© Copyright 2003, Object Systems Group

AntiPatterns Using Select Jakarta Projects

Bill Dudney

Object Systems Group

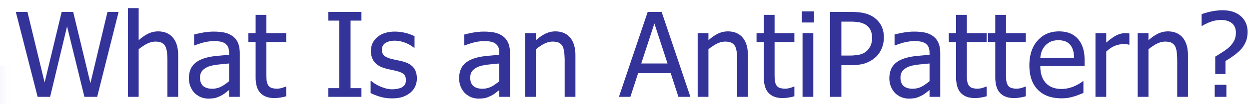
bill@dudney.net

© Copyright 2003, Object Systems Group

Agenda

- What is an AntiPattern?
- What is a Refactoring?
- Cactus/JUnit
- Ant
- Struts

© Copyright 2003, Object Systems Group



What Is an AntiPattern?

- Recurring Solution with negative outcome
  - *i.e.* "I've done the wrong thing lost of times, don't repeat my mistakes"
- Consists of:
  - Name
  - Catalog Items
    - Also Known As
    - Refactorings
    - Anecdotal Evidence
  - Background
  - General Form

Colorado Software Summit

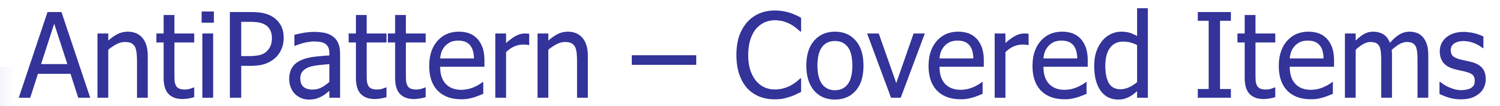
© Copyright 2003, Object Systems Group

What Is an AntiPattern?
(Continued)

- Symptoms & Consequences
- Typical Causes
- Known Exceptions
- Refactorings
- Variations
- Example
- Related Solutions

Colorado Software Summit

© Copyright 2003, Object Systems Group



AntiPattern – Covered Items

- Name
- General Form
- Symptoms & Consequences
- Refactorings
- Example

© Copyright 2003, Object Systems Group

What Is Refactoring?

- A means to improve the design of existing software without breaking (*i.e.* rewriting) every piece of code that uses the refactored code.
- Consists Of:
  - Before and After Avatar
    - Sometimes UML
    - Sometimes Code
  - Motivation
    - To get out of the AntiPatterns
  - Mechanics
  - Example

© Copyright 2003, Object Systems Group

Cactus/JUnit

- AntiPatterns
  - No Assert
  - Unreasonable Assert
  - Console-Based Testing
  - Unfocused Test Method
  - Failure to Isolate Tests
  - Failure to Isolate Subjects

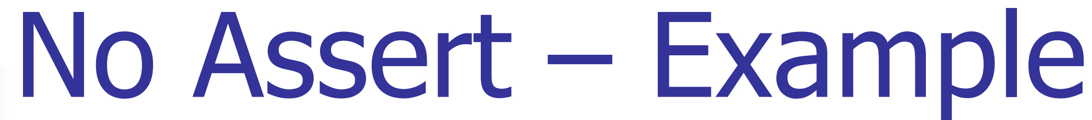
Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: No Assert

- General Form
  - No use of the JUnit `assert` method
  - Methods being called but return values ignored
  - No Check on the state changes
- Symptoms & Consequences
  - Lots of bugs in testing
  - Failure to meet the 'intent of the API'
  - Lack of confidence in the code base
- Refactorings
  - Assert the Intent

© Copyright 2003, Object Systems Group



No Assert – Example

```
public void testGetItems() throws Exception {
    InitialContext ic = new InitialContext();
    ShoppingCartLocalHome sHome =
    (ShoppingCartLocalHome)ic.lookup(JNDI_NAME);
    ShoppingCartLocal cart = sHome.create();
    cart.getItems();
}
```

© Copyright 2003, Object Systems Group

AntiPattern: Unreasonable Assert

- General Form
  - Lots of asserts
  - Assertions about really unlikely things
  - No or few asserts about the API
- Symptoms & Consequences
  - Lots of bugs from the testing team
- Refactorings
  - Assert the Intent

