Swing Effects

Kirill Grouchnikov, Amdocs

Desktop Matters Conference

San Jose, March 8-9 2007

What is this about?

- Show two sample effects on buttons
- Show how this can be achieved with UI delegates
- Discuss alternative implementations

Demo

Possible ways to do this

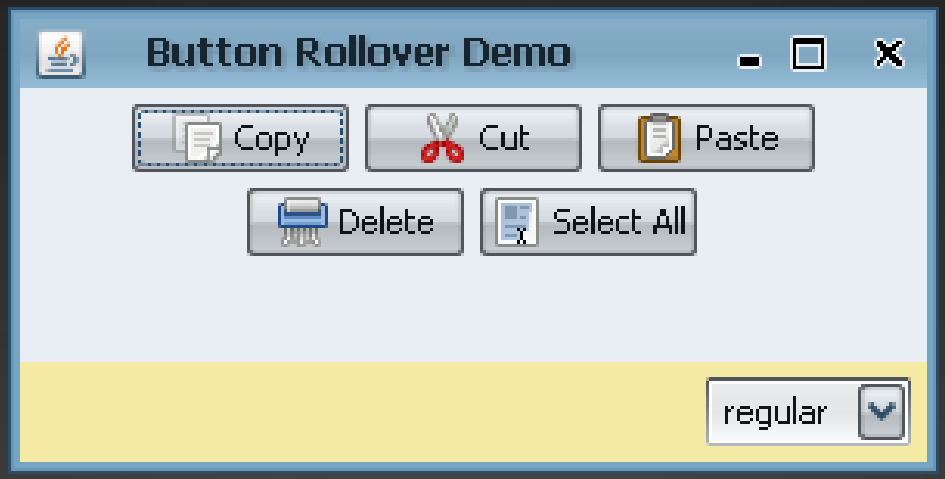
- Custom component – paintComponent
- Custom repaint manager
- Custom glass pane
- Custom UI delegates

Possible ways to do this

- Custom component – paintComponent
- Custom repaint manager
- Custom glass pane
- Custom UI delegates (*)

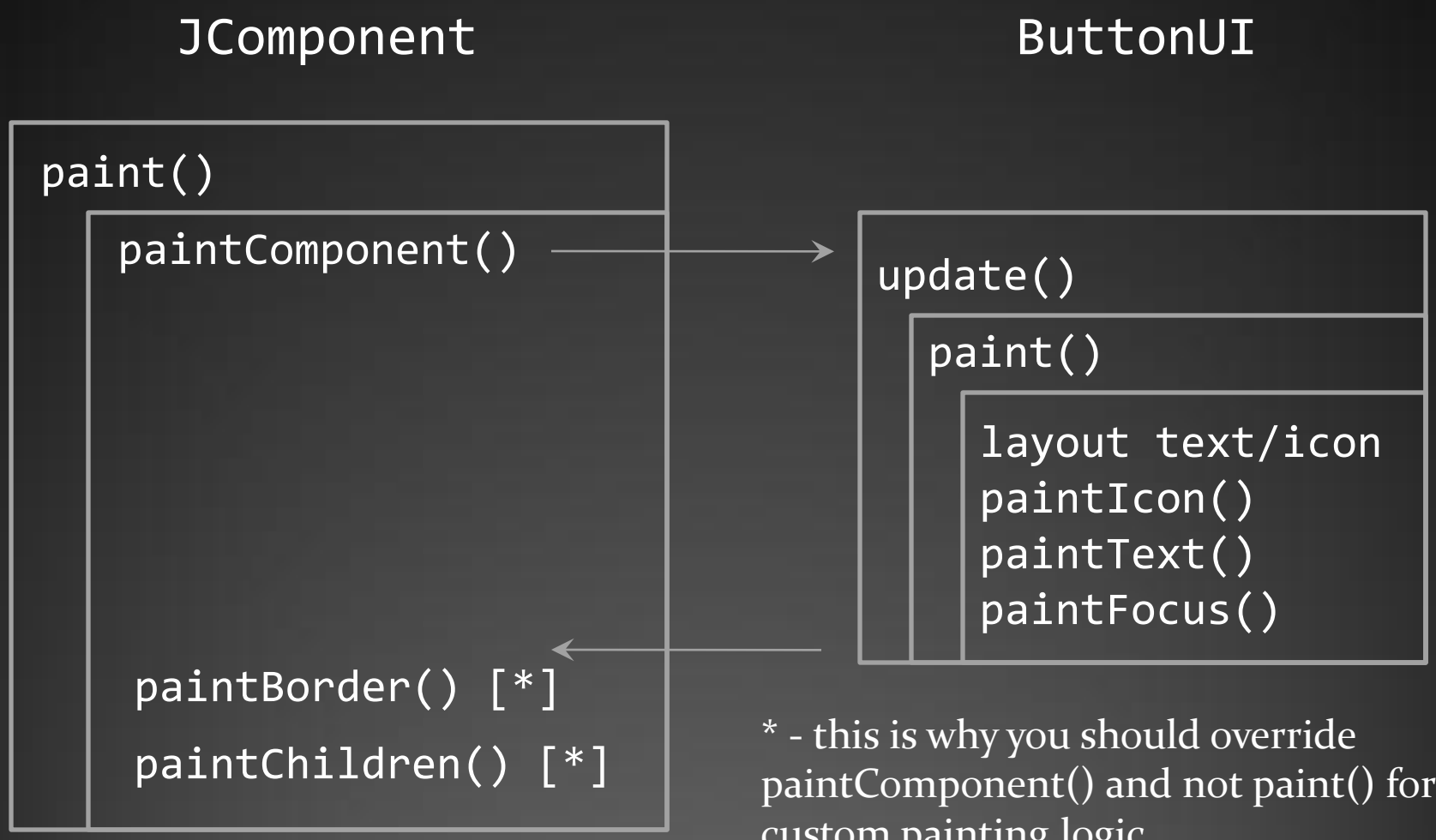
* - next slides describe the UI delegates approach

