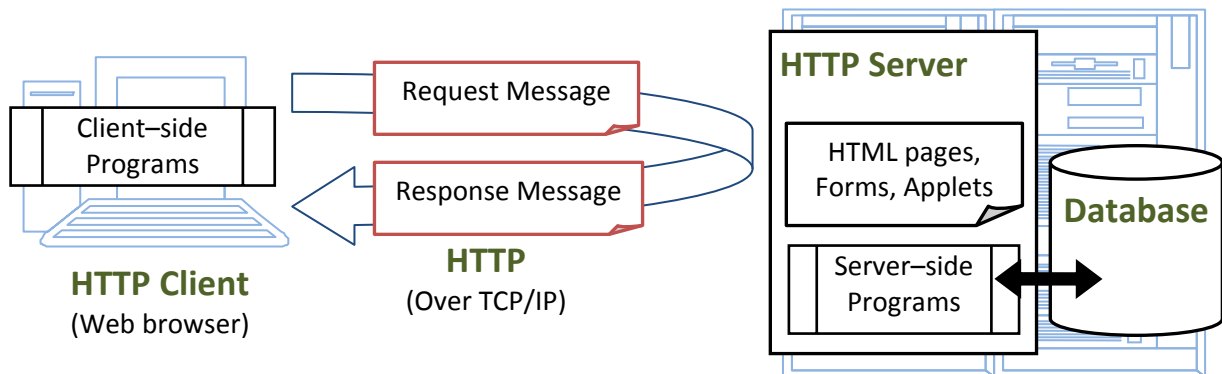


# Java Servlet Case Study: A Java-based E-shop

## 1. Objective

In this case study, we shall develop an *e-shop* based on the Java Platform. This *e-shop* is a typical *Internet business-to-consumer (b2c) 3-tier client/server database* application.

## 2. Overview



**Figure 1:** A typical 3-tier web database applications.

A typical *3-tier client/server web database application* consists of 5 components (as shown in **Figure 1**):

- A HTTP Server (or commonly known as Web Server), such as Apache HTTP Server, Apache Tomcat Server, GlassFish, or Microsoft Internet Information Server (IIS).
- A HTTP Client, typically a web browser, such as FireFox, or Chrome or IE.
- A Relational Database, such as Oracle, IBM DB2, MySQL (open-source), MS SQL Server, Informix, and Sybase, MS Access.
- Client-side programs, running inside the browser, which send requests to the server and process server's response. Client-side programs can be written in many ways, e.g., HTML form, JavaScript, VBScript, Java Applet, Flash, ActiveX Control, and others.
- Server-side programs, running inside the HTTP server, which process clients' request. The server-side programs extract the query information submitted by the client-side programs and perform the relevant database queries. Server-side programs can also be written in many ways, e.g., CGI Perl, Java Servlet/JSP/JSF, ASP, PHP, and others.

The client and server interact with each other by *exchanging messages* using a protocol called HTTP (HyperText Transfer Protocol). HTTP is an *asymmetric request-response protocol*. That is, the client sends a *request message* to the server. The server processes the request and returns a *response message*. In other words, in HTTP, the client *pulls* information from the server.

A typical sequence of operations for an e-shop is as follows:

- A client requests and downloads an HTML page containing an HTML form (or other client-side programs).
- The client enters information into the form (such as search criteria), and submits the information back to a server-side program for processing.
- The server-side program extracts the query information, performs the database query, and returns the query result back to the requesting client.
- The client displays the query result, and repeats step (b) to (d) for further request-response exchange.

Since this course is about Java, we shall build our web applications in Java. We shall write our server-side programs in *Java servlets* and *JSPs (JavaServer Pages)*. We shall write our client-side programs in *HTML forms* and *Java Applets*.

### What are Java Servlets?

*Servlets* are Java programs that add functionality to a web server in a manner similar to the way *applets* add functionality to a browser. Servlets execute on the *Java-capable HTTP Server*, while applets are downloaded from a server and execute on the *Java-capable web browser*. Java Servlet is the *foundation* technology for Java server-side programming. Other technologies such as JSP (JavaServer Pages), JSF (JavaServer Faces) are built on top of the Java Servlet technology.

## 3. Setting up the Database

**Important:** I assume that you are familiar with relational database and SQL or have done the earlier “Database programming exercises”.

The first step in building our e-shop is to setup a database. We shall call our database “**eshop**” which contains only one table “**books**”, as illustrated below:

books
isbn: INTEGER (PRIMARY KEY, NOT NULL)
title: VARCHAR(50)
author: VARCHAR(50)
price: FLOAT
qty: INTEGER

The table “books” has 5 columns: isbn, title, author, price and qty. isbn is the *primary key* of the table, having unique value and cannot be NULL.

Insert these records into the table books:

isbn	title	author	price	qty
1001	Java for Dummies	Tan Ah Teck	11.11	11
1002	More Java for Dummies	Tan Ah Teck	22.22	22
1003	Java ABC	Mohamad Ali	33.33	33
1004	Only Java	Mohamad Ali	44.44	44
1005	A Cup of Java	Kumar	55.55	55
1006	A Teaspoon of Java	Kevin Jones	66.66	66
...	...	...	...	...

### MS Access with ODBC Connection

Create a new database called “eshop.mdb” (Access 2003) or “eshop.accdb” (Access 2007) and save in your directory “d:\workshop”. Name the columns and enter the above records.

Close the Access before proceeding to the next step to define ODBC connection (otherwise, you will get an error “invalid directory or file path”).

An ODBC connection is needed to connect to the Access database. Define an ODBC connection called “eshopODBC” which selects the database you have just created, i.e., “eshop.mdb” or “eshop.accdb”.

From “Control Panel” ⇒ “Administrator Tools” ⇒ “Data Source (ODBC)” ⇒ Choose “**System DSN**” (System Data Source Name) tag ⇒ “Add” ⇒ select “Microsoft Access Driver (\*.mdb)” or “Microsoft Access Driver (\*.accdb)” ⇒ click “Finish” ⇒ Enter “eshopODBC” in “Data Source Name” ⇒ use “Select” to select “eshop.mdb” or “eshop.accdb” ⇒ OK.

## MySQL

Start a MySQL client (or monitor) and run these SQL statements:

```
mysql> create database if not exists eshop;
mysql> use eshop;
mysql> CREATE TABLE books (isbn INTEGER NOT NULL PRIMARY KEY,
    title VARCHAR(50), author VARCHAR(50), price FLOAT,
    qty INTEGER);
mysql> INSERT INTO books VALUES (1001, 'Java for dummies',
    'Tan Ah Teck', 11.11, 11);
mysql> INSERT INTO books VALUES (1002, 'More Java for dummies',
    'Tan Ah Teck', 22.22, 22);
mysql> .....
mysql> exit
```

(Alternatively, you could use the GUI tool “MySQL Administrator” to set up the database.)

