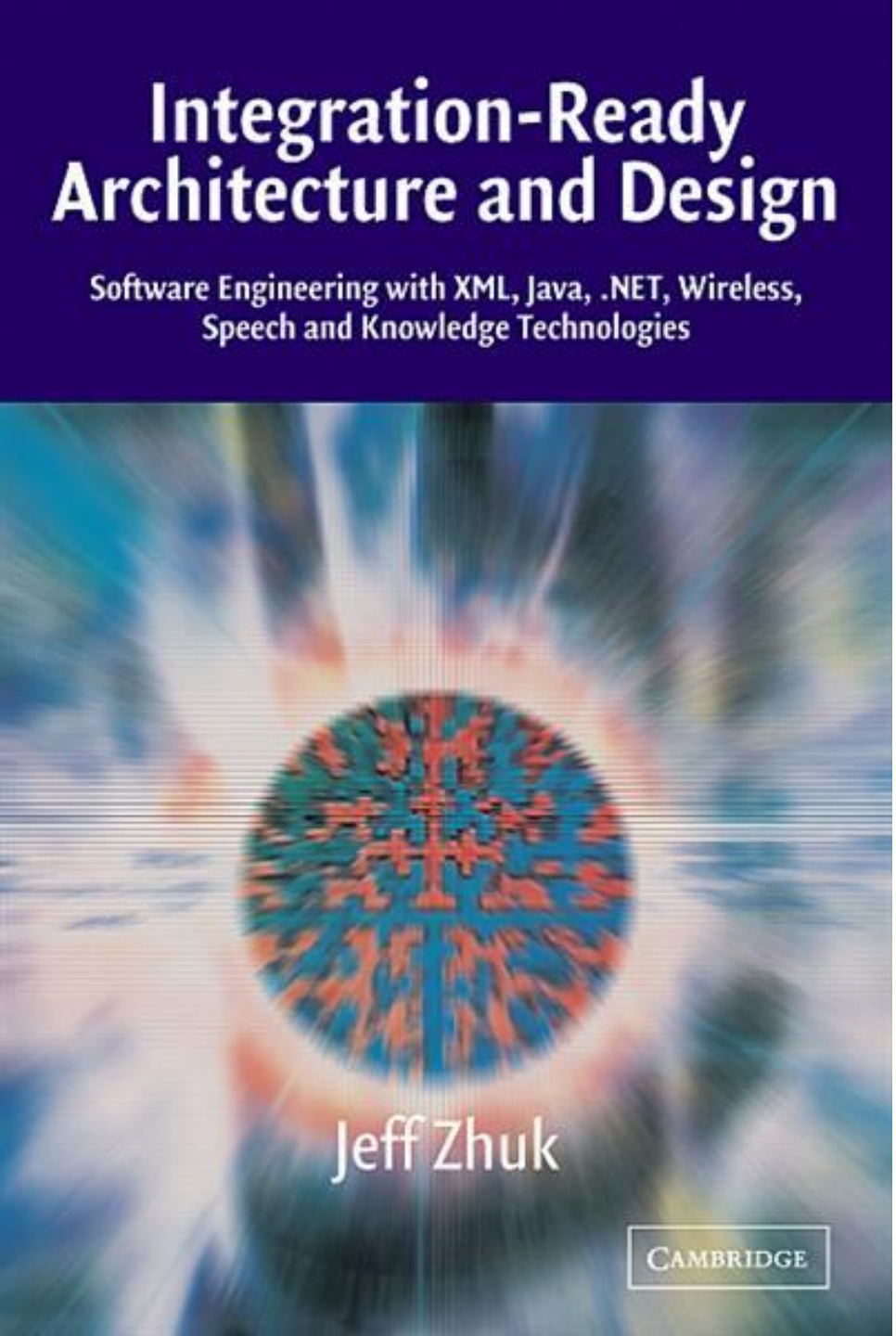


Creating Web Application with BEA WebLogic Workshop

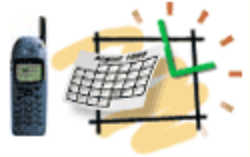
Jeff (Yefim) Zhuk

author of the book

"Integration-Ready Architecture and Design"
by
Cambridge University Press

JavaSchool.com

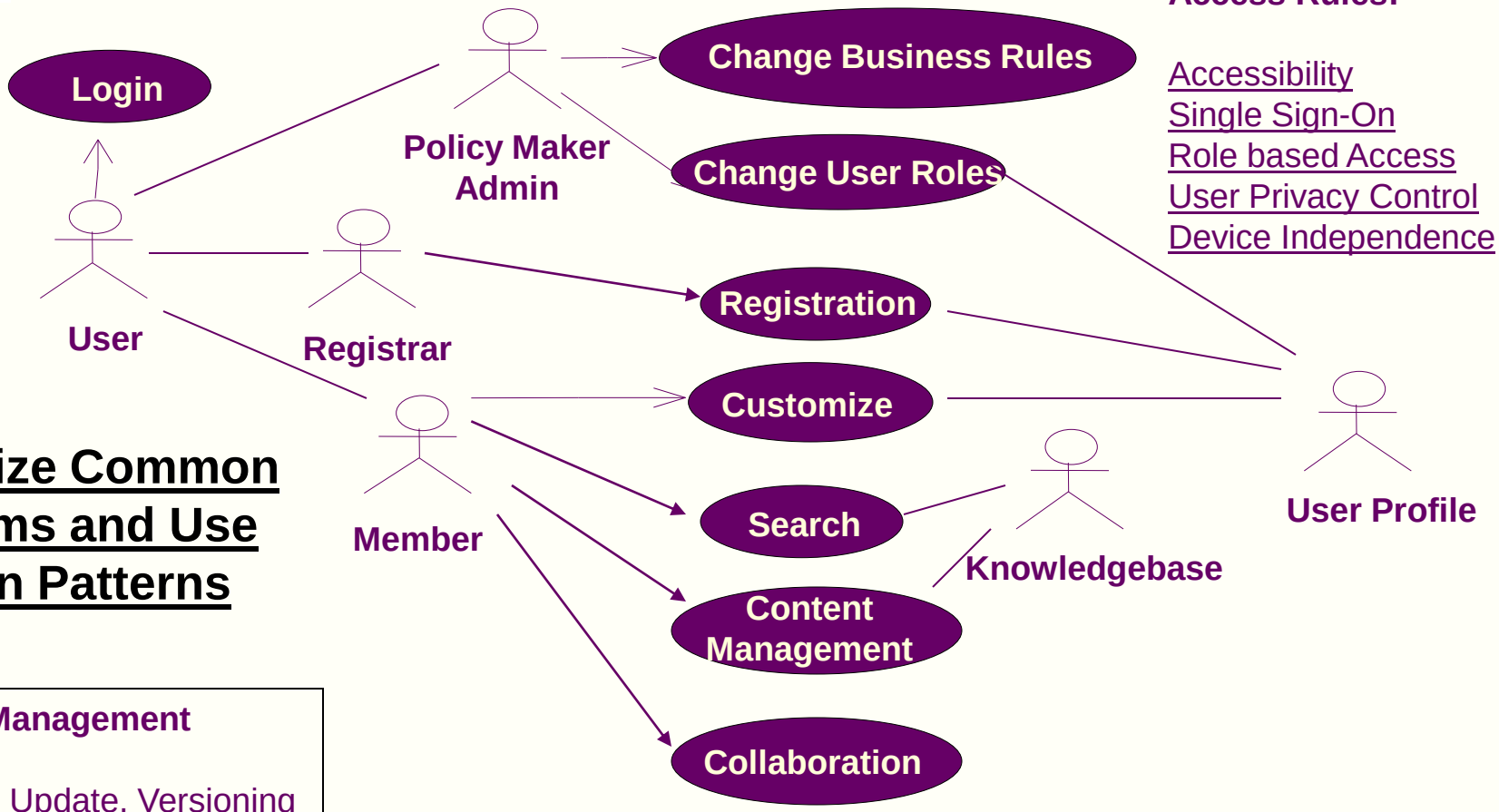
Software Engineering With XML, Java, .NET, Wireless, Speech and Knowledge Technologies



Creating Web Application with BEA WebLogic Workshop

- Reusable solutions for Data Intensive Web Applications
- Using BEA WebLogic Workshop 8.1
  - Use Page Flow Facilities
  - Provide Java code in the LoginController and Login Bean classes
  - Add configuration facilities
  - Build and Run

Common Use Cases



Access Rules:

Accessibility
Single Sign-On
Role based Access
User Privacy Control
Device Independence

