

Exercises: Getting Started

Since these exercises are of the “test your setup and deployment process” variety, you probably want to do all of them, in order. See `C:\Servlets+JSP\Exercise-Solutions\` 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://volume1.coreservlets.com/archive/` in your browser (you probably want to bookmark the location), then right-click on `Hello.html` and `Hello.jsp` (Chapter 2) to download them to the `C:\Servlets+JSP` directory. It might be easier to use Firefox to download, because Internet Explorer sometimes adds `.txt` to the end of files whose file extensions it does not recognize (e.g., `blah.jsp` and `Blah.java`). Copy the files onto the shortcut to `webapps/ROOT`, then access them in the browser with `http://localhost/Hello.html` and `http://localhost/Hello.jsp`.
- 3.** Download `HelloServlet.java` into the `C:\Servlets+JSP` directory. Compile it using “`javac HelloServlet.java`” (DOS Window) or via `Tools --> Compile Java` from TextPad. Then, and copy the `.class` file to the shortcut to `webapps/ROOT/WEB-INF/classes`. Finally, check that the servlet is working by using the URL `http://localhost/servlet/HelloServlet`.
- 4.** Download `HelloServlet2.java` into the `C:\Servlets+JSP\coreservlets` directory. Compile it, then deploy by copying (not moving!) the *entire* `coreservlets` directory onto the shortcut to `webapps/ROOT/WEB-INF/classes`. One of the easiest ways to copy it is by using the right mouse to drag the `coreservlets` directory onto the shortcut and selecting “Copy.” Access it with `http://localhost/servlet/coreservlets.HelloServlet2`
- 5.** Download `HelloServlet3.java` and `ServletUtilities.java` into the `coreservlets` directory. Compile `HelloServlet3.java`, deploy it, and access it in your browser.
- 6.** The only thing we know (yet!) related to the SCWCD exam is a little bit about the structure of Web apps. Look inside the `ROOT` folder and take note where HTML goes, JSP goes, class files go, and the `web.xml` file is located.

Exercises: Servlet Basics

1. Modify HelloServlet so that it says “Hello *Your-Name*”. Compile and run it.
2. Create a servlet that makes a bulleted list of four random numbers.
 - Reminder 1: you use `Math.random()` to output a random number in Java.
 - Reminder 2: you make a bulleted list in HTML as follows

```
<UL>
  <LI>List item 1
  <LI>List item 2
  ...
</UL>
```
3. Create a servlet that uses a loop to output an HTML table with 25 rows and 3 columns. For instance, each row could contain “RowX, Col1”, “RowX Col2”, and “RowX Col3”, where X is the current row number.
4. Create a subdirectory called “servletBasics” 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* servletBasics directory onto the shortcut to the `WEB-INF\classes` directory. For the rest of the week, I suggest you make a separate package/directory for each set of exercises.

You do not need to use my ServletUtilities class, but if you want to, either copy ServletUtilities to the servletBasics directory and change “package coreservlets” to “package *yourPackage*”, or (better!) leave it in the coreservlets directory and add “import coreservlets.*” to whichever of your servlets uses ServletUtilities.
5. If you have some previous servlet/JSP experience and want to try something we haven’t yet covered in detail in class, try making a custom Web application. See the short summary in the notes (but we will cover it in detail later). You don’t need to start from scratch; you can start by copying and renaming the ROOT directory.
6. For the SCWCD exam, make sure you know the method signature of `doGet` (which is the same as for `doPost`, `doHead`, and `service`). Make sure you know that `init` gets called only when the servlet is first loaded into the server’s memory, and that there is only one servlet instance per URL: servlet’s do *not* get instantiated for each request.

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 from the form.
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. Make a variation of the `ThreeParams` form and servlet that uses `POST` instead of `GET`.
4. Use the `ThreeParams` form to send data to the `ThreeParams` servlet that contains HTML-specific characters. Verify that this can cause malformed results. Make a variation of the servlet that filters the strings before displaying them.
5. 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 give an error message) if the user omits any of the three required parameters.
6. **[Hard; for the truly inspired.]** Make a variation of the previous servlet and accompanying form, but, in this case, if the user fails to supply a value for any of the fields in the form, you should redisplay the form but with an error message saying that the value is missing. Don’t make the user reenter values that they’ve entered already.

