```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ======================================================================
     Buildfile for Project 3, version 6.0
     Computer Science E-259

     This buildfile "shipped" in the root of the following hierarchy.

     project3-6.0/
       conf/
       src/
        cscie259/
          project3/
            wahoo/
       temp/
       webapps/
         ROOT/
           docs/
             cscie259/
               project3/
                 wahoo/
           dtd/
           images/
           WEB-INF/
           xml/
             cache/
           xsl/

     To compile Wahoo, execute ‘ant compile‘ or ‘ant‘ from within
     project3-6.0/.

     To generate Javadoc for your code (in project3-6.0/webapps/ROOT/docs/),
     execute ‘ant javadoc‘ from within project3-6.0/.

     To publish Javadoc for your code at
     http://www.people.fas.harvard.edu/~username/cscie259/javadoc/project3-6.0/,
     where username is your FAS username, execute ‘ant publish-javadoc‘
     from within project3-6.0/.

     To delete your bytecodes as well as Tomcat’s logs and runtime
     directories, execute ‘ant clean‘ from within project3-6.0/.
====================================================================== -->

<project name="project3" default="compile" basedir=".">

    <description>Project 3</description>

    <!-- set global properties for this build -->
    <property name="build" location="webapps/ROOT/WEB-INF/classes"/>
    <property name="conf" location="conf"/>
    <property name="docs" location="webapps/ROOT/docs"/>
    <property name="logs" location="logs"/>
    <property name="src" location="src"/>
    <property name="temp" location="temp"/>
    <property name="work" location="work"/>

    <!-- init -->
    <target name="init">

        <!-- set the standard DSTAMP, TSTAMP, and TODAY properties -->
        <!-- according to the default formats                      -->
        <tstamp/>

        <!-- Create the build directory structure used by compile -->
        <mkdir dir="${build}"/>

    </target>

    <!-- compile -->
    <target name="compile"
            depends="init"
            description="compile cscie259.project3.wahoo.*">
        <javac srcdir="${src}"
               destdir="${build}"
               debug="true"
               fork="true"
               includes="cscie259/project3/wahoo/*"
               listfiles="true"/>
    </target>

    <!-- javadoc -->
    <target name="javadoc"
            description="generate Javadoc">
        <javadoc packagenames="cscie259.project3.*"
                 sourcepath="${src}"
                 destdir="${docs}"
                 author="true"
                 version="true"
                 private="true"
                 nodeprecated="true"
                 windowtitle="Project 3"
                 header="Project 3"/>
    </target>

     <!-- publish javadoc -->
     <target name="publish-javadoc"
             depends="javadoc"
             description="publish Javadoc">
         <copy todir="${user.home}/public_html/cscie259/javadoc/project3-6.0">
              <fileset dir="${docs}"/>
         </copy>
         <chmod parallel="false"
                perm="a+rX"
                dir="${user.home}/public_html/cscie259"
                includes="**/*"
                type="both"/>
     </target>

    <!-- clean -->
    <target name="clean"
            description="remove class, log, temp, and work files">
        <delete dir="${build}"/>
        <delete dir="${conf}/Catalina"/>
        <delete dir="${logs}"/>
        <delete includeemptydirs="true">
            <fileset dir="${temp}" includes="**/*"/>
        </delete>
        <delete dir="${work}"/>
    </target>

</project>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ============================================================================
     Configuration file for Project 3, version 6.0
     Computer Science E-259

     Be sure to set the value of the Server element's port
     attribute to an integer between 1024 and 65535, inclusive.
     Of course, that port cannot already be in use.  Nor can it
     be the same value you choose for the Connector element's port.

     Also be sure to set the value of the Connector element's port
     attribute to an integer between 1024 and 65535, inclusive.
     Of course, that port cannot already be in use.  Nor can it
     be the same value you choose for the Server element's port.
============================================================================== -->


<!-- A "Server" is a singleton element that represents the entire JVM,
     which may contain one or more "Service" instances.  The Server
     listens for a shutdown command on the indicated port.

     Note:  A "Server" is not itself a "Container", so you may not
     define subcomponents such as "Valves" or "Loggers" at this level.
