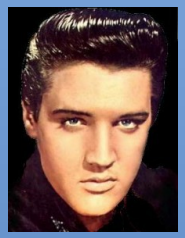
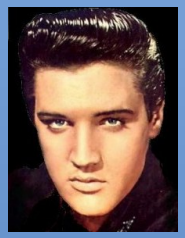

'06
JAVAPOLIS
11 - 15 DECEMBER ■ ANTWERP ■ BELGIUM
Adobe
JBoss
a division of Red Hat
ORACLE
Sun
microsystems
The Network is the Computer™

Swing Application Framework

JSR-296

Hans Muller
JSR-296 Specification Lead
Sun Microsystems



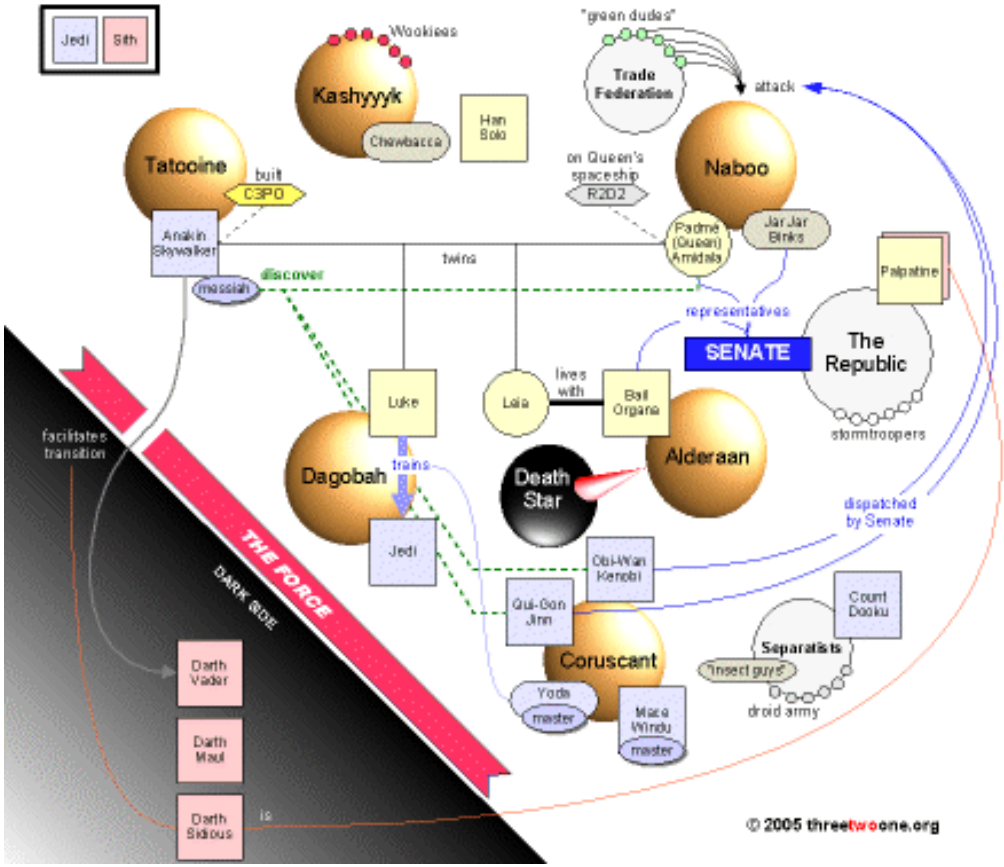


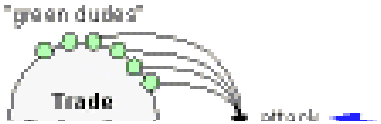
An in depth tour of the prototype JSR-296 Swing Application Framework

