

# Exercises: Getting Started

Do items 1-4 in order, then try whichever of the remaining exercises grab your interest and you have time for. See C:\Servlets+JSP\Exercise-Solutions\lecture1 for solutions.

1. Open the C:\Servlets+JSP\ directory and double-click the icon to start Tomcat. Verify that the server is running by opening `http://localhost/` in your browser.
2. Open `http://archive.coreservlets.com` in your browser (or use your local copy if you have one), then right-click on `HelloWorld.java` (Chapter 2) to download it to the C:\Servlets+JSP directory. Compile it using “`javac HelloWorld.java`” (DOS Window) or with “`javac “%f”`” (from the UltraEdit Advanced/DOS Command menu), and copy the `.class` file to `tomcat_install_dir\webapps\ROOT\WEB-INF\classes` (there should be a shortcut in C:\Servlets+JSP to make this task easier). Then, check that the servlet is working by using the URL `http://localhost/servlet/HelloWorld`.
3. Download `HelloWWW.java` into the C:\Servlets+JSP directory, edit it so that it outputs “`Hello YourName`”, compile it, and copy the `.class` file to the shortcut to `.../WEB-INF/classes`. Execute the servlet with `http://localhost/servlet/HelloWWW`.
4. Create a “`coreservlets`” directory within C:\Servlets+JSP. Open `http://archive.coreservlets.com` in your browser (or use your local copy if you have one), then right-click on `ServletUtilities.java` and `HelloWWW3.java` (Chapter 2) in order to download them to that directory. Compile `HelloWWW3.java` and copy both `HelloWWW3.class` and `ServletUtilities.class` to `tomcat_install_dir\webapps\ROOT\WEB-INF\classes\coreservlets`. The easiest way is to just copy the **entire** `coreservlets` directory into `tomcat_install_dir\webapps\ROOT\WEB-INF\classes`. Do this by using the right mouse to drag the `coreservlets` directory onto the shortcut to the `WEB-INF\classes` directory and selecting “`Copy.`”  
  
Verify that the process works by invoking `http://localhost/servlet/coreservlets>HelloWWW3`.
5. Create a subdirectory called “`exercise1`” within C:\Servlets+JSP, put a simple servlet in it (be sure to have the package name correspond to the directory name), compile it, and copy the `.class` file to the appropriate location within the Tomcat directory. The easiest approach is to copy the entire `exercise1` directory onto the shortcut to the `WEB-INF\classes` directory. If you want to use `ServletUtilities`, either copy `ServletUtilities` to your directory and change “`package coreservlets`” to “`package yourPackage`”, or (better!) add “`import coreservlets.*`” to whichever of your servlets use `ServletUtilities`.
6. Create a servlet that uses a loop to output an HTML table with 25 rows. For instance, each row could contain “`RowX, Col1`”, “`RowX Col2`”, and “`RowX Col3`”, where `X` is the current row number.

# Exercises: Form Data

Try the first exercise and whichever others best fit your interests, background, and available time.

- 1.** Download `ThreeParamsForm.html` from the Core Servlets and JSP archive site and install it in `tomcat_install_dir\webapps\ROOT`. Load it by means of the URL `http://localhost/ThreeParamsForm.html`. Install the `ThreeParams` servlet and verify that you can send data to it.
- 2.** Change the `ThreeParams` servlet to reside in your package/directory. Make appropriate changes to the form that sends data to it. Test it.
- 3.** Change the `ThreeParams` form and servlet to use `POST` instead of `GET`. If you aren't very familiar with HTML, see the `METHOD` attribute on page 391 of *Core Servlets and JavaServer Pages*. What changes do you need to make to your servlet to have it accept `POST` instead of `GET`?
- 4.** Make a "registration" form that collects a first name, last name, and email address. Send the data to a servlet that displays it. Feel free to modify the `ThreeParams` HTML form and servlet to accomplish this. Next, modify the servlet to use a default value (or send an error message) if the user omits any of the three required parameters.
- 5.** Use the `ThreeParams` form to send data to the `ThreeParams` servlet that contains HTML-specific characters. Verify that this can cause malformed results. Modify the servlet to filter the strings before displaying them.

# Exercises: Request Headers

Do whichever exercises fit your background and interests. They are ordered in approximate order of difficulty. As always, feel free to experiment on your own with other related exercises.