© Copyright 2003, Object Systems Group

Unreasonable Assert – Example

```
public void testGetItems() throws Exception {
    InitialContext ic = new InitialContext();
    assertNotNull(ic);
    ShoppingCartLocalHome sHome =
    (ShoppingCartLocalHome)ic.lookup(JNDI_NAME);
    assertNotNull(sHome);
    ShoppingCartLocal cart = sHome.create();
    assertNotNull(cart);
    assertNotNull(cart.getItems());
}
```

© Copyright 2003, Object Systems Group

Refactoring: Assert the Intent

- Before

```
public void testEquals() throws Exception {
   one.equals(oneA);
   oneA.equals(one);
}
```

- After

```
public void testEquals() throws Exception {
   assertEquals(one, oneA);
   assertEquals(oneA, one);
}
```

© Copyright 2003, Object Systems Group

Assert the Intent – Mechanics

1. Start with the simplest-to-test subject
   i. Start with what is there or create a new test
2. Review the documentation for the method to identify the intent
   i. With no doc, you will have to review the implementation
   ii. Sometimes bugs in the docs will be exposed during this exercise
   iii. Any statement about changes of internal state or actions are good candidates for tests

© Copyright 2003, Object Systems Group

Assert the Intent – Mechanics

3. For each stated intent, assert the outcome or state change.
4. Repeat these steps for each method
   i. Make sure not to get stuck in 'Unreasonable Assert' during this process, *i.e.* assert the intent of the method, not everything you can think of.

© Copyright 2003, Object Systems Group

AntiPattern: Console-Based Testing

- General Form
  - Virtually no asserts
  - Lots of System.out
- Symptoms & Consequences
  - Lots of bugs from the testing team
  - Very tired eyes!
- Refactorings
  - System.out Becomes Assert

© Copyright 2003, Object Systems Group

Console Based Testing – Example

```
public void testGetCustomers() throws Exception {
  List customers = cache.getCustomers();
  // there is no check here
  // (except visually by the developer)
  // of what the count is, and its not
  // altogether apparent
  System.out.println("Count = " +
      customers.size());
  System.out.println("Customers = " +
      customers);
}
```

© Copyright 2003, Object Systems Group

Refactoring: System.out -> Assert

- Before

```
public void testOne() throws Exception {
   System.out.println("expected foo got" +
                                    one);
}
```

- After

```
public void testOne() throws Exception {
   assertEquals("foo", one);
}
```

© Copyright 2003, Object Systems Group

System.out Becomes Assert – Mechanics

1. Start with simplest existing test method
2. For each System.out.println discover the intent
   i. Just to see what is there, to make sure it's not null
   ii. Compare an expected value with the actual value
3. Change each simple review to assertNotNull
4. Change each expected value to assertEquals
5. Review the intent of the API under test to look for missed intent
6. Repeat these steps for each test method

Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: Unfocused Test Method

- General Form
  - Very long test methods
  - Not very many test methods
- Symptoms & Consequences
  - Hard to maintain tests
- Refactorings
  - Keep it Simple

Colorado Software Summit

© Copyright 2003, Object Systems Group

Unfocused Test Method – Example

```
public void testGetCustomers() throws Exception {
  List customers = cache.getCustomers();
  assertEquals(5, customers.count());
  Customer cust1 = (Customer)customers.get(1);
  Customer cust2 = (Customer)customers.get(2);
  assertEquals(cust1.getFirstName(),
               testCustomer.getFirstName());
  ...
}
```

© Copyright 2003, Object Systems Group

Refactoring: Keep It Simple

- Before

```
public void testBean() throws Exception {
    Collection lineItems = invoice.getLineItems();
    List removedItems = new ArrayList();
    assertEquals(5, lineItems.size());
    Iterator itr = lineItems.iterator();
    int count = 0;
    while(itr.hasNext()) {
        // remove the even items to test removal
        LineItem lineItem = (LineItem)itr.next();
        if((count % 2) == 0) {
            invoice.removeLineItem(lineItem);
            removedItems.add(lineItem);
        }
    }
```

