Capstone Courseware, LLC
33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Java Servlets

Seshadri Gokul
Will Provost

Instructor's Guide

Revision 2.4.3

Module Overview and Philosophy

This module is designed for the following audiences:
- Experienced J2SE programmers, who want to enter the realm of server-side Java computing
- Programmers with experience in J2SE and JSP

The module's initial emphasis is on inducting people with J2SE experience into the world of Java enterprise computing. Thus, the discussions in the first chapter may look familiar to the second audience, but this disparity will fall away quickly, as we get into the details of the Servlets API in the following chapters.

The first day of the course attempts to lay down a strong foundation on the basic features of Servlets and server-side Java in general. The discussions include basic Servlets API, request and response objects, how to process inputs from HTML forms, etc. Towards the end of the day, when students are more comfortable with interactive and session-aware applications, we migrate to advanced topics like Web applications architecture and servlet designs.

It is during the second day of the course that we start discussing situations closer to real-life enterprises and what servlet technologies can be used to deliver such applications. We begin with details of JDBC and procedures to connect servlets to a database – followed by issues of servlet configuration and context, sharing complex information throughout an application using JavaBeans, and servlet filters. Throughout, we have tried to provide practical tips on applying object-oriented methodologies to servlet concepts – apart from focusing on the necessary APIs and classes. For example, in the JDBC chapter, we discuss decoupling of persistence login from business login in the main servlets, and how to develop database handler or DAO classes to address this need.

In order to enable the students better to appreciate the points we discuss, we take up a couple of scenarios and build them up as case studies. Thus, the same examples – a simple application-managed login process and an electronic store –grow to encompass composite technologies like JDBC, JavaBeans, sessions management and filtering, as the course proceeds.

Timeline

Day 1

Chapter 1	Web Applications
Chapter 2	Servlets Architecture
Chapter 3	Interactive Web Applications

(May get a head-start on Chapter 4.)

Day 2

Chapter 4	Session Management
Chapter 5	Database Access
Chapter 6	Configuration and Context
Chapter 7	Filters

Ant Build Process

We use **ant** for all our builds. Though most students will be happy to let the **ant** command take care of things, some students (and most instructors) will want to understand the inner workings here a little better. Each project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant** which is typically **c:\Capstone\Servlets\Ant**. Information here defines a routine for building a web application.

The build will create a **build** subdirectory that is the root of a Java web application, and compile Java classes from **src** into **build/WEB-INF/classes**. It will copy the full tree of files under **docroot** to **build** – these are our JSPs, CSS, supporting data and image files, and configuration files including **web.xml**. Prior to building (and you could do the above tasks only, by typing **ant build**), the default Ant target (**all**) will undeploy any prior version of the application, and after the build it will (re-)deploy. It does this by generating a **<Context>** file into Tomcat's **conf/Catalina/localhost** tree, thus pointing Tomcat to the **build** directory as the root of the web application.

All this means that students can simply type **ant** from the command line set to any example step, demo or lab directory as the working directory – if there's a **build.xml** in the directory, you're good to go. Any change to Java, JSP, XML, etc., will be reflected in the re-deployed application – just give Tomcat it's 30 seconds to sweep for changes and re-install before testing from a browser.

Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse WTP 1.5 for this course. (See the course Setup Guide for download URLs.) Instructors, use this package on your own initiative and at your own risk. You should have experience yourself with Eclipse before using the overlay package in the classroom. The workspace and projects have been tested lightly with the course but are not part of the standard product.

That said, this overlay should save a good deal of work for those who wish to use Eclipse instead of the text editor and command-line tools that are standard for the course. See the file **c:\Capstone\Servlets\Eclipse\ReadMe.html** for general notes on how to use the Eclipse overlays for Capstone courses. Be prepared to walk students through the first few exercises in Eclipse; the notes in this file are for experienced Eclipse users, and will not be clear to many students on their own.

