An Introduction To Struts: Form Beans

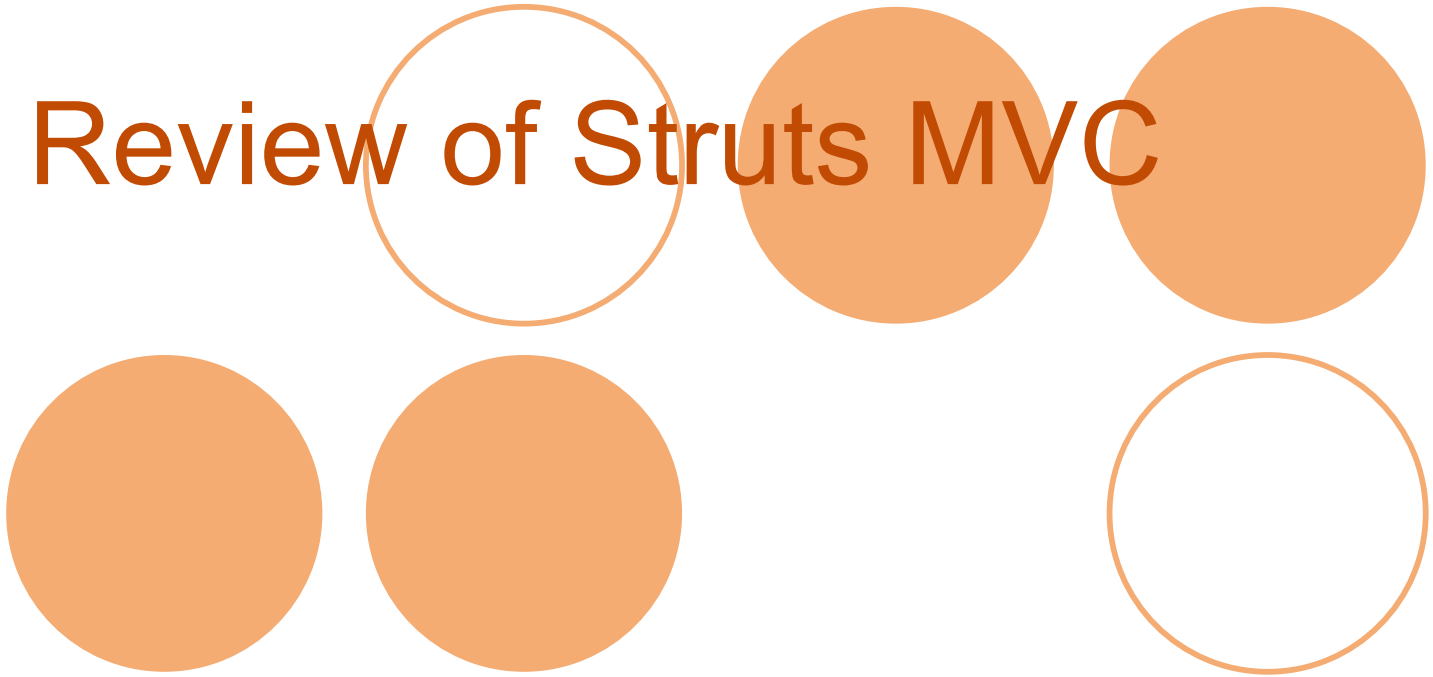
Presented by

DevelopIntelligence LLC

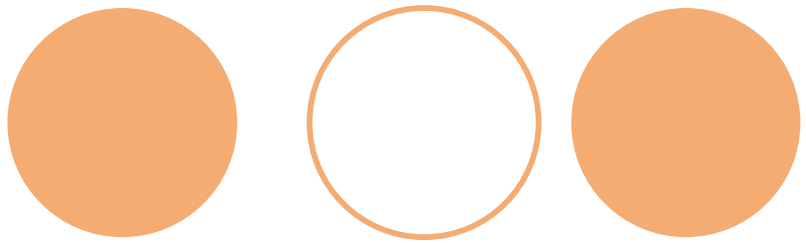
Presentation Topics

- Review of Struts MVC
- Introducing Form Beans
- Using Form Beans
- Other Variations of Form Beans
- Design Considerations

