

Swing Application Framework and Beans Binding

Simon Cook
UK Software Practice
Sun Microsystems

Agenda

- Introduction
- The Swing Application Framework
- Beans Binding
- Putting it all together
- Swing in NetBeans
- Demonstration
- Questions (and maybe some answers)

Goals

- To introduce the Swing Application Framework and Beans Binding for Swing
- To demonstration the technologies working well together
- To show how good the support in in Netbeans for Swing, SAF and Binding
- Convince you that Swing is cool again

A History Lesson

- Swing debuts in 1997
 - It's almost as old as Java
- Supersedes Abstract Window Toolkit
- Very widely adopted today
- However, there's **no** standard framework
- It's about time for one

Why Swing is Cool

- It's 100% Java (cross platform is good)
- It supports pluggable look & feels
- It provides a cross platform look & feel
- It has great support in IDEs
- It provide MVC for desktop applications
- It has some great components
- Performance is very good (now)

Why Swing is not Cool

- Has relied on some dodgy containers
- No standard framework
- Lot's of boilerplate code for simple tasks
- Multi-threading can be interesting
- Applications haven't looked native
- Performance wasn't great (but is now)

Motivation for SAF

- Consider the following code sample:

```
public static void main(String args[]) {  
    /*  
     * Best of luck with Swing, fill  
     * your boots!  
     */  
}
```

