



the
POWER
of
JAVA™



JavaOne
Part of the Network and Business Solutions

A Simple Framework for Desktop Applications

Hans Muller

Software Engineer
Sun Microsystems
www.sun.com

TS-3399

What You Will Need to Learn

For the **test** at the conclusion of the presentation*

The Swing Application Framework
Project: why we're doing it, what it is,
why you'll want it

*OK, there's no test. Just try and stay alert, or you'll miss the demo at the end.

Agenda

What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

- Application Class

- Lifecycle: starting up, shutting down, milestones

- Actions and Resources

Demo!

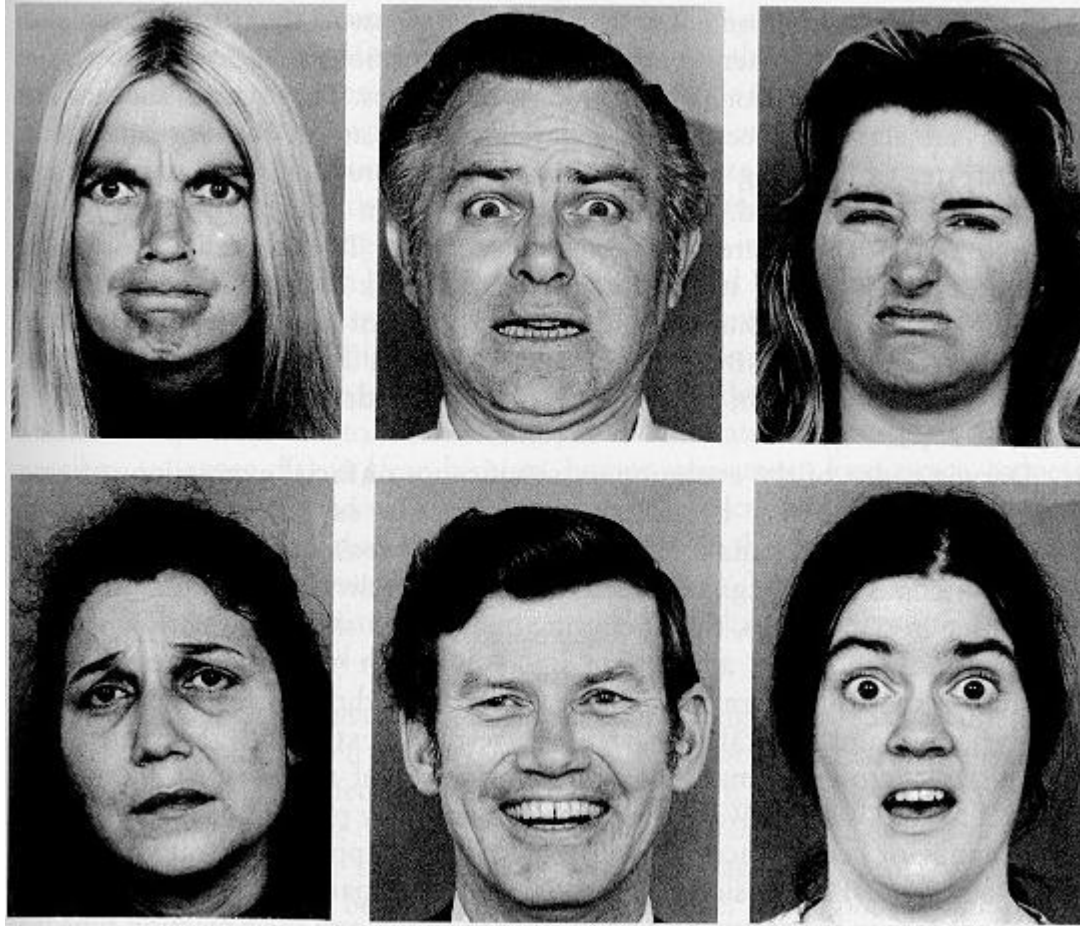
Where we're headed, the JSR

Disclaimer

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

What's the Problem?

