Java Server-side Programming

A Java Servlet E-shop Case Study - Continue

In this article, we shall continue from "A Java Servlet E-shop Case Study".

I shall assume that you are familiar with:

1. MySQL (read "How to install MySQL and Get Start with Java Database Programming").
2. Tomcat (read "How to install Tomcat and Get Started with Java Servlet Programming").
3. JDBC (read "JDBC Basic").
4. You have completed the "Java Servlet E-Shop Case Study".

1. Touching Up our E-Bookshop

Before we proceed, let us first touch up the e-bookshop that we have written earlier. We shall create a new webapp called "`yaebookshop`" (*yet another e-bookshop*).

I suggest that you use an IDE (with a graphic debugger), such as NetBeans or Eclipse, to develop our webapp, so as to improve your productivity and efficiency. Read:

- Eclipse: "Developing and Deploying Web Applications in Eclipse for Java EE".
- NetBeans: "Developing and Deploying Web Application in NetBeans".

1.1 Create a New Webapp "yaebookshop"

Without IDE (With Text Editor and JDK)

Create the following webapp directory structure for "`yaebookshop`" under Tomcat's "`webapps`" directory:

1. **`webapps\yaebookshop`**: The *context root* for the webapp "`yaebookshop`". Contains resources visible to the web users, such as HTML, CSS, images and scripts.
2. **`webapps\yaebookshop\WEB-INF`**: Hidden directory for protected resources of this webapp. Contains the web application deployment descriptor "`web.xml`", and,
   a. **`webapps\yaebookshop\WEB-INF\src`**: Keeps the Java Source file.
   b. **`webapps\yaebookshop\WEB-INF\classes`**: Keeps the Java classes.
   c. **`webapps\yaebookshop\WEB-INF\lib`**: Keep the external library's JAR-files, e.g., the MySQL JDBC Driver.
3. **`webapps\yaebookshop\META-INF`**: Hidden directory for server-related resource specific to the server (such as Tomcat, Glassfish), e.g., configuration file "`context.xml`". In contrast, "`WEB-INF`" is for resources independent of the web server.

Eclispe-JaveEE (v2021-12)

Reference: "Developing and Deploying Web Applications in Eclipse for Java EE".

Create a webapp called "`yaebookshop`":

1. Launch Eclipse-JavaEE, choose a workspace directory (any directory of your choice but NOT the Tomcat's "`webapps`" directory).
2. From "File" menu ⇒ New ⇒ Othes ⇒ Web ⇒ Dynamic Web Project ⇒ Next ⇒ Under the Project Name: enter "yabookshop" ⇒ Next ⇒ Next ⇒ Observe that "Context Root" is "yabookshop", "Content directory" is "WebContent" ⇒ Finish.

NetBeans-JavaEE (v7)

Create a web application called "`yaebookshop`":

1. From "File" menu ⇒ choose "New Project".
2. In "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
3. In "Name and Location" ⇒ In "Project Name", enter "`yaebookshop`" ⇒ In "Project Location", select a suitable directory to save your works ⇒ In Project Folder, use the default ⇒ Check "Set as Main Project" ⇒ Next.
4. In "Server and settings" ⇒ In "Server", select your Tomcat server, or "add" a new Tomcat server ⇒ In "Java EE Version", choose the latest such as Java EE 6 Web ⇒ In "Context Path", use the default "`/yaebookshop`" ⇒ Next.
5. In "Frameworks" ⇒ Select none for pure servlet/JSP application ⇒ Finish.

1.2 Setup the Database

We shall use the same database as the basic case study.

```
Database: ebookshop
Table: books
+-------+---------------------------+---------------+---------+-------+
| id    | title                     | author        | price   | qty   |
| (INT) | (VARCHAR(50))             | (VARCHAR(50)) | (FLOAT) | (INT) |
+-------+---------------------------+---------------+---------+-------+
| 1001  | Java for dummies          | Tan Ah Teck   | 11.11   | 11    |
| 1002  | More Java for dummies     | Tan Ah Teck   | 22.22   | 22    |
| 1003  | More Java for more dummies| Mohammad Ali  | 33.33   | 33    |
| 1004  | A Cup of Java             | Kumar         | 44.44   | 44    |
| 1005  | A Teaspoon of Java        | Kevin Jones   | 55.55   | 55    |
+-------+---------------------------+---------------+---------+-------+
```

```
Database: ebookshop
Table: order_records
+-------+-------------+---------------+---------------+------------+
| id    | qty_ordered | cust_name     | cust_email    | cust_phone |
| (INT) | (INT)       | (VARCHAR(30)) | (VARCHAR(30)) | CHAR(8)    |
+-------+-------------+---------------+---------------+------------+
```

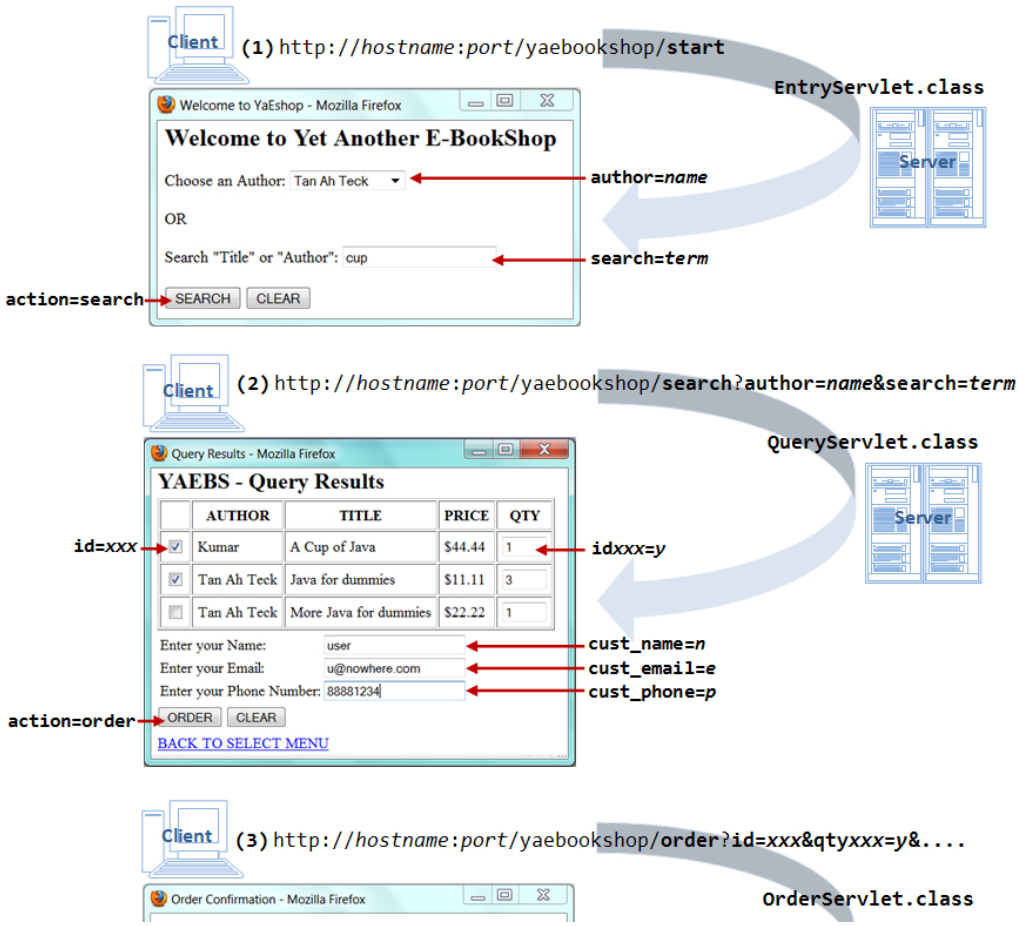
You can create the database by running the following SQL script:

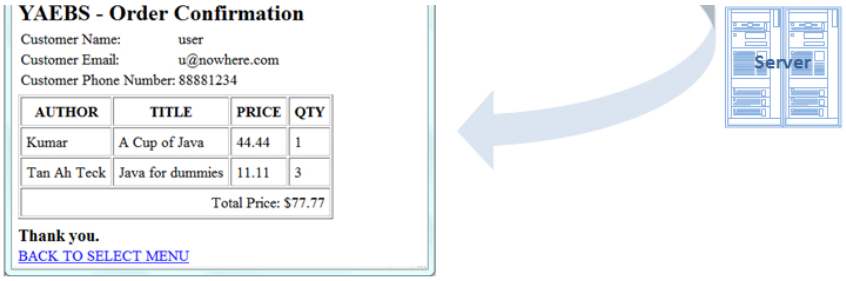
```
create database if not exists ebookshop;

use ebookshop;

drop table if exists books;
create table books (
  id     int,
  title  varchar(50),
  author varchar(50),
  price  float,
  qty    int,
  primary key (id));

insert into books values (1001, 'Java for dummies', 'Tan Ah Teck', 11.11, 11);
insert into books values (1002, 'More Java for dummies', 'Tan Ah Teck', 22.22, 22);
insert into books values (1003, 'More Java for more dummies', 'Mohammad Ali', 33.33, 33);
insert into books values (1004, 'A Cup of Java', 'Kumar', 44.44, 44);
insert into books values (1005, 'A Teaspoon of Java', 'Kevin Jones', 55.55, 55);

drop table if exists order_records;
create table order_records (
  id          int,
  qty_ordered int,
  cust_name   varchar(30),
  cust_email  varchar(30),
  cust_phone  char(8));

select * from books;
```

1.3 Sequence of Events

The sequence of events is as follows:

1. A client can start the webapp by issuing URL `http://hostname:port/yaebookshop/start`, which is mapped to "`EntryServlet`". The servlet outputs an HTML form of a search menu. The form is to be submitted to URL "`/search`", with request parameters `author=name` and `search=term`.
2. The URL "`/search`" maps to "`QueryServlet`", which retrieves the request parameters `author=name` and `search=term`, queries the database, and outputs the list of books that meets the query criteria in an HTML form. Each book is identified by a checkbox (with `name=value` pair of `id=xxx`) and a quantity text field (with name=value pair of `idxxx=qty`, where *xxx* is the book's `id`). The form is to be submitted to URL "`/order`".
3. The URL "`/order`" maps to "`OrderServlet`", which retrieves the book's id and quantity ordered, and update the table books, by reducing the quantity available. It also create a transaction record in a table `order_records`.





1.4 Entry Page - "EntryServlet" (with URL "\start")

The entry servlet (called "`EntryServlet`" with URL "`\start`") prints an HTML form with a pull-down menu and a text field for users to issue query. It populates the pull-down menu with all the available authors, by querying the database. It also provides a text field for users to enter a search term to search for the desired title or author with pattern matching. Take note that the entry point of this webapp is a servlet with URL `http://hostname:port/yaebookshop/start`, instead of an HTML page as in the previous exercises.

"EntryServlet.java"

For proper deployment, Java classes shall be kept in a named package (instead of the default *no-name* package). Let's call our package "`mypkg`".

Create a sub-directory called "`mypkg`" under "`WEB-INF\src`", and save the "`EntryServlet.java`" under "`mypkg`".