## 4. Apache Tomcat Server

Next, we have to setup a HTTP server to handle HTTP requests from the clients. In this case study, we shall use Apache’s Tomcat Server as our HTTP server. Tomcat is an *open-source* project @ <http://tomcat.apache.org>, provided *free* by Apache Software Foundation @ <http://www.apache.org>. Tomcat is the official *Reference Implementation* for Java Servlets and JSP.

Read [http://www3.ntu.edu.sg/home/ehchua/programming/howto/Tomcat\\_HowTo.html](http://www3.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_HowTo.html) on how to install and configure Tomcat server.

**Step 1: Configure the Tomcat Server.** Suppose that Tomcat has been installed in directory “d:\tomcat”. To configure the web server, edit the Tomcat’s configuration file “d:\tomcat\conf\server.xml”. You can change the HTTP port number (which is set to 8080) and add new *contexts* (i.e., *web applications*) to your web server.

Let’s define a new *context* (*web application*) called “ws” (case-sensitive):

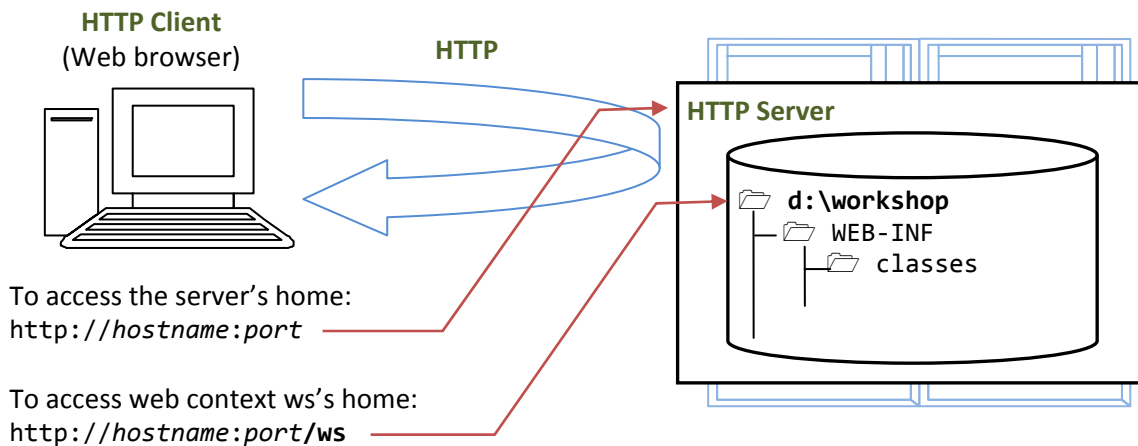
(a) Add the following lines to the configuration file “d:\tomcat\conf\server.xml” before the </Host> end-tag:

```
.....
<Context path="/ws" docBase="d:/workshop" reloadable="true">
</Context>
</Host>
.....
```

These lines defines a *context-path* called “/ws”. Clients can access this application by issuing URL <http://hostname:port/ws>.

The *docBase* (*document base* directory or *context root* or *home* directory) specifies the directory where you keep your application codes in the server’s file system. In this case, you should keep your application codes for this context in “d:\workshop”.

Observe that there is a *mapping* between URL pathname (for Internet clients to access this web application) and the directory in the server's file system, as illustration in Figure 2.



**Figure 2:** Mapping of context URL-Path to the document base directory in the server's file system.

- (b) Create a sub-directory called "WEB-INF" under the root directory "d:\workshop" and create a sub-sub-directory "classes" under "WEB-INF" (case-sensitive!). You have to store your Java servlets in "d:\workshop\WEB-INF\classes" in your later exercises.

**Step 2: Start the Tomcat Server.** To start the Tomcat server, run the batch file "d:\tomcat\bin\startup.bat" (from a *Command shell*). Tomcat will be started in a new console window. Monitor this console, as the information and error messages, and `System.out.println()` issued by your programs will be sent to this console.

**Step 3: Access the Tomcat Server.** The Tomcat Server has been started on TCP port 8080. The default TCP port number for HTTP protocol is 80. To access a HTTP server not running on the default TCP port 80, the port number must be explicitly specified in the URL.

To access your Tomcat Server, start a web browser (e.g., FireFox , IE or Chrome) and issue the following HTTP URL:

`http://hostname:8080`

The *hostname* is pasted on your PC. (To be exact, for Windows XP, the *hostname* can be found in Control Panel → System → Computer Name → Full computer name.)

(If you are working on your own computer, make sure that you don't have any special character or white space in your hostname – they are not allowed in the URL.)

You could also use the IP address to access your HTTP server. You can find out the IP address by running program such as "ipconfig", "winipcfg", "ping", and etc.

IP provides a *local loop-back* testing *hostname* called localhost (with IP address of 127.0.0.1). If you are *testing* from the same PC that runs your HTTP server, you could use:

`http://localhost:8080`  
`http://127.0.0.1:8080`

You shall see the *home page* your Tomcat Server.



**Step 4: Access your Application's Home Directory.** You can access your application's *home directory*, by issuing an URL selecting *your context* `"/ws"` as follows:

```
http://hostname:8080/ws
```

You shall see the directory listing of your home directory `"d:\workshop"`. (Because the context `"/ws"` is mapped to docbase of `"d:\workshop"`.)

You should keep all your HTML files and applets in your home directory so that they can be accessed by your clients.

Try clicking the file `"HelloApplet.html"`, which contains your first Java applet `"HelloApplet.class"`. Now, the applet will go thru the Internet before running on your local machine.

You can check out the home page of *your peers* by issuing:

```
http://YourPeerHostname:8080/ws
```

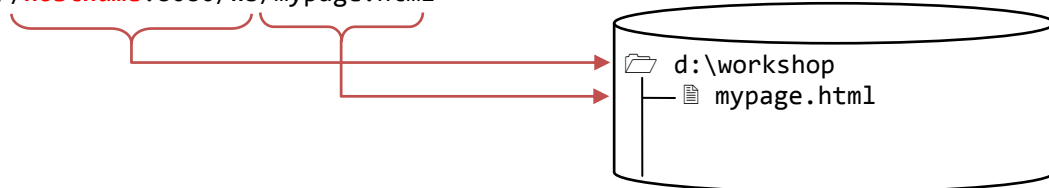
with a valid `"YourPeerHostname"`, provided that your peer has started his/her web server.