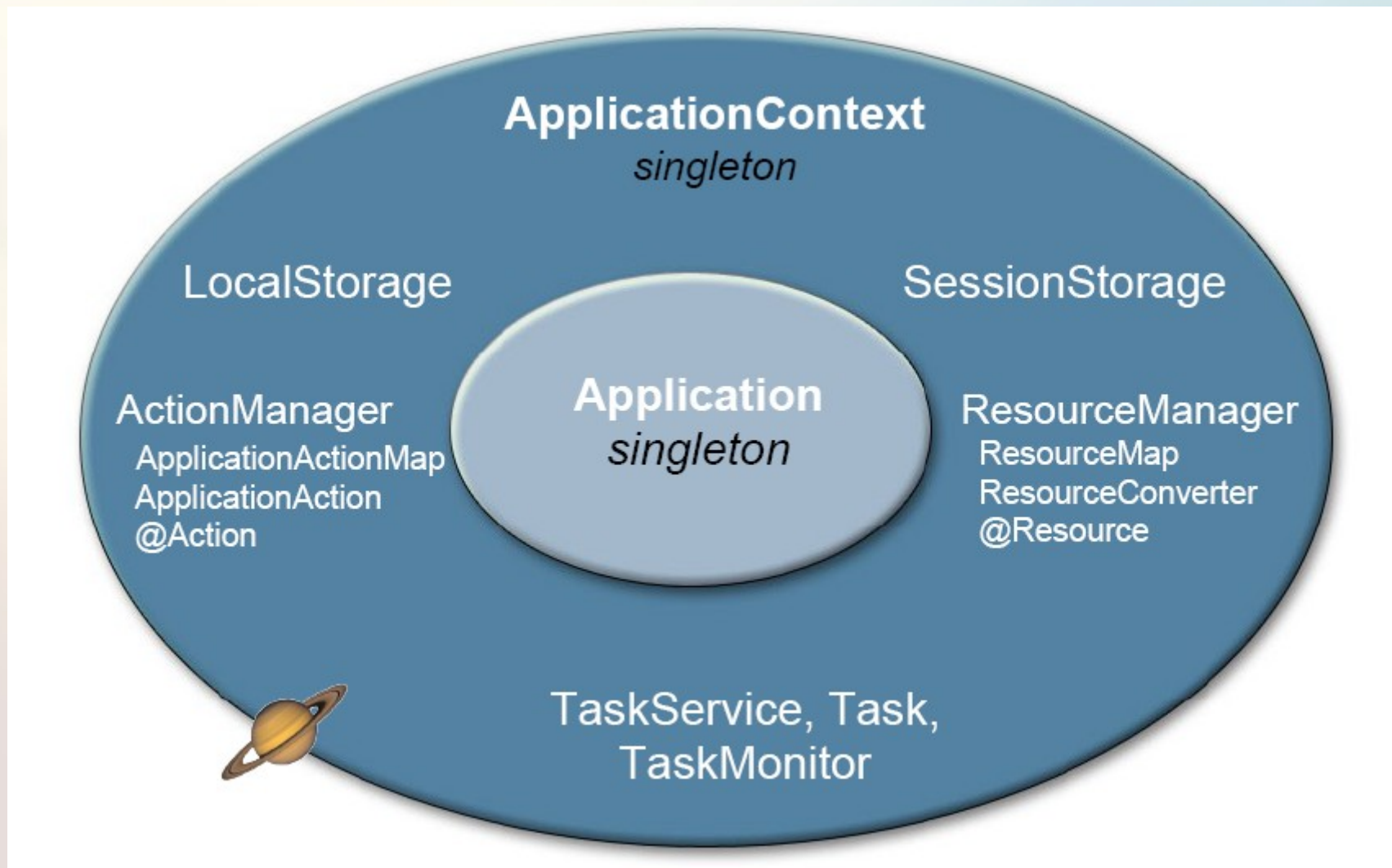
Framework Goals

- Make it simple to use
 - As small as possible
 - 1 hour to learn it
 - Small to medium apps should be simple
- 'Toolable' – Nice consequence
- Not in scope
 - Modularity, docking, file systems, data model, help systems, updates, etc.
 - It's not a Rich Client Platform replacement

Frame Core Features

- Lifecycle Management
- Resource Management
- Actions
- Tasks
- Session State

Framework Architecture



Lifecycle Management

- You're on your own

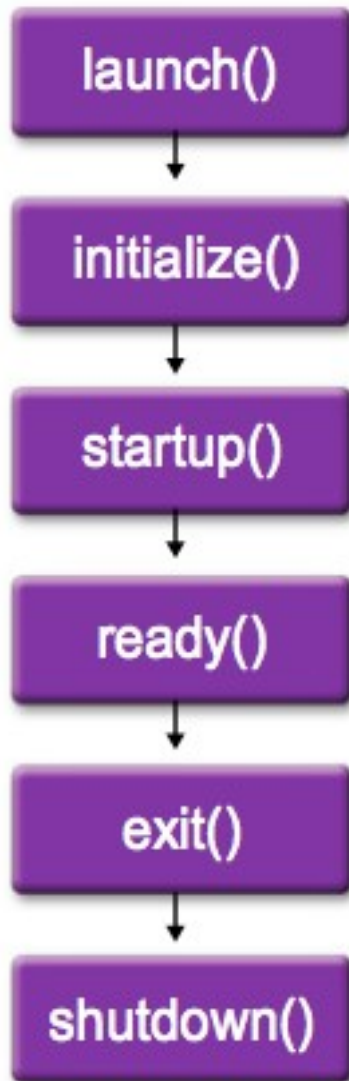
```
public class HelloWorldSwing {  
  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("HelloWorldSwing");  
        final JLabel label = new JLabel("Hello World");  
        frame.getContentPane().add(label);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

Lifecycle Management

- Using the Swing Application Framework

```
public class MyApp extends SingleFrameApplication {  
  
    @Override protected void startup() {  
        JLabel label = new JLabel("Hello World");  
        show(label);  
    }  
  
    public static void main(String[] args) {  
        //SingleFrameApplication is a subclass  
        Application.launch(MyApp.class, args);  
    }  
}
```

SAF Lifecycle Support



- Calls `initialize()`, `startup()` and `ready` on EDT thread
- Performs initializations that must happen before the GUI is displayed (e.g. Look & Feel)
- Creates the initial GUI and shows it. All applications will override this method
- Any works that must wait until the GUI is visible and ready for input
- Calls `shutdown()` unless `exitListeners` do not veto. Main frame's `WindowListener` calls `exit()`
- Take the GUI down and do final cleanup

Resource Management

- Defined using normal ResourceBundles
- ResourceManager is used
- Three options available:
 - Manual load using ResourceMap objects
 - Automatically injects resource in UI components
 - Automatically injects resources in object fields

Manual Resource Management

```
#resources/MyForm.properties
aString = Just a string
aMessage = Hello {0}
anInteger = 123
aBoolean = True
anIcon = myIcon.png
aFont = Arial-PLAIN-12
colorRGBA = 5, 6, 7, 8
color0xRGB = #556677
```