Refactoring: Keep It Simple

- After

```
public void testLineItemsCount() throws Exception {
      // 5 line items are put in during setUp
      Collection lineItems = invoice.getLineItems();
      // make sure they are there
      assertEquals(5, lineItems.size());
  }
  public void testLineItemRemoval() throws Exception {
     Collection lineItems = invoice.getLineItems();
     assertEquals(5, lineItems.size());
     Iterator itr = lineItems.iterator();
     // remove the first element
     invoice.removeLineItem((LineItem)itr.next());
     // make sure there are four items left
     assertEquals(4, invoice.getLineItems().size());
  }
```

© Copyright 2003, Object Systems Group

Keep It Simple – Mechanics (1 of 2)

- For each large and complex test method, extract groups of asserts that relate to common intent of the subject API.
  - "Large" and "complex" are, of course, subjective. The question to keep in mind is: "Will this test method be likely to be abandoned as the subject's API is changed?" If the answer is yes, then the method is probably too complex.
- Place each related group of asserts into a separate test method.
  - Name these new methods after the intent that is being asserted.

© Copyright 2003, Object Systems Group

Keep It Simple – Mechanics (2 of 2)

- Further decompose the remaining complex test methods.
  - As a general rule, each JUnit test method should have one assert, each Cactus method should have only a few.
- Deploy and test.
- Repeat theses steps for each test that is trapped in the pitfall.

Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: Failure to Isolate Tests

- General Form
  - Inter-test dependencies
  - External scripts
- Symptoms & Consequences
  - Intermittent Test Failures
  - Corrupt or Unreliable Database
- Refactorings
  - Introduce Test Decorators

Colorado Software Summit

© Copyright 2003, Object Systems Group

Failure to Isolate Tests – Example

```
public void testInitialCustomer() throws Exception {
    InitialContext ic = new InitialContext();
    cHome = (CustomerHome) ic.lookup(CUSTOMER_REF);
    CustomerPO custPO = new CustomerPO("Elmer", "Fudd");
    Customer customer = cHome.create(custPO);
    assertNotNull(customer);
    assertNotNull(customer.getId());
    assertNull(customer.getAddress());
    assertTrue(customer.getInvoices().size() == 0);
    assertEquals("Elmer", customer.getFirstName());
    assertEquals("Fudd", customer.getLastName());
}
```

© Copyright 2003, Object Systems Group

Refactoring: Use setUp & tearDown

- Before

```
public void testInitialCustomer() throws Exception {
  InitialContext ic = new InitialContext();
  cHome = (CustomerHome) ic.lookup(CUSTOMER_REF);
  CustomerPO custPO = new CustomerPO("Elmer", "Fudd");
  Customer customer = cHome.create(custPO);
  assertEquals("Elmer", customer.getFirstName());
  assertEquals("Fudd", customer.getLastName());
}
```

- After

```
public void testInitialCustomer() throws Exception {
  assertEquals("Elmer", elmerFudd.getFirstName());
  assertEquals("Fudd", elmerFudd.getLastName());
}
```

© Copyright 2003, Object Systems Group

Use setUp & tearDown – Mechanics (1 of 2)

1. Review tests for creation, initialization and deletion of subject objects
   i. Look especially for commonality between the initialization or creation
   ii. If the creation differs significantly, consider creating a new test class.
2. Review the tests for order dependencies
   i. Look for instances assumed to be available
3. Review manual processes needed to make the tests runable.
   i. Typically SQL scripts

© Copyright 2003, Object Systems Group

Use setUp & tearDown – Mechanics (2 of 2)

1. Create a setUp and tearDown method pair
   i. Consider using a test decorator if the setUp or tearDown takes a long time or is complex
2. Add an instance variable to the test class for each distinct kind of subject
   i. Each kind of subject will be used to test a different part of the API
3. Move the create or initialization code from the tests to the setUp method.
4. Move the delete code to the tearDown method

Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: Failure to Isolate Subjects

- General Form
  - Integration tests instead of unit tests
  - Lots of code in each test method to set up the test
- Symptoms & Consequences
  - Frequently Broken Tests, especially when a new version of a framework is introduced
- Refactorings
  - Introduce Mock Objects

Colorado Software Summit

© Copyright 2003, Object Systems Group

Failure to Isolate Subject – Example

```
public void testInvoiceTotal() throws Exception {
  BigDecimal total = subject.getInvoiceTotal();
  BigDecimal expected = new BigDecimal(COUNT * COST *
                                        QUANTITY);
  // setting up a value that can be used to assert the intent
  // of the API documented with the method.
  expected = expected.setScale(2, BigDecimal.ROUND_HALF_UP);
  // assert the intent
  assertEquals(expected, total);
}
```

