Overriding Declarative Methods in ADF Model

Dr.Ahmad Taufik Jamil
BScMed, MD, MPH, MSc(IT)
Head, Center for Information Technology,
National University Hospital, Malaysia (HUKM)

Introduction

- Much of what we need can be handled **declaratively** through the various bindings and use of expression language.
- However, it is inevitable that you will need to **write code** that will interact with the bindings, either in the form of:
  - Accessing (read & set) bound data values
  - Manually executing operations and method
- The default for ADF data model is **declarative**

Overriding

- Adding **logic** before or after the method execute.
- Adding logic to a declarative method by **creating a new method and property on a managed bean** that provide access to the **associated action binding**.

Managed bean is required

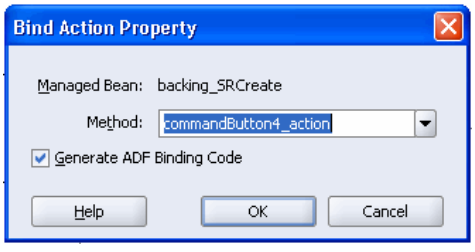
- You must have
  - a **managed bean** to hold
  - a new **method** to which the command component will be bound.
- If your page has a **backing bean** associated with it, JDeveloper adds the code needed to access the binding object to this backing bean.
- If your page does not have a backing bean, JDeveloper asks you to create one.

How to override

- Drag a method to be overidden onto the JSF page and drop it as a UI command component.
- On the JSF page, double-click on the component.

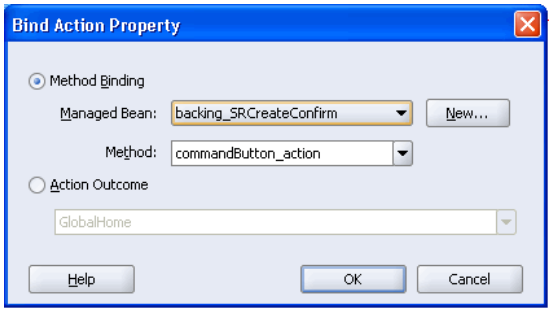
How to override

- If auto-binding has been enabled on the page, the backing bean is already selected for you



How to override

- If the page is not using auto-binding, you can select from an existing backing bean or create a new one.



How to override- caveat

- You cannot use the following procedure if the command component currently has an EL expression as its value for the Action attribute, as JDeveloper will not overwrite an EL expression.
- You must remove this value before continuing.

How to override

- After identifying the backing bean and method, click OK in the Bind Action Property dialog.
- JDeveloper opens the managed bean in the source editor.

```
1.  private BindingContainer bindings;

2.  public BindingContainer getBindings() {
3.      return this.bindings;
4.  }
5.  public void setBindings(BindingContainer bindings) {
6.      this.bindings = bindings;
7.  }
8.  public String commandButton_action1() {
9.  //Add your code here
10.     BindingContainer bindings = getBindings();
11.     OperationBinding operationBinding =
bindings.getOperationBinding("Delete");
12.     Object result = operationBinding.execute();
13.     if (!operationBinding.getErrors().isEmpty()) {
14.         return null;
15.     }
16. //Add your code here
```

Accessor method

Access the current binding container

A command button is bound to the "Delete" method

How to override

- You can now add logic either before or after the binding object is accessed.

1. FacesContext fc = FacesContext.getCurrentInstance();
2. ValueBinding expr = fc.getApplication().
   createValueBinding("#{bindings.SomeAttrBinding.inputValue}");
3. DCIteratorBinding ib = (DCIteratorBinding) expr.getValue(fc);

Example

1. FacesContext fc = FacesContext.getCurrentInstance();
2. ValueBinding expr = fc.getApplication().createValueBinding("#{bindings.patientId.inputValue}");
3. DCIteratorBinding ib = (DCIteratorBinding)expr.getValue(fc);

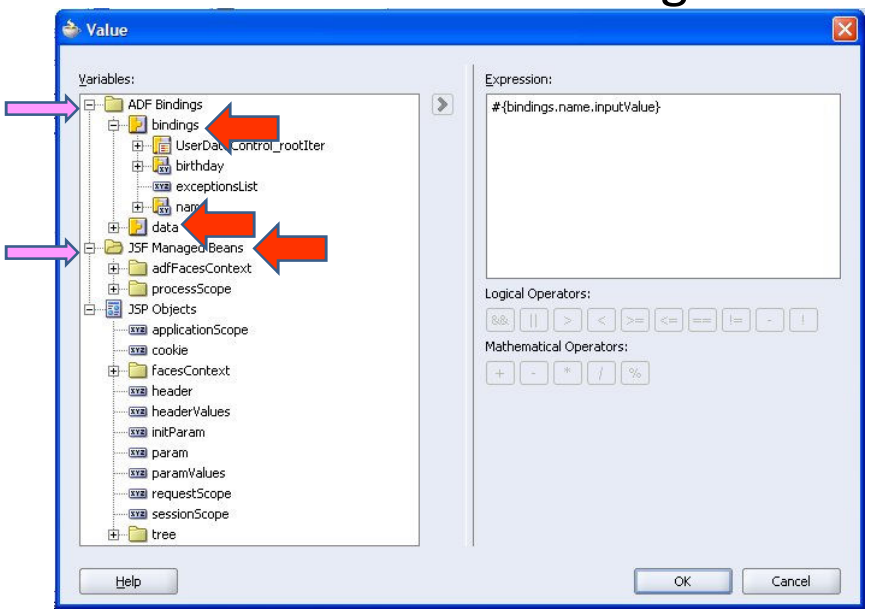
To get the result of an EL expression, using "ValueBinding" class

```
1.  public String commandButton_action1() {
2.  BindingContainer bindings = getBindings();
3.      OperationBinding operationBinding =
    bindings.getOperationBinding("Delete");
4.      Object result = operationBinding.execute();
5.      if (!operationBinding.getErrors().isEmpty()) {
6.          return null;
7.      }
8.   FacesContext fc = FacesContext.getCurrentInstance();
9.    ValueBinding expr =
    fc.getApplication().createValueBinding("#{bindings.patientId.inputValue}");
10.   DCIteratorBinding ib = (DCIteratorBinding)expr.getValue(fc);
11. return null;
12. }
```