Speaker's Qualifications

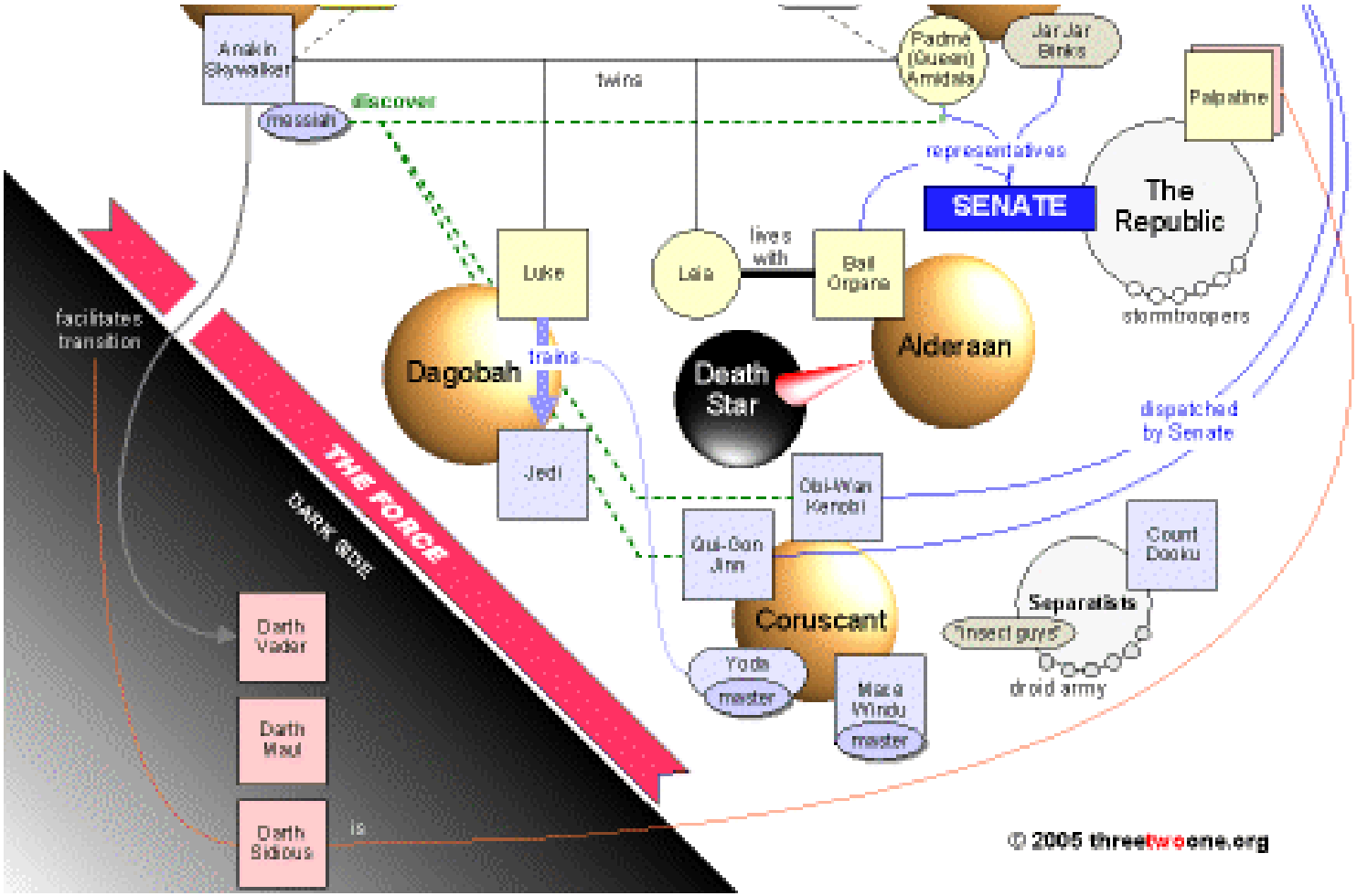
- Hans Muller is an engineer at Sun Microsystems
- He led the original Swing team and has been involved for with desktop Java for as long as we've had them.
- Hans has worked on client APIs for J2ME and J2EE, and has served as Sun's desktop CTO
- Uses NetBeans, still loves Emacs