Recognize Common Problems and Use Design Patterns

Content Management

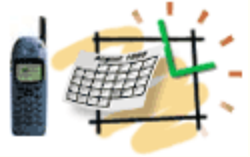
Authoring, Update, Versioning
Forms, Permits, Applications
Scheduling events/facilities
Workflow Routing
Planning and Approval
Document/Photo Imaging
Task Tracking and Reporting
Data and Service Evaluation

Collaboration

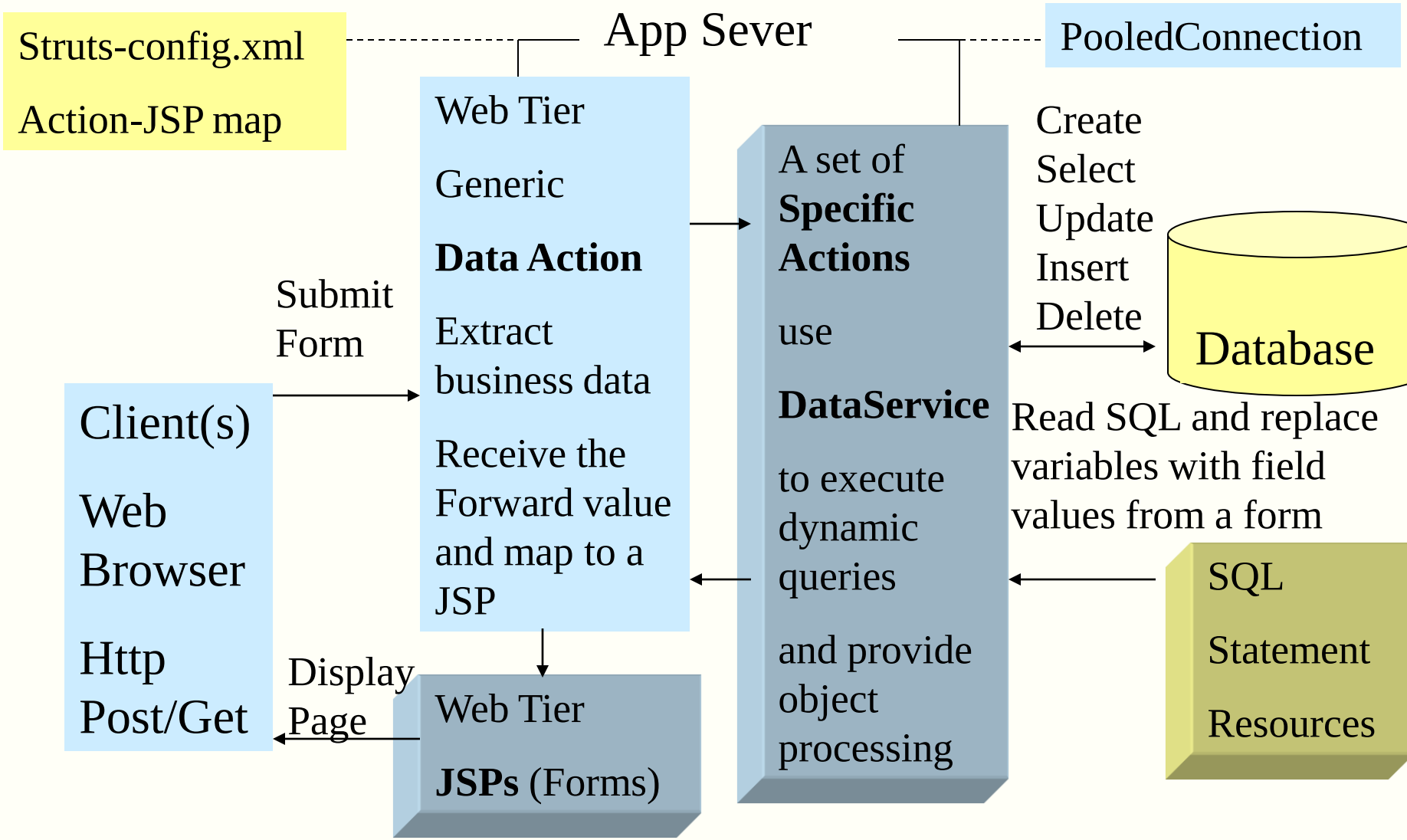
Email
Conferencing
Instant Messaging
Privilege-based Data/Service Sharing

Search

Search for Data and Services
GIS (Maps and Routes)
Linking Related Cases
Content-based Subscription

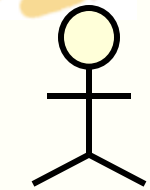


Data Intensive Web Application



The DataAction alone with the DataService implement common functions and provide generic application behavior.

Sequence Diagram



Client

DataAction | **Specific Service Action** | **DataService**

Events/Operations

Http Post/Get

Extract data from the page and call a proper service

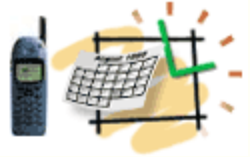
Request SQL execution with SQL statement name and map of parameters

Generic data processing: Retrieve SQL from resources; Replace run-time variables; Execute SQL, process ResultSet (if any) and return a list of object-beans

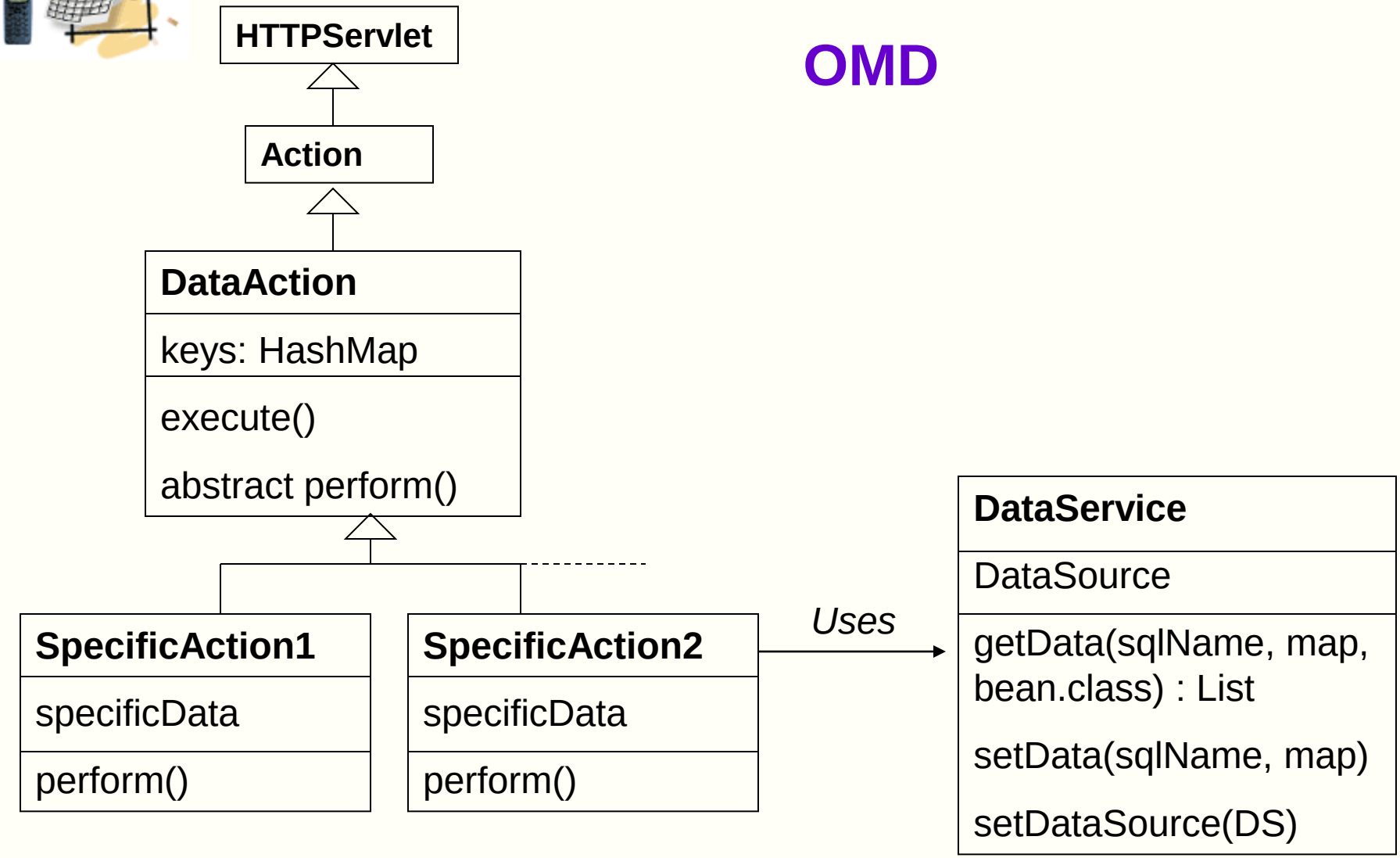
Specific processing; Work with objects and prepare data for JSP

