



**Capstone Courseware, LLC**

33 Boylston Street  
Jamaica Plain, MA 02130

877-227-2477  
capstonecourseware.com

# Java Development for Secure Systems

**Version 1.4**

**Instructor's Guide**



## Overview

It is true of many technology topics that various people will have widely different expectations of a course in that topic; but I think it's unusually so for Java security. There are security aspects to code design and implementation strategy, authentication practices, authorization policies, network protocols, server administration, and several other areas of practice for each one I just listed. For better or worse, this course focuses on a few Java-security concepts and techniques that seem to be central and most popular. Mostly this means the J2SE code-security architecture, JAAS, and J2EE authentication/authorization standards. Minor topics that relate to these are also included in the course: these would be key and certificate management, secure code design, and application-level cryptography.

And this says something about the arc of the course, because if you look over the chapter titles you'll see that the first, fifth, and sixth chapters cover the "major" topics, and two through four cover the "minor" ones. More about this in the timeline section, but you will often find that audiences are most interested in those three major chapters. In writing the course I've tried to tell a coherent, single story, as much as possible, and especially to bridge the gap between J2SE and J2EE security where possible. I do encourage everyone to cover the full range of the course topics. But as a practical matter you may well find that your students are especially keen to learn certain pieces of the bigger architecture, and are not really looking at security holistically. And so, by all means, the chapter divisions in this course are meant to make it fairly easy for you to emphasize some things and either de-emphasize or even exclude others.





## Timeline

The following breakdowns are approximate, and every class will vary.

### Day 1

5 hours	Chapter 1
3 hours, runs to day 2	Chapter 2

### Day 2

2.5 hours	Chapter 3
2.5 hours	Chapter 4

(Might get a start on Chapter 5 at the end of the day.)

### Day 3

5 hours	Chapter 5
3 hours	Chapter 6

The main thing is, don't panic if it's getting late in the first day and you're still on Chapter one! There's a whole lot of information to communicate there, and though I've looked for it there just doesn't seem to be a sensible way to break this topic down into smaller units. So we have a sort of behemoth chapter to get started. But it, like the rest of the course, is very heavy on quick, targeted examples and interactive demos – so hopefully students will still be with you by the end of it.

The JAAS chapter is similarly beefy, and between it and the final chapter on J2EE you will need at least a full day. Try to be certain either to get a jump on Chapter 5 by the end of the second day, or at least be ready for a clean start into it on the last day.

A two-day timeline is possible for this course, if some material is skipped. The likely candidates, for most audiences, will be Chapters 3 and 4. Both are potentially related to other course topics, but they are not needed for later material to be understood. If you skip Chapter 3, do take a little time to introduce the LandUse application that first appears there; it will be the basis of many later labs and so worth following a little bit through its infancy here.



## Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse 3.2 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\JavaSecurity\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.

One foible of Eclipse is that workspaces cannot be arbitrarily relocated without confusing the workbench a bit. To wit, if you choose not to install the workspace to the default, `c:\Capstone`, you will probably find that projects either don't build when they should, or that when you try to run them the Eclipse launcher "can't find the main class." The bottom line is the projects all need to be refreshed if they are opened from a path other than the one at which they were last open. So in this case the best process is to have everyone begin by opening, refreshing, and then closing all projects; this should clean up any odd behavior due to the "surprise" location.

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.

We always welcome feedback on our courseware, and we would appreciate whatever comments and criticism you might have. Eclipse, like all IDEs, tries to promote ease of use by providing many different ways of doing the same thing; this is convenient for users but does leave a lot of questions as to what's best practice. At this time we believe consensus is still forming around a number of basic practices, and we'd like to hear how you use Eclipse for training situations and how this overlay works out for you. Please contact Will Provost at [provost@capstonecourseware.com](mailto:provost@capstonecourseware.com).





## Teaching Notes

### Chapter 1

As mentioned in the timeline section, this chapter is a bit of a monster. What it does have going for it is a nice linear progression of topics, such that each new concept should be easy to digest: security manager to access controller to permission to policy. At the same time, the basic code-security model does have several major moving parts, and often the biggest challenge of this chapter is to reach the point where students see that whole picture, without getting confused along the way by details of one part before other parts have been introduced. Try to get students to accept a little hand-waving over the latter concepts – protection domains and policies – while you establish the earlier ones.

You may also run into some impatience among students who came to the class looking for J2EE security training. They may not think they have any real interest in or need for the J2SE model – and they may be right! But it's often a good move to point out the parallels early on, by first noting that a J2EE server is really nothing more than a very sophisticated, standalone J2SE application – and one that's publicly available and often responsible for sensitive resources. Essentially these are the points made at the beginning of Chapter 6, and it may help to forward-reference them in the early going. How will a J2EE server protect itself and its host from malevolent or (more likely) carelessly-written code?

### Chapter 2

This chapter is, largely, an extension of the first one. Some topics just had to be pulled out to a separate chapter, and the details of key management were severable in a way that many other topics were not. Here we learn about keys, certificates, and keystores, and in the process we begin to expand on our policy vocabulary, because now we can grant to a signer as well as a code source. (Chapter 5 will complete this progression, of course, by bringing in principals.)

