Bringing Life to Swing Desktop Applications

Alexander Potochkin
Sun Microsystems

Kirill Grouchnikov
Amdocs Inc.

TS-3414

Presentation Goal

Learn advanced painting techniques
to enrich your Swing applications

Agenda

Advanced Effects

- Custom Components
- Playing With Opacity
- Custom RepaintManager
- GlassPane
- Layering in UI Delegates

Rainbow Demo

Q&A

Advanced Effects

Introduction

- Translucency
- Non-rectangular components
- Layering
- Image filtering
- Animation

Agenda

Advanced Effects

- **Custom Components**
- Playing With Opacity
- Custom RepaintManager
- GlassPane
- Layering in UI Delegates

Rainbow Demo

Q&A

Custom Components

Implementation

- Put `setOpaque(false)` in constructor for translucent components
- Override `paint()`
  - To change the graphics state of the superclass
  - To paint over the whole component
  - Don't forget to call `super.paint()`
- Override `contains()`
  - For non-rectangular components

Custom Components

setOpaque()

- `setOpaque(false)` == “draw stuff behind me”
  - Useful for translucent or non-rectangular components
- `setOpaque(true)` == “I’ll handle it”
  - During repainting of an opaque component Swing doesn’t repaint any components behind
- If component is entirely opaque, this method doesn’t change its visual appearance

Custom Components

paint()

- Responsible for painting the whole component
- Can be used to render a component to an image

```
public void paint(Graphics g) {

   paintComponent(g);

   paintBorder(g);
   paintChildren(g);
}
```

Custom Components

contains()

- Override it to implement a custom filter for MouseEvents
- If contains() returns false a MouseEvent with x,y coordinates will be rejected otherwise accepted

```
public boolean contains(int x, int y) {
   return super.contains(x, y);
}
```

Custom Components

Non-rectangular component

```
public class OvalButton extends JButton {
  public OvalButton(String text) {
      super(text);
      setOpaque(false);
  }

  // Define the new shape for component
  private Shape getShape() {
    return new Ellipse2D.Float
      (0, 0, getWidth()/2, getHeight());
    }
```

Custom Components

Non-rectangular component

```
    // Clip the graphics
    public void paint(Graphics g) {
      Graphics2D g2 = (Graphics2D) g;
      g2.setClip(getShape());
      super.paint(g);
    }

    // Skip mouse events outside the shape
    public boolean contains(int x, int y) {
      return getShape().contains(x, y);
    }
}
```

Custom Components

Advanced effects

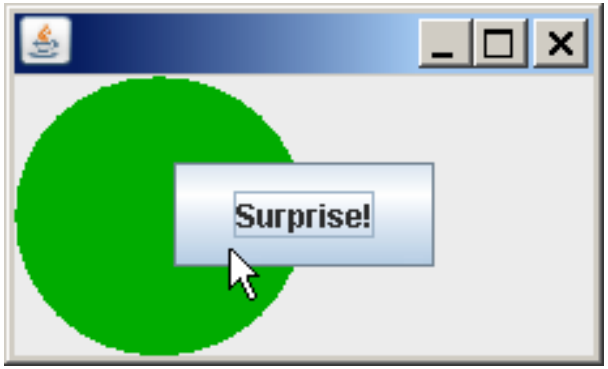
- Simple custom components can support any layering, translucency, transparency etc…
- Some components may contain child components (JComboBox, JTable or JPanel)
- Effects should work for compound components as well
- Let's try the more complex scenario

Custom Components

Non-rectangular container—the problem

```
JPanel panel = new OvalPanel();
panel.setBackground(Color.GREEN);

frame.add(panel);
panel.add(new JButton("Surprise!"));
```



Custom Components

Points to remember

- You can implement any effect for a custom component
- Generally, container's paint() doesn't get called when a children is repainted itself
- More efforts should be made to expand effects on child components

Agenda

Advanced Effects

- Custom Components
- **Playing With Opacity**
- Custom RepaintManager
- GlassPane
- Layering in UI Delegates

Rainbow Demo

Q&A

Playing With Opacity

Introduction

- By default, all controls are opaque
- An opaque control fills every pixel in its bounds
- Doesn't allow proper painting of overlapped components