 -->

<Server port="" shutdown="SHUTDOWN">

  <!-- Global JNDI resources -->
  <GlobalNamingResources>

    <!-- Editable user database that can also be used by
         UserDatabaseRealm to authenticate users -->
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
       description="User database that can be updated and saved"
           factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
          pathname="conf/tomcat-users.xml" />

  </GlobalNamingResources>

  <!-- A "Service" is a collection of one or more "Connectors" that share
       a single "Container" (and therefore the web applications visible
       within that Container).  Normally, that Container is an "Engine",
       but this is not required.  -->

  <!-- Define the Tomcat Stand-Alone Service -->
  <Service name="Catalina">

    <!-- A "Connector" represents an endpoint by which requests are received
         and responses are returned.  Each Connector passes requests on to the
         associated "Container" (normally an Engine) for processing. -->

    <!-- Define a non-SSL HTTP/1.1 Connector -->
    <Connector port=""
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true" />

    <!-- An Engine represents the entry point (within Catalina) that processes
         every request.  The Engine implementation for Tomcat stand alone
         analyzes the HTTP headers included with the request, and passes them
         on to the appropriate Host (virtual host). -->

    <!-- Define the top level container in our container hierarchy -->
    <Engine name="Catalina" defaultHost="localhost">

      <!-- Because this Realm is here, an instance will be shared globally -->

      <!-- This Realm uses the UserDatabase configured in the global JNDI
           resources under the key "UserDatabase".  Any edits
           that are performed against this UserDatabase are immediately
           available for use by the Realm.  -->
      <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
             resourceName="UserDatabase"/>

      <!-- Define the default virtual host -->
      <Host name="localhost" appBase="webapps"
       unpackWARs="true" autoDeploy="true"
       xmlValidation="true" xmlNamespaceAware="true">

        <!-- Root Context -->
        <Context path="" docBase="ROOT" debug="0" reloadable="true">
          <!-- disable persistent sessions -->
          <Manager pathname=""/>
        </Context>

      </Host>

    </Engine>

  </Service>

</Server>
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.IOException;
 4: import java.io.PrintWriter;
 5:
 6: import javax.servlet.ServletException;
 7:
 8: import javax.servlet.http.HttpServletRequest;
 9: import javax.servlet.http.HttpServletResponse;
10: import javax.servlet.http.HttpSession;
11:
12: import javax.xml.transform.Transformer;
13: import javax.xml.transform.TransformerException;
14: import javax.xml.transform.TransformerFactory;
15:
16: import javax.xml.transform.stream.StreamResult;
17: import javax.xml.transform.stream.StreamSource;
18:
19:
20: /**
21:  * This servlet logs the user into the Wahoo portal.
22:  *
23:  * You MAY modify this file.
24:  *
25:  * @author  Computer Science E-259
26:  * @version 6.0
27:  *
28:  * @author  YOUR NAME GOES HERE
29:  **/
30:
31: public class Login extends WahooServlet
32: {
33:     /**
34:      * Presents user with login screen and processes login attempts
35:      * and account creations.
36:      *
37:      * @param request  HTTP request object
38:      * @param response HTTP response object
39:      *
40:      * @throws IOException      if a file-related error occurs
41:      * @throws ServletException if a servlet-related error occurs
42:      */
43:     protected void doWork(HttpServletRequest request,
44:                           HttpServletResponse response)
45:     throws IOException, ServletException
46:     {
47:         // check to see if a session already exists; if so, invalidate it
48:         HttpSession session = request.getSession(false);
49:         if (session != null)
50:         {
51:             // attempt to retrieve username from session
52:             String username = (String) session.getAttribute("username");
53:
54:             // if username is found, log user out (by invalidating session)
55:             if (username != null)
56:                 System.out.println("User " + username + " logged out");
57:             session.invalidate();
58:
59:             // display the login screen
60:             displayLoginForm(request, response, null);
61:
62:             return;
63:         }
64:
65:         // process form submission, else display login screen
66:         String operation = request.getParameter("submit");
67:         if (operation != null &&
68:             (operation.equals("Log In") || operation.equals("Register")))
69:         {
70:             // retrieve username from HTTP request
71:             String username = request.getParameter("username");
72:
73:             // if username missing, display the login screen with an
74:             // informative error message
75:             if (username == null || username.equals(""))
76:             {
77:                 displayLoginForm(request, response, "Please enter a username");
78:                 return;
79:             }
80:
81:             // retrieve password from HTTP request
82:             String password = request.getParameter("password");
83:
84:             // if password missing, display the login screen with an
85:             // informative error message
86:             if (password == null || password.equals(""))
87:             {
88:                 displayLoginForm(request, response, "Please enter a password");
89:                 return;
90:             }
91:
92:             // retrieve User object with given username; if password incorrect,
93:             // display the login screen with an informative error message
94:             if (operation.equals("Log In"))
95:             {
96:                 User user = UserManager.getUser(username);
97:                 if (user == null || !user.getPassword().equals(password))
98:                 {
99:                     displayLoginForm(request, response,
100:                                      "Invalid username or password");
101:                     return;
102:                 }
103:             }
104:             else
105:             {
106:                 // attempt create a new user, erring if desired username
107:                 // username is already in use
108:                 User user = UserManager.addUser(username, password);
109:                 if (user == null)
110:                 {
111:                     displayLoginForm(request, response,
112:                                      "Unable to add user " + username + "." +
113:                                      "Please try a different username.");
114:                     return;
115:                 }
116:
117:                 // save the new user!
118:                 UserManager.save();
119:             }
120:
121:             // report successful login
122:             System.out.println("User " + username + " logged in");
123:
124:             // establish a new session and store the username in it
```

```
125:             session = request.getSession(true);
126:             session.setAttribute("username", username);
127:
128:             // redirect user to view servlet
129:             redirect(response, "/servlet/view");
130:
131:             return;
132:         }
133:
134:         // else display the login screen by default
135:         else
136:         {
137:             displayLoginForm(request, response, null);
138:         }
139:     }
140:
141:
142:     /**
143:      * Displays the login form, with an optional error string.