How does Swing paint?



- UI delegates – classes responsible for painting Swing components.
- The panels are painted by the PanelUI delegate (unless a panel overrides the paintComponent method).
- The buttons are painted by the ButtonUI delegate (unless a button overrides the paintComponent method).

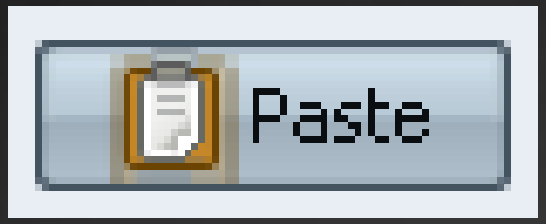
How is the button painted?

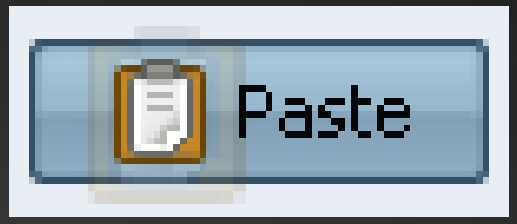


* - this is why you should override paintComponent() and not paint() for custom painting logic.

Icon ghosting - schematics

Icon ghosting - details





We add the custom painting code in the following places:

- ButtonUI.paintIcon – before calling super(), paint the same icon scaled up and translucent (and offset).
- PanelUI.update – paint the part of the icon that animates outside the child (button) bounds.

In addition, a ChangeListener is registered on the button model, tracking changes to the rollover state.

What to keep in mind?

- The icon is not an *Image*. Call *paintIcon* after the *Graphics* has been scaled and a composite has been applied to it.
- The ghost image must be centered around the original icon.

- The ghost effect may "spill" from the immediate parent and overlay sibling components.