Playing With Opacity

Points to remember

- Is a boolean setting—doesn't provide built-in support for custom translucency values
- Might interfere with existing application logic (property change listeners)
- Component might look different depending on opaque state

```
JLabel label = new JLabel("Opaque");
label.setBackground(Color.MAGENTA);
frame.add(label);

label.setText("Non Opaque");
label.setOpaque(false);
```

Transition Effects

Using opacity for transition effects

- Problem—UIs changes are immediate
  - Showing/hiding a control
  - Moving a control to new location
  - Tab switch
- Solution—use transitions (cross fades, fly-in/out)
- Making controls non-opaque to enable the transition effects

DEMO

Transition Layout Demo

Transition Effects

Laf-Widget solution—available to look and feels

```
JTabbedPane myTabbedPane = ...;
TransitionLayoutManager.getInstance().
      track(myTabbedPane, true);

JPanel myPanel = ...;
TransitionLayoutManager.getInstance().
      track(myPanel, true);
```

- Implemented animation/transition effects
  - Play with opacity (set to false during animation cycle)
  - Set translucency (for fades)
  - Support in the UI delegates (bytecode injection)
  - Custom layout manager (for sliding effects)

Transition Effects

Possible scenarios

- Remains visible and has the same bounds
- Remains visible and has different bounds
- Becomes invisible
- Added or becomes visible
- Remains invisible

Transition Effects

Many issues

- Components' borders—not painted by UI delegates
- JDesktopPane—ignores opacity setting
- Playing with layout manager, opacity and visibility
- Removed components
- Requires changes in some LAF methods—to respect the translucency

Agenda

Advanced Effects

- Custom Components
- Playing With Opacity
- **Custom RepaintManager**
- GlassPane
- Layering in UI Delegates

Rainbow Demo

Q&A

RepaintManager

Introduction

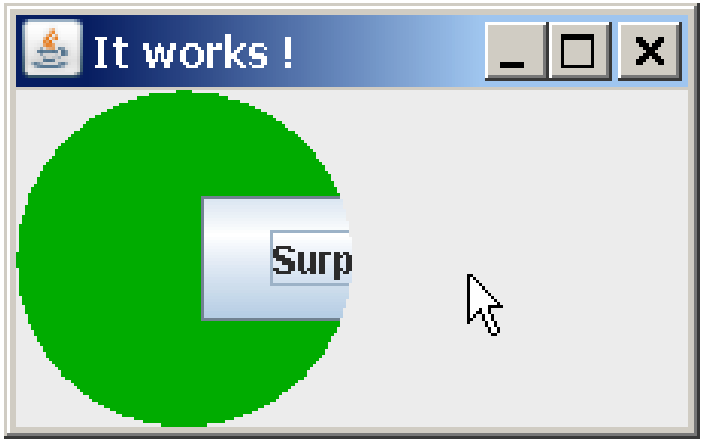
- Controls Swing component's repainting
  - double buffering, repaint() coalescing
- One RepaintManager for all components
- Can be changed by a programmer at any time
  - **public static void setCurrentManager(RepaintManager)**
- Can be used to force child components to repaint with their container

RepaintManager

Custom implementation

```
RepaintManager.setCurrentManager(
                        new MyRepaintManager());

JButton b = new JButton("Surprise!")
panel.add(b);
```



RepaintManager

Custom implementation

```
class MyRepaintManager extends RepaintManager {

  // This can be optimized
  public void addDirtyRegion(JComponent c,
                  int x, int y, int w, int h) {

    JComponent parent = (JComponent)
      SwingUtilities.getAncestorOfClass(MyPanel.class,c);

    // Child is repainted, repaint the whole parent
    if (parent != null) {
        super.markCompletelyDirty(parent);
    } else {
        super.addDirtyRegion(c, x, y, w, h);
    }
  }
}
```

RepaintManager

Summary

- **Pros**
  - Does not affect any component's state
  - Easy to use
- **Cons**
  - Conflicts with another custom RM are possible

```
if (!(currentManager instanceof MyRepaintManager)) {

     RepaintManager.
         setCurrentManager(new MyRepaintManager());
}
```