© Copyright 2003, Object Systems Group

- Before

```
public void setUp() throws Exception {
  subject = new Invoice();
  List items = new ArrayList();
  LineItem item = null;
  for(int i = 0;i < COUNT;i++) {
    item = new LineItem(subject);
    item.setUnitPrice(new BigDecimal(COST));
    item.setQuantity(new Integer(QUANTITY));
    items.add(item);
  }
  subject.setLineItems(items);
}
```

© Copyright 2003, Object Systems Group

Refactoring: Introduce Mocks

- After

```
public void setUp() throws Exception {
  subject = new Invoice();
  List items = new ArrayList();
  LineItem item = null;
  for(int i = 0;i < COUNT;i++) {
   BigDecimal total = (new BigDecimal(COST)).
        setScale(2, BigDecimal.ROUND_HALF_UP);
   BigDecimal quantity = (new BigDecimal(QUANTITY)).
        setScale(2, BigDecimal.ROUND_HALF_UP);
   total = total.multiply(quantity).
        setScale(2, BigDecimal.ROUND_HALF_UP);
   items.add(new MockLineItem(subject, total));
  }
  subject.setLineItems(items);
}
```

© Copyright 2003, Object Systems Group

Introduce Mocks – Mechanics

1. Identify each class required to implement the subject
2. For each class write a Mock implementation, or use a mock maker tool
3. For classes, write a subclass and override the required methods
4. In the test, initialize the mock with the values to expect and return
5. Initialize the subject with the mock
6. Call the method being tested
7. Assert the state changes or return values are as expected
8. Assert the state changes expected in the mock.
9. Deploy and Test

© Copyright 2003, Object Systems Group

Ant

- Copy/Paste Reuse
- Shoe Horn Many builds into One
- Building Subprojects
- No Logging From Custom Tasks

© Copyright 2003, Object Systems Group

AntiPattern: Copy/Paste Reuse

- General Form
  - Large pieces of repeated Ant XML
- Symptoms & Consequences
  - More difficult maintenance
  - Subtle bugs/differences in targets
- Refactorings
  - Introduce Antcall

Copy/Paste Reuse – Example

```
<target name=“compile.entity” ...
  <mkdir dir=“${build.dir}/entity”/>
  <javac srcdir=“${src.dir} ...
  <mkdir dir=“${build.dir}/entity/META-INF”/>
  <copy todir=“${build.dir}/entity/META-INF”/>
...
<target name=“compile.session” ...
  <mkdir dir=“${build.dir}/session”/>
  <javac srcdir=“${src.dir} ...
  <mkdir dir=“${build.dir}/session/META-INF”/>
  <copy todir=“${build.dir}/session/META-INF”/>
...
```

© Copyright 2003, Object Systems Group

Refactoring: Introduce Antcall

- Before

```
<target name="compile.entity" ...
 <mkdir dir="${build.dir}/entity"/>
 <javac srcdir="${src.dir} ...
...
<target name="compile.session" ...
  <mkdir dir="${build.dir}/session"/>
 <javac srcdir="${src.dir} ...
...
```

© Copyright 2003, Object Systems Group

Refactoring Introduce Antcall

- After

```
<target name="compile.entity" ...
  <antcall target="_compile.ejb"/>
    <param name="ejbpkg" value="entity"/>
  </antcall>
...
<target name="compile.session" ...
 <antcall target="_compile.ejb"/>
    <param name="ejbpkg" value="session"/>
  </antcall>
...
```

© Copyright 2003, Object Systems Group

Introduce Antcall – Mechanics

1. Review targets for similar structure
   1. If you know which targets are copy and pasted, start there
   2. Another great place start are the targets associated with various J2EE deployment units, as in the example
2. Start with one group of similar targets and create another target that is a copy of one of the existing targets.

