Contents

Unauthorized reproduction or distribution is prohibited
Evaluation Copy

Chapter 1 - Course Introduction 7

Course Objectives 8
Course Overview 10
Using the Workbook 11
Suggested References 12

Chapter 2 - Web Applications and MVC 15

Web Applications 16
JSPs and Servlets 18
Model-View-Controller 20
Model 2 Architecture 22
The WAR File 24
web.xml 26
Building the WAR 28
Deploying the WAR 30
Labs 32

Chapter 3 - JavaServer Pages 35

Introduction to JSP 36
JSP Syntax 38
JSP Scripting Elements 40
Request and Response Implicit Objects 42
page Directive 44
Error Handling 46
The include directive 48
include and forward Actions 50
Labs 52

Chapter 4 - Java Servlets 55

HTTP Requests 56
HttpServlet 58
Servlet Lifecycle 60
@WebServlet Annotation 62

RequestDispatcher 64
HttpSession 66
ServletContext 68
Servlet Filters 70
JSP vs. Servlet 72
Labs 74

Chapter 5 - JavaBeans 77

What is a JavaBean? 78
Rules 80
Properties 82
Using JavaBeans in JSPs 84
Properties and Forms 86
Data Access Objects 88
Resource Reference 90
Bean Scopes in Servlets 92
Bean Scopes in JSPs 94
Labs 96

Chapter 6 - JSP Expression Language 99

JSP Expression Language 100
Literals 102
Variables 104
The . and [ ] Operators 106
Other Operators 108
Implicit Objects 110
Labs 112

Chapter 7 - Introduction to JSTL 115

What is JSTL? 116
Core Tags — Conditionals 118
Core Tags — Iteration and Import 120
Variables, Output, and Exceptions 122
XML Manipulation Tags 124
Internationalization Tags 126
SQL Tags 128
Labs 130

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

 © 2013 ITCourseware, LLC

Chapter 8 - Security 133

Concepts 134
Constraints 136
Roles 138
login-config 140
BASIC Authentication 142
FORM Authentication 144
Login and Error Pages 146
Labs 148

Appendix A - Tag Libraries 151

Custom Tags 152
Using Custom Tags 154
Defining Tags 156
Tags with Attributes 158
Fragments and Variables 160
Packaging Tag Files 162
Labs 164

Appendix B - Ant 167

What Is Ant? 168
build.xml 170
Tasks 172
Properties and Property Files 174
Managing Files and Directories 176
Filesets 178
Java Tasks 180
Creating Java Archives 182
Specifying Paths 184
Miscellaneous Tasks 186

Solutions 189

Index 251

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

CHAPTER 1 - COURSE INTRODUCTION

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

COURSE OBJECTIVES

- Write web applications that combine Java Servlets, JavaServer Pages, and JavaBeans using the Model-View-Controller architecture.
- Use JavaBeans to encapsulate business and data access logic.
- Generate HTML or XML output with JavaServer Pages.
- Process HTTP requests with Java Servlets.
- Configure your web applications with the *web.xml* deployment descriptor.
- Create scriptless JSPs by using JSTL tags combined with JSP Expression Language for functionality, such as conditionals, iteration, internationalization, and XML processing.

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

 © 2013 ITCourseware, LLC

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

COURSE OVERVIEW

- **Audience**: Java programmers who need to develop web applications using JSPs and Servlets.
- **Prerequisites**: Java programming experience and basic HTML knowledge are required.
- **Classroom Environment:**
  - A workstation per student.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

USING THE WORKBOOK

This workbook design is based on a page-pair, consisting of a Topic page and a Support page. When you lay the workbook open flat, the Topic page is on the left and the Support page is on the right. The Topic page contains the points to be discussed in class. The Support page has code examples, diagrams, screen shots and additional information. Hands On sections provide opportunities for practical application of key concepts. Try It and Investigate sections help direct individual discovery.

In addition, there is an index for quick look-up. Printed lab solutions are in the back of the book as well as on-line if you need a little help.

The Topic page provides the main topics for classroom discussion.

The Support page has additional information, examples and suggestions.

Topics are organized into first (❋), second (➢) and third (▪) level points.

Code examples are in a fixed font and shaded. The on-line file name is listed above the shaded area.

Callout boxes point out important parts of the example code.

Screen shots show examples of what you should see in class.

Pages are numbered sequentially throughout the book, making lookup easy.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Suggested References

Basham, Bryan, Kathy Sierra, and Bert Bates. 2004. ***Head First Servlets and JSP: Passing the Sun Certified Web Component Developer Exam (SCWCD)***. O'Reilly & Associates, Sebastopol, CA. ISBN 0596005407.

Bergsten, Hans. 2003. ***JavaServer Pages, 3rd Edition***. O'Reilly & Associates, Sebastopol, CA. ISBN 0596005636.

Hall, Marty and Larry Brown. 2003. ***Core Servlets and JavaServer Pages, Vol. 1: Core Technologies, 2nd Edition***. Prentice Hall, Englewood Cliffs, NJ. ISBN 0130092290.

Hall, Marty, Larry Brown and Yaakov Chaikin. 2006. ***Core Servlets and JavaServer Pages, Volume II (2nd Edition)***. Prentice Hall, Englewood Cliffs, NJ. ISBN 0131482602.

