



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Java Programming

Will Provost

Instructor's Guide

Revision 1.4.3



Revision Notes

Revision 1.4.3 drops two chapters – Threads and Reflection – to avoid redundancy with the new Course 106, “Advanced Java.” Course 106 begins with these two topics. (Note that we keep the final chapter on Serialization, even though this also appears in 106. We like ending with this topic in this course, but find that many other Java courses don’t include it, so 106 repeats the chapter for those coming to it from other vendors’ courses. If you are teaching both this course and 106, it’s probably better to leave it until that second week of training and bring it in during the normal run of 106.)

Also changed the convention for numbering labs to use a leading zero, as in “Lab02”, so that directories and project names all sort nicely.

Revision 1.4.2 is a maintenance release designed specifically to take cognizance of the 1.5 release of the J2SE. The course at this stage still teaches 1.4, but includes tips on what to expect when migrating to 1.5.

- Various quick tips are sprinkled through the course, especially where pointing out a current limitation of 1.4 and the 1.5 improvement. Watch for the green “Java 1.5” icons.
- Chapter 10 seemed the best place to spend a few pages laying out some of the most popular 1.5 features in more detail: generics, for-each, autoboxing, varargs, and C-style formatting. This section includes two non-working examples – which is to say they work just fine with a 1.5 JDK! but not with the 1.4.2 SDK that will be in the classroom

Revision 1.4.1 is a maintenance release, with various fixes for book and code. A few highlights:

- A problem with environment variables as used in Linux .sh scripts has been corrected.
- Discussion of object references and arrays had been referring to these as JVM primitives. This was arguable at best and confused the discussion for students. This language has been rewritten and clarified.
- A couple of labs that had no build and run scripts in their starter directories have been corrected.





Revision 1.4 reorganizes the course in several major ways:

- The five-module structure is collapsed to a single one-week course. (We may well break this back up into two modules in the near future, allowing the first 9 chapters to be covered at a slower pace in Course 102.)
- The old fourth module on AWT has been dropped in favor of more depth on other subjects.
- The flow of topics has been tuned to make the learning curve less steep for those with no OO experience, and for non-C-family programmers, with more incremental treatment and lots of labs in the early going. Those with good experience in these areas will move at a faster pace and will probably skip several of the labs in the first half of the course.
- A few topics that were threaded into the early, non-OO part of the course have been deferred until after the OO chapters: exception handling and threads.
- There are new chapters on "Using Classes Effectively" and Java Reflection.

Revision 1.3 adds tested support for Linux and for J2SE 1.4. There have been a few minor fixes as part of this testing; the course content is largely unchanged.

Revision 1.2 is primarily a structural revision, bringing this course's document and file arrangement in line with Capstone's current standards. Labs have been re-tested to assure compliance with JDK/JRE 1.3. There are some technical and typographical fixes, as well.

Revision 1.1 fixes some minor typos and lab and install glitches.

Revision 1.0 is the initial revision.





Course Overview and Philosophy

The biggest challenge with teaching a language as complex as Java, from scratch, is the great variety of possible student backgrounds. This course is intended for programmers with some experience in languages other than Java. It is not for brand-new programmers, and for very new programmers or those with experience only in languages very unlike Java – COBOL, PL/1, SQL, various 4GLs – this may not be the perfect fit. The ideal student has been programming for a year or so in C or C++, but VB programmers and other 3GL programmers should be comfortable here, and other audiences may simply pursue earlier sections of the course at a slower pace.

With this likely diversity of backgrounds in mind, the course has been redesigned most recently to accommodate students for whom certain Java concepts will be new: specifically, strong type models and compilation (vs. weak typing as in VB or scripting languages); structured programming; and especially object-oriented concepts including encapsulation, inheritance, and polymorphism. The course is meant to be adaptable to many different paces, with lots of optional labs in the first half of the course which will probably be appropriate to some audiences but for, say, experienced C++ coders, can be skipped.

We continue to target a classroom environment that is stripped down to a J2SE SDK and a text editor. This is partly to assure that the course exposes students to “nuts-and-bolts” topics and practices for certain sections of the course, and partly because there are still so many popular IDEs that it didn't seem to make sense to single one out. Many instructors will prefer to introduce students to an IDE as part of the training. We encourage this, and the layout of the lab software should make it very easy to wrap exercises in IDE projects.