Review of Struts MVC

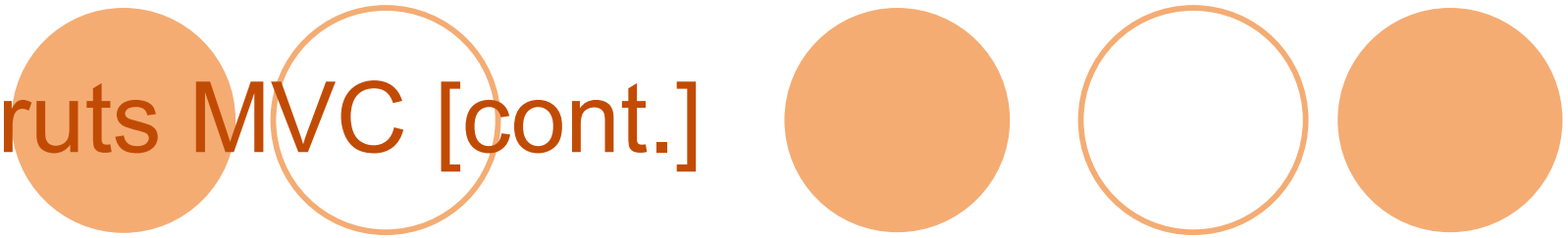


Struts MVC



- Struts implementation of MVC is loosely implemented using classic MVC + OO Design Patterns
  - Model - accessed through Action
  - View - represented as JSPs
  - Controller - implemented using Front Controller and Command Pattern
    - Front Controller - `ActionServlet`
    - Controller - `RequestProcessor`
    - Command Pattern - `Action`

Struts MVC [cont.]

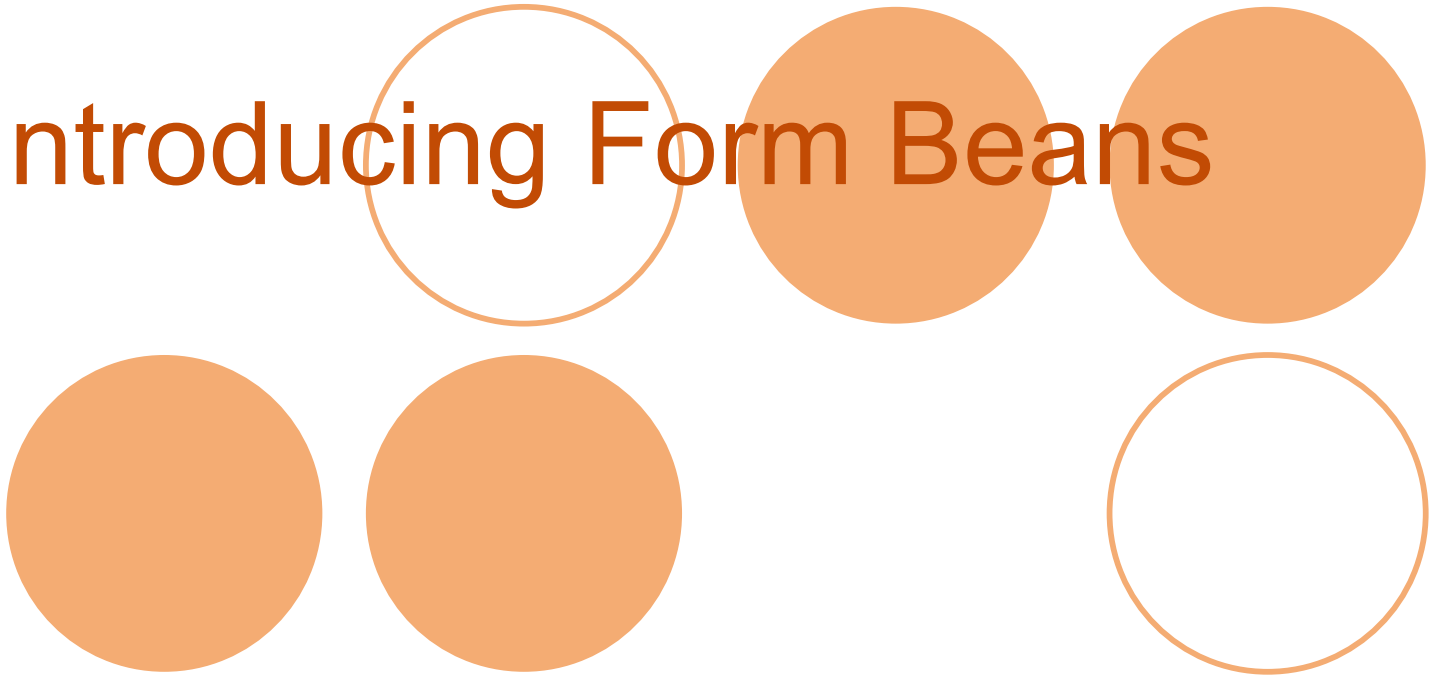


- **View-Controller Associations** done through JSP using things like:
  - `<html:link>`
  - `<html:form>`
- **Controller-Model Associations** done through action mappings, like:
  - `Action`
    - `<action path="/Search" type="com.myco.actions.SearchAction" />`
  - `ForwardAction`
    - `<action path="/Search" type="org.apache.struts.actions.ForwardAction" parameter="/Results.jsp" />`
    - `<action path="/Search" forward="/Results.jsp" />`
  - `IncludeAction`
    - `<action path="/Search" type="org.apache.struts.actions.IncludeAction" parameter="/header.jsp" />`
    - `<action path="/Search" include="/header.jsp" />`

Struts MVC Challenges

- Separation of concern makes data passing a challenge
  - Need to limit dependencies across Model, View, and Controller
  - But view data needs to be accessible by Model and vice-versa
- Framework needs generalized mechanism for passing data
  - Could use `HttpServletRequest`, but then loose strong-typing
  - Could use `HttpSession`, but then loose cohesion

