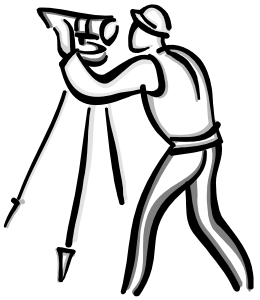
Made from the Same Mold: Templating Approaches for ADF Faces Applications

Peter Koletzke
Technical Director &
Principal Instructor

Survey

- Java development
  - 1-2 years?
  - 3-12 years?
  - More than 12 years?
- JDeveloper
  - 1-3 years?
  - More than 3 years?
  - 10.1.3.x?
  - 11*g* preview?
- Template work
  - Which technologies?



The Original Template

Nature made him,
and then broke the mould.

Natura il fece,
e poi ruppe la stampa.

—*Ludovico Ariosto* (1474–1533),
Orlando Furioso, canto x. stanza 84

Uses of The Template

Long shall we seek his likeness, long in vain,
And turn to all of him which may remain,
Sighing that Nature formed but one such man,
And broke the die—in moulding Sheridan!

—*George Gordon Noel Byron* (1788–1824),
Monody on the Death of Sheridan

We need role models
who are going to break the mold.

– Carly Simon (1945-)

Agenda

- Approaches for template use
- Common ADF look-and-feel options
- Templates with ADF Faces
- Conclusions and the Future

Slides and white paper will be on the NoCOUG website.

Why Templates?

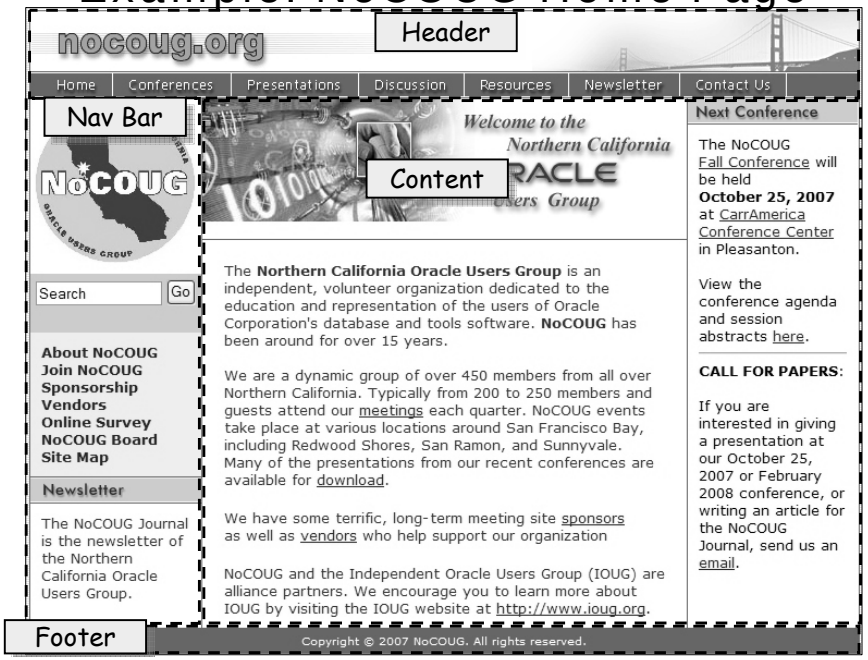
1. Enhance user productivity
   - Consistent look and feel means users can quickly grasp how an application works
   - No need to relearn a technique for each page
2. Improved maintainability
   - Common elements are in only one place
   - Changes to these elements require minimal coding
   - Reuse is a GRBP

The Process in JDeveloper

1. Select the template file in the navigator
2. Select File | Save As and locate the new directory
3. Name the file and click Save

- This creates a copy of whatever was in the template
- **Objective**: reference as much as possible in the template

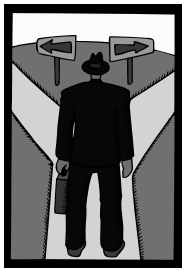
Example: NoCOUG Home Page



Approaches

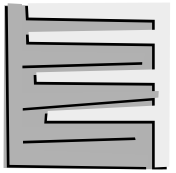
- Applies to Java EE code
  - JavaServer Pages (JSP), for example
  - Can also be applied to other types of code: standard HTML, Oracle Forms?
- Different levels of file use

