Practice Exercises

Exercises used in Marty's live Struts and advanced JSP training courses.
- See http://courses.coreservlets.com/ for public course schedule.
- Email hall@coreservlets.com to arrange an onsite course at *your* location.
- See http://courses.coreservlets.com/Course-Materials/ for Struts notes and examples.

Exercises: Servlet and JSP Review

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. Find Hello.html and Hello.jsp in the C:\Servlets+JSP directory. 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. Find HelloServlet.java in the C:\Servlets+JSP directory. Compile it using "javac HelloServlet.java" (DOS Window) or with "Compile Java"" (from the TextPad Tools menu), and copy the .class file to the shortcut to webapps/ROOT/WEB-INF/classes. Then, check that the servlet is working by using the URL http://localhost/servlet/HelloServlet.

4. Find HelloServlet2.java in 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. 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.

6. 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="...">.

7. 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.

8. 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.

Exercises: Servlet and JSP Review (Oracle Version)

1. Find Hello.html and Hello.jsp at http://volume1.coreservlets.com/. Make a Web application in JDeveloper and make sure you can run and view the two pages.

2. Download HelloServlet.java (a packageless servlet) from http://volume1.coreservlets.com/. Put it in the top-level of your Java code hierarchy, give it a URL, build your app in JDeveloper, and run the servlet.

3. Do the same thing with HelloServlet2.java. The difference is that HelloServlet2 is in the coreservlets package, so must go in the coreservlets directory (and the package name must appear in the web.xml declaration).

4. Create a subdirectory called "servletBasics" in your JDeveloper app, put a simple servlet in it (be sure to have the package name correspond to the directory name), build the application, and run the servlet.

5. Make a JSP page that randomly selects a background color for each request. Just choose at random among a small set of predefined colors. If you start with one of my examples (from http://volume1.coreservlets.com), be sure you do *not* use the JSP-Styles.css style sheet, since it overrides the colors given by
<BODY BGCOLOR="...">.

6. 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.

7. 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.

Exercises: the MVC Architecture

1. Make a "color preference" form that collects the user's preferred foreground and background colors. Send the data to a page that displays some message using those colors. Use a default value for any form value that the user fails to supply. So, for example, if the user goes directly to the servlet (bypassing the input form), the page should still work fine.

2. 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.

3. 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. How do you avoid race conditions?

4. Prohibit the foreground and background colors from being the same.

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.

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., ROOT) to make your Web app, you have this already. If you make your directory structure by hand, copy the web.xml file from ROOT (or from the archive.moreservlets.com Web site) to your Web app.

2. Put a servlet in your Web app and access it. Unless you've read ahead, you will have to use the invoker servlet (i.e., http://localhost/*webappName*/servlet/*ServletName*). However, in the next lecture we will see how to 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.

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 your 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.

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.

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. If you feel inspired, follow the directions in the *More Servlets and JSP* book and get SSL running on Tomcat on your PC. Note that you do not need to separately install JSSE, since you are running JDK 1.4 (which has JSSE included). Create a servlet or JSP page that displays a red heading when accessed via regular HTTP, but a green heading when accessed via SSL (https).

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: in JDK 1.4, 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.

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.).

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.

Exercises: Advanced Custom Tags

1. Make an *unblink* tag: one that removes all `BLINK` tags from the text it surrounds. Recall the replaceAll method from the filters exercises.

2. Make a `replace` tag that lets the JSP author specify a target string and a replacement. All occurrences of the target string within the enclosed text should be replaced by the replacement string. For example:
```
<sample:replace target="Marty" replacement="Monty">
<H1>Marty Hall's Home Page</H1>
</sample:replace>
```
should result in
Monty Hall's Home Page

Note: in JDK 1.4, 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.

3. Make a `heading` tag that expands into `<H1 ALIGN="CENTER">`... Make an `emphasis` tag that normally puts the enclosed text in bold. However, since headings are already in bold, if `emphasis` is used inside `heading`, it should put the enclosed text in italics instead.

Exercises: Struts Intro

1. Install Struts. Test with the struts-blank Web app.

2. Make your own Web app derived from struts-blank. Put a simple HTML or JSP page in it and make sure you can access it.

3. Take a look at WEB-INF/web.xml in struts-blank. See if you can figure out the purpose of the various entries (especially the servlet-mapping entries).

Struts and Actions

1. Make an application that lets users select a preferred health plan. The HTML form should gather the employee name, employee ID, and the name of the health plan selected. No matter what the user sends you, simply display a page that says "Thanks for Signing Up." Start by copying/renaming struts-blank, but I recommend you strip everything out of struts-config.xml that you are not using. In this case, all you need is a single entry in action-mappings; everything else can be deleted.

2. Gather the same information, but this time flip a coin (call Math.random() and compare to 0.5) and display either "We Are Honored to Have You in Our Plan" or "Get Lost, Loser!" Don't make a separate Web application; have all exercises on this page be different actions within the same app.

3. Continue with the same health plan application, and gather the same input data. But this time, say "OK" if all three values are present; otherwise say "Missing Name", "Missing ID", or "Missing Plan Name".

