Session 15
Struts Framework

Agenda

- **Introduction to Struts**
- **Advantages and Disadvantages of Struts**
- **Processing Requests with Action Objects Struts: Flow of Control**
  - The six basic steps in using Struts
  - Example: one result mapping - Same output page in all cases
  - Example: multiple result mappings - Different output pages depending on the input
- **Handling Request Parameters with Form Beans: Flow of Control**
  - Automatically creating bean to represent request data
  - Using bean:write to output bean properties

What is Apache Struts?

- Struts provides a **unified framework** for WAD based on J2EE that use the MVC architecture.
- Struts provides **utility classes** to handle many of the most common tasks in Web application development
- Struts provides **custom tag libraries** for outputting bean properties, generating HTML forms, iterating over various types of data structures, and conditionally outputting HTML.
- The <u>proper</u> way to view Struts: depend on the using purpose, but the MVC framework is the most common way of looking at Struts.

Advantages of Struts (vs. MVC Using RequestDispatcher)

- **Centralized file-based configuration**
  - Replace hard-coding by representing by configuration files (XML or property files) → loose coupling → easy changing
- **Form beans**
  - In JSP, you can use property="*" with jsp:setProperty to automatically populate a JavaBean component based on incoming request parameters.
  - Struts extends this capability to Java code and adds in several useful utilities to simplify the processing of request parameters.
- **Bean tags**
  - Struts provides a set of custom JSP tags (bean:write, in particular) that let you easily output the properties of JavaBeans components. Basically, these are concise and powerful variations of the standard jsp:useBean and jsp:getProperty tags.

Advantages of Struts (vs. Standard MVC), Continued

- **HTML tags**
  - Struts provides a set of custom JSP tags to create HTML forms that are associated with JavaBeans components.
- **Form field validation**
  - Struts has builtin capabilities for checking that form values are in the required format. If values are missing or in an improper format, the form can be automatically redisplayed with error messages and with the previously entered values maintained.
- **Consistent approach**
  - Struts encourages consistent use of MVC throughout your application.

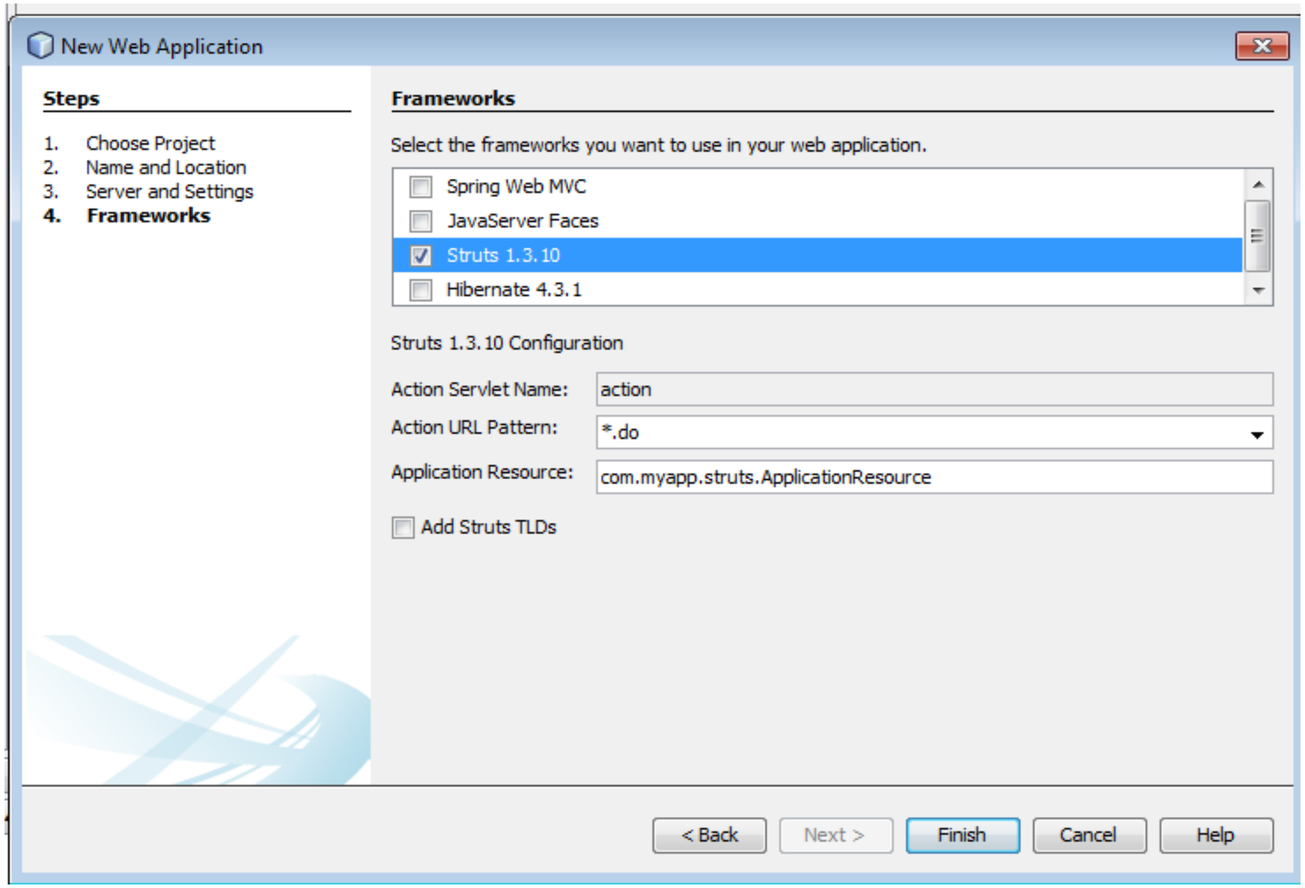
Disadvantages of Struts (vs. MVC)

- **Bigger learning curve**
- **Worse documentation than JSP**
- **Less transparent**
  - Harder to understand
  - Harder to benchmark and optimize
- **Rigid approach**
  - The flip side of the benefit that Struts encourages a consistent approach to MVC is that Struts makes it difficult (but by no means impossible) to use other approaches.

Downloading and Configuring Struts

Already existed on Netbeans

- Create new project → Java Web → Web Application
- Next → Next → Framework → Struts 1.3.10 → Finish



New Web Application

Steps

1. Choose Project
2. Name and Location
3. Server and Settings
4. **Frameworks**

Frameworks

Select the frameworks you want to use in your web application.

- [ ] Spring Web MVC
- [ ] JavaServer Faces
- [x] Struts 1.3.10
- [ ] Hibernate 4.3.1