- Swing has been 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 API **is** pretty big
 - How do they feel about building apps from scratch?
- Laboratory results



Reasons Why A Desktop App Framework is Needed: For Starters

- 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

Reasons Why A Desktop App Framework is Needed: For Starters

- There are some attractive bad paths
 - Build the app on the main thread
 - Your app IS-A JFrame
 - Tangle of actionPerformed methods block the EDT
 - Just English is good enough
- Make getting to the finish line more likely

Reasons Why A Desktop App Framework is Needed: For Starters

- Today's tool support: minimalist

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

- Tool support could be much much better

Desktop App Framework Fears

Aren't App Frameworks Giant Scary Monsters?

- Can be too much **frame** not enough **work**
 - Over design
 - Try and do too much
 - A shrine for great hacks
- Swing App Framework goals
 - As small and simple as possible (but not more so)
 - Explain it all in one hour
 - Work very well for small and medium scale apps
 - No integral docking framework, generic app data model, scripting language, GUI definition schema ...

Agenda

What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

- Application Class

- Lifecycle: starting up, shutting down, milestones

- Actions and Resources

Demo!

Where we're headed, the JSR

A Very Brief Survey of What Exists

- NetBeans™ Platform
- Spring RCP
- Eclipse RCP

Agenda



What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

Application Class

Lifecycle: starting up, shutting down, milestones

Actions and Resources

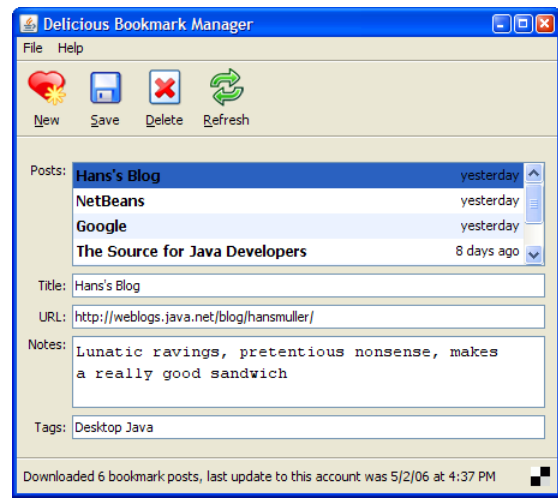
Demo!

Where we're headed, the JSR

Overview: The Obligatory Diagram

Elements of the application framework

State and Behavior

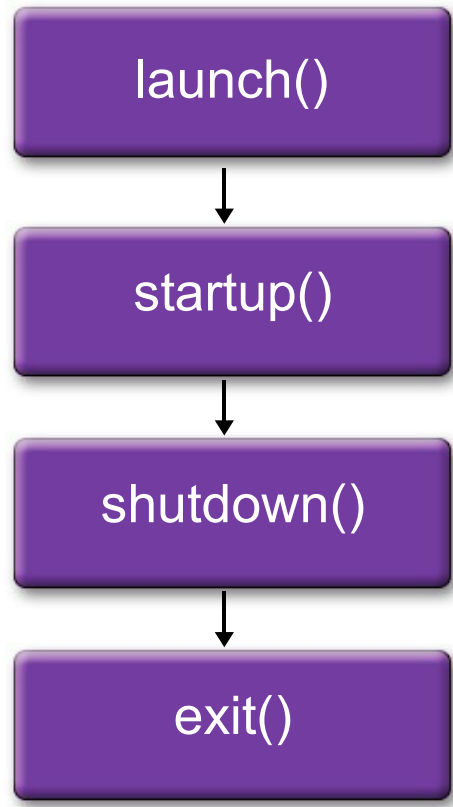


Resources

Preferences

Actions

LifeCycle



Application Class

- Base class for Desktop Java™ based applications
- Defines lifecycle: startup, shutdown, etc
- Provides access to
 - Resources
 - Actions
 - Preferences

```
public class Application {
    protected Application()
    public static synchronized Application getInstance()
    public static synchronized <T extends Application>
        void launch(Class<T> appClass, String[] args)

    protected void startup(String[] args)
    protected void shutdown()
    public String getMilestone()
    public void setMilestone(String milestone)

    public ResourceMap getResourceMap(Class cls)
    public ActionMap getActionMap(Class cls)
    public Preferences getPreferences(Class cls)

    // Boilerplate for supporting bound Java Beans properties ...
    public void addPropertyChangeListener(PropertyChangeListener l)
    public void removePropertyChangeListener(PropertyChangeListener l)
    public PropertyChangeListener[] getPropertyChangeListeners()
    public void addPropertyChangeListener(String name, PropertyChangeListener l)
    public void removePropertyChangeListener(String propertyName, PropertyChangeListener l)
    public PropertyChangeListener[] getPropertyChangeListeners(String name)
    protected void firePropertyChange(String name, Object oldValue, Object newValue)
}
```


Define an Application Subclass and Launch!