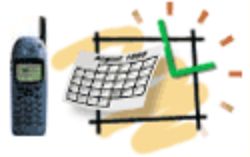
Pass data saved in data beans, Session, or Request objects to JSP

Map the Forward value to a proper JSP (via Struts-config.xml) and display the next JSP page



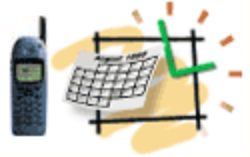
OMD





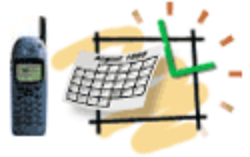
Generic DataAction Code

```
public class DataAction extends Action {
   private static Logger log = Logger.getLogger("DataAction");
   protected HashMap keys;
   public ActionForward execute(ActionMapping map, ….) {
      // generic code in the execute() method
      String forwardTo = null;
      keys = prepareKeys(request); // capture form and other key data
      try {    // perform specific action implemented in a subclass
          forwardTo = perform(request);
      } catch(Exception e) {
          log.severe(getClass().getName() + " e="+e);
          // prepare message for a generic error page
          request.setAttribute("msg", e.getMessage());
          throw e;
      }
      forward = map.findForward(forwardTo); // ActionMapping map
      return (forward);
   }
   public String perform(HttpServletRequest request) throws Exception {
      return null;
   }
}
```

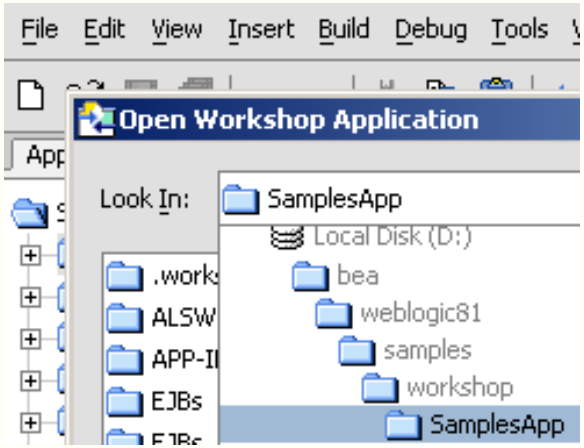


Specific Action Implementation

```
public class LoginAction extends DataAction {

    public String perform(HttpServletRequest request) throws Exception {
          ……
        List beans = DataService.getData("getLogin", keys, LoginBean.class);
        if(beans.size() == 1) { // SUCCESS!
            //Create instance of SessionBean and bind it to session
            SessionBean sessionBean = new SessionBean();
            LoginBean bean = (LoginBean) beans.get(0);
            sessionBean.setUserName(bean.getLoginName());
            session.setAttribute("sessionBean", sessionBean);
            return "success";
        }
        …….
        return "failure";
    }
```



Start a New Project in SampleApp area



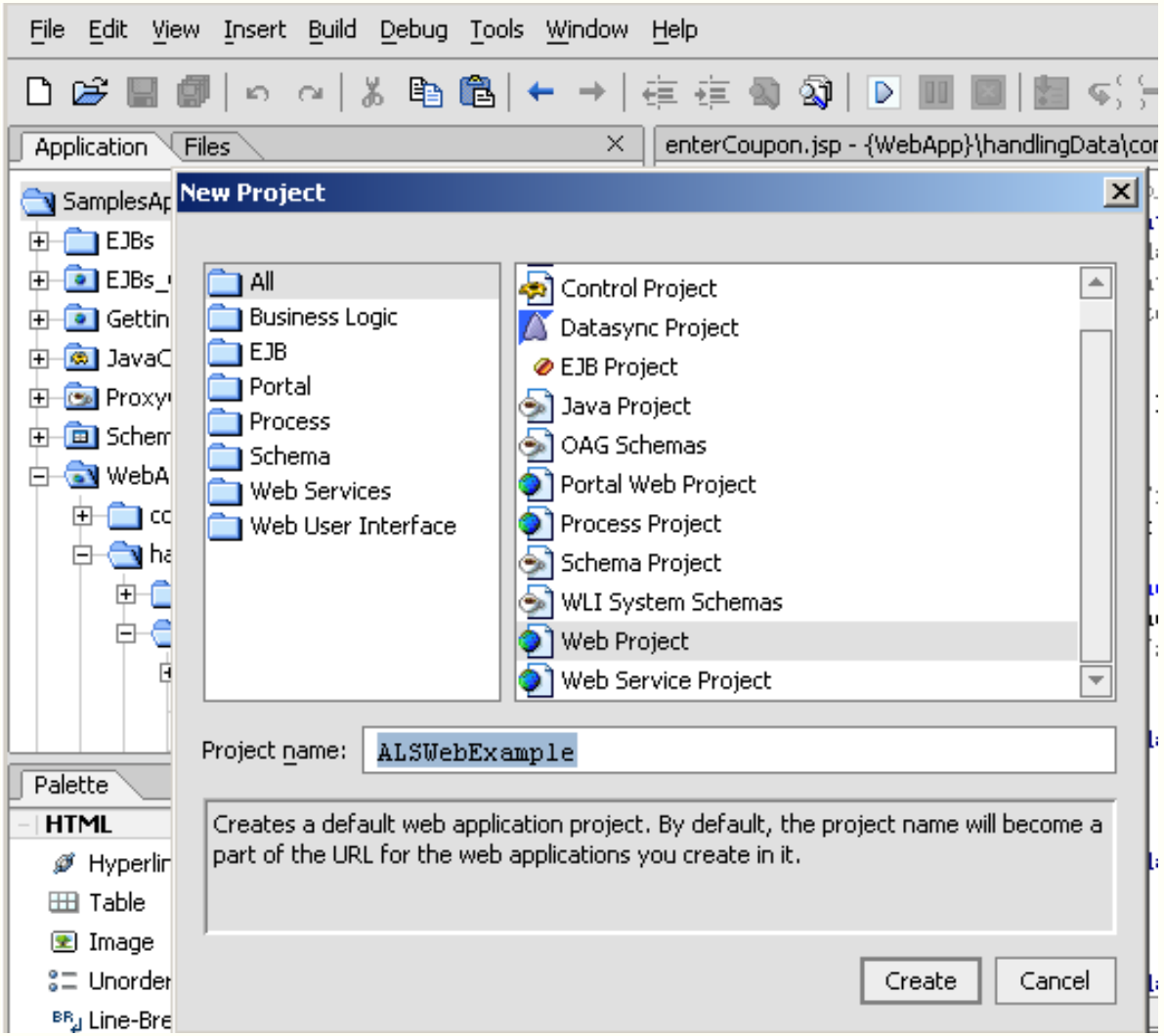
File – Open –

Open Application:

/bea/weblogic81/samples/workshop/SampleApp

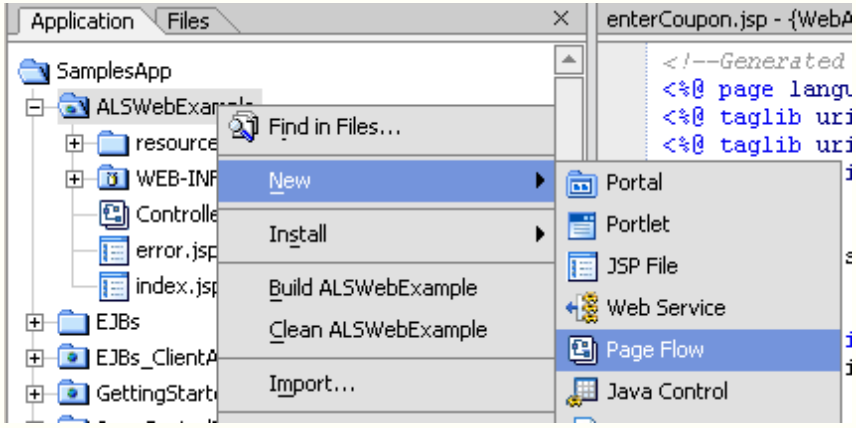
Then right-mouse-click on the SampleApp – New – Project

Enter: “WebExample”