ButtonUI code (part I)

```
protected void paintIcon(Graphics g, JComponent c, Rectangle iconRect) {
   Graphics2D graphics = (Graphics2D) g.create();
   FadeTracker fadeTracker = FadeTracker.getInstance();
   AbstractButton b = (AbstractButton) c;
   if ((ButtonBackgroundDelegate.getKind(b) == ButtonTitleKind.NONE)
         && fadeTracker.isTracked(c, null,
                  FadeKind.GHOSTING_ICON_ROLLOVER, true)) {
         float fade10 = fadeTracker.getFade10(c,
                  FadeKind.GHOSTING_ICON_ROLLOVER);
         // 0.0 --> 0.5
         // 10.0 --> 0.0
         float opFactor = -0.5f * fade10 / 10.0f + 0.5f;
         graphics.setComposite(TransitionLayout.getAlphaComposite(c,
                  opFactor));

         Icon icon = SubstanceCoreUtilities.getIcon(b);
         if (icon != null) {
                  → see the next slide
         }
   }

   graphics.setComposite(TransitionLayout.getAlphaComposite(c));
   super.paintIcon(graphics, c, iconRect);
   graphics.dispose();
}
```

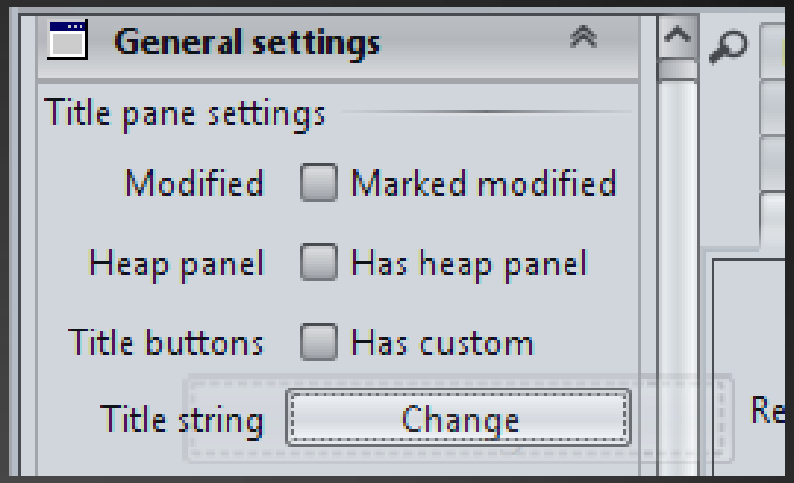
ButtonUI code (part II)

```
double iFactor = 1.0 + fade10 / 10.0;
double iWidth = icon.getIconWidth() * iFactor;
double iHeight = icon.getIconHeight() * iFactor;
BufferedImage iImage = SubstanceCoreUtilities.getBlankImage(
   (int) iWidth, (int) iHeight);
Graphics2D iGraphics = (Graphics2D) iImage.createGraphics();
iGraphics.scale(iFactor, iFactor);
icon.paintIcon(b, iGraphics, 0, 0);
iGraphics.dispose();
int dx = (int) ((iWidth - icon.getIconWidth()) / 2);
int dy = (int) ((iHeight - icon.getIconHeight()) / 2);
graphics.drawImage(iImage, iconRect.x - dx, iconRect.y - dy,
   null);
```

Eye candy

Icon ghosting over multiple components

Press ghosting over multiple components



Multiple icon and press ghostings

UI delegates – pros & cons

- (+) Minimal changes in the application code.
- (+) No need for custom painting code
- (+/-) Available only under the specific LAF
  - Can use bytecode injection to change existing third-party LAFs (possible but slightly complicated – stay tuned for more in the future)
- (-) Handling “spilling” is in container delegates
- (-) Custom paintComponent implementations

Other approaches

- How to handle painting order (background, ghost icon, icon+text)?
- Custom components
  - (+/-) Work under all LAFs (???)
  - (-) Can handle only painting in the component bounds
- Custom repaint manager
  - (+/-) Work under all LAFs (???)
  - (-) More complicated to write and test
  - (-) Interfere with existing repaint manager (e.g. from SwingX)
- Custom glass pane
  - (+/-) Work under all LAFs (???)
  - (-) Needs to track multiple overlaying animations
  - (-) Interfere with existing app glass pane

Additional information

- Kirill's blog
  - http://weblogs.java.net/blog/kirillcool
- Alex's blog
  - http://weblogs.java.net/blog/alexfromsun
- Substance LAF project
  - https://substance.dev.java.net
- Laf-Widget project
  - https://laf-widget.dev.java.net