SwingX Project

Custom RepaintManager

- JXPanel—a special container which supports
  - Translucency
    - JXPanel.setAlpha(float)
  - Painters API
  - Image filtration
- Uses custom RepaintManager
  - To make JXPanel repaint with its children

SwingX Project

Example

```
JXPanel panel = new JXPanel();
panel.add(new JButton("JButton"));
frame.add(panel);

panel.setAlpha(.5f);

panel.setBackgroundPainter(new Painter() {

  public void paint(Graphics2D g2,
                    Object o, int w, int h) {
     g2.setColor(Color.MAGENTA);
     g2.fillRect(0, 0, w, h);
  }
});
```

Agenda

Advanced Effects

- Custom Components
- Playing With Opacity
- Custom RepaintManager
- **GlassPane**
- Layering in UI Delegates

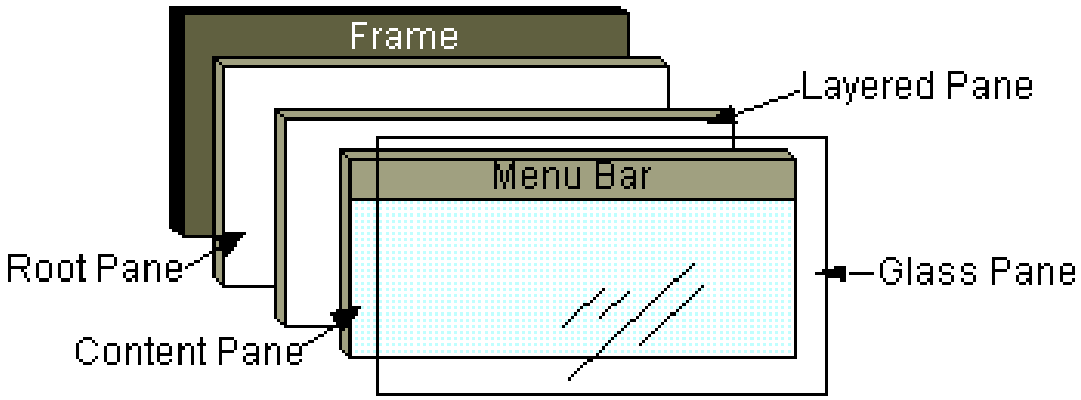
Rainbow Demo

Q&A

GlassPane

Introduction

- The topmost component in a frame
- Transparent—setOpaque(false)
- Invisible by default



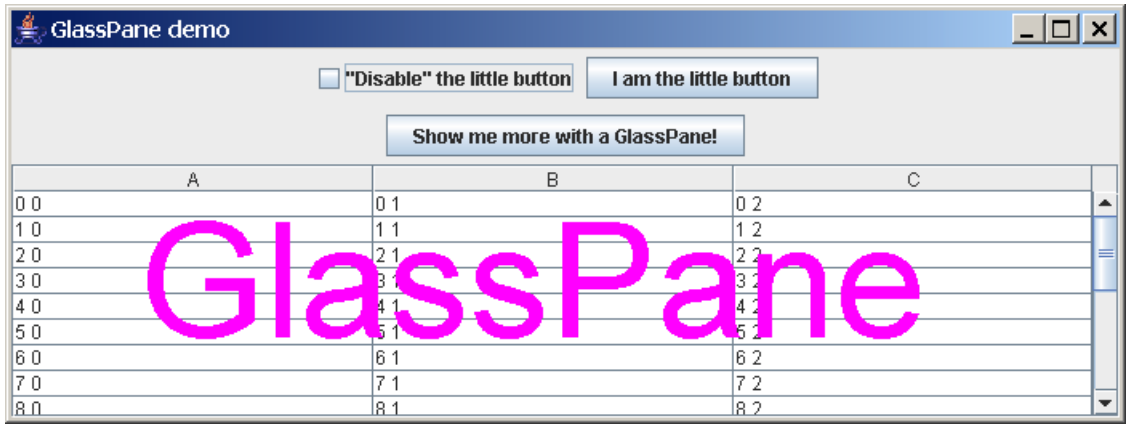
GlassPane

Custom component

- Painting over the all components

```
frame.setGlassPane(new CustomGlassPanel());

frame.getGlassPane().setVisible(true);
```