144:      *
145:      * @param request  HTTP request object
146:      * @param response HTTP response object
147:      * @param error    optional error string
148:      *
149:      * @throws IOException      if an IO-related error occurs
150:      * @throws ServletException if a servlet-related error occurs
151:      */
152:     public void displayLoginForm(HttpServletRequest request,
153:                                  HttpServletResponse response,
154:                                  String error)
155:     throws IOException, ServletException
156:     {
157:         // output text/html MIME type
158:         response.setContentType("text/html");
159:
160:         // retrieve PrintWriter
161:         PrintWriter out = response.getWriter();
162:
163:         // attempt to apply login.xsl to dummy.xml
164:         try
165:         {
166:                         // prepare to process login.xsl by way of TrAX
167:             TransformerFactory tFactory = TransformerFactory.newInstance();
168:             StreamSource xslSource =
169:             new StreamSource(getServletContext().getRealPath("/xsl/login.xsl"));
170:             Transformer transformer = tFactory.newTransformer(xslSource);
171:
172:             // if there's an error message waiting to be displayed,
173:             // pass it to login.xsl as a stylesheet parameter
174:             if (error != null && !error.equals(""))
175:                 transformer.setParameter("error", error);
176:
177:             // apply login.xsl to dummy.xml by way of TrAX;
178:             // of course, a non-minimalist login screen might actually
179:             // make use of more interesting data (hint, hint)
180:             StreamSource xmlSource =
181:             new StreamSource(getServletContext().getRealPath("/xml/dummy.xml"));
182:             transformer.transform(xmlSource, new StreamResult(out));
183:         }
184:
185:         // catch any exceptions
186:         catch (TransformerException e)
187:         {
188:             e.printStackTrace();
189:             throw new ServletException("Can't process login.xsl: " +
190:                                        e.getMessage());
191:         }
192:     }
193: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.BufferedReader;
 4: import java.io.IOException;
 5: import java.io.InputStream;
 6: import java.io.InputStreamReader;
 7:
 8:
 9: /**
10:  * This class gets news headlines by connecting to Moreover.
11:  *
12:  * You MAY modify this file.
13:  *
14:  * @author  Computer Science E-259
15:  * @version 6.0
16:  *
17:  * @author  YOUR NAME GOES HERE
18:  **/
19:
20: public final class NewsProvider
21: {
22:     /**
23:      * Default constructor is private so that this utility
24:      * class cannot be instantiated.
25:      */
26:     private NewsProvider() {}
27:
28:
29:     /**
30:      * Returns an XML representation of the headlines for a given
31:      * category.
32:      *
33:      * @param category category for which to fetch headlines
34:      *
35:      * @return XML representation of headlines for given category
36:      *
37:      * @throws IOException if an IO-related error occurs
38:      */
39:     public static String getHeadlines(String category)
40:     throws IOException
41:     {
42:         // retrieve newsfeed from Moreover, taking care to escape URL
43:         // to the application/x-www-form-urlencoded MIME format
44:         System.out.println("Getting category = " + category);
45:         String urlStr = "http://p.moreover.com/cgi-local/page?c=" +
46:                         java.net.URLEncoder.encode(category, "UTF-8") +
47:                         "&o=xml";
48:         java.net.URL url = new java.net.URL(urlStr);
49:         InputStream in = url.openStream();
50:         BufferedReader reader = new BufferedReader(new InputStreamReader(in));
51:         String line;
52:         String xmlStr = "";
53:         boolean start = false;
54:
55:         // iterate through retrieved XML in order to grab root element
56:         // and its descendants
57:         do
58:         {
59:             line = reader.readLine();
60:             if (line != null)
61:             {
62:                 // This is a trick to eliminate the <?xml?> declaration and
63:                 // the DOCTYPE declaration in order that this be a legal
64:                 // XML fragment that can be embedded into another document!
65:                 if (line.indexOf("<moreovernews>") != -1)
66:                     start = true;
67:                 if (start)
68:                     xmlStr += line + "\n";
69:             }
70:         }
71:         while (line != null);
72:
73:         // return root element and its descendants as a String
74:         return xmlStr;
75:     }
76:
77: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.BufferedReader;
 4: import java.io.IOException;
 5: import java.io.InputStream;
 6: import java.io.InputStreamReader;
 7:
 8: import javax.servlet.ServletContext;
 9: import javax.servlet.ServletException;
10:
11: import javax.servlet.http.HttpServlet;
12: import javax.servlet.http.HttpServletRequest;
13: import javax.servlet.http.HttpServletResponse;
14:
15:
16: /**
17:  * This servlet displays personalized preferences in the Wahoo portal.
18:  *
19:  * You MUST modify this file.
20:  *
21:  * @author  Computer Science E-259
22:  * @version 6.0
23:  *
24:  * @author  YOUR NAME GOES HERE
25:  **/
26:
27: public class Prefs extends WahooServlet
28: {
29:     /**
30:      * a cached copy of the categories XML file
31:      */
32:     private String categoriesXML_ = "";
33:
34:
35:     /**
36:      * Called automatically by servlet container; initializes the servlet
37:      * and the categories list.
38:      *
39:      * @throws ServletException if a servlet-related error occurs
40:      */
41:     public void init()
42:     throws ServletException
43:     {
44:         // ensure UserManager is instantiated
45:         super.init();
46:
47:         // try to read categories into memory
48:         try
49:         {
50:             // grab categories from cached file
51:             ServletContext context = getServletContext();
52:             InputStream in
53:              = context.getResourceAsStream("/xml/cache/nested_category_list.xml");
54:             BufferedReader reader =
55:              new BufferedReader(new InputStreamReader(in));
56:             String line;
57:             boolean start = false;
58:
59:             // iterate through XML in order to grab root element
60:             // and its descendants
61:             do
62:             {
63:                 line = reader.readLine();
64:                 if (line != null)
65:                 {
66:                     // This is a trick to eliminate the <?xml?> declaration and
67:                     // the DOCTYPE declaration in order that this be a legal
68:                     // XML fragment that can be embedded into another document!