This Slide Gains Your Audience's Attention





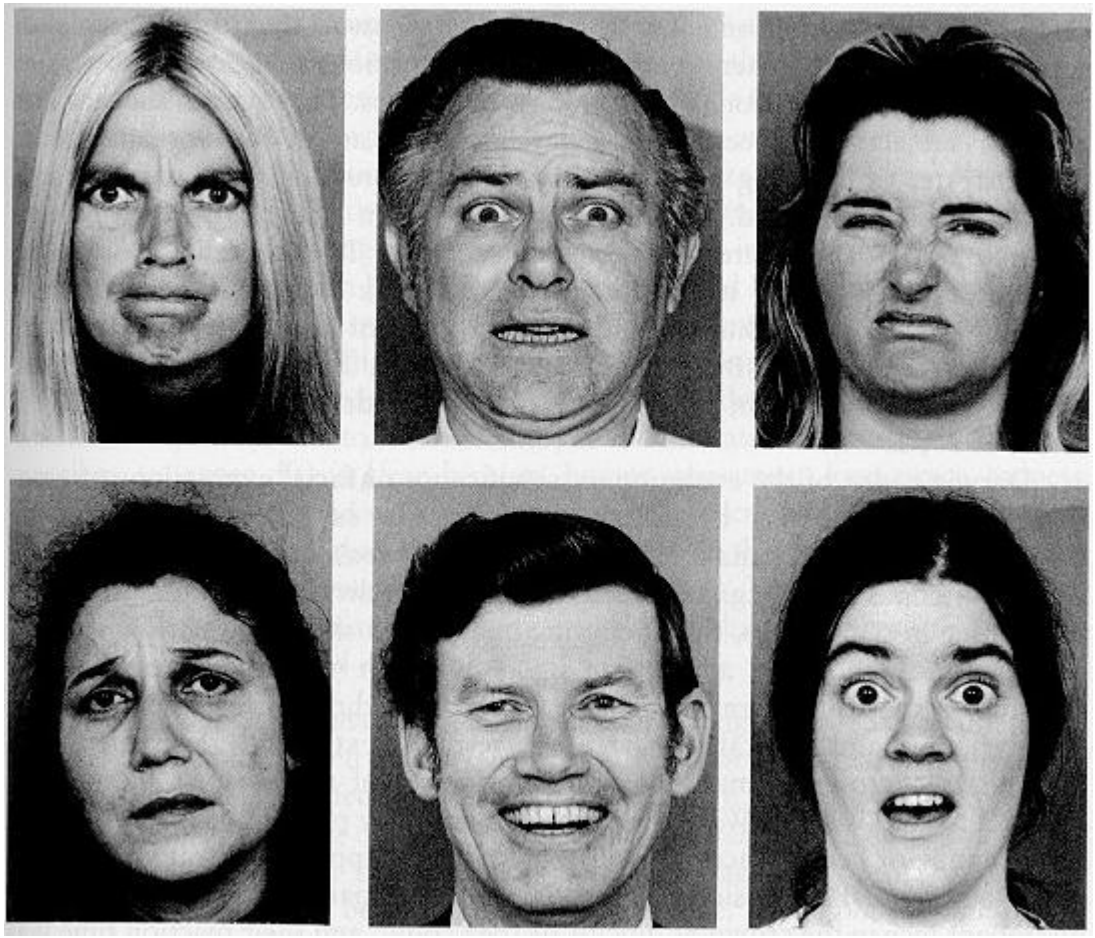
This Slide Gains Your Audience's Attention



What's the Problem?

- Swing: available for nearly a decade
  - Jillions of apps have been written without a standard desktop application framework
  - Experienced developers oftentimes actually enjoy building domain specific application frameworks
- But what about novices?
  - The Java API is pretty big
  - How do they feel it?
  - Laboratory results

Lab Results



Why a Framework is Needed

- Too many possible paths: developers freeze
  - For many developers, particularly new ones, the absence of any advice about how to structure an application is an obstacle in and of itself
  - Developers should focus on their problem domain, not on the application architecture domain
- Pave a standard road to start out on