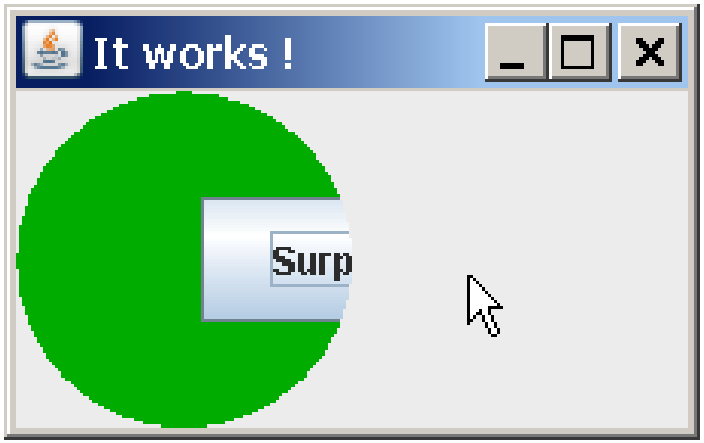


GlassPane

Surprising effect

```
JButton b = new JButton("Surprise!")
panel.add(b);
frame.add(panel);

frame.getGlassPane().setVisible(true);
```



GlassPane

Points to remember

- If glassPane is visible then Swing needs to repaint every component together with it
  - Swing repaints a component, starting from their common ancestor—JRootPane
- GlassPane is a global resource
- Transparent panel inside your component will work the same way

Transparent Panel

Summary

- **Pros**
  - Does not affect component's state nor any global setting
- **Cons**
  - Additional component in hierarchy

JXLayer

Transparent panel

- **JXLayer**—a special container, which supports
  - Painters API
  - Image filtering
  - Translucency
    - PainterModel.setAlpha(float)
  - Non-rectangular components
    - MouseEvents filtering

JXLayer

Implementation

- It is a component wrapper like JScrollPane
  - You have access to the wrapped component's state
- It does not use glassPane from the frame
  - It has its own a transparent panel on the top
- JXLayer.setPainter() allows to completely change component's appearance
  - JXLayer.paint() delegates all painting to the painter

JXLayer

Example

```
JTextField tf = new JTextField("Hello");

JXLayer<JTextField> layer =
                              new JXLayer<JTextField>(tf);

// Apply custom painting
layer.setPainter(myPainter);

// Apply mouseEvents filter
layer.setMouseClipShaper(myPainter);

frame.add(layer);
```

JXLayer

Custom painter

```
class MyPainter extends AbstractPainter<JTextField> {

  public void paint(Graphics2D g2,
                    JXLayer<JTextField> l) {
    l.paint(g2);

    if ("green".equals(l.getView().getText())) {
      g2.setColor(Color.GREEN);
      g2.fillRect(0, 0, l.getWidth(), l.getHeight());
    }
  }

  public boolean contains(int x, int y,
                          JXLayer<JTextField> l) {
    return !"break".equals(l.getView().getText());
  }
}
```

DEMO

JXLayer Demo

Agenda

Advanced Effects

- Custom Components
- Playing With Opacity
- Custom RepaintManager
- GlassPane
- **Layering in UI Delegates**