Heffelfinger, David, 2010. ***Java EE 6 with GlassFish 3 Application Server***. Packt Publishing, Birmingham, UK. ISBN 1849510369

Jendrock, Eric, et.al. 2010. ***The Java EE 6 Tutorial: Basic Concepts (4th Edition).*** Prentice Hall, Upper Saddle River, NJ. ISBN 0137081855

Steelman, Andrea, Joel Murach. Bergsten, Hans. 2008. ***Murach's Java Servlets and JSP, 2nd Edition***. Mike Murach & Associates. ISBN 1890774448.

Java Servlet Technology: ***http://www.oracle.com/technetwork/java/index-jsp-135475.html***
JSP Technology: ***http://www.oracle.com/technetwork/java/jsp-138432.html***
JSTL Technology: ***http://www.oracle.com/technetwork/java/jstl-137486.html***
Java EE 6 Tutorial: ***http://download.oracle.com/javaee/6/tutorial/doc/***

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

CHAPTER 2 - WEB APPLICATIONS AND MVC

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

OBJECTIVES

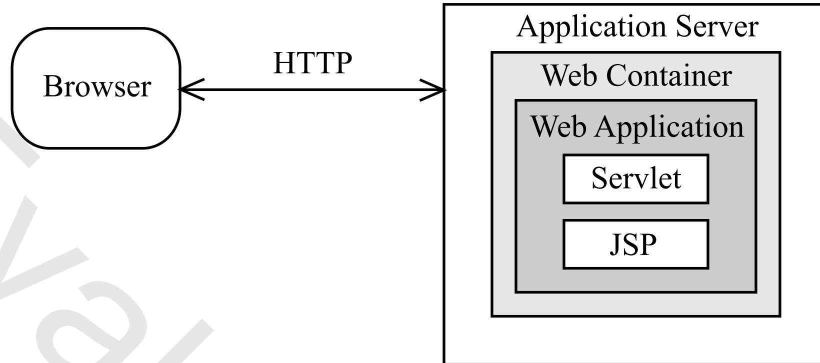
- Describe Java web technologies.
- Explain how the Model-View-Component architecture applies to a web application.
- Describe the structure of WAR files.
- Build and deploy a web application.

WEB APPLICATIONS

- *Web applications* are applications that the end user can access using a standard web browser.
- The Java Platform Enterprise Edition (Java EE) defines a web application as a collection of web components and supporting files.
  - Web components include Java servlets and JSP files.
  - Supporting files include static HTML documents, image files, and supporting classes.
- Your web application runs in the environment of a web container, which is managed by an application server.
  - Web containers can contain several web applications.
    - Your applications can work together or operate independently.
- Each web application is addressed with a context path.
  - The context path is determined when the application is deployed.
  - A web container can contain a "default" application, which has an empty context path.
  - To access a component or file in the web application from a browser, you must include the context path in the request URL.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC



Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

JSPs AND SERVLETS

- The standard protocol for communication between browsers and servers is designed for static documents.
  - Typically, a web server returns the contents of a static file in response to a browser request.
- Use servlets and JavaServer Pages (JSP) to handle requests from a browser dynamically.
  - A *servlet* is a web component which receives an object encapsulating the browser request and constructs a response to the browser.
    - The response typically contains an HTML document.
  - JSP pages start as text documents containing HTML or XML with special tags for executing Java code.
    - JSP pages are compiled into servlets automatically.
    - HTML designers do not need to learn Java.
    - Java developers do not need to learn HTML.
- You get some important benefits by using Java's web component architecture:
  - Your application will be portable across web containers.
  - You can get better performance and security from servlets than from standard CGI.
  - You can make full use of the vast set of Java APIs.
  - You can use facilities provided by the web container to maintain state.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

MODEL-VIEW-CONTROLLER

- The Model-View-Controller (MVC) architecture was originally described by Smalltalk for implementing GUI applications.

- The primary goal of MVC is to separate user interface code (the *view*) from domain code (the *model*).

  - The *controller* is introduced as a separate body of code that manages the translation of events in the view to procedures in the model.

  - The view only accesses the model to retrieve values for display.

  - The model should not have any knowledge of the view or the controller.

- The benefits of MVC are similar to encapsulation.

  - Changes in the model can be made without impacting the view.

  - The view can be modified, or new views can be implemented without impacting the model.

  - Developers can focus on their skills — database programmers do not need to understand user interface issues.

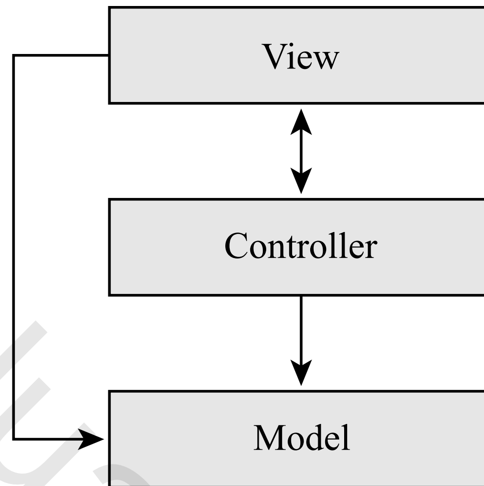
- MVC adapts well to the needs of web applications.

  - The view is further separated from the model both architecturally and physically.