```
ApplicationContext c =
    Application.getContext(MyForm.class).getInstance();
ResourceMap r = c.getResourceMap(MyForm.class);
r.getString("aMessage", "World") => "Hello World"
r.getColor("colorRBGA") => new Color(5, 6, 7, 8)
r.getFont("aFont") => new Font("Arial", Font.PLAIN, 12)
```

Resource Injection

Resource file:

```
btnShowTime.text = Show current time!  
btnShowTime.icon = refresh.png
```

Java class:

```
btnShowTime = new JButton();  
txtShowTime = new JTextField();  
  
btnShowTime.setName("btnShowTime");  
txtShowTime.setName("txtShowTime");
```

Resources – Field Injection

Resource file:

```
MyPanel.greetingMsg = Hello, %s, a string was injected!
```

Java class:

```
@Resource
String greetingMsg;

ResourceMap resource =
    ctxt.getResourceMap(MyPanel.class);
resource.injectFields(this);

String personalMsg = String.format(greetingMsg,
txtName.getText());
JOptionPane.showMessageDialog(this, personalMsg);
```

Actions

- Instead of ActionListeners
- Asynchronous actions are easy
- @Action annotations are used
 - Possible simple logic

```
@Action(enabledProperty = "changesPending")
public void save() { ... }

public boolean getChangesPending() { ... }
```

Actions

```
// define sayHello Action – pops up message Dialog
@Action public void sayHello() {
    String s = textField.getText();
    JOptionPane.showMessageDialog(s);
}
```