Struts 1.3.10 Configuration

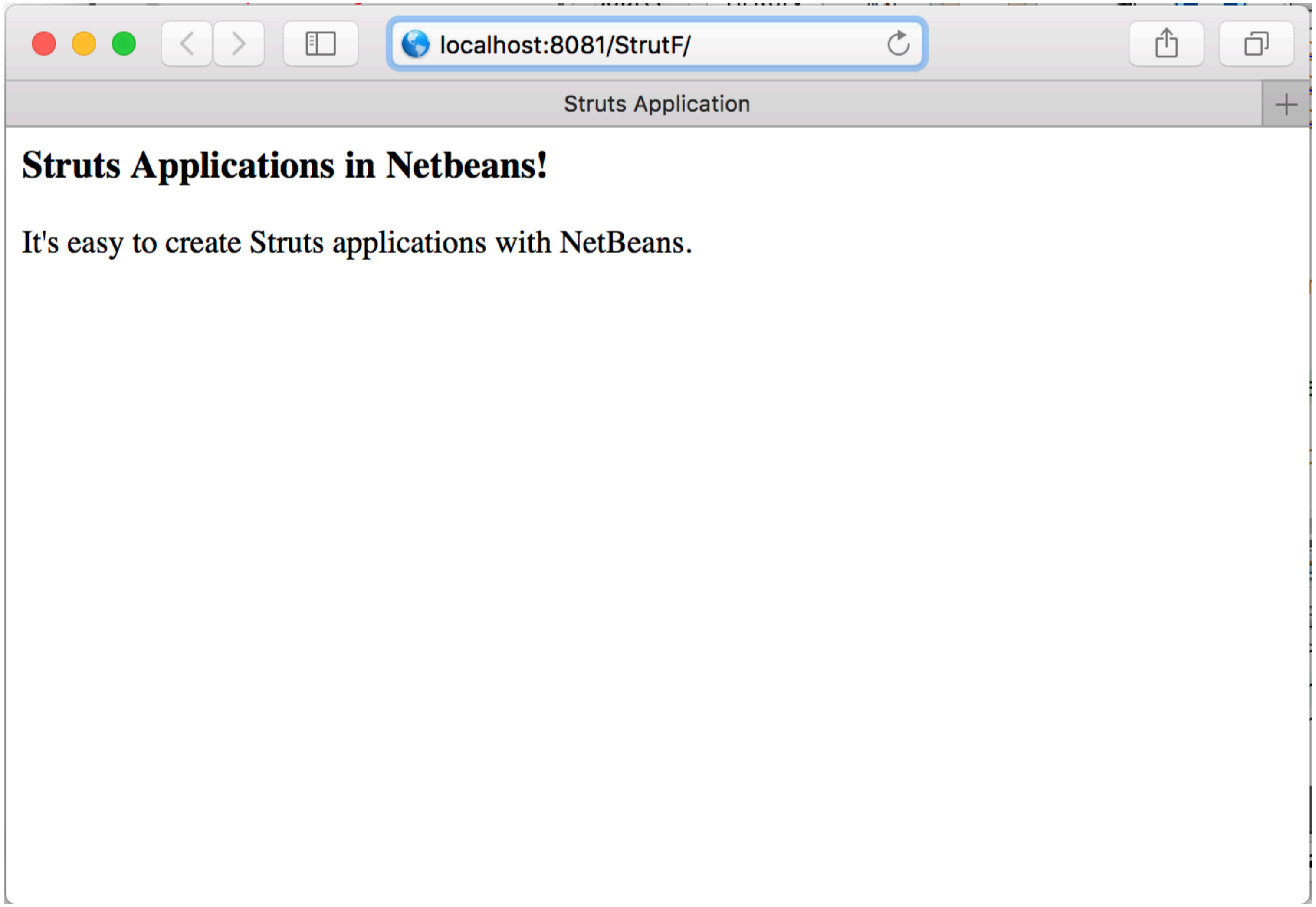
Action Servlet Name: action

Action URL Pattern: *.do

Application Resource: com.myapp.struts.ApplicationResource

- [ ] Add Struts TLDs

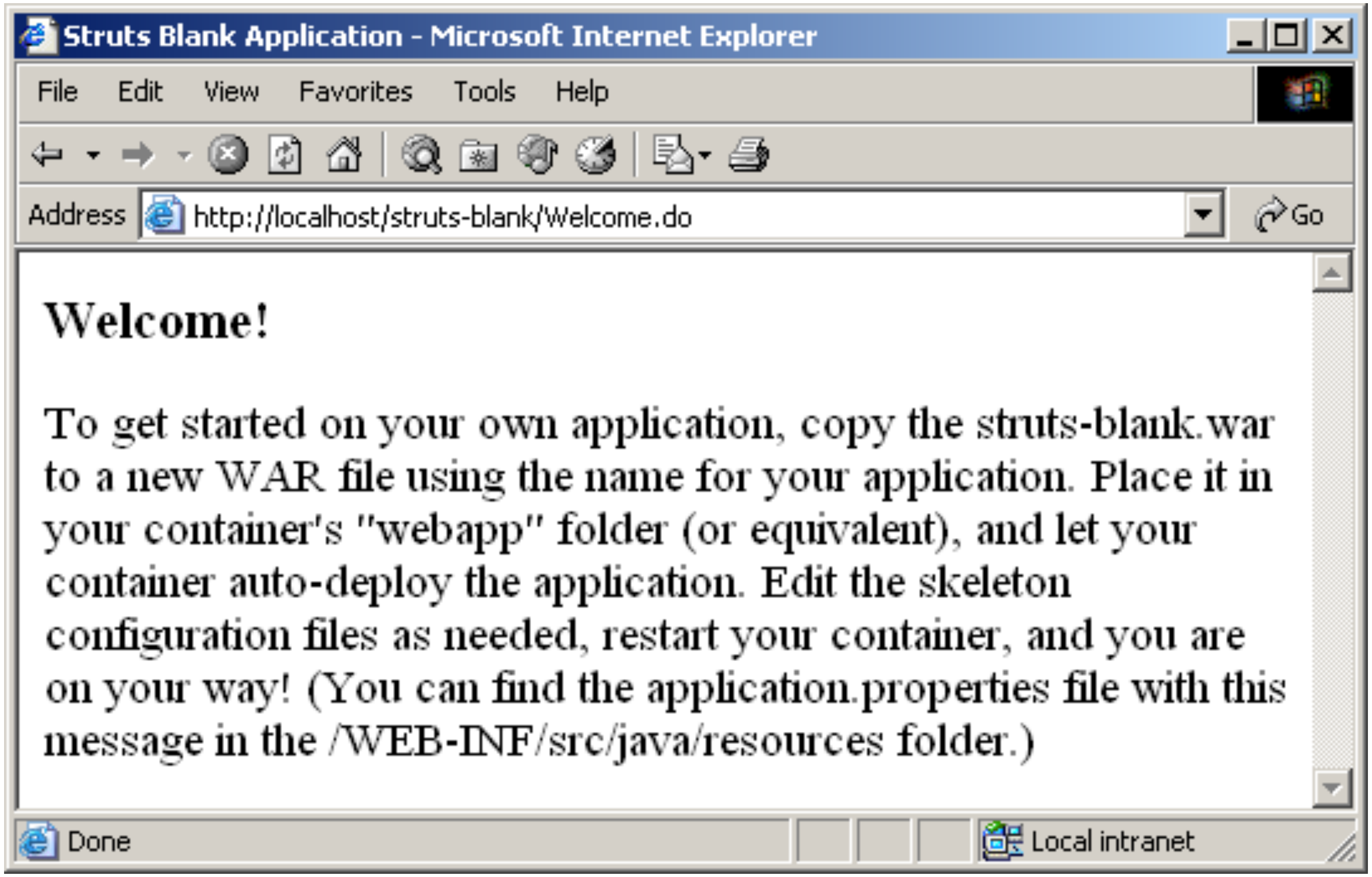
Testing Struts



Struts Applications in Netbeans!

It's easy to create Struts applications with NetBeans.

Testing Struts (Results)


Struts Blank Application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/struts-blank/Welcome.do
Go
Done
Local intranet

Welcome!

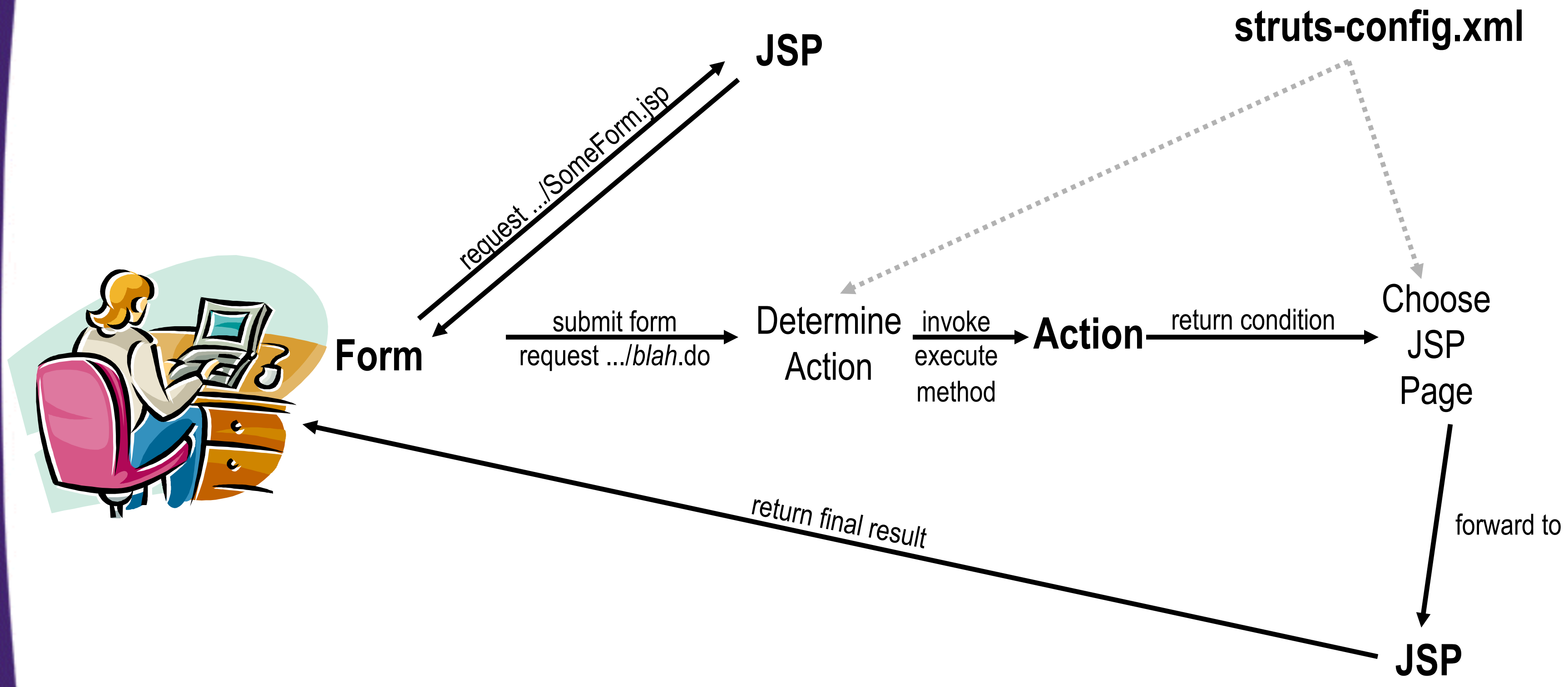
To get started on your own application, copy the struts-blank.war to a new WAR file using the name for your application. Place it in your container's "webapp" folder (or equivalent), and let your container auto-deploy the application. Edit the skeleton configuration files as needed, restart your container, and you are on your way! (You can find the application.properties file with this message in the /WEB-INF/src/java/resources folder.)

Testing Struts on Netbeans http://www.quickprogrammingtips.com/struts2/struts-2-netbeans-tutorial.html

Ten-Step Struts Development Plan

1. Gather and define the application requirements.
2. Define and develop each screen requirement in terms of the data collected and/or displayed.
3. Determine all the access paths for each screen.
4. Define the ActionMappings that correlate to the application business logic.
5. Create the ActionForms with defined properties from the screen requirements (this can include the validation portions as well).
6. Develop Actions to be called by the ActionMappings that, in turn, call the appropriatehelpers and forward to JSPs.
7. Develop the application business logic (Beans, EJBs, etc.).
8. Create JSPs to match the workflows using the ActionMappings.
9. Build the appropriate configuration files; this includes struts−config.xml and web.xml.
10. Build, test, deploy.

Processing Requests with Action Objects - Struts Flow of Control



Struts Flow of Control

- **The user requests a form**
  - For now, we use normal HTML to build the form
    - Later we will use the Struts html:form tag
- **The form is submitted to a URL of the form *blah*.do.**
  - That address is mapped by struts-config.xml to an Action class
- **The execute method of the Action object is invoked**
  - One of the arguments to execute is a form bean that is automatically created and whose properties are automatically populated with the incoming form data
  - The Action object then invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope.
  - The Action uses mapping.findForward to return a condition, and the conditions are mapped by struts-config.xml to various JSP pages.
- **Struts forwards request to the appropriate JSP page**
  - The page can use bean:write or the JSP 2.0 EL to output bean properties
  - The page can use bean:message to output fixed strings

The Six Basic Steps in Using Struts

1. **Modify struts-config.xml.**
   Use WEB-INF/struts-config.xml to:
   – Map incoming .do addresses to Action classes
   – Map return conditions to JSP pages
   – Declare any form beans that are being used.
   – *Be sure to restart the server after modifying struts-config.xml*; the file is read only when the Web application is first loaded.
2. **Define a form bean.**
   – This bean is a class the extends ActionForm and will represent the data submitted by the user. It is automatically populated when the input form is submitted. Beans are postponed until the next section.

The Six Basic Steps in Using Struts

3. Create results beans.
   - In the MVC architecture, the business-logic and data-access code create the results and the JSP pages present them. To transfer the results from one layer to the other, they are stored in beans. These beans differ from form beans in that they need extend no particular class, and they represent the output of the computational process, not the input to the process. Beans will be discussed in the next section.
4. **Define an Action class to handle requests.**
   - The struts-config.xml file designates the Action classes that handle requests for various URLs. The Action objects themselves need to do the real work: invoke the appropriate business- and data-access-logic, store the results in beans, and designate the type of situation (missing data, database error, success category 1, success category 2, etc.) that is appropriate for the results. The struts-config.xml file then decides which JSP page should apply to that situation.

The Six Basic Steps in Using Struts

5. **Create form that invokes *blah*.do.**
   - Create an input form whose ACTION corresponds to one of the .do addresses listed in struts-config.xml.
   - In a later lecture, we will discuss the advantages of using the Struts html:form tag to build this input form.
6. **Display results in JSP.**
   - Since Struts is built around MVC, these JSP pages should avoid JSP scripting elements whenever possible. For basic Struts, these pages usually use the bean:write tag, but in JSP 2.0 the JSP 2.0 expression language is a viable alternative.
   - In most cases, the JSP pages only make sense when the request is funneled through the Action, so the pages go in WEB-INF.
   - If the JSP pages makes sense independently of the Action (e.g., if they display session data), then the JSP pages should be placed in a regular subdirectory of the Web application, and the forward entries in struts-config.xml should say <forward ... redirect="true"/>.