Also, note that the server configurations are the same as in the Eclipse workspaces for our JSP and JSTL courses. If you're teaching more than one of these modules, and don't want to have to swap workspaces, you can import the JSP and/or JSTL projects into the servlets workspace, and test them from there. <u>Be sure to build the project before trying to publish and test it on the server!</u> This is done in advance in the JSP and JSTL workspaces, but if you forget this step (a) it won't work, and (b) it really will be a huge pain to clean up.

Again, Capstone Courseware can only offer complete technical support on the standard course, and while we hope this overlay is convenient, it is not as thoroughly tested as the core lab image at this time. If a given exercise is giving trouble, please be certain to build and run it from the command line, using the SDK tools as prescribed in the student guide, before contacting Capstone.

Chapter 1

The main emphasis of this chapter is to introduce the world of server-side computing to programmers who have exposure to J2SE. Thus, the chapter begins with discussions of the HTTP protocol, Web applications, server-side programming, etc. It is quite possible that people with prior exposure to server-side computing or J2EE technology such as JSP will find these discussions less engaging. At the discretion of the instructor, these pages may either be quickly covered or can even be skipped altogether.

However, the later part of chapter on Java Servlets must be covered in depth. This portion introduces the concept of Servlets: topics like the API specification, components and containers, Web application structure, deployment descriptors and WAR files are exposed here for the first time. It is important that the students grasp these things properly, before proceeding further. Labs will be useful to get them familiar with the Tomcat Web container, Ant deployment and servlet invocation – knowledge which will be vital for all forthcoming labs.

The UML diagram on the "Servlets 'Kernel'" page may be a good one for students to dog-ear for further reference over the next few chapters.

Chapter 2

Now that the students are familiar with what servlets are, we need to delve deep into the servlets API to take a closer look at what classes and interfaces make up the API. This is not really a comprehensive academic discussion on the API; we have discussed only those classes and interfaces which will be frequently used by students during practical development.

In order to make these discussions easier and stick to a flow, we take a simple servlet code from the previous lab, dissect it line-by-line, and focus on the specifics. From the students' point of view, this will bring an in-depth understanding of the servlet code they saw earlier in labs, and not a dry discussion of APIs.

Once we are done with APIs, the generic servlet diagram will be a useful reference point, to which the students may want to return during labs. This single page encapsulates all the essentials for writing simple servlets. This is followed by discussions on servlet life cycle and servlet containers. It is important to emphasize the concept of containers, because of its importance in the overall J2EE architecture.

Chapter 3

Here, we begin to put servlets to use by developing simple responses to HTML requests. The initial discussions on HTML may be skipped, if the students are familiar with the topics.

During the latter part of the chapter, the topic gets more interesting with discussions on servlet design patterns. It is important to emphasize the importance of Web application design and architecture at this stage. The initial discussions look simple, but design four on generic template parsing may have to cross lot of debates and discussions before it is understood properly.

Chapter 4

We dedicate a chapter to session management at this point, since, along with user interaction through request parameters, it forms the basis of most Web application logic. The device of session attributes is quite simple – and we do take a moment to discuss the magic by which sessions are supported on behalf of the servlet – but the implications for application design are significant. After all, it is only with the help of session abstractions that Web applications have been able to approximate the interactivity and intimacy of use to which we've all grown accustomed with standalone/desktop applications. There is just one simple lab in this chapter, but several labs in later chapters reinforce the importance of session-management techniques.

Chapter 5

This chapter begins with a crash course in JDBC; this is necessary because JDBC experience is not a prerequisite for the course. Still, the focus is more on architectural and design considerations in developing Java servlets that access databases than on a throrough treatment of the JDBC API. Even the initial discussions on JDBC API may or may not be covered in detail, depending upon the proficiency of students.

The concept of database handlers – one might prefer the term data access objects – is introduced here, with tips on writing handlers for specific applications. Both of the labs reinforce the idea of separating business and persistence logic, although the designs by which persistence logic is abstracted are different.
It is important to setup the MySQL database properly and to populate it with required tables, as instructed in the student guide. All the exercises that follow - in the next two chapters, depend on these tables. Instructors are encouraged to pay special, advance attention to tool setup and testing in this area.