Source of data

- Binding object
- Page definition file
- Javabeans

"Bind to Data" dialog



Contents

- ADF Bindings
  - Bindings (Most cases we are interested in data listed under this node, it contain the available data in the context of the current page)
  - Data (use to access the bound data element of any page within application)
- JSF Managed Beans
  - Contain managed bean
- JSP Objects
  - General expression scope available in JSPs, such as sessionScope, coookies etc.

Faces-config.xml

```
1.  <managed-bean>
2.    <managed-bean-name>backing_searchForEdit4</managed-bean-name>
3.    <managed-bean-class>view.backing.SearchForEdit4</managed-bean-class>
4.    <managed-bean-scope>request</managed-bean-scope>
5.    <managed-property>
6.      <property-name>bindings</property-name>
7.      <value>#{bindings}</value>
8.    </managed-property>
9.    <!--oracle-jdev-comment:managed-bean-jsp-link:1searchForEdit4.jsp-->
10.  </managed-bean>
```

Getting attribute Values Using getInputValue()

- Access value from binding object:

```
1. AttributeBinding deptBinding = (AttributeBindings) getBindings().getControlBinding("DepartmentName");
2. String departmentName = (String) deptBinding.getInputValue();
```

- **This code is simpler than the code earlier. In this code we need to cast the ControlBinding by the getControlBinding() to an AttributeBinding*

Getting attribute Values Using getInputValue() ~ another way

- Use JSF Expression Language to access a bound data value within backing bean (within a method).

1. *FacesContext ctx = FacesContext.getCurrentInstance();*
2. *Application app = ctx.getApplication();*
3. *ValueBinding bind = app.createValueBinding("#{binding.DepartmentName.inputValue}");*
4. *String departmentName = (String) bind.getValue(ctx);*

Setting attribute Values Using setInputValue()

1. *AttributeBinding deptBinding = (AttributeBindings) getBindings().getControlBinding("DepartmentName");*
2. *deptBinding.**setInputValue**("Special Projects");*

Reset attribute Value

- Reset attribute value using resetValue() method to force it to refresh its internal state.

1. *AttributeBinding deptBinding = (AttributeBindings) getBindings().getControlBinding("SepartmentName");*
2. *deptBinding.setInputValue("Special Projects");*
3. *This.getDepartmentNameField().resetValue();*

Accesing PageDef File Parameters

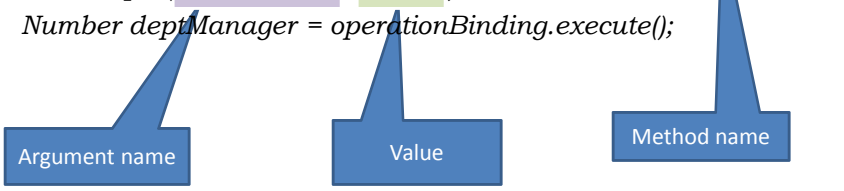
1. *DCBindingContainer dcBindings = (DCBindingContainer) getBindings();*
2. *DCParameter param = dcBindings().findParameter("deptno");*
3. *String paramValue = (String) param.getValue();*

Executing method

1. *BindingConatainer bindings = getBindings();*
2. *OperationBinding operationBinding = bindings.getOperation("First");*
3. *Object result = operationBinding.execute();*

Executing method with argument

1. *OperationBinding operationBinding = getBindings().getOperationBinding("findDepartmentManagerId");*
2. *Map params = operationBinding.getParamsMap();*
3. *Params.put("searchTerm","Sales");*
4. *Number deptManager = operationBinding.execute();*



Backing bean best practice

Backing Beans

- If a web page requires **code to process information on the page**, that code should be contained in a backing bean for the page.
- The backing bean is a one-stop-shop for the code associated with a web page and generally speaking **it is good practice**, if you require code to support a page, to have **one backing bean per page**.

- JDeveloper can create the backing bean for you automatically, and in doing **so can create accessors** for all the control items on the web page.
- When do you this, try to ensure a clear **naming convention** for the UI items and the backing bean accessors and use packages to logically partition the code.

- You should also get in the habit of **keeping the backing bean as "clean"** as possible by removing any accessor code not required for your application.
- It is also generally a best practice **not to use the backing beans for persisting information across pages (use managed beans instead)** and therefore recommended that backing beans be placed in request scope.

Managed Beans

- If you **require state information to be held for the user interface** then this should be done through managed beans.
- You can then use the managed beans to access the state rather than through the session/request directly.
- If you access directly you lose the layer of abstraction that is hiding the implementation details of JSF (which could change).

References

- Peter Koletzke , Duncan Mills ,Oracle JDeveloper 10g for Forms & PL/SQL Developers: A Guide to Web Development with Oracle ADF, Osborne Oracle Press
- An Introduction to ADF Best Practices, An Oracle White Paper, July 2006