© Copyright 2003, Object Systems Group

Introduce Antcall – Mechanics

3. Rename the target and remove the description attribute if there was one.
   1. This is going to be a private target, so adopt a naming convention and stick with it.
4. Replace the body of one of the similar targets with an antcall to the new private target
   1. Don’t choose the one you copied
   2. Parameterize for each value that needs to change

© Copyright 2003, Object Systems Group

Introduce Antcall – Mechanics

5. Comment out the old body for the target
   1. Don't delete anything until you are sure the new stuff is working
6. Test the reworked target
7. Repeat for each similar target and each group of similar targets.

© Copyright 2003, Object Systems Group

AntiPattern: No Build Type Distinction

- General Form
  - Hard-coded development values for deployment parameters
  - No distinction of files to include in deployment
- Symptoms & Consequences
  - Rework/Customization to move to test and acceptance environments
- Refactorings
  - Introduce properties file

© Copyright 2003, Object Systems Group

No Build Type Distinction – Example

```
<target name="deploy" ...
  <java classname="com.sun.enterprise.tools.deployment.main.Main"
        fork="yes">
    <classpath path="${j2eeri.classpath}"/>
    <sysproperty key="org.omg.CORBA.ORBInitialPort"
                 value="1050"/>
    ...
  </java>
...
</target>
```

© Copyright 2003, Object Systems Group

Refactoring: Introduce Properties File

- Before

```
<sysproperty key="org.omg.CORBA.ORBInitialPort"
                value="1050"/>
```

- After

```
<sysproperty key="org.omg.CORBA.ORBInitialPort"
                value="${j2eeri.initial.port}"/>
```

© Copyright 2003, Object Systems Group

Introduce Properties File – Mechanics

1. Create an empty file called build.properties
   1. This is where we will put the default values
2. Create an empty file called build.properties.local
   1. This is where the values will go for the particular environment
3. Identify aspects of the build that depend on the environment
   1. Look for things like absolute paths, host names, port numbers and so on.

Colorado Software Summit

© Copyright 2003, Object Systems Group

Introduce Properties File – Mechanics

4. Place a property for each item identified in step 3
    1. Be consistent in the naming of properties – that is, if you are adding several properties related to EJBs, use a prefix like 'ejb'
5. Put two properties file imports into the build file, one for each properties file created
    1. Import the build.properties.local file first
6. Replace all the hard-coded values with references to the properties
7. Test

Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: No Logging from Custom Tasks

- General Form
  - Custom Ant Task without log messages
- Symptoms & Consequences
  - Hard to use task
  - Hard to determine why it does not work
- Refactorings
  - Introduce Logging

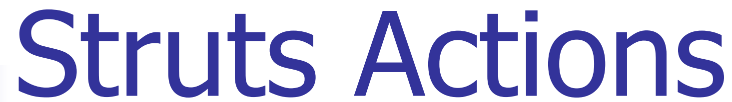
Colorado Software Summit

© Copyright 2003, Object Systems Group

Refactoring: Add Logging – Mechanics

- Look for places where tasks could fail
  - The first place to look is any catch block. There should, at the very least, be a debug log for every exception caught.
- Log a message for each point of failure, informing the user what went wrong and how to fix it.
  - These are good points to include in the docs for your task too
- Test the changes

© Copyright 2003, Object Systems Group



Struts Actions

- Business-Tier Code in Actions
- Copy/Past code in Actions
- Accessing ActionForms in the Session

© Copyright 2003, Object Systems Group

AntiPattern: Business-Tier Code in Actions

- General Form
  - Direct Access to EJBs
  - Business Logic
- Symptoms & Consequences
  - Poor Encapsulation
  - Painful Maintenance
- Refactorings
  - Move Business logic to BusinessDelegate

Colorado Software Summit

© Copyright 2003, Object Systems Group

Business-Tier Code in Actions – Example Business Logic

```
public ActionForward execute(…) {
   ...
   // Apply discount, if any
   BigDecimal amount = invoice.getAmount();
   if (amount.compareTo(THRESHOLD) > 0) {
      BigDecimal discountAmt =
           amount.multiply(DISCOUNT_FACTOR);
      invoice.setAmount(discountAmt);
   }
   ...
}
```