  - The controller typically takes a broader role, managing the view as well as the model.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC



Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

MODEL 2 ARCHITECTURE

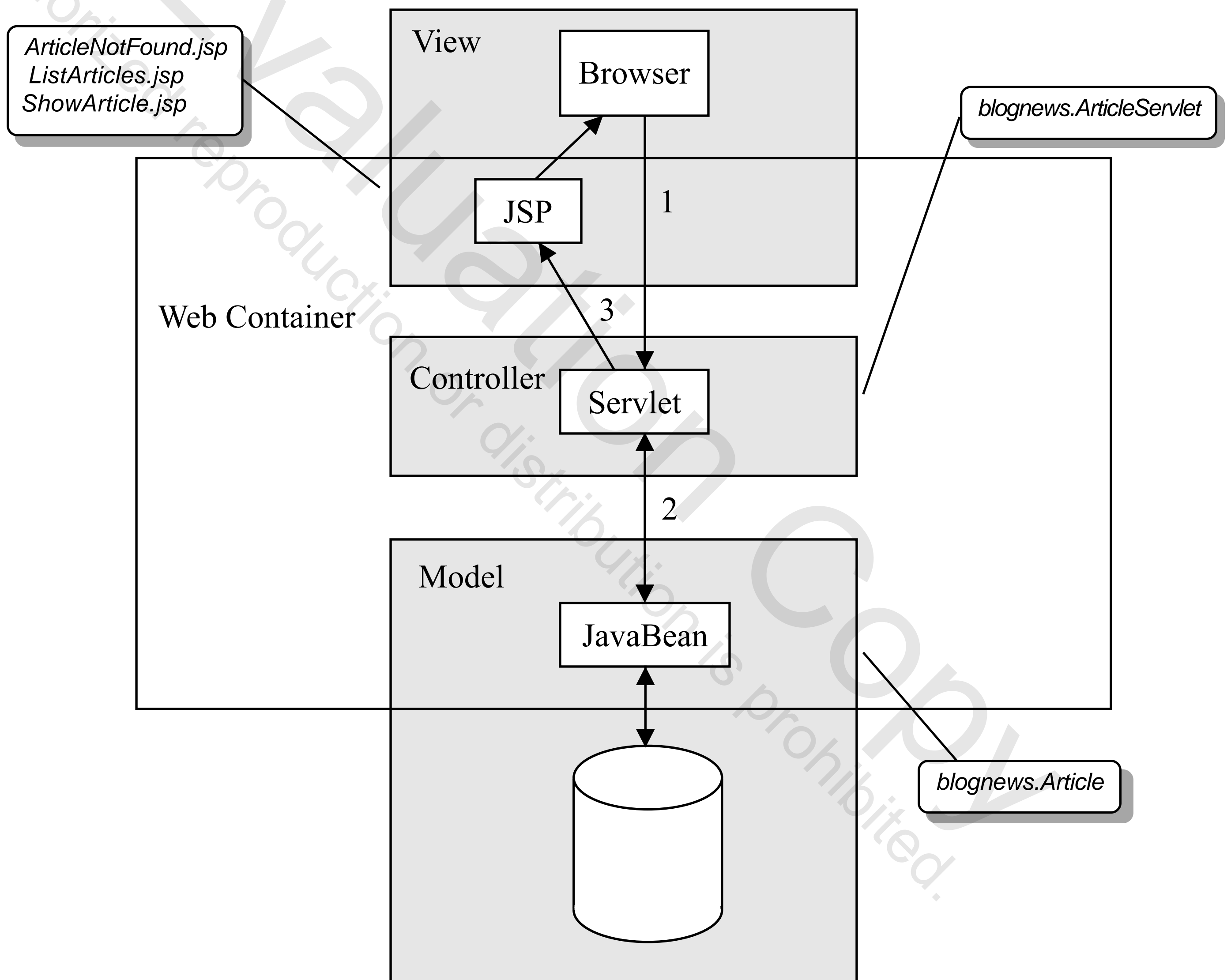
- The typical adaptation of MVC to Java web applications is called the *Model 2 Architecture*.
- Use JavaBeans to define the model.
  - The controller should encapsulate business object data in JavaBeans to make the data accessible to the JSP views.
  - The controller can provide helper JavaBeans to convert data from the business object to formats appropriate to the view.
    - For example, a JavaBean could convert a date to an appropriate string value.
- Use a servlet as the controller.
  - It will extract data needed to handle the request from the browser.
  - The servlet will also call methods on business objects to process the request.
  - Finally, it will forward the request to the JSP page, including any beans needed to generate the view.
    - The servlet might choose between JSP pages based on the results of the request.
- Use JSP pages to generate the view — typically an HTML or XML document.
  - The view will retrieve information to display from the beans included by the servlet.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited

In the early days of JSP, the popular architecture was what is now referred to as "Model 1." In this architecture, a browser request is handled directly by a JSP file, which, in turn, creates JavaBeans to access the business objects.

In both Model 1 and Model 2 architectures, JavaBeans are the preferred mechanism for accessing business objects. The JSP specification has strong support for working with JavaBeans objects, which makes it easier to separate the display logic of the JSP file from the business logic of the application.

Unauthorized reproduction or distribution is prohibited. Evaluation Copy



The filenames shown in the diagram refer to files in the BlogNews example application.