Hint: you have two main choices given the tools available to us so far: have one big servlet that generates both the HTML form and the results page (the form submits the data to itself), or two separate servlets (one that generates the HTML form and the other that generates the results page). There is an example of the first approach in the book. If you want to try the second approach, you might want to know about `response.sendRedirect`, which lets you redirect the browser to a specified page. For example, here is a `doGet` method that sends the user to `www.whitehouse.gov`:

```
public void doGet(HttpServletRequest request,
                 HttpServletResponse response)
    throws ServletException, IOException {
    response.sendRedirect("http://www.whitehouse.gov/");
}
```

7. The SCWCD objectives are pretty vague about form processing. They just say you should know how to “write code to retrieve HTML form parameters from the request”. The most important thing in real life is to know `getParameter` and the fact that empty fields result in empty strings and missing params result in null. But for the exam, I would know `getParameterNames`, `getParameterMap`, `getParameterValues`, `getReader`, and `getInputStream`. You rarely use any of those, but still...
<http://www.coreservlets.com>

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 Firefox 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://host-name/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 just says "Hello." Use a red background and a yellow foreground for Internet Explorer users; use a yellow background and a red foreground for Firefox and other non-IE users. If you are a bit rusty with HTML, you set 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).
4. Some informational sites want users to access pages in a certain order. They want to prohibit users from jumping directly to a bookmarked page later in the sequence because the data on the pages may have changed since the user bookmarked the page.

Create two pages. The first page should have a link to the second page. If a user accesses the first page and then follows the link to the second page, it works normally. But, if the user directly types in the address of the second page (or follows a link from a page with a different name than the first page), they should get sent back to the first page automatically.

Hints:

- Use `response.sendRedirect` (as on previous exercise) to send users back to page1
- It is not necessary to make page1 be a servlet.

5. All the SCWCD exam objectives say is that you have to know how to "retrieve HTTP request header information." In real life, `getHeader` and the need to check for null are the important things. But I would also memorize the names of the methods on page 7 of the slides.

Exercises: HTTP Status Codes

1. Write a servlet that sends about half the users to <http://www.mozilla.org> and about half to <http://www.microsoft.com>. Choose at random (compare the output of `Math.random()` to 0.5).
2. 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.”
3. Write a servlet that sends the first 9 requests to [washingtonpost.com](http://www.washingtonpost.com) (an organization that uses servlets and JSP extensively, by the way), the next request to [nytimes.com](http://www.nytimes.com), and then repeats. That is, every tenth request should get sent to [nytimes.com](http://www.nytimes.com) and the rest should get sent to [washingtonpost.com](http://www.washingtonpost.com). Don’t just use probabilities (i.e., don’t compare `Math.random` to 0.9); actually count the overall requests. So, for example, if the server received between 1000 and 1009 overall requests since last being booted, exactly 100 of them should have sent users to [nytimes.com](http://www.nytimes.com), and the rest should have sent users to [washingtonpost.com](http://www.washingtonpost.com).

Hint: use an instance variable (aka field, data member).

4. Amazingly, the SCWCD exam objectives do not explicitly mention status codes *anywhere*. Presumably they are lumped under the HTTP response. At the very least, you should know 200, 301, 302, 404, 403 (covered later), `setStatus`, `sendRedirect`, and `sendError`. Another section mentions the need to understand multithreading problems and race conditions, so make sure you know the issues with instance variables.