(NetBeans) Expand on your project node ⇒ Right-click on "Source Packages" ⇒ New ⇒ Others ⇒ In "Categories", select "Web" ⇒ In "File Types", select "Servlet" ⇒ In "Class Name", enter "`EntryServlet`" ⇒ In "Package", enter "`mypkg`" ⇒ Next ⇒ Check "Add Information to deployment descriptor (web.xml)" ⇒ In "URL Pattern", enter "`/start`" ⇒ "Finish". (Alternatively, you can first create a new "package" called "`mypkg`" and then create the Java Servlet.)

```
1  package mypkg;
2
3  import java.io.*;
4  import java.sql.*;
5  import java.util.logging.*;
6  import javax.servlet.*;
7  import javax.servlet.http.*;
8
9  public class EntryServlet extends HttpServlet {
10
11    private String databaseURL, username, password;
12
13    @Override
14    public void init(ServletConfig config) throws ServletException {
15       // Retrieve the database-URL, username, password from webapp init parameters
16       super.init(config);
17       ServletContext context = config.getServletContext();
18       databaseURL = context.getInitParameter("databaseURL");
19       username = context.getInitParameter("username");
20       password = context.getInitParameter("password");
21    }
22
23    @Override
24    protected void doGet(HttpServletRequest request, HttpServletResponse response)
25            throws ServletException, IOException {
26       response.setContentType("text/html;charset=UTF-8");
27       PrintWriter out = response.getWriter();
28
29       Connection conn = null;
30       Statement stmt = null;
31       try {
32          conn = DriverManager.getConnection(databaseURL, username, password);
33          stmt = conn.createStatement();
34          String sqlStr = "SELECT DISTINCT author FROM books WHERE qty > 0";
35          // System.out.println(sqlStr);  // for debugging
36          ResultSet rset = stmt.executeQuery(sqlStr);
37
38          out.println("<html><head><title>Welcome to YaEshop</title></head><body>");
39          out.println("<h2>Welcome to Yet Another E-BookShop</h2>");
40          // Begin an HTML form
41          out.println("<form method='get' action='search'>");
42
43          // A pull-down menu of all the authors with a no-selection option
44          out.println("Choose an Author: <select name='author' size='1'>");
45          out.println("<option value=''>Select...</option>");  // no-selection
46          while (rset.next()) {  // list all the authors
47             String author = rset.getString("author");
48             out.println("<option value='" + author + "'>" + author + "</option>");
49          }
50          out.println("</select><br />");
51          out.println("<p>OR</p>");
52
53          // A text field for entering search word for pattern matching
54          out.println("Search \"Title\" or \"Author\": <input type='text' name='search' />");
55
56          // Submit and reset buttons
57          out.println("<br /><br />");
58          out.println("<input type='submit' value='SEARCH' />");
59          out.println("<input type='reset' value='CLEAR' />");
60          out.println("</form>");
61
```

```
62          out.println("</body></html>");
63       } catch (SQLException ex) {
64          out.println("<h3>Service not available. Please try again later!</h3></body></html>");
65          Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
66       } finally {
67          out.close();
68          try {
69             if (stmt != null) stmt.close();
70             if (conn != null) conn.close();
71          } catch (SQLException ex) {
72             Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
73          }
74       }
75    }
76
77    @Override
78    protected void doPost(HttpServletRequest request, HttpServletResponse response)
79            throws ServletException, IOException {
80       doGet(request, response);
81    }
82 }
```

Dissecting the Program:

1. In Line 1, we place the source file in package "`mypkg`" to facilitate deployment.
   (If you are using JDK with CMD shell) As the source file is saved under "`WEB-INF\src\mypkg`", we need to use the following command to compile the source and place the resultant class in "`WEB-INF\classes\mypkg`":

   ```
   // change directory to "yaebooshop\WEB-INF\classes"
   > javac -d . ..\src\EntryServlet.java
   ```

2. Instead of hard-coding the database-URL, username and password in the `getConnection()` method, we shall keep them in the web application's initialization parameters. We will configure the init parameters later in "`web.xml`". They are available to all the servlets under this web application (i.e., having application scope). In the `init()` method, which runs once when the servlet is loaded into the container, we retrieve the `ServletContext` via `ServletConfig`, and then retrieve the initialization parameters.
3. In the `doGet()` method, which runs once per user request, we print an HTML form, consisting of a pull-down menu of authors (with request parameter name of "`author=`*`name`*") and a search text field (with request parameter name of "`search=`*`term`*"). The list of author is obtained via a database query.
4. The form will be submitted to a URL '`/query`", using GET request method. In production, we shall change to POST request, with `doPost()` re-directed to `doGet()`. The URL triggered shall be:

   ```
   http://hostname:port/yaebookshop/query?author=name&search=term
   ```

5. I also replace the `printStackTrace()` with proper logging framework, for production use.

Application Deployment Descriptor "`web.xml`"

(NetBeans) You can create "`web.xml`" by right-click on the project node ⇒ New ⇒ Others ⇒ In "Categories", select "Web" ⇒ In "File Types", select "Standard Deployment Descriptor (web.xml)". The "`web.xml`" is available under the "Configuration Files" node.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="3.0"
3      xmlns="http://java.sun.com/xml/ns/javaee"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
6
7    <context-param>
8      <param-name>databaseURL</param-name>
9      <param-value>jdbc:mysql://localhost:3306/ebookshop</param-value>
10   </context-param>
11   <context-param>
12     <param-name>username</param-name>
13     <param-value>myuser</param-value>
14   </context-param>
15   <context-param>
16     <param-name>password</param-name>
17     <param-value>xxxx</param-value>
18   </context-param>
19
20   <servlet>
21     <servlet-name>EntryServlet</servlet-name>
22     <servlet-class>mypkg.EntryServlet</servlet-class>
23   </servlet>
24
25   <servlet-mapping>
26     <servlet-name>EntryServlet</servlet-name>
27     <url-pattern>/start</url-pattern>
28   </servlet-mapping>
29
30   <session-config>
31     <session-timeout>30</session-timeout>
32   </session-config>
33   <welcome-file-list>
34     <welcome-file>start</welcome-file>
35   </welcome-file-list>
36 </web-app>
```

Notes:

1. This "`web.xml`" contains webapp's initialization parameters (namely, database-URL, username and password), which are available to all servlets under this webapp (i.e., having applications scope). They are make available via the `ServletContext` object.
2. We configure request URL "`/start`" to "`mypkg.EntryServlet.class`".
3. We also set the "`start`" as a welcome file. In other word, directory request to `http://hostname:port/yaebookshop` will also start the application.

5. We also set the "start" JSP as the welcome file. In other words, a directory request to http://localhost:port/yaebookshop will also start the application.

1.5 Handling the Query - "QueryServlet" (with URL "/query")

"QueryServlet.java"

```
package mypkg;

import java.io.*;
import java.sql.*;
import java.util.logging.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class QueryServlet extends HttpServlet {

   private String databaseURL, username, password;

   @Override
   public void init(ServletConfig config) throws ServletException {
      super.init(config);
      ServletContext context = config.getServletContext();
      databaseURL = context.getInitParameter("databaseURL");
      username = context.getInitParameter("username");
      password = context.getInitParameter("password");
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      Connection conn = null;
      Statement stmt = null;

      try {
         // Retrieve and process request parameters: "author" and "search"
         String author = request.getParameter("author");
         boolean hasAuthorParam = author != null && !author.equals("Select...");
         String searchWord = request.getParameter("search");
         boolean hasSearchParam = searchWord != null && ((searchWord = searchWord.trim()).length() > 0);

         out.println("<html><head><title>Query Results</title></head><body>");
         out.println("<h2>YAEBS - Query Results</h2>");

         if (!hasAuthorParam && !hasSearchParam) {  // No params present
            out.println("<h3>Please select an author or enter a search term!</h3>");
            out.println("<p><a href='start'>Back to Select Menu</a></p>");
         } else {
            conn = DriverManager.getConnection(databaseURL, username, password);
            stmt = conn.createStatement();

            // Form a SQL command based on the param(s) present
            StringBuilder sqlStr = new StringBuilder();  // more efficient than String
            sqlStr.append("SELECT * FROM books WHERE qty > 0 AND (");
            if (hasAuthorParam) {
               sqlStr.append("author = '").append(author).append("'");
            }
            if (hasSearchParam) {
               if (hasAuthorParam) {
                  sqlStr.append(" OR ");
               }
               sqlStr.append("author LIKE '%").append(searchWord)
                     .append("%' OR title LIKE '%").append(searchWord).append("%'");
            }
            sqlStr.append(") ORDER BY author, title");
            //System.out.println(sqlStr);  // for debugging
            ResultSet rset = stmt.executeQuery(sqlStr.toString());

            if (!rset.next()) {  // Check for empty ResultSet (no book found)
               out.println("<h3>No book found. Please try again!</h3>");
               out.println("<p><a href='start'>Back to Select Menu</a></p>");
            } else {
               // Print the result in an HTML form inside a table
               out.println("<form method='get' action='order'>");
               out.println("<table border='1' cellpadding='6'>");
               out.println("<tr>");
               out.println("<th> </th>");
               out.println("<th>AUTHOR</th>");
               out.println("<th>TITLE</th>");
               out.println("<th>PRICE</th>");
               out.println("<th>QTY</th>");
               out.println("</tr>");

               // ResultSet's cursor now pointing at first row
               do {
                  // Print each row with a checkbox identified by book's id
                  String id = rset.getString("id");
                  out.println("<tr>");
                  out.println("<td><input type='checkbox' name='id' value='" + id + "' /></td>");
```

```
                    out.println("<td>" + rset.getString("author") + "</td>");
                    out.println("<td>" + rset.getString("title") + "</td>");
                    out.println("<td>$" + rset.getString("price") + "</td>");
                    out.println("<td><input type='text' size='3' value='1' name='qty" + id + "' /></td>");
                    out.println("</tr>");
                } while (rset.next());
                out.println("</table><br />");

                // Ask for name, email and phone using text fields (arranged in a table)
                out.println("<table>");
                out.println("<tr><td>Enter your Name:</td>");
                out.println("<td><input type='text' name='cust_name' /></td></tr>");
                out.println("<tr><td>Enter your Email (user@host):</td>");
                out.println("<td><input type='text' name='cust_email' /></td></tr>");
                out.println("<tr><td>Enter your Phone Number (8-digit):</td>");
                out.println("<td><input type='text' name='cust_phone' /></td></tr></table><br />");

                // Submit and reset buttons
                out.println("<input type='submit' value='ORDER' />");
                out.println("<input type='reset' value='CLEAR' /></form>");

                // Hyperlink to go back to search menu
                out.println("<p><a href='start'>Back to Select Menu</a></p>");
             }
          }
          out.println("</body></html>");
       } catch (SQLException ex) {
          out.println("<h3>Service not available. Please try again later!</h3></body></html>");
          Logger.getLogger(QueryServlet.class.getName()).log(Level.SEVERE, null, ex);
       } finally {
          out.close();
          try {
             if (stmt != null) stmt.close();
             if (conn != null) conn.close();
          } catch (SQLException ex) {
             Logger.getLogger(QueryServlet.class.getName()).log(Level.SEVERE, null, ex);
          }
       }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       doGet(request, response);
    }
}
```

Dissecting the Program:

1. This servlet retrieve the query parameters `author` and `search` to form a SQL SELECT query. Take note that many lines of codes are used to check the validity of input parameters (which could be simplified by other techniques). The SQL SELECT statement is as follows. We use pattern matching (LIKE operator) to handle the search term.

   ```
   SELECT * FROM books
     WHERE qty > 0
     AND (author = 'name' OR author LIKE '%term%' OR title LIKE '%term%')
   ```

2. The servlet outputs an HTML form, listing all the books selected, with a checkbox for ordering and a text field for entering the quantity. The `name=value` pair of the checkbox is the `id=xxx`; whereas the `name=value` pair of the quantity text field is `qty+id=qtyOrdered`, for example, `id=1001` and `qty1001=5`.
3. It also prints three text fields to prompt for the customer's name, email and phone number, with `name` attribute of `cust_name`, `cust_email` and `cust_phone`, respectively.
4. The form is to be submitted to URL "`/order`" using GET request (shall change to POST for production).

Application Deployment Descriptor "`web.xml`"

```
<servlet>
  <servlet-name>QueryServlet</servlet-name>
  <servlet-class>mypkg.QueryServlet</servlet-class>
</servlet>

......