**Step 5: Write your Home Page.** Create the following HTML page and save as `"mypage.html"` in your home directory (i.e. `"d:\workshop"`).

```
<html>
<head>
  <title>My Home Page</title>
</head>
<body>
  <h1>My Name is so and so. This is my HOME.</h1>
</body>
</html>
```

You can browse this page by issuing:

```
http://hostname:8080/ws/mypage.html
```



If you issue a URL for a *directory request*:

```
http://hostname:8080/ws
```

A listing of your home directory `"d:\workshop"` will be displayed. You can then click the item `"myspage.html"`.

If you save the HTML page as `"index.html"`, the page will be automatically shown when you make a *directory request*, i.e., `"http://hostname:8080/ws"`.

**Step 6: Shutting down the Tomcat Server.** To orderly shutdown the Tomcat web server, run the batch file `"d:\tomcat\bin\shutdwon.bat"`, from a Command shell.

## 5. Set up a Simple Client-side Query HTML Form

Let's write an HTML script to create a *"query form"* using checkboxes (as shown in Figure 3). Save the HTML file as `"Query.html"` in your home directory `"d:\workshop"`. Replace the *hostname* with your IP address, DNS name or machine name.

```

<html>
<head>
  <title>One More Bookshop</title>
</head>
<body>
  <h2>One More Bookshop</h2>
  <form method="get"
    action="http://hostname:8080/ws/servlet/HardCodedServlet">
    <b>Choose an author:</b>
    <input type="checkbox" name="author" value="Tan Ah Teck">Ah Teck
    <input type="checkbox" name="author" value="Mohamad Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <input type="submit" value="Search">
  </form>
</body>
</html>

```

Browse the HTML page by issuing the URL  
<http://hostname:8080/ws/Query.html>

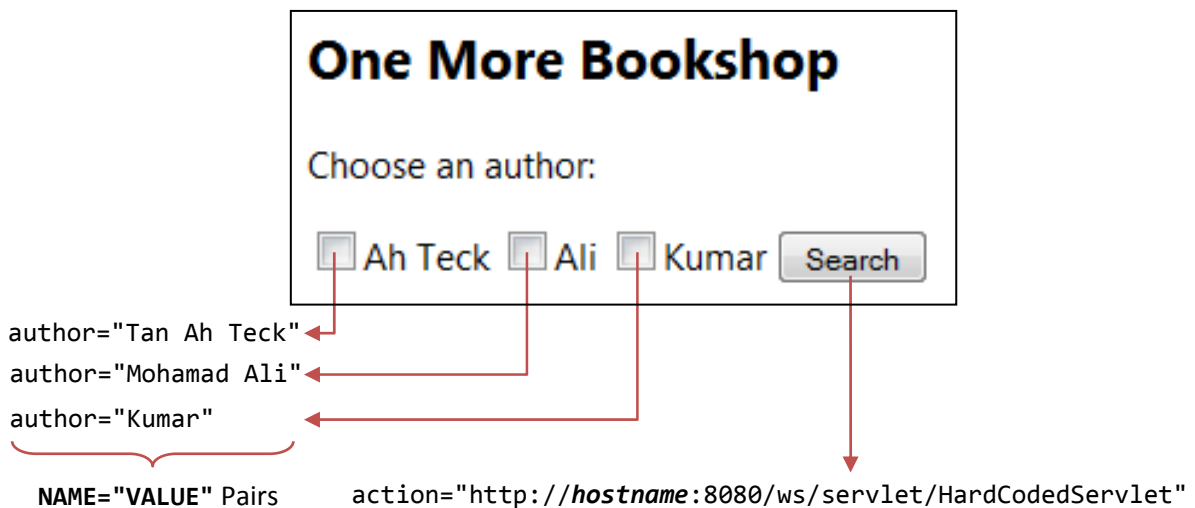
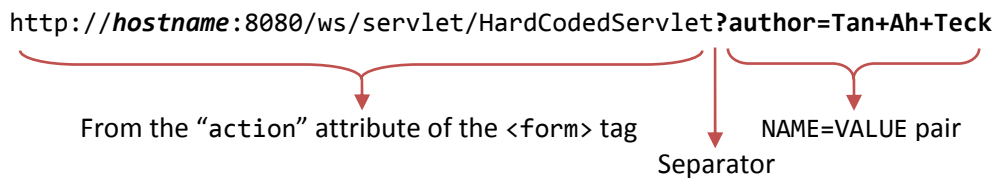


Figure 3: A HTML “Form” and the input fields’ NAME=VALUE pairs.

Check a box and click the “Search” button. An HTTP GET request will be issued to the HTTP server as specified in the “action” attribute of the <form> tag. Observe the URL of the HTTP GET request:



You are expected to receive an “Error 404: Page Not Found” at this stage as you have yet to write the server-side program (i.e., “HardCodedServlet”).

## 6. A simple Java Database Servlet with a Hard-coded SQL Statement

The next step is to write the server-side program, which responds to the client’s request by querying the database. We shall write use Java servlet technology in our servlet-side programming.

Recap that the steps involved in database programming are:

**Step 1:** Load the JDBC driver.

**Step 2:** Create a Connection object.

**Step 3:** Create a Statement object, from the Connection object created.

**Step 4:** Execute a SQL SELECT query by calling the `executeQuery()` method of the Statement object, which returns the query result in a `ResultSet` object; or

Execute a SQL INSERT|UPDATE|DELETE command by calling the `executeUpdate()` method of the Statement object, which returns an `int` indicating the number of rows affected.

**Step 5:** Process the query result.

**Step 6:** Free the resources by closing the `ResultSet` (for `executeQuery()`), `Statement` and `Connection`.

Let's begin by writing a simple Java servlet with a *hard-coded* SQL SELECT statement to illustrate how the servlet interfaces with the database. (In the next section, instead of *hard-coding* the SQL statement, you will learn how to form a query based on the client's request.)

**Step 1: Write the servlet:** enter the following codes and save as:

"d:\workshop\WEB-INF\classes\HardCodedServlet.java".