© Copyright 2003, Object Systems Group

Business-Tier Code in Actions – Example EJB lookup

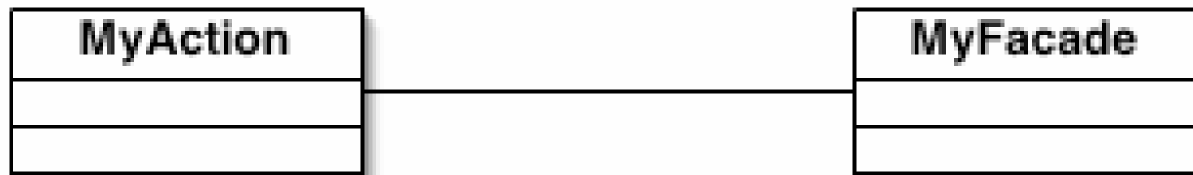
```
public ActionForward execute(…) {
    ...
    try {
      // lookup and narrow
      // create
      // DO search and return
    } catch(NamingException ne) {
     ...
    } catch (CreateException ce) {
    ...
}
```

Colorado Software Summit

© Copyright 2003, Object Systems Group

Refactoring: Move Business-Tier Code to Business Delegate

- Before



- After

© Copyright 2003, Object Systems Group

Move Business - Tier Code to Business Delegate – Mechanics

1. Find business-tier code in your Actions
   1. EJB Home lookup calls
   2. Any EJB find type methods (*i.e.* get DO's on sessions)
   3. Business logic
   4. Start with simplest Action
2. Create a ServiceLocator
   1. For applications using EJB move EJB Home lookup calls here
   2. For applications using R/O mapping tools move controller lookup here

© Copyright 2003, Object Systems Group

Move Business - Tier Code to Business Delegate – Mechanics

3. Create BusinessDelegate
    1. Replace session based get's in Actions with calls to the BusinessDelegate
4. Move business logic to your SessionFacade
    1. Could require a new façade or the expansion of the responsibilities of the existing façade.
    2. Could require adding functionality to an underlying entity and exposing that functionality through the façade.
5. Deploy & Test
6. Repeat for the rest of your Actions

© Copyright 2003, Object Systems Group

AntiPattern: Accessing ActionForms in the Session

- General Form
  - get/setAttribute on the session
  - Reimplementing the Struts code that manages the ActionForms
  - Directly invoking the code from Struts
- Symptoms & Consequences
  - NullPointerExceptions
  - Subtle bugs with UI that uses secondary forms
- Refactorings
  - Add ActionForm locator to base Action class

Colorado Software Summit

© Copyright 2003, Object Systems Group

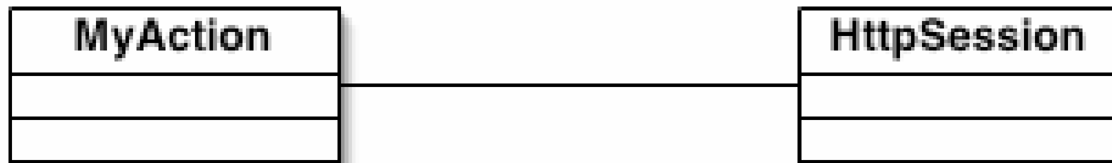
Accessing ActionForms in Session – Example

```
public ActionForward execute(...) {
    ...
    InvoiceForm alternateForm = (InvoiceForm)
      request.getSession().getAttribute("invoiceForm");
    ...
}
```