Exercises: HTTP Response Headers

- 1.** Redo the first problem from the previous set of exercises. Instead of using `response.sendRedirect`, set the appropriate status code and response header explicitly. Do you get the same result?
- 2.** Write a servlet that instructs the browser to reconnect every five seconds. Display the time (print `new java.util.Date()`) on each connection.
- 3.** Write a servlet that generates an Excel spreadsheet with 10 rows and 5 columns. Display a random number in each cell. Try it from both Firefox and Internet Explorer. What happens when you hit the “Reload” button?
- 4.** 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 Firefox users to `http://www.mozilla.org` after the same delay. You’ll have to read about the Refresh header in the book (page 203) to see the option of supplying a specific URL to connect to.
- 5.** The SCWCD exam objectives say “Using the `HttpServletResponse` interface, write code to set an HTTP response header [`setHeader`], set the content type of the response [`setContentType` or `setHeader` with first argument “Content-Type”], acquire a text stream for the response [`getWriter`], acquire a binary stream for the response [`getOutputStream`], redirect an HTTP request to another URL [`sendRedirect`], or add cookies to the response [`addCookie` - see next lecture].” My guess is that you also need to know the most important HTTP headers and the methods on pages 6 and 7 of the slides

Exercises: Cookies

1. In a previous problem, you had two pages. You used the Referer header to force users to get to page 2 by following a link from page 1. Now, suppose that it is not necessary to *always* go to page 1 before page 2; you just want to make sure that users have been to page 1 *at least once* before they are allowed to visit page 2. For example, page 1 might be a page that introduces the site and gives some legal disclaimers, and users are required to visit that page at least once before they can see the second page.

Create two pages. If a user visits page 2 before having ever visited page 1, they should get redirected to page 1. **Note:** some people find this problem easier if they manually delete cookies between tests. To do this in IE 5/6, start at the Tools menu, then select Internet Options, General, and Delete Cookies. With Firefox, click on Tools, then Options, then Privacy, then Cookies—you'll have various options from there.

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 small page that displays some simple information of your choosing. Make another page that lets users choose the foreground and background color that they want the first page to use. For example, if users never visit the the color choice page, the main informational page should use default colors of your choosing. But if a user visits the color choice page and chooses a yellow background and red foreground, all subsequent visits to the main page by that user should result in red on yellow. There is no need to vet the colors; you can accept whatever color values the user gives you.

If you are a bit rusty with HTML, you set 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).

Cookie Exercises, Continued

4. Repeat exercise number one, but with session cookies, not persistent cookies. Visit it twice in IE, 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 Firefox. Notice that they work differently.
5. Check out your cookie files in IE and Firefox. 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. With Firefox, do a search (“Find”) for a file called cookies.txt in a Mozilla 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.
6. The SCWCD exam objectives just vaguely state that you need to know how to “retrieve a cookie from the request [getCookies]” and “add a cookie to the response [addCookie]”. I would also know how cookies work abstractly (both session-- in-memory cookies and persistent -- on disk cookies), plus know the following Cookie methods: Cookie constructor, setMaxAge, getName, getValue, setDomain, setPath, and setSecure. In real life, the absolutely essential ones are getCookies, addCookie, constructor, setMaxAge, and getName.

Exercises: Session Tracking

- 1.** Use session tracking to redo the servlet (from the Cookies lecture) 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 a servlet that prints a list of the URLs of the pages that the current user has used to link to it (within the current browsing session). That is, if one user has followed hypertext links from three different pages to the servlet, the servlet should show the user the URLs of those three pages. Test out your servlet by making a couple of different static Web pages that link to it.

If you feel inspired, modify the basic approach from the notes so that you do not store repeated entries; if the same user follows a link from page1 to the servlet twice, the servlet should list the URL of page1 only once in the list. If you are unfamiliar with the list-related data structures in Java, see note at bottom regarding the `ArrayList` class.

Session Tracking, Continued

4. [Hard, for the truly inspired.] The previous problem tracked the referring page and ignored repeated entries. In this problem, you should track a request parameter and count the repeats. Make a servlet that keeps track of the number of each item being ordered. For example, if the user orders yacht, car, book, yacht, the servlet should show something like this:

- yacht (2)
- car (1)
- book (1)

To simplify your code, you can just use the item name as sent from the HTML form; there is no need to check each name against a table of legal names as in the much-more-complicated Shopping Cart example in the book.

