Session Bean Example

Example

- CurrencyBean… stateless session bean that converts euros to dollars
- Files
  - » ICurrency.java (Business logic interface)
  - » CurrencyBeanRemote.java (Remote interface)
  - » CurrencyBeanLocal.java (Local interface)
  - » CurrencyBean.java (Implementation Code)
  - » CurrencyClient.java (Test program)

ICurrency.java

```
package ie.nuigalway.ct414;

import java.io.Serializable;

public interface ICurrency extends Serializable {

        public float euro2dollars(float amount);
}
```

CurrencyBeanRemote.java

```
package ie.nuigalway.ct414;
import javax.ejb.Remote;

@Remote
public interface CurrencyBeanRemote extends ICurrency {

}
```

CurrencyBeanLocal.java

```
package ie.nuigalway.ct414;
import javax.ejb.Local;

@Local
public interface CurrencyBeanLocal extends ICurrency {

}
```

CurrencyBean.java

```
package ie.nuigalway.ct414;
import javax.ejb.Stateless;


@Stateless (mappedName="CurrencyTest")
public class CurrencyBean extends Object implements CurrencyBeanRemote, CurrencyBeanLocal {
        private static final long serialVersionUID = 1L;
        public CurrencyBean() {
                super();
        }
        public float euro2dollars(float amount) {
                return amount / (float) 1.3;
        }
}
```

CurrencyClient.java

```
package ie.nuigalway.ct414;
import javax.naming.InitialContext;
import javax.naming.NamingException;


public class CurrencyClient {
        public static void main(String[] args) {
                try {
                        InitialContext ctx = new InitialContext();
                        CurrencyBeanRemote bean = (CurrencyBeanRemote)
                                ctx.lookup("CurrencyTest");
                        float value = bean.euro2dollars((float)10.00);
                        System.out.printf("€10.00 = US$%.2f", value);
                        } catch (NamingException ex) {  ex.printStackTrace(); }
}
```

Using Entity EJBs

- Entity EJBs
  - Object-based representations of information-tier data
    - e.g., data stored in relational database
  - Represents a particular unit of data
    - e.g., record in a database table
  - Two types:
    - Bean-managed persistence
    - Container-managed persistence

`Employee` Entity EJB

- Build an EJB that represents an **`Employee`**
  - Example uses Bean-managed persistence
  - Container-managed persistence would be easier but is not possible in all cases
  - Example taken from from Deitels Advanced Java How to Program Book (available in library)

```
1    // Employee.java
2    // Employee is the remote interface for the Address EJB.
3    package com.deitel.advjhtp1.ejb.entity;
4
5    // Java core libraries
6    import java.rmi.RemoteException;
7
8    // Java standard extensions
9    import javax.ejb.EJBObject;
10
11   public interface Employee extends EJBObject {
12
13      // get Employee ID
14      public Integer getEmployeeID() throws RemoteException;
15
16      // set social security number
17      public void setSocialSecurityNumber( String number )
18         throws RemoteException;
19
20      // get social security number
21      public String getSocialSecurityNumber()
22         throws RemoteException;
23
24      // set first name
25      public void setFirstName( String name )
26         throws RemoteException;
27
28      // get first name
29      public String getFirstName() throws RemoteException;
30
31      // set last name
32      public void setLastName( String name )
33         throws RemoteException;
34
```

All EJB remote interfaces must extend interface **`EJBObject`**

Interface **`Employee`** provides access methods for each **`Employee`** property

Distributed Systems Course.
See Deitels Book for details.

```
35      // get last name
36      public String getLastName() throws RemoteException;
37
38      // set title
39      public void setTitle( String title )
40         throws RemoteException;
41
42      // get title
43      public String getTitle() throws RemoteException;
44
45      // set salary
46      public void setSalary( Double salary ) throws RemoteException;
47
48      // get salary
49      public Double getSalary() throws RemoteException;
50   }
```