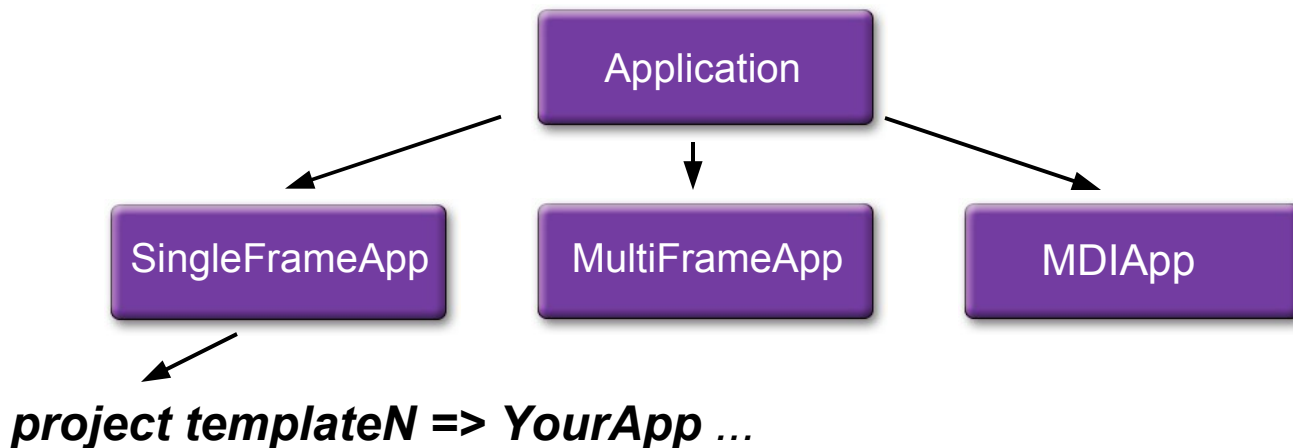
```
class MyApp extends Application {
    protected void startup(String[] args) {
        JFrame frame = new JFrame("My App");
        frame.add(new JLabel("Hello World"));
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        Application.launch(MyApp.class, args);
    }
}
```

Application.launch(MyApp.class, args);

- MyApp.class is constructed on the EDT
- MyApp.startup() runs on the EDT
 - Command line argument processing errors can show an error dialog and/or log a warning
- Application.getInstance() returns **the** instance of MyApp

Will My App Subclass Application?

- Probably not
- A few standard Application subclasses will be provided
 - Support some useful GUI archetypes
 - App “shells” encapsulated by IDE project **templates**



Agenda



What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

Application Class

Lifecycle: starting up, shutting down, milestones

Actions and Resources

Demo!

Where we're headed, the JSR

Application Lifecycle

- `Application.startup()`
 - Called by `Application.launch()` on EDT
 - Create and show the GUI
 - Milestone changes mark progress
- `Application.shutdown()`
 - Called on EDT when application attempts to exit
 - “Mother may I” `exitListeners` can veto

Startup Milestones

- Startup progress reported via a bound property
 - `get/setMilestone(String milestone)`
- Standard milestones are
 - `before/afterResourcesLoaded`
 - `before/afterGUICreated`
 - `before/afterGUIRealized`
 - `before/afterGUIShown`
 - `afterGUIReady`
- `Application.startup()` is responsible for calling `setMilestone`

May I exit? `ExitListener`

ExitListeners must confirm app exit

```
Application app = Application.getInstance();
app.addExitListener(new ConfirmExitListener());

class ConfirmExitListener implements ExitListener {
    public boolean canExit() {
        String message = rMap.get("confirmExit.message");
        String title = rMap.get("confirmExit.title");
        int option = JOptionPane.showConfirmDialog(
            mainFrame, message, title,
            JOptionPane.YES_NO_OPTION);
        return (option == JOptionPane.YES_OPTION);
    }
}
```

Agenda



What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

- Application Class

- Lifecycle: starting up, shutting down, milestones

- Actions **and Resources**

Demo!