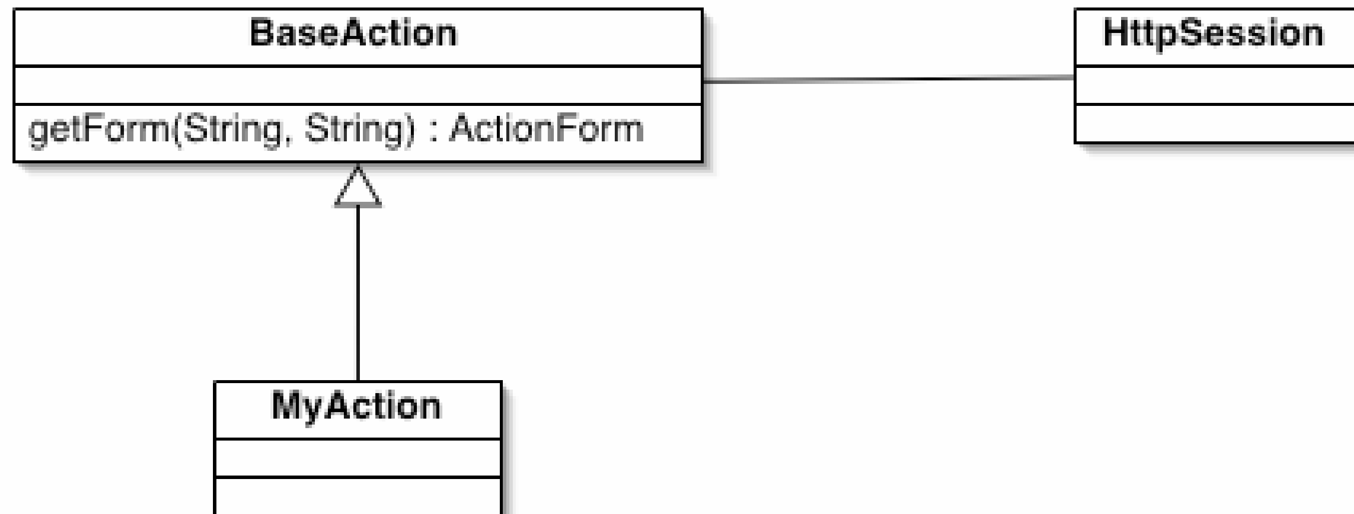
© Copyright 2003, Object Systems Group

Refactoring: Add ActionForm Locator Method to Base Class

- Before



- After



© Copyright 2003, Object Systems Group

Add Form Locator Method to Base Action – Mechanics

1. Implement a getForm Method in your base Action class
2. Locate all Actions that interact with more than one form
3. Ensure they subclass from the base Action
   1. This could require more work if the superclass must be changed
4. Replace any code used to get the form with the new method

© Copyright 2003, Object Systems Group

Struts TagLibs and JSPs

- Hard-coded Strings in JSPs
- Hard-coded Keys in JSPs
- Performing Business Logic in JSPs
- Hard-Coded Options in HTML Select Lists

© Copyright 2003, Object Systems Group

AntiPattern: Hard-Coded Strings in JSPs

- General Form
  - Hard coded strings in JSPs
- Symptoms & Consequences
  - Virtually impossible to internationalize
  - Maintenance headaches
- Refactorings
  - Introduce Resource Bundle

© Copyright 2003, Object Systems Group

Hard-Coded Strings in JSPs – Example

```
<h1>Wow A Heading</h1>
```

© Copyright 2003, Object Systems Group

Refactoring: Use Resource Bundles

- Before

```
<tr>
  <td>Invoice Number:</td>
  <td><html:text property="InvoiceNumber"/></td>
</tr>
```

- After

```
<tr>
  <td><bean:message key="label.invoice.number"/></td>
  <td><html:text property="InvoiceNumber"/></td>
</tr>
```

Colorado Software Summit

© Copyright 2003, Object Systems Group

Use Resource Bundles – Mechanics

1. Create a .properties file
   1. Place the file so it can be found at runtime – typically this is in the same directory as the struts-config.xml file
2. Begin with a single JSP.
3. Search for hard-coded strings.
4. Create an entry in the .properties file
   1. Use a naming convention that will help you to identify what the intent of the property is, like label.search.value

Colorado Software Summit

© Copyright 2003, Object Systems Group

Use Resource Bundles – Mechanics

5. Replace the hard-coded strings with bean:message tags.
6. Deploy and test
7. Repeat for each JSP

© Copyright 2003, Object Systems Group

AntiPattern: Hard-Coded Keys in JSPs

- General Form
  - Keys hard coded into JSPs
- Symptoms & Consequences
  - Maintenance Headaches
- Refactorings
  - Introduce Constants Class

© Copyright 2003, Object Systems Group

Refactoring: Introduce Constants Class

- Before

```
<tr>
  <td><bean:message key="label.invoice.number"/></td>
  <td><html:text property="InvoiceNumber"/></td>
</tr>
```

- After

```
<tr>
  <td>
    <bean:message key="${constants.labelInvoiceNumber}"/>
  </td>
  <td><html:text property="InvoiceNumber"/></td>
</tr>
```