Interface **Employee** provides access methods for each **Employee** property

`Employee` EJB with Bean-Managed Persistence

- **`Employee`** EJB
  - Bean-managed persistence
    - Use JDBC to store **`Employee`** data in an underlying database

```
1    // EmployeeEJB.java
2    // EmployeeEJB is an entity EJB that uses bean-managed
3    // persistence to persist Employee data in a database.
4    package com.deitel.advjhtp1.ejb.entity.bmp;
5
6    // Java core libraries
7    import java.sql.*;
8    import java.rmi.RemoteException;
9
10   // Java standard extensions
11   import javax.ejb.*;
12   import javax.sql.*;
13   import javax.naming.*;
14
15   public class EmployeeEJB implements EntityBean {
16
17      private EntityContext entityContext;
18      private Connection connection;
19
20      private Integer employeeID;
21      private String socialSecurityNumber;
22      private String firstName;
23      private String lastName;
24      private String title;
25      private Double salary;
26
27      // get Employee ID
28      public Integer getEmployeeID()
29      {
30         return employeeID;
31      }
32
```

All EJB implementations must implement interface **`EntityBean`**

`EntityContext` provides EJB with information about container that deploys the EJB

Member variables to store data retrieved from the database and updates from the client

```
33      // set social security number
34      public void setSocialSecurityNumber( String number )
35      {
36         socialSecurityNumber = number;
37      }
38
39      // get social security number
40      public String getSocialSecurityNumber()
41      {
42         return socialSecurityNumber;
43      }
44
45      // set first name
46      public void setFirstName( String name )
47      {
48         firstName = name;
49      }
50
51      // get first name
52      public String getFirstName()
53      {
54         return firstName;
55      }
56
57      // set last name
58      public void setLastName( String name )
59      {
60         lastName = name;
61      }
62
63      // get last name
64      public String getLastName()
65      {
66         return lastName;
67      }
```

Distributed Systems Course.
See Deitels Book for details.

```
68
69       // set title
70       public void setTitle( String jobTitle )
71       {
72          title = jobTitle;
73       }
74
75       // get title
76       public String getTitle()
77       {
78          return title;
79       }
80
81       // set salary
82       public void setSalary( Double amount )
83       {
84          salary = amount;
85       }
86
87       // get salary
88       public Double getSalary()
89       {
90          return salary;
91       }
92
93       // create Employee
94       public Integer ejbCreate( Integer primaryKey )
95          throws CreateException
96       {
97          employeeID = primaryKey;
98
```

When a client invokes interface **`EmployeeHome`** method **`create`**, the EJB container invokes method **`ejbCreate`**

```
99          // INSERT new Employee in database
100         try {
101
102            // create INSERT statement
103            String insert = "INSERT INTO Employee " +
104               "( employeeID ) VALUES ( ? )";
105
106            // create PreparedStatement to perform INSERT
107            PreparedStatement insertStatement =
108               connection.prepareStatement( insert );
109
110            // set values for PreparedStatement
111            insertStatement.setInt( 1, employeeID.intValue() );
112
113            // execute INSERT and close PreparedStatement
114            insertStatement.executeUpdate();
115            insertStatement.close();
116
117            return employeeID;
118         }
119
120         // throw EJBException if INSERT fails
121         catch ( SQLException sqlException ) {
122            throw new CreateException( sqlException.getMessage() );
123         }
124      } // end method ejbCreate
125
126      // do post-creation tasks when creating Employee
127      public void ejbPostCreate( Integer primaryKey ) {}
128
129      // remove Employee information from database
130      public void ejbRemove() throws RemoveException
131      {
```

Create a **PreparedStatement** to **INSERT** the new **Employee** in the database

INSERT the **Employee** in the database

EJB container invokes method **ejbPostCreate** after invoking method **ejbCreate** to perform required tasks

When a client invokes interface **EmployeeHome** method **remove**, the EJB container invokes method **ejbRemove**

```
132         // DELETE Employee record
133         try {
134
135            // get primary key of Employee to be removed
136            Integer primaryKey =
137               ( Integer ) entityContext.getPrimaryKey();
138
139            // create DELETE statement
140            String delete = "DELETE FROM Employee WHERE " +
141               "employeeID = ?";
142
143            // create PreparedStatement to perform DELETE
144            PreparedStatement deleteStatement =
145               connection.prepareStatement( delete );
146
147            // set values for PreparedStatement
148            deleteStatement.setInt( 1, primaryKey.intValue() );
149
150            // execute DELETE and close PreparedStatement
151            deleteStatement.executeUpdate();
152            deleteStatement.close();
153         }
154
155         // throw new EJBException if DELETE fails
156         catch ( SQLException sqlException ) {
157            throw new RemoveException( sqlException.getMessage() );
158         }
159      } // end method ejbRemove
160
161      // store Employee information in database
162      public void ejbStore() throws EJBException
163      {
```