Rainbow Demo

Q&A

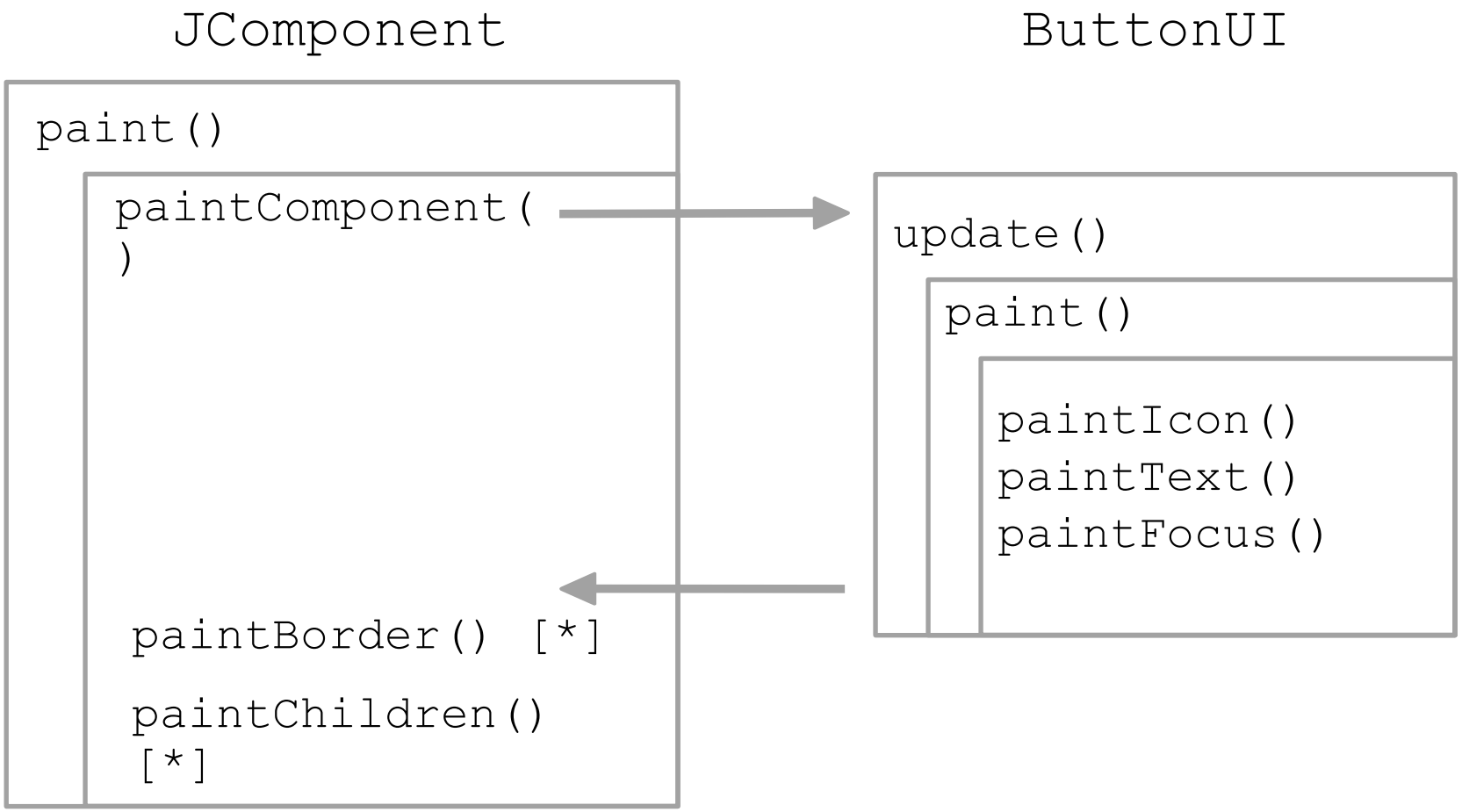
Layering in UI Delegates

Introduction

- UI delegates—classes responsible for painting Swing components
  - JPanel—PanelUI delegate [*]
  - JButton—ButtonUI delegate [*]
  - …(41 different UI delegates)
- Provide flexible control over painting different visual layers of Swing components

Layering in UI Delegates

Example—how is a button painted?