69:                     if (line.indexOf("<nested_category_list>") != -1)
70:                         start = true;
71:                     if (start)
72:                         categoriesXML_ += line + "\n";
73:                 }
74:             }
75:             while (line != null);
76:         }
77:
78:         // catch any error
79:         catch (IOException e)
80:         {
81:             throw new ServletException("Can't read categories: " +
82:                                        e.getMessage());
83:         }
84:     }
85:
86:
87:     /**
88:      * Displays and processes the preferences form; if user attempts to execute
89:      * this servlet before user's authenticated (via the login
90:      * servlet), should redirect user to login servlet.
91:      *
92:      * @param request  HTTP request object
93:      * @param response HTTP response object
94:      *
95:      * @throws IOException      if an IO-related error occurs
96:      * @throws ServletException if a servlet-related error occurs
97:      */
98:     public void doWork(HttpServletRequest request,
99:                        HttpServletResponse response)
100:    throws IOException, ServletException
101:    {
102:        // TODO
103:    }
104: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.util.LinkedList;
 4: import java.util.List;
 5:
 6:
 7: /**
 8:  * This class represents a single user of the Wahoo portal.
 9:  *
10:  * You MAY modify this file.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 6.0
14:  *
15:  * @author  YOUR NAME GOES HERE
16:  **/
17:
18: public class User
19: {
20:     /*
21:      * user's username
22:      */
23:     private String username_;
24:
25:
26:     /*
27:      * user's password
28:      */
29:      private String password_;
30:
31:
32:     /*
33:      * user's list of preferred categories
34:      */
35:      private List newsCategories_ = new LinkedList();
36:
37:
38:     /**
39:      * Creates a new User object, recording user's name and a password.
40:      *
41:      * @param username new user's username
42:      * @param password new user's password
43:      */
44:     public User(String username, String password)
45:     {
46:         username_ = username;
47:         password_ = password;
48:     }
49:
50:
51:     /**
52:      * Add a news category to the preferred list, if it not already in the
53:      * list.
54:      *
55:      * @param cat category to be added to user's preferred categories
56:      */
57:     public void addNewsCategory(String cat)
58:     {
59:         if (!newsCategories_.contains(cat))
60:             newsCategories_.add(cat);
61:     }
62:
63:
64:     /**
65:      * Get the user's preferred news categories.
66:      *
67:      * @return list of user's preferred news categories
68:      */
69:     public List getNewsCategories()
70:     {
71:         return newsCategories_;
72:     }
73:
74:
75:     /**
76:      * Get the password for this user.
77:      *
78:      * @return user's password
79:      */
80:     public String getPassword()
81:     {
82:         return password_;
83:     }
84:
85:
86:     /**
87:      * Get the username for this user.
88:      *
89:      * @return user's username
90:      */
91:     public String getUserName()
92:     {
93:         return username_;
94:     }
95:
96:
97:     /**
98:      * Removes a news category to the preferred list if it is on the list.
99:      *
100:     * @param cat category to be removed from user's preferred categories
101:     */
102:    public void removeNewsCategory(String cat)
103:    {
104:        newsCategories_.remove(cat);
105:    }
106:
107:
108:    /**
109:     * Saves the user's preferred news categories, overwriting previous
110:     * settings.
111:     *
112:     * @param newsCategories list of user's preferred news categories
113:     */
114:    public void setNewsCategories(List newsCategories)
115:    {
116:        newsCategories_ = newsCategories;
117:    }
118: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.FileNotFoundException;
 4: import java.io.FileOutputStream;
 5:
 6: import java.util.HashMap;
 7:
 8: import java.io.IOException;
 9: import java.io.InputStream;
10:
11: import java.util.Iterator;
12: import java.util.Map;
13: import java.util.Properties;
14:
15: import javax.servlet.ServletContext;
16:
17: import javax.xml.parsers.SAXParser;
18: import javax.xml.parsers.SAXParserFactory;
19:
20: import javax.xml.transform.OutputKeys;
21:
22: import org.apache.xml.serializer.Serializer;
23: import org.apache.xml.serializer.SerializerFactory;
24:
25: import org.xml.sax.Attributes;
26: import org.xml.sax.ContentHandler;
27: import org.xml.sax.InputSource;
28: import org.xml.sax.SAXException;
29:
30: import org.xml.sax.helpers.AttributesImpl;
31: import org.xml.sax.helpers.DefaultHandler;
32:
33:
34: /**
35:  * This class keeps track of users of the Wahoo portal by writing out
36:  * information to an xml file.
37:  *
38:  * You MAY modify this file.
39:  *
40:  * @author  Computer Science E-259
41:  * @version 6.0
42:  *
43:  * @author  YOUR NAME GOES HERE
44:  **/
45:
46: public final class UserManager
47: {
48:     /**
49:      * a reference to the one and only instance of this class
50:      */
51:     private static UserManager instance_;
52:
53:
54:     /**
55:      * a reference to the servlet context
56:      */
57:     private ServletContext context_;
58:
59:
60:     /**
61:      * a hash table of all users
62:      */
63:     private Map users_ = new HashMap();
64:
65:
66:     /**
67:      * A constructor that reads the user DB from an input stream.