Create a **Prepared-Statement** to **DELETE** the **Employee** from the database

DELETE the **Employee** from the database

EJB container invokes **ejbStore** to save **Employee** data in the database

```
164         // UPDATE Employee record
165         try {
166
167            // get primary key for Employee to be updated
168            Integer primaryKey =
169               ( Integer ) entityContext.getPrimaryKey();
170
171            // create UPDATE statement
172            String update = "UPDATE Employee SET " +
173               "socialSecurityNumber = ?, firstName = ?, " +
174               "lastName = ?, title = ?, salary = ? " +
175               "WHERE employeeID = ?";
176
177            // create PreparedStatement to perform UPDATE
178            PreparedStatement updateStatement =
179               connection.prepareStatement( update );
180
181            // set values in PreparedStatement
182            updateStatement.setString( 1,socialSecurityNumber);
183            updateStatement.setString( 2,firstName );
184            updateStatement.setString( 3,lastName );
185            updateStatement.setString( 4,title );
186            updateStatement.setDouble( 5,salary.doubleValue());
187            updateStatement.setInt( 6, primaryKey.intValue() );
188
189            // execute UPDATE and close PreparedStatement
190            updateStatement.executeUpdate();
191            updateStatement.close();
192         }
193
194         // throw EJBException if UPDATE fails
195         catch ( SQLException sqlException ) {
196            throw new EJBException( sqlException );
197         }
198      } // end method ejbStore
```

Create a **Prepared-Statement** to **UPDATE** the **Employee** information in the database

UPDATE the **Employee** information in the database

Distributed Systems Course.
See Deitels Book for details.

```
199
200     // load Employee information from database
201     public void ejbLoad() throws EJBException
202     {
203        // get Employee record from Employee database table
204        try {
205
206           // get primary key for Employee to be loaded
207           Integer primaryKey =
208              ( Integer ) entityContext.getPrimaryKey();
209
210           // create SELECT statement
211           String select = "SELECT * FROM Employee WHERE " +
212              "employeeID = ?";
213
214           // create PreparedStatement for SELECT
215           PreparedStatement selectStatement =
216              connection.prepareStatement( select );
217
218           // set employeeID value in PreparedStatement
219           selectStatement.setInt( 1, primaryKey.intValue() );
220
221           // execute selectStatement
222           ResultSet resultSet = selectStatement.executeQuery();
223
224           // get Employee information from ResultSet and update
225           // local member variables to cache data
226           if ( resultSet.next() ) {
227
228              // get employeeID
229              employeeID = new Integer( resultSet.getInt(
230                 "employeeID" ) );
231
```

EJB container invokes **ejbLoad** to copy **Employee** data from the database to **Employee** member variables

Create a **PreparedStatement** to **SELECT** the **Employee** information in the database

SELECT the **Employee** information in the database

Distributed Systems Course.
See Deitels Book for details.

```
232             // get social-security number
233             socialSecurityNumber = resultSet.getString(
234                "socialSecurityNumber" );
235
236             // get first name
237             firstName = resultSet.getString( "firstName" );
238
239             // get last name
240             lastName = resultSet.getString( "lastName" );
241
242             // get job title
243             title = resultSet.getString( "title" );
244
245             // get salary
246             salary = new Double( resultSet.getDouble(
247                "salary" ) );
248
249          } // end if
250
251          else
252             throw new EJBException( "No such employee." );
253
254          // close PreparedStatement
255          selectStatement.close();
256
257       } // end try
258
259       // throw EJBException if SELECT fails
260       catch ( SQLException sqlException ) {
261          throw new EJBException( sqlException );
262       }
263    } // end method ejbLoad
264
```