- 1.** Install the servlet that shows all request headers (ShowRequestHeaders). Access it from both Netscape and Internet Explorer. Remember that, since the servlet is in the coreservlets package, you have to install it in the coreservlets directory and use the URL `http://hostname/servlet/coreservlets.ShowRequestHeaders`.
- 2.** Make a tiny Web page that contains a hypertext link to the servlet that shows all request headers. What new header shows up that wasn't there previously? Remember that HTML documents (and images, JSP pages, etc.) go in `tomcat-install-dir\webapps\ROOT`.
- 3.** Write a servlet that sends a different result to Internet Explorer users than to Netscape users. If you prefer one browser over the other, feel free to have the servlet insult users of the other browser. Hint: review printout of request headers from slides.
- 4.** Write a servlet that checks the Referer header to know which banner ad to display. Link to it from a couple of different Web pages, and display different images depending on how the user got to the servlet. Collect images by right-clicking on them in your browser and saving them to a banner-ads subdirectory of the server's root directory (not the server's classes directory!). You can use `<IMG SRC="/banner-ads/file.gif" ...>` to refer to the images.

Note: in Java, you insert a double quote into a String by putting a backslash in front.

# Exercises: The HTTP Response

Do whichever exercises fit your background and interests. You are not expected to do nearly all of them.

- 1.** Write a servlet that sends half the users to `http://home.netscape.com` and half to `http://www.microsoft.com`. Choose at random (compare the output of `Math.random()` to 0.5).
- 2.** Write a servlet that sends Netscape users to `http://home.netscape.com` and Internet Explorer users to `http://www.microsoft.com` (or, vice versa :-).
- 3.** Write a servlet that returns a “page not found” error page (404) unless the user supplies a `favoriteLanguage` request (form) parameter with a value of “Java.” Note: some Tomcat versions incorrectly omit the user-supplied error message string.
- 4.** Write a servlet that generates an Excel spreadsheet. Use the appropriate MIME type from the table in the notes. Use plain text rows with entries separated by tabs (`\t` in a Java string). Try it from both Netscape and Internet Explorer. You need to have MS Office installed for this to work. Hint: this one is easy.
- 5.** Write a servlet that instructs the browser to reconnect every five seconds. Display the time (print new `java.util.Date()`) on each connection.
- 6.** Write a servlet that returns a page to Internet Explorer users saying they will be sent to `http://www.microsoft.com` after 10 seconds. Send them there after that timeout. Send Netscape users to `http://home.netscape.com` after the same delay. You’ll have to read about the Refresh header in the book (page 152) to see the option of supplying a specific URL to connect to.
- 7.** If you are familiar with Java I/O, write a servlet that returns an image (reading it from the disk). Since you will be sending binary data, use `getOutputStream` instead of `getWriter` to get the stream that sends the image. Use the appropriate MIME type. This one is probably too hard unless you already know how to read files from disk.

# Exercises: Cookies

These exercises are longer than most. Start with one of the first four, then the last one, then come back to the others if you have time. Number five is easy if you've done number one; number four is easy if you've done number two.

1. Make a servlet that says “Welcome aboard” to first-time visitors and “Welcome back” to repeat visitors. First verify that it works within a browsing session, then quit your browser and restart to verify that the cookie is persistent on disk.
2. Write a servlet that displays the values of the `firstName`, `lastName`, and `emailAddress` request parameters. If a parameter is missing and the client is a first-time visitor, have the servlet list “Unknown” for the missing values. If a parameter is missing and the client is a repeat visitor, have the servlet use previously-entered values for the missing values.
3. Make a servlet that keeps per-client access counts. That is, have it print “This is visit number  $x$  to this page.” Visit it several times each with Netscape and Internet Explorer to verify that it is keeping separate counts for each.
4. Make an HTML form that lets you send a background color and foreground color to a servlet that displays a Web page in those colors. If you go straight to the servlet without going through the HTML form, have the servlet use cookie values to provide reasonable default color choices (i.e., your previous choice if you made one). If you aren't too familiar with HTML, you set the colors as follows:  
`<BODY BGCOLOR="colorName" TEXT="colorName">`  
or  
`<BODY BGCOLOR="#RRGGBB" TEXT="#RRGGBB">`  
(where R, G, and B are hex values for the red, green and blue components. I.e., #FF00FF is magenta -- 255 for red, 0 for green, and 255 for blue).
5. Repeat exercise number one, but with session cookies, not persistent cookies. Visit it twice in Internet Explorer, then click on the IE icon *on the desktop* to start a new browser. Is this considered the same session or a new one? Repeat the process with Netscape. Notice that they work differently.
6. Check out your cookie files in IE and Netscape. On IE, start at the Tools menu, then do Internet Options, General, Settings, View Files. It is easier to find the cookie files if you do “Delete Files” first. On Netscape, do a search (“Find”) for a file called `cookies.txt` in a Netscape folder. See if you notice a cookie from `doubleclick.net` in both cases. Take a look at the structure of these files to see if you can figure out how they represent the various cookie parameters.