Note: Using the ArrayList Class

The `java.util.ArrayList` class 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. E.g. `ArrayList list = new ArrayList();`
- Adding items to end of list: call "add". E.g. `list.add("Item One");`
- Number of items in list: call "size". E.g. `int numItems = list.size();`
- Get an item out: "get". E.g. `String item = (String)list.get(0);`
- Testing if an item is already in the list: "contains". E.g.
`if (list.contains("Item One")) ...`

5. The SCWCD exam objectives have an entire section on session tracking. Know how sessions get created and destroyed, know the methods on pages 14 and 15 of the slides, and know what URL rewriting entails. The objectives also state you have to know that session values that implement `HttpSessionActivationListener` are notified when session values move from one VM to another in a cluster.

Exercises: JSP Intro

- 1.** Download OrderConfirmation.jsp and install it on your computer. Also download the JSP-Styles.css file that OrderConfirmation.jsp uses. Compare the time it takes to access it the first time to the time it takes on subsequent requests.
- 2.** If you wanted a servlet to display an image, where would you put the image and how would you refer to it from the servlet? How does this differ from JSP? (This question is trickier than it sounds at first.)
- 3.** Make a very simple static HTML page. If you want, download one from the Web or use one from a previous exercise. 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. 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?)
- 4.** For the SCWCD exam, make sure you understand the JSP lifecycle as shown in the table on the slides. (Lots more on JSP for the SCWCD exam later!)

Exercises:

JSP Scripting Elements

1. Make a JSP page that randomly selects a background color for each request. Just choose at random among a small set of predefined colors. Be sure you do not use the JSP-Styles.css style sheet, since it overrides the colors given by `<BODY BGCOLOR="...">`.
2. Make a JSP page that lets the user supply a request parameter indicating the background color. If no parameter is supplied, a background color should be selected at random.
3. Make a JSP page that lets the user supply a request parameter indicating the background color. If no parameter is supplied, the most recently used background color (from a previous request by *any* user) should be used.
4. The SCWCD exam objectives say you have to know all three JSP scripting elements (expressions, scriptlets, and declarations). It also says you need to know JSP comment syntax and the predefined variables (“implicit objects”). Some of the sample exams also imply that given the choice, they think that the dangling brace use of scriptlets is preferable to scriptlets containing if statements or loops with explicit print statements in them.

Exercises: The page Directive

1. Use JSP to make an Excel spreadsheet where the first row says “Year,” “Apples,” and “Oranges.” It should then have two rows of data (2005 and 2006), where each entry is a random number between 0 and 10. I.e. the result should look something like this:

Year	Apples	Oranges
2005	9.23456	3.98765
2006	4.45678	2.223344

2. Make an Excel spreadsheet with a random number of rows.
3. 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. Use the online API at <http://java.sun.com/j2se/1.5.0/docs/api/> to see the syntax for the `BigInteger` constructor and squaring operations.
4. 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 Firefox and Internet Explorer. Repeat the experiment with the JSP page not allowing concurrent access. Verify the slower result.

The page Directive, Continued

5. [**Hard.**] The above example worked because our version of Tomcat implements `isThreadSafe="false"` by queueing up the requests and passing them one at a time to the servlet instance. But, that is not the only legal implementation. Create a test case that will definitively show which of the following three approaches a server uses for `isThreadSafe="false"`:

- Keeps a single servlet instance and queues up the requests to it
- Makes a pool of instances but lets each instance only handle one request at a time
- Ignores `isThreadSafe` altogether

Note that *all three* approaches have been represented by Tomcat versions in the past.

6. [**Just for fun.**] 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 page 366 if you want more detail). Now, supply a non-numeric value for one of the parameters. Next, supply a legal number for furlongs, but 0 for fortnights. Can you explain the unexpected result you get?

7. The SCWCD exam objectives say that you have to know the following page directive attributes: `import`, `session`, `contentType`, and `isELIgnored` (we will cover this one later). It does not mention `buffer`, `errorPage`, `isErrorPage`, `extends`, or `isThreadSafe`. Go figure!