<servlet-mapping>
  <servlet-name>QueryServlet</servlet-name>
  <url-pattern>/search</url-pattern>
</servlet-mapping>
```

1.6 Handling the Order - "`OrderServlet`" (with URL "`/order`")

"`OrderServlet.java`"

```
1   package mypkg;
2
3   import java.io.*;
4   import java.sql.*;
5   import java.util.logging.*;
6   import javax.servlet.*;
7   import javax.servlet.http.*;
8
9   public class OrderServlet extends HttpServlet {
10
11     private String databaseURL, username, password;
12
```

```
    @Override
    public void init(ServletConfig config) throws ServletException {
       super.init(config);
       ServletContext context = config.getServletContext();
       databaseURL = context.getInitParameter("databaseURL");
       username = context.getInitParameter("username");
       password = context.getInitParameter("password");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();

       Connection conn = null;
       Statement  stmt = null;
       ResultSet  rset = null;
       String     sqlStr = null;

       try {
          out.println("<html><head><title>Order Confirmation</title></head><body>");
          out.println("<h2>YAEBS - Order Confirmation</h2>");

          // Retrieve and process request parameters: id(s), cust_name, cust_email, cust_phone
          String[] ids = request.getParameterValues("id");  // Possibly more than one values
          String custName = request.getParameter("cust_name");
          boolean hasCustName = custName != null && ((custName = custName.trim()).length() > 0);
          String custEmail = request.getParameter("cust_email").trim();
          boolean hasCustEmail = custEmail != null && ((custEmail = custEmail.trim()).length() > 0);
          String custPhone = request.getParameter("cust_phone").trim();
          boolean hasCustPhone = custPhone != null && ((custPhone = custPhone.trim()).length() > 0);

          // Validate inputs
          if (ids == null || ids.length == 0) {
             out.println("<h3>Please Select a Book!</h3>");
          } else if (!hasCustName) {
             out.println("<h3>Please Enter Your Name!</h3>");
          } else if (!hasCustEmail || (custEmail.indexOf('@') == -1)) {
             out.println("<h3>Please Enter Your e-mail (user@host)!</h3>");
          } else if (!hasCustPhone || (custPhone.length() != 8)) {
             out.println("<h3>Please Enter an 8-digit Phone Number!</h3>");
          } else {
             // Display the name, email and phone (arranged in a table)
             out.println("<table>");
             out.println("<tr><td>Customer Name:</td><td>" + custName + "</td></tr>");
             out.println("<tr><td>Customer Email:</td><td>" + custEmail + "</td></tr>");
             out.println("<tr><td>Customer Phone Number:</td><td>" + custPhone + "</td></tr></table>");

             conn = DriverManager.getConnection(databaseURL, username, password);
             stmt = conn.createStatement();

             // Print the book(s) ordered in a table
             out.println("<br />");
             out.println("<table border='1' cellpadding='6'>");
             out.println("<tr><th>AUTHOR</th><th>TITLE</th><th>PRICE</th><th>QTY</th></tr>");

             float totalPrice = 0f;
             for (String id : ids) {
                sqlStr = "SELECT * FROM books WHERE id = " + id;
                //System.out.println(sqlStr);  // for debugging
                rset = stmt.executeQuery(sqlStr);

                // Expect only one row in ResultSet
                rset.next();
                int qtyAvailable = rset.getInt("qty");
                String title = rset.getString("title");
                String author = rset.getString("author");
                float price = rset.getFloat("price");

                int qtyOrdered = Integer.parseInt(request.getParameter("qty" + id));
                sqlStr = "UPDATE books SET qty = qty - " + qtyOrdered + " WHERE id = " + id;
                //System.out.println(sqlStr);  // for debugging
                stmt.executeUpdate(sqlStr);

                sqlStr = "INSERT INTO order_records values ("
                        + id + ", " + qtyOrdered + ", '" + custName + "', '"
                        + custEmail + "', '" + custPhone + "')";
                //System.out.println(sqlStr);  // for debugging
                stmt.executeUpdate(sqlStr);

                // Display this book ordered
                out.println("<tr>");
                out.println("<td>" + author + "</td>");
                out.println("<td>" + title + "</td>");
                out.println("<td>" + price + "</td>");
                out.println("<td>" + qtyOrdered + "</td></tr>");
                totalPrice += price * qtyOrdered;
             }

```

```
            out.println("<tr><td colspan='4' align='right'>Total Price: $");
            out.printf("%.2f</td></tr>", totalPrice);
            out.println("</table>");

            out.println("<h3>Thank you.</h3>");
            out.println("<p><a href='start'>Back to Select Menu</a></p>");
         }
         out.println("</body></html>");
      } catch (SQLException ex) {
         out.println("<h3>Service not available. Please try again later!</h3></body></html>");
         Logger.getLogger(OrderServlet.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
         out.close();
         try {
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();
         } catch (SQLException ex) {
            Logger.getLogger(OrderServlet.class.getName()).log(Level.SEVERE, null, ex);
         }
      }
   }

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      doGet(request, response);
   }
}
```

Dissecting the Program:

1. The servlet retrieves the request parameters, such as `id=1001`, `qty1001=1`, `id=1002`, `qty1002=2` and update the table books by reducing the quantity available. It also insert a transaction records in the table `order_records`.
2. [TOOD] more

Application Deployment Descriptor "`web.xml`"

```
<servlet>
  <servlet-name>OrderServlet</servlet-name>
  <servlet-class>mypkg.OrderServlet</servlet-class>
</servlet>

......

<servlet-mapping>
  <servlet-name>OrderServlet</servlet-name>
  <url-pattern>/order</url-pattern>
</servlet-mapping>
```

1.7 Rewrite the "`OrderServlet`" to do More Checking

I rewrite the "`OrderServlet`" to do more checking on inputs and handle the abnormal conditions:

1. The phone number must be exactly 8 numeric digits.
2. The quantity ordered shall be less than or equal to quantity available.
3. Replace special HMTL characters >, <, & and " with escape sequences in request parameters `cust_name`, `cust_email`.
4. I also disable the auto-commit, which commit every SQL statement. Instead, a commit is issued only if the entire order (i.e., all books) can be met. Partial update will be roll-backed.
5. [TODO]

Helper Utility "`myutil.InputFilter`"

The utility class `InputFilter` provide these `static` methods to filter or verify the inputs entered by the client.

1. `htmlFilter`: replace special HTML characters >, <, & and " with escape sequences.
2. `isValidPhone`: Check if the given input string is a valid phone number.
3. `parsePositiveInt`: Parse the given input string to a positive integer or zero otherwise. Useful for checking the quantity ordered.

```
package myutil;

public final class InputFilter {
   /**
    * Filter the specified message string for characters that are sensitive
    * in HTML.  This avoids potential attacks caused by including JavaScript
    * codes in the request URL that is often reported in error messages.
    */
   public static String htmlFilter(String message) {
      if (message == null) return null;
      int len = message.length();
      StringBuilder result = new StringBuilder(len + 20);
      char aChar;

      for (int i = 0; i < len; ++i) {
         aChar = message.charAt(i);
         switch (aChar) {
         case '<':
             result.append("&lt;");
             break;
         case '>':
             result.append("&gt;");
             break;
```

```
24          case '&':
25              result.append("&amp;");
26              break;
27          case '"':
28              result.append("&quot;");
29              break;
30          default:
31              result.append(aChar);
32          }
33       }
34       return (result.toString());
35    }
36
37    /**
38     *  Given a phone number string, return true if it is an 8-digit number
39     */
40    public static boolean isValidPhone(String phoneNumber) {
41       if (phoneNumber.length() != 8) {
42          return false;
43       }
44       for (int i = 0; i < phoneNumber.length(); ++i) {
45          char c = phoneNumber.charAt(i);
46          if (c < '0' || c > '9') {
47             return false;
48          }
49       }
50       return true;
51    }
52
53    /**
54     * Given a string, return a positive integer if the string can be parsed into
55     * a positive integer. Return 0 for non-positive integer or parsing error.
56     */
57    public static int parsePositiveInt(String str) {
58       if (str == null || (str = str.trim()).length() == 0) {
59          return 0;
60       }
61
62       int result;
63       try {
64          result = Integer.parseInt(str);
65       } catch (NumberFormatException ex) {
66          return 0;
67       }
68       return (result > 0) ? result : 0;
69    }
70 }
```

"OrderServlet.java" (re-written)

The codes, which uses the utility methods, become rather messy! Checking valid inputs and handling abnormal conditions are never easy!

```
1  package mypkg;
2
3  import java.io.*;
4  import java.sql.*;
5  import java.util.logging.*;
6  import javax.servlet.*;
7  import javax.servlet.http.*;
8  import myutil.InputFilter;
9
10 public class OrderServlet extends HttpServlet {
11
12    private String databaseURL, username, password;
13
14    @Override
15    public void init(ServletConfig config) throws ServletException {
16       super.init(config);
17       ServletContext context = config.getServletContext();
18       databaseURL = context.getInitParameter("databaseURL");
19       username = context.getInitParameter("username");
20       password = context.getInitParameter("password");
21    }
22
23    @Override
24    protected void doGet(HttpServletRequest request, HttpServletResponse response)
25            throws ServletException, IOException {
26       response.setContentType("text/html;charset=UTF-8");
27       PrintWriter out = response.getWriter();
28
29       Connection conn = null;
30       Statement  stmt = null;
31       ResultSet  rset = null;
32       String     sqlStr = null;
33
34       try {
35          out.println("<html><head><title>Order Confirmation</title></head><body>");
36          out.println("<h2>YAEBS - Order Confirmation</h2>");
37
38          // Retrieve and process request parameters: id(s), cust_name, cust_email, cust_phone
39          String[] ids = request.getParameterValues("id");  // Possibly more than one values
```