Example 1: One Result Mapping

- URL
  - http://hostname/struts-actions/register1.do
- Action Class
  - RegisterAction1
    - RegisterAction1 extends Action and is in the coreservlets package.
    - The execute method of RegisterAction1 always returns "success"
- Results page
  - /WEB-INF/results/confirm.jsp
    - But the URL shown will still be register1.do

Step 1A (Modify struts-config.xml)

- **Map incoming .do addresses to Action classes**
  - In this case, we designate that RegisterAction1 should handle requests for register1.do. To accomplish this, we add an action entry to action-mappings, where action has the following attributes.
    - **path**: the relative path that should be mapped to the Action, minus the .do extension. Thus, path="/register1" refers to http://hostname/webAppName/register1.do.
    - **type**: the fully qualified class name of the Action class that should be invoked when a request for the path is received.

```
<action-mappings>
          <!-- .do implied automatically -->
  <action path="/register1"
          type="coreservlets.RegisterAction1">
    ...
  </action>
</action-mappings>
```

Step 1B (Modify struts-config.xml)

- **Map return conditions to JSP pages**
  - In this case, we use the forward element to say that confirm.jsp applies when the execute method of RegisterAction1 returns "success", as follows:

```
<forward name="success"
            path="/WEB-INF/results/confirm.jsp"/>
```

  - Note: if the same forward is used by multiple actions, you can put the forward declaration in a global-forwards section (before action-mappings) instead of in the action.

```
<global-forwards>
  <forward name="success"
            path="/WEB-INF/results/confirm.jsp"/>
</global-forwards>
```

Step 1 (Modify struts-config.xml) – Final struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC ... >
<struts-config>
  <action-mappings>
    <action path="/register1"
            type="coreservlets.RegisterAction1">
      <forward name="success"
               path="/WEB-
   INF/results/confirm.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

Steps 2 and 3

- **Define a form bean.**
  - Beans are postponed until the next section, so this step is omitted for now.
- **Create results beans.**
  - Beans are postponed until the next section, so this step is omitted for now.

Step 4 (Define an Action Class to Handle Requests)

- **Action subclasses should... be in a package.**
  - In this case, we have

    ```
    package coreservlets;
    ```

    - This means that the class file should go in *your_web_app*/WEB-INF/classes/coreservlets/.
- **Action subclasses should... add Struts-specific import statements to whatever imports are otherwise needed.**
  - In this case, we have

    ```
    import javax.servlet.http.*;
    import org.apache.struts.action.*;
    ```

Step 4 (Define an Action Class to Handle Requests)

- Action subclasses should ... extend Action
- Action subclasses should ... override execute
  - In this case, we have

```
public class RegisterAction1 extends Action {
  public ActionForward
        execute(ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response)
      throws Exception {
    ...
  }
}
```

Step 4 (Define an Action Class to Handle Requests)

- **Action subclasses should ... return mapping.findForward.**
  - The execute method should have one or more return values.
  - These values will then be mapped to specific JSP pages by forward entries in struts-config.xml. In this case, we simply return "success" in all situations.

```
public ActionForward
      execute(ActionMapping mapping,
              ActionForm form,
              HttpServletRequest request,
              HttpServletResponse response)
    throws Exception {
  return(mapping.findForward("success"));
}
```

Step 4 (Define an Action Class to Handle Requests) – Final Code

```
package coreservlets;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class RegisterAction1 extends Action {
  public ActionForward
             execute(ActionMapping mapping,
                     ActionForm form,
                     HttpServletRequest request,
                     HttpServletResponse response)
      throws Exception {
    return(mapping.findForward("success"));
  }
}
```

Step 5 (Create form that invokes *blah*.do)

- We need an HTML form that invokes http://hostname/struts-actions/register1.do.

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>New Account Registration</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>New Account Registration</H1>
<FORM ACTION="register1.do" METHOD="POST">
  Email address: <INPUT TYPE="TEXT"
  NAME="email"><BR>
  Password: <INPUT TYPE="PASSWORD"
  NAME="password"><BR>
  <INPUT TYPE="SUBMIT" VALUE="Sign Me Up!">
</FORM>
</CENTER>
</BODY></HTML>
```

Step 6 (Display results in JSP)

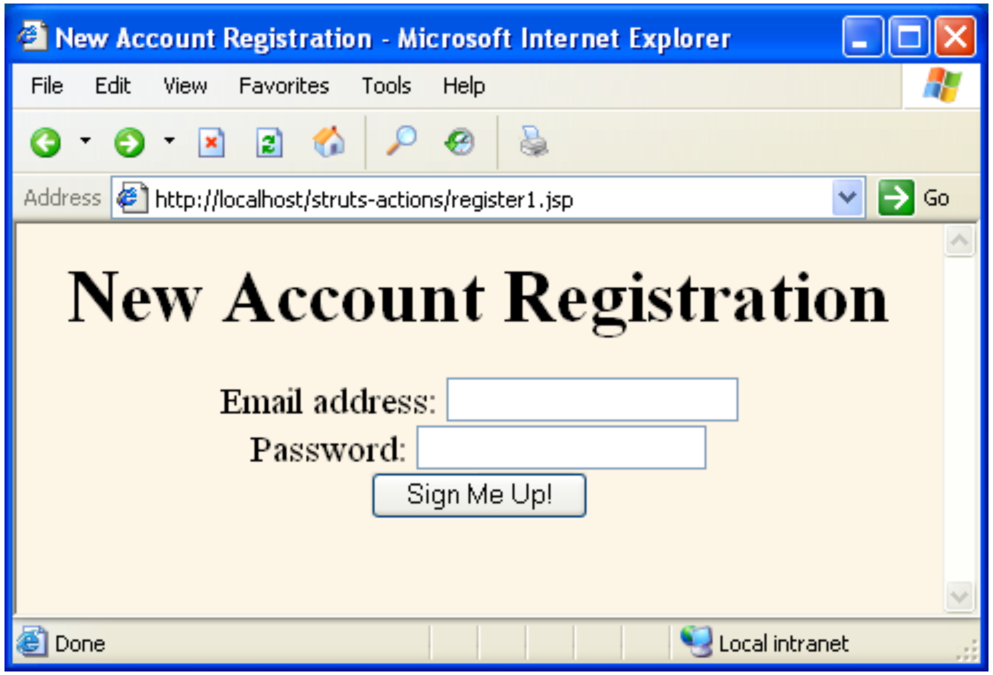
- **In general, there can be several possible JSP pages**
  - Corresponding to the various possible return values of the execute method of the Action.
- **In struts-config.xml, each JSP page is declared in a forward entry within the appropriate action.**
  - In this simple case, the only return value is "success", so /WEB-INF/results/confirm.jsp is used in all cases.
- **This JSP page will just display a simple message (see next slide)**

Step 6 (Display results in JSP) – Final Code

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Success</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>You have registered successfully.</H1>
Congratulations
</CENTER>
</BODY></HTML>
```

Example 1: Results

- **First, the HTML form is invoked with the URL http://localhost/struts-actions/register1.jsp**


New Account Registration - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Address
http://localhost/struts-actions/register1.jsp
Go
Done
Local intranet

New Account Registration

Email address:

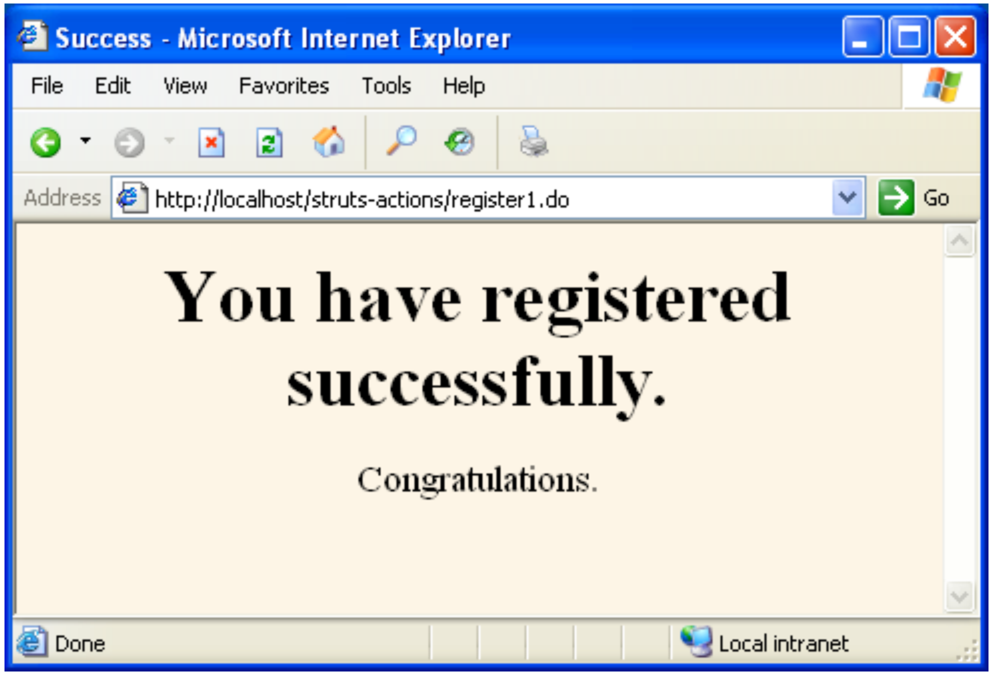
Password:

Sign Me Up!

Example 1: Results

- **This form is then filled in and submitted,**
  - With the form's ACTION resulting in the URL http://localhost/struts-actions/register1.do.
- **This address is mapped by struts-config.xml**
  - To the RegisterAction1 class, whose execute method is invoked.
- **This method returns mapping.findForward**
  - With a value of "success"
- **That value is mapped by struts-config.xml**
  - To /WEB-INF/results/confirm.jsp,
    - Which is the final result displayed to the user.
    - However, since the JSP page is invoked with RequestDispatcher.forward, not response.sendRedirect, the URL displayed to the user is register1.do, not confirm.jsp.

Example 1: Results


Success - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Done
Local intranet

http://localhost/struts-actions/register1.do

You have registered successfully.

Congratulations.

Example 2: Multiple Result Mappings

- **URL**
  - http://hostname/struts-actions/register2.do
- **Action Class**
  - RegisterAction2.
    - The execute method of RegisterAction2 returns "success",
      "bad-address", or "bad-password"
- **Results pages**
  - /WEB-INF/results/confirm.jsp,
  - /WEB-INF/results/bad-address.jsp, and
  - /WEB-INF/results/bad-password.jsp, respectively.
- **Main new feature of this example**
  - The use of multiple forward entries within the action element.

Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
  - In this case, we use the action element to designate that RegisterAction2 should handle requests for register2.do (again, note that .do is implied, not listed explicitly).
- **Map return conditions to JSP pages**
  - In this case, we use multiple forward elements, one for each possible return value of the execute method of the RegisterAction2 class.
- **Declare any form beans that are being used.**
  - Beans are postponed until the next section, so this step is omitted for now.

Step 1 (Modify struts-config.xml) – Final Code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC ... >
<struts-config>
  <action-mappings>
    ...
    <action path="/register2"
            type="coreservlets.RegisterAction2">
      <forward name="bad-address"
               path="/WEB-INF/results/bad-address.jsp"/>
      <forward name="bad-password"
               path="/WEB-INF/results/bad-password.jsp"/>
      <forward name="success"
               path="/WEB-INF/results/confirm.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

Steps 2 and 3

- **Define a form bean.**
  - Beans are postponed until the next section, so this step is omitted for now.
- **Create results beans.**
  - Beans are postponed until the next section, so this step is omitted for now.

Step 4 (Define an Action Class to Handle Requests)

- **Similar to the previous example except for multiple mapping.findForward entries**
  - We return "bad-address" if the email address is missing, is less then three characters long, or does not contain an "@" sign.
  - We return "bad-password" if the password is missing or is less than six characters long.
  - Otherwise we return "success".
- **In this simple example we use request.getParameter explicitly.**
  - In later examples we let Struts automatically populate a bean from the request data.

Step 4 (Define an Action Class to Handle Requests) – Final Code

```
public class RegisterAction2 extends Action {
  public ActionForward
              execute(ActionMapping mapping,
                      ActionForm form,
                      HttpServletRequest request,
                      HttpServletResponse response)
      throws Exception {
    String email = request.getParameter("email");
    String password = request.getParameter("password");
    if ((email == null) ||
        (email.trim().length() < 3) ||
        (email.indexOf("@") == -1)) {
      return(mapping.findForward("bad-address"));
    } else if ((password == null) ||
               (password.trim().length() < 6)) {
      return(mapping.findForward("bad-password"));
    } else {
      return(mapping.findForward("success"));
}}}
```

Step 5 (Create Form that Invokes *blah*.do)

- We need an HTML form that invokes http://hostname/struts-actions/register2.do.

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>New Account Registration</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>New Account Registration</H1>
<FORM ACTION="register2.do" METHOD="POST">
  Email address: <INPUT TYPE="TEXT" NAME="email"><BR>
  Password: <INPUT TYPE="PASSWORD" NAME="password"><BR>
  <INPUT TYPE="SUBMIT" VALUE="Sign Me Up!">
</FORM>
</CENTER>
</BODY></HTML>
```

Step 6 (Display results in JSP) First Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Email
  Address</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Email Address</H1>
Address must be of the form username@host.
Please <A HREF="register2.jsp">
try again</A>.
</CENTER>
</BODY></HTML>
```

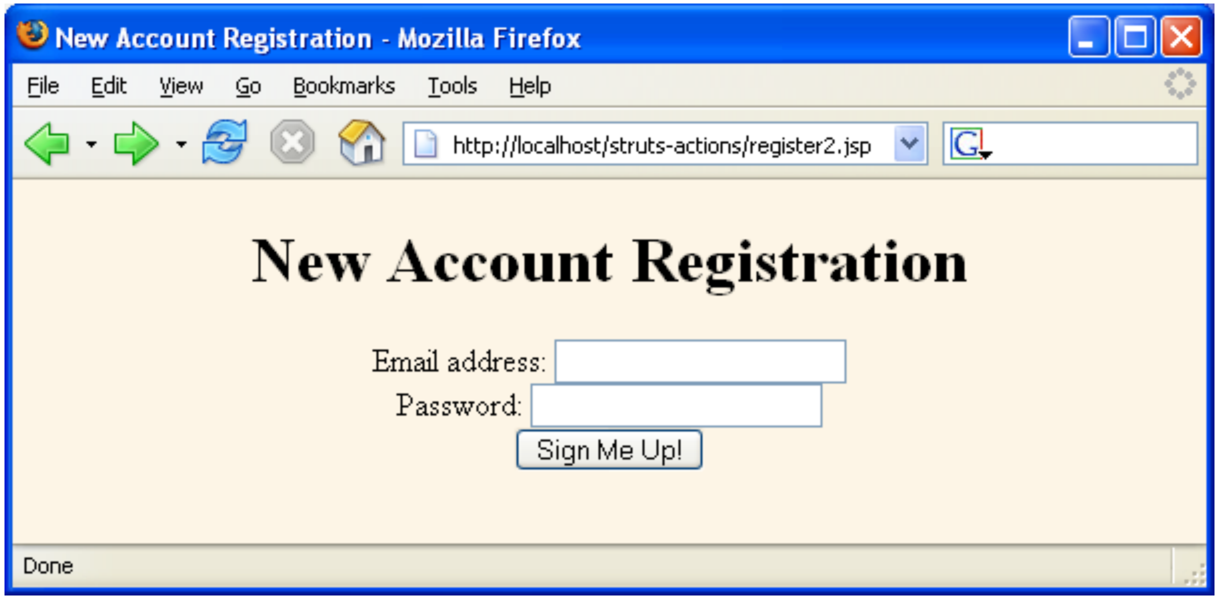
Step 6 (Display results in JSP) Second Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Password</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Password</H1>
Password must contain at least six characters.
Please <A HREF="register2.jsp">
try again</A>.
</CENTER>
</BODY></HTML>
```

Step 6 (Display results in JSP) Same confirm.jsp Shown Earlier

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Success</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>You have registered successfully.</H1>
Congratulations
</CENTER>
</BODY></HTML>
```

Example 2: Results (Initial Form)

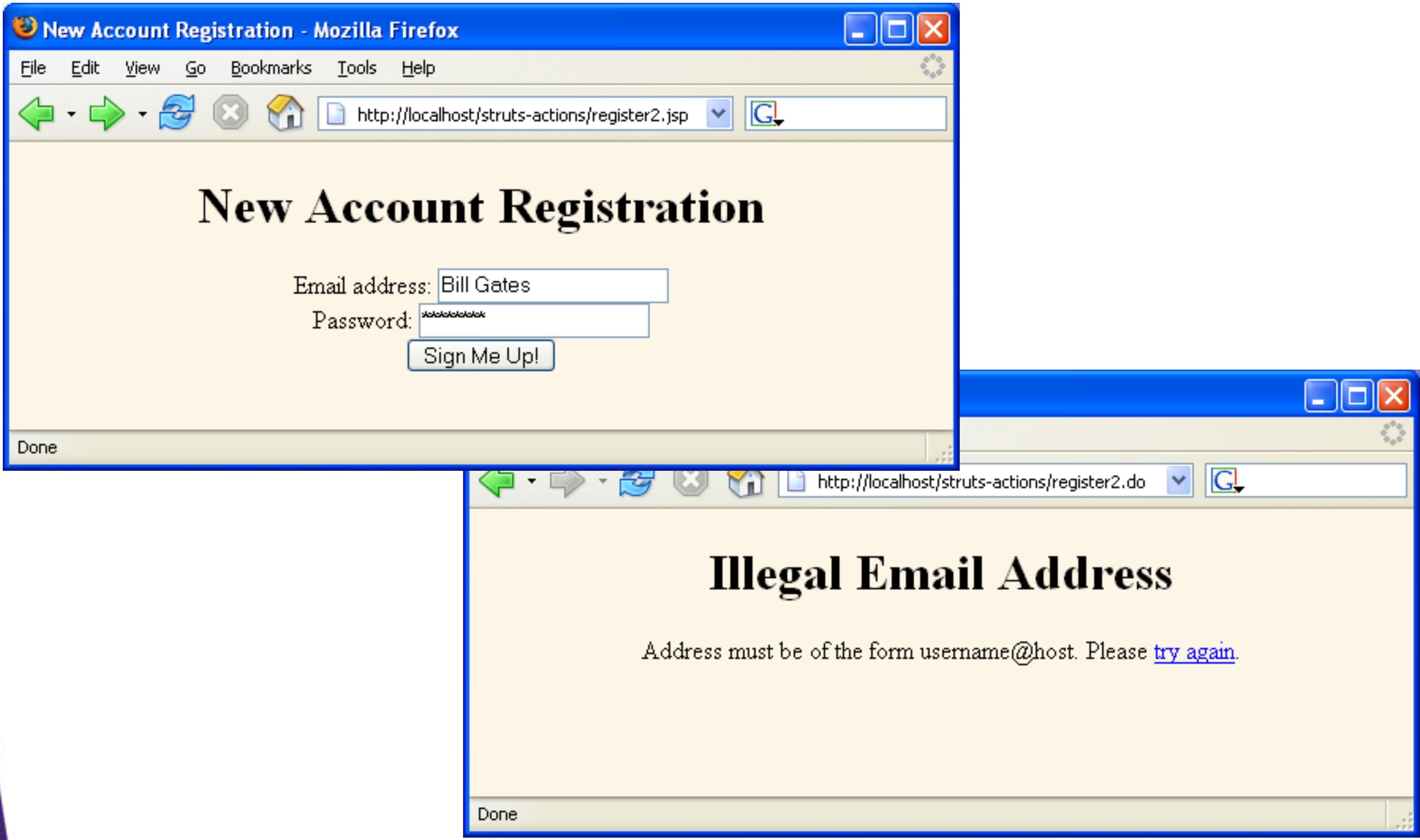

New Account Registration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/struts-actions/register2.jsp
Sign Me Up!
Done

New Account Registration

Email address:

Password:

Example 2: Results (Bad Address)


New Account Registration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/struts-actions/register2.jsp
Sign Me Up!
Done
http://localhost/struts-actions/register2.do
Done

New Account Registration

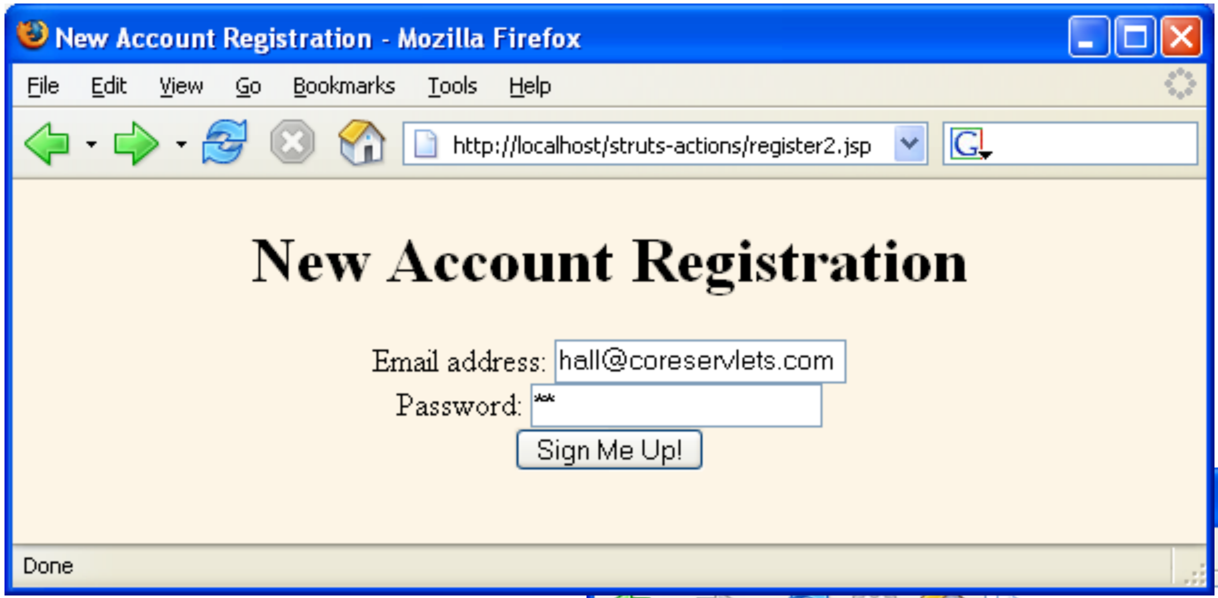
Email address: Bill Gates

Password: *********

Illegal Email Address

Address must be of the form username@host. Please try again.