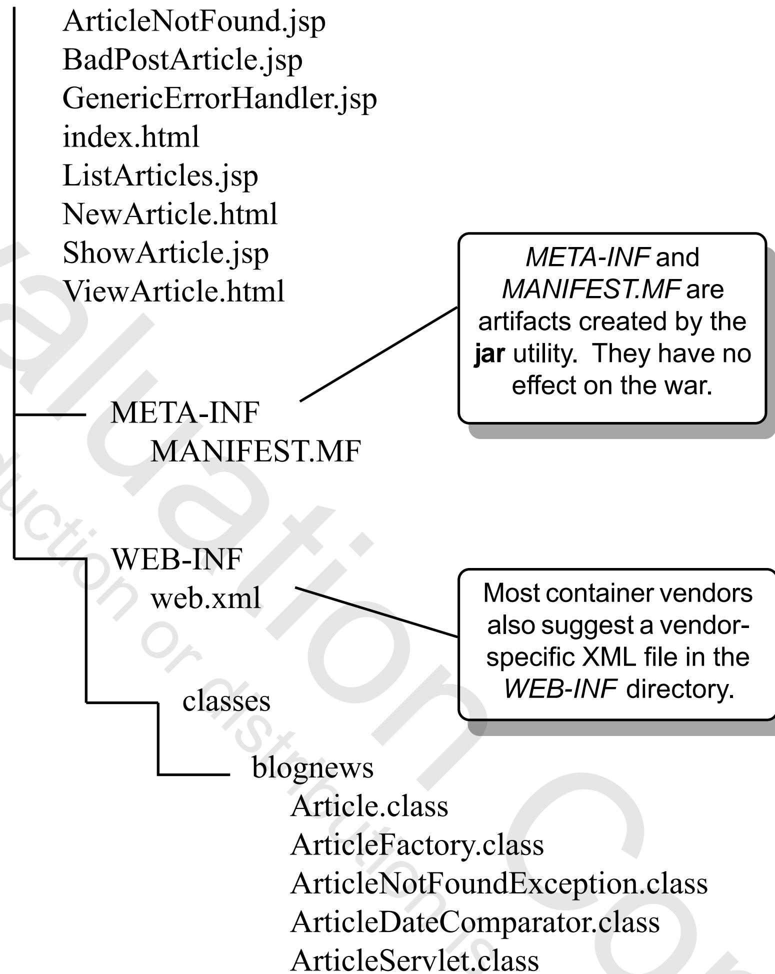
THE WAR FILE

- You must organize your web application using a specific directory structure.
  - The application root directory acts as the document root for your application.
    - You put your JSP, HTML, and other supporting files here.
    - You can use subdirectories to organize your application.
  - Store your application files in a subdirectory named *WEB-INF*.
    - Place the optional *web.xml* configuration file here.
    - This subdirectory is not accessible via the web server.
  - Put your servlet classes and supporting classes in the *WEB-INF/classes* directory.
  - Put any JAR files specific to your application in the *WEB-INF/lib* directory.
    - This is the preferred method for storing your JavaBeans.
    - If a JAR file will be used by other applications, it may make more sense to put it in a system-wide or server-wide directory.
- You can package your application for distribution in a Web ARchive (WAR) file.
  - A *WAR* file is a JAR file that contains all of the files in your application.
  - Since WAR files must conform to the Java EE specifications, they are portable between different web containers.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Contents of *BlogNews.war*:



Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

WEB.XML

- Provide an optional *deployment descriptor* to supply additional configuration information for your web application.
  - Create it as *WEB-INF/web.xml* in your web application directory.
- List files the container should look for when the user request specifies a context with the **<welcome-file-list>** element.
- Use the **<error-page>** element to delegate error handling to your own servlets, JSP pages, or HTML files.
  - This allows you to customize the appearance of your error pages dynamically.
  - HTTP errors are mapped by the 3-digit status code.

```
<error-page>
  <error-code>404</error-code>
  <location>/errors/PageNotFound.jsp</location>
</error-page>
```

  - Exceptions are mapped by the full class name of the exception handled.

```
<error-page>
  <exception-type>java.io.IOException</exception-type>
  <location>/errors/IOException.jsp</location>
</error-page>
```

    - The web container looks for a page matching the class of the exception thrown or one of its superclasses.
  - When the web container invokes your error handler, it provides request attributes with the error code or exception, and the original request URI.
    - Use these attributes to customize your error response.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Examples/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://
java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" id="WebApp_ID" version="3.0">
  <description>
    This is the BlogNews application. Setup is straightforward.
    No changes should be necessary at deployment unless you wish to
    change the default error page or the welcome file.
    Articles will be stored in the WEB-INF/articles directory which
    is created automatically the first time the servlet is accessed.
  </description>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <error-page>
    <error-code>404</error-code>
    <location>/GenericErrorHandler.jsp</location>
  </error-page>
</web-app>
```

You can use a **<description>** element to provide documentation.

The location is specified relative to the web application root.

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

Note:

Prior to the Servlet 3.0 specification, *web.xml* was required. Servlet 3.0 defined several annotations that you can add to your code to take the place of many, but not all, of the *web.xml* entries.

BUILDING THE WAR

- You build your application using these steps:

1. Create a directory in which to build your web application.

```
mkdir webapp
```

2. Compile your classes putting the resulting class files in *WEB-INF/classes*.

```
javac -d webapp/WEB-INF/classes *.java
```

   - If you create any JAR files, put them in *WEB-INF/lib*.

3. Copy your JSP files, HTML files, and other supporting files into the application directory.

4. Optionally, create your deployment descriptor in *WEB-INF/web.xml*.

- To build the WAR file, use the **jar** command to archive the application directory.