Exercises: Including Files and Applets

1. Make an HTML “signature” block with your name and email address. Include it in two JSP pages.
2. The value of the page attribute of `jsp:include` is allowed to be a JSP expression. Use this idea to make a JSP page that includes a “good news” page or a “bad news” message at random.
3. 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. 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`

For testing, I do not care if you write an HTML form to collect the `bgColor` parameter or if you simply attach it onto the end of the URL “by hand.”

4. 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 a `randomInt` method.

Inclusion, Continued

- 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 Firefox and IE. Try both Firefox 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.
- 6.** The SCWCD exam objectives state "Given a specific design goal for including a JSP segment in another page, write the JSP code that uses the most appropriate inclusion mechanism (the include directive or the jsp:include standard action)." Gulp! I hope the exam authors understand the serious maintenance problems with the include directive, and that the scenarios where it is "appropriate" on the exam are only the scenarios where jsp:include wouldn't work at all (as with the included page defining a variable that the main page accesses). Make sure you understand the table on page 11 of the slides.

Exercises: Beans

For these exercises, avoid *any* Java code in the JSP pages.

- 1.** Define a class called `ColorBean` that stores strings representing a foreground color and a background color. Compile and test it separately (i.e., without using a servlet or JSP page). Note: if your tester 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 BeanTester.java` and then `java yourPackage.BeanTester`.
- 2.** Make a “color preference” form that collects the user’s preferred foreground and background colors. Send the data to a JSP page that displays some message using those colors. This JSP page should use a default value for any form value that the user fails to supply (but don’t worry about empty strings). So, for example, if the user goes directly to the JSP page (bypassing the form), the JSP page should still work fine. For now, don’t worry about the user sending you whitespace; just handle totally missing values.
- 3.** Redo the color preference example, but if the user fails to supply either of the colors, use whatever value they gave last time. If you have no previous value, use a default. (Hint: this problem is almost exactly the same difficulty as the previous one.)
- 4.** Redo the color preference example, but if the user fails to supply any of the parameters, use whatever color the most recent user gave last time. Why could this give bad results?
- 5.** Fix problem #2 so that it also handles whitespace. Do so without adding any explicit Java code to the JSP page.
- 6.** The SCWCD exam objectives state you have to know `jsp:useBean` (and the `id`, `scope`, `class`, and `type` attributes -- we will cover “`type`” in the next section), `jsp:setProperty` (presumably with all the options discussed in the notes, although they don’t say), `jsp:getProperty` (again, presumably with all attributes, although they don’t say), and `jsp:param`. Another section mentions the need to understand multithreading problems and race conditions, so make sure you know the issues with application-scoped data.

Exercises: the MVC Architecture

- 1.** Redo problem #2 from the previous exercises, but use MVC this time. Also, don't let the foreground color be the same as the background color.
- 2.** Redo problem #3 from the previous exercises. Don't let the foreground color be the same as the background color.
- 3.** Redo problem #4 from the previous exercises. Don't let the foreground color be the same as the background color.
- 4.** **[Question to ponder; no need to write code.]** Suppose you wanted to prohibit RED and GREEN as colors, to make the pages more accessible to color-blind users. How could you do this? Why would it work better with the MVC approach than with the do-everything-in-JSP approach?
- 5.** **[Question to ponder; no need to write code.]** Suppose you wanted to change all three problems so that a designated user could change the default colors (i.e., the colors that result if no user-selected color is found). How could you do this? Why would it work better with the MVC approach than with the do-everything-in-JSP approach?
- 6.** For the SCWCD exam, you definitely have to know how RequestDispatcher works, the various scopes, and how to retrieve the results in the JSP page (both ways, but we will cover the EL in the next section). You also have memorize *all* the pattern names from the notes. Try not to fall asleep when doing so. Yawn!

Exercises:

the JSP 2.0 Expression Language

- 1.** Redo problem #1 from the MVC exercises, but use the expression language in the JSP page.
- 2.** Redo problem #2 from the MVC exercises, but use the expression language in the JSP page.
- 3.** Redo problem #3 from the MVC exercises, but use the expression language in the JSP page.
- 4.** The SCWCD exam objectives imply that you have to know virtually everything from the lecture including the options for disabling scripting and disabling the expression language. Sigh. In real life, the important two things are: you need a web.xml declaration mentioning servlets 2.4 or later, and in the MVC results page, you output bean properties with `${beanAttributeName.beanPropertyName}`.