Tomcat server picks up the servlets from sub-directory "WEB-INF\classes" of your context's root directory. That is to say, **you must keep all your servlets in the directory "d:\workshop\WEB-INF\classes"**.

```
// Saved as "d:\workshop\WEB-INF\classes\HardCodedServlet.java".
import java.io.*;           // needed for I/O processing
import java.sql.*;         // needed for SQL query
import javax.servlet.*;    // Generic Servlet
import javax.servlet.http.*; // HttpServlet

public class HardCodedServlet extends HttpServlet {

    private Connection conn;    // A shared Connection
    private Statement stmt;    // A shared SQL statement

    // init() runs when servlet is loaded into the server
    public void init() throws ServletException {
        try {
            // Step 1: load JDBC driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // Access
            // Step 2: Create a database connection
            conn = DriverManager.getConnection("jdbc:odbc:eshopODBC");
            // Step 3: Create a Statement object inside the Connection
            stmt = conn.createStatement();
        } catch (ClassNotFoundException ex) { // for Class.forName()
            ex.printStackTrace();
        } catch (SQLException ex) { // for java.sql methods
            ex.printStackTrace();
        }
    }

    // doGet() runs once per HTTP GET request to this servlet.
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        // Set the MIME type for the response message
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
    }
}
```

```

try {
    // Step 4: Execute a SQL statement
    // Use a hardcoded SQL statement here!
    String sqlStr =
        "SELECT * FROM books WHERE author='Tan Ah Teck'";
    // print an HTML page as output of query
    out.println("<h3>Thank you for your query.</h3>");
    out.println("<p>You query is: " + sqlStr + "</p>");
    ResultSet rset = stmt.executeQuery(sqlStr);

    // Step 5: Process the query result
    int count = 0;
    while(rset.next()) {
        out.println("<p>" + rset.getString("author")
            + ", " + rset.getString("title")
            + ", $" + rset.getDouble("price") + "</p>");
        count++;
    }
    out.println("<p>==== " + count + " records found =====</p>");

    // Step 6: Close the ResultSet
    rset.close();
} catch (SQLException ex) { // for java.sql methods
    ex.printStackTrace();
}

// destroy() runs when the servlet is unloaded from the server
public void destroy() {
    try {
        // Step 6: Close the Connection
        stmt.close();
        conn.close();
    } catch (SQLException ex) { // for java.sql methods
        ex.printStackTrace();
    }
}
}
}

```

## MySQL

The above code is meant for MS Access. If you are using MySQL, replace step 1 and step 2 by:

```

// Step 1: Load JDBC Driver
Class.forName("com.mysql.jdbc.Driver");
// Step 2: Create a database Connection object
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://127.0.0.1/eshop", "username", "password");

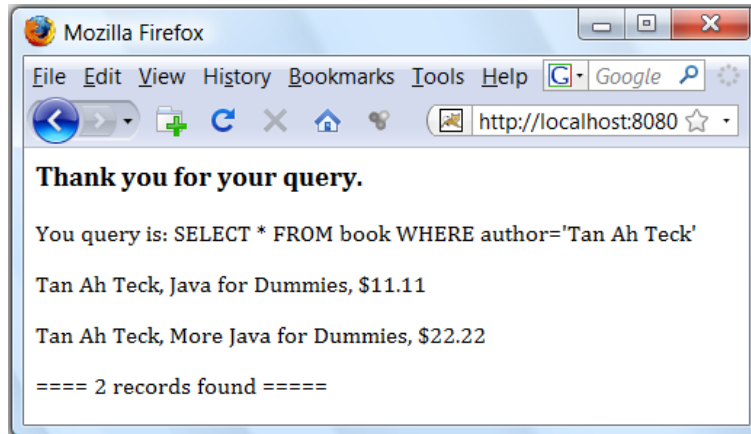
```

**Step 2:** Compile the “HardCodedServlet.java” into “HardCodedServlet.class”

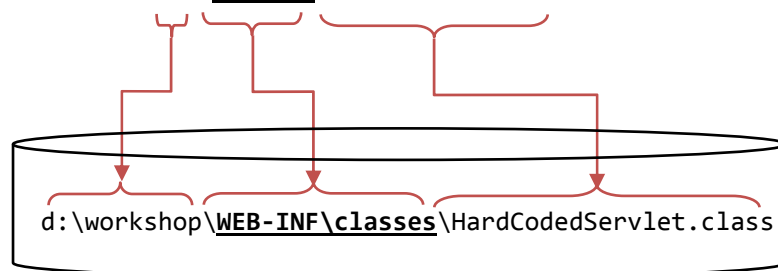
**Step 3:** You can now try out the servlet. To invoke a servlet, issue an HTTP URL as follows:

http://**hostname**:8080/ws/servlet/HardCodedServlet

This URL says that you are looking for a “servlet” called “HardCodedServlet.class”, under the *web context* “ws”. Tomcat will look for the servlet from the servlet directory of the context (i.e. “d:\workshop\WEB-INF\classes”). In other word, the keyword “servlet” in the URL maps to directory “WEB-INF\classes” in the server’s file system.



http://hostname:8080/ws/servlet/HardCodedServlet



In the browser, choose “View” ⇒ “Source” to study the output produced by the servlet. It is important to note that the client has no access to the servlet’s codes, but merely the outputs produced by the servlet.

```
<h3>Thank you for your query.</h3>
<p>You query is: SELECT * FROM books WHERE author='Tan Ah Teck'</p>
<p>Tan Ah Teck, Java for Dummies, $11.11</p>
<p>Tan Ah Teck, More Java for Dummies, $22.22</p>
<p>==== 2 records found =====</p>
```

## Exercises

- (a) Use the client-side HTML form that you have created earlier (i.e., “Query.html”) to *trigger* this HardCodedServlet. (Observe that the action attribute of <form> tag in Query.html is set to HardCodedServlet).
- (b) If you are not familiar with SQL, try out other SQL SELECT statements such as:
  - i. To select all the records from table books:
 

```
SELECT * FROM books
```
  - ii. To select all the records from table books by the author named 'Tan Ah Teck'
 

```
SELECT * FROM books WHERE author = 'Tan Ah Teck'
```

(Note: string in SQL is enclosed with single quotes).
  - iii. To select columns author, title and price from the table books for price below 50 and qty more than 0; and order the result in descending author, then ascending price:
 

```
SELECT author, title, price FROM books
WHERE price < 50 AND qty > 0
ORDER BY author DESC, price ASC
```

In summary:

- (a) Java *standalone programs* are run in a computer using the Java runtime "java.exe".
- (b) Java *applets* are embedded in an HTML page and run inside a web browser using the web browser's built-in JRE.
- (c) Java *servlets* are run inside a web server in response to an HTTP request from the web browser. The servlet typically returns an HTML page to the client as the response.

