B

General Exercise Instructions

The exercises in the book almost all involve building and deploying web applications. This appendix tells you what you will need to complete the exercises and covers

- What software you need and where to get it
- How to create a directory structure for your web applications and where to put the files you create
- How to set up your environment to compile Java source
- Where to put other files related to your web applications
- How to package your web applications
- How to deploy your web applications on the Tomcat application server
- How to make the supplied solutions work and how to inspect the source code

Downloading and Installing Software

There are three things you will need:

1. Java2 Standard Edition (J2SDK), version 1.4.2
2. Java2 Enterprise Edition (J2EE), version 1.4
3. Tomcat application server

When you download the Java2 Enterprise Edition (item 2 in the preceding list), you will have the opportunity to download the J2SDK (item 1 in the list) at the same time. So if you don't already have J2SDK 1.4.2, this might be a good option for you. Both of these pieces of software are available from Sun:

- J2SDK 1.4.2: http://java.sun.com/j2se/1.4.2/index.jsp
- J2EE 1.4: http://java.sun.com/j2ee/1.4/download.html#sdk

You should download

- All of J2SDK 1.4.2.
- Only the Platform API Documentation for J2EE 1.4 (include J2SDK 1.4.2 as well if you have not downloaded this separately). You won't need Sun's Application Server or the J2EE 1.4 SDK examples for the purposes of this book (though you're welcome, or course, to download everything anyway).

Follow Sun's download and installation instructions for your platform (this is a very straightforward process).

Tomcat is a freely available, open-source application server from Jakarta Project, part of the Apache Source Foundation. It also (still) advertises itself as the official reference implementation for Java Servlet and JavaServer Pages technologies. This may seem a little at odds with Sun's J2EE site, which now supplies its own application server (Sun Java System Application Server Platform Edition 8.1 at the time of writing). I've chosen to go with Tomcat for the purposes of this book. My opinion is that it's been around longer, has better documentation, and is a bit easier to use than the Sun equivalent. And since it seems to retain its status as an official reference implementation (so should behave exactly as the examiners expect), I'm happier recommending it. Of course, you can choose to do your own thing—use the Sun reference implementation instead, or perhaps a commercial application server like BEA's Weblogic or IBM's WebSphere. However, you'll have to adapt the instructions here and elsewhere in the book to suit. While I take care to point out when a description is server specific (so not part of the exam syllabus) as opposed to core J2EE behavior, this appendix and the exercise instructions rather assume that Tomcat is your deployment platform.

Tomcat can be downloaded by taking a link from the product's home page, which is http://jakarta.apache.org/tomcat/index.html. Version 5.0.n is the best to use for your exam preparation. I've used 5.0.27 in this book, but also cross-checked all the solution code on Tomcat 5.5. The reason for not using Tomcat 5.5 is that you should ideally run it on a 1.5.0 JDK (without exploiting any of the new language features). However, if you do decide to use Tomcat 5.5, you can adapt it to run on J2SE 1.4.2, but you will need to pay careful attention to the release notes (particularly the file RUNNING.txt). In all events, Tomcat is easy to install—just follow its simple installation instructions.

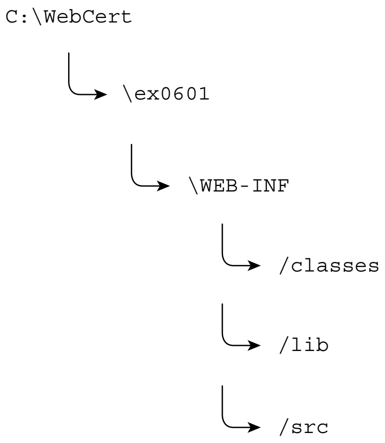
Creating a Directory Structure for Web Applications

As discussed in Chapter 2, J2EE web applications adhere to a fairly strict directory structure. Here's how I suggest you organize directories to perform the exercises in the book.

1. Create a top-level directory for all your exercise work, for example, C:\WebCert.
2. Within this, create a directory that represents the chapter and exercise number within the chapter as you come to it. The convention I follow throughout (in the solution files as well) is exCCEE, where "ex" is constant,

"CC" is the number of the chapter (01 for Chapter 1, 02 for Chapter 2, etc.), and "EE" is the number of the exercise (01 for the first exercise in a chapter, 02 for the second, and so on). So the first exercise in the sixth chapter has a dedicated directory called C:\WebCert\ex0601. This is equivalent to the context directory in the deployed web application. You can use the same naming convention for the context directory as I do—but do something different if you want to deploy your own code and the solution code side by side on the Tomcat server.