Exercises: Web Applications

1. Create a Web application with a name of your choosing. Use a regular directory structure (not a WAR file) for the Web app. Put the directory in C:\Servlets+JSP, and deploy by copying the entire directory (drag with the right mouse) to the shortcut to *tomcat_dir/webapps*. Put two JSP pages in the Web app and verify that you can access them.

Note that your Web app is required to have a legal *web.xml* file in the WEB-INF directory. If you copy an existing Web app (e.g., Exercise-Solutions/app-blank or ROOT) to make your Web app, you have this already. If you make your directory structure by hand, copy the *web.xml* file from app-blank or ROOT (or from the archive.moreservlets.com Web site) to your Web app.

2. Put a servlet in your Web app and access it. Don't use the invoker servlet (i.e., <http://localhost/webappName/servlet/ServletName>). Instead, use a URL that is explicitly registered in *web.xml* (i.e., <http://localhost/webappName/anything>).
3. Deploy a second Web app, but use a WAR file this time. If you want to use Java to make the WAR file, open a DOS window, go into the top-level of an existing Web app, and say "jar -cvf someName.war *". You can also use WinZip or the Windows XP "create compressed folder" R-mouse option to make a file called *someName.war*. Either way, keep the original copy of *someName.war* in C:\Servlets+JSP. To deploy *someName.war*, copy it to *tomcat_dir/webapps* (there is a shortcut in C:\Servlets+JSP), and use the URL <http://localhost/someName/...>
4. Try to share data between your Web apps by means of the servlet context. Will the default Tomcat setting let you do this?
5. Make a servlet that creates a cookie. Be sure to set the path appropriately. Make another servlet (or a JSP page) that displays the cookie value. Copy the cookie-displaying program to another Web application and verify that you can see cookies that were created in the first Web app.
6. For the SCWCD exam, you have to know the Web app structure thoroughly. It says you must know how to "Construct the file and directory structure of a Web Application that may contain (a) static content [top-level dir or any non-special subdirectory], (b) JSP pages [usually same as (a) but remember JSP pages under WEB-INF for MVC], (c) servlet classes [WEB-INF/classes/subdirMatchingPackage, unless in JAR file], (d) the deployment descriptor [WEB-INF/web.xml], (e) tag libraries [Java classes in normal places; tag files in WEB-INF/tags; TLD files under WEB-INF; covered in later lecture], (d) JAR files [WEB-INF/lib], and (e) Java class files [same as "servlet classes" mentioned in (c)]; and describe how to protect resource files from HTTP access [put under WEB-INF or use security constraint].

Exercises: web.xml

1. Make a blank Web application (the easiest way is to copy and rename the app-blank Web app). Copy the servlet you wrote for the previous exercise, give it a custom address in web.xml, and access it using that address.

For the remaining problems, you are the CTO of Enroff, a company that is so popular that other big companies keep buying you out for higher-and-higher prices. That's the good news. The bad news: the company name keeps changing, making you keep changing your home page.

2. Make a servlet that reads the company name from an initialization parameter and then uses it in the same page.
3. Repeat problem 2 (using an init parameter), but use a JSP page instead of a servlet.
4. Make sure that this JSP page is displayed if the user requests *http://host/yourCompany/*, with no filename specified.
5. Turn off the invoker servlet (if you made your own Web app from scratch), or verify that the invoker servlet is already disabled (if you started with app-blank).
6. Have a servlet read the company name from an init parameter, store it in the servlet context, and use it in the servlet's output. Have two JSP pages that use the same company name, but don't have the JSP pages re-read the init parameter.
7. Be sure nothing bad happens if someone tries to access the above JSP pages before the above servlet that sets the company name.
8. Make a servlet or JSP page that reads a request parameter and prints a message saying if that value is a substring of the current company name. If the request parameter is missing, display a designated page that is not accessible directly (i.e., there is no URL that the client can supply that directly yields the error page). Don't use any try/catch blocks or explicit checks for null.
9. The SCWCD exam objectives simply state you must know "the semantics of the deployment descriptor". Aaargh. It doesn't say which of the zillions of options you have to know. I'd suggest being very familiar with the main ones (servlet, servlet-mapping, welcome-file-list, and the ones that will show up later for security, filters, and listeners), and skim the others.