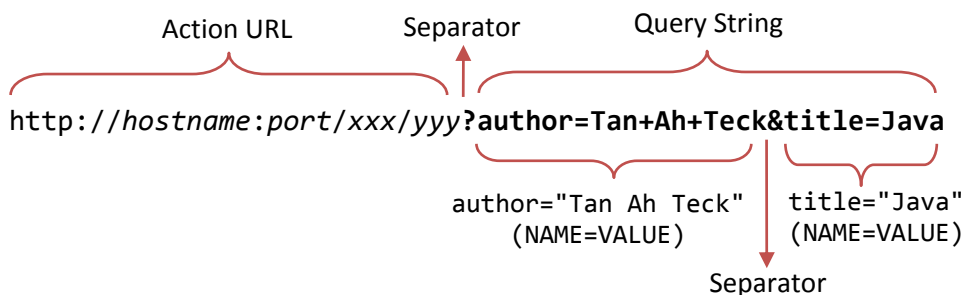
### Brief Explanation of the program:

- (a) There are three methods in a servlet. The method `init()` runs when the servlet is loaded into the servlet's container. The method `destroy()` runs when the servlet is unloaded from the servlet's container. The method `doGet()` runs *once for each* HTTP GET request to this servlet.
- (b) During the initialization in the `init()` method, the servlet loads the JDBC driver. It then creates a Connection object and a SQL Statement object, to be used throughout the servlet's life-time. The connection and Statement objects will be shared and re-used by all subsequent requests. During the termination, the `destroy()` method closes the Statement and Connection.
- (c) For each HTTP GET request to this servlet, `doGet()` method will be invoked once. This method takes two arguments, a request (of `HttpServletRequest`) and a response (of `HttpServletResponse`) representing the request and response messages. `doGet()` processes the request coming through the HTTP, and responds according to the program logic.
- (d) In this simple servlet, a SQL query statement is hard-coded. The servlet executes the SQL query and returns the query result. The result is written out as an HTML page and sends back to the requesting client.

## 7. Java Servlets with User-supplied Search Parameters

Notice that the simple servlet you have just created contains a *hard-coded* SQL statement. In actual usage, the query criteria must come from the client and cannot be hard-coded.

Recall that an HTTP GET request appends a *query string* after the URL. An URL *with a query string* looks like:



The query string is made up of NAME=VALUE pairs. In the above query string, two sets of NAME=VALUE pairs are present: `author="Tan Ah Teck"` and `title="Java"`.

Notice that the query string is appended behind the URL after a '?'. The NAME=VALUE pairs are separated by an '&'. Special characters are not permitted in the URL, they are *encoded* with a '%' sign followed by their hexadecimal value, e.g., '~' is encoded into "%7e". "Blank" is encoded into "%20" or "+".

We shall modify our hard-coded servlet to accept HTTP GET request *with query string* and translate the query parameter into an appropriate SQL statement. For example, if the following URL is issued:

`http://hostname:8080/ws/servlet/QueryServlet?author=Tan+Ah+Teck`

We shall create a SQL SELECT as follows:

```
SELECT * FROM books WHERE author = 'Tan Ah Teck'
```

The name of the table and the columns are still *hard-coded*. However, the query criterion is supplied by the clients.

**Step 1:** Modify the `HardCodedServlet` as follows:

- i. Change the class name from `HardCodedServlet` to `QueryServlet` and save the program as `d:\workshop\WEB-INF\classes\QueryServlet.java`.
- ii. Modify the SQL query string `sqlStr` in the `doGet()` method to:

```
String sqlStr = "SELECT * FROM books WHERE author = "  
                + "'" + request.getParameter("author") + "'" + "  
                + " AND qty > 0 ORDER BY price DESC ";
```

(The SQL string is broken up into many substrings and concatenated with '+' operator for readability. Make sure to provide a white space between substrings.)

The `request.getParameter("NAME")` method returns the "VALUE" assigned to the `NAME=VALUE` pair. For example, if the HTTP URL is:

```
http://hostname:8080/ws/servlet/QueryServlet?author=Tan+Ah+Teck
```

The method `request.getParameter("author")` returns a String `"Tan Ah Teck"`.

The resultant SQL statement `sqlStr` becomes:

```
SELECT * FROM books  
WHERE author='Tan Ah Teck' AND qty > 0  
ORDER BY price DESC
```

Note that you do not have to handle encoded URL characters such as `"%20"`, `"+"`, `"?"` and `"&"`. They will be properly decoded by the `getParameter()` method.

**Step 2:** Compile the program and try out the servlet by issuing a proper HTTP GET request with a appropriate query string, e.g.,  
`http://hostname:8080/ws/servlet/QueryServlet?author=Tan+Ah+Teck`

**Step 3:** Modify the client-side HTML form `Query.html` to *trigger* this `QueryServlet`, as follows:

```
<form method="get"  
  action="http://hostname:8080/ws/servlet/QueryServlet">
```

## Relative URL vs. Absolute URL

In the `Query.html`, the URL in the `action` attribute (i.e., `http://hostname:8080/ws/servlet/QueryServlet`) is called an *absolute URL*. The hostname, port number and context path are all hard-coded in an absolute URL. This will cause problem if you decide to relocate your program to another host (e.g., from the testing host into the production host).

Instead of using absolute URL, we would use a *relative URL* as follows:

```
<form method="get" action="servlet/QueryServlet">
```

Relative URL is *relative* to the *currently displayed page*. Since the current page `Query.html` is located at directory `http://hostname:8080/ws`, the *relative* URL of `servlet/QueryServlet` resolves into an *absolute* reference of `http://hostname:8080/ws/servlet/QueryServlet`.

Relative URL should be used instead of absolute URL in your HTML scripts, whenever possible, so that the HTML pages can be easily relocated from one web context to another web context, or to another server, under difference base directory. You should only use absolute URL for referencing resources from other server.

Try it out!

### More Exercise (1): Multi-value Query Parameter

If you check more than one author and submit your request to the QueryServlet, the query result shows only the first author. Modify the QueryServlet (call it QueryMultiValues) to handle one query parameter with multiple values, e.g.,

```
http://host:8080/ws/servlet/QueryMultiValues?author=kumar&author=Ali
```

You have to use method `request.getParameterValues()` instead of `request.getParameter()` to handle multi-value parameters. `request.getParameterValues()` returns an *array* of String containing all the values of that parameter (whereas `request.getParameter()` returns a single String).