1. Single common elements file
2. Multiple common elements files

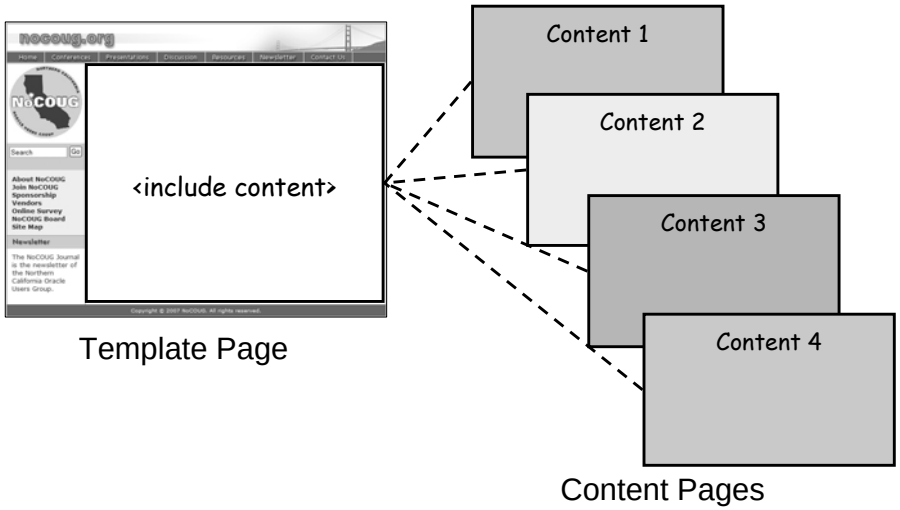


Single Common Elements File

- Template is a single JSP file
  - Includes header, footer, and nav bar
  - Includes area for contents to be included
- Controller determines which page to include
  - jsp:include
- All pages show a single template file with contents specific to the function

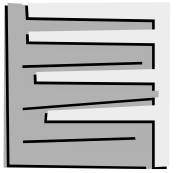


Single Common Elements File



Single Common Elements File

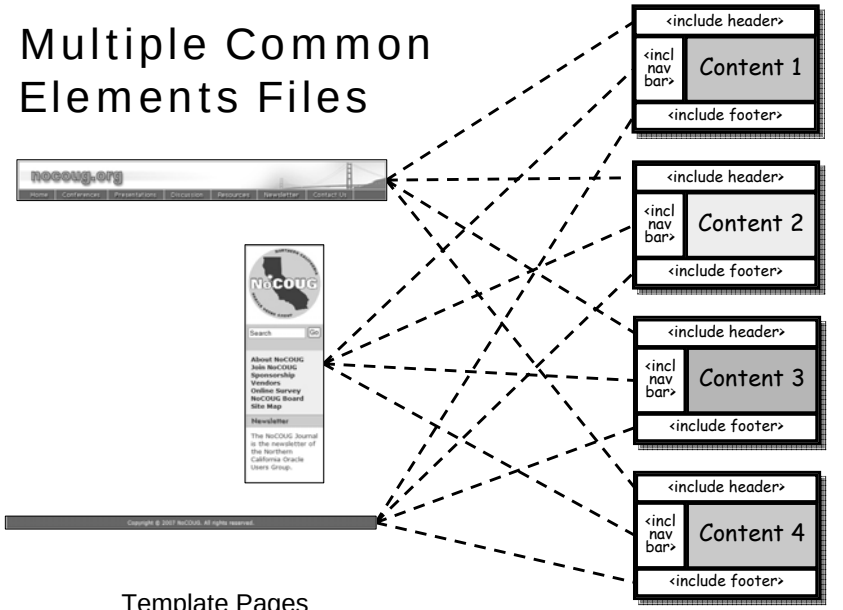
- Benefits
  - Four-cell layout does not need to be repeated on each page
    - Easier to change *look and feel* (LAF)
    - New pages are less work to create
  - Only one file to change if design changes, no change to contents pages
- Drawback
  - Non-standard use of Controller
    - Requires routing code for page flow



Multiple Common Elements Files

- One JSP file for each common area
  - header.jsp
  - navbar.jsp
  - footer.jsp
- Four-cell arrangement is coded into each content page
  - Cells use jsp:include to display the common element files
- Template files have no cell layout

Multiple Common Elements Files



Template Pages

Content Pages

Multiple Common Elements Files

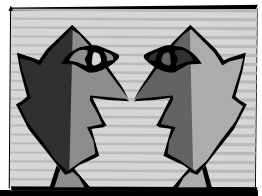
- Benefits
  - Common template areas used by all pages (same as Include Content)
    - Content pages in this case hold includes for common elements
  - No special Controller code is required
    - Can use more declarative code
- Drawback
  - Layout cells repeat on each page
    - If this design changes, all pages need to be changed
    - Can do a lot with copy and paste in the Structure window