Where we're headed, the JSR

Good Old ResourceBundle

- Initial, read-only values
- Typically just strings
- Typically defined in “.properties” files

Resource Bundles in the Framework

Resources for **MyClass** in `mypkg/resources/MyClass.properties`

```
myLabel.text=Hello World  
myLabel.icon=hello-world.png
```

Resources for **mypkg** in `mypkg/resources/mypkg.properties`

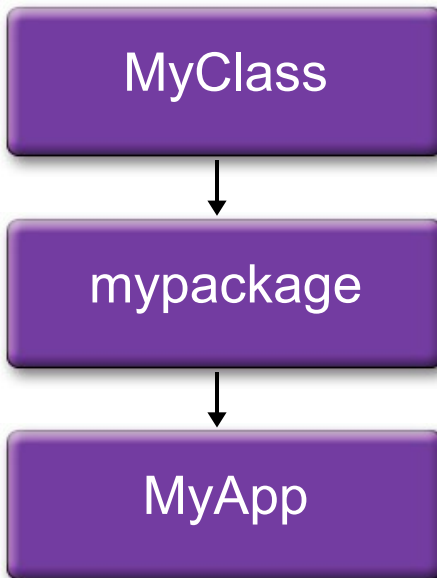
```
myPackage.webSvcErrMsg={0} failed because {1}  
myPackage.displayBackground=#556677
```

Resources for **MyApp** in `mypkg/resources/MyApp.properties`

```
Application.title=My Application  
Application.vendor=Sun Microsystems, Inc.
```

ResourceMaps

- Encapsulate a small set of ResourceBundles
 - Read-only, key set doesn't change dynamically
 - Chained together with "parent" links



```

ResourceMap myRM =
    myApp.getResourceMap(MyClass.class);
myRM.getString("myLabel.text")
    => "Hello World"
myRM.getString("Application.title")
    => "Bookmark Manager"
  
```

ResourceMap API: Convert and Cache

StringConverter: Convert resource strings to type

```
public class ResourceMap {
    public Object getObject(String key, Class type)
    // ...
    public static abstract class StringConverter {
        protected StringConverter(Class type)
        public boolean supportsType(Class testType)
        public abstract Object
            convertString(String s, ResourceMap r)
    }
    public static
        void addStringConverter(StringConverter sc)
    public static
        StringConverter stringConverterFor(Class type)
}
```

ResourceMap: StringConverters for common Desktop GUI types

- ResourceMap.getIcon(), Color, Font, KeyStroke, KeyCode, ...
- ResourceMap.getString(), Boolean, Integer...
- Looking up message strings with arguments:

```
String getString(String key, Object... args)
```

```
=>
```

```
getObject(key, MessageFormat.class).format(args)
```

Example: String Resources

```
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

# resourceMap.getString("aMessage", "World");
# resourceMap.getColor("colorRGBA");
# resourceMap.getFont("aFont");
```

ResourceMap API: Component Resource Injection

- Set component properties whose names match a resource name

```
public class ResourceMap {  
    // ...  
    public void injectComponent(Component target)  
    public void injectComponents(Component root)  
}
```

- Inject just one component, or all components in a hierarchy
- Resource names must match:
`component.getName () + ".propertyName"`
- Resource type, same as property type

Example: Injecting Component Resources

- `resourceMap.injectComponents(myToolBar)`



```
# resources/MyApp.properties ResourceBundle
    newButton.text = New
    newButton.icon = new-icon.png
    saveButton.text = Save
    saveButton.icon = save-icon.png
    deleteButton.text = Delete
    deleteButton.icon = delete-icon.png
refreshButton.text = Refresh
refreshButton.icon = refresh-icon.png
```


Resources for All Properties?

