Eight Ways to Be More Productive Developing Swing Applications

Ben Galbraith

Swing Consultant
http://www.galbraiths.org/

TS-1913

Presentation Goal

Learn how to become more productive with Swing in eight easy steps!

While this talk focuses on ideas, open-source code you can use in your projects will accompany this talk.

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Use a Cross-Platform Look-and-Feel

- Complex applications often employ tricky layouts and require custom widget and painting tweaks
  - Do you really want to do that two or more times?
  - Customizing the OS X “plaf” is a big pain

Which Look-and-Feel?

- Three reasonably slick looks
  - JGoodies Plastic family (starting to look dated)
  - Incors Alloy (also starting to look dated)
  - Synthetica, especially the “Moon” themes
- Rolling your own isn’t very hard, but can be a lot of work
  - Synth (and Synthetica) helps lower the curve quite a bit
- If you must use a “plaf”, check out
  - WinLAF for Windows
  - Quaqua for OS X

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing's Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing's Action

Introduce a Form Concept

Use a GUI Builder

- Swing GUI builders have traditionally been sub-par
  - That’s changed over the past two years
- At least three high-quality GUI builders exist
  - JFormDesigner
  - Sun’s NetBeans™ software
  - JetBrains’ IDEA

Decouple Your App From Your GUI Builder

Practice code-centric GUI building

- Load UI definitions at run-time and bind behaviors to them
  - Decoupling UI definitions from a specific GUI builder is a good idea but impractical
- Dynamic and static GUI building can be mixed easily
- Very easy to tweak a visually built GUI

Runtime Form Loading API

```
public abstract class RuntimeForm {
    public JComponent getRootComponent();
    public JComponent getComponent(String name);
}

public class RuntimeFormFactory {
    public RuntimeForm getRuntimeForm(String key) { ... }

    // for eager caching of key forms
    public void cacheRuntimeForm(String key) { ... }
}
```

JFormDesigner Runtime Form Loading Implementation

```
// exception handling hidden
// JFormDesigner-specific API in green
public class RuntimeFormJFormDesigner
        extends RuntimeForm {
    private FormCreator creator;

    public RuntimeFormJFormDesigner(FormCreator fc) {
        creator = fc;
    }

    public JComponent getRootComponent() {
        return (JComponent) creator.create();
    }

    public JComponent getComponent(String name) {
        return (JComponent) create.getComponent(name);
    }
}
```

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Avoid Swing's Default Layout Managers

- Save yourself the trouble of learning how all the default layout managers work and how to combine them, etc.
- Everything you need is in two modern layout managers
  - JGoodies FormLayout
  - Sun's GroupLayout (new)

DEMO

- The first three tips in action
  - Cross-platform look and feels; WinLAF and Quaqua
  - GUI builders
  - Better layout

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Externalize Widget Styling

- Manually styling widgets leads to inconsistencies
  - And, frankly, is a pain
- Manual widget styling can be almost impossible to get right
  - e.g., setting a font to bold in most GUI builders results in hard-coding the font family/type
- Think CSS (Cascading Style Sheets) for Swing

CSS Review

- CSS provides simple and powerful styling for the Web
  - HTML

    ```
    <div id="foo"> Ajax sucks, Swing rocks ;-)
    </div>
    <div class="bar"> … </div>
    <p class="bar"> … </p>
    ```
  - CSS

    ```
    #foo { font-family: Arial,sans-serif;
    border: 1px solid black }
    .bar { margin: 4pt }
    ```

CSS for Swing

- Why not do the same for Swing?
- Use client properties to assign selectors

```
org.galbraiths.clarity.styleClass
org.galbraiths.clarity.styleId (or use Swing's name property)
```

- Use a syntax like CSS to do styling
  - Via external file

```
JTextField.mySyleClass {
      font-size: -2pt;
      font-weight: bold;
      font-family: Courier New;Courier;
  }
```

  - Via code

```
JComponent.putClientProperty("style", "font-size:
  -2pt; …");
```

Applying Styles to Swing Components

```
JFrame frame = new JFrame("My Frame");

RuntimeForm form =
        RuntimeFormFactory.getRuntimeForm("Foo");
frame.getContentPane().add(form.getRootComponent());

FormDecorator.decorate(frame.getContentPane());

frame.setVisible(true);
```

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing's Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing's Action

Introduce a Form Concept

Employ Declarative Widget Configuration

- Performing common configuration on widgets can be needlessly tedious
  - Tables are the best example: consider the amount of code required to center the contents of a column
- A declarative widget configuration system helps dramatically
  - DSL, XML, a properties file, or whatever else you prefer

DEMO

- The next two tips in action
  - Externalized widget styling
  - Declarative widget configuration

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Use Binding and Validation Frameworks

- Getting/setting values on widgets and converting them to the appropriate type is tedious
  - So is displaying meaningful error messages to the user
- Binding and validation frameworks perform all of this plumbing for you

Binding Frameworks

- The key architectural decision for binding frameworks
  - When are values copied from the widgets to the beans?
- Options
  - Use PropertyChangeListeners and Swing listeners
    - Manually invoke "firePropertyChanged" in all setters
    - Use AOP to provide this support automatically
  - Copy values at explicit moments
    - e.g., copyValuesFromUI(), copyValuesToUI()
  - Hybrid approach
    - Use listeners with widgets but explicitly copy from beans

Binding Frameworks

- Key binding frameworks
  - JGoodies Bindings
  - SwingLabs Bindings
- Key validation frameworks
  - JGoodies Validation

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Enhance Swing's Action

- Event handling in Swing has a few weaknesses
  - Disabling components properly is tricky
  - Threading can be painful and tedious
  - Reusing event handling logic across multiple event types is tedious
- Action can be subclassed and enhanced to solve these problems
  - You can also add a lot of convenience functionality to action in the process

Simplified Listener API

- SWT introduced a generic listener API
- Enhanced Actions can emulate this approach
  - bindAction(action, component, Event.MouseClicked)
- You can define a sensible, default event mapping for components that don’t natively support actions
  - e.g., bindAction on a JTable binds to selection changing

Agenda: The Eight Tips

Use a Cross-platform Look-and-feel

Use a GUI Builder

Avoid Swing’s Default Layout Managers

Externalize Widget Styling

Employ Declarative widget Configuration

Use Binding and Validation Frameworks

Enhance Swing’s Action

Introduce a Form Concept

Introduce a Form Concept

- The act of creating a “screen”, displaying it, handling navigation, etc. involves a lot of concerns
- Standardizing how these are resolved increases development speed and productivity

DEMO

- The final three tips in action
    - Binding/validation
    - Enhanced Actions
    - Forms

Summary

- You can achieve tremendous productivity with Swing by
  - Focusing on a single look-and-feel
  - Using a GUI builder and new layout managers
  - Reducing API complexity—and the amount of code you need to write—by externalizing styling and configuration, automating binding/validation, and standardizing forms

For More Information

- The source code for this presentation is online at
  - http://www.galbraiths.org/javaone2006
- Use it in your own projects

Q&A

Eight Ways to Be More Productive Developing Swing Applications

Ben Galbraith

Swing Consultant
http://www.galbraiths.org/

TS-1913