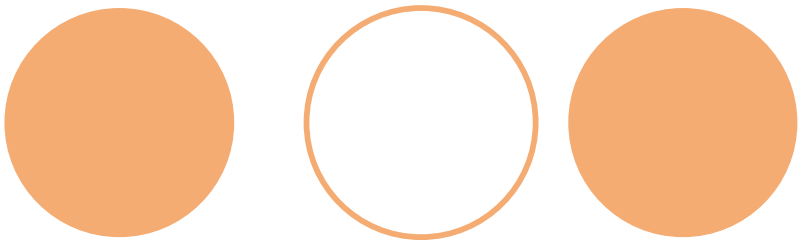
Introducing Form Beans



Introducing Form Beans

- Form Beans address MVC challenges by:
  - Creating “glue” passed between Model, View, and Controller
  - Without breaking OO concepts
- Form Beans are:
  - Specialized implementations of `ActionForm`
  - Managed by struts framework
    - Lifecycle
    - “Glue” associations
  - JavaBeans

ActionForm



- Considered JavaBeans
- Is attached to one or more actions via an `ActionMapping`
- Passed to Action's `execute` method
- Has methods for:
  - Property validation - `validate`
  - Property re-initialization - `reset`
- Have a well defined lifecycle invoked by the `RequestProcessor`
  1. Retrieve or Create Form Bean associated with `Action`
  2. "Store" Form Bean in appropriate scope (request or session)
  3. Reset the properties of the Form Bean
  4. Populate the properties of the Form Bean
  5. Validate the properties of the Form Bean
  6. Pass Form Bean to `Action`

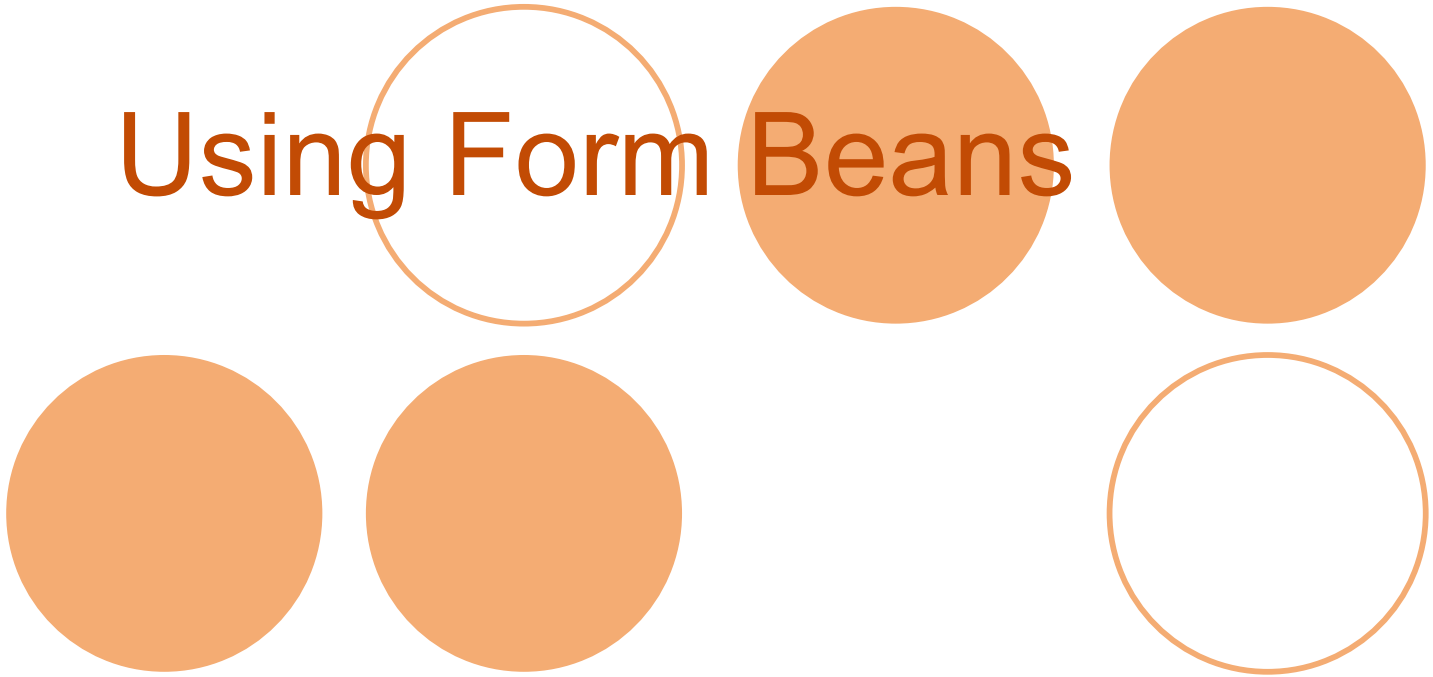
Form Beans "is a" ActionForm

- All Form Beans should be specialization of `ActionForm`
- Should override two methods:
  - `public ActionErrors validate(ActionMapping mapping, HttpServletRequest request)`
    - Used to validate properties after they have been populated; Called before Form Bean is handed to `Action`
    - Returns a collection of `ActionError` as `ActionErrors`
      - Empty collection or null signifies no errors, processing continues
      - Non-empty collection signifies errors, flow is sent back to input form
  - `public void reset(ActionMapping mapping, HttpServletRequest request)`
    - Used to reset bean to some default state
    - May cause issues if dealing with a bean in session scope