- No, no, no
- Use them for values that might vary by:
 - Locale
 - Platform
 - Look and feel
 - Deployment
- GUI Builders, like NetBeans Matisse software, are the best way to configure the rest

ResourceMap API: Field Injection, @Resource

- Initialize fields marked with **@Resource***
 - For programmatically configured GUI elements

```
public class ResourceMap {  
    // ...  
    public void injectFields(Object target)  
}
```

- Resource names must match:
`target.getClass().getSimpleName() + ".field-name"`
- Resource type, same as field type
- *Thanks to Romain Guy and Daniel Spiewak for pioneering this idea in the java.net Fuse project

Field @Resource Example

- Icon valued fields initialized from a ResourceMap:

```
public class MyForm extends JPanel {
    @Resource Icon busyIcon;
    @Resource Icon readyIcon;
    void showStatus(isBusy boolean) {
        myLabel.setIcon(isBusy ? busyIcon : readyIcon;
    }
    MyForm() {
        Application app = Application.getInstance();
        ResourceMap rMap = app.getResourceMap(MyForm.class);
        rMap.injectFields(this);
    }
}
```

- ResourceBundle properties File:

```
# resources/MyForm.properties ResourceBundle
MyForm.busyIcon = busy-icon.png
MyForm.readyIcon = ready-icon.png
```

Agenda



What's the problem?

Aren't app frameworks giant scary monsters?

A (very) brief survey of app frameworks

Swing Application Framework

- Application Class

- Lifecycle: starting up, shutting down, milestones

- Actions** and Resources

Demo!

Where we're headed, the JSR

What We've Got: Actions, ActionMaps

- Actions
 - Semantics: `actionPerformed()`, `enabled`, `selected`
 - A smattering of presentation attributes
 - `myComponent.setAction(myAction)`
- ActionMaps
 - Map names to Actions
 - Have a parent Actionmap
 - Used with InputMaps to map KeyStrokes to Actions

What We Need

- Action presentation attributes
 - Per locale, platform, look and feel
 - Per Toolbar/Menu/Button/etc context
- Actions that do work on a background thread
- ActionMaps
 - For the entire application
 - For GUI elements, like a form or a dialog
- More annotations!

Defining Actions: @Action

- One “Actions” class can define many Actions
- @Action annotation per actionPerformed method
 - ActionEvent argument is optional

```
class MyActions {
    @Action
    void saveItem() { ... }

    @Action
    void moveItem(ActionEvent e) { ... }
}
```

Define @Actions, enabled/selected

- @Action arguments
 - enabledProperty – names bound boolean property
 - selectedProperty - likewise
 - name – resource prefix, default is method name

Define @Actions, enabled/selected (continued)

```
class MyActions {  
    @Action(enabledProperty=selectedItemValid)  
    void saveItems() { ... }  
  
    @Action(enabledProperty=selectedItemValid)  
    void moveItems(ActionEvent e) { ... }  
  
    boolean isSelectedItemValid() { ... }  
    boolean setSelectedItemValid(boolean b) { ... }  
}
```

Now Make me an ActionMap

- Create an Action for each @Action:

```
Application app = Application.getInstance();
ActionMap aMap = app.getActionMap(MyActions.class)
saveButton.setAction(aMap.get("saveItem"));
moveButton.setAction(aMap.get("moveItem"));
```

- Action attributes come from a ResourceMap:

```
# resources/MyActions.properties ResourceBundle
saveItem.actionName = &Save
saveItem.actionAcceleratorKey = control S
saveItem.actionShortDescription = \
    Save the item in the data warehouse

moveItem.actionName = &Move
moveItem.actionAcceleratorKey = control D
moveItem.actionShortDescription = \
    Move the item to a new data warehouse
```

Action Attributes per “context”

- Create 3 Actions for one @Action
 - [*context*]—context specific attribute
 - Other attributes are common

```
# resources/MyActions.properties ResourceBundle
saveItem.actionName = &Save
saveItem.actionName[Menu] = Save Item to Warehouse
saveItem.actionIcon[Toolbar] = save-toolbar-icon.png
saveItem.actionAcceleratorKey = control S
saveItem.actionShortDescription = \
    Save the item in the data warehouse
```