Q&A

kirillcool@yahoo.com

Icon ghosting animation

Container UI code (part I)

```
@Override
public void update(Graphics g, JComponent c) {
  // Paint regular container stuff. In most cases -
  // background (fill, gradient, watermark)
  …
  // Paint ghost images of the animated overlapping
  // components
  GhostPaintingUtils.paintGhostImages(c, g);
}
```

Container UI code (part II)

```
public static void paintGhostImages(Component mainComponent, Graphics g) {
   Graphics2D graphics = (Graphics2D) g.create();
   Rectangle panelRect = mainComponent.getBounds();
   panelRect.setLocation(mainComponent.getLocationOnScreen());
   if (FadeConfigurationManager.getInstance().fadeAllowed(
         FadeKind.GHOSTING_ICON_ROLLOVER, mainComponent)) {

         Set<Component> animComps = FadeTracker.getInstance()
                 .getAllComponentsByFadeKind(
                         FadeKind.GHOSTING_ICON_ROLLOVER);
         for (Component comp : animComps) {
                 // see next slide
         }
   }
   graphics.dispose();
}
```

Container UI code (part III)

```
Rectangle compRect = comp.getBounds();
compRect.setLocation(comp.getLocationOnScreen());

int dx = compRect.x - panelRect.x;
int dy = compRect.y - panelRect.y;

compRect.x -= compRect.width / 2;
compRect.y -= compRect.height / 2;
compRect.width *= 2;
compRect.height *= 2;

if (panelRect.intersects(compRect)) {
   // see next slide
}
```

Container UI code (part IV)

```
float fade10 = FadeTracker.getInstance().getFade10(comp,
   null, FadeKind.GHOSTING_ICON_ROLLOVER);
// 0.0 --> 0.5,  10.0 --> 0.0
float opFactor = 0.5f * (1.0f - fade10 / 10.0f);
graphics.setComposite(TransitionLayout.getAlphaComposite(mainComponent,
   opFactor));
Icon icon = null;
Rectangle iconRect = null;
if (comp instanceof AbstractButton) {
   AbstractButton button = (AbstractButton) comp;
   icon = SubstanceCoreUtilities.getIcon(button, false);
   iconRect = (Rectangle)button.getClientProperty(SubstanceButtonUI.ICON_RECT);
}

if ((icon != null) && (iconRect != null)) {
   // see next slide
}
```

Container UI code (part V)

```
double iFactor = 1.0 + fade10 / 10.0;
double iWidth = icon.getIconWidth() * iFactor;
double iHeight = icon.getIconHeight() * iFactor;
BufferedImage iImage = SubstanceCoreUtilities.getBlankImage((int)
   iWidth, (int) iHeight);
Graphics2D iGraphics = (Graphics2D) iImage.createGraphics();
iGraphics.scale(iFactor, iFactor);
icon.paintIcon(comp, iGraphics, 0, 0);
iGraphics.dispose();
dx -= (int) ((iWidth - icon.getIconWidth()) / 2);
dy -= (int) ((iHeight - icon.getIconHeight()) / 2);
graphics.drawImage(iImage, dx + iconRect.x, dy + iconRect.y, null);
```

Press ghosting animation

Button press ghosting

What to keep in mind?

- The ghost image must be centered around the button image.

- The ghost effect may “spill” from the immediate parent and overlay sibling components.

Container UI code (partial)

```
double iFactor = 1.0 + fade10 / 10.0;
double iWidth = bounds.width * iFactor;
double iHeight = bounds.height * iFactor;
BufferedImage iImage =
  SubstanceCoreUtilities.getBlankImage(
      (int) iWidth, (int) iHeight);
Graphics2D iGraphics = (Graphics2D)iImage.createGraphics();
iGraphics.scale(iFactor, iFactor);
comp.paint(iGraphics);
iGraphics.dispose();
dx -= (int) ((iWidth - bounds.width) / 2);
dy -= (int) ((iHeight - bounds.height) / 2);
graphics.drawImage(iImage, bounds.x - dx, bounds.y - dy,
  null);
```

Some corner cases

- Not paint too much (trigger repaint only of the affected area)
- Handle change in showing / visibility during the animation
- Account for large button / icons (start from lower alpha)
- Cache animating child image for reuse during painting in the same animation cycle on different containers