Why a Framework is Needed

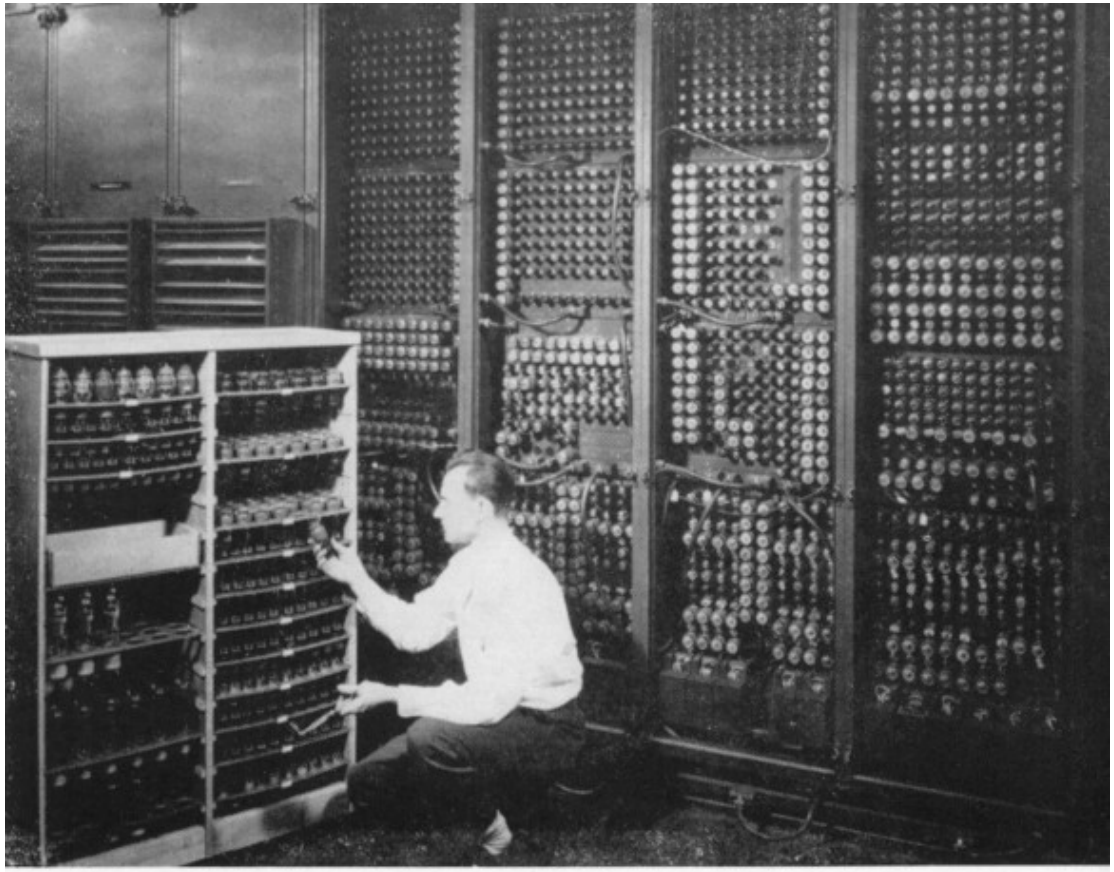
- Today's tool support: minimalist

```
public class YourDesktopApp {
    public static void main(String[] args) {
        // Good Luck!
    }
}
```

- Tool support could be much better

But, Aren't Application Frameworks Giant Scary Monsters?

- Can be too much frame, not enough work



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Not Scary

- Swing Application Framework goals
  - As small and simple as possible (not more so)
  - Explain it all in one hour
  - Work very well for small/medium apps
- No integral docking framework, generic data model, scripting language, GUI markup schema