Action Attributes per “context” (cont)

- Action names get the context qualifier

```
Application app = Application.getInstance();  
ActionMap aMap = app.getActionMap(MyActions.class)  
  
saveButton.setAction(aMap.get("saveItem"));  
  
saveMenuItem.setAction(aMap.get("saveItem[Menu]"));  
  
saveTButton.setAction(aMap.get("saveItem[Toolbar]"));
```

Asynchronous Actions, SwingWorker

- Actions need to run on a background thread if
 - They might take longer than 10-20ms
 - They might block
- SwingWorker class facilitates this

```
class DoWork extends SwingWorker<Object, Object> {
    @Override protected Object doInBackground() {
        // while you're at it:
        // call progress(0 .. 100) if feasible
        return null;
    }
    @Override protected void done() {
        // update GUI - you're on the EDT
    }
}
```

Asynchronous @Actions

- Async @Actions return a SwingWorker object

```
@Action  
SwingWorker<Object, Object> saveItems () {  
    return new DoSaveItems (getMyItems ()) ;  
}
```

Asynchronous @Actions ActionDisplay

- Status changes are reflect in the GUI via the ActionDisplay class:

```
public interface ActionDisplay {  
    public void progress(Object src, int value);  
    public void message(Object src, String text);  
    public void status(Object src, int value);  
}
```

```
Application app = Application.getInstance();  
app.getActionDisplay().message(app, "Working...")
```

- Application subclasses typically override `getActionDisplay()`

Async @Action Example

```
class DoSaveItems extends SwingWorker<Object, Object> {
    private final List<MyItem> myItems;
    DoSaveItems(List<MyItem> myItems) {
        this.myItems = myItems;
    }
    @Override protected Object doInBackground() {
        int nSaved = 0;
        for(MyItem myItem : myItems) {
            saveMyItem(myItem);
            setProgress(percentDone(nSaved++));
        }
        return null;
    }
    // continued, next slide ...
}
```


Async @Action Example

```
// ResourceBundle resources/MyApp.properties
// saveItems.doneMessage = Saved {0} items

class DoSaveItems extends SwingWorker<Object, Object> {
    // ... continued from previous slide
    @Override protected void done() {
        int nSaved = myItems.size();
        String key = "saveItems.doneMessage";
        String msg = rMap.getString(key, nSaved);
        getActionDisplay().message(this, msg);
    }
}

@Action
SwingWorker<Object, Object> saveItems() {
    return new DoSaveItems(getMyItems());
}
```

@Actions That Block

- Block keyword specifies scope:
`@Action(block=Action.Block.APPLICATION)`
- Three scopes: NONE, WINDOW, APPLICATION
- `Application.getBlockingDialog()` creates dialog
- Blocking dialog can provide an `ActionDisplay`

DEMO

A Simple Desktop Client for del.icio.us

Where We're Headed, the JSR

- This has been a review of the prototype of the Swing Application Framework
- The JSR-296 Expert Group will begin work on the standard API later this summer
- We will try and
 - Design an API that can be explained in an hour
 - Limit the scope of the framework to common generic desktop application infrastructure
 - Build upon what already exists
- Overall goal is: make building desktop applications (much) easier

Where We're Headed, the JSR (Cont.)

- Development will be similar to JSR-295
 - Public java.net project
 - Implementations planned for Java SE 1.5,1.6,1.7
 - Hope to be bundled with 1.7

Summary

- Swing developers would benefit from an application framework
- We're working on one
- It's narrow in scope: Application class, lifecycle, resources, actions
- The final framework spec will be developed through JSR-296

Q&A

Hans.Muller@Sun.COM

For More Information

- JSR 296 on <http://jcp.org>
- Related Sessions I hope you've attended
 - TS-4635: Best Practices: Data Access Strategies (Thursday, 11:00AM)
 - TS-1074: Desktop Patterns and Data Binding (Thursday, 1:30PM)



the
POWER
of
JAVA™



JavaOne
Part of the Network and Business Solutions

A Simple Framework for Desktop Applications

Hans Muller

Software Engineer
Sun Microsystems
www.sun.com

TS-3399