Form Beans as Java Beans

- Should support JavaBeans specification
  - public `Serializable` class
  - public no-argument constructor
  - public get/set method pairs identifying properties
- Properties can be:
  - Primitive types - struts will do conversion automatically
  - Reference types -
    - `String`, Wrappers, etc.
    - Other Form Beans
    - Indexed properties using arrays or lists
- Properties are populated using reflection

Using Form Beans



Form Beans Development

- Pick a paradigm
  - View-populate-process-View (VPPV)
    - Used in generation of pre-processing view (pre-population)
    - Used in generation of post-processing view
  - Populate-process-view (PPV)
    - Used in generation of post-processing view
  - Populate-process (PP)
    - Populated and processed as result of View action
    - Not used in generation of view
- Paradigm choice may govern Form Bean design and development strategy
  - VPPV and PPV - view driven model
  - PP - model driven model

Form Bean Development Steps

1. Create HTML UIs
   1. One containing form; one rendering results
   2. Use Struts HTML tag library and JSTL
   3. Make sure to define action for form
2. Create Form Bean to support form
   1. Create class that extends `ActionForm`
   2. Included get/set methods for each form field you wish to have populated
   3. Override `validate` and `reset` where necessary
3. Update `MessageResource.properties` with error keys and messages associated with `ActionErrors` from validate
4. Create `Action` class
5. Edit `struts-config.xml`
   1. Add `<form-beans><form-bean …/></form-beans>` entry
   2. Add `<action-mapping><action …/></action-mappings>` entry

Step 1: Create HTML UIs (index.jsp)

Initial View

- Struts Tag Lib (line 2)
- HTML Form (lines 16 - 38)
- Associates action (Compute) with form (line 16)
- Uses Form Bean (ellipsoid) to pre-populate Form (lines 21, 25, 29)

```
1   <!-- index.jsp -->
2   <%@ taglib prefix="html" uri="http://struts.apache.org/tags-html" %>
3
4   <html>
5     <head> <title>Elliptical Math</title> </head>
6
7     <body bgcolor="#F8F8E8" >
8
9       <center><h2>Elliptical Math</h2></center>
10      <p>
11        This application will classify an ellipsoid as defined by its three
12        &quot;semi-axis&quot; lengths A, B and C, and will calculate the volume
13        of the ellipsoid.
14      </p>
15
16      <html:form action="/Compute">
17        <center>
18          <table border="0" cellpadding="8" >
19            <tr>
20              <td>Semi-axis A length:</td>
21              <td><html:text property="a" value="${ellipsoid.a}" /></td>
22            </tr>
23            <tr>
24              <td>Semi-axis B length:</td>
25              <td><html:text property="b" value="${ellipsoid.b}" /></td>
26            </tr>
27            <tr>
28              <td>Semi-axis C length:</td>
29              <td><html:text property="c" value="${ellipsoid.c}" /></td>
30            </tr>
31            <tr>
32              <td colspan="2" align="center">
33                <html:submit title="Compute" />
34              </td>
35            </tr>
36          </table>
37        </center>
38      </html:form>
39    </body>
40  </html>
41
```

Step 1: Create HTML UIs (results.jsp)

Results View

- Invoked based on "success" forward action mapping
- Uses Form Bean results (Ellipsoid) to populate table (lines 13, 17, 21, 25, 29, 33)

```
1  <!-- results.jsp -->
2
3  <html>
4    <head> <title>Elliptical Math</title> </head>
5
6    <body bgcolor="#F8F8E8" >
7
8      <center><h2>Elliptical Math</h2></center>
9      <center>
10       <table border="0" cellpadding="8" width="90%" >
11         <tr>
12           <td>Semi-axis A length:</td>
13           <td>${ellipsoid.a}</td>
14         </tr>
15         <tr>
16           <td>Semi-axis B length:</td>
17           <td>${ellipsoid.b}</td>
18         </tr>
19         <tr>
20           <td>Semi-axis C length:</td>
21           <td>${ellipsoid.c}</td>
22         </tr>
23         <tr>
24           <td>Volume:</td>
25           <td>${ellipsoid.volume}</td>
26         </tr>
27         <tr>
28           <td>Class of ellipse:</td>
29           <td>${ellipsoid.type}</td>
30         </tr>
31         <tr>
32           <td valign="top">Definition:</td>
33           <td>${ellipsoid.definition}</td>
34         </tr>
35       </table>
36     </center>
37     <center><p><b><a href="index.jsp">BACK</a></b></p></center>
38   </body>
39 </html>
40
```

