.setComposite(Composite)`
 - `Graphics2D.getComposite()`

- Basic alpha compositing rules
 - Translucency effects
- 12 rules
 - Porter and Duff, 1984, “Compositing Digital Images”
 - Affect pixels opacity and color
- Alpha channel
 - Value in range [0..1]
 - Defines the opacity (1.0 is 100% opaque)

- Instance with rule:

```
Graphics 2D g2 = // ...
Composite old = g2.getComposite();

Composite c = AlphaComposite.getInstance(
    AlphaComposite.SRC_OVER);
g2.setComposite(c);

// paint stuff...

// restore previous state
g2.setComposite(old);
```

- Instance with rule and opacity:

```
Graphics 2D g2 = // ...
Composite old = g2.getComposite();

Composite c = AlphaComposite.getInstance(
    AlphaComposite.SRC_OVER, 0.5f);
g2.setComposite(c);

// paint stuff...

// restore previous state
g2.setComposite(old);
```

- Draw translucent primitives into destination
- Semi-transparent logo:

```
private BufferedImage logoImage;

protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g.create();
    Composite c = AlphaComposite.getInstance(
        AlphaComposite.SRC_OVER, 0.5f);
    g2.setComposite(c);
    g2.drawImage(logoImage, 0, 0, null);
}
```

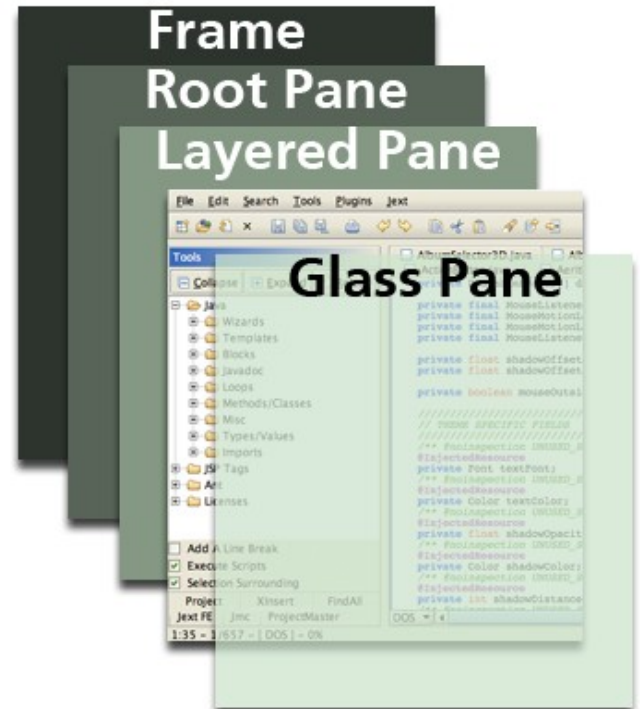
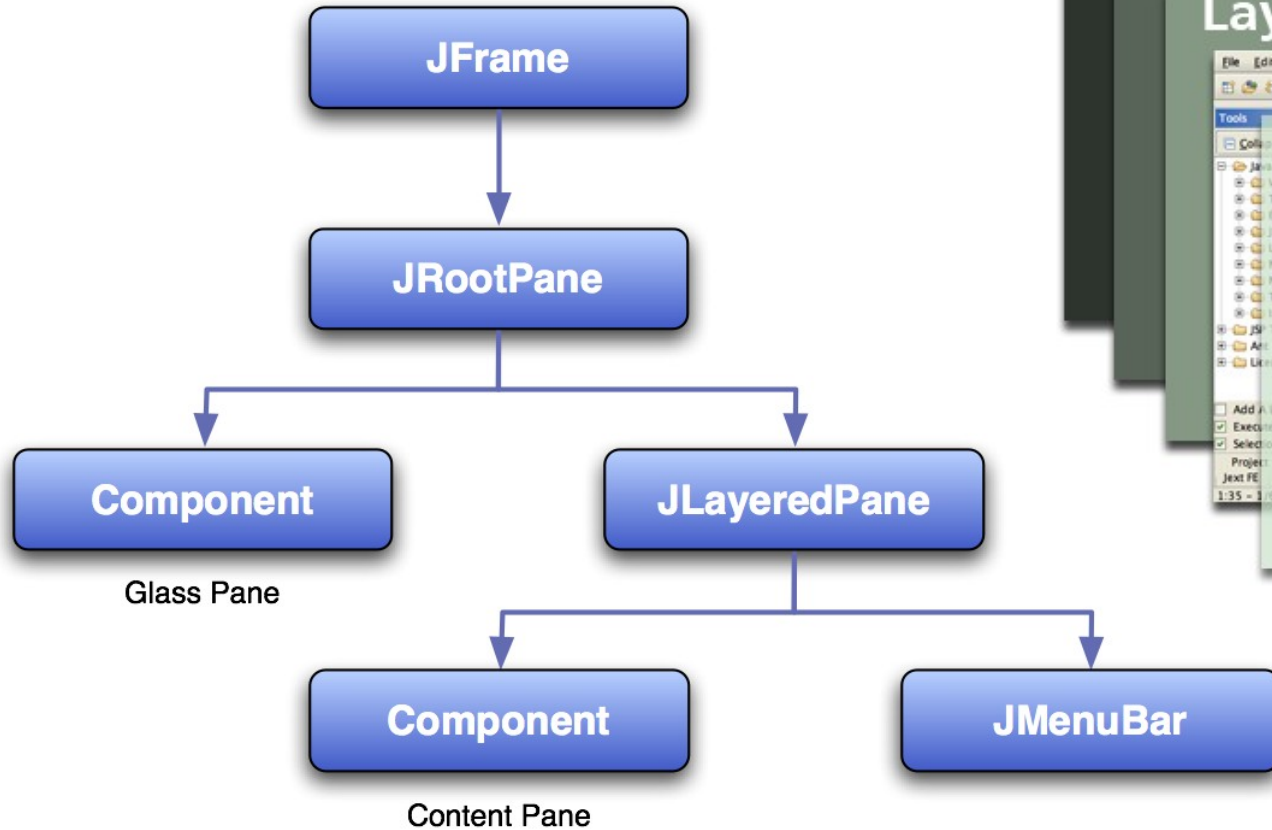
DEMO