68:      */
69:     private UserManager(ServletContext context)
70:     {
71:         context_ = context;
72:         readUsers(context_.getResourceAsStream("/xml/users.xml"));
73:     }
74:
75:
76:     /**
77:      * Must be called (e.g., by Login's init() method) once before
78:      * using any other method in class.
79:      *
80:      * @param context servlet's context object
81:      */
82:     public static synchronized void init(ServletContext context)
83:     {
84:         if (instance_ == null)
85:             instance_ = new UserManager(context);
86:     }
87:
88:
89:     /**
90:      * Adds a new user using the given username and password.
91:      * Returns the new User object if the operation succeeded or null if the
92:      * user could not be added because the username was already taken.
93:      *
94:      * @param username user's username
95:      * @param password user's password
96:      *
97:      * @return User object belonging to given username and password
98:      */
99:     public static synchronized User addUser(String username, String password)
100:    {
101:        return instance().addUserImpl(username, password);
102:    }
103:
104:
105:    /**
106:     * Adds a new user using the supplied username and password.
107:     * Returns the new User object if the operation succeeded or null if the
108:     * user could not be added becuase the username was already taken.
109:     *
110:     * @param username user's username
111:     * @param password user's password
112:     *
113:     * @return User object belonging to given username and password
114:     */
115:    private User addUserImpl(String username, String password)
116:    {
117:        // check to see if user already exists, returning immediately if so
118:        if (getUserImpl(username) != null)
119:            return null;
120:
121:        // add user to in-memory database
122:        System.out.print("Adding \"" + username + "\" to main memory with " +
123:                         "password \"" + password + "\"...  ");
124:        User user = new User(username, password);
```

```
125:         users_.put(username, user);
126:         System.out.println("Done.");
127:
128:         return user;
129:     }
130:
131:
132:     /**
133:      * Returns a User object for given username or null if one is not found.
134:      *
135:      * @param username user's username
136:      *
137:      * @return User object belonging to given username
138:      */
139:     public static synchronized User getUser(String username)
140:     {
141:         return instance().getUserImpl(username);
142:     }
143:
144:
145:     /**
146:      * Returns a User object for given username or null if one is not found.
147:      *
148:      * @param username user's username
149:      *
150:      * @return User object belonging to given username
151:      */
152:     private User getUserImpl(String username)
153:     {
154:         return (User)users_.get(username);
155:     }
156:
157:
158:     /**
159:      * Returns the one and only instance of this class, throwing an exception
160:      * if it has not been initialized.
161:      */
162:     private static UserManager instance()
163:     {
164:         if (instance_ == null)
165:             throw new RuntimeException("UserManager must be initialized " +
166:                                        "before use");
167:         return instance_;
168:     }
169:
170:
171:     /**
172:      * Read the user info from the input stream.
173:      *
174:      * @param in input stream
175:      */
176:     private void readUsers(InputStream in)
177:     {
178:         // try to parse users.xml, using internal UserReader class
179:         // as a ContentHandler, catching any errors
180:         try
181:         {
182:             SAXParserFactory factory = SAXParserFactory.newInstance();
183:             SAXParser parser = factory.newSAXParser();
184:             parser.parse(new InputSource(in), new UserReader());
185:         }
186:         catch (IOException e)
187:         {
188:             throw new RuntimeException("Error accessing user config file : " +
189:                                        e.getMessage());
190:         }
191:         catch (SAXException e)
192:         {
193:             throw new RuntimeException("Error parsing user config file: " +
194:                                        e.getMessage());
195:         }
196:         catch (Exception e)
197:         {
198:             e.printStackTrace();
199:         }
200:     }
201:
202:
203:     /**
204:      * Saves the information about all users to the information database
205:      * (stored as a file).  Call this methods whenever you have made changed
206:      * to User objects you would like to save.  If you do not call this method,
207:      * your chages will be lost next time you start up the server.
208:      */
209:     public static synchronized void save()
210:     {
211:         instance().writeUsers();
212:     }
213:
214:
215:     /**
216:      * Writes users info to the config file, overwriting old contents.