Store database data in **`Employee`** member variables

Distributed Systems Course.
See Deitels Book for details.

```
265     // find Employee using its primary key
266     public Integer ejbFindByPrimaryKey( Integer primaryKey )
267        throws FinderException, EJBException
268     {
269        // find Employee in database
270        try {
271
272           // create SELECT statement
273           String select = "SELECT employeeID FROM Employee " +
274              "WHERE employeeID = ?";
275
276           // create PreparedStatement for SELECT
277           PreparedStatement selectStatement =
278              connection.prepareStatement( select );
279
280           // set employeeID value in PreparedStatement
281           selectStatement.setInt( 1, primaryKey.intValue() );
282
283           // execute selectStatement
284           ResultSet resultSet = selectStatement.executeQuery();
285
286           // return primary key if SELECT returns a record
287           if ( resultSet.next() ) {
288
289              // close resultSet and selectStatement
290              resultSet.close();
291              selectStatement.close();
292
293              return primaryKey;
294           }
295
296           // throw ObjectNotFoundException if SELECT produces
297           // no records
298           else
299              throw new ObjectNotFoundException();
```

When a client invokes interface **EmployeeHome** method **findByPrimaryKey**, the EJB container invokes method **ejbFindByPrimaryKey**

Obtain record from database via primary key and **SELECT** statement

Distributed Systems Course.
See Deitels Book for details.

```
300         }
301
302         // throw EJBException if SELECT fails
303         catch ( SQLException sqlException ) {
304            throw new EJBException( sqlException );
305         }
306      } // end method ejbFindByPrimaryKey
307
308      // set EntityContext and create DataSource Connection
309      public void setEntityContext( EntityContext context )
310         throws EJBException
311      {
312         // set entityContext
313         entityContext = context;
314
315         // look up the Employee DataSource and create Connection
316         try {
317            InitialContext initialContext = new InitialContext();
318
319            // get DataSource reference from JNDI directory
320            DataSource dataSource = ( DataSource )
321               initialContext.lookup(
322                  "java:comp/env/jdbc/Employee" );
323
324            // get Connection from DataSource
325            connection = dataSource.getConnection();
326         }
327
328         // handle exception if DataSource not found in directory
329         catch ( NamingException namingException ) {
330            throw new EJBException( namingException );
331         }
332
```

EJB container invokes method **`setEntityContext`** after the EJB is first created, but before the EJB is associated with a particular database record

Use JNDI name and **`InitialContext`** to locate **`Employee`** database in JNDI directory

Distributed Systems Course.
See Deitels Book for details.

```
333         // handle exception when getting Connection to DataSource
334         catch ( SQLException sqlException ) {
335            throw new EJBException( sqlException );
336         }
337      } // end method setEntityContext
338
339      // unset EntityContext
340      public void unsetEntityContext() throws EJBException
341      {
342         entityContext = null;
343
344         // close DataSource Connection
345         try {
346            connection.close();
347         }
348
349         // throw EJBException if closing Connection fails
350         catch ( SQLException sqlException ) {
351            throw new EJBException( sqlException );
352         }
353
354         // prepare connection for reuse
355         finally {
356            connection = null;
357         }
358      }
359
360      // set employeeID to null when container passivates EJB
361      public void ejbPassivate()
362      {
363         employeeID = null;
364      }
365
```

EJB container invokes method **`unsetEntityContext`** when EJB is no longer needed

EJB container invokes method **`ejbPassivate`** to place active EJB back in inactive pool

```
366     // get primary key value when container activates EJB
367     public void ejbActivate()
368     {
369        employeeID = ( Integer ) entityContext.getPrimaryKey();
370     }
371  }
```

EJB container invokes method **`ejbActivate`** to activate EJB from inactive pool

Case Study

- Online bookstore e-business application
  - Web Components
  - EJBs
  - XML
  - XSLT
  - MVC architecture

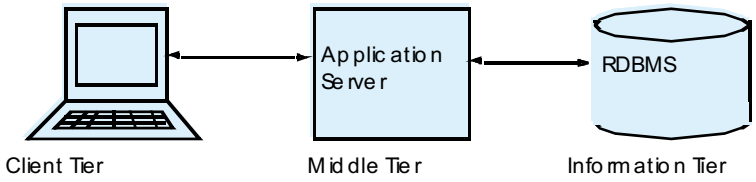
Online Bookstore Application

- Case study
  - Implement functionality for commercial on-line store
    - Provide product catalog
    - Provide shopping cart
    - Provide customer registration
    - Allow customers to view previous orders
    - Provide functionality for several clients
      - Standard Web browsers
      - cHTML (Compact HTML) for i-Mode browsers