```
40          String custName = request.getParameter("cust_name");
41          boolean hasCustName = custName != null &&
42                  ((custName = InputFilter.htmlFilter(custName.trim())).length() > 0);
43          String custEmail = request.getParameter("cust_email").trim();
44          boolean hasCustEmail = custEmail != null &&
45                  ((custEmail = InputFilter.htmlFilter(custEmail.trim())).length() > 0);
46          String custPhone = request.getParameter("cust_phone").trim();
47          boolean hasCustPhone = custPhone != null &&
48                  ((custPhone = InputFilter.htmlFilter(custPhone.trim())).length() > 0);
49
50          // Validate inputs
51          if (ids == null || ids.length == 0) {
52             out.println("<h3>Please Select a Book!</h3>");
53          } else if (!hasCustName) {
54             out.println("<h3>Please Enter Your Name!</h3>");
55          } else if (!hasCustEmail || (custEmail.indexOf('@') == -1)) {
56             out.println("<h3>Please Enter Your e-mail (user@host)!</h3>");
57          } else if (!hasCustPhone || !InputFilter.isValidPhone(custPhone)) {
58             out.println("<h3>Please Enter an 8-digit Phone Number!</h3>");
59          } else {
60             // We shall build our output in a buffer, so that it will not be interrupted
61             //  by error messages.
62             StringBuilder outBuf = new StringBuilder();
63             // Display the name, email and phone (arranged in a table)
64             outBuf.append("<table>");
65             outBuf.append("<tr><td>Customer Name:</td><td>").append(custName).append("</td></tr>");
66             outBuf.append("<tr><td>Customer Email:</td><td>").append(custEmail).append("</td></tr>");
67             outBuf.append("<tr><td>Customer Phone Number:</td><td>").append(custPhone).append("</td></tr></table>");
68
69             conn = DriverManager.getConnection(databaseURL, username, password);
70             stmt = conn.createStatement();
71             // We shall manage our transaction (because multiple SQL statements issued)
72             conn.setAutoCommit(false);
73
74             // Print the book(s) ordered in a table
75             outBuf.append("<br />");
76             outBuf.append("<table border='1' cellpadding='6'>");
77             outBuf.append("<tr><th>AUTHOR</th><th>TITLE</th><th>PRICE</th><th>QTY</th></tr>");
78
79             boolean error = false;
80             float totalPrice = 0f;
81             for (String id : ids) {
82                sqlStr = "SELECT * FROM books WHERE id = " + id;
83                //System.out.println(sqlStr);  // for debugging
84                rset = stmt.executeQuery(sqlStr);
85
86                // Expect only one row in ResultSet
87                rset.next();
88                int qtyAvailable = rset.getInt("qty");
89                String title = rset.getString("title");
90                String author = rset.getString("author");
91                float price = rset.getFloat("price");
92
93                // Validate quantity ordered
94                String qtyOrderedStr = request.getParameter("qty" + id);
95                int qtyOrdered = InputFilter.parsePositiveInt(qtyOrderedStr);
96                if (qtyOrdered == 0) {
97                   out.println("<h3>Please Enter a valid quantity for \"" + title + "\"!</h3>");
98                   error = true;
99                   break;
100               } else if (qtyOrdered > qtyAvailable) {
101                  out.println("<h3>There are insufficient copies of \"" + title + "\" available!</h3>");
102                  error = true;
103                  break;
104               } else {
105                  // Okay, update the books table and insert an order record
106                  sqlStr = "UPDATE books SET qty = qty - " + qtyOrdered + " WHERE id = " + id;
107                  //System.out.println(sqlStr);  // for debugging
108                  stmt.executeUpdate(sqlStr);
109
110                  sqlStr = "INSERT INTO order_records values ("
111                         + id + ", " + qtyOrdered + ", '" + custName + "', '"
112                         + custEmail + "', '" + custPhone + "')";
113                  //System.out.println(sqlStr);  // for debugging
114                  stmt.executeUpdate(sqlStr);
115
116                  // Display this book ordered
117                  outBuf.append("<tr>");
118                  outBuf.append("<td>").append(author).append("</td>");
119                  outBuf.append("<td>").append(title).append("</td>");
120                  outBuf.append("<td>").append(price).append("</td>");
121                  outBuf.append("<td>").append(qtyOrdered).append("</td></tr>");
122                  totalPrice += price * qtyOrdered;
123               }
124            }
125
126            if (error) {
127               conn.rollback();
128            } else {
129               // No error, print the output from the StringBuilder.
```

```
                out.println(outBuf.toString());
                out.println("<tr><td colspan='4' align='right'>Total Price: $");
                out.printf("%.2f</td></tr>", totalPrice);
                out.println("</table>");

                out.println("<h3>Thank you.</h3>");
                out.println("<p><a href='start'>Back to Select Menu</a></p>");
                // Commit for ALL the books ordered.
                conn.commit();
             }
          }
          out.println("</body></html>");
       } catch (SQLException ex) {
          try {
             conn.rollback();  // rollback the updates
             out.println("<h3>Service not available. Please try again later!</h3></body></html>");
          } catch (SQLException ex1) { }
          Logger.getLogger(OrderServlet.class.getName()).log(Level.SEVERE, null, ex);
       } finally {
          out.close();
          try {
             if (stmt != null) stmt.close();
             if (conn != null) conn.close();
          } catch (SQLException ex) {
             Logger.getLogger(OrderServlet.class.getName()).log(Level.SEVERE, null, ex);
          }
       }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       doGet(request, response);
    }
}
```

Dissecting the Program:

1. [TODO]

2. Database Connection Pooling

Using a new connection to service each request is highly inefficient, due to the high overhead involved in initializing and maintaining the connection. A common practice is to setup a common pool of database connections. A servlet picks up an available connection from the pool to perform database operation, and returns the connection to the pool once it is done. Tomcat supports database connection pooling via JDNI (Java Directory and Naming Interface).

Create a new web application called "`yaebsdbcp`" (*yet another e-bookshop database connection pooling*). Copy all the servlets in the previous exercises into this webapp.

The steps to set up database connection pooling in Tomcat is as follows:

Step 1: Configure a JNDI DataSource

Write a JNDI (Java Naming and Directory Interface) DataSource configuration in "`context.xml`" as follows. For application-specific configuration, save it under application's "`META-INF`". (For server-wide configuration, put the `<Resource>` element under `<GlobalNamingResources>` in `$CATALINA_HOME\conf\server.xml`.)

(NetBeans) The "`context.xml`" can be found under the "Configuration Files" node.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<Context reloadable="true">
  <!--
    maxActive: Maximum number of dB connections in pool. Set to -1 for no limit.
    maxIdle: Maximum number of idle dB connections to retain in pool. Set to -1 for no limit.
    maxWait: Maximum milliseconds to wait for a dB connection to become available
             Set to -1 to wait indefinitely.
  -->
  <Resource name="jdbc/mysql_ebookshop" auth="Container" type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000" removeAbandoned="true"
     username="myuser" password="xxxx" driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/ebookshop" />
</Context>
```

The above configuration declares a *JNDI resource name* called "`jdbc/mysql_ebookshop`" corresponds to the MySQL connection "`mysql://localhost:3306/ebookshop`". Check your database-url, username and password.

Step 2: Application Deployment Descriptor "`web.xml`"

Configure "`WEB-INF\web.xml`" to reference the JDNI "`jdbc/mysql_ebookshop`".

```
<web-app ......>

   <resource-ref>
      <description>DB Connection Pool</description>
      <res-ref-name>jdbc/mysql_ebookshop</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
   </resource-ref>
   ......
</web-app>
```

Note: This step seems to be optional?!

Step 3: Modify your Servlet to Use Connection Pool

Modify all the servlet to use database connection pooling as follows:

Modify all the servlet to use database connection pooling as follows:

```
......

public class EntryServlet extends HttpServlet {

   private DataSource pool;  // Database connection pool

   @Override
   public void init(ServletConfig config) throws ServletException {
      try {
         // Create a JNDI Initial context to be able to lookup the DataSource
         InitialContext ctx = new InitialContext();
         // Lookup the DataSource, which will be backed by a pool
         //   that the application server provides.
         pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql_ebookshop");
         if (pool == null)
            throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
      } catch (NamingException ex) {
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      Connection conn = null;
      Statement stmt = null;
      try {
         // Get a connection from the pool
         conn = pool.getConnection();

         stmt = conn.createStatement();
         ......
         ......

      } catch (SQLException ex) {
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
         out.close();
         try {
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();  // return the connection to the pool
         } catch (SQLException ex) {
            Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
         }
      }
   }
   ......
}
```

Explanation:

1. The `init()` method looks up the JNDI directory and setup the database connection pool (`DataSource`) as configured in the `<context.xml>`.
2. In `doGet()`, `pool.getConnection()` gets an available connection from the pool. The `conn.close()` is important now, as it returns the connection back to the pool.

Modify all the servlets (`EntryServlet`, `QueryServlet`, and `OrderServlet`) to use connection pooling.

[TODO] How to monitor the connection pool?

3. Session Management (aka "Shopping Cart")

HTTP is a *stateless* protocol. In other words, the current request does not know what has been done in the previous requests. This creates a problem for applications that runs over many requests, such as online shopping. You need to maintain a so-called *session* with a *shopping cart* to pass data among the multiple requests. The user will *check out* his/her shopping cart once he/she is done.

Java Servlet API provide a `HttpSession` that greatly simplifies the session management.

For details on session management, read "Java Servlets - Session Tracking".

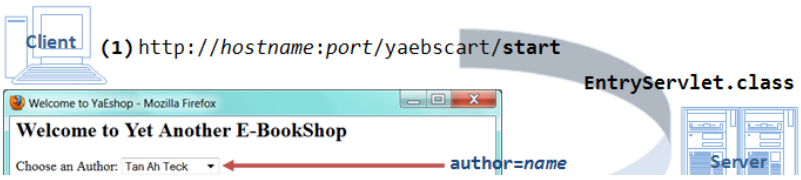
3.1 E-Shop with Shopping Cart

Create a New Webapp

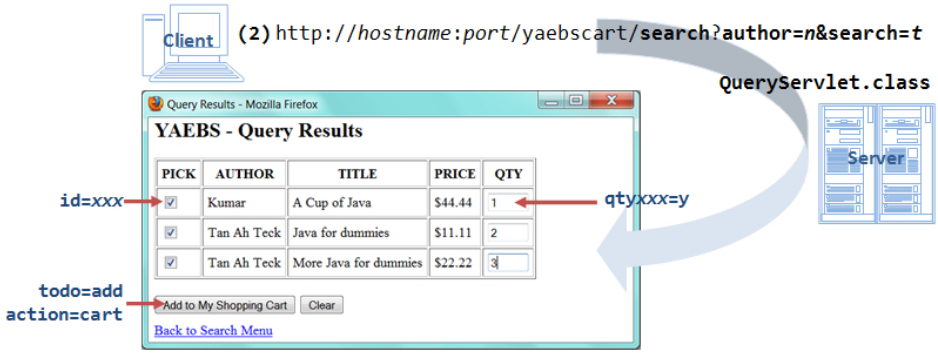
Create a new webapp called "`yaebscart`" (*yet another e-bookshop with shopping cart*). We shall re-use codes in database connection pooling exercise.

Sequence Diagram

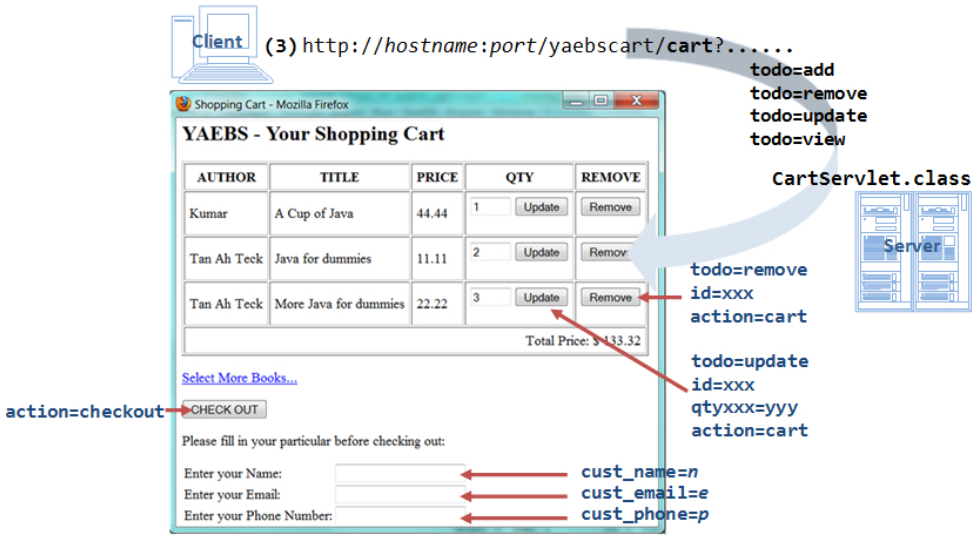
1. A client issues a URL `http://hostname:port/yaebscart/start` to start the webapp, which is mapped to "`EntryServlet`". The servlet responses with an HTML query form. The form is to be submitted to URL "`/search`" (mapped to "`QueryServlet`") with request parameters `author=name` and/or `search=term`.