We can use the IN predicate in the WHERE clause to check for a set of values, for example,

```
SELECT * FROM books
WHERE author IN ('Tan Ah Teck', 'Mohamad Ali', 'Kumar')
```

The above statement is the same as:

```
SELECT * FROM books
WHERE author='Tan Ah Teck' OR author='Mohamad Ali' OR author='Kumar'
```

Let's include all the selected authors in our servlet as follows:

```
String[] authors = request.getParameterValues("author");
String sqlStr = "SELECT * FROM books WHERE author IN (";
sqlStr += "'" + authors[0] + "'";
for (int i = 1; i < authors.length; i++) {
    sqlStr += ", '" + authors[i] + "'";
}
sqlStr += ") AND qty > 0 ORDER BY price DESC";
```

The `getParameterValues(NAME)` and `getParameter(NAME)` returns null if the query string does not contain NAME (i.e., the user did not check any box). This will cause an exception in using `authors.length` in the above codes. You can use the following expression to check for the existence of a parameter:

```
if (request.getParameterValues("author") == null) {
    out.println("<p>Please go back and select an author</p>");
} else {
    // Perform the database query
    .....
}
```

### More Exercise (2): Multiple Query Parameters

Modify the QueryServlet (call it QueryMultiParams) to handle two query parameters, e.g.,

```
http://host:8080/ws/servlet/QueryMultiParams?author=kumar&price=50
```

You can form the SQL statement as follows:

```
String sqlStr = "SELECT * FROM books WHERE author ="
    + " '" + request.getParameter("author") + "'"
    + " AND price < " + request.getParameter("price")
    + " AND qty > 0 ORDER BY price DESC ";
```

Note that:

- i. In the SQL SELECT statement, “author” is a string and must be enclosed by a pair of single quotes; “price” is a number and cannot be quoted.
- ii. The “NAME=VALUE” pair of price=“50” is interpreted as price<50 in the SELECT.

Modify your client-side HTML form to submit two query criteria, as follows.

Note that:

- i. *Checkboxes* are used for author and *radio buttons* are used for price.
- ii. A “clear” button is added (type=“reset”)

```
<html>
<head>
  <title>One More Bookshop</title>
</head>
<body>
  <h2>One More Bookshop</h2>
  <form method="get" action="servlet/QueryMultiParams">
    <b>Choose an author:</b>
    <input type="checkbox" name="author" value="Tan Ah Teck">Ah Teck
    <input type="checkbox" name="author" value="Mohamad Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <br><b>Choose a price range:</b>
    <input type="radio" name="price" value="50">less than $50
    <input type="radio" name="price" value="100">less than $100
    <br><input type="submit" value="Search">
    <input type="reset" value="Clear">
  </form>
</body>
</html>
```

### More Exercise (3): HTTP POST Request

Try out the HTTP POST method:

- i. Modify the HTML form to submit a POST request (instead of a GET request) by changing the attribute method=“get” to method=“post” in the <form> tag;
- ii. Modify the QueryServlet (call it QueryPost) by renaming the method doGet() to doPost().

What is the difference between POST and GET? For POST request, the query string is not shown in the URL (i.e., no “?author=Tan+Ah+Teck&...”). Instead, the query string is sent in the *body* of the HTTP request message. The advantages are: (a) client will not see the query string; (b) GET request’s query string length is limited, because it is part of the URL. POST request can send unlimited data.

In practice, it is common to use the *same* method to handle both the GET and POST requests. In this case, you could simply redirect doPost() to doGet() as follows:

```
public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    .....
    .....
}
```

```

public void doPost (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}

```

## 8. Programming the Client-side (Front-end)

So far, we have setup the server-side programs (using Java servlets) to process HTTP request. Let's continue on the front-end (i.e. the client-side). We shall try the traditional HTML "form", followed by Java applet as the client-side programs.

In previous section, we created a simple HTML "form" of *checkboxes*. HTML also provides other input fields, such as text field, radio button, pull down menu, and etc.

### HTML Form with "Pull-down Menu" as Front-end

Create the following HTML *form* with "pull-down menu" and save as "QueryWithMenu.html" in your home directory "d:\workshop"

```

<html>
<head>
  <title>One More Bookshop</title>
</head>
<body>
  <h2>One More Bookshop</h2>
  <form method="get" action="servlet/QueryServlet">
    <b>Choose an author:</b>
    <select name="author" size="1">
      <option value="Tan Ah Teck">Ah Teck</option>
      <option value="Mohamad Ali">Ali</option>
      <option value="Kumar">Kumar</option>
    </select>
    <input type="submit" value="Search">
  </form>
</body>
</html>

```

Try out the other HTML form's input components such as text field, radio button, etc. Check your notes on the details of coding these components.

### A Java Applet as the Client-side Program

Instead of a simple HTML *Form*, we shall now use a *Java applet* as the client-side program. Enter the following codes and save as "AppletQuery.java" in your home directory "d:\workshop".

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;          // needed for constructing URL
import java.io.*;          // needed to receive output from server

public class AppletQuery extends Applet implements ActionListener {
  private TextField tfAuthor;    // For entering the author name
  private Button btnSearch;     // Search button
  private Button btnClear;      // Clear button

```

```

public void init() {
    // Create GUI
    add(new Label("Enter the Author: "));

    tfAuthor = new TextField(15);
    add(tfAuthor);

    btnSearch = new Button("Search");
    add(btnSearch);
    btnSearch.addActionListener(this);

    btnClear = new Button("Clear");
    add(btnClear);
    btnClear.addActionListener(this);
}

// Event handler for Buttons
public void actionPerformed(ActionEvent evt) {
    if (evt.getSource() instanceof Button) { // Button click?
        String str = evt.getActionCommand();
        if (str.equals("Search")) { // Search Button
            try {
                // Create the HTTP GET URL, and invoke the link
                // Use relative URL. getCodeBase() returns the base.
                URL u = new URL(getCodeBase(),
                    "servlet/QueryServlet?author=" + tfAuthor.getText());

                // receive the output from the server
                getAppletContext().showDocument(u);
            } catch (MalformedURLException ex) {
                showStatus("Err: " + ex);
            } catch (IOException ex) {
                showStatus("Err: " + ex);
            }
        } else if (str.equals("Clear")) { // Clear button
            tfAuthor.setText("");
        }
    }
}
}
}
}

```

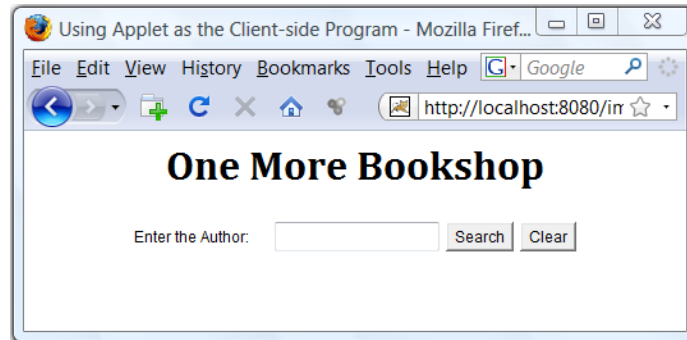
Write an HTML script "AppletQuery.html" to embed the applet and save in your home directory "d:\workshop":