3. Within this, create a WEB-INF subdirectory (e.g., C:\WebCert\ex0601\WEB-INF).
4. With the WEB-INF subdirectory, create three subdirectories: classes, lib, and src: C:\WebCert\ex0601\WEB-INF\classes, C:\WebCert\ex0601\lib, and C:\WebCert\ex0601\src in our example. The following illustration shows the complete example structure.



You'll place Java source for servlets and any supporting classes in the WEB-INF\src subdirectory for a given exercise's context directory. If you are using package names, you'll need to create the relevant subdirectories under WEB-INF\src to support these. My solution files follow this package structure: webcert.chCC.exCCEE, where CC is the chapter number and EE is the exercise number within the chapter (zero prefixed). Therefore, the last element of the package name is the same as the context directory. You can use the same naming convention or use your own.

When I say "if you are using package names," it's not really an option in J2EE 1.4. The so-called "default package" is disallowed—and quite right, too. Whatever

Java source you're writing, there's no excuse for not providing a package structure! Although some JSP containers will let you get away with using the default package, I suggest you get into good habits from the start. As a bonus for me telling you this, note that the package name topic might come up in the exam!

Compiling Java Source

Having placed your web application source in the relevant directory, compiling from a command line is no different from normal Java projects. First change to the src directory for the web application you're working on. For example,

```
cd C:\WebCert\examp0401\WEB-INF\src
```

Since most of what you will be doing relies on servlet technology, however, you will have to ensure that the base J2EE servlet packages are available on your class path. These will live with your Tomcat installation in <tomcat-installation-directory>/common/lib/servlet-api.jar. For example, on my Windows machine, I set the classpath as follows:

```
set classpath=.;C:\Java\jakarta-tomcat-5.0.27\common\lib\
servlet-api.jar
```

To compile a Java source file (servlet or otherwise),

- Switch to the src directory under WEB-INF for the context directory for the exercise or lab.
- Enter a compilation command in the form

```
javac your/package/structure/YourServlet.java -d ..\classes
```

By using `-d ..\classes`, the compiled class files will be placed in the classes directory under WEB-INF.

Other Files

Any other files you need to create for the exercise should be placed in the web application directory structure as indicated by the exercise instructions. For example,

- web.xml—the web application deployment descriptor—will live directly in the WEB-INF directory.
- JavaServer Pages will normally live in the context directory of your web application directory structure (the one above WEB-INF, e.g., ex0601).

Packaging Web Applications

So you have your compiled code and other related files. The next thing you need to do is to package up your web application for deployment on a suitable server such as Tomcat. To do this, you use the `jar` utility to create a java archive, with the subtle difference that you call the resulting file a .war file (web archive). Web archive files are fully described in Chapter 2, but for now all you need to know is how to create one.

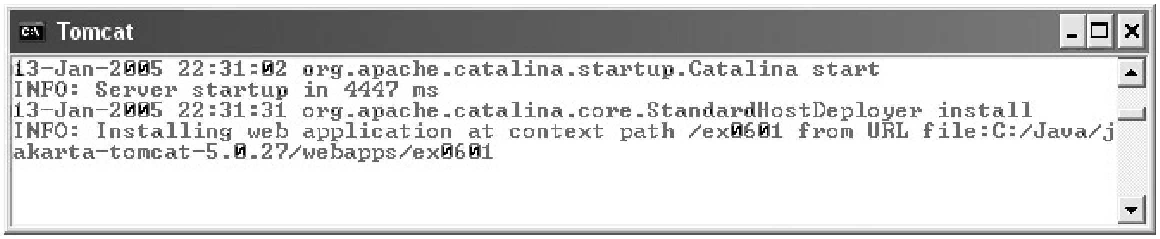
Suppose the name of the exercise context directory is ex0601. On a command line, navigate to the directory above this (C:\WebCert). Then enter the following:

```
jar cvf0 ex0601.war ex0601
```

This creates an archive file called ex0601.war in C:\WebCert, which contains the full contents of the C:\WebCert\ex0601 directory (including all the sub-directories and files within them).