```
// use sayHello – set the action property
Action sayHello = Application.getContext().
getActionMap(MyForm.class, this).get("sayHello");
textField.setAction(sayHello);
button.setAction(sayHello);
```

Tasks

- Inherit the `SwingWorker` API
- Support for monitoring
 - Title, start, done, etc.
- Asynchronous `@Action` returns `Tasks`
- Easy to handle non-blocking actions

Session State

- Easily save properties
 - Window size
 - Application position
 - Sizes of components (table columns, etc.)
- `ApplicationContext.getSessionStorage()`
- `SessionStorage.save(RootComponent, filename)`
- Uses `XMLEncoder` & `XMLDecoder`

SAF vs NetBeans RPC

- NetBeans is for applications that need more
 - Windows System
 - File/Data Systems
 - Help System
 - Update System
 - Visual Designer
 - Explorer Widgets
 - Etc.
- JSR 296 is easier to learn

When and Where?

- When will SAF be part of the JDK?
 - Planned for Java SE 7
- I can't wait that long
 - Version 1.0
 - Packaged with NetBeans 6.0
 - Source available from the open project
 - <https://appframework.dev.java.net>

Beans Binding

- Keeps the properties of two object synchronised
- Source properties can be specified using an expression language
 - e.g. “`${attendee}`” or “`${attendee.name}`”
- Does not require special objects
 - `Bindings.createAutoBinding(READ_WRITE, source, sourceProperty, target, targetProperty);`