217:      */
218:     private void writeUsers()
219:     {
220:         // try to write out users
221:         try
222:         {
223:             // prepare to pretty-print (manually)
224:             char [] LF  = new char[1];
225:             char [] TAB = new char[1];
226:             LF[0]  = '\n';
227:             TAB[0] = '\t';
228:             String path = context_.getRealPath("/xml/users.xml");
229:             System.out.println("Writing user info to : " + path);
230:             FileOutputStream fos = new FileOutputStream(path);
231:             Properties format = new Properties();
232:             format.put(OutputKeys.ENCODING, "UTF-8");
233:             format.put(OutputKeys.INDENT, "no");
234:             format.put(OutputKeys.METHOD, "xml");
235:             Serializer serializer = SerializerFactory.getSerializer(format);
236:             serializer.setOutputStream(fos);
237:             ContentHandler handler = serializer.asContentHandler();
238:
239:             // start outputting
240:             handler.startDocument();
241:             AttributesImpl attributes = new AttributesImpl();
242:             handler.startElement(null, null, "users", attributes);
243:             handler.characters(LF, 0, 1);
244:
245:             // iterate over the users and write them out
246:             Iterator iter = users_.keySet().iterator();
247:             while (iter.hasNext())
248:             {
```

```
249:                 String username = (String)iter.next();
250:                 User user = (User)users_.get(username);
251:                 AttributesImpl atts = new AttributesImpl();
252:                 atts.addAttribute(null, null, "name", null,
253:                                   user.getUserName());
254:                 atts.addAttribute(null, null, "password", null,
255:                                   user.getPassword());
256:                 handler.characters(TAB, 0, 1);
257:                 handler.startElement(null, null, "user", atts);
258:                 handler.characters(LF, 0, 1);
259:
260:                 // iterate over the user's news categories and write them out
261:                 atts = new AttributesImpl();
262:                 handler.characters(TAB, 0, 1);
263:                 handler.characters(TAB, 0, 1);
264:                 handler.startElement(null, null, "news-categories", atts);
265:                 handler.characters(LF, 0, 1);
266:                 Iterator cats = user.getNewsCategories().iterator();
267:                 while (cats.hasNext())
268:                 {
269:                     String cat = (String)cats.next();
270:                     AttributesImpl atts2 = new AttributesImpl();
271:                     atts2.addAttribute(null, null, "name", null, cat);
272:                     handler.characters(TAB, 0, 1);
273:                     handler.characters(TAB, 0, 1);
274:                     handler.characters(TAB, 0, 1);
275:                     handler.startElement(null, null, "category", atts2);
276:                     handler.endElement(null, null, "category");
277:                     handler.characters(LF, 0, 1);
278:                 }
279:                 handler.characters(TAB, 0, 1);
280:                 handler.characters(TAB, 0, 1);
281:                 handler.endElement(null, null, "news-categories");
282:                 handler.characters(LF, 0, 1);
283:                 handler.characters(TAB, 0, 1);
284:                 handler.endElement(null, null, "user");
285:                 handler.characters(LF, 0, 1);
286:             }
287:
288:             // stop outputting
289:             handler.endElement(null, null, "users");
290:             handler.characters(LF, 0, 1);
291:             handler.endDocument();
292:         }
293:
294:         // catch any errors
295:         catch (FileNotFoundException e)
296:         {
297:             throw new RuntimeException("Can't open file for writing:: " +
298:                                        e.getMessage());
299:         }
300:         catch (IOException e)
301:         {
302:             throw new RuntimeException("Can't open file for writing:: " +
303:                                        e.getMessage());
304:         }
305:         catch (SAXException e)
306:         {
307:             throw new RuntimeException("Can't write out users: " +
308:                                        e.getMessage());
309:         }
310:     }
311:
312:
313:     /**
314:      * An inner class which is the SAX2 handler used to read the user
315:      * XML file into Java objects.
316:      */
317:     private class UserReader extends DefaultHandler
318:     {
319:         // reference to user being parsed
320:         private User curUser_;
321:
322:
323:         /**
324:          * SAX handler for elements
325:          *
326:          * @param namespaceURI element's namespace (unused)
327:          * @param localName element's local name (unused)
328:          * @param qName element's name
329:          * @param atts element's attributes
330:          *
331:          * @throws SAXException
332:          */
333:         public void startElement(String namespaceURI,
334:                                  String localName,
335:                                  String qName,
336:                                  Attributes atts)
337:         throws SAXException
338:         {
339:             // process user elements
340:             if (qName.equals("user"))
341:             {
342:                 String username = atts.getValue("name");
343:                 if (username == null || username.equals(""))
344:                     throw new SAXException("user element must have a " +
345:                                            "name attribute");
346:                 String password = atts.getValue("password");
347:                 if (password == null || password.equals(""))
348:                     throw new SAXException("user element must have a " +
349:                                            "password attribute");
350:                 curUser_ = addUserImpl(username, password);
351:                 if (curUser_ == null)
352:                     throw new SAXException("Can't add user: " +
353:                                            "username " + username + " " +
354:                                            "is already taken");
355:             }
356:
357:             // process category elements
358:             else if (qName.equals("category"))
359:             {
360:                 String name = atts.getValue("name");
361:                 if (name == null || name.equals(""))
362:                     throw new SAXException("category element must have a " +
363:                                            "name attribute");
364:                 System.out.print("Associating \"" + name +
365:                                  "\" with \"" + curUser_.getUserName() +
366:                                  "\"...  ");
367:                 curUser_.addNewsCategory(name);
368:                 System.out.println("Done.");
369:             }
370:         }
371:     }
372:
```

```
373: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.IOException;
 4:
 5: import javax.servlet.ServletException;
 6:
 7: import javax.servlet.http.HttpServletRequest;
 8: import javax.servlet.http.HttpServletResponse;
 9:
10:
11: /**
12:  * This servlet displays the main personalized view in the Wahoo portal.
13:  *
14:  * You MUST modify this file.
15:  *
16:  * @author  Computer Science E-259
17:  * @version 6.0
18:  *
19:  * @author  YOUR NAME GOES HERE
20:  **/
21:
22: public class View extends WahooServlet
23: {
24:     /**
25:      * Displays the current user's main view; if user attempts to execute
26:      * this servlet before user's authenticated (via the login
27:      * servlet), should redirect user to login servlet.