Agenda

- Approaches for template use
- Common ADF look-and-feel options
- Templates with ADF Faces
- Conclusions and the Future

Review: ADF Faces

- Oracle-invented, open source, JavaServer Faces (JSF) tag library
  - F.k.a., ADF UIX
  - Rich UI components: tables, trees, shuttles, date and color pickers
  - AJAX-like operations using JavaScript & XML
- Supports multiple display formats
  - Web browser, wireless, telnet
  - Being used to develop Fusion Applications
- Well-supported in JDeveloper



Skins in ADF Faces

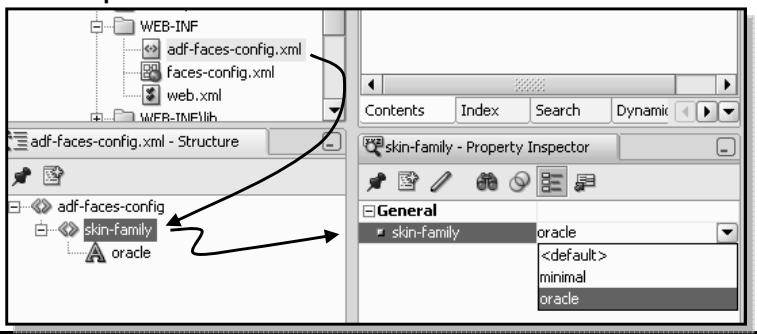
- May be needed in conjunction with templates to supply common LAF
- Skins are style sheets and a resource files for text in the component
- Use them to highly customize the appearance
- Default skin for ADF Faces is BLAF (a.k.a.,"oracle")

Oracle Browser Look And Feel

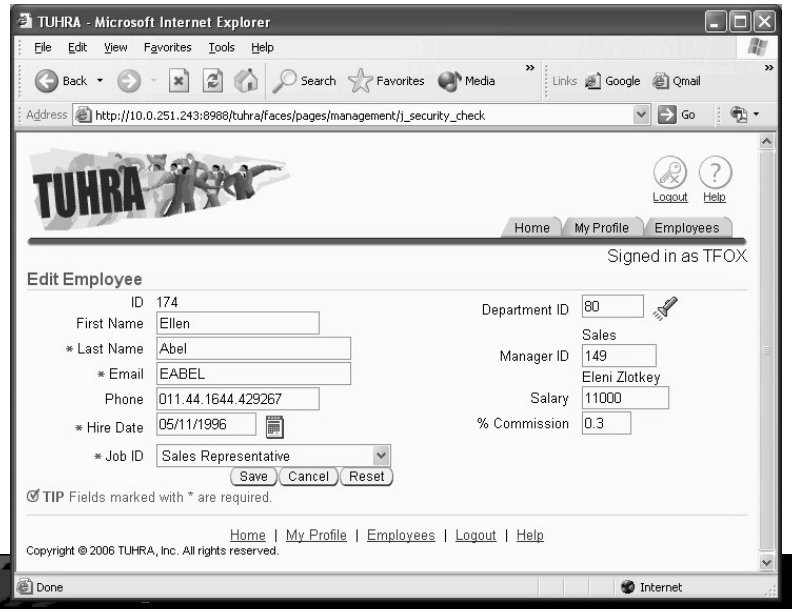
- BLAF: a highly evolved UI standard
  - 300+ pages of documentation
  - Includes page flow standards
  - www.oracle.com/technology/tech/blaf
- Used in Oracle E-Business Suite
  - EBS is coded in UIX but the same design applies to ADF Faces
- If you have EBS applications, this might be a logical choice
- The skin assignment is easy to change