```
jar cf MyApplication.war webapp
```

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

To simplify the building and deployment process, we will use the Apache Software Foundation's Ant tool.

To use Ant, you create an XML configuration file called *build.xml* and define targets that define the steps for building and deploying your application. The targets also set dependencies, so that you can compile, build, and deploy with a single command.

For this class, we have provided *build.xml* files for you with targets for compiling the Java files, building the WAR file, and deploying the WAR file.

To complete all of the steps on the preceding page, you simply need to run the **ant** command with the appropriate build target. Your instructor will give you the details of how to run Ant and which target to use.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

DEPLOYING THE WAR

- You deploy a web application with these fundamental steps.

  1. Pass the WAR file to your web container.

     - You might simply copy the file to a specific location or use a tool to locate the file.

  2. Specify the context path for the application.

     - The context path often defaults to the name of the WAR file.

  3. Configure any container-managed resources as specified in the deployment descriptor.

     - These might include database connections, JNDI services, and security roles.

- The mechanisms for performing these is determined by your web container.

  - Some container providers have GUI or web-based tools for deploying applications.

  - You may need to create the appropriate configuration files manually and include them in your WAR file.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Try It:

The BlogNews application is a simple example of a web application using the Model 2 architecture. The application allows you to create and display a list of articles.

The servlet creates a new directory within the *WEB-INF* directory of the deployed application. The articles are stored in this new directory as serialized objects. The upside of this implementation is that you do not need to configure a storage directory or database. The downside is that this directory is usually destroyed when the application is redeployed.

The classes in the **blognews** package represent the controller, the model, and the business logic. The **Article** class is our model JavaBean. The **ArticleFactory** and supporting class **ArticleNotFoundException** provide the business logic for working with articles. The **ArticleServlet** is the controller

Start Tomcat, then build and deploy the application by running the **ant** command in the chapter's *Examples* directory. To view the application, navigate with your browser to ***http://localhost:8080/BlogNews***.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Labs

❶ Modify the deployment descriptor so that the 405 error (generated when a user tries to use a **GET** when a **POST** is required) is handled by the **GenericErrorHandler**. You can generate this error by entering the URL for the **View** command manually in your browser's address bar (*http://localhost:8080/BlogNews/Article/View*).
(Solution: *Solutions-Lab1/WebContent/WEB-INF/web.xml*)

❷ Modify the servlet so that it throws a runtime exception (try dividing by zero or dereference a null pointer). Deploy the application and observe the error displayed. Now, modify the deployment descriptor so that *GenericErrorHandler.jsp* is displayed instead.
(Solutions: *Solutions-Lab2/src/blognews/ArticleServlet.java*, *Solutions-Lab2/WebContent/WEB-INF/web.xml*)

❸ (Optional) Create a new view which lists only the titles of the articles. Add an action to the servlet to display the view, and add a link to *index.html* in order to access this action .
(Solutions: *Solutions-Lab3/src/blognews/ArticleServlet.java*, *Solutions-Lab3/WebContent/ListTitles.jsp*, *Solutions-Lab3/WebContent/index.html*)

❹ (Optional) Create a new view that allows the user to edit the body of an article. You will need to create a form to enter the title of the article to edit and a new action in **ArticleServlet** to display the new view using the article entered in the form. The new view can use the existing **Post** action in the servlet to save the changes. Add a link to *index.html* to display the new form.
(Solutions: *Solutions-Lab4/WebContent/EditArticle.jsp*, *Solutions-Lab4/WebContent/EditArticle.html*, *Solutions-Lab4/src/blognews/ArticleServlet.java*, *Solutions-Lab4/WebContent/index.html*)

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

CHAPTER 5 - JAVABEANS

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

OBJECTIVES

- Describe the JavaBean component model.
- Use JavaBeans to manage data and logic.
- Access JavaBeans from servlets and JSPs.
- Use the Data Access Object design pattern to encapsulate persistence logic.

WHAT IS A JAVABEAN?

- Originally, JavaBeans were defined as reusable software components that could be manipulated in an Integrated Development Environment (IDE).
  - They are used in client-side drag-and-drop user interface programming.
  - Beans expose properties that can be configured in your favorite IDE.
  - You can use classes and interfaces from the **java.beans** package to customize the behavior of the bean within an IDE.
- On the server side, JavaBeans are simply defined as reusable software components.
  - You can write a JavaBean for one web application and reuse it in another web application.
  - Many developers place their business logic in JavaBeans, rather than JSPs or servlets.
    - The same bean that contains business logic methods will most likely expose properties that correspond to the business data that the methods operate on.
    - In a Model-View-Controller architecture, the JavaBean serves as the model; it stores the data to be displayed by the view.
  - Many of the rules for client-side bean development do not apply to server-side beans.
    - You don't use drag-and-drop programming with server-side JavaBeans.
    - The **java.beans** package is not used for these types of beans.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

RULES

- When creating a JavaBean, you must keep a few rules in mind:
  - Your bean must have a no-argument constructor.
    - The compiler-supplied default constructor is fine.
  - In order to be useful, your bean should expose properties by defining **get()** and **set()** methods.
  - You can also place regular methods in a JavaBean.
  - Your bean should be part of a package.
  - Placing your beans in a JAR file is useful if you are planning on distributing them.
  - Making your JavaBeans implement **java.io.Serializable** is a good idea, especially if the bean needs to be converted to a stream for storage to the file system or for passing across the network.
    - Clustered environments may require you to have serializable JavaBeans.
- Compile your JavaBeans into your web application's *WEB-INF/classes* directory.