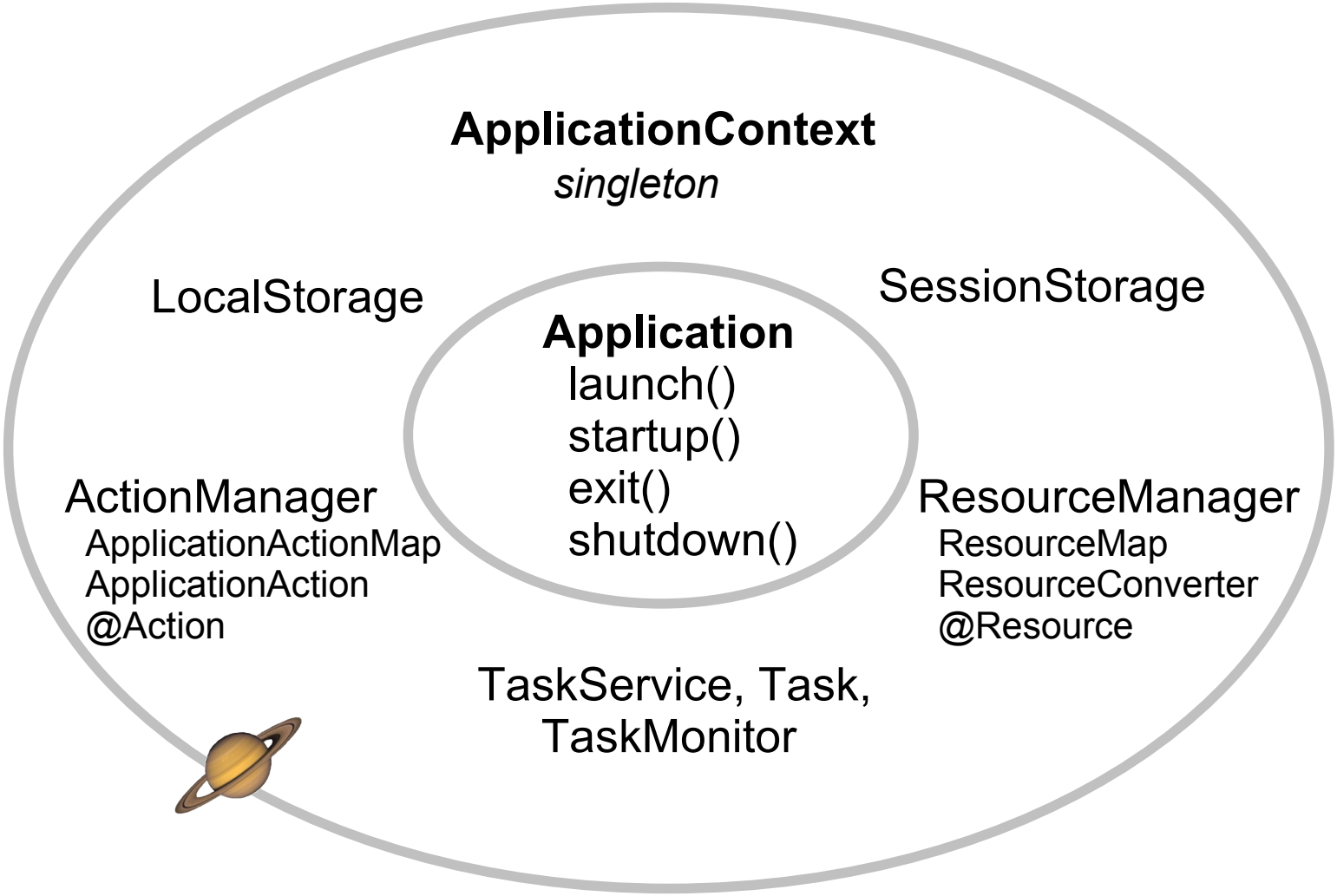
Disclaimer

- This is a review of my prototype
- The details will almost certainly change
- The fundamentals could change too

What the Framework does

- Lifecycle
- Resources
- Actions
- Tasks
- Session state

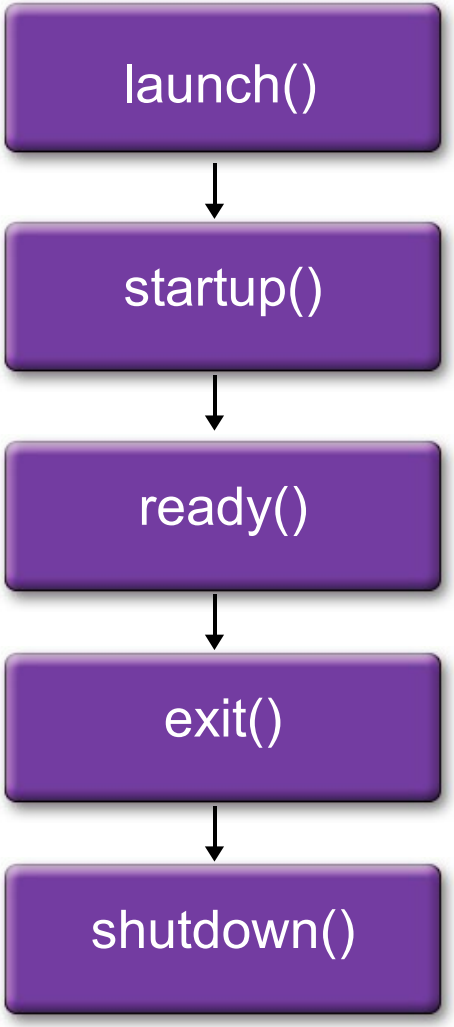
Framework Architecture Overview



The Application Class

- Application Class
- Resources
- Actions
- Tasks
- Sessions

The Application Class: Lifecycle



Call startup() on the Event Dispatching Thread.
A static method, usually called from main().