Composites



- Gradients
- Composites
- **Glass Pane**
- Blur
- Shadow
- Animation
- Animated Transitions

Glass Pane



- Allows painting over the entire window area
- Easy to use

```
JFrame frame = new JFrame("Glass Pane");  
frame.setGlassPane(new VeilGlassPane());  
frame.setVisible(true);  
frame.getGlassPane().setVisible(true);
```

- Glass pane is not visible by default
- Glass pane is a simple JComponent
 - Use non-opaque components

Custom Glass Pane

```
class VeilGlassPane extends JComponent {
    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setColor(Color.WHITE);
        Composite c = AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, 0.5f);
        g2.setComposite(c);
        g2.fillRect(0, 0, getWidth(), getHeight());
        g2.dispose();
    }
}
```

DEMO

Glass Pane



- Gradients
- Translucency
- Glass Pane
- **Blur**
- Shadow
- Animation
- Animated Transitions

- A blur is done with ConvolveOp
 - A specific kind of BufferedImageOp
- BufferedImageOps are filters on images
 - Like this:

```
BufferedImageOp op = // createImageOp...  
BufferedImage image = loadImage();  
image = op.filter(image, null);
```

- or this:

```
BufferedImageOp op = // createImageOp...  
BufferedImage image = loadImage();  
Graphics2D g2 = // get Graphics object...  
g2.drawImage(image, op, 0, 0);
```

- Performs a convolution from a source image to a destination
- Each pixel and its neighbors are multiplied by a convolution kernel
- A kernel is a matrix:

$$kernel = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

- Pixels:

255, 255, 255	255, 255, 255	255, 255, 255
255, 255, 255	0, 0, 0	255, 255, 255
255, 255, 255	255, 255, 255	255, 255, 255

- Convolution:

255, 255, 255 x 1/9	255, 255, 255 x 1/9	255, 255, 255 x 1/9
255, 255, 255 x 1/9	0, 0, 0 x 1/9	255, 255, 255 x 1/9
255, 255, 255 x 1/9	255, 255, 255 x 1/9	255, 255, 255 x 1/9

- The new color of the pixel is the sum of all operations
- In this example:
 - $r = 8/9 \times 255 + 1/9 \times 0 = 227$
 - $g = 8/9 \times 255 + 1/9 \times 0 = 227$
 - $b = 8/9 \times 255 + 1/9 \times 0 = 227$
- The result is a light gray
- This kernel is a simple blur

ConvolveOp: Blurring an Image

```
float[] data = new float[] {  
    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,  
    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f,  
    1.0f/9.0f, 1.0f/9.0f, 1.0f/9.0f  
};  
Kernel k = new Kernel(3, 3, data);  
ConvolveOp op = new ConvolveOp(k);  
  
blurImage = op.filter(blurImage, null);
```

- “Box Blur”: Kernels with equal pixel weights
 - Can be faster to compute
 - Artifacts for high-contrast edges
- “Gaussian Blur”: Normal distribution of weights
 - More expensive calculations
 - Better blending in high-contrast situations
- Performance Tips:
 - Up-scale for cheap blur effects
 - Or combine up-scale with convolution on smaller image
 - Can be faster to use smaller/separate filters than one larger one