System Architecture

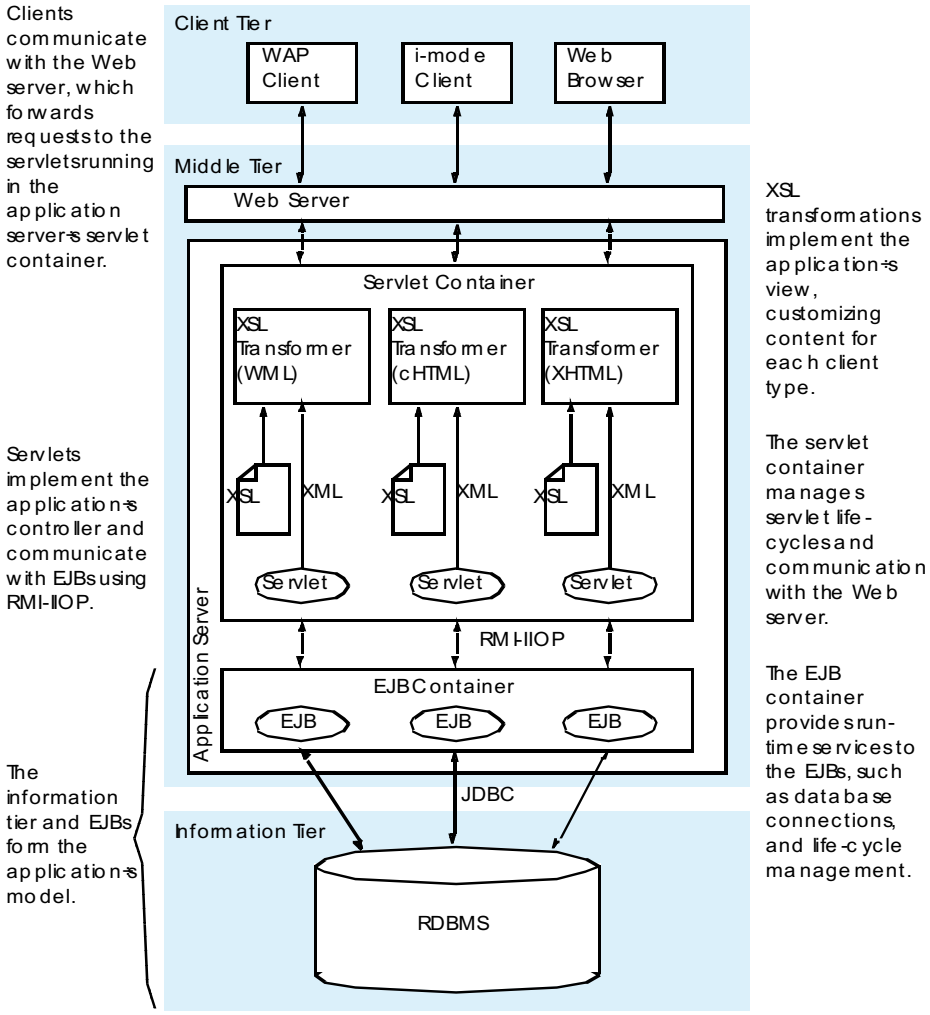
- Multi-tier application
  - Information tier
    - Maintains data for application (RDBMS)
  - Middle tier
    - Implements business logic and controller logic
      - Control interactions between information and client tiers
  - Client tier
    - Application's user interface (e.g., Web browser)

System Architecture (cont.)



Three-tier application model in Deitel Bookstore.

System Architecture (cont.)



Detailed architecture of Deitel Bookstore Enterprise Java case study.