Step 2: Create Form Bean [pt. 1]

Ellipsoid Class

- Specialization of ActionForm (line 5)
- Contains public no-argument constructor
- Contains Java Bean properties for a,b,c,type

```
1  package cc.math;
2
3  import org.apache.struts.action.ActionForm;
4
5  public class Ellipsoid extends ActionForm {
6
7    private double a = 1;
8    private double b = 1;
9    private double c = 1;
10   private String type = "";
11
12   public Ellipsoid() {}
13
14   public double getA() {
15     return a;
16   }
17
18   public void setA(double a) {
19     this.a = a;
20   }
21
22   public double getB() {
23     return b;
24   }
25
26   public void setB(double b) {
27     this.b = b;
28   }
29
30   public double getC() {
31     return c;
32   }
33
34   public void setC(double c) {
35     this.c = c;
36   }
```

Step 2: Create Form Bean [pt. 2]

Ellipsoid Class

- Contains Java Bean properties for type
- Notice no implementation of `reset` or `validate`; Ellipsoid uses inherited definitions
- Since no error validation is being performed, don't need to do Step 3

```
38    public void setType(String type) {
39      this.type = type;
40    }
41
42    public String getType() {
43      return type;
44    }
45
46    public double getVolume() {
47      return 4 * Math.PI * a * b * c / 3;
48    }
49
50    public String getDefinition() {
51      String type = getType();
52
53      if (type.equals("Sphere"))
54        return "Where A = B = C, offering perfect rotational symmetry in "
55               + "all three dimensions.";
56
57      if (type.equals("Triaxial ellipsoid"))
58        return "Where none of A, B and C are equal;  the most general " +
59               "classification of an ellipsoid.";
60
61      if (type.equals("Oblate spheroid"))
62        return "A \"squashed\" ellipsoid with a single axis of " +
63               "symmetric rotation that is shorter than the other two " +
64               "axes (which are equal).";
65
66      if (type.equals("Prolate spheroid"))
67        return "An \"elongated\" ellipsoid with a single axis of " +
68               "symmetric rotation that is longer than the other two " +
69               "axes (which are equal).";
70
71      return "ERROR -- UNKNOWN TYPE.";
72    }
73  }
74
```

Step 4: Create Action Class [pt. 1]

```
1  package cc.math;
2
3  import org.apache.struts.action.Action;
4  import org.apache.struts.action.ActionForm;
5  import org.apache.struts.action.ActionForward;
6  import org.apache.struts.action.ActionMapping;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 public class Compute extends Action {
11
12   public ActionForward execute(ActionMapping mapping, ActionForm form,
13                                HttpServletRequest request, HttpServletResponse response) {
14
15     String forwardResult = "error";
16
17     if (!(form instanceof Ellipsoid)) {
18       return mapping.findForward(forwardResult);
19     }
20
21     Ellipsoid ellipsoid = (Ellipsoid) form;
22     String type = determineType(ellipsoid);
23     ellipsoid.setType(type);
24
25     if (ellipsoid.getVolume() >= 0) {
26       forwardResult = "success";
27     }
28
29     return mapping.findForward(forwardResult);
30   }
```

Compute Class

- Specialization of Action (line 10)
- Overridden implementation of excute (lines 12-30)
- Accesses Form Bean (lines 21-27)
- Returns ActionForward (line 29)

Step 4: Create Action Class [pt. 2]

```
31
32    private String determineType(Ellipsoid ellipsoid) {
33      String returnValue = "N/A";
34      double a = ellipsoid.getA();
35      double b = ellipsoid.getB();
36      double c = ellipsoid.getC();
37
38      if (a == b && a == c) {
39        returnValue = "Sphere";
40      } else if (a != b && a != c && b != c) {
41        returnValue = "Triaxial ellipsoid";
42      } else {
43
44        double unique;
45        double paired;
46        if (a != b) {
47          if (a == c) {
48            unique = b;
49            paired = a;
50          } else {
51            unique = a;
52            paired = b;
53          }
54        } else {
55          unique = c;
56          paired = a;
57        }
58
59        returnValue = (unique < paired) ? "Oblate spheroid" : "Prolate spheroid";
60      }
61
62      return returnValue;
63    }
64  }
65
```