Example 2: Results (Bad Password)


New Account Registration - Mozilla Firefox

http://localhost/struts-actions/register2.jsp

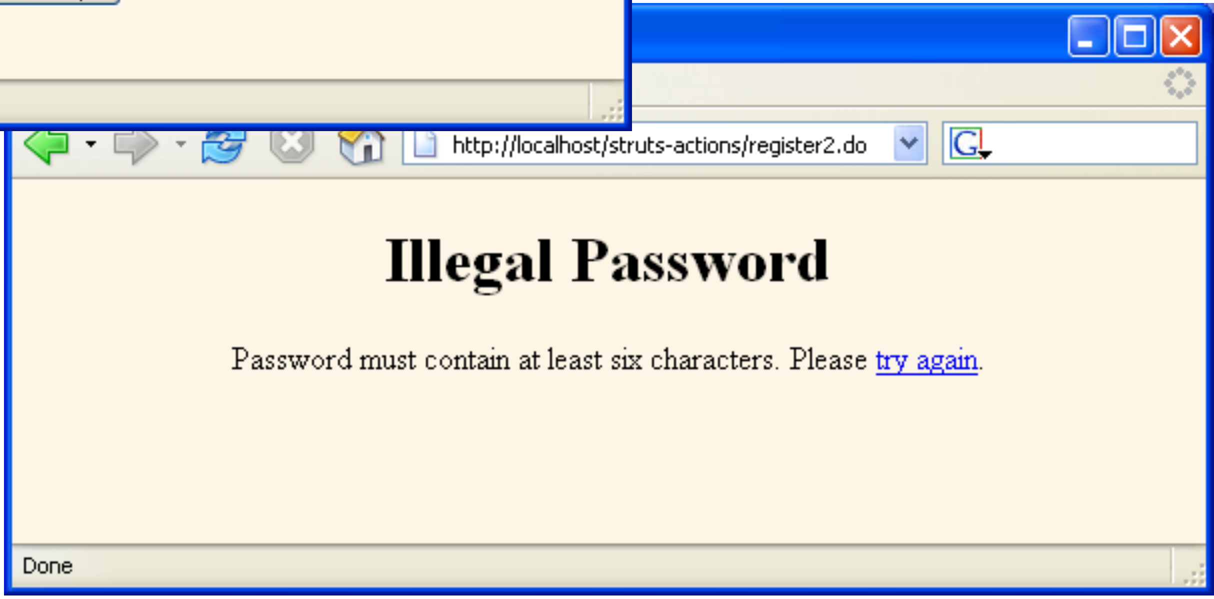
New Account Registration

Email address: hall@coreservlets.com

Password: **

Sign Me Up!

Done

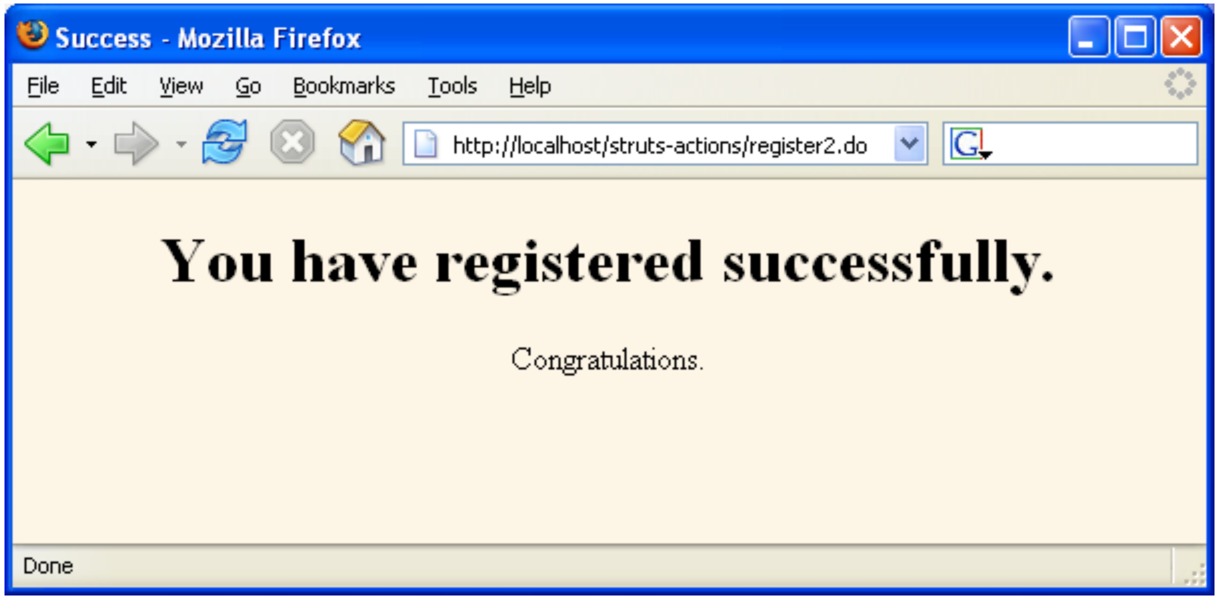

http://localhost/struts-actions/register2.do

Illegal Password

Password must contain at least six characters. Please try again.

Done

Example 2: Results (Success)


Success - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/struts-actions/register2.do
Done

You have registered successfully.

Congratulations.

Combining Shared Condition (Forward) Mappings

- **Idea**
  - If the same condition is mapped to the same JSP page in multiple actions, you can move the forward to a global-forwards section to avoid repetition
- **Syntax**
  - The global-forwards section goes before action-mappings, not within it
  - The forward entries within global-forwards have the same syntax and behavior as forward entries within action
- **Example**

```
<global-forwards>
  <forward name="success"
           path="/WEB-
  INF/results/confirm.jsp"/>
</global-forwards>
```

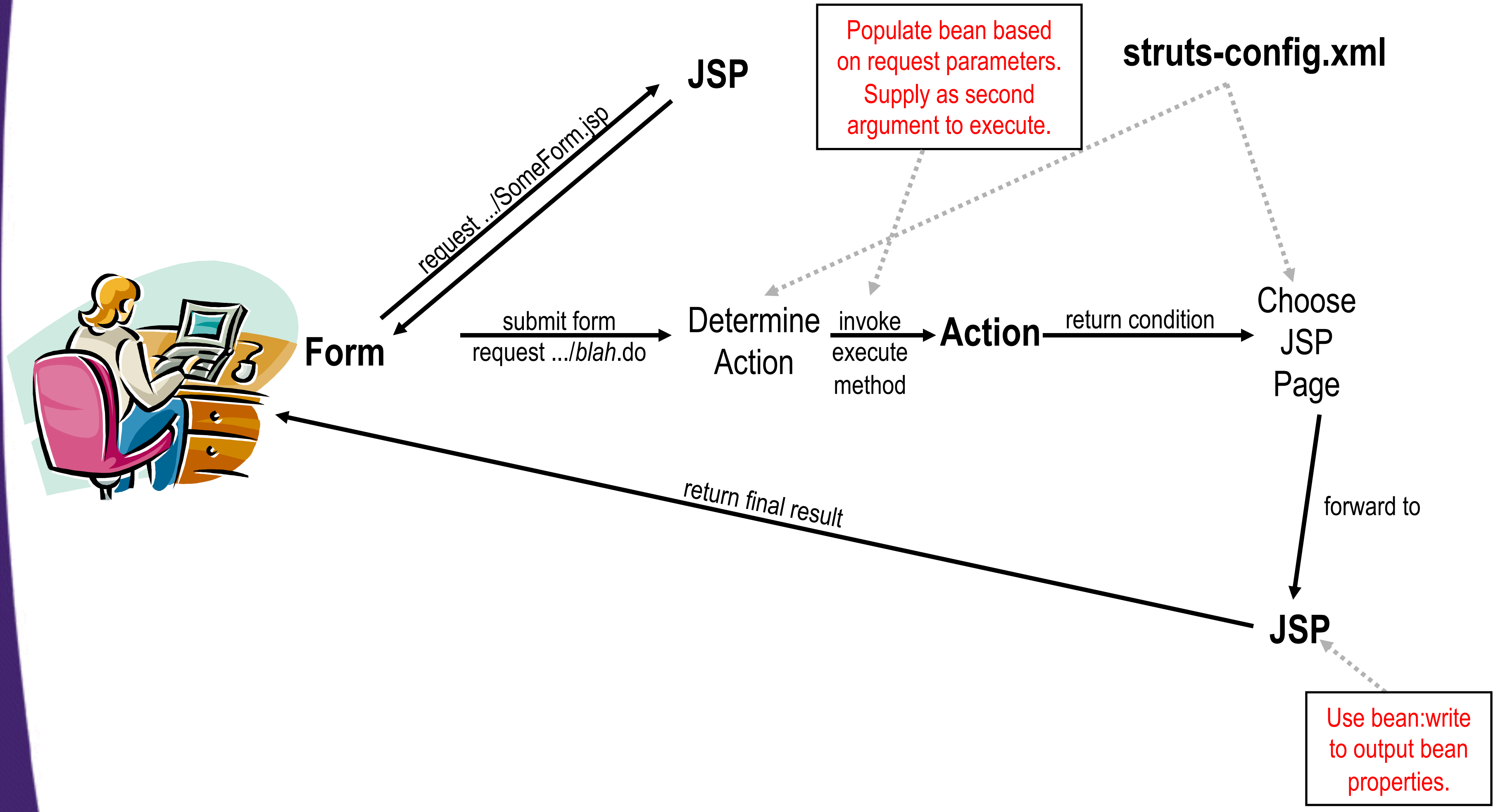
Combining Shared Condition (Forward) Mappings: Old

```
<action-mappings>
  <action path="/register1"
          type="coreservlets.RegisterAction1">
    <forward name="success"
             path="/WEB-INF/results/confirm.jsp"/>
  </action>
  <action path="/register2"
          type="coreservlets.RegisterAction2">
    <forward name="bad-address"
             path="/WEB-INF/results/bad-
address.jsp"/>
    <forward name="bad-password"
             path="/WEB-INF/results/bad-
password.jsp"/>
    <forward name="success"
             path="/WEB-INF/results/confirm.jsp"/>
  </action>
  ...
</action-mappings>
```