Enterprise JavaBeans

- EJBs
  - Implement business logic and database abstraction layer
  - Stateful session EJB
    - Represents a customer's shopping cart
  - Entity EJB
    - Provide object-based interface to information tier
- Web Components (Servlets / JSP)
  - Use EJB business logic to create an on-line store

Entity EJBs

- Entity EJB
  - Provide object-based interface to information tier
  - Represents object stored in application's relational database
    - Class **`Customer`** stores
      - First name / Last name
      - Billing address
      - Shipping address
      - Credit-card information
    - Class **`Product`**
    - Class **`Order`**
    - Class **`OrderProduct`**
      - Many-to-many relationship between Orders and Products

Entity EJBs (cont.)

- Entity EJB
  - Each entity EJB has a corresponding model class
    - This is a utility class that encapsulates the data associated with a particular entity EJB.
    - e.g., **`Product`** EJB has corresponding **`ProductModel`**
      - **`ProductModel`** is a serializable object that encapsulates the attributes of a product and has properties for
        - **`Product`** ISBN
        - **`Product`** price
        - **`Product`** author

Stateful Session EJBs

- **`ShoppingCart`**
  - Stateful session EJB
  - Manages customer's shopping cart, used by customers to gather products as they browse store
  - Contains a collection of OrderProductModel objects
  - Implements business logic for managing each shopping cart
  - Is stateful so it will persist throughout user's session

17.5 Servlet Controller Logic

- Servlets
  - Middle-tier interface between client and business logic
  - Implement the controller in MVC architecture
  - Interact with EJB business-logic components
    - Handle client requests (via HTTP)
    - Process data as XML documents
    - Pass XML documents through XSL transformations
      - Produce presentation for each client

XSLT Presentation Logic

- Generate appropriate presentation for each client
  - Each servlet employs XSL **`Transformer`** and XSLTs
    - One XSLT for producing XHTML
    - One XSLT for producing cHTML

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <catalog>
3       <product>
4          <isbn>0130284173</isbn>
5          <publisher>Prentice Hall</publisher>
6          <author>Deitel, Deitel, Nieto, Lin &amp;amp; Sadhu</author>
7          <title>XML How to Program</title>
8          <price>$69.95</price>
9          <pages>1200</pages>
10         <image>images/xmlhtp1.jpg</image>
11         <media>CD</media>
12         <quantity>500</quantity>
13      </product>
14   </catalog>
```

XML document marks up a product, including the product's ISBN, title, author, publisher and price

```
1    <?xml version = "1.0"?>
2
3    <!-- ProductDetails.xsl                                    -->
4    <!-- XSLT stylesheet for transforming content generated by -->
5    <!-- GetProductServlet into XHTML.                         -->
6
7    <xsl:stylesheet version = "1.0"
8       xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
9
10      <xsl:output method = "xml" omit-xml-declaration = "no"
11         indent = "yes" doctype-system = "DTD/xhtml1-strict.dtd"
12         doctype-public = "-//W3C//DTD XHTML 1.0 Strict//EN"/>
13
14      <!-- include template for processing error elements -->
15      <xsl:include href = "/XSLT/XHTML/error.xsl"/>
16
17      <!-- template for product element -->
18      <xsl:template match = "product">
19         <html xmlns = "http://www.w3.org/1999/xhtml"
20               xml:lang = "en" lang = "en">
21
22         <head>
23            <title>
24               <xsl:value-of select = "title"/> -- Description
25            </title>
26
27            <link rel = "StyleSheet" href = "styles/default.css"/>
28         </head>
29
30         <body>
31
32            <!-- copy navigation header into XHTML document -->
33            <xsl:for-each select =
34               "document( '/XSLT/XHTML/navigation.xml' )">
35               <xsl:copy-of select = "."/>
```

Extract relevant pieces of information from the XML document to create appropriate XHTML representation

Distributed Systems Course.
See Deitels Book for details.