Often the toughest concepts to convey are those having to do with certificates, certificate chains, and certificate authorities. There's a Russian-doll quality to this problem that frustrates many students, as certificates are supposed to solve the problem of trust but sometimes seem only to perpetuate it. The key point, and the one that usually ends the discussion (satisfactorily!), is that a root CA solves the trust problem not by being technologically special in any way, but by transcending the technical domain completely, and allowing the trust chain to travel into the business domain – which is where questions of who trusts whom ultimately should be solved. Digital signatures are a great invention, but they don't tell you who's trustworthy and who isn't – and they couldn't possibly.

## Chapter 3

In all honesty, this is likely to be the least popular chapter of the course. Students probably came to the class looking for technical details: APIs, implementation techniques, and so on. This chapter is much more about code design than implementation. But there are some eye-opening warnings here, for instance the Java Serialization vulnerability. And then some audiences will just eat this stuff up – and be ready for that, because the ideas in this chapter are also probably the most debatable in the course. That is, they're quite supportable, but while no one's going to argue with you that you have the signature of **checkPermission** wrong, they may be more bold on these design-level, best-practice proclamations.

Do be sure to cover the lab in this chapter – as a demo or code review if not as a full, step-by-step lab. For one thing, the LandUse application that's introduced here is used heavily the rest of the way, and for another, this is a nice example of defining and using a custom permission class, which is mentioned in Chapter 1 but not demonstrated.

## Chapter 4

Most audiences will either have a known requirement for cryptography, or they won't. Or, it's also pretty common that there's a need for signatures but not for encryption. This material may well be a bit more popular than what's in Chapter 3, but it is also even more severable from the rest of the course. This is an asset, too: it means that each piece of the chapter is a nice self-standing section, easy to digest and move on or easy to skip. If there is time and interest, the encryption lab at the end can be a fun one, as students can export their public keys to each other and pass secret messages.

## Chapter 5

This chapter approaches JAAS in an unconventional way, which is to carry out coding exercises on the authorization part of the API first, with no authentication piece in play at all. This may seem a bit odd, but it has a couple significant advantages. First, it's a way of chipping off one piece of the larger API at a time. That is, you can't have LoginContexts and LoginModules without Subjects, but you can, and we do, have Subjects without the other pieces. So it's more incremental. Second, it immediately ties JAAS in to what's been studied so far, because we see the connection to policy files right off the bat.

The demo on ANDs and ORs with **doAs** and **doAsPrivileged** methods is quite the brain-teaser, and usually students really turn on to it. But it's a challenge – good idea to go over this one with a fine-toothed comb before presenting it, because it's awfully easy to get things backward or confused while standing at the front of the room.



This chapter is the most lab-heavy, with three labs on the application side and one on the provider side of JAAS. Labs A and B overlap considerably, but if there's time, they do very well as a sequence, because B illustrates the flexibility of the callback architecture, and also how demanding it can be to code to it. C will rarely be of interest to enough students to make it worth the class time. It is illustrative, but maybe it's as much about JAAS' limitations as it is about JAAS' usefulness. So only the real diehards will want to tackle this one. Do note though that the problem with JFC event-handling threads will show up again in a place more near and dear to the typical student's heart, and that is in Tomcat's JAAS implementation. More on this in a moment.

## Chapter 6

For this chapter on J2EE security, we have to change gears in a couple ways. One is we're no longer studying a coherent, single architecture, as we were for J2SE, because there are still several overlapping approaches in J2EE that have yet to be sorted out into anything all that consistent, or that integrates with J2SE completely. The other is that we have to set at least one foot outside the area of standards and portable code, because so much of J2EE security is left to the server vendor. To this end we focus on one popular tool and use it for hands-on exercises: this of course is Tomcat.

The chapter covers pure-J2EE approaches such as web security constraints and HTTP authentication, and then tries to tie it all together by showing the Tomcat JAASRealm as a clean integration of JAAS with J2EE. Don't get too burdened with that last bit, unless students show a real interest. The demo and lab that get JAAS and J2SE policies working for a J2EE web application can be a nice final act to the show, so to speak; but it's a lot of work to make these things fit together, and there's plenty of value in showing J2EE security by itself, along with J2SE but not integrated with it.

During any of the later demonstrations and labs that involve on HTTP authentication, if you have extra time or strong interest in the subject, you might show off the actual HTTP headers involved. An example project not mentioned in the student guide is **HTTPSniffer**, which will forward packets from one local port to another (8079 to 8080 by default), and show the traffic in the console and in a log file. Try putting it between the browser and server for any J2EE application that declares a login configuration.



## Quiz: The Java 2 Security Model

You may find the quiz on the following page to be a useful learning device. We've had good feedback on it, when presented soon after covering Chapter 1, as a way of reinforcing the concepts in the first chapters. It could wait until after Chapter 2 as well, especially since this usually means it will time out to the end of the first day or the morning of the second day.