Combining Shared Condition (Forward) Mappings: New

```
<global-forwards>
  <forward name="success"
           path="/WEB-INF/results/confirm.jsp"/>
</global-forwards>
<action-mappings>
  <action path="/register1"
          type="coreservlets.RegisterAction1">
  </action>
  <action path="/register2"
          type="coreservlets.RegisterAction2">
    <forward name="bad-address"
             path="/WEB-INF/results/bad-
address.jsp"/>
    <forward name="bad-password"
             path="/WEB-INF/results/bad-
password.jsp"/>
  </action>
  ...
</action-mappings>
```

Handling Request Parameters with Form Beans – Struts Flow of Control



Struts Flow of Control

- **The user requests a form**
  - For now, we use normal HTML to build the form
    - Later we will use the Struts html:form tag
- **The form is submitted to a URL of the form *blah*.do.**
  - That address is mapped by struts-config.xml to an Action class
- **The execute method of the Action object is invoked**
  - One of the arguments to execute is a form bean that is automatically created and whose properties are automatically populated based on incoming request parameters of the same name
  - The Action object then invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope.
  - The Action uses mapping.findForward to return a condition, and the conditions are mapped by struts-config.xml to various JSP pages.
- **Struts forwards request to the appropriate JSP page**
  - The page can use bean:write or the JSP 2.0 EL to output bean properties
  - The page can also use bean:message to output fixed strings

New Capabilities

- **Defining a form bean**
  - Struts lets you define a bean that represents the incoming request data. Struts will create and populate the bean for you, and pass it to the Action as the second argument to the execute method.
    - Bean must extend ActionForm
    - Bean must be declared in struts-config.xml with form-beans
- **Outputting bean properties**
  - You can use the Struts bean:write tag to output bean properties in JSP pages.

The Six Basic Steps in Using Struts: Updates for Bean Use

1. **Modify struts-config.xml.**
   Use WEB-INF/struts-config.xml to:
   - Map incoming .do addresses to Action classes
   - Map return conditions to JSP pages
   - Declare any form beans that are being used.
   - Restart server after modifying struts-config.xml.
2. **Define a form bean.**
   - This bean extends ActionForm and represents the data submitted by the user. It is automatically populated when the input form is submitted. More precisely:
     1. The reset method is called (useful for session-scoped beans)
     2. For each incoming request parameter, the corresponding setter method is called
     3. The validate method is called (possibly preventing the Action)

The Six Basic Steps in Using Struts: Updates for Bean Use

3. **Create results beans.**
   – These are normal beans of the sort used in MVC when implemented directly with RequestDispatcher. That is, they represent the results of the business logic and data access code. These beans are stored in request, session, or application scope with the setAttribute method of HttpServletRequest, HttpSession, or ServletContext, just as in normal non-Struts applications.
4. **Define an Action class to handle requests.**
   – Rather than calling request.getParameter explicitly as in the previous example, the execute method casts the ActionForm argument to the specific form bean class, then uses getter methods to access the properties of the object.

The Six Basic Steps in Using Struts: Updates for Bean Use

5. **Create form that invokes *blah*.do.**
   - For now, we will use static HTML
     - Later, we will use the html:form tag to guarantee that the textfield names correspond to the bean property names, and to make it easy to fill in the form based on values in the app
     - Later, we will also use bean:message to output fixed strings from a properties file
6. **Display results in JSP.**
   - The JSP page uses the **bean:write** tag to output properties of the form and result beans.
   - It may also use the bean:message tag to output standard messages and text labels that are defined in a properties file (resource bundle).

Example 1: Form and Results Beans

- **URL**
  - http://*hostname*/struts-beans/register1.do
- **Action Class**
  - BeanRegisterAction
    - Instead of reading form data explicitly with request.getParameter, the execute method uses a bean that is automatically filled in from the request data.
    - As in the previous example, this method returns "success", "bad-address", or "bad-password"
- **Results pages**
  - /WEB-INF/results/confirm-registration1.jsp
  - /WEB-INF/results/bad-address1.jsp
  - /WEB-INF/results/bad-password1.jsp

New Features of This Example

- **The use of a bean to represent the incoming form data.**
  - This bean extends the ActionForm class, is declared in struts-config.xml with the form-bean tag, and is referenced in struts-config.xml with name and scope attributes in the action element.
- **The use of a regular bean to represent custom results.**
  - As with beans used with regular MVC, this bean need not extend any particular class and requires no special struts-config.xml declarations.
- **The use of the Struts bean:write tags to output bean properties in the JSP page that displays the final results.**
  - This is basically a more powerful and concise alternative to the standard jsp:getProperty tag. Before we use bean:write, we have to import the "bean" tag library as follows.

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
                prefix="bean" %>
```

Step 1 (Modify struts-config.xml)

- **Map incoming .do address to Action classes**
  - As before, we use the action element (to designate that BeanRegisterAction should handle requests for register1.do).
- **Map return conditions to JSP pages**
  - As before, we use multiple forward elements, one for each possible return value of the execute method
- **Declare any form beans that are being used.**
  - Use **form-bean** (within form-beans) with these two attributes:
    - **name**: a name that will match the name attribute of the action element.
    - **type**: the fully qualified classname of the bean.
  - Note that the form-beans section goes *before* action-mappings in struts-config.xml, not inside action-mappings
  - Here is an example:

```
<form-beans>
  <form-bean name="userFormBean"
             type="coreservlets.UserFormBean"/>
</form-beans>
```

Step 1 (Modify struts-config.xml), Continued

- **Update action declaration**
  - After declaring the bean in the form-beans section, you need to add two new attributes to the action element: name and scope
    - **name**: a bean name matching the form-bean declaration.
    - **scope**: request or session. Surprisingly, session is the default, but always explicitly list the scope anyhow. We want request here.
  - Here is an example.

```
<action path="/register1"

  type="coreservlets.BeanRegisterAction"
        name="userFormBean"
        scope="request">
```

Step 1 (Modify struts-config.xml) -- Final Code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC ... >
<struts-config>
  <form-beans>
    <form-bean name="userFormBean"
               type="coreservlets.UserFormBean"/>
  </form-beans>
  <action-mappings>
     <action path="/register1"
             type="coreservlets.BeanRegisterAction"
             name="userFormBean"
             scope="request">
      <forward name="bad-address"
               path="/WEB-INF/results/bad-address1.jsp"/>
      <forward name="bad-password"
               path="/WEB-INF/results/bad-password1.jsp"/>
      <forward name="success"
          path="/WEB-INF/results/confirm-registration1.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

Step 2 (Define a Form Bean)

- **A form bean is a Java object that will be automatically filled in based on the incoming form parameters, then passed to the execute method. Requirements:**
  - **It must extend ActionForm.**
    - The argument to execute is of type ActionForm. Cast the value to your real type, and then each bean property has the value of the request parameter with the matching name.
  - **It must have a zero argument constructor.**
    - The system will automatically call this default constructor.
  - **It must have settable bean properties that correspond to the request parameter names.**
    - That is, it must have a set*Blah* method corresponding to each incoming request parameter named *blah*. The properties should be of type String (i.e., each set*Blah* method should take a String as an argument).
  - **It must have gettable bean properties for each property that you want to output in the JSP page.**
    - That is, it must have a get*Blah* method corresponding to each bean property that you want to display in JSP without using Java syntax.

Step 2 (Define a Form Bean) -- Code Example

```
package coreservlets;
import org.apache.struts.action.*;

public class UserFormBean extends ActionForm {
  private String email = "Missing address";
  private String password = "Missing password";

  public String getEmail() { return(email); }

  public void setEmail(String email) {
    this.email = email;
  }

  public String getPassword() { return(password); }

  public void setPassword(String password) {
    this.password = password;
  }
}
```

Step 3 (Create Results Beans)

- **These are normal beans of the type used in regular MVC (i.e., implemented with RequestDispatcher)**
  - The form bean represents the *input* data: the data that came from the HTML form. In most applications, the more important type of data is the *result* data: the data created by the business logic to represent the results of the computation or database lookup.
  - Results beans need to have getter and setter methods like normal JavaBeans, but need not extend any particular class or be declared in struts-config.xml.
    - They are stored in request, session, or application scope with the setAttribute method of HttpServletRequest, HttpSession, or ServletContext, respectively.

Step 3, Bean Code Example

```
package coreservlets;

public class SuggestionBean {
  private String email;
  private String password;

  public SuggestionBean(String email, String password) {
    this.email = email;
    this.password = password;
  }

  public String getEmail() {
    return(email);
  }

  public String getPassword() {
    return(password);
  }
}
```

Step 3, Business Logic Code (To Build SuggestionBean)

```
package coreservlets;

public class SuggestionUtils {
  private static String[] suggestedAddresses =
    { "president@whitehouse.gov",
      "gates@microsoft.com",
      "palmisano@ibm.com",
      "ellison@oracle.com" };
  private static String chars =
    "abcdefghijklmnopqrstuvwxyz0123456789#@$%^&*?!";

  public static SuggestionBean getSuggestionBean() {
    String address = randomString(suggestedAddresses);
    String password = randomString(chars, 8);
    return(new SuggestionBean(address, password));
  }
  …
}
```

Step 4 (Define an Action Class to Handle Requests)

- **This example is similar to the previous one except that we do do not call request.getParameter explicitly.**
  - Instead, we extract the request parameters from the already populated form bean.
    - Specifically, we take the ActionForm argument supplied to execute, cast it to UserFormBean (our concrete class that extends ActionForm), and then call getter methods on that bean.
  - Also, we create a SuggestionBean and store it in request scope for later display in JSP.
    - This bean is the result of our business logic, and does not correspond to the incoming request parameters

Step 4 (Define an Action Class to Handle Requests) -- Code

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             ... request, ... response)
    throws Exception {
  UserFormBean userBean = (UserFormBean)form;
  String email = userBean.getEmail();
  String password = userBean.getPassword();
  if ((email == null) ||
      (email.trim().length() < 3) ||
      (email.indexOf("@") == -1)) {
    request.setAttribute("suggestionBean",
                         SuggestionUtils.getSuggestionBean());
    return(mapping.findForward("bad-address"));
  } else if ((password == null) ||
             (password.trim().length() < 6)) {
    request.setAttribute("suggestionBean",
                         SuggestionUtils.getSuggestionBean());
    return(mapping.findForward("bad-password"));
  } else {
    return(mapping.findForward("success"));
  }
}
```

Step 5 (Create Form that Invokes *blah*.do)

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>New Account Registration</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>New Account Registration</H1>
<FORM ACTION="register1.do"
      METHOD="POST">
  Email address: <INPUT TYPE="TEXT"
  NAME="email"><BR>
  Password: <INPUT TYPE="PASSWORD"
  NAME="password"><BR>
  <INPUT TYPE="SUBMIT" VALUE="Sign Me Up!">
</FORM>
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Alternatives for Beans

- **Use JSP scripting elements.**
  - This approach is out of the question; it is precisely what Struts is designed to avoid.
- **Use jsp:useBean and jsp:getProperty.**
  - This approach is possible, but these tags are a bit clumsy and verbose.
- **Use the JSTL c:out tag.**
  - This approach is not a bad idea, but it is hardly worth the bother to use JSTL just for this situation. So, unless you are using JSTL elsewhere in your application anyhow, don't bother with c:out.