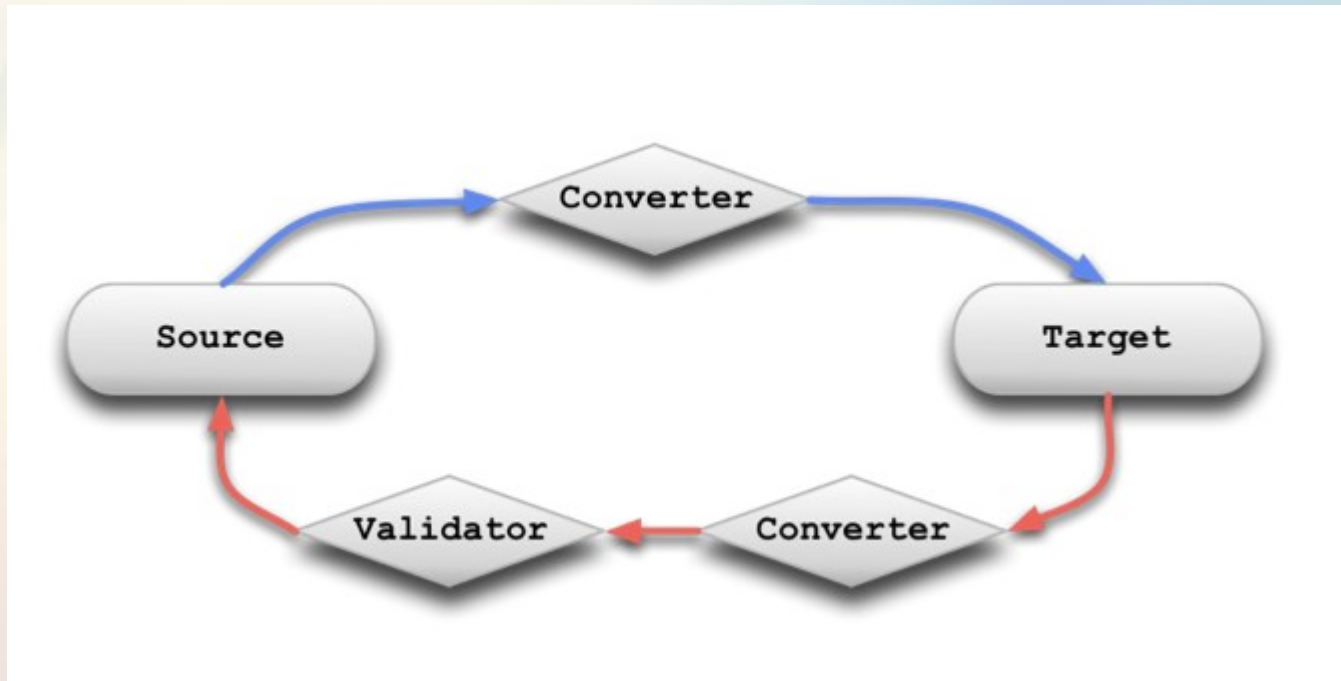
Builds Upon

- JavaBeans
 - PropertyChangeListeners
 - Doesn't require a strict following of the spec
- Collection classes
 - Standard way to encapsulate common data types

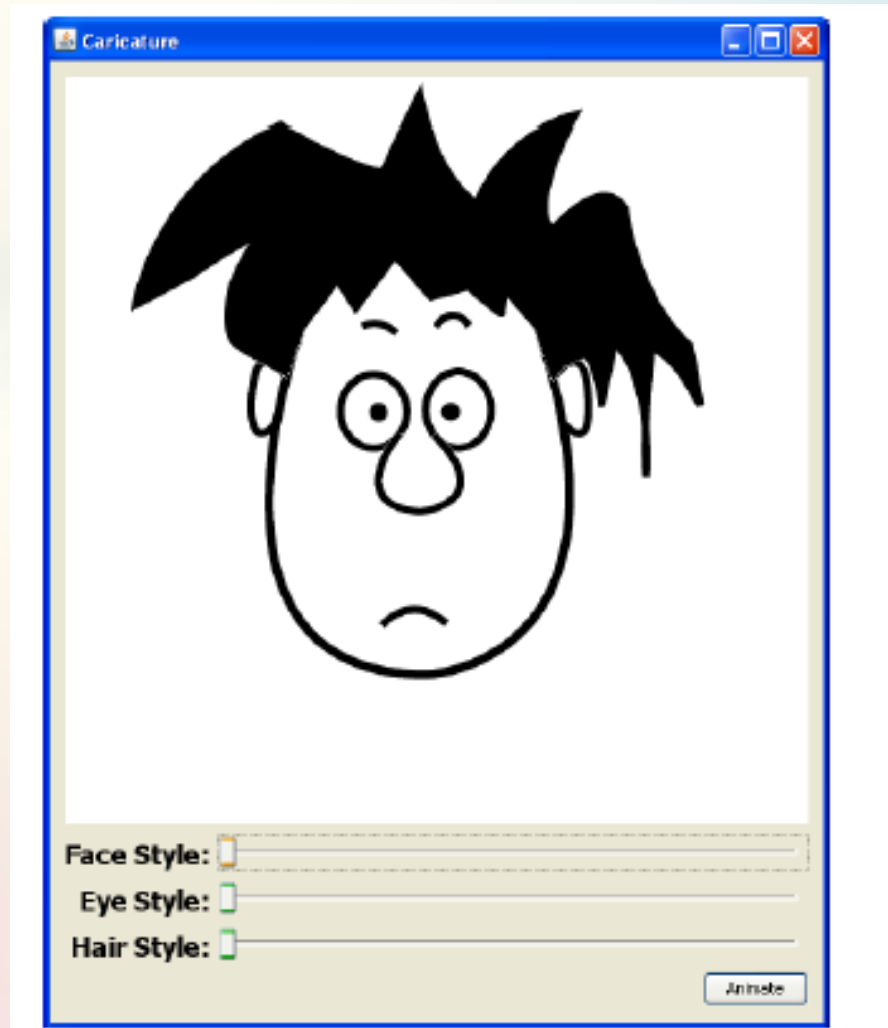
Features

- Multiple update strategies
 - Read once
 - Read only from source
 - Synchronise source and target
- Ability to do validation
- Ability to perform transformations
 - String to Date
 - Color to String

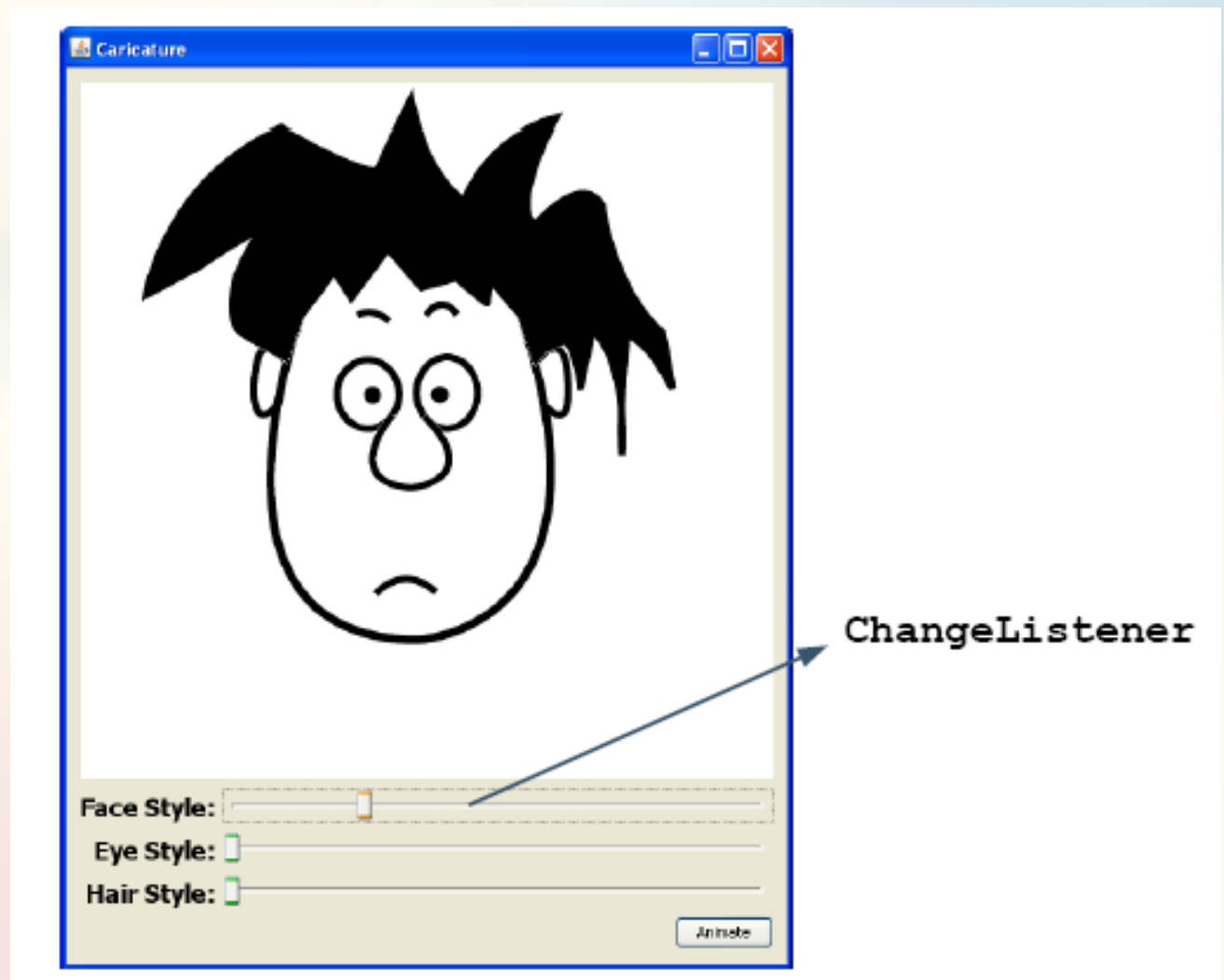
Binding



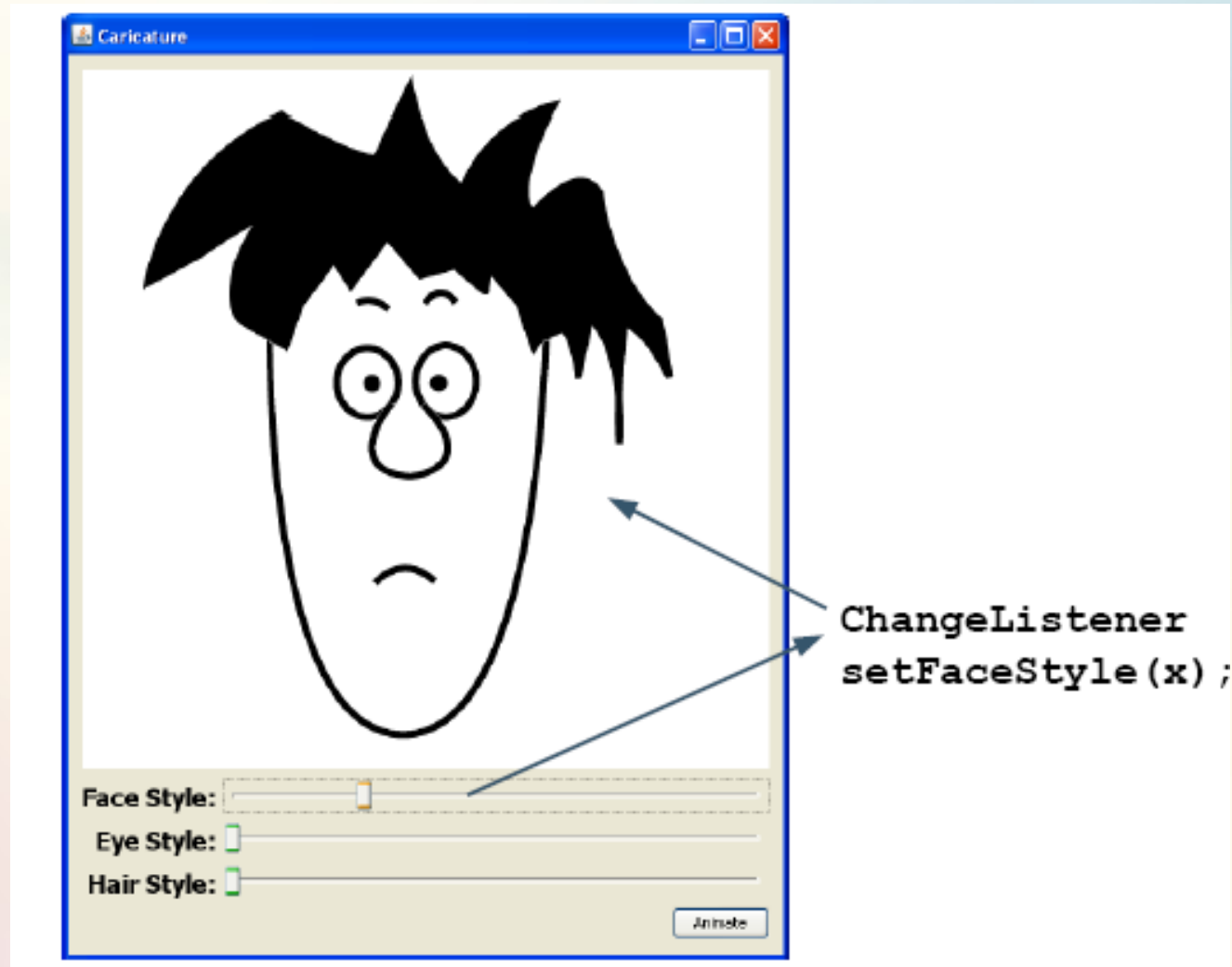
Example – Crazy Faces



Example – Crazy Faces

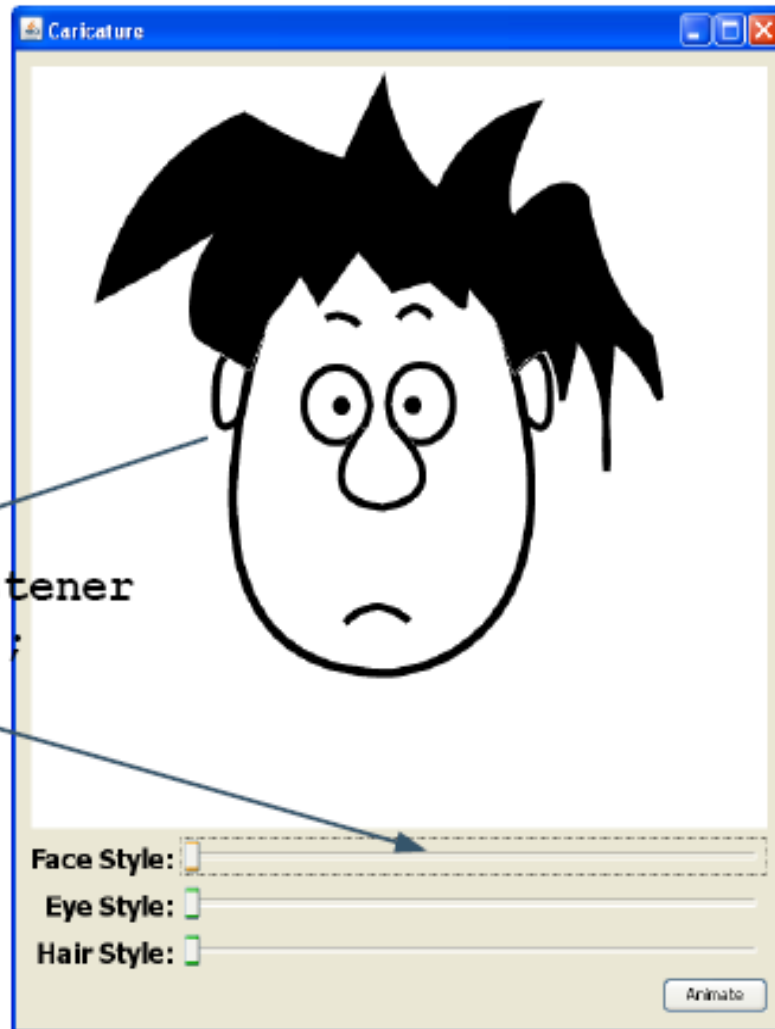


Example – Crazy Faces



Example – Crazy Faces