```

<html>
<head>
    <title>Using Applet as Client-side Program</title>
</head>
<body>
    <h1 align="center">One More Bookshop</h1>
    <applet code="AppletQuery.class" width="500" height="100">
    </applet>
</body>
</html>

```

Browse the page to run the applet and observe the result.



## 9. Placing an Order

So far, we have tested the “query” part of the e-shop. Let’s continue to allow the client to place an order in the e-shop.

The sequence of events shall be:

- (a) A client issues URL `http://hostname:8080/ws/EshopQuery.html`. The server responds with a list of available authors (in checkboxes).
- (b) The client checks the authors and sends the query request to the server. The server retrieves the authors, queries the database, and returns a list of available books by the authors. The books are display inside another HTML form, so as to allow the client to order.
- (c) The client checks the books and sends the order request to the server. The server retrieves the book information, updates the database, and returns a confirmation response to the client.

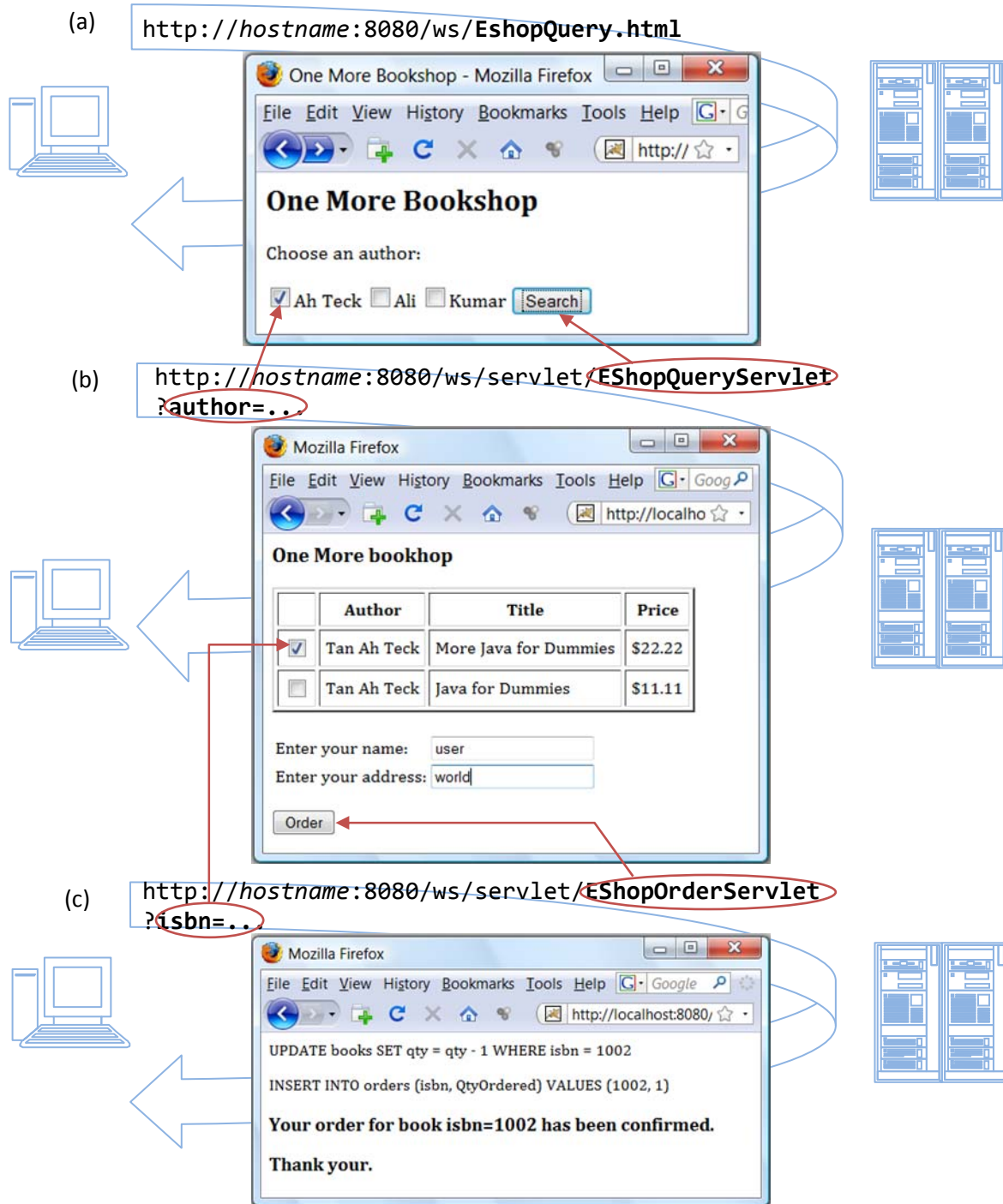


Figure 4: Sequence of Events

The QueryServlet that we wrote earlier returns a *static* HTML page. We shall modify it (called EshopQueryServlet) to return a *dynamic* HTML page containing a *form* of checkboxes to allow the client to order books as shown in Figure 4(a).

Recall that a checkbox is produced by an `<input>` tag as follows:

```
<input type="checkbox" name="aName" value="aValue">aLabel
```

We shall use isbn as the ordering criterion. Hence, isbn is used as the NAME=VALUE pair of the checkbox. We shall use author, title and price as *label* for the checkbox. Hence, the EshopQueryServlet must produce `<input>` tag which looks like:

```
<input type="checkbox" name="isbn" value="1001">
Tan Ah Teck, Java For Dummies, $11.11
```

EshopQueryServlet shall send the order to another servlet called EshopOrderServlet in the same directory (which we shall write later). Hence the <form> tag shall look like:

```
<form method="get" action="EshopOrderServlet">
  <input ...>
  <input ...>
</form>
```

Modify the QueryServlet: (i) change the class name to “EshopQueryServlet”; (ii) change the ResultSet processing to generate the desired HTML <form> and <input> tags as described above.