Step 6 (Display Results in JSP) Alternatives for Beans

- **Use the JSP 2.0 expression language.**
  - This is perhaps the best option if the server supports JSP 2.0. In these examples, we will assume that the application needs to run on multiple servers, some of which support only JSP 1.2.
- **Use the Struts bean:write tag.**
  - This is by far the most common approach when using Struts. Note that, unlike c:out and the JSP 2.0 expression language, bean:write automatically filters special HTML characters, replacing < with &lt; and > with &gt;. You can disable this behavior by specifying

```
<bean:write name="beanName"
            property="beanProperty"
            filter="false">
```

  - So, in this example we use bean:write. Before we do so, however, we have to import the "bean" tag library as follows.

```
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
```

Step 6 (Display Results in JSP) First Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Email Address</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Email Address</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
The address
"<bean:write name="userFormBean" property="email"/>"
is not of the form username@hostname (e.g.,
<bean:write name="suggestionBean" property="email"/>).
<P>
Please <A HREF="register1.jsp">try again</A>.
</CENTER>
</BODY></HTML>
```

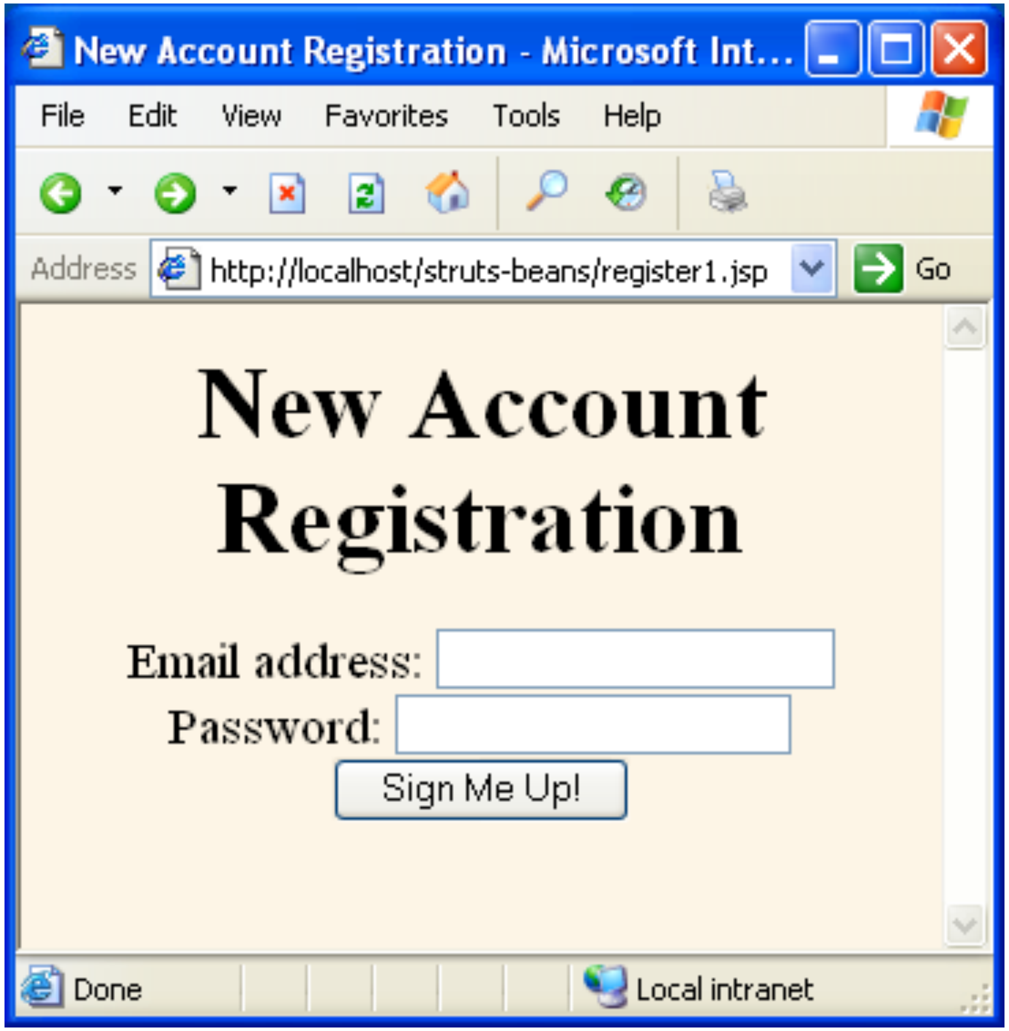
Step 6 (Display Results in JSP) Second Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Illegal Password</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Illegal Password</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
The password
"<bean:write name="userFormBean" property="password"/>"
is too short; it must contain at least six characters.
Here is a possible password:
<bean:write name="suggestionBean"
  property="password"/>.
<P>
Please <A HREF="register1.jsp">try again</A>.
</CENTER>
</BODY></HTML>
```

Step 6 (Display Results in JSP) Third Possible Page

```
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Success</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>You have registered successfully.</H1>
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
<UL>
  <LI>Email Address:
      <bean:write name="userFormBean" property="email"/>
  <LI>Password:
      <bean:write name="userFormBean"
  property="password"/>
</UL>
Congratulations
</CENTER>
</BODY></HTML>
```

Example 1: Results (Initial Form)


New Account Registration - Microsoft Int...
File Edit View Favorites Tools Help
Address
Go
Sign Me Up!
Done
Local intranet

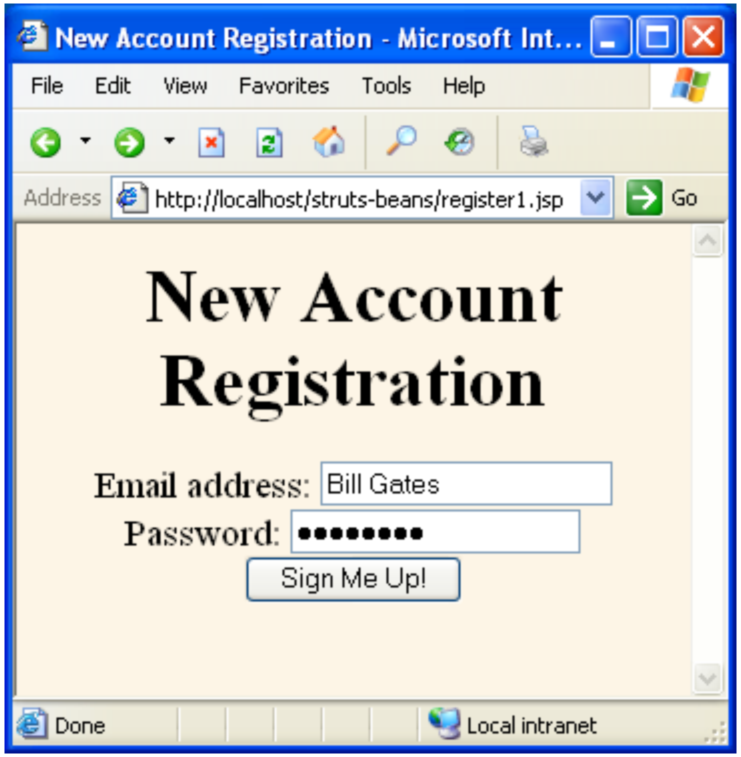
http://localhost/struts-beans/register1.jsp

New Account Registration

Email address:

Password:

Example 1: Results (Illegal Address)

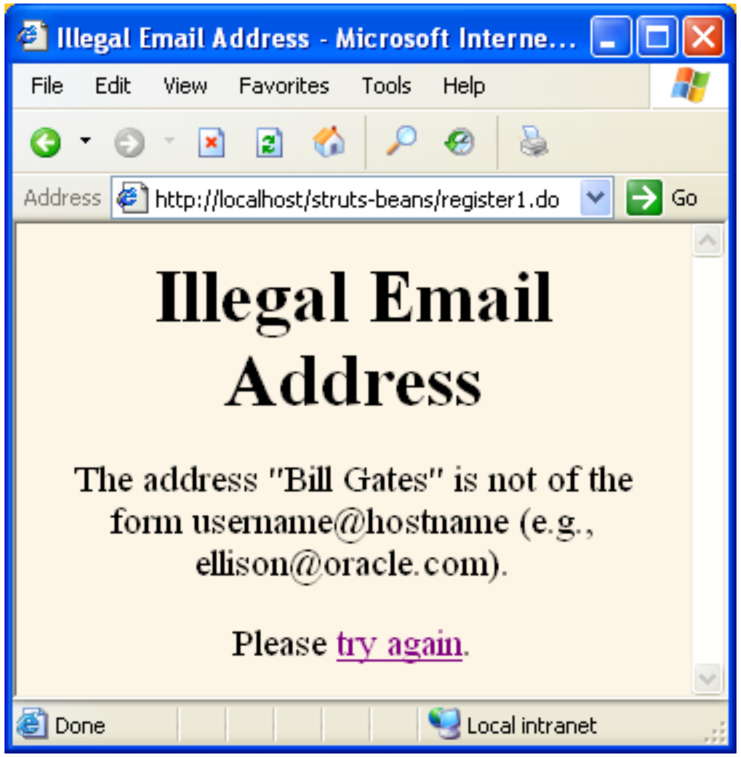

New Account Registration - Microsoft Int...
File Edit View Favorites Tools Help
Address http://localhost/struts-beans/register1.jsp
Go
Done
Local intranet

New Account Registration

Email address: Bill Gates

Password: ●●●●●●●●●

Sign Me Up!