```
PropertyChangeListener  
slider.setValue();
```



Before JSR 295

```
faceSlider.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
        caricature.setFaceStyle(faceSlider.getValue());
    }
});
caricature.addPropertyChangeListener(new PropertyChangeListener() {
    public void propertyChange(PropertyChangeEvent e) {
        if (e.getPropertyName() == "faceStyle") {
            faceSlider.setValue(caricature.getFaceStyle());
        }
    }
});
```

With JSR 295

//Create the properties

```
Property faceStyleProp = ELProperty.create("${faceStyle}");  
Property sliderProp = BeanProperty.create("value");
```

//Create the binding for the source and target object

```
Binding binding = Bindings.createAutoBinding(UpdateStrategy.READ_WRITE,  
    caricature, faceStyleP, faceSlider, slider P);
```

//Bind the Objects

```
binding.bind();
```

When and Where?

- When will BB be part of the JDK?
 - Planned for Java SE 7
- I can't wait that long
 - Version 1.2.1 available
 - Packaged with NetBeans 6.0
 - Source available from the open project
 - <https://beansbinding.dev.java.net>

In Summary

- JSR 295 and 296 make Swing programming easier and fun again
- Both are very well supported in NetBeans 6.0 under the Mattise Project
- Java development becoming more interesting for VB and Delphi developers
- Big focus on desktop technologies for Java
- Try NetBeans 6.0 for Java GUIs

References

- The Swing Application Framework (JSR 296)
 - <https://appframework.dev.java.net/>
- Beans Binding (JSR 295)
 - <https://beansbinding.dev.java.net/>
- Beans Validation (JSR 303)
 - <http://www.jcp.org/en/jsr/detail?id=303>
- NetBeans 6.0
 - <http://www.netbeans.org>

Thank you for your time

**Swing Application Framework
and Beans Binding**



Simon Cook
simon@sun.com