Changing Skins

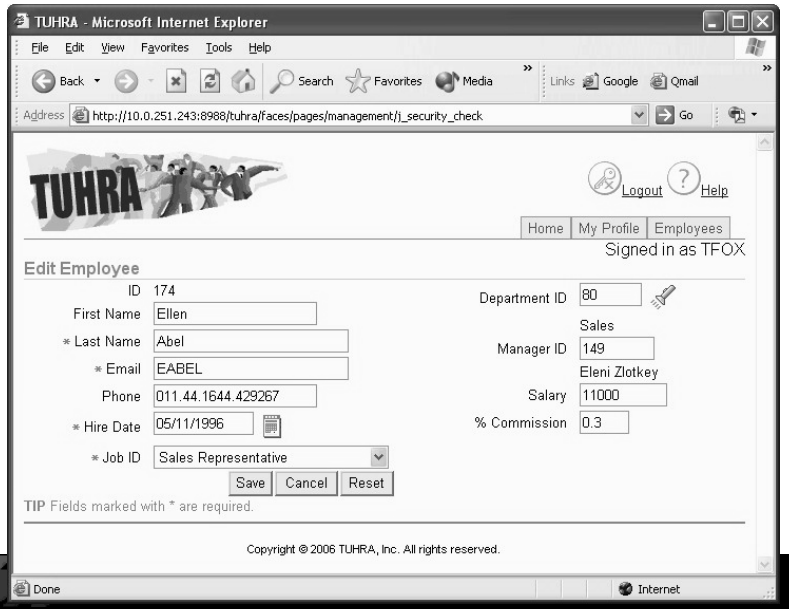
- Single property for all pages in application
  - adf-faces-config.xml - in WEB-INF directory
  - Change the property **`skin-family`**
  - Three default skins – oracle (BLAF), minimal, simple



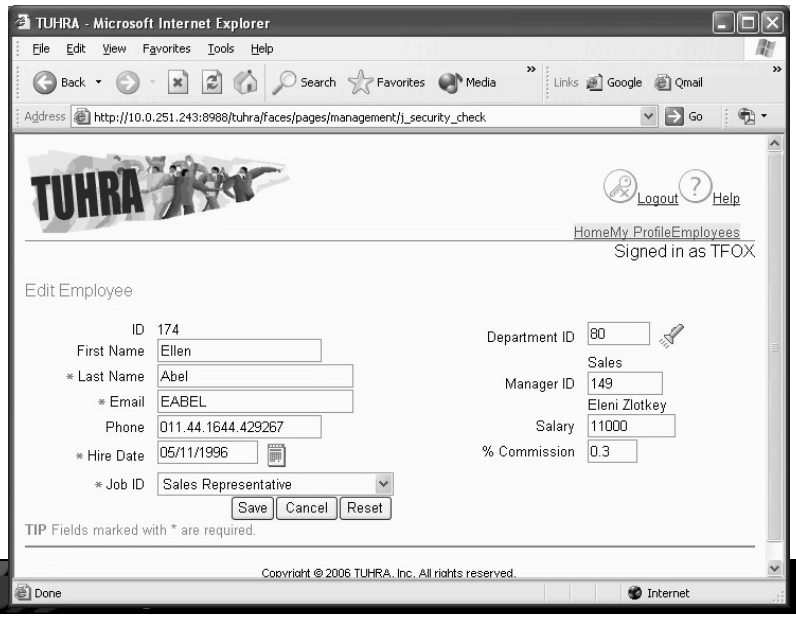
Oracle Skin



Minimal Skin



Simple Skin



"I Don't Like Those Skins"

- Fine. Then roll your own.
  - Be sure to dedicate enough time to this task
  - Read up before beginning (references coming up)
- Skins use style sheets and a resource bundle (for text inside the components)
  - Your work is mostly in the style sheets
- Start by extending the simple skin
- Declare CSS selectors to override the simple skin defaults
- Register the skin in adf-faces-skins.xml

Use These Resources

- JDeveloper help system
  - *Selectors for Skinning ADF Faces Components*
- Chapter 22 - *ADF Developer's Guide*
  - 22.3 Using Skins to Change the Look and Feel
- Sample skin with OTN LAF:
  - blogs.oracle.com/jheadstart/2006/12/22#a122
- ADF Faces skin selectors
  - Docs on the styles used for skins
  - www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/doc/skin-selectors.html

More Resources

- Olaf Heimburger - *ADF Faces 10.1.3: Setting skins per user role*
  - blogs.oracle.com/olaf/2007/04/23
- *Developing and Using ADF Faces Skins*
  - Jonas Jacobi
  - www.oracle.com/technology/products/jdev/101/howtos/adfskins/ index.html
- *Oracle WebCenter Framework Developer's Guide*
  - Defining and Applying Styles to Core Customizable Components

Agenda

- Approaches for template use
- Common ADF look-and-feel options
- Templates with ADF Faces
- Conclusions and the Future

First: What Does the Java Community Offer?