© Copyright 2003, Object Systems Group

Introduce Constants Class – Mechanics

1. Create a Constants Class
   1. For ease of use in JSPs, make the constants class a Java Bean
   2. In JDK 1.5 this might become an enum
2. Begins with a single, simple JSP
3. Search for hard-coded keys
4. For each key create a new constant in the Java Class
   1. For large lists you should consider grouping your keys into multiple classes

© Copyright 2003, Object Systems Group

Introduce Constants Class – Mechanics

5. Replace each key with a constant
    1. If you can not use the expression language then you will use Scriptlets.
6. Test the changes
7. Repeat for each JSP

Colorado Software Summit

© Copyright 2003, Object Systems Group

AntiPattern: Perform Business Logic in JSPs

- General Form
  - Large scriptlets
- Symptoms & Consequences
  - Increased Complexity in JSPs
  - Increased Maintenance costs
- Refactorings
  - Move Logic to Java Bean

Colorado Software Summit

© Copyright 2003, Object Systems Group

Perform Business Logic in JSPs – Example

```
<tr>
  <td>Invoice Amount:</td>
  <td><%=invoice.getAmount() * discount%></td>
</tr>
```

© Copyright 2003, Object Systems Group

Refactoring: Move Logic to Java Bean

- Before

```
<%
  InvoiceDO invoice = (InvoiceDO)request.getSession().
        getAttribute("invoice");
  Integer value = invoice.doSomeStuff();
  ...
%>
```

Colorado Software Summit

© Copyright 2003, Object Systems Group

Refactoring: Move Logic to Java Bean

- After

```
public void doSomethingInteresting() {
   InvoiceDO invoice = (InvoiceDO)request.getSession().
         getAttribute("invoice");
   Integer value = invoice.doSomeStuff();
   ...
}
```

© Copyright 2003, Object Systems Group

Move Logic to Java Bean – Mechanics

1. Begin with the simplest JSP that contains one or more calculations.
2. Add a getter method in the corresponding value object.
3. Add a property, including a getter and setter, for a string version of the value in the corresponding ActionForm.
4. Modify the rendering code in the JSP to display the new form property.
5. Deploy and Test.
6. Continue with the next JSP.

© Copyright 2003, Object Systems Group

AntiPattern: Hard-Coded Options in HTML Select Lists

- General Form
  - Hard coded HTML select elements
- Symptoms & Consequences
  - Increased Maintenance
  - UI Inconsistent with Model
- Refactorings
  - Move Option Values to Helper Class

© Copyright 2003, Object Systems Group

Refactoring: Move Option Values to Helper Class

- Before

```
<html:select property="paymentTerm">
 <html:option value="0">None</html:option>
 <html:option value="1">Nnet 30 Days</html:option>
</html:select>
```

© Copyright 2003, Object Systems Group

Refactoring: Move Option Values to Helper Class

- After

```
<html:select property="paymentTerm">
 <bean:define id="values" name="invoiceForm"
              property="options.paymentTerms"
              type="java.util.ArrayList">
 <html:options collection="values" property="value"
               labelProperty=label"/>
 <html:option value="1">Nnet 30 Days</html:option>
</html:select>
```

© Copyright 2003, Object Systems Group

Move Option Values to Helper Class – Mechanics

1. Create a helper class that will return the list of value/label pairs
   1. This functionality might belong in an existing helper class
2. Add a constant to the appropriate value object to represent the value for the label
   1. It's best to keep the constants close to the value objects they represent

© Copyright 2003, Object Systems Group

Move Option Values to Helper Class – Mechanics

3. Add a method to the helper class to return a list of LabelValueBeans
   1. Create a LabelValueBean for each option and add it to the list
4. Replace the hard coded options with an html:options tag that uses the helper bean to get its values

Colorado Software Summit

© Copyright 2003, Object Systems Group

References

- *J2EE AntiPatterns*
  Bill Dudney, Stephen Asbury, Joseph Krozak, Kevin Wittkopf
  John Wiley & Sons; First edition (August 11, 2003)
  ISBN: 0-47114-615-3
- *Jakarta Pitfalls: Time-Saving Solutions for Struts, Ant, JUnit, and Cactus (Java Open Source Library)*
  Bill Dudney, Jonathan Lehr
  John Wiley & Sons; (July 2003)
  ISBN: 0-47144-915-6