Create the initial GUI and show it.
All apps will override this method.

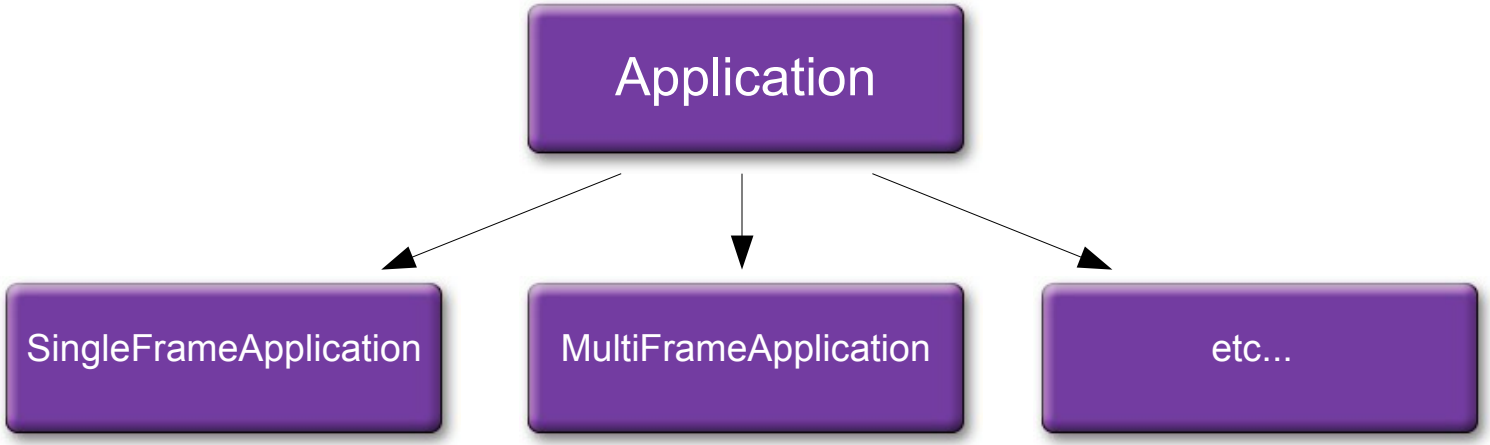
Any work that must wait until the GUI is visible and ready for input.

Call shutdown, if the exitListeners don't veto.
Main frame's WindowListener calls exit().

Take the GUI down, final cleanup.

Will my App subclass Application?

- Probably not
- Plan to provide some useful subclasses
  - For common GUI archetypes
  - It's likely you'll extend one of those instead



Application Framework: Hello World

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

How the show method works

```
protected void show(JFrame f) {
    f.addWindowListener(new FrameListener());
    f.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
    ApplicationContext c = ApplicationContext.getInstance();
    ResourceMap r = c.getResourceMap(getClass());
    r.injectComponents(f);
    f.pack();
    f.setLocationRelativeTo(null);  // center the frame
    f.setVisible(true);
}

private class FrameListener extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        exit();  // exitListeners, then shutdown()
    }
}
```

May I exit? Application exit listeners

- The exit() method checks exitListeners first

```
public interface ExitListener extends EventListener {
    boolean canExit();
}
```

- If they all return false:
  - call Application.shutdown()
  - System.exit()

Resources

- Application Class
- Resources
- Actions
- Tasks
- Sessions

Application Framework Resources

- Based on ResourceBundle
- Organized in resources subpackges
- Used to initialize properties specific to:
  - locale
  - platform
  - [TBD] look and feel
  - a few related values ...