28:      *
29:      * @param request HTTP request object
30:      * @param response HTTP response object
31:      *
32:      * @throws IOException      if an IO-related error occurs
33:      * @throws ServletException if a servlet-related error occurs
34:      */
35:     public void doWork(HttpServletRequest request,
36:                        HttpServletResponse response)
37:     throws IOException, ServletException
38:     {
39:         // TODO
40:     }
41: }
```

```
 1: package cscie259.project3.wahoo;
 2:
 3: import java.io.IOException;
 4:
 5: import javax.servlet.http.HttpServlet;
 6: import javax.servlet.http.HttpServletRequest;
 7: import javax.servlet.http.HttpServletResponse;
 8:
 9: import javax.servlet.RequestDispatcher;
10: import javax.servlet.ServletException;
11:
12:
13: /**
14:  * Base class for Wahoo's servlets.
15:  *
16:  * You MAY modify this file.
17:  *
18:  * @author  Computer Science E-259
19:  * @version 6.0
20:  *
21:  * @author  YOUR NAME GOES HERE
22:  **/
23:
24: public abstract class WahooServlet extends HttpServlet
25: {
26:     /**
27:      * Common initialization tasks for this group of servlets.
28:      * Initializes the UserManager instance.
29:      *
30:      * @throws ServletException if a servlet-related error occurs
31:      */
32:     public void init()
33:     throws ServletException
34:     {
35:         UserManager.init(getServletContext());
36:     }
37:
38:
39:     /**
40:      * Responds to GETs in the same manner as POSTs.
41:      *
42:      * @param request HTTP request object
43:      * @param response HTTP response object
44:      *
45:      * @throws IOException      if an IO-related error occurs
46:      * @throws ServletException if a servlet-related error occurs
47:      */
48:     public void doGet(HttpServletRequest request,
49:                       HttpServletResponse response)
50:     throws IOException, ServletException
51:     {
52:         doWork(request, response);
53:     }
54:
55:
56:     /**
57:      * Responds to POSTs in the same manner as GETs.
58:      *
59:      * @param request HTTP request object
60:      * @param response HTTP response object
61:      *
62:      * @throws IOException      if an IO-related error occurs
63:      * @throws ServletException if a servlet-related error occurs
64:      */
65:     public void doPost(HttpServletRequest request,
66:                        HttpServletResponse response)
67:     throws IOException, ServletException
68:     {
69:         doWork(request, response);
70:     }
71:
72:
73:     /**
74:      * To be implemented by descendants, handles GETs and POSTs identically.
75:      *
76:      * @param request HTTP request object
77:      * @param response HTTP response object
78:      *
79:      * @throws IOException      if an IO-related error occurs
80:      * @throws ServletException if a servlet-related error occurs
81:      */
82:     protected abstract void doWork(HttpServletRequest request,
83:                                    HttpServletResponse response)
84:     throws IOException, ServletException;
85:
86:
87:     /**
88:      * Used to forward the request and response objects to another servlet
89:      * (whose path is of the form "/servlet/foo") for processing.
90:      *
91:      * @param request   HTTP request object
92:      * @param response  HTTP response object
93:      * @param path      the path to the servlet being forwarded to
94:      *
95:      * @throws IOException      if an IO-related error occurs
96:      * @throws ServletException if a servlet-related error occurs
97:      */
98:     public void forward(HttpServletRequest request,
99:                         HttpServletResponse response,
100:                        String path)
101:    throws IOException, ServletException
102:    {
103:        RequestDispatcher dispatch =
104:         request.getRequestDispatcher(path);
105:        dispatch.forward(request, response);
106:    }
107:
108:
109:    /**
110:     * Used to redirect the user to another url (or servlet in the
111:     * same container), without preserving the current
112:     * HttpServletRequest object.
113:     *
114:     * @param response  HTTP response object
115:     * @param url      the url to which the user's being redirected
116:     *
117:     * @throws IOException      if an IO-related error occurs
118:     */
119:    public void redirect(HttpServletResponse response, String url)
120:    throws IOException
121:    {
122:        response.sendRedirect(url);
123:    }
124: }
```

```
    1: <!ELEMENT moreovernews (article*)>
    2: <!ELEMENT article (url,headline_text,source,media_type,cluster,tagline,document_ur
l,harvest_time,access_registration,access_status)>
    3: <!ATTLIST article id ID #IMPLIED>
    4: <!ELEMENT url (#PCDATA)>
    5: <!ELEMENT headline_text (#PCDATA)>
    6: <!ELEMENT source (#PCDATA)>
    7: <!ELEMENT media_type (#PCDATA)>
    8: <!ELEMENT cluster (#PCDATA)>
    9: <!ELEMENT tagline (#PCDATA)>
   10: <!ELEMENT document_url (#PCDATA)>
   11: <!ELEMENT harvest_time (#PCDATA)>
   12: <!ELEMENT access_registration (#PCDATA)>
   13: <!ELEMENT access_status (#PCDATA)>
```

```
1: <!ELEMENT nested_category_list (channel*)>
2: <!ELEMENT channel (channel_name, category*)>
3: <!ELEMENT channel_name (#PCDATA)>
4: <!ELEMENT category (category_name, feed_name)>
5: <!ELEMENT category_name (#PCDATA)>
6: <!ELEMENT feed_name (#PCDATA)>
```

```
1: <!-- redirects user to Login servlet by default -->
2:
3: <% response.sendRedirect("/servlet/login"); %>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<!-- ============================================================================
     Configuration file for Project 3, version 6.0
     Computer Science E-259
============================================================================== -->

<web-app>