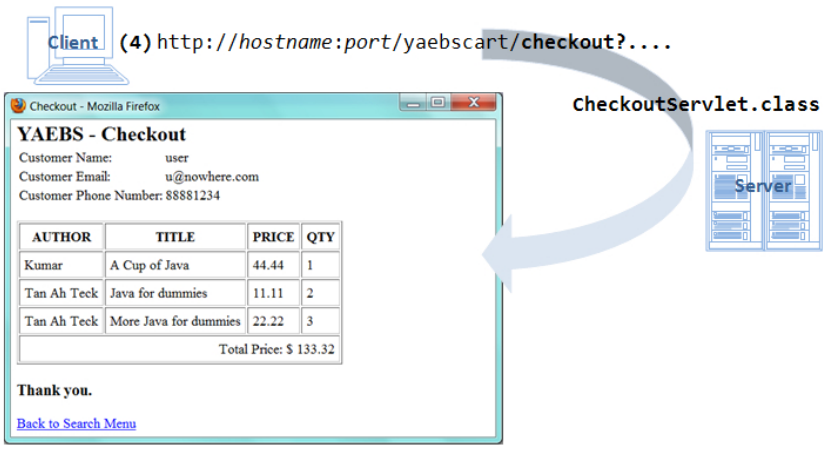
2. The client checks the item(s) and sends the request to "`QueryServlet`". The servlet extracts the request parameters, queries the database, and returns the list of books in an HTML form. Each item has a checkbox (with `id=xxx` and `qtyxxx=yyy`). The checkboxes are enclosed within a form with a hidden input `todo=add`. The form is to be submitted to URL "`\cart`" (mapped is "`CartServlet`").



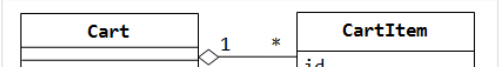
3. The "`CartServlet`" create a new `HttpSession` and a `Cart` (during the first access), and places the `Cart` inside the `HttpSession`. The `CartServlet` handle these processes:
   a. `todo=add, id=1001, qty1001=2, [id=1002, qty1002-3...]`: Add the books into the shopping cart. If the book is already in the cart, increase its quantity ordered. Display the shopping cart.
   b. `todo=remove, id=1001`: Remove the book (identified by the book's id) from the shopping cart. Display the shopping cart.
   c. `todo=update, id=1001, qty1000=5`: Update the quantity ordered for that particular book's id, in the shopping cart. Display the shopping cart.
   d. `todo=view`: Display the shopping cart.

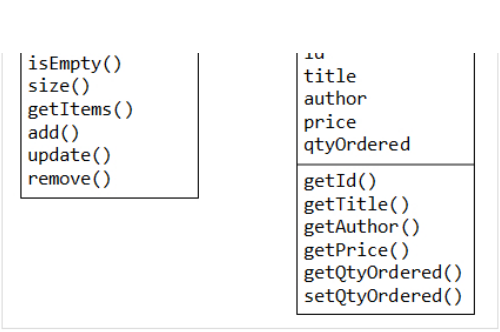


4. The "`CheckoutServlet`" performs the checkout process. It retrieves the orders form the shopping cart, updates the database tables `books` by reducing the quantity available, and inserts a transaction record in `order_records` table.



Create the Java Classes to Support Shopping Cart - `Cart` and `CartItem`





The `CartItem` class models individual item placed inside the shopping cart. In our e-bookstore, we need to keep track of the id, title, author, price and quantity ordered.

```
package mypkg;

/**
 * The class CartItem models an item in the Cart.
 * This class shall not be accessed by the controlling logic directly.
 * Instead Use Cart.add() or Cart.remove() to add or remove an item from the Cart.
 */
public class CartItem {

   private int id;
   private String title;
   private String author;
   private float price;
   private int qtyOrdered;

   // Constructor
   public CartItem(int id, String title,
           String author, float price, int qtyOrdered) {
      this.id = id;
      this.title = title;
      this.author = author;
      this.price = price;
      this.qtyOrdered = qtyOrdered;
   }

   public int getId() {
      return id;
   }

   public String getAuthor() {
      return author;
   }

   public String getTitle() {
      return title;
   }

   public float getPrice() {
      return price;
   }

   public int getQtyOrdered() {
      return qtyOrdered;
   }

   public void setQtyOrdered(int qtyOrdered) {
      this.qtyOrdered = qtyOrdered;
   }
}
```

The `Cart` class stores the items in a `List` of `CartItem`. It also provides these public methods:

- `add()`: Add a item into the card. It checks if the `id` is already in the cart. If so, it adjust the quantity ordered. Otherwise, it creates a new `CartItem` and adds to the `ArrayList`.
- `update()`: Update the quantity order for the given book's id.
- `remove()`: Remove a particular item from the shopping cart, identified via the book's id.
- `isEmpty()`: Return true if the cart is empty.
- `size()`: Return the number of items in the shopping cart.
- `getItems()`: Return all the items of the shopping cart in a `List<CartItem>`.
- `clear()`: Empty the contents of the shopping cart.

```
package mypkg;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/**
 * The Cart class models the shopping cart, which contains CartItem.
 * It also provides method to add() and remove() a CartItem.
 */
public class Cart {

   private List<CartItem> cart;  // List of CartItems

```

```
   // constructor
   public Cart() {
      cart = new ArrayList<CartItem>();
   }

   // Add a CartItem into this Cart
   public void add(int id, String title, String author, float price, int qtyOrdered) {
      // Check if the id is already in the shopping cart
      Iterator<CartItem> iter = cart.iterator();
      while (iter.hasNext()) {
         CartItem item = iter.next();
         if (item.getId() == id) {
            // id found, increase qtyOrdered
            item.setQtyOrdered(item.getQtyOrdered() + qtyOrdered);
            return;
         }
      }
      // id not found, create a new CartItem
      cart.add(new CartItem(id, title, author, price, qtyOrdered));
   }

   // Update the quantity for the given id
   public boolean update(int id, int newQty) {
      Iterator<CartItem> iter = cart.iterator();
      while (iter.hasNext()) {
         CartItem item = iter.next();
         if (item.getId() == id) {
            // id found, increase qtyOrdered
            item.setQtyOrdered(newQty);
            return true;
         }
      }
      return false;
   }

   // Remove a CartItem given its id
   public void remove(int id) {
      Iterator<CartItem> iter = cart.iterator();
      while (iter.hasNext()) {
         CartItem item = iter.next();
         if (item.getId() == id) {
            cart.remove(item);
            return;
         }
      }
   }

   // Get the number of CartItems in this Cart
   public int size() {
      return cart.size();
   }

   // Check if this Cart is empty
   public boolean isEmpty() {
      return size() == 0;
   }

   // Return all the CartItems in a List<CartItem>
   public List<CartItem> getItems() {
      return cart;
   }

   // Remove all the items in this Cart
   public void clear() {
      cart.clear();
   }
}
```

EntryServlet.java (URL "/start")

```
package mypkg;

import java.io.*;
import java.sql.*;
import java.util.logging.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.DataSource;

public class EntryServlet extends HttpServlet {

   private DataSource pool;  // Database connection pool

   @Override
   public void init(ServletConfig config) throws ServletException {
      try {
         // Create a JNDI Initial context to be able to lookup the DataSource
         InitialContext ctx = new InitialContext();
         // Lookup the DataSource
         pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql ebookshop");
```

```
         if (pool == null)
            throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
      } catch (NamingException ex) {
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      Connection conn = null;
      Statement stmt = null;
      try {
         conn = pool.getConnection();  // Get a connection from the pool
         stmt = conn.createStatement();
         String sqlStr = "SELECT DISTINCT author FROM books WHERE qty > 0";
         ResultSet rset = stmt.executeQuery(sqlStr);

         out.println("<html><head><title>Welcome to YaEshop</title></head><body>");
         out.println("<h2>Welcome to Yet Another E-BookShop</h2>");
         out.println("<form method='get' action='search'>");

         // A pull-down menu of all the authors with a no-selection option
         out.println("Choose an Author: <select name='author' size='1'>");
         out.println("<option value=''>Select...</option>");  // no-selection
         while (rset.next()) {  // list all the authors
            String author = rset.getString("author");
            out.println("<option value='" + author + "'>" + author + "</option>");
         }
         out.println("</select><br />");
         out.println("<p>OR</p>");

         // A text field for entering search word for pattern matching
         out.println("Search \"Title\" or \"Author\": <input type='text' name='search' />");

         // Submit and reset buttons
         out.println("<br /><br />");
         out.println("<input type='submit' value='SEARCH' />");
         out.println("<input type='reset' value='CLEAR' />");
         out.println("</form>");

         // Show "View Shopping Cart" if the cart is not empty
         HttpSession session = request.getSession(false); // check if session exists
         if (session != null) {
            Cart cart;
            synchronized (session) {
               // Retrieve the shopping cart for this session, if any. Otherwise, create one.
               cart = (Cart) session.getAttribute("cart");
               if (cart != null && !cart.isEmpty()) {
                  out.println("<P><a href='cart?todo=view'>View Shopping Cart</a></p>");
               }
            }
         }

         out.println("</body></html>");
      } catch (SQLException ex) {
         out.println("<h3>Service not available. Please try again later!</h3></body></html>");
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
         out.close();
         try {
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();  // Return the connection to the pool
         } catch (SQLException ex) {
            Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
         }
      }
   }

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      doGet(request, response);
   }
}
```