- Nothing in the Java EE standards yet
- Lots of frameworks
  - ~~None natively supported in JDeveloper~~
  - They are all supported as is any Java framework
- Some popular frameworks
  - Tiles
  - Facelets
  - Velocity

Tiles

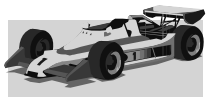
- Struts heritage
  - Still part of Struts
  - Use with the Struts controller framework
  - Standalone version on the horizon
    - "Standalone Tile" or "Tiles2"
- Good for applications that use Struts
  - If you're doing JSF, you may not be doing Struts
    - JSF has a native Controller
- struts.apache.org/struts-tiles

Facelets

- Destined to be a standard for Java EE apps
- More than templates, but good support of all template concepts
- Facelets home page discusses integration with JDeveloper
- A strong contender for new JSF apps that need Java EE support
- facelets.dev.java.net

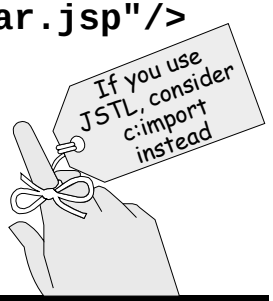
Velocity

- Apache Velocity Engine
  - Open source
  - Used for web pages and more
- Includes a scripting language
  - Conditional and iteration statements
- Enforces MVC design
  - Different programmers can work on different parts
- velocity.apache.org



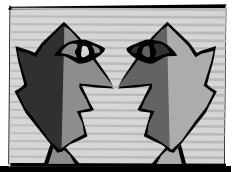
Native JSP Tag

- **`jsp:include`**
- Standard JSP tag for embedding pages inside other pages
- Example, in template.jsp
  - **`<jsp:include page="/regions/leftNavBar.jsp"/>`**
- Tags from the leftNavBar.jsp page are rendered when this tag is reached in template.jsp



Now What About ADF Faces?

- **`af:region`**
  - An ADF Faces component
- Advantages over **`jsp:include`**
  - It's a JSF component
    - JSF backing bean support – programmatic control
  - It's an ADF Faces component
    - More properties: binding, rendered, attributeChangeListener
    - More likely to have enhancements



Using af:region

- Scenario: You want to include header.jspx inside template.jspx

1. Create header.jspx with its contents inside **`af:regionDef`** tags
2. Register header.jspx as a region component in region-metadata.xml
3. Use **`af:region`** in template.jsp to reference the component

- Code samples later

Sidebar: ADF Faces Container Components

- ADF Faces supplies a number of container components
  - Components into which you put other components
  - Each one has a specific behavior and *facets* (prebuilt locations for components)
  - Usually, **`af:panel<`*`something`*`>`**
- The following example uses **`af:panelBorder`**
  - This provides facets for top, bottom, left, right, start, end (and "inner" versions of all those)

1. header.jspx

- Used for the header part of the template
- Create a JSF JSP in a regions directory
- Remove the af:view tag and its children
- Repeat for navbar.jspx and footer.jspx

```
<!-- boilerplate tags created by New Gallery JSF JSP Wizard
   here -->

<af:regionDef var="attrs">
    <af:objectImage source="/images/nocougtop.jpg"/>
</af:regionDef>

<!-- boilerplate JSP tags created by New Gallery -->
```

2. region-metadata.xml

- This file is created when you add the first af:regionDef tag to any JSP
- Add sections for navbar and footer

```
<!-- boilerplate tags created by JDeveloper here -->
<component>
  <component-type>
     hrapp.view.region.Header
  </component-type>
  <component-class>
     oracle.adf.view.faces.component.UIXRegion
  </component-class>
  <component-extension>
    <region-jsp-ui-def>
       /regions/header.jspx
    </region-jsp-ui-def>
  </component-extension>
</component>
```