For now, don't worry about the situation if more than one value is missing; just give an error message for the first missing parameter. Also, for now, use separate pages for each of the error conditions. We'll see how to incorporate both enhancements later.

4. Make a third address that also lets the user select a preferred health plan, gathering the same information. But, this time just say "OK" if all three values are present; "Missing Data" otherwise. [Hint: you do not need to write a new Action.]

Struts and Beans

Make a new health plan application. You might want to copy/rename the previous Web app so that you can start with pieces of your existing code.

1. Use beans instead of calling request.getParameter explicitly.

2. Display the name, ID, and health plan in the confirmation page

3. Make a single page to display messages about missing data. List *all* the missing parameters, not just the first missing one as before.

Struts and Forms

Make another health plan application (copy/rename your previous one).

1. Make "Joe User" the default employee name, "t1234" the default employee ID, and "Blue Cross" the default health care provider. Display these values in the input form.

2. Redisplay the form if any of the values are missing or if the health care provider is anything other than "Blue Cross," "Red Cross," or "Black Diamond."

3. Extend problem 2 so that a warning message is given when the form is redisplayed. Hint: instead of using a separate bean to store the error message, put the error message in the form bean.

Struts and Messages

1. Take the title and prompts of your health plan signup form from a properties file. (Note that the button title is trickier than it seems: look closely at my example.)

2. Use parameterized messages for the error messages. (Remember that <%= ... %> provides values in standard JSP scripting.)

3. Internationalize your messages from #1 by allowing values in two other languages. (Go to google.com and click on "Language Tools.")

Struts and Advanced Actions

1. Make a new app where you gather the same information as in your health plan app. However, have three radio buttons: "New Enrollment", "Changed Information", and "Cancel Previous Enrollment". In all three cases, give a simple confirmation page that showed what choice they made.

2. Do the same thing, but by starting at three different input pages (one page each for "New Enrollment", "Changed Information", and "Cancel Previous Enrollment"). Hint: use hidden fields.

3. Convert your non-Struts-MVC example from the first set of exercises to use a form bean.

4. If you feel inspired, experiment with getMethodName by changing #1 to use push buttons instead of radio buttons.

Struts Declarative
Exception Handling

1. Use struts-config.xml to capture a NullPointerException. Create an Action that throws that exception. Create a JSP page that also does (read a non-existent request parameter and call toUpperCase or equals). Does it work from *any* JSP page?

2. Try throwing the exception from a form bean. What's the problem?

3. Use the standard web.xml approach to create an error page for NullPointerException. Which takes precedence: this approach or the struts-config.xml approach?

4. Use the page directive to create a page-specific error page for NullPointerException. Which takes precedence?

5. Create a custom error handler that prints a message to standard output. Change problem #1 to use that handler. Make sure that the error page of problem #1 still takes effect, even with the custom handler.

Manual Validation

1. For the health-plan application, enforce the restriction that all three values must be non-empty and the health plan name must be one of "Blue Cross," "Red Cross," or "Black Diamond.". When any fields are missing or illegal, redisplay the form with a list of missing fields. Do this in the execute method of the Action.

Note that you may have already done most or all of this in the previous set of exercises. If so, just use that as a starting point.

2. Do the same thing, but using the validate method of the form bean. Use the validate method for the rest of the examples as well.

3. Put the error messages in a red sidebar on the right side of the page, next to the HTML form. (Hint: copy the previous Web app, and then the only thing you need to change is the properties file.)

4. Put the error messages right next to the input form.

5. Require the employee ID to be a single character followed by exactly four digits.

Automatic Validation

Note. For details on the validator rules, see
http://struts.apache.org/userGuide/dev_validator.html

1. Enable server-side validation. Enforce the restriction that all three values must be non-empty.

2. Require the employee ID to be five characters long.

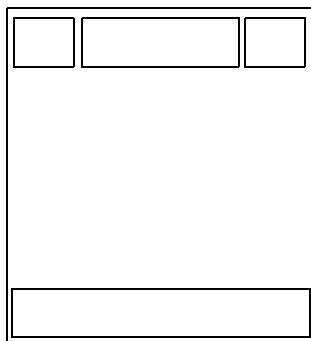**3.** Require the employee ID to be a single character followed by exactly four digits.
Note: here is the necessary regular expression for `mask`

```
<var>
  <var-name>mask</var-name>
  <var-value>^[a-zA-Z]{1}\d{4}$</var-value>
</var>
```

4. Enable client-side validation.

5. Try to evade the client-side validation.

6. Make a form that collects a name, password, employee number (6 digits without dashes), and email address. Verify that all values are present and of the appropriate format.

Tiles

1. Create a layout that looks like this:

   The top left and top right should be company logos (pictures), the top-center should be the page title in large letters, and the bottom should have contact information.

2. Make three concrete versions of this: corporate home page (title: the company name), corporate jobs page (title: "Careers at *xxxx*"), corporate contact page (title: "Contact Us").

3. Redo problem 2 using Tiles definitions.

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.

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?