Exercises: Declarative Security

Note: to get the usernames and passwords from the notes, just grab <http://archive.moreservlets.com/Security-Code/tomcat-users.xml> and put it in *install_dir/conf*.

- 1.** Create a JSP page that can be accessed only by registered users or administrators. Use form-based authentication.
- 2.** Repeat the above, but with BASIC authentication. Hint: just copy your Web application, give it a different name, and make one change to the web.xml file.
- 3.** Back to the first Web app (the one that uses form-based authentication): create a servlet that is also accessible only to registered users and administrators.
- 4.** Make a “color preference” form that collects the user’s preferred foreground and background colors. Send it to a page that presents some information using those colors. Use the MVC approach, and make sure the resource is accessible only to registered users and administrators.
- 5.** For the SCWCD exam, unfortunately you have to memorize all the login-config, security-constraint, and security role elements and sub-elements. Borrrrrring. Make sure you also know the j_security_check, j_username, and j_password names. You do *not* need to know anything about the Tomcat-specific user/role/password file.

Exercises: Programmatic Security

- 1.** Make a JSP page that is accessible only to employees and executives. Display the username of the person accessing the page.
- 2.** Make a variation of the above page that also keeps track of the username of the previous person to access the page. Show the current username to everyone; show the name of the previous user only to executives. The simplest way to simulate two different users is to use two different browsers.
- 3.** Make a servlet that requires users to supply a non-empty username and password (any!) in order to access it. Have it simply display the username and password.
- 4.** The SCWCD exam objectives are unclear as to how much programmatic security you need to know. You need to know `isUserInRole` and `getRemoteUser`, at the very least.

Exercises: Filters

- 1.** Download the ReportFilter. Apply it to at least one servlet and at least one JSP page. Check the Tomcat window to verify that it is reporting accesses to those pages.
- 2.** Make a filter that prints a report saying whether or not the user accessing the designated resource is already logged in as an administrator. Test it by applying it to resources from the previous exercise. (Hint: just copy/rename the Web app that supported form-based security, and add in the filter.)⁷
- 3.** The name of your CEO keeps changing, but it is already embedded in several pages. Make a filter that updates those pages. Don't hardcode the names into the filter itself.
- 4.** Make a filter that turns the entire page (HTML tags and all) into lower case. Don't worry about whether or not this results in legal HTML (e.g., changing case of the DOCTYPE line is not legal, but you don't have to worry about that).
- 5.** Make a filter that removes all BLINK tags from the resources to which it is applied.

Note: the `replaceAll` method can be used to replace all occurrences of a given substring (regular expression, actually) by another substring. For example,

```
"foobarbazfoobarbaz".replaceAll("foo", "Test");  
returns
```

```
"TestbarbazTestbarbaz"
```

Prepend the first argument with `"(?i)"` for a case-insensitive replacement.

- 6.** The SCWCD exam objectives say or imply that you have to know all the methods in the Filter interface, the filter and filter-mapping web.xml elements, and the HttpServletResponseWrapper idea.