    <!-- servlet mappings -->
    <servlet>
        <servlet-name>login</servlet-name>
        <servlet-class>cscie259.project3.wahoo.Login</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>view</servlet-name>
        <servlet-class>cscie259.project3.wahoo.View</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>prefs</servlet-name>
        <servlet-class>cscie259.project3.wahoo.Prefs</servlet-class>
    </servlet>

</web-app>
```

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <dummy/>
```

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <users>
 3:         <user name="jharvard" password="crimson">
 4:                 <news-categories>
 5:                         <category name="Boston news"/>
 6:                         <category name="Top stories"/>
 7:                         <category name="Top technology stories"/>
 8:                 </news-categories>
 9:         </user>
10: </users>
```

```
    1: <?xml version="1.0" encoding="UTF-8"?>
    2:
    3: <!-- XSLT stylesheet for Wahoo!'s login page -->
    4: <xsl:stylesheet exclude-result-prefixes="xalan" version="1.0" xmlns="http://www.w3
.org/1999/xhtml" xmlns:xalan="http://xml.apache.org/xslt" xmlns:xsl="http://www.w3.org/19
99/XSL/Transform">
    5:
    6:   <!-- an optional error string -->
    7:   <xsl:param name="error" select="''"/>
    8:
    9:   <!-- output pretty-printed results as XHTML 1.0 -->
   10:   <xsl:output doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" doctype-syst
em="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" encoding="UTF-8" indent="yes
" method="xml" xalan:indent-amount="4"/>
   11:
   12:   <xsl:template match="/">
   13:     <html xml:lang="en" lang="en">
   14:       <head>
   15:         <title>Wahoo!</title>
   16:       </head>
   17:       <body onLoad="document.form.username.focus()">
   18:         <center>
   19:           <img src="/images/wahoo.gif" alt="Wahoo!"/>
   20:           <p/>
   21:           <form action="/servlet/login" method="POST" name="form">
   22:             <table>
   23:               <tr>
   24:                 <td width="40%">
   25:                   <strong>Login:</strong>
   26:                 </td>
   27:                 <td width="60%">
   28:                   <input type="text" name="username" value="" size="19" selected="
selected"/>
   29:                 </td>
   30:               </tr>
   31:               <tr>
   32:                 <td>
   33:                   <strong>Password:</strong>
   34:                 </td>
   35:                 <td>
   36:                   <input type="password" name="password" value="" size="19"/>
   37:                 </td>
   38:               </tr>
   39:               <xsl:if test="$error">
   40:                 <tr>
   41:                   <td colspan="2" align="center">
   42:                     <strong style="color:red">
   43:                       <xsl:value-of select="$error"/>
   44:                     </strong>
   45:                   </td>
   46:                 </tr>
   47:               </xsl:if>
   48:               <tr>
   49:                 <td colspan="2" align="center">
   50:                   <input name="submit" type="submit" value="Log In"/>  <
input name="submit" type="submit" value="Register"/>
   51:                 </td>
   52:               </tr>
   53:             </table>
   54:           </form>
   55:         </center>
   56:       </body>
   57:     </html>
   58:   </xsl:template>
   59: </xsl:stylesheet>
```

```
    1: <?xml version="1.0" encoding="UTF-8"?>
    2:
    3: <!-- XSLT stylesheet for Wahoo!’s preferences page -->
    4: <xsl:stylesheet version="1.0" xmlns:xalan="http://xml.apache.org/xslt" xmlns:xsl="
http://www.w3.org/1999/XSL/Transform">
    5:
    6:   <!-- output pretty-printed results as XHTML 1.0 -->
    7:   <xsl:output doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" doctype-syst
em="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" encoding="UTF-8" indent="yes
" method="xml" xalan:indent-amount="4"/>
    8:
    9:     <xsl:template match="/">
   10:         <html xml:lang="en" lang="en">
   11:             <head>
   12:                 <title>Wahoo!</title>
   13:             </head>
   14:             <body>
   15:                 <div align="center">
   16:
   17:                     <!-- Wahoo!’s logo -->
   18:                     <img alt="Wahoo’s logo" src="/images/wahoo.gif" />
   19:                     <br /><br />
   20:
   21:                     <!-- TODO -->
   22:
   23:                 </div>
   24:             </body>
   25:         </html>
   26:     </xsl:template>
   27:
   28: </xsl:stylesheet>
```

```
    1: <?xml version="1.0" encoding="UTF-8"?>
    2:
    3: <!-- XSLT stylesheet for Wahoo!’s main page -->
    4: <xsl:stylesheet version="1.0" xmlns:xalan="http://xml.apache.org/xslt" xmlns:xsl="
http://www.w3.org/1999/XSL/Transform">
    5:
    6:   <!-- output pretty-printed results as XHTML 1.0 -->
    7:   <xsl:output doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" doctype-syst
em="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" encoding="UTF-8" indent="yes
" method="xml" xalan:indent-amount="4"/>
    8:
    9:     <xsl:template match="/">
   10:         <html xml:lang="en" lang="en">
   11:             <head>
   12:                 <title>Wahoo!</title>
   13:             </head>
   14:             <body>
   15:                 <div align="center">
   16:
   17:                     <!-- Wahoo!’s logo -->
   18:                     <img alt="Wahoo’s logo" src="/images/wahoo.gif" />
   19:                     <br /><br />
   20:
   21:                     <!-- TODO -->
   22:
   23:                 </div>
   24:             </body>
   25:         </html>
   26:     </xsl:template>
   27:
   28: </xsl:stylesheet>
```