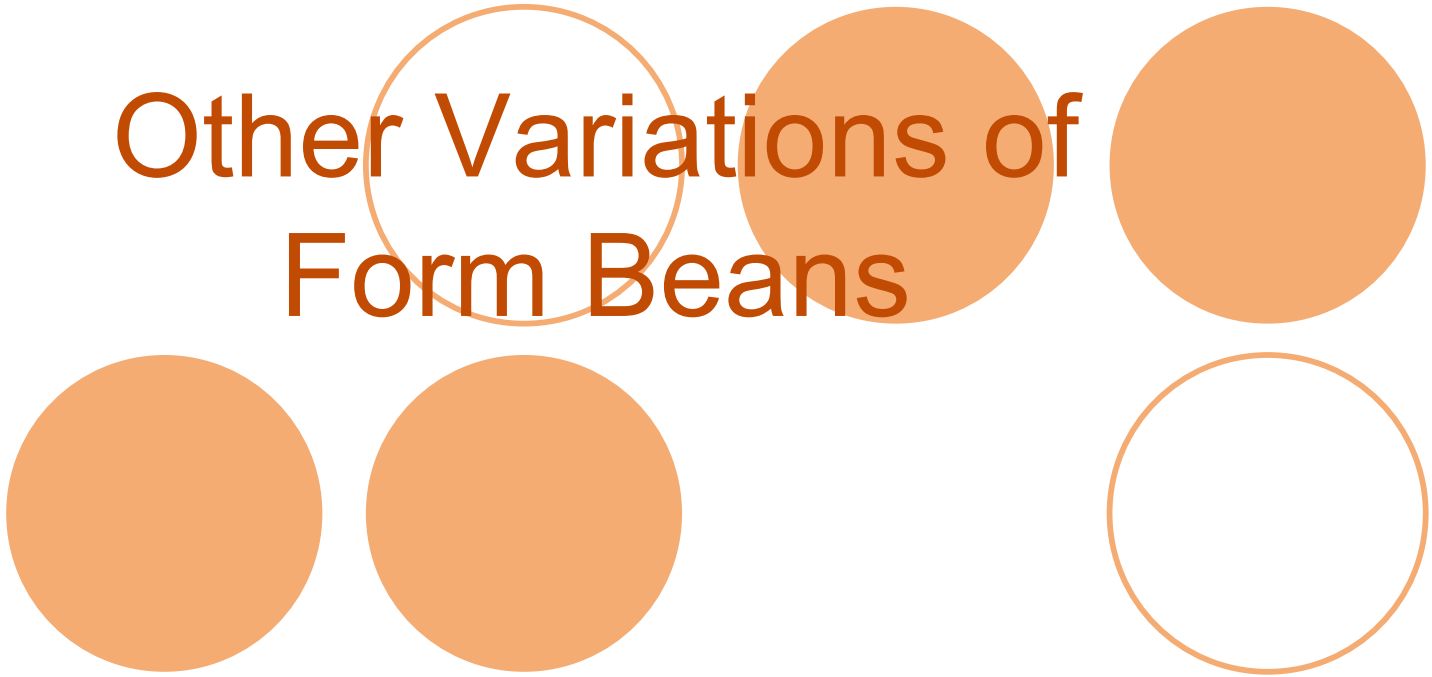
Step 5: Edit struts-config.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <!DOCTYPE struts-config PUBLIC
4     "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
5     "http://struts.apache.org/dtds/struts-config_1_3.dtd">
6
7   <struts-config>
8     <form-beans>
9       <form-bean name="ellipsoid" type="cc.math.Ellipsoid" />
10    </form-beans>
11
12    <action-mappings>
13
14      <action path="/Compute" type="cc.math.Compute"
15              name="ellipsoid" input="/index.jsp"
16              scope="request" >
17
18        <forward name="success" path="/results.jsp" />
19        <forward name="error" path="/index.jsp" />
20
21      </action>
22
23    </action-mappings>
24  </struts-config>
```

struts-config.xml

- declare ellipsoid form bean (line 9)
- declare Compute action mapping (line 14-21)
- associate form bean with action (line 15)
- define success and error action forwards (lines 18,19)

Other Variations of Form Beans



Motivations for Other Varieties

- Most web applications are user input driven
  - Many different forms, collecting many variations of data
  - UIs change quite frequently during development of web application
- `ActionForm` strategy may not be best
  - End up creating either a 1-1 or 1-n strategy in terms of UI Form and Form Bean
  - Minor Form changes can cause ripple effect in Form Bean
- Might want to consider more dynamic association with UI and Form Bean
  - `DynaActionForm`
  - `LazyDynaBean`

DynaActionForm Motivations

- Addresses HTML Form - Java class dependencies
  - Introduced in Struts 1.1
  - Subclass of `ActionForm`
- Makes Java development "easier"
  - Removes need to create concrete Java Bean class
  - No need to create repetitive get/set property code
- Makes web development "easier"
  - Bean properties, including name, types, and defaults, are declared in `struts-config.xml`
  - If form changes, just change struts-config

Action Implications [Model]

- `DynaActionForm` is the `ActionForm` instance passed to `Action`
  - `DynaActionForm` stores properties in a `Map`
    - Loose get/set property call mechanism
    - get/set property based on key access into `Map`
    - Have to cast results of get call
    - May be limited to Object-based properties
  - Loose some level of OO
    - Encapsulation - always have get/set properties
    - Cohesion - property state must be calculated outside bean

JSP Implications [View]

- `DynaActionForm` is the `ActionForm` instance available in the view
  - Since `DynaActionForm` stores properties in a map
    - Loose get property call mechanism
    - Have to access properties using different notation, like:
      - Struts bean tag