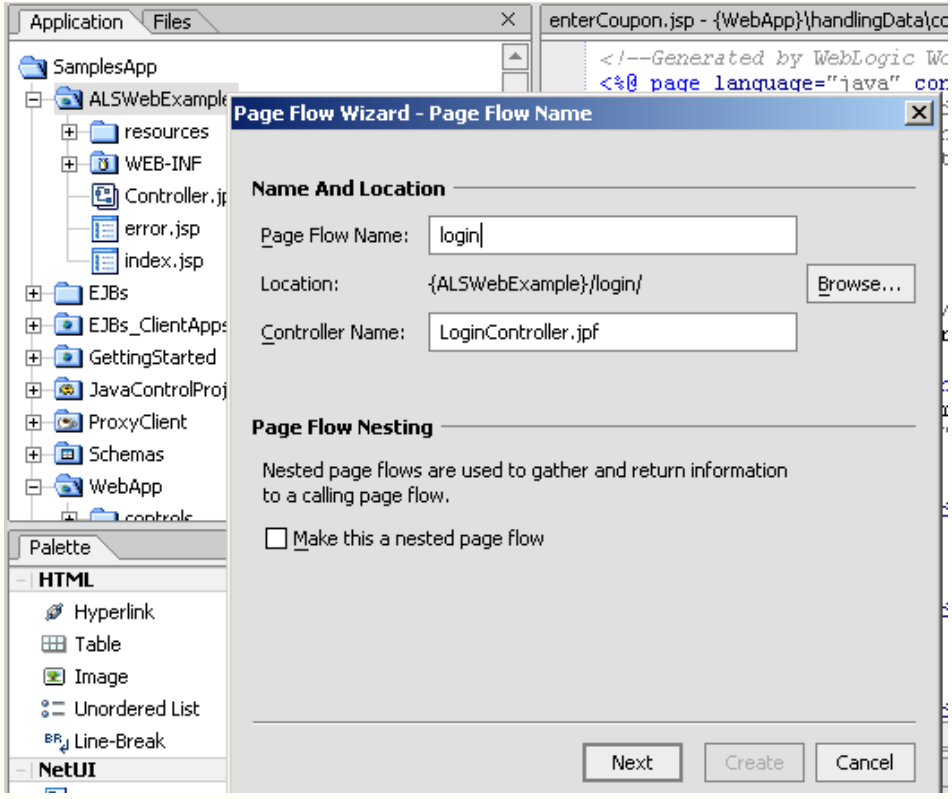


Start a New Page Flow

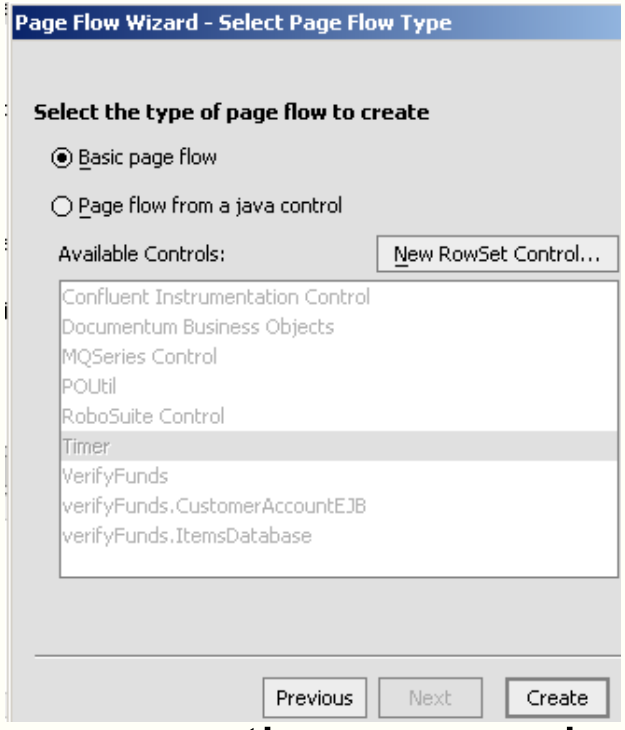
1



2



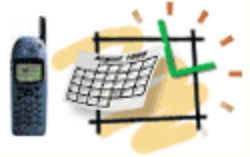
3



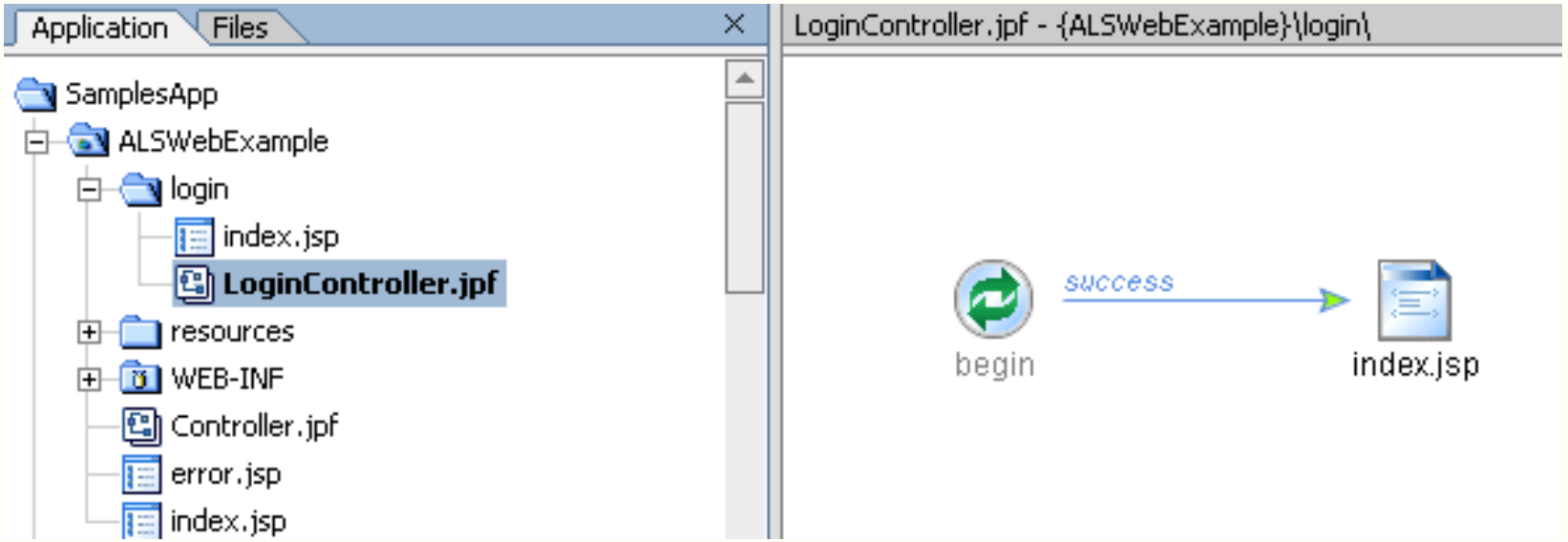
One – two – three …and bingo,

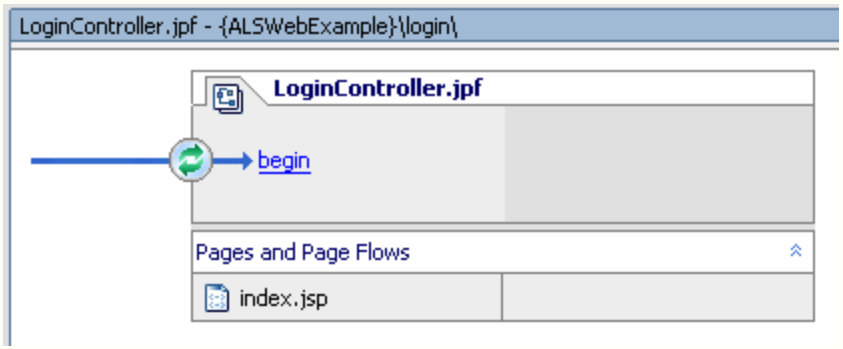
You created a starting point!

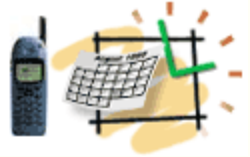
Do not forget to name this page flow in the step 2.