```
// Print the <form> begin tag
out.println("<form method=\"get\" action=\"EshopOrderServlet\">");
// For each row in ResultSet, print one checkbox
while(rset.next()) {
    out.println("<p><input type=\"checkbox\" name=\"isbn\" value=\"
        + \"\" + rset.getString(\"isbn\") + \"\">\"
        + rset.getString(\"author\") + \", \"
        + rset.getString(\"title\") + \", $\"
        + rset.getString(\"price\") + "</p>");
}
// Print the submit button and </form> end tag
out.println("<p><input type=\"submit\" value=\"ORDER\">");
out.println("</form>");
```

Modify your Query.html (call it EshopQuery.html) to trigger “EshopQueryServlet”.

Launch EshopQuery.html. Select an author and submit the request to EshopQueryServlet. Observe the output (via “View” → “Source”).

“Order” a book and observe the HTTP GET request triggered. (But you are expected to receive a 404 error because you have yet to write the EshopOrderServlet).

## 10. Processing the order

Placing the order (Figure 4(b)) invokes another servlet called “EshopOrderServlet” (Figure 4(c)). The URL triggered looks like:

```
http://hostname:8080/ws/servlet/EshopOrderServlet?isbn=1001
```

Let’s write the “EshopOrderServlet” to process the order by:

- (a) Retrieve the isbn number from the request.
- (b) In table books, reduce the “qty” by 1 for that particular isbn number. To update records in the books table, use SQL UPDATE statement as follows:  

```
UPDATE books SET qty = qty - 1 WHERE isbn = 1234
```
- (c) Create a transaction record in a new table called “orders” and insert a transaction record.

The steps are as follows:

**Step 1:** Create a new table called “orders” in the same database with four columns: isbn (int), qtyOrdered (int), orderBy (String), and shippingAddr (String).

**Step 2:** Write the EshopOrderServlet which retrieve the isbn number from the request, and update the database:

In SQL, to insert a complete record into the orders table, use INSERT statement as follows:

```
INSERT INTO orders
VALUES (1001, 2, 'Kelvin Jones', '99 Sunset Ave')
```

To insert a *partial* record with selected fields into the orders table, use INSERT statement as follows:

```
INSERT INTO orders
(isbn, qtyOrdered) VALUES (1002, 1)
```

To execute SQL UPDATE and INSERT INTO statements, you have to use method executeUpdate(sqlStr) of the Statement object. The method executeUpdate(sqlStr) returns an int, indicating the number of rows affected by the UPDATE or INSERT, e.g.,

```
int count;
count = stmt.executeUpdate(
    "INSERT INTO orders (isbn, qtyOrdered) VALUES (1234, 1)")
```

The doGet() of EshopOrderServlet shall look like:

```
String[] isbnns = request.getParameterValues("isbn");
if (isbnns != null) {
    String sqlStr;
    for (int i = 0; i < isbnns.length; i++) {
        sqlStr = "UPDATE books SET qty = qty - 1 WHERE isbn = " +
            isbnns[i];
        out.println("<p>" + sqlStr + "</p>"); // for debugging
        stmt.executeUpdate(sqlStr);
        sqlStr = "INSERT INTO orders (isbn, QtyOrdered) VALUES ("
            + isbnns[i] + ", 1)";
        out.println("<p>" + sqlStr + "</p>"); // for debugging
        stmt.executeUpdate(sqlStr);
        out.println("<h3>Your order for book isbn=" + isbnns[i]
            + " has been confirmed.</h3>");
    }
    out.println("<h3>Thank your.<h3>");
} else {
    out.println("<h3>Please go back and select a book...</h3>");
}
```

Check the database to confirm the updating.

### More Exercises:

- Modify the EshopQueryServlet to handle multi-value query parameter, i.e., user checks more than one boxes.
- Add two text fields: orderBy and shippingAddr in EshopQueryServlet. The <input> tag for text field is:

```
<p>Enter your name: <input type="text" name="orderBy"></p>
<p>Enter your address: <input type="text" name=" shippingAddr ">
</p>
```

The "VALUE" for a text field (of the NAME=VALUE pair) is the string entered into the text field.

## 11. Touching Up

You may wish to touch up your e-shop:

- In EShopQueryServlet, instead of writing a paragraph <p> for each row, put the rows into table, as shown:



The syntax for displaying HTML table is:

```
<table>
  <tr>
    <th>heading for column 1</th>
    <th>heading for column 2</th>
    ...
  </tr>
  <tr>
    <td>data for column 1</td>
    <td>data for column 2</td>
    ...
  </tr>
  ...
</table>
```

`<table></table>` markups a *table*. `<tr></tr>` markups a *row*. `<th></th>` is a *header cell* in the row and `<td></td>` is a *data cell*.

- (b) In Figure 4(a), the names of authors are hard-coded in `EshopQuery.html`. Instead of hard-coding, write a servlet (called `EshopAuthorServlet`) to generate the list of authors.

Hint:

Use the following SQL statement to retrieve all the *distinct* authors from table `books`:

```
SELECT DISTINCT author FROM books WHERE qty > 0
OR
SELECT author FROM books WHERE qty > 0 GROUP BY author
```

## 12. What's Next

You have created an e-shop, consists of Tomcat HTTP server and a database, with Java servlet as the server-side programs and HTML form (and Java applet) as the client-side programs. This e-shop, although primitive, is functional.

Some critical components are still missing in our e-shop:

- (a) Session management (or "Shopping Cart"): HTTP is a *stateless* protocol. It does not maintain state information across requests. That is, the 2<sup>nd</sup> HTTP request does not know what was done in the 1<sup>st</sup> request. This poses a problem for our e-shop. If a user chooses a few books over multiple requests, the books chosen have to be "remembered" and placed inside a "shopping cart". The user will check-out the items in the "shopping cart" after he finishes all his shopping.
- (b) A secured payment gateway.

These topics are beyond the scope of this workshop. Read “E-shop Case Study, continue...”.

### 13. References

- [1] JDK 1.6 (JDK 6.0, Java SE 6.0) @ <http://java.sun.com/javase>, Sun Microsystems.
- [2] Apache Tomcat @ <http://tomcat.apache.org>, Apache Software Foundation.
- [3] Java Database Connectivity (JDBC) @ <http://java.sun.com/products/jdbc>, Sun Microsystems.
- [4] Java Servlet Technology @ <http://java.sun.com/products/servlet>, Sun Microsystems.
- [5] The Java EE 5 Tutorial @ <http://java.sun.com/javaee/5/docs/tutorial/doc/>.
- [6] White Fisher, et al., “*JDBC API Tutorial and Reference*”, 3rd eds, Addison Wesley.
- [7] SQL.org @ <http://www.sql.org>.
- [8] Marty Hall, et al., “*Core Servlets and JavaServer Pages*”, vol.1 (2nd eds, 2003) and vol. 2 (2nd eds, 2006), Prentice Hall.
- [9] RFC2616 “*Hypertext Transfer Protocol HTTP 1.1*”, World-Wide-Web Consortium (W3C), June 1999.
- [10] “*HTML 4.01 Specification*”, W3C Recommendation, 24 Dec 1999.
- [11] Eric Ladd and Jim O’Donnell, “Using HTML, Java and CGI” and “Using HTML 4, XML and Java 1.2”, Que.

Case study for the Workshop on Java Programming, version 2009a (Last Modified: September 10, 2009)  
Feedback, comments, and corrections can be sent to Chua Hock Chuan at [ehchua@ntu.edu.sg](mailto:ehchua@ntu.edu.sg)