



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Advanced Java Programming

**Will Provost
Edward Rayl**

Instructor's Guide

Version 6.0



Revision Notes

Revision 6.0 updates to Java 6 and also pares the course down to its first module, dropping some material and letting other courses carry the load for other topics including JDBC, security, Swing, and design patterns.

Revision 5.0 broadly updates and rearranges the course with the primary aim of supporting training in Java 5.0 and uses the 5.0 JDK. It also adds a large case-study application that illustrates J2SE multi-tier development, many design patterns, and best practices. Specific changes by module:

- The first module drops the Serialization chapter (which is already in our Courses 103 and 104) and adds chapters on Generics and Annotations. The Threads chapter has a all-new code exercises; the Reflection chapter is updated to take cognizance of Generics
- A new second module shows off the HR case-study application, and its chapters are meant to be interspersed with the remaining modules – see the discussion under the Timeline section later in this guide.
- The JFC module has been trimmed back by one chapter, with the case-study module doing more to show off patterns and use of advanced controls JList, JTable and JTree anyway.
- The RMI module has been condensed to two chapters and generally cut back to more of a quick practical treatment.
- The JDBC module now supports Derby 10.1, which in turn supports updateable result sets. The separate code that had been required for Lab 3B is gone, and the standard code will work for Derby. The module also adds support for a fourth RDBMS – PostgreSQL – and incorporates RowSets as part of the 5.0 Core API.
- The instructions for creating the course schema objects for each supported RDBMS are now in external XHTML files.

Revision 1.4 is the initial revision, targeting the J2SE 1.4 SDK.





Course Overview and Philosophy

Where previous versions of this course attempted to fill an entire week with some combination of topics that seemed to fill the market's craving for "advanced Java," this latest revision is greatly simplified: it is a two-day course on a few topics that we can really say are advanced Java, as opposed to being perhaps a study of some specific wing of the broad Core API (JDBC, Swing, RMI, etc.) That is, we're focusing on better understanding and more effective use of the Java language – e.g. use of threads, language features such as generics and annotations, etc. The one outlier might be the final chapter on sockets programming; this certainly is a specific core package, but it's also a basic and useful skillset that doesn't occur in most first-week Java courses (ours included).

Also, material in this course is meant to combine pretty much seamlessly with our intro and intermediate Java courses, when custom solutions are necessary, and this is a common scenario – some chapters from Course 103 and some chapters from this one, maybe along with JDBC or Swing material. All these courses run with a bare-bones environment of build and run scripts, and shifting from one lab module to another should be easy for students.





Timeline

Day 1

Chapter 1	Generics
Chapter 2	Threads
Chapter 3	Reflection (probably spans the two days)

Day 2

Chapter 4	Annotations
Chapter 5	Sockets





The Eclipse Overlay

Some students and instructors may prefer to install an IDE and use it in working through the course exercises. Capstone provides an optional package of workspace and project files for Eclipse 3.1 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 workspaces and projects have been tested lightly with the course but are not part of the standard product.

That said, these overlays 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\JavaAdv\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 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.





Teaching Notes

Chapter 1:

Though our standard/intermediate Java courses (103 and 104) cover generics, it seems appropriate for this audience to go deeper. This chapter begins with the basics, assuming no prior knowledge of generics, gives a few quick examples including use of collections, and then moves into the truly advanced territory of designing and implementing one's own generic types.

Depending on who one talks to, the generics feature in Java is either a clever 80/20 compromise of a few advanced features for immediate usability and backward compatibility ... or hopelessly broken. As the author of this chapter I'm drawn more to the former assessment. Generics bring some nice immediate benefits in type safety and making code more self-documenting – a `List<String>` is clearer about its intent and meaning than a `List` – and they make coding easier, especially in conjunction with the for-each loop. But the later part of this chapter confronts some of the implications of the generics design that are somewhere between counter-intuitive and truly bizarre. The aim is not to catalog every do and don't of generics – and it's a little early in the use of Java 5 to do so without being presumptuous. Hopefully students will get a good exposure to some useful techniques, as well as some common pitfalls, and be in a good position to carry out their own designs and experiments with generics.

The `javap` tool is a nice way to illustrate type erasure. As a quick demo, run `javap` on a generic type and see the type parameter converted to `Object` in all cases. Note, however, that `javap` is not using all the latest features of the Reflection API – you can come back to this when you hit the last section in Chapter 3.

Another implication of type erasure, not mentioned in the student guide, is that a class cannot implement two interfaces `G<X>` and `G<Y>`, since at runtime the method signatures would collide.

Chapter 2:

Threads really are a basic feature of the Java language. However, most programmers won't use threads consciously for at least a year into full-time Java programming, either because they're writing things that don't use multithreading at all or because they move on to J2EE, where for component-building multithreading is discouraged or forbidden. Still, it's good to understand how threads function in the JVM; if nothing else to know that they're there, understand how the garbage collector is working, and to know where those exception stack traces come from! So this chapter starts almost from scratch, introducing threading concepts as well as the Java threading API. In this one chapter we just get a taste of the larger API; there is certainly room to expand here if students are interested.



Chapter 3:

Again, in one chapter we can't go too deep into the Reflection API; we could spend a day on this for an audience of specialists. This chapter concentrates on the role of meta-data in the Java runtime, its exposure to application code through the API, and the ability to read meta-data from an application. The middle section introduces the powerful concept that reflection can be used to make things happen, and not just to read about them and perform diagnostics. Here again one could go further if students are interested.

Feel free to skip or skim the last section on reflecting on generic types – it is interesting, and actually teaches more about generics than about reflection, but it doesn't develop any new skills with either.

Chapter 4:

Annotations are one of the less well-understood features of Java. They may be so foreign to students as to be unapproachable – or they may be seen as a critical meta-data feature. This chapter is fairly short, so it shouldn't be a problem to work through it even if students are not strongly motivated in this area. At the least, it's a good idea to introduce the built-in annotations, such as `@Override` and `@SuppressWarnings`, as these are already appearing in production code here and there, and students will otherwise be in the dark about the grammar/legality/meaning of these things.

Chapter 5:

This is the first of two APIs that connect Java code to standards outside the Java architecture – the second being JDBC, with its relationship to SQL. Here we must first understand the OSI/RM and "what's a socket?" before teaching the API. There's no need to go much deeper than the presentation material in this first section, unless students are curious; but it sets up key concepts and terms including TCP and UDP. Then we pursue two common goals for sockets programming: connected conversation over TCP – specifically HTTP, which is so very common – and connectionless communication using UDP and datagrams. The labs should give both a "cookbook" sense of how to get started with a sockets programming task and an introduction to the sorts of processing problems that are common in sockets coding: timing sends and receives based on content length or an end-of-transmission byte, blocking vs. non-blocking, etc.





Troubleshooting and Tool Tips

First, be sure to be familiar with the lab and environment setup instructions in the Table of Contents and the course Setup Guide. Assure that the instructions in the Setup Guide have been followed, and that you know the locations of the installed tools, passwords, etc.

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