Timelines

Day 1

Chapter 1	The Java Environment
Chapter 2	Language Fundamentals
Chapter 3	Data Types
Chapter 4	Flow Control (spans days)

Day 2

Chapter 4	Flow Control (spans days)
Chapter 5	Object-Oriented Software
Chapter 6	Classes and Objects

Day 3

Chapter 7	Inheritance and Polymorphism
Chapter 8	Using Classes Effectively
Chapter 9	Interfaces and Abstract Classes

Day 4

Chapter 10	Collections
Chapter 11	Exception Handling
Chapter 12	Inner Classes
Chapter 13	Streams (spans days)

Day 5

Chapter 13	Streams (spans days)
Chapter 14	Working with Files
Chapter 15	Advanced Stream Techniques
Chapter 16	Object Serialization

As discussed above, different audiences will require different paces. If time is short, the whole last day on streams can be dropped in favor of a slower pace over the first 12 chapters – this may be most appropriate for audiences coming from languages or environments very unlike Java, and should probably be decided prior to class time.





The Eclipse Overlay

Some students and instructors may prefer to install an IDE and use it in working through the course exercises. Capstone Courseware provides an optional package of workspace and project files for Eclipse 3.0.2 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\Java\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 especially with this new undertaking 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**.





Chapter 1:

This is conceptually one of the more challenging chapters to present, because there are so many misconceptions about the nature of Java, the Java architecture, the JVM. We try to lay out as clearly as possible how Java software is supposed to work, what the major roles are, and why Java is what it is and does what it does. This chapter is also appropriate for non-programmers, for instance administrators or managers who want some high-level understanding of Java as a technology. The lab does not involve any coding, just exercises in working with the SDK installation, classpath, etc.

The installation disk for this course runs the **javadoc** tool on all the examples, labs and demos after they have been unpacked to the hard drive. Thus the installation will take several minutes. On some older machines it has taken as long as a half-hour. So be warned: it's not a good idea to wait until after your first lecture to have students install the courseware! We recommend beginning the day with a quick walk through the table of contents, which introduces the lab installer and the directory structure for the software, and this is a good time to have students kick off the installer, so that it's done while you're starting the first-chapter lecture.

Chapter 2:

We begin a climb from the overview in Chapter 1 to round out an understanding of how to write procedural Java code. This takes three chapters to do! and the road is sort of dusty and dry, but we do have to start with the nuts and bolts of the grammar. This chapter lays out the most atomic parts of the grammar (at least the most atomic parts anyone in class is likely to care about): identifiers, literals, operators, and a few keywords. Then we build various expressions, and get comfortable with the expression-based coding style that leads to lots of nesting and chaining, which may be unfamiliar to some students. The labs for this and the next chapter are necessarily limited, since we haven't introduced things like conditionals and loops and so can't do very interesting processing.

Chapter 3:

This chapter covers the primitive data types including object references. It should play out straightforwardly. The part on object references is probably the only tricky part, and more so because it will raise questions about OO Java in general, and we're pointedly trying to avoid that subject for these first few chapters, trying to get basic coding skills nailed down first. It is really only here at all because (a) object references are, technically, primitive types in the JVM, and (b) we want to start working with strings. Arrays round out the chapter, and this shouldn't be difficult, especially since the harder work of actually processing them requires information in the following chapter.





Chapter 4:

This chapter covers basic flow control constructs, including conditionals and loops, and labeling of loops for use with **break** and **continue**. These techniques are essential, of course, and with them in hand students should be able to go wild on various sorts of programming problems. This chapter seems to complete a kernel of knowledge about procedural Java programming, and as such there are many labs, ranging from basic to optional practice to a very challenging final exercise in writing sorting algorithms. (This last lab will only rarely be undertaken in class; it's more for review and/or work after class, or perhaps as a midterm exam of sorts.) The part on recursion near the end of the chapter might be a bit much for some, but we introduce it here in a very simple way and then come back to it in some later labs as a practical tool.

Chapter 5:

Now we jump up onto the object-oriented plateau, and so start thinking about Java software more properly as classes and objects. This first chapter of the section is not Java-specific in any way. It introduces OO concepts and is meant to be covered by students without a strong OO background – which will be most students, really. Only those with plenty of practical experience in C++, Smalltalk, Ada, or another OO language (and no, VB doesn't count) should skip this chapter. The chapter also introduces some rudimentary UML, since that will serve as an important notation system through the rest of the course.

This chapter also introduces the primary case study for the course: the car dealership application. It's considered as an abstract analysis-and-design problem for the moment, and in Chapters 6-9 it will be worked up to completion, and the design enhanced along the way to illustrate topics in those chapters, from basic encapsulation through to interfaces and abstract classes.

Chapter 6:

Here we dive into Java as a true object-oriented language for the first time. This chapter focuses on encapsulation, and the relationships between classes and objects, object references, etc. This chapter is the longest in the course, because there's so much to discuss all at once about OO Java, and a good deal of that has to be covered before real lab work can begin. Don't hesitate to take plenty of time on this and the next chapter, as they include the concepts that are the hardest to grasp. Also, there are a couple of topics that are not strictly tied to OO but which seem to fit best here – those are packages and JARs. Packages are essential, and are used throughout the rest of the course; JARs could certainly be skipped or left for work after class.