```
36            </xsl:for-each>
37
38            <div class = "header">
39               <xsl:value-of select = "title"/>
40            </div>
41
42            <div class = "author">
43               by <xsl:value-of select = "author"/>
44            </div>
45
46            <!-- create div element with details of Product -->
47            <div class = "productDetails">
48               <table style = "width: 100%;">
49                  <tr>
50                     <td style = "text-align: center;">
51                        <img class = "bookCover"
52                           src = "images/{image}"
53                           alt = "{title} cover image."/>
54                     </td>
55
56                     <td>
57                        <p style = "text-align: right;">
58                           Price: <xsl:value-of select = "price"/>
59                        </p>
60
61                        <p style = "text-align: right;">
62                           ISBN: <xsl:value-of select = "ISBN"/>
63                        </p>
64
65                        <p style = "text-align: right;">
66                           Pages: <xsl:value-of select = "pages"/>
67                        </p>
68
```

Distributed Systems Course.
See Deitels Book for details.

```
69                         <p style = "text-align: right;">
70                            Publisher:
71                            <xsl:value-of select = "publisher"/>
72                         </p>
73
74                         <!-- AddToCart button -->
75                         <form method = "post" action = "AddToCart">
76                            <p style = "text-align: center;">
77                               <input type = "submit"
78                                  value = "Add to cart"/>
79
80                               <input type = "hidden" name = "ISBN"
81                                  value = "{ISBN}"/>
82                            </p>
83                         </form>
84                      </td>
85                   </tr>
86                </table>
87             </div>
88
89          </body>
90          </html>
91       </xsl:template>
92    </xsl:stylesheet>
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "DTD/xhtml1-strict.dtd">
4    <html xmlns="http://www.w3.org/1999/xhtml"
5       lang="en" xml:lang="en">
6    <head>
7       <title>XML How to Program -- Description</title>
8       <link href="styles/default.css" rel="StyleSheet" />
9    </head>
10   <body>
11      <div>
12         <div class="logo">
13            <table style="width: 100%;">
14               <tr>
15                  <td style="text-align: left;">
16                     <img src="images/logotiny.gif"
17                        alt="Deitel &amp; Associates, Inc. logo." />
18                  </td>
19
20                  <td style="text-align: right;">
21                     <div style=
22                        "position: relative; bottom: -50px;">
23                        <form action="ProductSearch" method="get">
24                        <p><input type="text" size="15"
25                           name="searchString" />
26                           <input type="submit" value="Search" />
27                        </p>
28                        </form>
29                     </div>
30                  </td>
31               </tr>
32            </table>
33         </div>
34
```

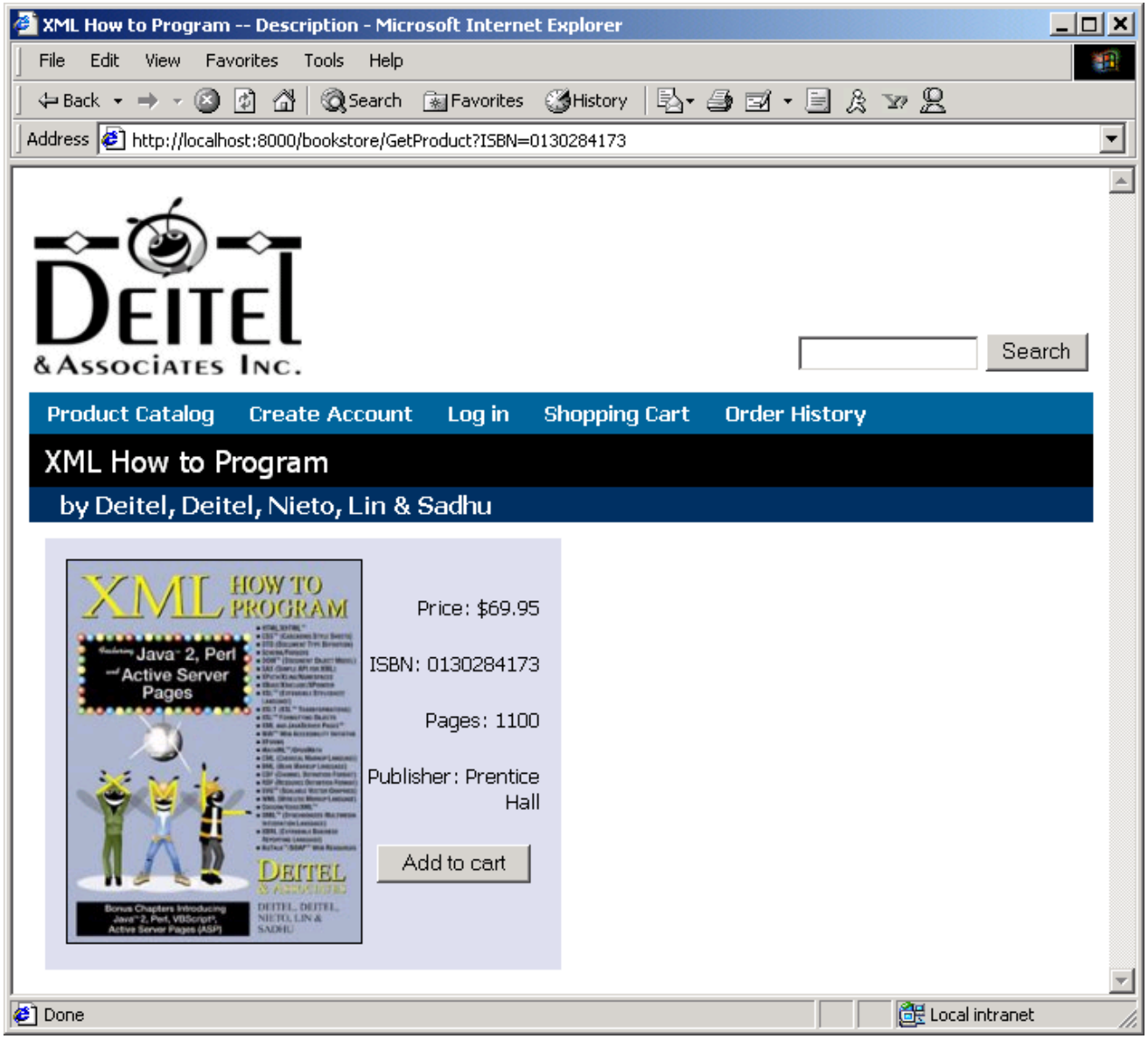
Distributed Systems Course.
See Deitels Book for details.