Flow View and Action View



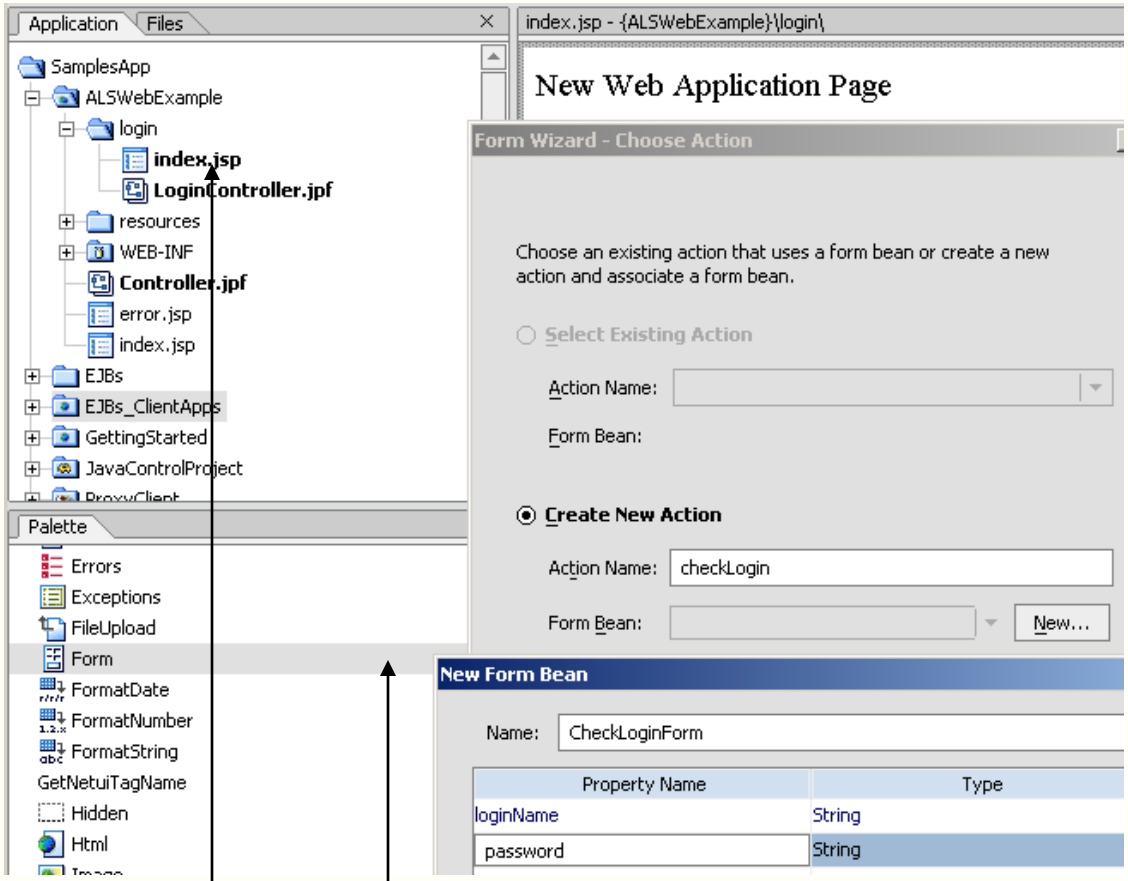




Source View

```
/**
 * @jpf:controller
 * @jpf:view-properties view-properties::
 * <view-properties>
* ::
 */
public class LoginController extends PageFlowController
{

/**
     * This method represents the point of entry into the pageflow
     * @jpf:action
     * @jpf:forward name="success" path="index.jsp"
     */
    protected Forward begin()
    {
        return new Forward("success");
    }
}
```

Create a Form and a FormBean



1. Double-click on the “index.jsp” file

Set the Design View and find the Form in the Palette on the left pane.

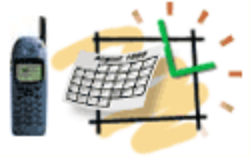
Drag-n-drop the form in the page on the right

2. Provide the action name for the form as “checkLogin” and press the NEW button to create a new form.

WebLogic creates CheckLoginForm bean that will store properties (values) of the new form.

Provide two names for the properties: “loginName” and “password” and press “OK”. You created a form and a supporting form bean.

3. Double-click on the LoginController and check Flow View, Action View, and Source View.



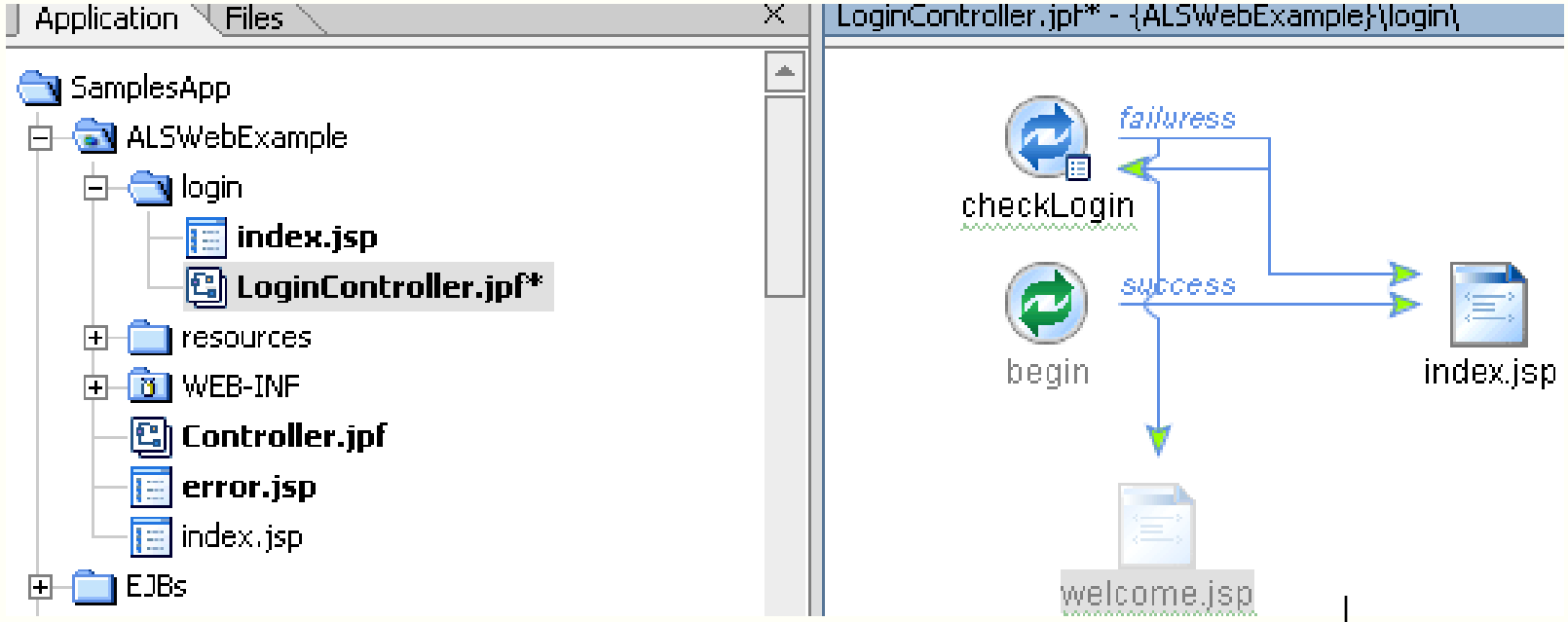
Customize Your Action

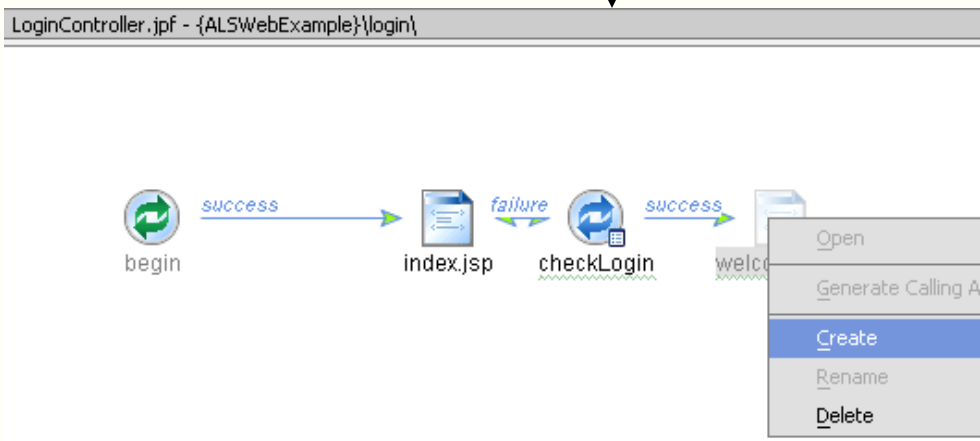
You will instruct the checkLogin() method to pass control to the “welcome.jsp” page in the case of success or back to the “index.jsp” in the case of failure.

