



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Intermediate Java Programming

Will Provost

Instructor's Guide

Revision 1.4.3



Revision Notes

Revision 1.4.3 is the initial revision – this course is based on Capstone Course 103 and was first created from revision 1.4.3 of that course.





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 previous experience in Java, but limited to procedural and structured programming. 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 – our Course 102, “Introduction to Java Programming,” will be a more likely fit. Not is it for C/C++ experts, who will probably want to learn the basics of structured programming in Java first, rather than making incorrect assumptions about similarities between Java and C that do not exist! For these students Course 103, “Java Programming,” will be a better choice. The ideal student for this course has either just completed Course 102 or has some practical experience with Java but lacks a clear understanding of the object-oriented nature of the language.

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. See the upcoming section on the optional Eclipse overlay available for the course.





Timelines

Day 1

| | |
|-----------|----------------------------------|
| Chapter 1 | Review of Java Fundamentals |
| Chapter 2 | Object-Oriented Software |
| Chapter 3 | Classes and Objects (spans days) |

Day 2

| | |
|-----------|----------------------------------|
| Chapter 3 | Classes and Objects (spans days) |
| Chapter 4 | Inheritance and Polymorphism |
| Chapter 5 | Using Classes Effectively |

Day 3

| | |
|-----------|---------------------------------|
| Chapter 6 | Interfaces and Abstract Classes |
| Chapter 7 | Collections |
| Chapter 8 | Exception Handling |

Day 4

| | |
|------------|--------------------|
| Chapter 9 | Inner Classes |
| Chapter 10 | Streams |
| Chapter 11 | Working with Files |

Day 5

| | |
|------------|----------------------------|
| Chapter 12 | Advanced Stream Techniques |
| Chapter 13 | Object Serialization |





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:

Though this course states fluency in structured Java programming as a prerequisite, in practice it is common enough to encounter audiences with mixed backgrounds, and some students may not have as strong a grasp as is needed, either of structured programming concepts or of the specifics of Java syntax, data types, and flow-control techniques. In the ideal case, you should be able to cover just the first few pages of this chapter – enough to introduce the structure of the course's code exercises and take students through a quick build-and-test of the first example program – and then move right on to Chapter 2. But, again, in practice, it is usually prudent to cover this chapter in toto: skim over the pages and gauge students' comfort level with the material as you go, and slow down or stop as necessary to reinforce certain concepts or skills. The labs in the chapter are worth at least a review, but full instructions are provided so that they can be executed in full if it seems that many or most students need that extra practice in basic coding or other skills. We allow around 2 hours for this chapter in the standard timeline just for this purpose – if this is really not needed then take the extra time on Chapter 3, which because it brings OO Java into play must break quite a lot of news all at once.

Chapter 2:

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 3-6 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 3:

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 4:

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 3, 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 5:

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 6:

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. 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 7:

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 8:

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 9:

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 10:

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 11:

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 12:

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 13:

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