Deploying Web Applications on Tomcat

Under Tomcat, deploying is the easy part. You'll need to follow the Tomcat documentation so you are at least familiar with starting and stopping the Tomcat server and looking at the Tomcat server console. Beyond that, the default behavior of Tomcat (for a new installation) is to automatically deploy web applications. All you have to do is to put the .war file you just created in the right place, which is <tomcat installation directory>/webapps. With the Tomcat application server started, you will see two things happen. First, messages appear in the console to denote that your application is being installed, as shown in the following illustration:



```
13-Jan-2005 22:31:02 org.apache.catalina.startup.Catalina start
INFO: Server startup in 4447 ms
13-Jan-2005 22:31:31 org.apache.catalina.core.StandardHostDeployer install
INFO: Installing web application at context path /ex0601 from URL file:C:/Java/j
akarta-tomcat-5.0.27/webapps/ex0601
```

Second, your .war file is "un-jarred" (unzipped) into its own directory structure under <tomcat installation directory>/webapps. The structure will mirror exactly the one you built in your development area C:\WebCert.

At this point, your application is ready to run. Again by default, Tomcat listens for your requests on port 8080 on the PC on which it is installed. Assuming you are running your browser on the same PC, you can access resources in your web application with a URL such as the following:

```
http://localhost:8080/<context name>/<resource name>
```

Therefore, if the context is ex0601 and the resource you want to access is called lifecycle.jsp, then the URL would be

```
http://localhost:8080/ex0601/lifecycle.jsp
```

If you get an HTTP 404 (page not found) error in your browser, then check the following:

- You spelled everything correctly, including using the correct upper- and lowercase letters.
- You have an appropriate `<servlet-mapping>` in web.xml (see Chapter 2).
- The application deployed correctly: There are no errors in the Tomcat server console window.
- The resource you're after has actually been expanded into the directory.

An HTTP 500 error indicates a Java problem running your servlet or server page—the error may show up in your browser window, or you may need to look for errors in the Tomcat server console window.

Tomcat is smart about recognizing when a .war file has been updated. Therefore, if you re-deploy a new .war to the webapps directory, overwriting one with an earlier date, Tomcat will elegantly close the current version of the application and deploy the new one.

In some of the earlier exercises, you bypass the steps of making and deploying the WAR file. On Tomcat, you can get away with copying the working directory structure for your application to the webapps directory. After a server restart, Tomcat will recognize your new application. However, this is more of a workaround until you are comfortable with WAR file creation—later exercises (from around Chapter 3) expect you to make WAR files from your working directories and deploy them, instead of copying your working directories directly to the Tomcat webapps directory.

Solution Code

The solution code for each of the exercises and the labs can be found on the CD. There's a .war file for almost every one (for the handful where this isn't true, the exercise is not directly code based). Deploying the solution code follows the same pattern as for your own code: Place each .war file as needed in <tomcat-installation-directory>/webapps. The .war will automatically expand and deploy. Look in the directory structure (under WEB-INF/src) for the accompanying Java source code, wherever appropriate.

Using Your Own IDE

What I've outlined here uses the most basic, but most commonly available development tooling—the command line facilities of the J2SDK. There is no compulsion to do this. If you are already familiar with an integrated development environment that supports servlet and JavaServer Page development and deployment, by all means use it. I freely confess that I haven't used the J2SDK in preparation of this book: I worked with the open-source Eclipse Java IDE, in combination with a plug-in called Lomboz, which facilitates J2EE web application development. Lomboz is freely available from http://www.objectlearn.com.

There are dangers in using an IDE to learn a technology for certification—their ease-of-use features sometimes obscure what you need to know for an exam. For example: IBM's Rational Application Developer has a sophisticated graphical user interface for manipulating the information in the web deployment descriptor, web.xml. For the exam, though, you must be familiar with web.xml in its naked, textual form. (And much to the credit of IBM Rational Application Developer, you have the choice to switch to a "source level" view of web.xml and edit it as a text file in tandem with the GUI approach.)

Some certification books take a purist approach and suggest that you avoid IDEs altogether for technology-learning purposes. I take a more pragmatic stance—do whatever you're comfortable with. Use the tools if you're already familiar with them and you're aware of the areas where the tools do too much for you. If you don't use tools already, or would rather avoid them while learning, work at ground level with a text editor and the J2SDK.