```
javac -d WEB-INF/classes MyBean.java
```

  - If your bean is in a JAR, place the JAR file into *WEB-INF/lib* instead.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Examples/src/library/SearchBean.java

```
package library;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.Date;

public class SearchBean implements java.io.Serializable {
   private String author = "";
   private String subject = "";
   private String title = "";
   private Date pubDate;
   private DateFormat formatter;

   public SearchBean() {
      formatter = DateFormat.getDateInstance(DateFormat.SHORT);
   }

   public String search() {
      return "Search engine temporarily unavailable";
   }

   // properties
   public String getAuthor() {
      return author;
   }
   public void setAuthor(String a) {
      author = a;
   }
   ...
}
```

Try It:

Run Ant in the chapter's *Examples* directory to build and deploy the examples for this chapter.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

PROPERTIES

- Define properties for your JavaBeans by writing **public get()** and **set()** methods.
  - You must follow the naming convention **get*Property*()** and **set*Property*()**.
    - You can use **is*Property*()** instead of **get*Property*()** for boolean properties.
  - The name of the property is dynamically derived by reflection from the get and set methods by lowercasing the first letter after the words "get" and "set."
    - For example, the methods **getFirstName()** and **setFirstName()** would yield a property called **firstName**.
  - The return type of the **get()** method must match the parameter type of the **set()** method.
  - The names of the properties do not have to correspond with the names of any underlying fields.
    - In fact, your properties can be derived; they don't have to be backed by fields.
- The web container will automatically convert JSP strings to basic types for your bean.
  - Basic types include Java's portable datatypes and **String**s.
  - If your property isn't one of the basic types, you can provide your own conversions.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Examples/src/library/SearchBean.java

```
...
public class SearchBean implements java.io.Serializable {
   private String author = "";
   private String subject = "";
   private String title = "";
   private Date pubDate;
   private DateFormat formatter;
   ...
   public String getAuthor() {
      return author;
   }
   public void setAuthor(String a) {
      author = a;
   }
   public String getPublicationDate() {
      if (pubDate != null)
         return formatter.format(pubDate);
      else
         return "";
   }
   public void setPublicationDate(String date) {
      try {
         pubDate = formatter.parse(date);
      }
      catch (ParseException e) {
         System.err.println(e);
         pubDate = new Date();
      }
   }
   public String getSubject() {
      return subject;
   }
   public void setSubject(String s) {
      subject = s;
   }
   public String getTitle() {
      return title;
   }
   public void setTitle(String t) {
      title = t;
   }
}
```

The property name doesn't have to match the field name.

The **publicationDate** property is stored as a **Date**, but exposed as a **String**.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

USING JAVABEANS IN JSPS

- Retrieve an instance of your bean with the **<jsp:useBean>** action.

```
<jsp:useBean id="myBean" class="BeanClass"/>
```

  - **id** is used to identify the bean at other places in your JSP.
  - **class** specifies the fully-qualified name of the class used to instantiate the bean.

- JSP provides special actions for retrieving and setting bean properties.

  - The **<jsp:getProperty>** action retrieves the value of a property from a bean.

```
<jsp:getProperty name="myBean" property="lastName"/>
```

  - The **<jsp:setProperty>** action sets the value of a bean property.

```
<jsp:setProperty name="myBean" property="lastName"
  value="Doe"/>
```

- Initialize bean properties by adding content between the beginning and ending **<jsp:useBean>** tags.

```
<jsp:useBean id="myBean" class="BeanClass">
  <jsp:setProperty name="myBean" property="lastName"
    value="Doe"/>
</jsp:useBean>
```

- Once you've used the bean, you can access it from JSP expressions.

```
<%=myBean.doIt()%>
```

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Examples/WebContent/ProcessSearch.jsp

```
<jsp:useBean id="bean" class="library.SearchBean"/>
<jsp:setProperty name="bean" property="subject"/>
<jsp:setProperty name="bean" property="title"/>
<jsp:setProperty name="bean" property="author"/>
<jsp:setProperty name="bean" property="publicationDate" param="date"/>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
  ...
  <body>
    Searching for:
     <jsp:getProperty name="bean" property="subject"/>
     <jsp:getProperty name="bean" property="title"/>
     <jsp:getProperty name="bean" property="author"/>
     <jsp:getProperty name="bean" property="publicationDate"/>
    <br/><br/>
    Results: <%= bean.search() %>
  </body>
</html>
```

Try It:

Fill in the form at ***http://localhost:8080/Beans/CatalogSearch1.html*** and submit the form. Even though the search engine is not running (it's just a stub), note that the parameters are passed to the results page.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

PROPERTIES AND FORMS

- HTML form parameters usually map to JavaBean properties.
  - You can set a bean property from a form parameter using an expression with the **<jsp:setProperty>** action.

```
<%String nm = request.getParameter("lastName");%>
<jsp:setProperty name="myBean" property="lastName"
  value="<%= nm %>"/>
```

  - This is such a common thing to do that JSP also provides a shorthand.

```
<jsp:setProperty name="myBean" property="lastName"/>
```

  - If the name of the property and the parameter don't match, use a slightly longer version.

```
<jsp:setProperty name="myBean" property="lastName"
  param="LastName"/>
```

  - You can even load all of the properties at once.