```
<%@ taglib prefix="bean"
uri="http://struts.apache.org/tags-bean" %>
<bean:write name=”ellipsoid" property="value(a)"/>
```

      - .map field in EL

```
${ellipsoid.map.a}
```

Controller Implications

- `DynaActionForm` is the `ActionForm` instance used by controller
  - Loose reset capabilities
    - Specify in `struts-config.xml`
    - Attribute of `<form-property>`
  - Loose validate capabilities - validation must be done elsewhere
    - In Action; yuck
    - On client with JavaScript
    - Using Validator plugin

Using DynaActionForm in Example

- Simplifies steps to create application
  - Go from 4 steps down to 3 steps
  - Remove Step 2, change order
- Process
  - Step 1: Create HTML Uis (index.jsp and results.jsp) - slight modification needed
  - Step 2: Edit struts-config.xml
  - Step 3: Create Action Class (ComputeDyna.java)

Step 1: Modify HTML UIs (index.jsp)

```
<html:form action="/Compute">
  <center>
    <table border="0" cellpadding="8" >
      <tr>
        <td>Semi-axis A length:</td>
        <td><html:text property="a" value="${ellipsoid.a}" /></td>
      </tr>
      <tr>
        <td>Semi-axis B length:</td>
        <td><html:text property="b" value="${ellipsoid.b}" /></td>
      </tr>
      <tr>
        <td>Semi-axis C length:</td>
        <td><html:text property="c" value="${ellipsoid.c}" /></td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <html:submit title="Compute" />
        </td>
      </tr>
    </table>
  </center>
</html:form>
```

```
<html:form action="/Compute">
  <center>
    <table border="0" cellpadding="8" >
      <tr>
        <td>Semi-axis A length:</td>
        <td><html:text property="a" value="${ellipsoid.map.a}" /></td>
      </tr>
      <tr>
        <td>Semi-axis B length:</td>
        <td><html:text property="b" value="${ellipsoid.map.b}" /></td>
      </tr>
      <tr>
        <td>Semi-axis C length:</td>
        <td><html:text property="c" value="${ellipsoid.map.c}" /></td>
      </tr>
      <tr>
        <td colspan="2" align="center">
          <html:submit title="Compute" />
        </td>
      </tr>
    </table>
  </center>
</html:form>
```

Step 1: Modify HTML UIs (results.jsp)

```
<table border="0" cellpadding="8" width="90%" >
  <tr>
    <td>Semi-axis A length:</td>
    <td>${ellipsoid.a}</td>
  </tr>
  <tr>
    <td>Semi-axis B length:</td>
    <td>${ellipsoid.b}</td>
  </tr>
  <tr>
    <td>Semi-axis C length:</td>
    <td>${ellipsoid.c}</td>
  </tr>
  <tr>
    <td>Volume:</td>
    <td>${ellipsoid.volume}</td>
  </tr>
  <tr>
    <td>Class of ellipse:</td>
    <td>${ellipsoid.type}</td>
  </tr>
  <tr>
    <td valign="top">Definition:</td>
    <td>${ellipsoid.definition}</td>
  </tr>
</table>
```

```
<table border="0" cellpadding="8" width="90%" >
  <tr>
    <td>Semi-axis A length:</td>
    <td>${ellipsoid.map.a}</td>
  </tr>
  <tr>
    <td>Semi-axis B length:</td>
    <td>${ellipsoid.map.b}</td>
  </tr>
  <tr>
    <td>Semi-axis C length:</td>
    <td>${ellipsoid.map.c}</td>
  </tr>
  <tr>
    <td>Volume:</td>
    <td>${ellipsoid.map.volume}</td>
  </tr>
  <tr>
    <td>Class of ellipse:</td>
    <td>${ellipsoid.map.type}</td>
  </tr>
  <tr>
    <td valign="top">Definition:</td>
    <td>${ellipsoid.map.definition}</td>
  </tr>
</table>
```

Step 2: Edit struts-config.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!DOCTYPE struts-config PUBLIC
4    "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
5    "http://struts.apache.org/dtds/struts-config_1_3.dtd">
6
7  <struts-config>
8    <form-beans>
9      <form-bean name="ellipsoid" type="org.apache.struts.action.DynaActionForm" >
10       <form-property name="a" type="double" initial="0"/>
11       <form-property name="b" type="double" initial="0"/>
12       <form-property name="c" type="double" initial="0"/>
13       <form-property name="volume" type="double" initial="0"/>
14       <form-property name="type" type="java.lang.String" />
15       <form-property name="definition" type="java.lang.String" />
16     </form-bean>
17   </form-beans>
18
19   <action-mappings>
20
21     <action path="/Compute" type="cc.math.ComputeDyna"
22             name="ellipsoid" input="/index.jsp"
23             scope="request" >
24
25       <forward name="success" path="/results.jsp" />
26       <forward name="error" path="/index.jsp" />
27
28     </action>
29
30   </action-mappings>
31 </struts-config>
```

struts-config.xml

- Declare ellipsoid form bean as DynaActionForm (line 9)
- Declare ellipsoid properties (lines 10-15)
- The rest is the same