Good old ResourceBundles

- Initial, read-only values
- Typically just strings
- Typically defined in .properties files
- Merge
  - locale-specific resources
  - locale-independent resources

ResourceMaps

- Automatically parent-chained
  - package-wide resources
  - application-wide resources
- Support *string to type* resource conversion
  - extensible
- Encapsulate list of ResourceBundles whose names are based on a class:
  - generic ResourceBundle; just the class name
  - per OS platform, class_os e.g. MyForm_OSX

Using ResourceMaps: example

```
# 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 = ApplicationContext.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

- Recall this, from the SingleFrameApplication.show() example:

```
ResourceMap r = c.getResourceMap(getClass());
r.injectComponents(mainFrame);
```

- ResourceMap.injectComponents()
  - Set the properties of named components
  - Convert types as needed

Resource Injection Example

- resourceMap.injectComponents(myPanel):

component.getName(): *label* *button1* *button2* *button3*

```
# resources/MyPanel.properties
  label.text = Choose one:
  label.font = Lucida-PLAIN-18
button1.icon = smiley.gif
button2.icon = scared.gif
button3.icon = sad.gif
```

Resource Injection Advantages

- Localizable by default
- No need to explicitly lookup/set resources
- Easy to
  - reconfigure visual app properties
  - review visual app properties
- But:
  - not intended to be a “styles” mechanism
  - not intended for general purpose GUI markup

But ... what about Actions?

Actions: review

- Encapsulation of an ActionListener and:
  - some purely visual properties
  - enabled and selected boolean properties

```
// define sayHello Action – pops up a message Dialog
Action sayHello = new AbstractAction("Hello") {
    public void actionPerformed(ActionEvent e) {
        String s = textField.getText();
        JOptionPane.showMessageDialog(s);
    }
};

// use sayHello – set the action property
textField.setAction(sayHello);
button.setAction(sayHello);
```

The *sayHello* Action in Action

- Disable the sayHello Action:

```
sayHello.setEnabled(false);
```

Actions: what we like

- Encapsulation of default GUI + behavior
- The enabled and selected properties
- Reusability

What we're not so happy about

- Overhead: creating Action objects is a pain
- Visual properties should be localized!
- Asynchronous Actions are difficult
- Proxy linkages can be messy
- It's tempting to make a little spaghetti:
  - backend logic that depends on Actions: find all the actions you need to enable/disable

The @Action Annotation

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

// use sayHello – set the action property
Action sayHello = getAction("sayHello");
textField.setAction(sayHello);
button.setAction(sayHello);
```

- ActionEvent argument is optional
- public methods only (for now)
- Used to define a “sayHello” ActionMap entry

@Actions, Class => ActionMap

```
// private utility method: look up an action for this class
Action getAction(String name) {
    ApplicationContext c = ApplicationContext.getInstance();
    ActionMap actionMap = c.getActionMap(getClass(), this);
    return actionMap.get(name);
}
```

- ApplicationContext.getActionMap()
  - creates an Action for each @Action method
  - default key is the action's method name
  - creates and caches an ActionMap
- You don't really need getAction() ...

@Action resources

- Loaded from the class's ResourceMap
- Component's action property can be injected too ...

```
# resources/MyForm.properties

sayHello.Action.text = Say &Hello
sayHello.Action.icon = hello.png
sayHello.Action.accelerator = control H
sayHello.Action.shortDescription = Say hello modally

textField.action = sayHello
button.action = sayHello
```

@Action enabled/selected linkage

- @Action parameter names bound property
  - The rest of the app depends on the property, not the Action object
  - You can use simple property expressions
- @Action(enabledProperty = “name”)
- @Action(selectedProperty = “name”)

@Action enabledProperty example

```
// Defines 3 Actions: revert, save, delete
public class MyForm extends JPanel {
    @Action(enabledProperty = "changesPending")
    public void revert() { ... }

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

    @Action(enabledProperty = "!selectionEmpty")
    public void delete() { ... }

    // These properties are bound, when they change
    // PropertyChangeEvents are fired
    public boolean getChangesPending() { ... }
    public boolean isSelectionEmpty() { ... }

    // ...
}
```

One @Action, Multiple Looks

button1 *button2* *button3*

```
# resources/MyForm.properties
sayHello.Action.text = Say Hello
sayHello.Action.icon = hello.png
button1.action = sayHello
button2.action = sayHello
button2.text = ${null}
button3.action = sayHello
button3.icon = ${null}
```

- Override Action's visual properties
  - action resource is set first
  - other resources override action's visuals
- Common case: Menu/Toolbar/Button

Tasks

- Application Class
- Resources
- Actions
- Tasks
- Sessions

Don't block the EDT

- Use a background thread for
  - computationally intensive tasks
  - tasks that might block, like network or file IO
- Background thread monitoring
  - starting, interrupting, finishing
  - progress
  - messages
  - descriptive information
- SwingWorker: most of what we need