```
<jsp:setProperty name="myBean" property="*"/>
```

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

Examples/WebContent/CatalogSearch1.html

```
...
<html>
    ...
    <form action="ProcessSearch.jsp" method="get">
      <table border="1" cellpadding="2" class="center">
        ...
        <tr>
          <td>Subject</td>
           <td><input name="subject" type="text" size="40"/></td>
        </tr>
        <tr>
          <td>Title</td>
           <td><input name="title" type="text" size="40"/></td>
        </tr>
        <tr>
          <td>Author</td>
           <td><input name="author" type="text" size="40"/></td>
        </tr>
        <tr>
           <td>Publication Date (MM/DD/YYYY)</td>
           <td><input name="date" type="text" size="40"/></td>
        </tr>
      </table>
      <input type="submit" value="Search"/>
    </form>
  </body>
</html>
```

Examples/WebContent/ProcessSearch.jsp

```
<jsp:useBean id="bean" class="library.SearchBean"/>
<jsp:setProperty name="bean" property="subject"/>
<jsp:setProperty name="bean" property="title"/>
<jsp:setProperty name="bean" property="author"/>
<jsp:setProperty name="bean" property="publicationDate" param="date"/>
...
  <body>
    Searching for:
     <jsp:getProperty name="bean" property="subject"/>
     <jsp:getProperty name="bean" property="title"/>
     <jsp:getProperty name="bean" property="author"/>
     <jsp:getProperty name="bean" property="publicationDate"/>
    <br/><br/>
    Results: <%= bean.search() %>
  </body>
</html>
```

Use the **param** attribute when your property name doesn't match the parameter name.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

DATA ACCESS OBJECTS

- Use the *Data Access Object* (DAO) design pattern to abstract your persistence logic from your business logic.
  - A DAO encapsulates the code that is used to talk to your persistence store.
    - You can change how you persist your data without affecting client code that accesses the DAO.
  - The DAO pattern does not define how the data could be stored.
    - Behind the scenes, your DAO can work with a relational database, the file system, an LDAP repository, etc.
  - A DAO can be used to query, as well as modify, persistent data.
- Return results from a DAO with a transfer object.
  - A *transfer object* is a simple Java class that acts as a data carrier that can be passed from one part of your application to another.
  - Use JavaBean conventions when building your transfer objects so they can integrate with JSPs.
- You can also pass a transfer object to your DAO if you want it to write to the underlying data store.
- For more information on the DAO design pattern, see: ***http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html***.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Examples/src/library/Book.java

Book is a transfer object.

```
...
public class Book implements java.io.Serializable {
  private String isbn;
  private String callNumber;
  private String title;
  private String status;
  private Collection<Author> authors;
  ...
}
```

Examples/src/library/LibraryDAO.java

```
...
public interface LibraryDAO {
  public Collection<Book> getBooksByIsbn(String isbn) throws Exception;
  public Collection<Book> getBooksByAuthor(String ln) throws Exception;
  public Collection<Book> getBooksByKeyword(String kw) throws Exception;
  public Collection<Book> getBooksByTitle(String t) throws Exception;
}
```

Each of these methods return a **Collection** of **Book** objects.

Examples/src/library/LibraryDAOImpl.java

```
...
public class LibraryDAOImpl implements LibraryDAO {
  ...
  public Collection<Book> getBooksByIsbn(String isbn) throws Exception {
    ...
    String sqltxt = "SELECT b.isbn, b.title, b.call_number, " +
      "b.status, a.id, a.first_name, a.middle_name, a.last_name " +
      "FROM book b, author a, book_author ba " +
      "WHERE b.isbn = ba.isbn " +
      "and a.id = ba.id " +
      "and b.isbn=?";
    try {
      conn = getConnection();
      pst = conn.prepareStatement(sqltxt);
      pst.setString(1, isbn);
      rs = pst.executeQuery();

      books = addBooksToCollection(rs);
    }
    ...
    return books;
  }
  ...
}
```

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

RESOURCE REFERENCE

- You will often need to connect to external resources from your web applications.
  - You may not know the configuration of the resource at compile time.
  - You may want to allow the resource to be reconfigured in a live system.
- You can use *resource references* to get connections to resources that can be configured at deployment.
- Declare your resource references in *web.xml* using a **<resource-ref>** element.
  - For **<res-ref-name>** specify the name your code uses to access the resource.
  - For **<res-type>** specify the Java class name of the resource factory type.
  - For **<res-auth>** specify **Container** if the deployer provides authorization information or **Application** if your code does.
- In your code, use JNDI to lookup the resource factory by the reference name you declared in *web.xml*.

```
InitialContext ctxt = new InitialContext();
DataSource ds = (DataSource)ctxt.lookup("java:comp/env/jdbc/books");
```

  - The name is relative to the *java:comp/env/* context.
- The deployer must map the **<res-ref-name>**, to a resource provided through the Java EE environment.
  - The details of this mapping are implementation-specific.

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

Examples/WebContent/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <resource-ref>
    <res-ref-name>jdbc/books</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

Examples/WebContent/META-INF/context.xml

```
<!-- Apache Jakarta Tomcat Deployment Descriptor -->
<Context path="/Beans" reloadable="true" crossContext="true">
  <Resource name="jdbc/books"
    auth="Container" type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    username="app" password="app"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby://localhost:1527/jweb"/>
</Context>
```

Map the **res-ref-name** defined in *web.xml* to the actual resource via the Tomcat *context.xml* config file.

Examples/src/library/LibraryDAOImpl.java