```
35          <div class="navigation">
36             <table class="menu">
37                <tr>
38                   <td class="menu">
39                      <a href="GetAllProducts">Product Catalog</a>
40                   </td>
41
42                   <td class="menu">
43                      <a href="registration.html">Create Account</a>
44                   </td>
45
46                   <td class="menu">
47                      <a href="login.html">Log in</a>
48                   </td>
49
50                   <td class="menu">
51                      <a href="ViewCart">Shopping Cart</a>
52                   </td>
53
54                   <td class="menu">
55                      <a href="ViewOrderHistory">Order History</a>
56                   </td>
57                </tr>
58             </table>
59          </div>
60
61       </div>
62       <div class="header">XML How to Program</div>
63       <div class="author">
64          by Deitel, Deitel, Nieto, Lin &amp; Sadhu</div>
65       <div class="productDetails">
66          <table style="width: 100%;">
67             <tr>
68                <td style="text-align: center;">
```

Distributed Systems Course.
See Deitels Book for details.

```
69                <img alt="XML How to Program cover image."
70                   src="images/xmlhtp1.jpg"
71                   class="bookCover" /></td>
72             <td>
73                <p style="text-align: right;">
74                   Price: $69.95</p>
75                <p style="text-align: right;">
76                   ISBN: 0130284173</p>
77                <p style="text-align: right;">
78                   Pages: 1100</p>
79                <p style="text-align: right;">
80                   Publisher: Prentice Hall</p>
81
82                <form action="AddToCart" method="post">
83                   <p style="text-align: center;">
84                      <input value="Add to cart"
85                         type="submit" />
86                      <input value="0130284173"
87                         name="ISBN" type="hidden" /></p>
88                </form>
89             </td>
90          </tr>
91       </table>
92    </div>
93  </body>
94 </html>
```

Distributed Systems Course.
See Deitels Book for details.



Distributed Systems Course.
See Deitels Book for details.

J2EE Summary and Benefits

- Java 2 Enterprise Edition
  - Portable application-server platform
- J2EE specification
  - API support
  - Security
  - Transaction management
  - Deployment processes

Commercial Application Servers

- Popular application servers
  - BEA WebLogic
  - iPlanet Application Server
  - IBM WebSphere
  - JBoss
  - Orbix E2A