Step 3: Create Action Class [pt. 1]

```
1  package cc.math;
2
3  import org.apache.struts.action.*;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6
7  public class ComputeDyna extends Action {
8
9    public ActionForward execute(ActionMapping mapping, ActionForm form,
10                                 HttpServletRequest request, HttpServletResponse response) {
11     String forwardResult = "error";
12
13     if (!(form instanceof DynaActionForm)) {
14       return mapping.findForward(forwardResult);
15     }
16
17     DynaActionForm ellipsoid = (DynaActionForm) form;
18     //modify the "properties" of the ellipsoid
19     double volume = modifyVolume(ellipsoid);
20     modifyType(ellipsoid);
21     modifyDefinition(ellipsoid);
22
23     if (volume >= 0) {
24       forwardResult = "success";
25     }
26
27     return mapping.findForward(forwardResult);
28   }
29
```

ComputeDyna Class

- Specialization of Action (line 7)
- Overridden implementation of excute (lines 9-28)
- Accesses Form Bean As DynaActionForm (lines 17-21)
- Returns ActionForward (line 27)

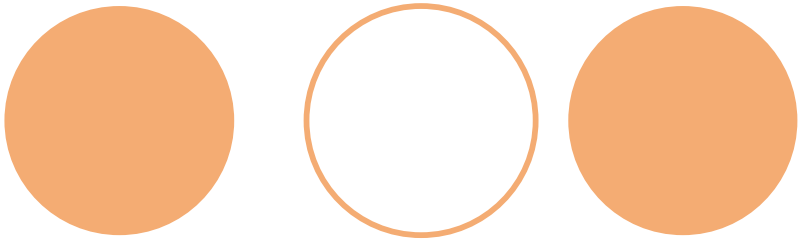
Step 3: Create Action Class [pt. 2]

```
30    private double modifyVolume(DynaActionForm ellipsoid) {
31      double a = (Double) ellipsoid.get("a");
32      double b = (Double) ellipsoid.get("b");
33      double c = (Double) ellipsoid.get("c");
34      double volume = 4 * Math.PI * a * b * c / 3;
35
36      ellipsoid.set("volume", (Double) volume);
37
38      return volume;
39    }
40
41    private String modifyType(DynaActionForm ellipsoid) {...}
76
77    private String modifyDefinition(DynaActionForm ellipsoid) {...}
104
105 }
106
```

ComputeDyna Class

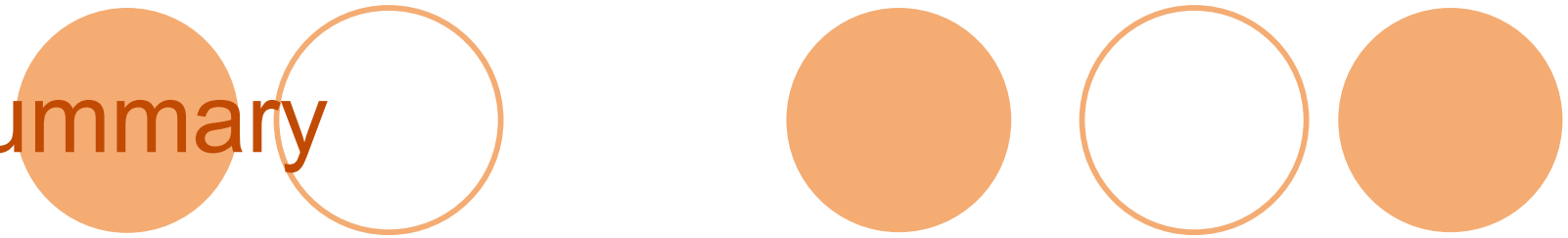
- Accesses DynaActionForm properties (lines 31-34)
- Modifies DynaActionForm property (line 36)

LazyDynaBean



- DynaActionForm “simplifies” development effort
  - Creating Bean replaced with <form-property>
  - Removes HTML-Java compile-time dependency
  - Arguable if saves on development time
- LazyDynaBean super simplifies things
  - Introduced as part of Struts 1.2.4
  - Quick and dirty
  - Don’t write any Java bean code
    - Declare it as a form bean in struts-config
    - No need to specify <form-property>
  - Has implications in View and Model
    - Similar to DynaActionForm in View
    - Loose property types in Model, need to do conversions

Summary



- Form Beans act as glue between components of MVC
- Form Beans are like a data transfer object or value object
- Three ways to create form beans
  - ActionForm subclass
  - DynaActionForm
  - LazyDynaBean

About DevelopIntelligence

- Founded in 2003
- Provide individual and organizational performance services in the area software development
- Represents over 35 years of combined experience enabling software development community through educational and performance services
- Represents over 50 years of combined software development experience
- Delivered training to over 25,000 developers worldwide

Contact Us

- For more information about our services, please contact us:
  - Kelby Zorgdrager
  - Kelby@DevelopIntelligence.com
  - 303-395-5340