QueryServlet.java (URL "/query")

```
package mypkg;

import java.io.*;
import java.sql.*;
import java.util.logging.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.*;
```

```
public class QueryServlet extends HttpServlet {

   private DataSource pool;  // Database connection pool

   @Override
   public void init(ServletConfig config) throws ServletException {
      try {
         // Create a JNDI Initial context to be able to lookup the DataSource
         InitialContext ctx = new InitialContext();
         // Lookup the DataSource.
         pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql_ebookshop");
         if (pool == null)
            throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
      } catch (NamingException ex) {
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      Connection conn = null;
      Statement stmt = null;

      try {
         // Retrieve and process request parameters: "author" and "search"
         String author = request.getParameter("author");
         boolean hasAuthorParam = author != null && !author.equals("Select...");
         String searchWord = request.getParameter("search").trim();
         boolean hasSearchParam = searchWord != null && (searchWord.length() > 0);

         out.println("<html><head><title>Query Results</title></head><body>");
         out.println("<h2>YAEBS - Query Results</h2>");

         if (!hasAuthorParam && !hasSearchParam) {  // No params present
            out.println("<h3>Please select an author or enter a search term!</h3>");
            out.println("<p><a href='start'>Back to Select Menu</a></p>");
         } else {
            conn = pool.getConnection();
            stmt = conn.createStatement();

            // Form a SQL command based on the param(s) present
            StringBuilder sqlStr = new StringBuilder();  // more efficient than String
            sqlStr.append("SELECT * FROM books WHERE qty > 0 AND (");
            if (hasAuthorParam) {
               sqlStr.append("author = '").append(author).append("'");
            }
            if (hasSearchParam) {
               if (hasAuthorParam) {
                  sqlStr.append(" OR ");
               }
               sqlStr.append("author LIKE '%").append(searchWord)
                     .append("%' OR title LIKE '%").append(searchWord).append("%'");
            }
            sqlStr.append(") ORDER BY author, title");
            //System.out.println(sqlStr);  // for debugging
            ResultSet rset = stmt.executeQuery(sqlStr.toString());

            if (!rset.next()) {  // Check for empty ResultSet (no book found)
               out.println("<h3>No book found. Please try again!</h3>");
               out.println("<p><a href='start'>Back to Select Menu</a></p>");
            } else {
               // Print the result in an HTML form inside a table
               out.println("<form method='get' action='cart'>");
               out.println("<input type='hidden' name='todo' value='add' />");
               out.println("<table border='1' cellpadding='6'>");
               out.println("<tr>");
               out.println("<th> </th>");
               out.println("<th>AUTHOR</th>");
               out.println("<th>TITLE</th>");
               out.println("<th>PRICE</th>");
               out.println("<th>QTY</th>");
               out.println("</tr>");

               // ResultSet's cursor now pointing at first row
               do {
                  // Print each row with a checkbox identified by book's id
                  String id = rset.getString("id");
                  out.println("<tr>");
                  out.println("<td><input type='checkbox' name='id' value='" + id + "' /></td>");
                  out.println("<td>" + rset.getString("author") + "</td>");
                  out.println("<td>" + rset.getString("title") + "</td>");
                  out.println("<td>$" + rset.getString("price") + "</td>");
                  out.println("<td><input type='text' size='3' value='1' name='qty" + id + "' /></td>");
                  out.println("</tr>");
               } while (rset.next());
               out.println("</table><br />");
```

```
101
102                 // Submit and reset buttons
103                 out.println("<input type='submit' value='Add to My Shopping Cart' />");
104                 out.println("<input type='reset' value='CLEAR' /></form>");
105
106                 // Hyperlink to go back to search menu
107                 out.println("<p><a href='start'>Back to Select Menu</a></p>");
108
109                 // Show "View Shopping Cart" if cart is not empty
110                 HttpSession session = request.getSession(false); // check if session exists
111                 if (session != null) {
112                    Cart cart;
113                    synchronized (session) {
114                       // Retrieve the shopping cart for this session, if any. Otherwise, create one.
115                       cart = (Cart) session.getAttribute("cart");
116                       if (cart != null && !cart.isEmpty()) {
117                          out.println("<p><a href='cart?todo=view'>View Shopping Cart</a></p>");
118                       }
119                    }
120                 }
121
122                 out.println("</body></html>");
123              }
124           }
125        } catch (SQLException ex) {
126           out.println("<h3>Service not available. Please try again later!</h3></body></html>");
127           Logger.getLogger(QueryServlet.class.getName()).log(Level.SEVERE, null, ex);
128        } finally {
129           out.close();
130           try {
131              if (stmt != null) stmt.close();
132              if (conn != null) conn.close();  // Return the connection to the pool
133           } catch (SQLException ex) {
134              Logger.getLogger(QueryServlet.class.getName()).log(Level.SEVERE, null, ex);
135           }
136        }
137     }
138
139     @Override
140     protected void doPost(HttpServletRequest request, HttpServletResponse response)
141           throws ServletException, IOException {
142        doGet(request, response);
143     }
144  }
```

CartServlet.java (URL "/cart")

```
 1  package mypkg;
 2
 3  import java.io.*;
 4  import java.sql.*;
 5  import java.util.logging.*;
 6  import javax.naming.*;
 7  import javax.servlet.*;
 8  import javax.servlet.http.*;
 9  import javax.sql.DataSource;
10
11  public class CartServlet extends HttpServlet {
12
13     private DataSource pool;  // Database connection pool
14
15     @Override
16     public void init(ServletConfig config) throws ServletException {
17        try {
18           // Create a JNDI Initial context to be able to lookup the DataSource
19           InitialContext ctx = new InitialContext();
20           // Lookup the DataSource.
21           pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql_ebookshop");
22           if (pool == null)
23              throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
24        } catch (NamingException ex) {
25           Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
26        }
27     }
28
29     @Override
30     protected void doGet(HttpServletRequest request, HttpServletResponse response)
31           throws ServletException, IOException {
32        response.setContentType("text/html;charset=UTF-8");
33        PrintWriter out = response.getWriter();
34
35        // Retrieve current HTTPSession object. If none, create one.
36        HttpSession session = request.getSession(true);
37        Cart cart;
38        synchronized (session) {  // synchronized to prevent concurrent updates
39           // Retrieve the shopping cart for this session, if any. Otherwise, create one.
40           cart = (Cart) session.getAttribute("cart");
41           if (cart == null) {  // No cart, create one.
42              cart = new Cart();
43              session.setAttribute("cart", cart);  // Save it into session
44           }
```

```
45        }
46
47        Connection conn   = null;
48        Statement  stmt   = null;
49        ResultSet  rset   = null;
50        String     sqlStr = null;
51
52        try {
53           conn = pool.getConnection();  // Get a connection from the pool
54           stmt = conn.createStatement();
55
56           out.println("<html><head><title>Shopping Cart</title></head><body>");
57           out.println("<h2>YAEBS - Your Shopping Cart</h2>");
58
59           // This servlet handles 4 cases:
60           // (1) todo=add id=1001 qty1001=5 [id=1002 qty1002=1 ...]
61           // (2) todo=update id=1001 qty1001=5
62           // (3) todo=remove id=1001
63           // (4) todo=view
64
65           String todo = request.getParameter("todo");
66           if (todo == null) todo = "view";  // to prevent null pointer
67
68           if (todo.equals("add") || todo.equals("update")) {
69              // (1) todo=add id=1001 qty1001=5 [id=1002 qty1002=1 ...]
70              // (2) todo=update id=1001 qty1001=5
71              String[] ids = request.getParameterValues("id");
72              if (ids == null) {
73                 out.println("<h3>Please Select a Book!</h3></body></html>");
74                 return;
75              }
76              for (String id : ids) {
77                 sqlStr = "SELECT * FROM books WHERE id = " + id;
78                 //System.out.println(sqlStr);  // for debugging
79                 rset = stmt.executeQuery(sqlStr);
80                 rset.next(); // Expect only one row in ResultSet
81                 String title = rset.getString("title");
82                 String author = rset.getString("author");
83                 float price = rset.getFloat("price");
84
85                 // Get quantity ordered - no error check!
86                 int qtyOrdered = Integer.parseInt(request.getParameter("qty" + id));
87                 int idInt = Integer.parseInt(id);
88                 if (todo.equals("add")) {
89                    cart.add(idInt, title, author, price, qtyOrdered);
90                 } else if (todo.equals("update")) {
91                    cart.update(idInt, qtyOrdered);
92                 }
93              }
94
95           } else if (todo.equals("remove")) {
96              String id = request.getParameter("id");  // Only one id for remove case
97              cart.remove(Integer.parseInt(id));
98           }
99
100          // All cases - Always display the shopping cart
101          if (cart.isEmpty()) {
102             out.println("<p>Your shopping cart is empty</p>");
103          } else {
104             out.println("<table border='1' cellpadding='6'>");
105             out.println("<tr>");
106             out.println("<th>AUTHOR</th>");
107             out.println("<th>TITLE</th>");
108             out.println("<th>PRICE</th>");
109             out.println("<th>QTY</th>");
110             out.println("<th>REMOVE</th></tr>");
111
112             float totalPrice = 0f;
113             for (CartItem item : cart.getItems()) {
114                int id = item.getId();
115                String author = item.getAuthor();
116                String title = item.getTitle();
117                float price = item.getPrice();
118                int qtyOrdered = item.getQtyOrdered();
119
120                out.println("<tr>");
121                out.println("<td>" + author + "</td>");
122                out.println("<td>" + title +  "</td>");
123                out.println("<td>" + price +  "</td>");
124
125                out.println("<td><form method='get'>");
126                out.println("<input type='hidden' name='todo' value='update' />");
127                out.println("<input type='hidden' name='id' value='" + id + "' />");
128                out.println("<input type='text' size='3' name='qty"
129                       + id + "' value='" + qtyOrdered + "' />" );
130                out.println("<input type='submit' value='Update' />");
131                out.println("</form></td>");
132
133                out.println("<td><form method='get'>");
134                out.println("<input type='submit' value='Remove'>");
```

```
                out.println("<input type='hidden' name='todo' value='remove'");
                out.println("<input type='hidden' name='id' value='" + id + "'>");
                out.println("</form></td>");
                out.println("</tr>");
                totalPrice += price * qtyOrdered;
             }
             out.println("<tr><td colspan='5' align='right'>Total Price: $");
             out.printf("%.2f</td></tr>", totalPrice);
             out.println("</table>");
          }

          out.println("<p><a href='start'>Select More Books...</a></p>");

          // Display the Checkout
          if (!cart.isEmpty()) {
             out.println("<br /><br />");
             out.println("<form method='get' action='checkout'>");
             out.println("<input type='submit' value='CHECK OUT'>");
             out.println("<p>Please fill in your particular before checking out:</p>");
             out.println("<table>");
             out.println("<tr>");
             out.println("<td>Enter your Name:</td>");
             out.println("<td><input type='text' name='cust_name' /></td></tr>");
             out.println("<tr>");
             out.println("<td>Enter your Email:</td>");
             out.println("<td><input type='text' name='cust_email' /></td></tr>");
             out.println("<tr>");
             out.println("<td>Enter your Phone Number:</td>");
             out.println("<td><input type='text' name='cust_phone' /></td></tr>");
             out.println("</table>");
             out.println("</form>");
          }

          out.println("</body></html>");

       } catch (SQLException ex) {
          out.println("<h3>Service not available. Please try again later!</h3></body></html>");
          Logger.getLogger(CartServlet.class.getName()).log(Level.SEVERE, null, ex);
       } finally {
          out.close();
          try {
             if (stmt != null) stmt.close();
             if (conn != null) conn.close();  // return the connection to the pool
          } catch (SQLException ex) {
             Logger.getLogger(CartServlet.class.getName()).log(Level.SEVERE, null, ex);
          }
       }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       doGet(request, response);
    }
}
```