# Exercises: Session Tracking

1. Make a servlet that says “Welcome Aboard” to first-time visitors (within a browsing session) and “Welcome Back” to repeat visitors. Was this servlet harder or easier than the equivalent version using cookies explicitly?
2. Write a servlet that displays the values of the `firstName`, `lastName`, and `emailAddress` request parameters. Have the servlet list “Unknown” for missing values of first-time visitors and the previously entered values for repeat visitors. This should definitely be easier than the version that used cookies explicitly.
3. Make an HTML form containing a textfield and a submit button, where the textfield is intended to gather the name of an item to be purchased. Every time you submit an item, have the servlet show a table or list containing that item plus all previously “purchased” items. If you feel truly inspired, you can check to see if the current item matches a previous item, and display a number indicating how many of that item are being purchased. Don’t forget to put the HTML form in `tomcat_install_dir\webapps\ROOT` and to access it through a URL of the form `http://localhost/YourForm.html`. Do not read the HTML form directly from disk—if you do, the browser won’t know how to interpret the `ACTION="/servlet/..."` part of the form. If you are unfamiliar with the list-related data structures in Java, see note at bottom regarding the `Vector` class.
4. Make a servlet that prints a list of the URLs of all pages that the current user has used to link to it. That is, for each user, have it keep track of all unique values of the `Referer` request header. Test it out by making a couple of different static Web pages, each of which link to it via `<A HREF="/servlet/...">click to visit servlet</A>`.

Put your pages in `tomcat_install_dir\webapps\ROOT` and access them through a URL of the form `http://localhost/YourPage.html`.

## Note: Using the Vector Class

The `java.util.Vector` class (or `java.util.ArrayList` if you know you are using JDK 1.2 or later) is useful for keeping lists of items when you don’t know how many items you will have. It has several key methods:

- Constructor: empty. Eg `Vector v = new Vector();`
- Adding items to end of vector: call “add”. Eg `v.add("Item One");`
- Number of items in in vector: call “size”. Eg `int numItems = v.size();`
- Get an item out: `elementAt`. Eg `String item = (String)v.elementAt(0);`

# Exercises: JSP Intro

- 1.** Download Expressions.jsp and install it on your server in `tomcat_install_dir\webapps\ROOT`. Also download the JSP-Styles.css file that Expressions.jsp uses. Access Expressions.jsp through the appropriate URL. Compare the time it takes to access it the first time to the time it takes on subsequent requests. Try supplying a value for the testParam form parameter.
- 2.** Question to ponder: if you wanted to use style sheets from regular servlets, where would you put the style sheet and how would you refer to it from the servlet? How about images? Notice how much simpler the process is with JSP.
- 3.** Make a very simple static HTML page. Download one from the Web if you want. Rename it to have a .jsp extension. Have an HTML form with nothing but a SUBMIT button that directs the user to the static page. Now, change the form to use POST instead of GET (see Listing 16.2 on page 388 and the METHOD section on page 391 if you don't remember how to tell the form to use POST). Why does it work both times? (Answer: the auto-generated code can't just be in the doGet method. Where *does* the auto-generated code go, then?)

# Exercises:

## JSP Scripting Elements

1. Make a JSP page that shows a random number between 0 and 1.
2. Make a JSP page that displays the values of the param1, param2, and param3 form parameters. That is, change the ThreeParams servlet into a JSP page that has the same result. Modify the ThreeParams HTML form to send data to that JSP page. You'll probably find that the JSP version is much cleaner and simpler than the equivalent servlet.
3. Make a JSP page that makes a bulleted list with a random number of entries in the list, each of which is a random int. To generate a random int, use `Math.random`, multiply, cast the result to an int, and add 1. You'll probably find that this JSP page is not nearly as clean and simple as that of #1. Why?
4. Define a method called `randomInt` that takes a number as an argument and returns a random int from 1 to that number (use `Math.random`, multiply, cast the result to an int, and add 1). Redo problem #3 using that method.
5. Make a JSP page that displays a random number the first time anyone visits it, and displays the *same* number on all subsequent visits (hint: you don't need cookies or session tracking).
6. Make a JSP page that always displays the same page content, but uses a background color of green, red, blue, or yellow, randomly chosen for each request. Make sure your page does *not* use the JSP-Styles style sheet, since that style sheet overrides the background color. (Or, if you feel truly inspired, you could have the style sheet be generated by a JSP page and set the background color that way.)
7. Visit the servlet you made earlier that sets some session data, then visit a JSP page that displays the data. Use the predefined "session" variable.
8. Make a JSP page that displays the value of a cookie with a designated name. (Hint: call `ServletUtilities` by doing "`coreservlets.ServletUtilities.someMethod(...)`"). First visit one of your servlets that sets a cookie, then visit the JSP page to display its value. The servlet will need to set the path so that the JSP page will get the cookie, since the JSP page is in a different directory hierarchy. That is, the servlet that sets the cookie should do:

```
Cookie c = new Cookie(...); // or coreservlets.LongLivedCookie
c.setPath("/");
response.addCookie(c);
```