```
...
public class LibraryDAOImpl implements LibraryDAO {
  ...
  private Connection getConnection() throws Exception {
    InitialContext ctxt = new InitialContext();
    DataSource ds =
      (DataSource) ctxt.lookup("java:comp/env/jdbc/books");
    Connection conn = ds.getConnection();
    return conn;
  }
}
```

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

BEAN SCOPES IN SERVLETS

- Your beans can be stored in one of four different scopes within your servlet code.
  - Store the bean as a local variable.
  - Store it as an attribute within the **HttpServletRequest** so that it will be available across includes and forwards.
  - Make the bean available across multiple pages for the current user by storing it in the **HttpSession**.
  - Store it within the **ServletContext** to make it globally available to the application; all pages for all users can see it.
- Beans stored in the **ServletContext** or the **HttpSession** are susceptible to race conditions.
  - Servlets and JSPs are multi-threaded by their very nature.
  - Use Java concurrency features, such as the **synchronized** keyword, to prevent race conditions.
  - Another strategy to avoid race conditions is to make your beans read-only.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

 © 2013 ITCourseware, LLC

Examples/src/library/ProcessSearchServlet.java

```
...
@WebServlet(urlPatterns = {"/ProcessSearch"})
public class ProcessSearchServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {

    String searchType = req.getParameter("searchType");
    String searchString = req.getParameter("searchString");

    LibraryDAO dao = new LibraryDAOImpl();
    Collection<Book> books = null;

    try {
      if (searchType.equals("isbn")) {
        books = dao.getBooksByIsbn(searchString);
      }
      else if (searchType.equals("author")) {
        books = dao.getBooksByAuthor(searchString);
      }
      else if (searchType.equals("subject")) {
        books = dao.getBooksByKeyword(searchString);
      }
      else if (searchType.equals("title")) {
        books = dao.getBooksByTitle(searchString);
      }

      if (books == null || books.size() == 0) {
        forwardToJSP(req, res, "/DisplayNoResults.jsp");
      }
      else {
        req.setAttribute("book", books.iterator().next());
        forwardToJSP(req, res, "/DisplayResults.jsp");
      }
    }
    catch (Exception e) {
      req.setAttribute("exception", e);
      forwardToJSP(req, res, "/DisplayErrors.jsp");
    }

  }
  ...
}
```

Use the DAO to talk to the database, instead of embedding persistence logic in the servlet.

For simplicity, store the first **Book** in the request.

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

BEAN SCOPES IN JSPS

- Within the **<jsp:useBean>** action, you can specify four different scopes for the bean.

```
<jsp:useBean id="name" class="BeanClass" scope="page"/>
```

  - **page** scope is essentially local; as soon as the JSP has completed generating the page, the bean is destroyed.
    - The bean is only available to the JSP that declares it.
    - **page** is the default scope.
  - **request** scope means the bean is available for the entire request, including any forwards.

```
<jsp:useBean id="name" class="BeanClass"
  scope="request"/>
```

  - **session** scope means the bean is available for the entire session, which may span **request**s.

```
<jsp:useBean id="name" class="BeanClass"
  scope="session"/>
```

  - **application** scope means the bean is available anywhere in the application.
- The web container will attempt to locate a bean based on the **id** and **scope**.
  - If it can't be found, then a new instance will be created for you.
  - If you embed **setProperty** actions within the **useBean** tags, they will only be invoked if a new instance is created.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited

 © 2013 ITCourseware, LLC

Examples/WebContent/DisplayResults.jsp

```
<jsp:useBean id="book" class="library.Book" scope="request"/>
...
<html>
  ...
  <body>
    <h2>Search Results</h2>
    <table border="1" class="center">
      <tr>
         <th>isbn</th><th>title</th><th>status</th><th>authors</th>
      </tr>
      <tr>
         <td><jsp:getProperty name="book" property="isbn"/></td>
         <td><jsp:getProperty name="book" property="title"/></td>
         <td><jsp:getProperty name="book" property="status"/></td>
         <td><jsp:getProperty name="book" property="authors"/></td>
      </tr>
    </table>
    <a href="CatalogSearch2.html">Try Again?</a>
  </body>
</html>
```

Use **scope="request"** since the servlet stored the **Book** instance in the request.

Try It:

Search for books by visiting ***http://localhost:8080/Beans/CatalogSearch2.html***. Your instructor will tell you how to start the database.

Unauthorized reproduction or distribution is prohibited.
Evaluation Copy

Labs

❶ You will find an incomplete employee search application in the *StarterCode* directory. Examine the files, then write an HTML form that prompts the user to enter an employee's id. The form's action attribute should be **"EmployeeSearch"**.
(Solution: *Solutions/WebContent/EmployeeSearch.html*)

❷ Examine *src/emp/EmployeeDAO.java*, *src/emp/EmployeeDAOImpl.java*, and *src/emp/Employee.java* from the chapter's *StarterCode* directory. Write a servlet that reads the id from the request and calls the DAO to retrieve the **Employee** transfer object. Store the transfer object in the request and forward to the JSP you will write in ❸.
(Solution: *Solutions/src/emp/EmployeeSearchServlet.java*)

❸ Write a JSP that displays the properties of the **Employee** transfer object that was stored in the request in ❷. Create another JSP that your servlet will invoke if any exceptions occur. Build and deploy your application.
(Solutions: *Solutions/WebContent/DisplayEmployee.jsp*, *Solutions/WebContent/DisplayErrors.jsp*)

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.

You can use the employee id **123443** to test your application.

Evaluation Copy
Unauthorized reproduction or distribution is prohibited.