The following page is the blank quiz, and the pages after that offers some reasonable answers. You may print these out and distribute them to the class at whatever times seem to make sense.

Though a specific quiz has not been worked up for this purpose, you might want to revisit this material after Chapter 5, since JAAS changes some of the answers and adds a few new interesting classes to the mix.





## Quiz: The Java 2 Security Model

Try the following exercise, as a way of challenging your knowledge of the Java 2 security model. You may be familiar with the OOAD exercise known as CRC, or class/responsibility/collaborator. This is a way to test an OO design by stating the responsibilities of each class in the system, and also its collaborators. The focus here is on who does what, and often a CRC exercise will point out a flaw in the design because either a requirement is not met by any of the classes (not shown as a responsibility, and often a class that might have the responsibility instead states that another collaborating class meets the requirement, when it doesn't) or it is met redundantly (more than one class has the responsibility).

Carry out a brief CRC exercise for the following classes. For each, state the class name, its responsibilities, and its collaborators. Each class should be described in just a few sentences; we don't need to state individual methods and what they do, but we do want a clear picture of how these classes work in concert.

The classes to describe are: System, SecurityManager, AccessController, Permission, ProtectionDomain, CodeSource, and Policy.





## The Java 2 Security Model – Some Answers!

Here is one possible CRC summary:

**Class:** System

**Responsibilities:** Registers a SecurityManager. The presence or absence of a SecurityManager dictates whether security checks are carried out.

**Collaborators:** SecurityManager

**Class:** SecurityManager

**Responsibilities:** Starting point for any security check. In the Java 2 model, simply delegates to AccessController.

**Collaborators:** AccessController

**Class:** AccessController

**Responsibilities:** Central point of decision-making for system security. Grants or denies specific permissions to specific protection domains. AccessController is responsible for applying the principle of least privilege, which it does by walking the call stack of the current thread; deriving the protection domain for each piece of code involved in the call; and checking permissions for that domain. Only if all domains involved in the history of the thread have a given permission will that permission be granted.

**Collaborators:** Permission, ProtectionDomain, Policy

**Class:** Permission

**Responsibilities:** Encapsulates the concept of a specific action which a piece of code might want to take and that is considered potentially dangerous to the system. Permissions define names and actions, and also must state what other permissions are implied by itself.

**Collaborators:** None, except within the Permissions hierarchy.





**Class:** ProtectionDomain

**Responsibilities:** Represents the security attributes of a specific piece of code within the running system: where the code comes from, if and how it is signed, and how it was loaded into its current state in memory.

**Collaborators:** CodeSource, ClassLoader

**Class:** CodeSource

**Responsibilities:** Represents the source from which some Java code originates, and also whether or not it is signed, and by whom.

**Collaborators:** None.

**Class:** Policy

**Responsibilities:** Grants permissions to specific code sources, perhaps with signature requirements. The policy is the final document of what should and shouldn't be allowed. As AccessController carries out its thread-walk it needs to check on the advisability of allowing a specific permission to a specific protection domain; Policy is the class that can answer that question.

**Collaborators:** CodeSource, ProtectionDomain





## Revision History

**Version 1.4** is the initial release of the course.





## Troubleshooting

If you run into any trouble with code exercises, the first and best thing to do is to double-check that the classroom machines have been set up precisely according to the course setup guide. Especially, the wrong version of a tool can cause significant problems; don't wander off-book in this way unless absolutely sure you can support the software that you prefer and that we haven't tested. Check environment variable settings carefully, too; these are the cause of a great many classroom glitches.

Below are some specific pitfalls that have come up in previous offerings of the course:

- You can run into unpleasant surprises with the configuration of the machines, especially where multiple JREs and/or SDKs are installed. This tends to crop up for the first time in the SOAPSniffer demo near the end of Chapter 1: if you edit the **java.policy** file and don't see any changes in behavior, there's a good chance you're running a different **java.exe** than you think. To be sure, try setting the executable path to just your own **%JAVA\_HOME%\bin**. (In later exercises, some build scripts use **xcopy**, which is in the Windows **System32** directory; so be sure to build up a more complete path, once you've diagnosed the problem.)





## Errata

At this time there are no errata for the course.





## Feedback

We very much appreciate whatever feedback we can get on our courseware – especially from the instructor's perspective. Naturally, the more specific, the better, and we strongly encourage you to make notes on issues you may encounter in the classroom, whether they're typos, missing files, or suggestions for clearer language to explain a concept. We can't guarantee that we'll act on every suggestion, but we're aggressive about stamping out problems and try to be highly responsive. Hopefully this means that when you give us good feedback, you get a better course the next time you need to teach it.

Please direct all courseware feedback to

Will Provost  
Capstone Courseware  
<mailto:provost@capcourse.com>  
877-227-2477

For anyone who's interested, we have a very informal defect-tracking system, based in Excel spreadsheets with columns to capture defect location, nature, status, and author feedback. Ultimately, feedback goes into these sheets, so if you want a template, we'll be happy to provide one, to facilitate the reporting process.