# Exercises: The page Directive

1. Make a JSP page that displays the value of a cookie with a designated name. First visit one of your servlets that sets a cookie, then visit the JSP page to display its value. Use the ServletUtilities class. Remember, the servlet will need to set the path so that the JSP page will get the cookie, since the JSP page is in a different directory hierarchy. That is, the servlet that sets the cookie should do:

```
Cookie c = new Cookie(...); // or coreservlets.LongLivedCookie
c.setPath("/");
response.addCookie(c);
```
2. The java.math package has a class called BigInteger that lets you create whole numbers with an arbitrary number of digits. Create a JSP page that makes a large BigInteger from a String you supply as a request parameter, squares it, and prints out the result.
3. Make an Excel spreadsheet where each entry is a random number. Use Internet Explorer to access it if you don't have the MS Office plugin in Netscape.
4. Make an Excel spreadsheet with a random number of rows.
5. Make a JSP page that sleeps for 20 seconds before returning the page. (Call Thread.sleep from inside a try/catch block that catches InterruptedException). Access it "simultaneously" from Netscape and Internet Explorer. Repeat the experiment with the JSP page not allowing concurrent access. Verify the slower result.
6. Download the ComputeSpeed and SpeedError pages from the archive site. Access the ComputeSpeed page with numeric values for the "furlongs" and "fortnights" form parameters attached to the end of the URL. (See section 11.10 starting on page 261 if you want more detail). Now, supply a non-numeric value for one of the parameters. Next, supply 0 for the fortnights parameter. Can you explain the unexpected result you get?

# Exercises: Including Files and Applets

1. Make an HTML “signature” block with your name and email address. Include it in two JSP pages.
2. Suppose that you have two different JSP pages that do two different things. However, for both pages you want to let the user supply a `bgColor` attribute to set the background color of the page. Implement this, but use an include mechanism to avoid repeating code. Hint: when you use `jsp:include`, the included page has the same request object as the main page. For example:  
White background: `http://host/path/page1.jsp`  
White background: `http://host/path/page2.jsp`  
Red background: `http://host/path/page1.jsp?bgColor=RED`  
Yellow background: `http://host/path/page2.jsp?bgColor=YELLOW`
3. Make two separate JSP pages that have bulleted lists containing random integers in a certain range. Avoid repeating code unnecessarily by including a page that defines the `randomInt` method from the exercise on JSP scripting expressions.
4. The value of the page attribute of `jsp:include` is allowed to be a JSP expression. Make a JSP page that includes a “good news” page or a “bad news” page at random.
5. If you are familiar with applets, make a trivial one that does nothing but set the background color to blue and print a string derived from the `MESSAGE` parameter embedded in the HTML by means of a `PARAM` element. Convert it to a version that uses the Java Plug-In. Note that, if this runs at all, it proves that you are correctly accessing the Plug-In. You don’t need to use Swing or Java2D to verify that you are using the Plug-In, since the tag generated by `jsp:plugin` is incompatible with the standard virtual machine used by Netscape and IE. Try both Netscape and Internet Explorer to see which (if any) of them has the Plug-In installed. Reminder: applets run on the client, not on the server. So your applet’s `.class` files can’t go in the server’s `WEB-INF/classes` directory. These `.class` files work the same way as for regular applets: they go in the same directory as the JSP/HTML file that uses the applet tag. The one exception is if the applets are in a package, in which case they go in a subdirectory (matching the package name) of the directory that contains the JSP file. Again, this is nothing specific to JSP, but is just the normal way applets work.