Enter two new lines in the header of the checkLogin() method:

```
/**
 * @jpf:action
 * @jpf:forward name=“success" path="welcome.jsp"
 * @jpf:forward name=“failure" path="index.jsp"
 */
protected Forward checkLogin(CheckLoginForm form)
{
    return new Forward("success");
}
```

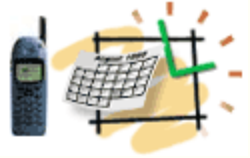
Re-arrange the Flow View



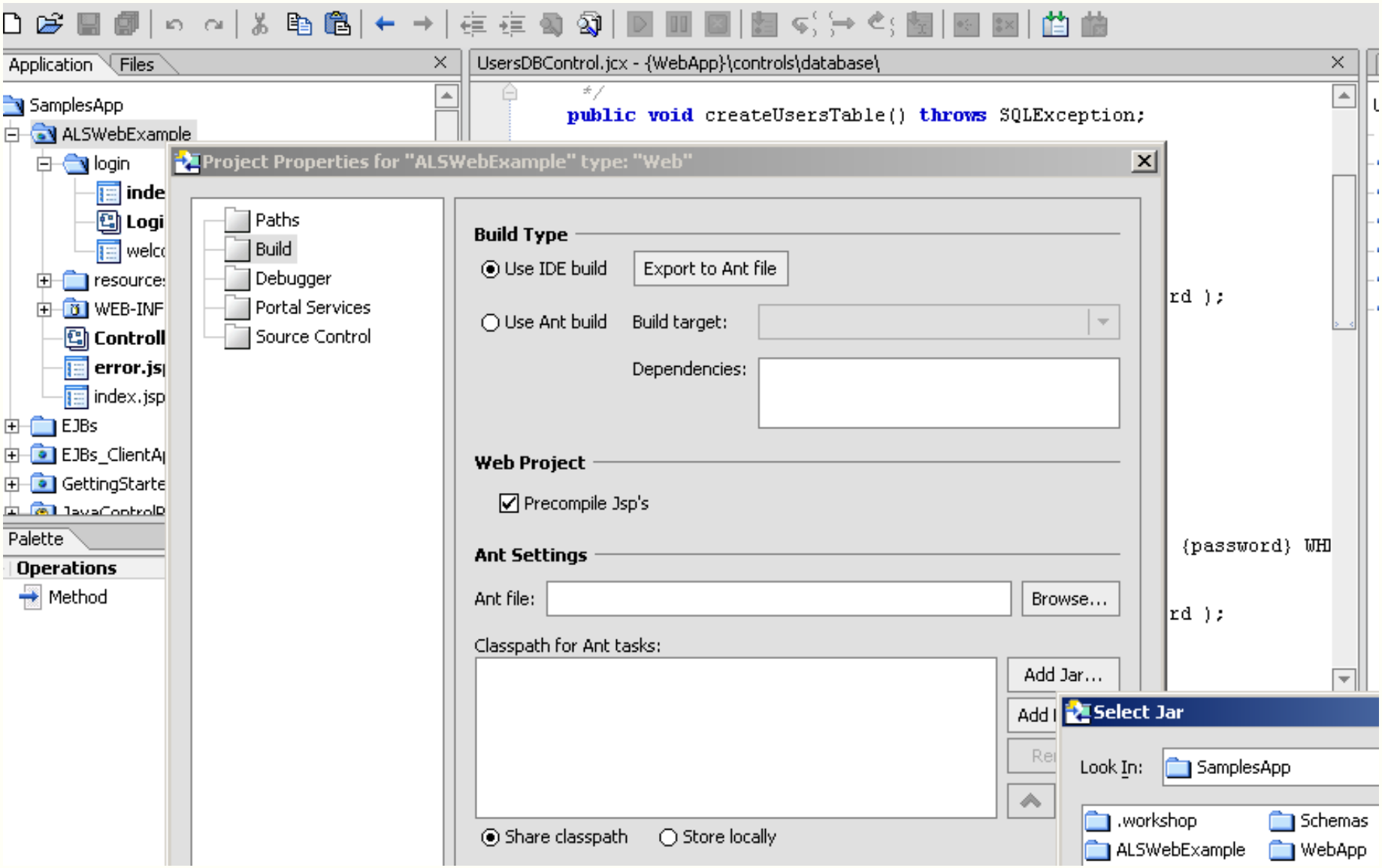


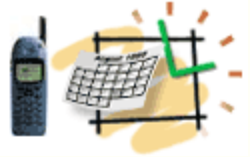
Drag-n-drop the checkLogin and the welcome.jsp icons to clarify the Flow View.

Then with right-mouse click create the welcome.jsp page.