Layering in UI Delegates

Alternatives and possibilities

- Repaint managers, glass pane and custom components—much higher level
- UI delegate can put painting code
  - After icon painting
  - But before text painting
- Opens the field to a wide array of effects
  - Ghost images/springs
  - Ripples
  - …

Ghosting Effects

Introduction

- Problem—UIs are not “live” enough
  - Moving the mouse over a button (rollover)
  - Pressing a button
- Solution—use spring/ghost effects for richer visual indications

DEMO

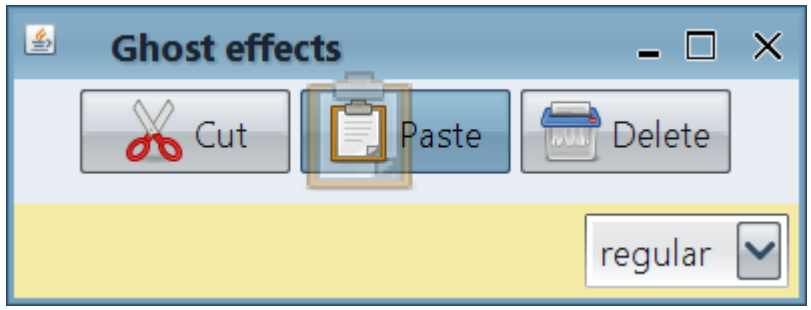
Ghosting Effects Demo

Ghosting Effects

Painting sequence

Ghosting Effects

Details



- Custom painting code in:
  - **ButtonUI.paintIcon**
  - PanelUI.update
- Listener to initiate the animations

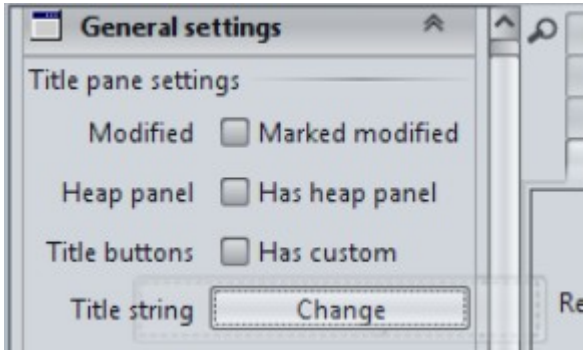
```
update()
    paint()
        paintIcon()
        paintText()
        paintFocus()
```

Ghosting Effects

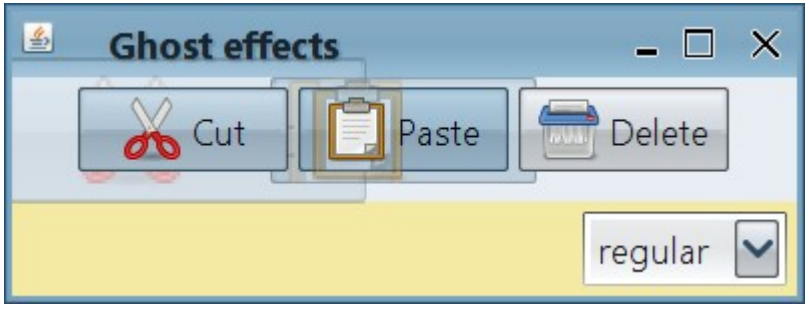
Eye candy

Icon ghosting over multiple components

Press ghosting over multiple components



Multiple icon and press ghostings



Ghosting Effects

Using in look-and-feels

- Available—button rollover (icon) and button press
- Manual changes—API to call
- Automatic changes—Ant tasks to change compiled UI delegates (bytecode injection)
- Later tested on core Windows LAF and seven third-party LAFs

Ghosting Effects

UI delegates—summary

- Pros
  - Minimal changes in the application code
  - No need for custom painting code
  - Available under multiple look and feels (use bytecode injection)
- Cons
  - Handling “spilling” is in container delegates
  - Custom paintComponent implementations

Agenda

Advanced Effects

- Custom Components
- Playing With Opacity
- Custom RepaintManager
- GlassPane
- Layering in UI Delegates

Rainbow Demo

Q&A

Links

- **JXLayer project**
  - https://swinghelper.dev.java.net/
  - Alexander Potochkin's blog
    - http://weblogs.java.net/blog/alexfromsun/
- **Laf-Widget project**
  - http://laf-widget.dev.java.net
  - Kirill's blog
    - http://weblogs.java.net/blog/kirillcool/
- **SwingX project**
  - http://swingx.dev.java.net/

Q&A

Alexander.Potochkin@sun.com

kirillcool@yahoo.com

Bringing Life to Swing Desktop Applications

Alexander Potochkin
Sun Microsystems

Kirill Grouchnikov
Amdocs Inc.

TS-3414