Exercises: Listeners

- 1.** Redo the JSP page that displays the company name. Have it read the company name from a servlet context attribute, and use a listener to set up the attribute. Have the listener also store the current value of the company stock, and display that value in the JSP page.
- 2.** Create a way for an authorized user in the executive role to change the company name.
- 3.** Arrange it so that, whenever the company name changes, the stock value increases by 25%.
- 4.** Make a listener that keeps track of how many sessions are currently active. Make a servlet or JSP page that displays the value.
- 5.** Create an HTML or JSP page that lets the user put Wonka's Wonder Widget in their shopping cart (that's the *only* item your site is selling). Use session tracking to show the user how many of these widgets are in their shopping cart—you don't have to handle the actual purchase, just the tracking of how many items each person has reserved. One wrinkle, though: if there are more than 20 active sessions, don't let **new** users create sessions: return an error message instead. Give users a "log out of session" button to simplify your testing. (Hint: in basic session tracking, you call `request.getSession()` (equivalent to `request.getSession(true)`), where the `true` indicates that the system should make a session if it cannot find a preexisting session for that client. If you call `request.getSession(false)`, the system returns an existing session if it finds it, and returns `null` otherwise.).
- 6.** Listeners are boring. I admit it. In real life, I hardly ever see anyone use a listener except for a `ServletContextListener` with `contextInitialized`. Sadly, however, for the SCWCD exam, you better memorize the class and method names of all of them.

Exercises: Basic Custom Tags

- 1.** Make a random number tag that inserts a random number (double) between 0 and 1 into the page. For example, you might use it like this: `<sample:randomNum/>`. Note that `Math.random()` returns a double between 0 and 1.
- 2.** Make a random int tag that inserts a random integer between 1 and some optional limit (default 10). For instance: `<sample:randomInt limit="782"/>`. If you multiple the result of `Math.random()` by the appropriate value, cast the result to an int, and add 1, you can easily generate random integers from 1 to some limit.
- 3.** Make a tag that results in whatever text it encloses being displayed in a large, bold, red, blinking format. For example:
`<sample:annoying>This is a test</sample:annoying>`.

If you want to see the text actually blinking, test on Netscape 4, Netscape 7, or Firefox; Internet Explorer and Netscape 6 wisely ignore the `BLINK` tag. (Firefox actually lets you set whether or not `BLINK` is ignored, but it is displayed by default.)

- 4.** Redo the previous three tags with JSP 2.0 tag files.
- 5.** The SCWCD exam objectives have two full sections that refer to tag libraries. You have to know Java-based tags, tag files, and (unfortunately!) the older (“classic”) version of the Java-based tags. You will probably never use the older version of the Java-based tags, so just skim that part. But, even though classic tags are not important in most real apps, tags in general are very important. So, the rest of the effort in learning tags is well worth it, SCWCD exam or no SCWCD exam.

Exercises: JSTL

1. Make a JSP page that creates a list of 13 random numbers. Use a simple JSP expression for each individual random number: `<%= Math.random() %>`.
2. Create a servlet that stores a bean containing an array of names, then forwards to a JSP page using `RequestDispatcher`. In the destination JSP page, loop down the array and put them into a bulleted list.
3. Repeat problem #2 with an `ArrayList` (or `LinkedList` or `Vector`) and `HashMap`.
4. Create a servlet that stores a bean with two arrays: one with first names and one with the corresponding last names. Have your JSP page make an HTML table with first names in the left table cell and last names in the right table cell. *Assume that the arrays have exactly five elements.*
5. Create a servlet that makes an array of `Name` objects, which have `firstName` and `lastName` properties. Repeat the HTML table, but this time you don't need to know how many entries you have in the array. Use the JSP 2.0 expression language as well as JSTL.
6. Repeat #4, but if the first name is "Marty," add "(Monty)" between the first and last names.
7. Repeat #4, but
 - If the first name is "Marty," replace it with "Monty" instead.
 - If the first name is "Bill," use "Microsoft" instead.
 - If the first name is "Scott," use "Sun" instead.
 - If the first name is "Larry," use "Oracle" instead.
8. In real life, the looping tags are very important. The conditional evaluation tags are slightly important. The others are virtually useless. However, the SCWCD exam objectives merely say "Given a design goal, use an appropriate JSP Standard Tag Library tag."

Student Survey

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

- Instructor: _____
- Topics: _____
- Handouts: _____
- Exercises: _____
- Book: _____
- Facilities: _____
- Overall: _____

2. What are your general comments on the course?
(I might quote you, but I won't use your name).

3. What were the strong points of the course?

4. What should be changed to improve it?