CheckoutServlet.java (URL "/checkout")

```
package mypkg;

import java.io.*;
import java.sql.*;
import java.util.logging.*;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.sql.DataSource;

public class CheckoutServlet extends HttpServlet {

   private DataSource pool;  // Database connection pool

   @Override
   public void init(ServletConfig config) throws ServletException {
      try {
         // Create a JNDI Initial context to be able to lookup the DataSource
         InitialContext ctx = new InitialContext();
         // Lookup the DataSource.
         pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql_ebookshop");
         if (pool == null)
            throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
      } catch (NamingException ex) {
         Logger.getLogger(EntryServlet.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
```

```
34
35      Connection conn = null;
36      Statement stmt = null;
37      ResultSet rset = null;
38      String sqlStr = null;
39      HttpSession session = null;
40      Cart cart = null;
41
42      try {
43         conn = pool.getConnection();  // Get a connection from the pool
44         stmt = conn.createStatement();
45
46         out.println("<html><head><title>Checkout</title></head><body>");
47         out.println("<h2>YAEBS - Checkout</h2>");
48
49         // Retrieve the Cart
50         session = request.getSession(false);
51         if (session == null) {
52            out.println("<h3>Your Shopping cart is empty!</h3></body></html>");
53            return;
54         }
55         synchronized (session) {
56            cart = (Cart) session.getAttribute("cart");
57            if (cart == null) {
58               out.println("<h3>Your Shopping cart is empty!</h3></body></html>");
59               return;
60            }
61         }
62
63         // Retrieve and process request parameters: id(s), cust_name, cust_email, cust_phone
64         String custName = request.getParameter("cust_name");
65         boolean hasCustName = custName != null && ((custName = custName.trim()).length() > 0);
66         String custEmail = request.getParameter("cust_email").trim();
67         boolean hasCustEmail = custEmail != null && ((custEmail = custEmail.trim()).length() > 0);
68         String custPhone = request.getParameter("cust_phone").trim();
69         boolean hasCustPhone = custPhone != null && ((custPhone = custPhone.trim()).length() > 0);
70
71         // Validate inputs
72         if (!hasCustName) {
73
74            out.println("<h3>Please Enter Your Name!</h3></body></html>");
75            return;
76         } else if (!hasCustEmail || (custEmail.indexOf('@') == -1)) {
77            out.println("<h3>Please Enter Your email (user@host)!</h3></body></html>");
78            return;
79         } else if (!hasCustPhone || custPhone.length() != 8) {
80            out.println("<h3>Please Enter an 8-digit Phone Number!</h3></body></html>");
81            return;
82         }
83
84         // Display the name, email and phone (arranged in a table)
85         out.println("<table>");
86         out.println("<tr>");
87         out.println("<td>Customer Name:</td>");
88         out.println("<td>" + custName + "</td></tr>");
89         out.println("<tr>");
90         out.println("<td>Customer Email:</td>");
91         out.println("<td>" + custEmail + "</td></tr>");
92         out.println("<tr>");
93         out.println("<td>Customer Phone Number:</td>");
94         out.println("<td>" + custPhone + "</td></tr>");
95         out.println("</table>");
96
97         // Print the book(s) ordered in a table
98         out.println("<br />");
99         out.println("<table border='1' cellpadding='6'>");
100        out.println("<tr>");
101        out.println("<th>AUTHOR</th>");
102        out.println("<th>TITLE</th>");
103        out.println("<th>PRICE</th>");
104        out.println("<th>QTY</th></tr>");
105
106        float totalPrice = 0f;
107        for (CartItem item : cart.getItems()) {
108           int id = item.getId();
109           String author = item.getAuthor();
110           String title = item.getTitle();
111           int qtyOrdered = item.getQtyOrdered();
112           float price = item.getPrice();
113
114           // No check for price and qtyAvailable change
115           // Update the books table and insert an order record
116           sqlStr = "UPDATE books SET qty = qty - " + qtyOrdered + " WHERE id = " + id;
117           //System.out.println(sqlStr);  // for debugging
118           stmt.executeUpdate(sqlStr);
119
120           sqlStr = "INSERT INTO order_records values ("
121                  + id + ", " + qtyOrdered + ", '" + custName + "', '"
122                  + custEmail + "', '" + custPhone + "')";
123           //System.out.println(sqlStr);  // for debugging
```

```
            stmt.executeUpdate(sqlStr);

            // Show the book ordered
            out.println("<tr>");
            out.println("<td>" + author + "</td>");
            out.println("<td>" + title + "</td>");
            out.println("<td>" + price + "</td>");
            out.println("<td>" + qtyOrdered + "</td></tr>");
            totalPrice += price * qtyOrdered;
         }
         out.println("<tr><td colspan='4' align='right'>Total Price: $");
         out.printf("%.2f</td></tr>", totalPrice);
         out.println("</table>");

         out.println("<h3>Thank you.</h3>");
         out.println("<a href='start'>Back to Search Menu</a>");
         out.println("</body></html>");

         cart.clear();   // empty the cart
      } catch (SQLException ex) {
         cart.clear();   // empty the cart
         out.println("<h3>Service not available. Please try again later!</h3></body></html>");
         Logger.getLogger(CheckoutServlet.class.getName()).log(Level.SEVERE, null, ex);
      } finally {
         out.close();
         try {
            if (stmt != null) stmt.close();
            if (conn != null) conn.close();  // Return the connection to the pool
         } catch (SQLException ex) {
            Logger.getLogger(CheckoutServlet.class.getName()).log(Level.SEVERE, null, ex);
         }
      }
   }

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      doGet(request, response);
   }
}
```

Application Deployment Descriptor "web.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <servlet>
        <servlet-name>EntryServlet</servlet-name>
        <servlet-class>mypkg.EntryServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>QueryServlet</servlet-name>
        <servlet-class>mypkg.QueryServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>CartServlet</servlet-name>
        <servlet-class>mypkg.CartServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>CheckoutServlet</servlet-name>
        <servlet-class>mypkg.CheckoutServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>EntryServlet</servlet-name>
        <url-pattern>/start</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>QueryServlet</servlet-name>
        <url-pattern>/search</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>CartServlet</servlet-name>
        <url-pattern>/cart</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>CheckoutServlet</servlet-name>
        <url-pattern>/checkout</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>start</welcome-file>
    </welcome-file-list>
</web-app>
```

"context.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/yaebscart" >
   <Resource name="jdbc/mysql_ebookshop" auth="Container" type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000" removeAbandoned="true"
     username="myuser" password="xxxx" driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/ebookshop" />
</Context>
```

3.2 With More Input Validation and Abnormal Condition Checks

Again, I rewrite the `CartServlet` and `CheckoutServlet` to do more input validation and check for abnormal conditions. The codes are messy!

- "`myutil.InputFilter`" (as in the above example)
- "`CartServlet.java`" [source link]
- "`CheckoutServlet.java`" [source link]

4. User and Role Management

[TODO] Intro

In a practical system, a user has a password; a user may take one or many roles.

4.1 User and Role Management via HttpSession

Setup Database

Set up tables `users` and `user_roles` under the database ebookshop as follows. Instead of storing plain-text password (which might exposes your password, if you use the same password for all your applications), we create a hash of password, via the MySQL `PASSWORD()` function, and store the hash value. The `PASSWORD()` function produces a 41-byte hash value. (Read "MySQL Manual: Password Hashing in MySQL").

The following SQL script can be used to setup the database.

```
use ebookshop;

drop table if exists user_roles;
drop table if exists users;
create table users (
  username char(16) not null,
  password char(41) not null,
  primary key (username)
);

create table user_roles (
  username char(16) not null,
  role     varchar(16) not null,
  primary key (username, role),
  foreign key (username) references users (username)
);

insert into users values
  ('user1', password('user1')),
  ('admin1', password('admin1'));

insert into user_roles values
  ('user1', 'user'),
  ('admin1', 'admin'),
  ('admin1', 'user');

select * from users;
select * from user_roles;
```

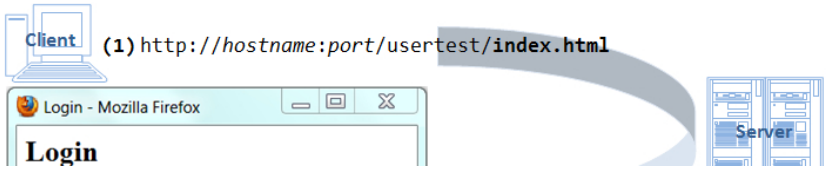
Sequence of Events

1. The webapp begins with a login form to prompt the user for username and password, which are submitted to a login script ("`LoginServlet`" with URL "`/login`".
2. The login script verifies the hash of the submitted password with that stored in the database for the username. If the username/password is successful verified, it creates a new `HttpSession`, and places the username and role(s) into the session. This information will be available for all sequence accesses.
3. All sequence pages retrieve the username and roles from the session, before performing its operations. Otherwise, it shall abort its operations.
4. The logout script invalidates the session.

To verify username/password pair, you can issue the following SQL statement. Take note that STRCMP function is not case-sensitive. It returns 0 if two strings are the same.

```
// Verify username and password.
SELECT * FROM users
  WHERE STRCMP(username, 'user1') = 0 AND STRCMP(password, PASSWORD('user1') = 0);

// Verify username and password, return the roles.
SELECT role FROM users, user_roles
  WHERE STRCMP(users.username, 'user1') = 0 AND STRCMP(users.password, PASSWORD('user1') = 0)
  AND users.username = user_roles.username;
```

Let's illustrate with an example with the following sequences:



Login Form - "index.html"

```
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
  <h2>Login</h2>
  <form method="get" action="login">
  <table>
    <tr>
      <td>Enter your username:</td>
      <td><input type='text' name='username' /></td>
    </tr>
    <tr>
      <td>Enter your password:</td>
      <td><input type='password' name='password' /></td>
    </tr>
  </table>
  <br />
  <input type="submit" value='LOGIN' />
  <input type="reset" value='CLEAR' />
  </form>
</body>
</html>
```

Take note the the password is sent in *clear text* in HTTP GET as well as POST request, although it is masked out on the screen with `<input type="password">` tag. To use an HTML form to send password, you have to run HTTP with SSL (HTTPS), which encrypts the data transferred.

The Login Script - "LoginServlet.java" (with URL "/login")

```
package mypkg;

import java.io.*;
import java.util.*;
import java.util.logging.*;
```

```java
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import javax.sql.*;

public class LoginServlet extends HttpServlet {

   private DataSource pool;  // Database connection pool

   @Override
   public void init(ServletConfig config) throws ServletException {
      try {
         // Create a JNDI Initial context to be able to lookup the DataSource
         InitialContext ctx = new InitialContext();
         // Lookup the DataSource
         pool = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql_ebookshop");
         if (pool == null)
            throw new ServletException("Unknown DataSource 'jdbc/mysql_ebookshop'");
      } catch (NamingException ex) {
         Logger.getLogger(LoginServlet.class.getName()).log(Level.SEVERE, null, ex);
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      Connection conn = null;
      Statement  stmt = null;
      try {
         out.println("<html><head><title>Login</title></head><body>");
         out.println("<h2>Login</h2>");

         conn = pool.getConnection();  // Get a connection from the pool
         stmt = conn.createStatement();

         // Retrieve and process request parameters: username and password
         String userName = request.getParameter("username");
         String password = request.getParameter("password");
         boolean hasUserName = userName != null && ((userName = userName.trim()).length() > 0);
         boolean hasPassword = password != null && ((password = password.trim()).length() > 0);

         // Validate input request parameters
         if (!hasUserName) {
            out.println("<h3>Please Enter Your username!</h3>");
         } else if (!hasPassword) {
            out.println("<h3>Please Enter Your password!</h3>");
         } else {
            // Verify the username/password and retrieve the role(s)
            StringBuilder sqlStr = new StringBuilder();
            sqlStr.append("SELECT role FROM users, user_roles WHERE ");
            sqlStr.append("STRCMP(users.username, '")
                  .append(userName).append("') = 0 ");
            sqlStr.append("AND STRCMP(users.password, PASSWORD('")
                  .append(password).append("')) = 0 ");
            sqlStr.append("AND users.username = user_roles.username");
            //System.out.println(sqlStr);  // for debugging

            ResultSet rset = stmt.executeQuery(sqlStr.toString());

            // Check if username/password are correct
            if (!rset.next()) {  // empty ResultSet
               out.println("<h3>Wrong username/password!</h3>");
               out.println("<p><a href='index.html'>Back to Login Menu</a></p>");
            } else {
               // Retrieve the roles
               List<String> roles = new ArrayList<>();
               do {
                  roles.add(rset.getString("role"));
               } while (rset.next());

               // Create a new HTTPSession and save the username and roles
               // First, invalidate the session. if any
               HttpSession session = request.getSession(false);
               if (session != null) {
                  session.invalidate();
               }
               session = request.getSession(true);
               synchronized (session) {
                  session.setAttribute("username", userName);
                  session.setAttribute("roles", roles);
               }

               out.println("<p>Hello, " + userName + "!</p>");
               out.println("<p><a href='dosomething'>Do Somethings</a></p>");
            }
         }
```

```
            out.println("</body></html>");

        } catch (SQLException ex) {
            out.println("<h3>Service not available. Try again later!</h3></body></html>");
            Logger.getLogger(LoginServlet.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            out.close();
            try {
               if (stmt != null) stmt.close();
               if (conn != null) conn.close();  // Return the connection to the pool
            } catch (SQLException ex) {
               Logger.getLogger(LoginServlet.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Inside the Login Session - "DoSomethingServlet.java" (URL "/dosomething")