DEMO

Blur



- Gradients
- Translucency
- Glass Pane
- Blur
- **Shadow**
- Animation
- Animated Transitions

- Simulate lighting
- Simulate depth
- Make GUI more realistic
 - Provides natural visual feedback for the user
- Simple shadows are easy

Simple Shadow

```
public static BufferedImage
    createDropShadow(BufferedImage image, int size) {

    // Create shadow to be larger to the right and bottom
    BufferedImage shadow = new BufferedImage(
        image.getWidth() + 4 * size,
        image.getHeight() + 4 * size,
        BufferedImage.TYPE_INT_ARGB);

    Graphics2D g2 = shadow.createGraphics();
    g2.drawImage(image, size * 2, size * 2, null);

    g2.setComposite(AlphaComposite.SrcIn);
    g2.setColor(Color.BLACK);
    g2.fillRect(0, 0, shadow.getWidth(), shadow.getHeight());

    g2.dispose();
    return shadow;
}
```

- Simple shadows are not realistic
 - Real world objects don't cast shadows with hard edges
- Realistic shadows require blur
 - Could use previous blur approach
 - And optimize it
- Or just use SwingLabs classes
 - `DropShadowBorder(lineColor, lineWidth, size, opacity, cornersize, top, left, bottom, right);`
 - `ShadowFactory(size, opacity, color);`
`BufferedImage createShadow(image);`
 - `DropShadowPanel()` (as parent to shadowed children)

DEMO

Drop Shadows



- Gradients
- Translucency
- Glass Pane
- Blur
- Shadow
- **Animation**
- Animated Transitions

- Varying Properties Over Time
- Examples:
 - Fading an object in (varying alpha)
 - Cross-fading between two scenes (varying multiple alpha values)
 - Moving an object (changing its location)
 - Glowing/pulsating (varying alpha)

- Simple, declarative model of animating properties

```
Animator anim =  
    PropertySetter.createAnimator(  
        duration, object, "property",  
        values...);
```

- Plus easy specification of timing behaviors:

```
// Ease in for first 30% of animation  
anim.setAcceleration(.3f);  
// Repeat infinitely  
anim.setRepeatCount(Animator.INFINITE);  
// Reverse at the end of each animation  
anim.setRepeatBehavior(RepeatBehavior.REVERSE);
```

TimingFramework: Fading out a custom component

```
float alpha = 1.0f;

public void setAlpha(float alpha) {
    this.alpha = alpha;
    repaint();
}

protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g.create();
    g2.setComposite(
        AlphaComposite.SrcOver.derive(alpha));
    // paint stuff...
    g2.dispose();
}

// Now, create and run the animation
Animator animator = PropertySetter.
    createAnimator(350, this, "alpha", 0.0f);
animator.start();
```

DEMO

Animation



- Gradients
- Translucency
- Glass Pane
- Blur
- Shadow
- Animation
- **Animated Transitions**

- The Big Idea
 - Communicate to the user, through animation between application states, how they got *here* from back *there*
- Short animations on changing components
 - Fade out (when components go away)
 - Fade in (when components come into being)
 - Move/Resize (when components change location/size)

- animatedtransitions.dev.java.net
- Simplifies transitions between application states
- You tell it:
 - The container to transition
 - The callback object for setting up the next screen
- It figures out the component deltas and runs default animated effects
 - fades, moves, resizes

Example Code

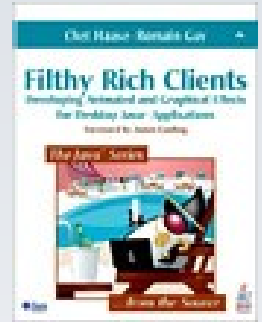
```
ScreenTransition transition = new ScreenTransition(  
    this, // transition container  
    this, // callback object  
    500); // duration of animation  
transition.start();  
  
public void setupNextScreen() {  
    // called from ScreenTransition after call to start()  
    // set up GUI for the next screen  
}
```

DEMO

Animated Transitions



- The Book: filthyrichclients.org
 - More details on these effects and others
 - All 82 book demos (complete source code!)
 - **Book signing at 3:15 today** (bookstore booth)
- mailman: Scott Violet's blog
 - weblogs.java.net/blog/zixle/archive/2006/11/extreme_gui_mak.html
- Timing Framework:
 - timingframework.dev.java.net
- Animated Transitions
 - animatedtransitions.dev.java.net
 - Article on java.net:
 - "Create Moving Experiences for Your Users with Animated Transitions"



Q&A

View JavaPolis talks @ www.parleys.com



Thank you for your
attention