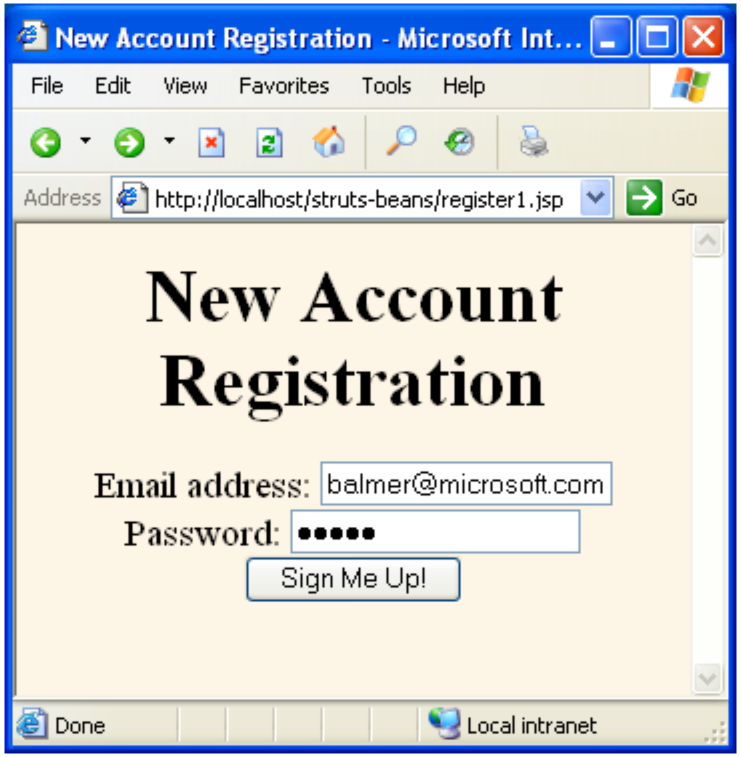

Illegal Email Address - Microsoft Interne...
File Edit View Favorites Tools Help
Address http://localhost/struts-beans/register1.do
Go
Done
Local intranet

Illegal Email Address

The address "Bill Gates" is not of the form username@hostname (e.g., ellison@oracle.com).

Please try again.

Example 1: Results (Illegal Password)

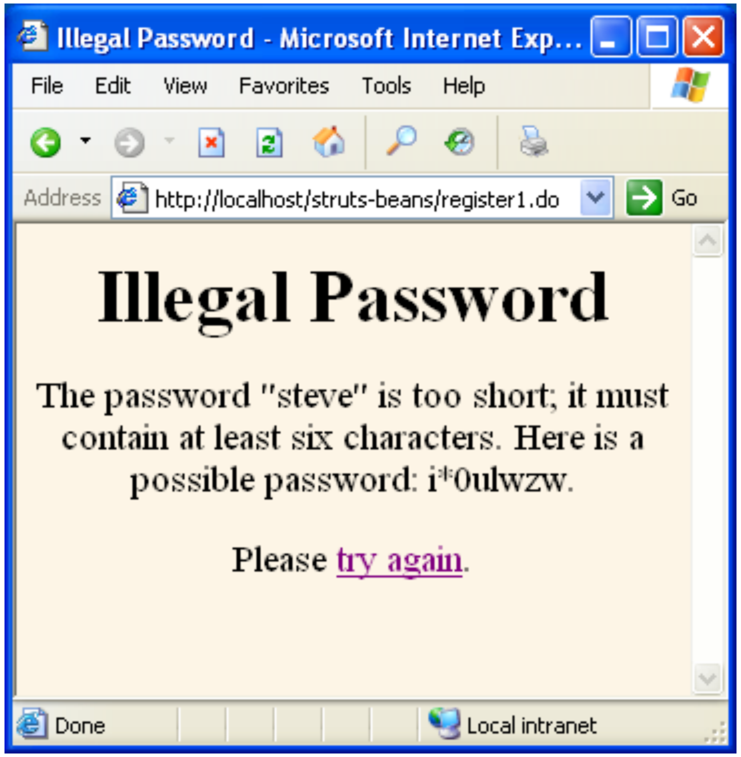

New Account Registration - Microsoft Int...

http://localhost/struts-beans/register1.jsp

New Account Registration

Email address: balmer@microsoft.com

Password: ●●●●●


Illegal Password - Microsoft Internet Exp...

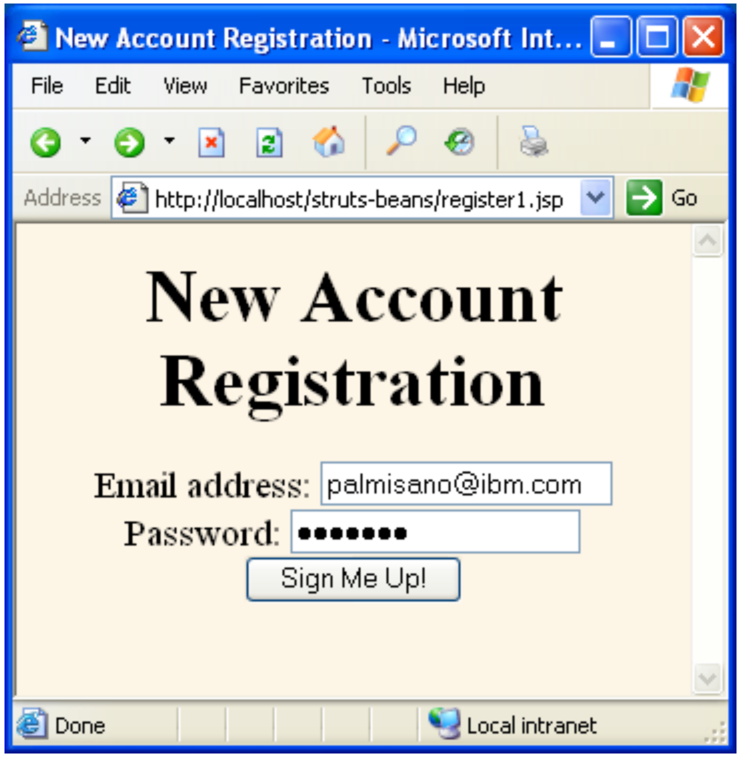
http://localhost/struts-beans/register1.do

Illegal Password

The password "steve" is too short; it must contain at least six characters. Here is a possible password: i*0ulwzw.

Please try again.

Example 1: Results (Legal Address and Password)

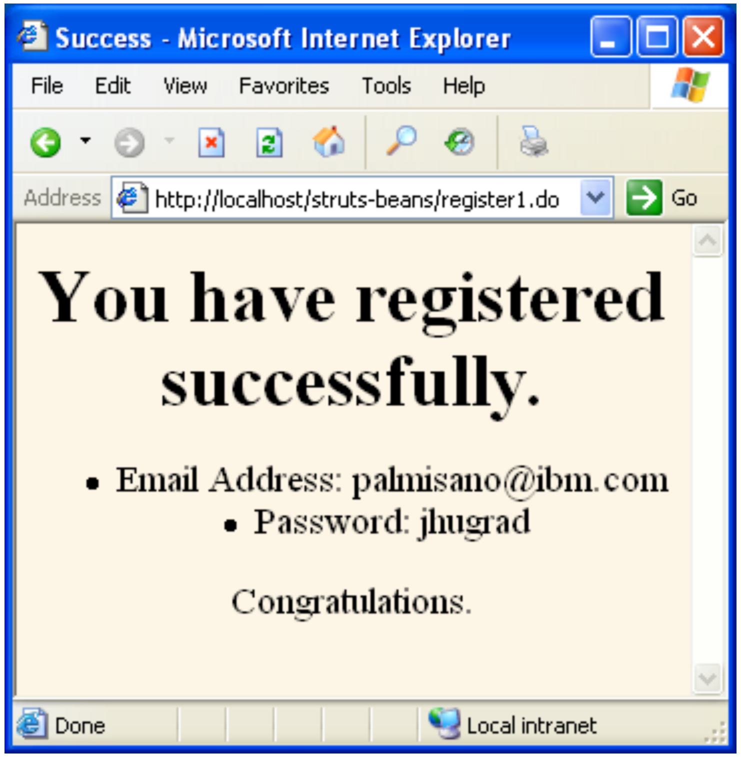

New Account Registration - Microsoft Int...
File Edit View Favorites Tools Help
Address http://localhost/struts-beans/register1.jsp
Go
Done
Local intranet

New Account Registration

Email address: palmisano@ibm.com

Password: ●●●●●●●●

Sign Me Up!


Success - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/struts-beans/register1.do
Go
Done
Local intranet

You have registered successfully.

- Email Address: palmisano@ibm.com
- Password: jhugrad

Congratulations.

Using the JSP 2.0 Expression Language with Struts

- **Pros**
  – The JSP 2.0 EL is shorter, clearer, and more powerful
- **Cons**
  – The bean:write tag filters HTML chars
  – There is no EL equivalent of bean:message
  – The EL is available only in JSP 2.0 compliant servers
    - E.g., Tomcat 5 or WebLogic 9, not Tomcat 4 or WebLogic 8.x
    - JSP 2.0 compliance list: http://theserverside.com/reviews/matrix.tss
- **To use the EL, use servlet 2.4 version of web.xml:**
  – You also must remove the taglib entries

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
         version="2.4">

  …
</web-app>
```

Struts Example: Confirmation Page (Classic Style)

```
…
Congratulations. You are now signed up for the
Single Provider of Alert Memos network!
<%@ taglib uri="http://struts.apache.org/tags-bean"
           prefix="bean" %>
<UL>
  <LI>First name:
  <bean:write name="contactFormBean"
   property="firstName"/>
  <LI>Last name:
  <bean:write name="contactFormBean" property="lastName"/>
  <LI>Email address:
  <bean:write name="contactFormBean" property="email"/>
  <LI>Fax number:
  <bean:write name="contactFormBean"
   property="faxNumber"/>
</UL>
…
```

Struts Example: Confirmation Page (JSP 2.0 Style)

```
…
Congratulations. You are now signed up for the
Single Provider of Alert Memos network!
<UL>
  <LI>First name: ${contactFormBean.firstName}
  <LI>Last name: ${contactFormBean.lastName}
  <LI>Email address: ${contactFormBean.email}
  <LI>Fax number: ${contactFormBean.faxNumber}
</UL>
…
```

Final exam review

- **Date & time: 22/06/2019: Duration: 120 mins**
- **Content:**
  - Answer the questions for both theory and practice. Develop a web-app for an eCommerce site that cover the following requirements: user management, database administration, interactive process, etc.
- **Web design**
  - HTML
  - CSS
  - Java script
- **Web program**
  - JSP (Data connection)
  - Servlet
  - Java bean
  - XML & Ajax
- **Web architecture**
  - MVC
  - Struts framework