3. template.jspx

- Add af:panelBorder
- Embed af:region inside f:subview in the facet

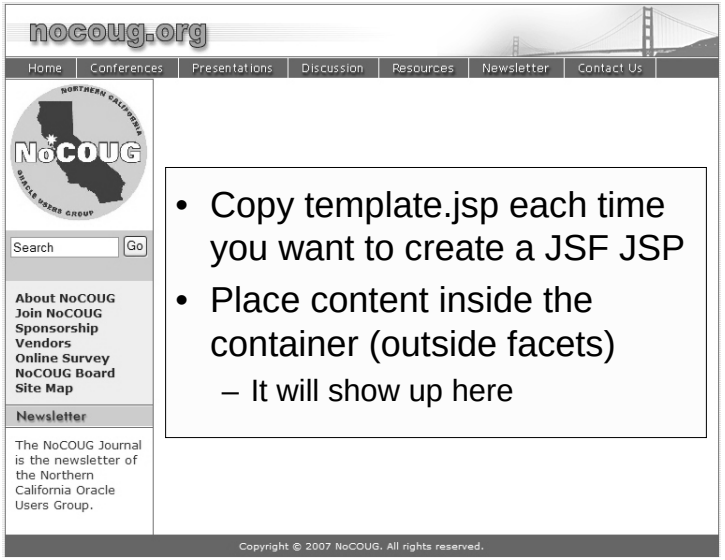
```
<!– boilerplate intro tags from JSF JSP Wizard here -->
<h:form>
  <af:panelBorder>
    <f:facet name="top">
      <f:subview id="topMargin">
        <af:region id="topMargin"
                   regionType= "hrapp.view.region.Header"/>
      </f:subview>
    </f:facet>
    <f:facet name="bottom"/>
    <f:facet name="left"/>
  </af:panelBorder>
</h:form>
<!-- more boilerplate tags -->
```

Reference the region component using the "component type" name.

Finishing Off the Template

- Add another region for the Footer region
- Add another region for the NavBar region
- The visual editor will show template.jspx with the includes taking effect
- At runtime, the regions will be included just as in the visual editor

Eureka!



- Copy template.jsp each time you want to create a JSF JSP
- Place content inside the container (outside facets)
  - It will show up here

Templates in JHeadstart

- The JHeadstart plug in (extra cost item) creates code with templates at its core
  - Velocity to generate JSF View and Controller code
  - Everything is based on a template
  - **`af:region`** to provide template reuse
- Learning the Velocity template language is helpful
  - You can get close to (even attain?) 100% generation if you do

Agenda

- Approaches for template use
- Common ADF look-and-feel options
- Templates with ADF Faces
- Conclusions and the Future

What to Do?

- Non-ADF shops using JDeveloper can plug Facelets into JDeveloper
- ADF shops use af:region
  - JHeadstart can speed up development
    - Automatically use Velocity templates currently
  - Seriously consider creating a skin
    - Lots of work, though
- For future ADF Faces work:
  - Examine JDeveloper 11*g*
    - Strong templating features
    - Preview version on OTN
  - Any work with templates now will help with JDev 11*g* later

JDev 11*g* Template Enhancements

- Creating a template
  - New gallery item for JSF template
  - Add container components, define facets, and arguments
- Using the template
  - Application's templates appear in the JSF wizard
  - The layout elements are referenced from the template
  - Like af:panelBorder except you define the facets
- Demo here:
  - www.oracle.com/technology /products/ jdev/ 11/index.html

But Janice Used a Template!

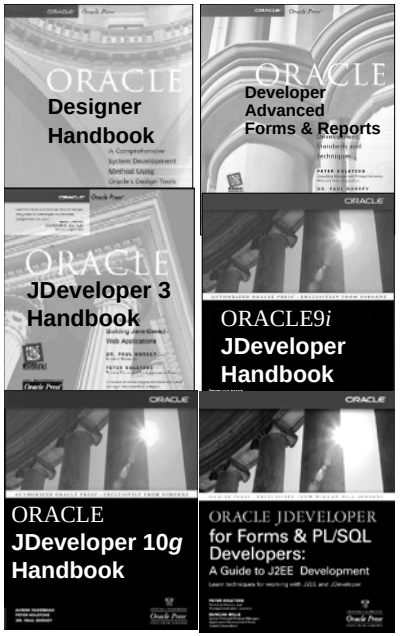
I break the mold.
JLo* wouldn't be here today
if it wasn't for me.

—Janice Dickinson (1955-)

* Not to be confused with
"jLo," the Java logging framework

Summary

- Templates help user productivity
- Templates ease maintenance
- Several approaches to template use
  - Reference as much as possible
- The Java community has many frameworks
  - E.g., Tiles, Facelets, Velocity
- ADF Faces offers af:region
- ADF Faces assists common look-and-feel with skins
- JDev 11*g* will have more template support
  - Work with templates now!



- **Please fill out the evals**
- **Books co-authored with Dr. Paul Dorsey, Avrom Roy-Faderman, & Duncan Mills**
- **Personal web site:**
  `http://ourworld.compuserve.com/homepages/Peter_Koletzke`

quovera

http://www.quovera.com

- **Founded in 1995 as Millennia Vision Corp.**
- **Profitable for 7+ years without outside funding**
- **Consultants each have 10+ years industry experience**
- **Strong High-Tech industry background**
- **200+ clients/300+ projects**
- **JDeveloper Partner**
- **More technical white papers and presentations on the web site**