Chapter 6

The theme of this chapter is the necessity of pulling various sorts of values and logic out of Java source code and into XML or other artifacts that are consumed at deployment time or at runtime. This is one of the most central patterns in the entire J2EE platform, and can't be overemphasized: Java source should encode application-specific logic, and not be laden with literal values, mode switches, resource locations, logon information, etc., that can instead be provided to the compiled classes. Emphasizing configurability makes a compiled class more flexible, portable, and even re-locatable, and, most importantly, more robust in the face of changes to its environment, such as relocation of databases and files.

Accordingly, several ways to manage configurable data are discussed: initialization parameters and properties files are the simplest approaches, and JNDI is the most generally useful and powerful, especially in the broader context of J2EE.

Chapter 7

Filters are the final subject for the course. Their tactical use is simple enough (as is the final lab!), but the implications of the filter architecture are profound. Use of filters offers the application developer much better options for rigorous object-oriented design by fostering decoupling of vertical and horizontal features. It is worth noting that the filter-chain pattern established here is replicated in the JAX-RPC specification for component-based Web services: filters are to servlets as JAX-RPC handlers are to Web-service endpoints. This is clearly the basis for a pattern that is increasingly key to the J2EE vision for software architecture.

Revision History

Revision 2.4.3 is a maintenance release involving general code cleanup and some significant upgrades to the course infrastructure, including:

- Supporting tools – Ant, Tomcat, MySQL, and Connector/J – are now bundled with the lab software, which greatly simplifies classroom setup.
- Ant builds now deploy using Tomcat context files rather than WARs, resulting in many fewer Tomcat restarts during the run of the course.
- There is now full support for Eclipse WTP 1.5, including integrated deployment and testing of the web applications.

Revision 2.4.2 is a maintenance release and includes various minor fixes to book and lab code.

Revision 2.4.1, and also includes the following changes:

- Labs updated to the stable release of Tomcat 5 and tested against Netscape 7.1 and IE 6.0.
- Reworked the introductory chapter to focus less on Web history and more on Servlets architecture: how Web applications are structured, deployment descriptors, WARs, how URLs are resolved to servlets, and the development process. Retitled this chapter “Web Applications.”
- Added an example application TicTacToe that illustrates session state and use of filters.
- Added UML diagrams of servlets “kernel” types and filters.
- Removed first part of Chapter 7 on sharing JavaBeans between servlets. This chapter now focuses exclusively on filters, and is retitled accordingly.

Revision 2.4 is the initial release, supporting version 2.4 of the Servlets specification.

Troubleshooting and Tool Tips

First, be sure to be familiar with the lab and environment setup instructions in the Table of Contents. Assure that the instructions in the module Setup Guide have been followed, and that you know the locations of the installed tools. If, those steps having been followed, a classroom machine is failing to run demos or lab answers correctly, here is a list of items to consider and to double-check:

- Probably the most common mistake students will make is to try the database-oriented labs without either starting MySQL or creating the relevant database – double-check these things first when trying to diagnose problems.

Errata

The following issues have been reported since the release of this version of the course:

- In the final steps of the **Store** case study, the XML data file will not be found if running under Eclipse, unless the initialization parameter that gives the URL for this file is adjusted. The prepared text file with initialization parameters has a URL with port 8080 prepared, which is correct when running under Tomcat 5 on its own. But the Tomcat server configuration for the Eclipse overlay uses port 9080, so that it's possible to run the two instances concurrently if desired. That initialization parameter must be adjusted to make this work in Eclipse. (Why is this errata? We had a note entered into the Eclipse workspace documentation set to this effect – but we failed to include the file in the final distribution! It will be added next time 'round.)

Feedback

We truly do welcome feedback, both of a specific nature (pointing out mistakes) and general suggestions. For the former sending email with a numbered list of corrections would be most helpful.

Please send feedback to:

Will Provost
Capstone Courseware
mailto:provost@capstonecourseware.com
www.capstonecourseware.com