Chapter 7:

The previous chapter covers basic encapsulation, which can be challenging for students with certain backgrounds. This chapter is often the one that really sets students back in their seats, because if they're still vague on the encapsulation and object-identity ideas, inheritance and especially polymorphism can be really elusive. As with Chapter 6, take the time you need to get these concepts across solidly. They will be reinforced in later labs, but successive chapters do build on these basic ideas. With a firm grasp of polymorphism, many later chapters will come clear very quickly – interfaces and abstract classes, collections, reflection, etc. – but conversely it will be hard to make good progress from here until they are really nailed down.

Chapter 8:

This chapter is meant to catch up on some peripheral matters relevant to Java classes that have been pushed off over the last two chapters in order to focus on the core concepts. Static fields and methods, although used here and there in previous chapters, are now confronted head-on – and this is one of those topics that really requires students to “get” polymorphism to be understood well. The latter half of the chapter addresses object-creation costs and uses this context to get to an important practical matter for most programmers, which is knowing when to use strings and when to use string buffers.

Chapter 9:

We come back to raw-OO concepts for one last chapter, having deferred the idea of abstract types until now. Conceptually this chapter is not so large, although it includes a good deal of lab work. Like Chapter 4, in a way, this chapter seems to complete a level of knowledge about Java programming, and so some extra lab work seems appropriate for students who are still catching up to the total OO philosophy and practice.





Chapter 10:

From this point forward, the style of the course shifts somewhat. Until now we've been developing fundamental language and OO skills. Now we start to move towards better effectiveness in Java development, by adding some key parts of the Core API to the toolbox. This chapter covers the Collections API, which, while not really a basic language skill, is essential to effective Java programming in most people's eyes. We don't delve into every collection type or API method, but do get a good sense of the utility of collections and some practical experience with Collections, Maps, and SortedSets, as well as algorithmic programming using Iterators.

This is the chapter in which the Java 1.5 language enhancements are discussed – and not just quickly referenced, as in other chapters. On one hand this is the natural location for it, since collections expose so many of the key features; on the other students will probably have been curious long before now! Nothing wrong with entertaining questions on 1.5 throughout the course, but it will probably be best to do a lot of pointing ahead to this chapter. Then take some time to consider these features in a bit more detail, show the code examples, etc.

For the adventurous (and expert), it is possible to set up a system that will run both the 1.4.2 examples that make up most of the course and the 1.5 examples mentioned here. Either install a 1.5 JDK and compile most code with the appropriate switches for 1.4, or install both a 1.4.2 and a 1.5 SDK on the same system, and control which one runs at a given time with paths. Since this is likely to be problematic for students, we strongly recommend against doing it on their machines; but an instructor may (at his or her own risk) give it a whirl, and then be able to demonstrate **Cars1.5** and **Scores1.5**.

Chapter 11:

One last fundamental skill! It's a rare student whose favorite topic is exception or error handling, but – sort of like a trip to the dentist – it must be done. Lab work in this chapter is relatively light, just enough to see the basic mechanism in play. Exception handling is threaded into some later labs as well, often in optional, final steps.

Chapter 12:

This short chapter on inner classes is a necessity, and some students will instinctively see the advantages of scoping class names and inner-to-outer object references. Static inner classes will be the most familiar to C++ coders. Most students, though, will just find the syntax bizarre and the motivation dubious – this may take some evangelizing. Anonymous and method classes are especially hard features to defend, even though they do have their place in practical solutions – GUI development is probably the easiest example for the whiteboard.





Chapter 13:

This chapter introduces the streams model without focusing on any particular stream backing (such as a file). Inevitably, some example code has to use files as a medium, but the conceptual emphasis should be on the structure of the **java.io** package, delegation from stream to stream in particular. Accordingly, the lab relies on the standard streams as media and focuses on building a filtering stream.

Chapter 14:

This chapter introduces file system management and file input/output. We do a little streaming, but the focus is on managing file systems, navigating, recursing through directories, etc. The lab gives more exercise in recursive methods, and in fact recursive creation of objects in this case.

Chapter 15:

This chapter works into the streams library in more detail, and in the process finally marries streams to files in all its lab work. Presentation should be pretty straightforward, and the lab looks at a common real-world requirement, which is simply that of writing and reading data in a compact but reliable binary format; we also do a little more work with exception handling.

Chapter 16:

The big finish! This chapter introduces Object Serialization as the logical extension of data formatting streams. Concentrate on the tremendous flexibility of the serialization engine: how completely generic, yet how powerful and simple to use. We note the basic pitfalls, in particular re-initialization of transient fields on object reads, but the chapter does not try to go deep on issues like versioning or externalization. The lab should be a nice way to wrap up the case study, and to show off the simplicity of use that the Serialization API offers.





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