```
package mypkg;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class DoSomethingServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      try {
         out.println("<html><head><title>Do Something</title></head><body>");
         out.println("<h2>Do Somethings...</h2>");

         // Retrieve and Display the username and roles
         String userName;
         List<String> roles;
         HttpSession session = request.getSession(false);
         if (session == null) {
            out.println("<h3>You have not login!</h3>");
         } else {
            synchronized (session) {
               userName = (String) session.getAttribute("username");
               roles = (List<String>) session.getAttribute("roles");
            }

            out.println("<table>");
            out.println("<tr>");
            out.println("<td>Username:</td>");
            out.println("<td>" + userName + "</td></tr>");
            out.println("<tr>");
            out.println("<td>Roles:</td>");
            out.println("<td>");
            for (String role : roles) {
               out.println(role + " ");
            }
            out.println("</td></tr>");
            out.println("<tr>");
            out.println("</table>");

            out.println("<p><a href='logout'>Logout</a></p>");
         }
         out.println("</body></html>");
      } finally {
         out.close();
      }
   }
}
```

The Logout Script - "LogoutServlet.java" (URL "/logout")

```
package mypkg;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LogoutServlet extends HttpServlet {
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
```

```
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      try {
         out.println("<html><head><title>Logout</title></head><body>");
         out.println("<h2>Logout</h2>");
         HttpSession session = request.getSession(false);
         if (session == null) {
            out.println("<h3>You have not login!</h3>");
         } else {
            session.invalidate();
            out.println("<p>Bye!</p>");
            out.println("<p><a href='index.html'>Login</a></p>");
         }
         out.println("</body></html>");
      } finally {
         out.close();
      }
   }
}
```

Web Application Deployment Descriptor "WEB-INF\web.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>mypkg.LoginServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>DoSomethingServlet</servlet-name>
    <servlet-class>mypkg.DoSomethingServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>mypkg.LogoutServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>DoSomethingServlet</servlet-name>
    <url-pattern>/dosomething</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/logout</url-pattern>
  </servlet-mapping>

  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

"META-INF\context.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/usertest" >
   <Resource name="jdbc/mysql_ebookshop" auth="Container" type="javax.sql.DataSource"
     maxActive="100" maxIdle="30" maxWait="10000" removeAbandoned="true"
     username="myuser" password="xxxx" driverClassName="com.mysql.jdbc.Driver"
     url="jdbc:mysql://localhost:3306/ebookshop" />
</Context>
```

4.2 User and Role Management via Container

[TODO] `HttpServletRequest`'s `authenticate()`, `login()` and `logout()`.

5. Input Validation

Observed that many lines of codes are used in validating the inputs provided by the client in the servlet (server-side program). You can perform input validation on the server-side as well as on the client-side.

5.1 Client-side Input Validation using JavaScript

We could perform client-side input validation using JavaScript. Validating user inputs with JavaScript before the data leaves the browser provides a much faster response, but it doesn't necessarily eliminate the checks you have to do on the server side. This is because a user might have disabled JavaScript or maliciously issue URLs that bypasses the client-side checks.

Read "JavaScript Examples - Validating Form Inputs".

5.2 Input Validation via JavaServer Faces (JSF)

[TODO]

6. Uploading Files (Servlet 3.0)

Before Servlet 3.0, processing file upload requires 3rd party libraries such as Apache Commons FileUpload. Servlet 3.0 (which requires Tomcat 7) builds in file upload support.

File upload over HTTP is specified in "RFC 1867 Form-based File Upload in HTML". Read "File Upload using multipart/form-data POST Request".

Client-side HTML Form: "FileUpload.html"

On the client-side, you provides an HTML `<form>` with an input element `<input type="file">` as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h2>Upload File</h2>
    <form method="post" enctype="multipart/form-data" action="upload">
      Choose a file: <input type="file" name="uploadedFile" /><br />
      <input type="submit" />
    </form>
  </body>
</html>
```

Server-side

Servlet 3.0 introduces a new annotation `@MultipartConfig` with these attributes:

1. `location`: An absolute path to a directory in your file system (NOT relative to your context root) to store files temporarily while processing parts, when the file is bigger than `fileSizeThreshold`. This directory shall exist; otherwise, an `IOException` will be thrown.
2. `fileSizeThreshold`: The file size in bytes after which the file will be stored temporarily in the `location`.
3. `maxFileSize`: The maximum file size in bytes. If the size of the file is bigger than this, Tomcat throws an `IllegalStateException`.
4. `maxRequestSize`: The maximum size in bytes for the entire `multipart/form-data` request (i.e., all the parts).

For example,

```
@MultipartConfig(location="d:\\temp\\upload", fileSizeThreshold=1024*1024, maxFileSize=5*1024*1024, maxRequestSize=2*5*1024*1024)
```

A new interface `javax.servlet.http.Part` is also introduced to represent a part of a form item that were received within a `multipart/form-data` POST request. The following methods are declared:

1. `getName()`: Get the name of this part.
2. `getSize()`: Get the size of this part.
3. `getInputStream()`: Get the content of this part as an `InputStream`.
4. `write(String filename)`: Write this part to file. The filename is relative to the "`location`" in `@MultipartConfig`. The container may simply rename the temporary file. Existing file will be overridden.
5. `delete()`: Delete the underlying storage, including the temporary file. Do not call `delete()` after `write()`.
6. `getContentType()`: Get the content type of this part.
7. `getHeader(String name)`, `getHeaders(String name)`, `getHeaderNames()`: Get the header.

The method `request.getParts()` (in `javax.servlet.http.HttpServletRequest`) returns a collection of all parts. The `request.getPart(String name)` returns a `Part` for given name attribute (if you have other input elements besides file).

"FileUploadServlet30.java" with URL "/upload"

```
package mypkg;

import java.io.*;
import java.util.Collection;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(
   name = "upload",
   urlPatterns = {"/upload"})
@MultipartConfig(
   location="d:\\temp\\upload",
   fileSizeThreshold=1024*1024,
   maxFileSize=5*1024*1024,
   maxRequestSize=2*5*1024*1024)
public class FileUploadServlet30 extends HttpServlet {

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
```

```
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      Collection<Part> parts = request.getParts();

      try {
         out.write("<h2>Number of parts : " + parts.size() + "</h2>");
         for(Part part : parts) {
            printPartInfo(part, out);
            String filename = getFileName(part);
            if (filename != null) {
               part.write(filename); // relative to location in @MultipartConfig
            }
         }
       } finally {
         out.close();
      }
   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
      getServletContext()
              .getRequestDispatcher("/FileUpload.html")
              .forward(request, response);
   }

   // Print the headers for the given Part
   private void printPartInfo(Part part, PrintWriter writer) {
      StringBuilder sb = new StringBuilder();
      sb.append("<p>Name: ").append(part.getName()).append("<br>");
      sb.append("ContentType: ").append(part.getContentType()).append("<br>");
      sb.append("Size: ").append(part.getSize()).append("<br>");
      for(String header : part.getHeaderNames()) {
         sb.append(header).append(": ").append(part.getHeader(header)).append("<br>");
      }
      sb.append("</p>");
      writer.write(sb.toString());
   }

   // Gets the file name from the "content-disposition" header
   private String getFileName(Part part) {
    for (String token : part.getHeader("content-disposition").split(";")) {
      if (token.trim().startsWith("filename")) {
        return token.substring(token.indexOf('=') + 1).trim()
            .replace("\"", "");
      }
    }
    return null;
  }
}
```

```
Total parts : 1

Name: uploadedFile
ContentType: text/plain
Size: 811
content-type: text/plain
content-disposition: form-data; name="uploadedFile"; filename="uploadedFile.txt"
```

1. The client-side "`FileUpload.html`" has one submission part, i.e., `<input type="file">`.
2. The `printPartInfo()` prints the headers of the given part. The output is as shown above.
3. The `part.write()` method is used to write the file under the `location` of the `@MultipartConfig` with the filename extracted from the `content-disposition` header.

An Multi-part HTML Form - "`FileUploadMultipart.html`"

The HTML form below has four input fields, which will be sent in four parts, as shown in the output below.


```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h2>Upload File</h2>
    <form method="post" enctype="multipart/form-data" action="upload">
```

```
      Who are you: <input type="text" name="username" /><br />
      Choose the file to upload: <input type="file" name="file1" /><br />
      Choose another file to upload: <input type="file" name="file2" /><br />
      Comments:<br />
      <textarea name="comment"></textarea><br />
      <input type="submit" value="SEND" />
    </form>
  </body>
</html>
```

```
Number of parts : 4

Name: username
ContentType: null
Size: 5
content-disposition: form-data; name="username"

Name: file1
ContentType: text/plain
Size: 811
content-type: text/plain
content-disposition: form-data; name="file1"; filename="uploadedFile.txt"

Name: file2
ContentType: audio/mpeg
Size: 4842585
content-type: audio/mpeg
content-disposition: form-data; name="file2"; filename="away.mp3"

Name: comment
ContentType: null
Size: 7
content-disposition: form-data; name="comment"
```

Notes:

1. You can use `request.getPart(`*name*`)` to retrieve a particular part with the given *name* attribute, instead of `request.getParts()`, which retrieves all parts in a `Collection<Part>`.
2. You can also use the following code to read the data from each part:

```
InputStream instream = request.getPart(part.getName()).getInputStream();
int byteRead;
while ((byteRead = instream.read()) != -1) {
   out.write(byteRead);
}
```

7. Deploying a Web Application in a WAR file

To deploy a Java webapp, you "zip" all the files and resources together in a single WAR (Web Application Archive) file. A WAR file, like a JAR file, uses ZIP algorithm, and can be opened using WinZIP or WinRAR.

You could use the JDK's `jar` utility to produce a WAR file as follows.

```
.... Change current directory to the webapp's context root
> jar cvf test.war .
```

(NetBeans) A war file is generated when you build the project, under the project's "`dist`" directory.

To deploy a WAR file, simply drop the WAR file (says `test.war`) into `$CATALINA_HOME\webapps`. A context called "`test`" will be created automatically. You can access the web application via URL `http://`*`host`*`:`*`port`*`/test`.

Tomcat will unpack the `test.war` into a "`test`" directory in `$CATALINA_HOME\webapps`, if the configuration option `unpackWARs="true"`. Otherwise, it will run directly from the WAR file without unpacking, which may involve some overhead. You may need to remove the unpakced directory, if you drop a new version.

8. A Secured Payment Gateway

[TODO]

REFERENCES & RESOURCES

1. TODO