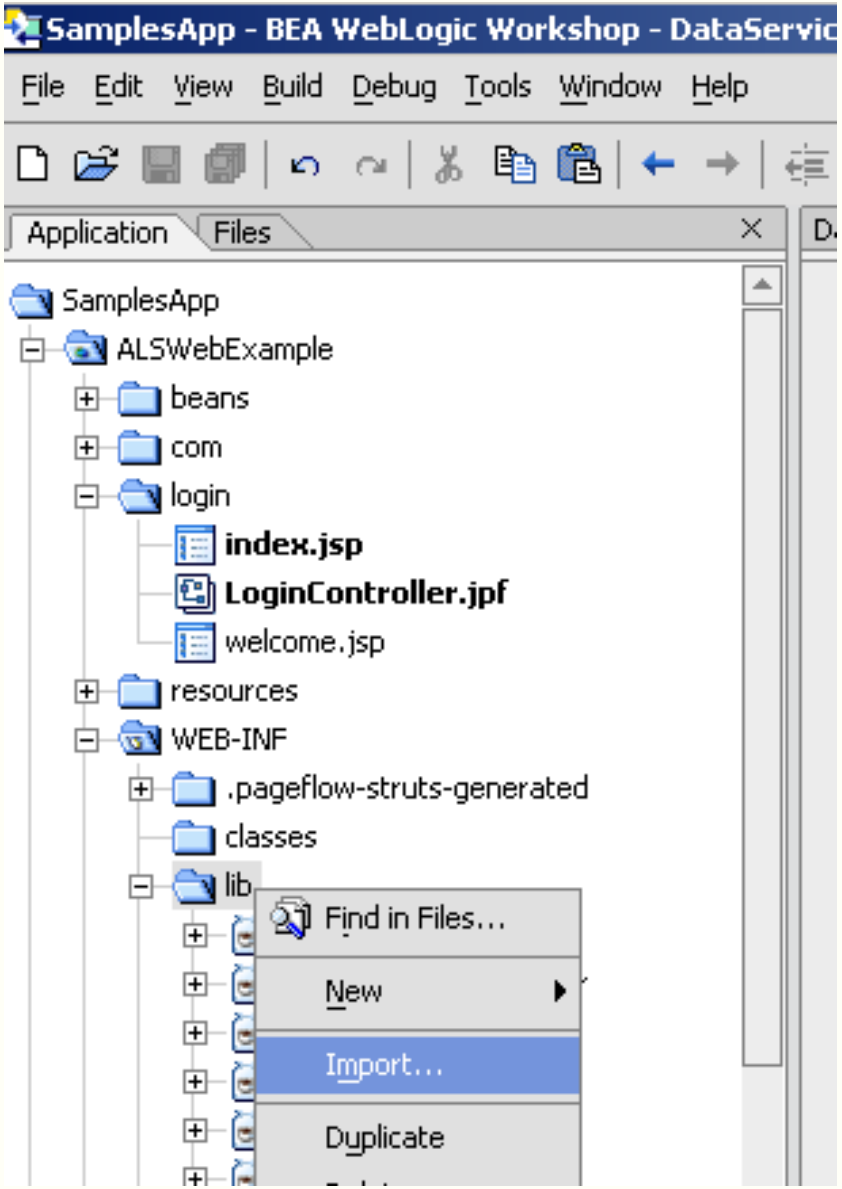


Add Jar files to the Application Build Facilities

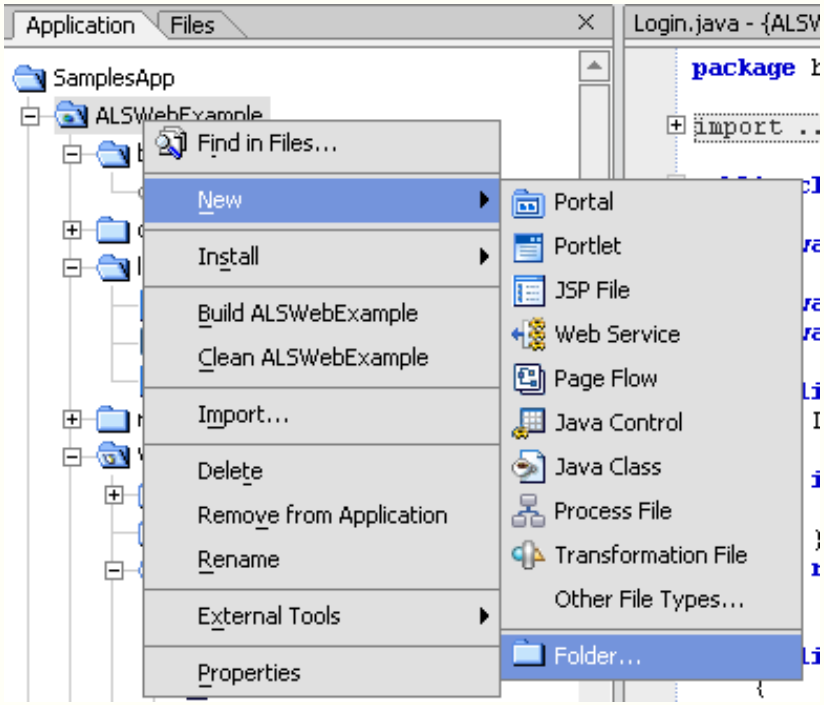




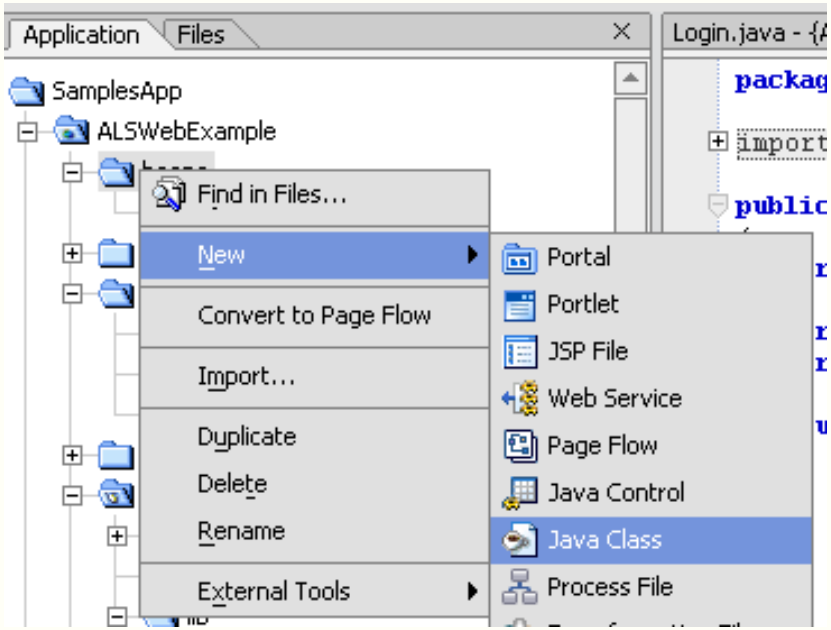
Import Extra Libraries to WEB-INF/lib

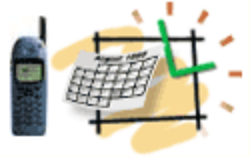


Create the Login Bean Class



1. Right-mouse click on the project and create NEW – FOLDER “beans”
2. Right-mouse click on the “beans” folder and create NEW – JAVA CLASS
3. Name it “Login”



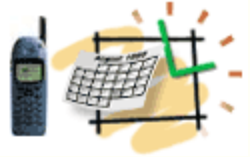


LoginBean Class Example

```
package beans;

import java.util.HashMap;
import java.util.logging.Logger;
/**
 * The LoginBean class matches the record selected by the getLogin.sql
 */

public class LoginBean
{
    private String loginName;
    private String password;

    // add getter and setter methods for data members above !!!
    public String getLoginName() {
        return loginName;
   }
   // TODO more getter and setter methods
}
```

The CheckLogin() Method in the LoginController

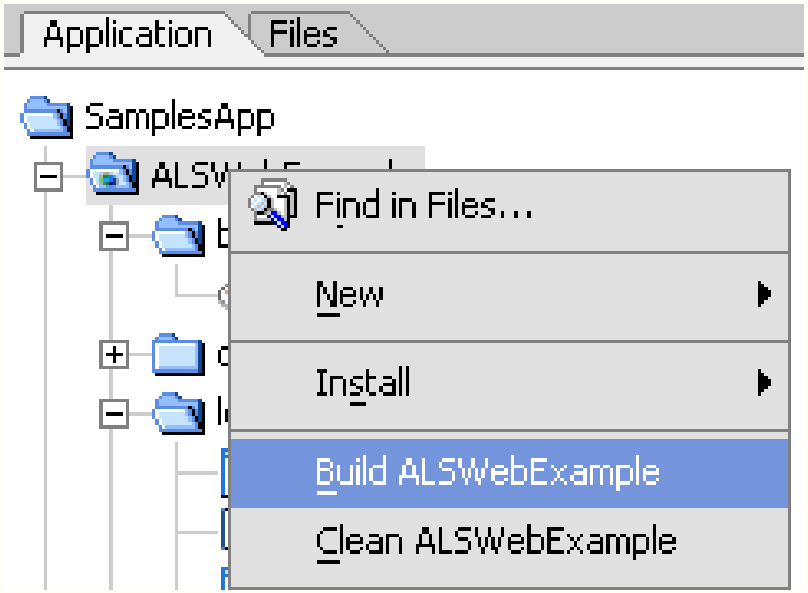
```
/**
 * @jpf:action
 * @jpf:forward name="success" path="welcome.jsp"
 * @jpf:forward name="failure" path="index.jsp"
 * Note, that the data access is performed with the DataService.getData() that
 * uses the SQL statement-file “getLogin.sql” stored in the “sqlLocation”
 */
 protected Forward checkLogin(CheckLoginForm form)
 {
    HashMap keys = new HashMap();
    keys.put("loginName", form.getLoginName());
    keys.put("password", form.getPassword());
    List logins = DataService.getData("getLogin", keys, LoginBean.class);
    String result = "failure";
    if(logins != null && logins.size() == 1) {
        result = "success";
    }
    getSession().setAttribute("result", result);
    return new Forward(result);
}
```

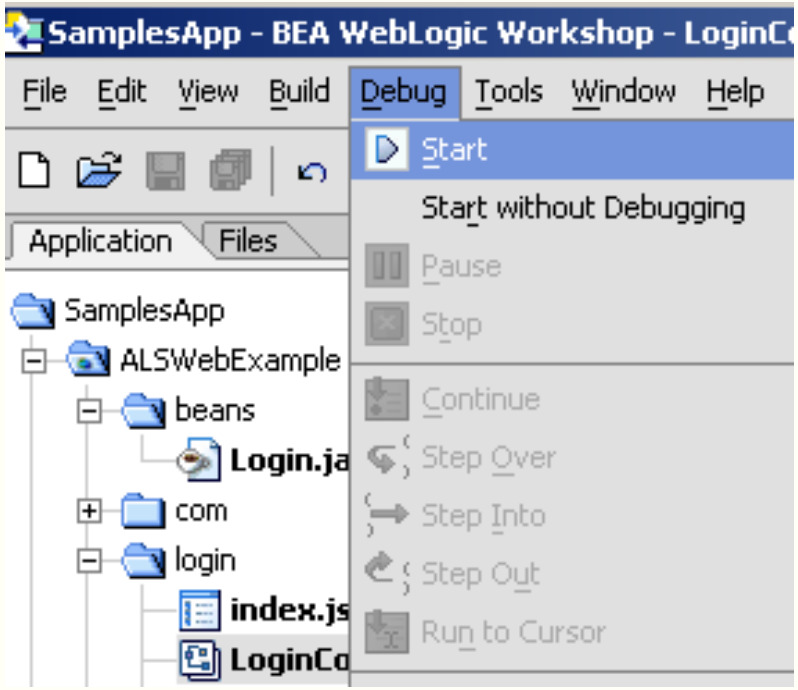


Example of Using the Session in the JSP
Add the scriplet to the index.jsp

```
 <body>
        <p>
<%
String result = (String) session.getAttribute("result");
if(result != null && result.equals("failure")) {
    out.println("Please try again");
} else {
    out.println("Enter your login name and password");
}
%>

        <netui:form action="checkLogin">
            <table>

                <tr>
                    <td colspan="2" align="center">
                    </td>
                </tr>
                <tr valign="top">
                    <td>Login Name:</td>
```