Asynchronous @Actions: Tasks

- Task isa SwingWorker isa Future
  - Futures compute a value on thread
  - They can be canceled/interrupted
- SwingWorker adds:
  - EDT done() and PropertyChange methods
  - publish/process for incremental results
  - progress property – percent complete
- Tasks: more support for monitoring

Tasks: tell me about yourself

- Task title, description properties
  - For users
  - Initialized from ResourceMap
- Task message property, method
  - myTask.setMessage("loading " + nThings)
  - myTask.message("loadingMessage", nThings)
    (resource) loadingMessage = loading {0} things
- Task start/done time properties
- Task useCanCancel property

Asynchronous @Action Example

```
// Say hello repeatedly
@Action public Task sayHello() {
    return new SayHelloTask();
}

private class SayHelloTask extends Task<Void, Void> {
    @Override protected Void doInBackground() {
        for(int i = 0; i <= 10; i++) {
            progress(i, 0, 10);    // calls setProgress()
            message("hello", i); // resource defines format
            Thread.sleep(150L);
        }
        return null;
    }
    @Override protected void done() {
        message(isCancelled() ? "canceled" : "done");
    }
}
```

Asynchronous @Actions that Block

- @Action annotation *block* parameter:
  - @Action(block = Block.NONE) – default
  - @Action(block = Block.ACTION)
  - @Action(block = Block.COMPONENT)
  - @Action(block = Block.WINDOW)
  - @Action(block = Block.APPLICATION)
- Resources for blocking (modal) dialogs

```
stop.Action.BlockingDialog.title = Blocking Application
stop.Action.BlockingDialog.message = Please wait ...
stop.Action.BlockingDialog.icon = wait.png
```

TaskServices

- Defines how a Task is executed, e.g.
  - serially
  - by a thread pool
  - etc..
- TaskService isa ExecutorService
  - named, constructed lazily
  - @Action(taskService = “database”)
- Application.getTaskServices()

Monitoring Tasks: TaskMonitor

- Desktop apps often run many threads
- TaskMonitor provides a summary
  - Bound properties, same as Task
  - Foreground task: first one started
- Handy for StatusBar implementations

Action and Tasks Summary

- Define Actions with @Actions, resources
- Link enabled/selected to a property
  - @Action(enabledProperty = “name”)
  - @Action(selectedProperty = “name)
- Asynchronous @Actions return Tasks
  - Provide title/description resources
  - Use message/progress methods/properties
  - Use the block parameter and resources
- Connect your status bar to a TaskMonitor

Sessions

- Application Class
- Resources
- Actions
- Tasks
- Sessions

Session State

- Make sure the application remembers where you left things.
- Most applications should do this
  - but they don't
  - what state to save?
  - where to store it (and what if you're unsigned)?
  - how to safely restore the GUI

SessionStorage

- ApplicationContext.getSessionStorage()
  - save(rootComponent, filename)
    - Supported types, named components only
    - Window bounds, JTable column widths, etc
    - archived with XMLEncoder
  - restore(rootComponent, filename)
    - conservative
    - restored with XMLDecoder
- LocalStorage abstracts per-user files

~\Application Data\Sun\MyApp\session.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0-rc" class="java.beans.XMLDecoder">
 <object class="java.util.HashMap">
  <void method="put">
   <string>mainFrame</string>
   <object class="application.SessionStorage$WindowState">
    <void property="bounds">
     <object class="java.awt.Rectangle">
      <int>436</int>
      <int>173</int>
      <int>408</int>
      <int>424</int>
     </object>
    </void>
    <void property="graphicsConfigurationBounds">
     <object class="java.awt.Rectangle">
      <int>0</int>
      <int>0</int>
      <int>1280</int>
      <int>800</int> ...
```

DEMO

Summary

- Swing Application Framework supports
  - actions, resources, tasks, sessions
  - Application and ApplicationContext singletons
  - you have to subclass Application
- JSR-296 expert group is responsible
  - for defining the framework's final form
  - finishing in time for Java 7

Watch javadesktop.org for announcements about the prototype code being available.

Build a Java Desktop Application.

Break free from the browser's chains!

Q&A

www.javapolis.com