# Exercises: Beans

- 1.** Make a simple bean called `BakedBean` with two `String` properties: `level` (default value “half-baked”) and `goesWith` (default value “hot dogs”). Compile and test it separately (i.e., without using a servlet or JSP page). Note: if your driver class (i.e., the one that has “`public static void main(String[] args) {...}`” in it) is in a package, remember that you have to use the package name when you run it from the command line. That is, you have to do “`javac BeanDriver.java`” and then “`java yourPackage.BeanDriver`”.
- 2.** Make a JSP page that makes a `BakedBean`, sets the `goesWith` property to “caviar”, and then prints out the `level` and `goesWith` properties.
- 3.** Make a JSP page that makes a `BakedBean`, sets the `goesWith` property based upon the value of some request parameter, and stores the bean in the `ServletContext`. Make a second JSP page that displays the value. Visit the first page from Netscape only (with a request parameter that sets `goesWith`), then visit the second page from both Netscape and Internet Explorer. Verify that both browsers see the same value.
- 4.** Make a JSP page that makes a `BakedBean`, sets the `goesWith` property based upon the value of some request parameter, and stores the bean in the `HttpSession` object. Make a second JSP page that displays the value. Visit the first page from Netscape only (with a request parameter that sets `goesWith`), then visit the second page from both Netscape and Internet Explorer. Verify that the two browsers see different values.

# Exercises: Custom Tags

1. Download all the files needed to get the SimplePrimeTag and PrimeTag examples working, and test them. There are quite a number of files needed.
2. Make a random number tag that inserts a random number between 0 and 1 into the page.
3. Make a random number tag that inserts a random integer between 0 and some specified number.
4. Make a tag that results in whatever text it encloses being displayed in a large, bold, red, blinking format. For example,  
`<yourTags:annoying>This is a test</yourTags:annoying>`  
should result in  
`<H1><FONT COLOR="RED"><BLINK>This is a test</BLINK></FONT></H1>`

Test on Netscape; Internet Explorer wisely ignores the BLINK tag.

# Exercises: Servlet/JSP Integration and the MVC Architecture

For each of the following exercises, you might want to start by creating a bean, then creating a servlet that populates it (and invokes the RequestDispatcher), then creating a JSP page that presents it.

- 1.** Write a servlet that generates a random number, puts it in a bean, and forwards it to a JSP page that displays it. Use request-based bean sharing.

Hint: you might want to start by simply forwarding from an empty servlet to a simple JSP page that doesn't use beans. Then, once you have the RequestDispatcher part working, you can go back and have the servlet store the bean object (request.setAttribute) and have the JSP page read it (jsp:useBean with scope="request").

- 2.** Write a servlet that reads the firstName and lastName request parameters. If they are both present, it forwards the user to a page that displays them. If either is missing or is an empty string, it uses previously seen values from that client. If values are still missing after that, it forwards the user to a page that tells them which parameter is missing. Use session-based bean sharing.
- 3.** Write a servlet that uses a request parameter to determine the desired size of a prime number. If the user asks for a particular size, it should create and store one of that size. If no size is specified, the system should use the previous prime number (if there is one). Either way, it should forward the user to a page that displays the number. Use application (servlet context) based bean sharing.

# Exercises: JDBC

- 1.** Make a servlet that displays a bulleted list of the names of all shipping companies used in the Northwind example. (Table name: shippers; column name: Company-Name — see the second of the screen shots from “Using Metadata”).
- 2.** Make an HTML form that collects a last name. Send the name to a servlet or JSP page. If there is an employee with that last name, show full details on him or her (just show the first employee if there are multiple people with the same name). If there is no employee with that last name, say so. See the screen shot from Access for details on the column names (table name: employees; column names firstname, lastname, title, birthdate).
- 3.** Make an HTML form that collects three values: a table name, a number of columns  $c$ , and a number of rows  $r$ . Send the data to a servlet or a JSP page. Display an HTML table containing the first  $c$  columns of the first  $r$  rows of the designated table. Be sure to handle the case where  $c$  or  $r$  are too big.

# Student Survey

**1.** Please rate the following from 1 (poor) to 5 (great):

- Instructor: \_\_\_\_\_
- Topics: \_\_\_\_\_
- Handouts: \_\_\_\_\_
- Exercises: \_\_\_\_\_
- Book: \_\_\_\_\_
- Facilities: \_\_\_\_\_
- Overall: \_\_\_\_\_

**2.** What is your overall opinion of the course?

**3.** What were the strong points of the course?

**4.** What should be changed to improve it?