Build and Debug





1. Right-mouse click on the project and select the Build “your project” option

Watch for the output in the Build console at the bottom – right

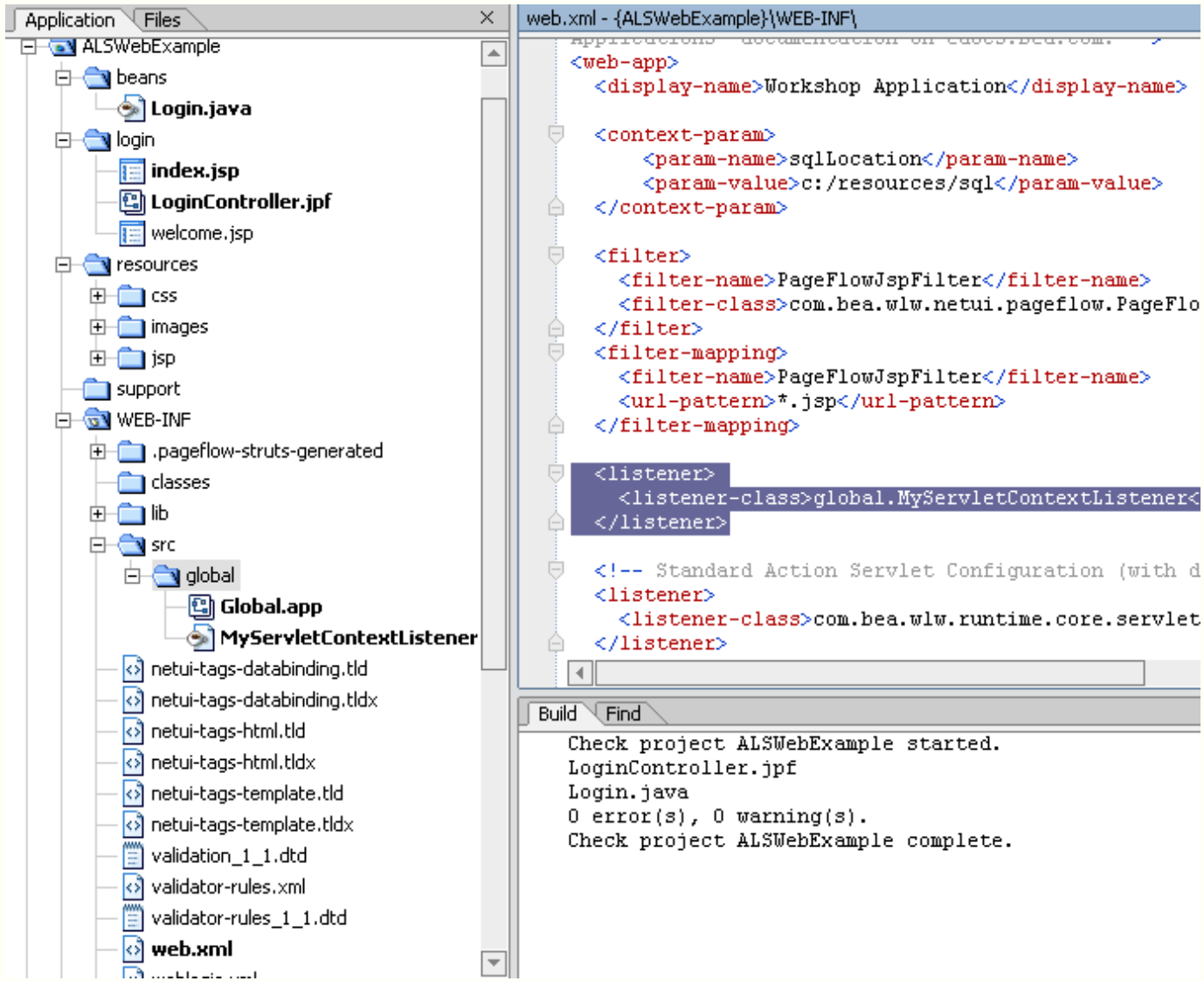
2. Right-click on the Controller and select DEBUG – START from the menu

See logger output in the main console (find it in the MS DOS window)

ServletContextListener Reads Config

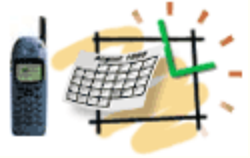
```
public class MyServletContextListener implements ServletContextListener {
   public void contextInitialized(ServletContextEvent sce) {
      ServletContext servletContext = sce.getServletContext();
      String sqlLocation = servletContext.getInitParameter("sqlLocation");
      String jndiName = "java:"+servletContext.getInitParameter("DataSource");
      …… InitialContext context = new InitialContext();
      …… DataSource dataSource = (DataSource)context.lookup(jndiName);
      com.its.util.DataService.setSqlLocation(sqlLocation);

      com.its.util.DataService.setDataSource(dataSource);

  }

  public void contextDestroyed(ServletContextEvent sce) { }

} ---------------------------------------------- WEB.XML --------------------------------
  <listener>
    <listener-class>com.its.actions.MyServletContextListener</listener-class>
   </listener>
   <context-param>
    <param-name>sqlLocation</param-name>
    <param-value>sql</param-value>
  </context-param>
  <context-param>
     <param-name>DataSource</param-name>
     <param-value>SpecificDataSourceNameProvidedInJBoss_ds_file</param-value>
   </context-param>
```

Configure the Application via web.xml



Find the configuration file wml.xml in the WEB-INF directory, edit the <listener> entry.

Then add the <context-param> entries with DataSource, etc.



Build your Web Application using the last section material

- Install WebLogic Workshop
- Build the Login page according to the section material
- Add Product and Registration pages
- Prepare SQL statements for existing data and store as files

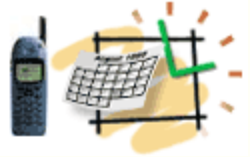
in the “sqlLocation” that is set in the web.xml and MyServletContectListener

(No SQL is needed if application creates data from scratch)

- Create bean classes like the LoginBean that support data structures used with data access and in the forms.
- Create the LoginController with the checkLogin() method.

Note, that the data access in the controller is performed with the DataService.getData() method that uses the SQL statement-file “getLogin.sql” stored in the “sqlLocation”

Prepare HashMap to replace SQL <<keys>> with run-time values



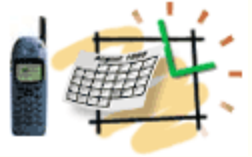
DataService API

Include the library "com.its.util.jar" in the CLASSPATH and import com.its.util.DataService

@param directoryName to set the location where your SQL statement files are stored
public static void setSqlLocation(String directoryName)
•
@param driverName your jdbc driver name
public static void setDriverName(String driverName)
•
@param set Connection URL for database access
public static void setConURL(String connectionURL)
•
// execute insert/delete/update SQL statements stored in the "sqlLocation"
@ param sqlStatementName for example "getLogin" stored as the "getLogin.sql"
@ param map of key-values to replace SQL <<keys>> with run-time values
@ return numberOfRowsEffected
public static int setData(String sqlStatementName, HashMap map)
•
@ param sqlStatementName for example "getLogin" stored as the "getLogin.sql"
@ param map of key-values to replace SQL <<keys>> with run-time values
@ param beanClass (e.g. LoginBean.class) supports records retrieved by the SQL statement
@ return list of objects of the beanClass
public static List getData(String sqlStatementName, HashMap map, Class beanClass)
•
@param dataSource your DataSource specified in JNDI context
public static void setDataSource(DataSource dataSource)

DataService Usage Example

```
// set DataService env variables in the initialization procedure
(MyServletContextListener)
DataService.setDataSource(DataSource ds);  // use JNDI to get DataSource
DataService.setSqlLocation(sqlLocation); //dsName and sqlLocation from web.xml

// This code belongs to the LoginController in the WebLogic example by JZ
HashMap map = new HashMap(); // map to replace run-time variables in SQL
map.put("loginName", form.getLoginName());
map.put("password", form.getPassword());
List users = DataService.getData("getLogin", map, LoginBean.class);
If(users != null && users.size() == 1) { // success
      // successful login
}

// Prepare SQL statements and store them in the “sqlLocation”
// /WEB-INF/sql/getLogin.sql – select statement to check login
// Note that run-time variable names are in the tags << .. >> or follow the “:” character
select username, password from LoginTable
where username = '<<loginName>>' and password = '<<password>>'

// Alternative to specify replacement keys is the ":" character
select username, password from LoginTable where username = ':loginName' and
password = ':password'
```



More DataService API

// If application creates data from scratch – less or no SQL is needed
// SQL is created on-the-fly by DataService implementation

boolean createTable(String tableName, Class class)

Example:
DataService.createTable(“LoginTable”, LoginBean.class);

int insert(String tableName, Object[] objects)

Example:
LoginBean[] logins; // array of beans populated in an action
int nRows = DataService.insert(“LoginBean”, logins);