



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Java Programming

Version 6.0

Instructor's Guide



Overview

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 designed 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.





Timeline

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

Day 1

Chapter 1	The Java Environment
Chapter 2	Language Fundamentals
Chapter 3	Data Types

Day 2

Chapter 4	Flow Control
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	Generics, Collections, and Algorithms
Chapter 11	Exception Handling and Logging
Chapter 12	Inner Classes
Chapter 13	Streams

Day 5

Chapter 14	Working with Files
Chapter 15	Advanced Stream Techniques
Chapter 16	Object Serialization
Chapter 17	Automated Unit Testing with JUnit

The above timeline should play out approximately if all chapters are covered and all labs (but not ones marked optional) are covered in their suggested times or with a little extra time. If anything, most likely you will find you're a half-chapter ahead of this timeline by the end of day 1, and for the rest of the week. However, as discussed above, different audiences will require different paces. Treat Chapter 17 as optional, for starters. If time is short, the whole last day on streams can be dropped in favor of a slower pace over the first 12 chapters – in fact, this may be most appropriate for audiences coming from languages or environments very unlike Java, and should probably be decided prior to class time.



Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse Europa 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 documentation inside the workspace itself for general notes on usage: when you first open the workspace you'll see a **ReadMe.txt** that directs you to deeper, HTML-based documentation. 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.

See also the file **Java Module Notes.html** (available from within the IDE) for specific matters related to using Eclipse with this course.

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.





Teaching Notes

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 structured 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. It also goes deeper on enumerated types, and highlights how they can be used to make a code design more rigorously object-oriented by offering state elements and even their own methods.

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.

This is also one of the more challenging chapters because it introduces parameterized types in the form of Java generics. This adds a dimension to students' understanding of OO design, and the raw syntax of generics can be difficult to absorb. We stay away from any exercises in implementing generic types or methods, and stick to basic use of generics with a focus on the Collections API, which tends to help crystallize the idea of generics as students practice with it. You may choose to go much deeper into this topic, if the class has the interest and experience for it.

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 sorted sets, as well as algorithmic programming using iterators. The final bit on the Collections utility is highly recommended; knowing that these utilities are available will simplify students' lives considerably as they start doing practical Java coding.

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. Logging is another good practice that we want to encourage early, and goes hand-in-hand with exception handling, so they've been combined here. Lab work in this chapter is relatively light, just enough to see the basic mechanisms 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. Try the new page on use of inner classes in GUI programming as a place to make a stand about why inner classes are good things. Anonymous and method classes are especially hard features to defend, even though they do have their place in practical solutions.

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

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.

Chapter 17

This chapter is optional, but if there's time it is recommended. Unit-testing is another good practice that, like exception-handling and logging, we'd like to inculcate during this initial class on Java programming. Most students find this chapter a fun way to finish up. It's also an interesting foray into the more liberal use of code annotations that's become common to so many Java EE APIs and other Java products these days. Up to this point we've only seen `@Override` and `@SuppressWarnings`; JUnit's use of `@Test`, `@Before`, and so on is a different ballgame, and students might find this intriguing.

Appendix B

If there is still time, or if students have a special interest, this appendix on code compatibility and migration strategies may be worth covering. It's difficult to simulate all the possible combinations of old and new code, using various features, that might occur in practice, but the ones covered here are (a) the biggest issues, typically, and (b) illustrative of the general idea that new language features in 5.0 have been implemented primarily in the compilation process, so that effect on runtime behavior, and hence compatibility, is limited.



Revision History

Revision 6.0 updates the course for Java 6. Actually, most Java-6 features are out of scope for this basic programming course, but we do support the 6.0 JDK, and there are a few other interesting improvements:

- We catch up on Java 5 a little bit by mentioning Java annotations. This has been shuffled off to our advanced-Java course until now, but at this point it seems that annotations are so prevalent that some familiarity with the syntax is essential. Also, we now cover the `@Override` and `@SuppressWarnings` annotations, in Chapter 7 and Appendix B respectively.
- Chapter 17 has been revised more heavily, as we've moved from JUnit 3 to JUnit 4. This too brings on more use of annotations, and the GUI-based test runners are gone, as we use the simpler **JUnitCore** façade. If you're using the Eclipse overlay, note that Europa integrates JUnit 4 as well, and the appropriate projects have been configured to support this.

Revision 5.0.2 is a maintenance revision, with several minor fixes and enhancements, and eliminates errata accumulated over offerings of 5.0.1.

Revision 5.0.1 is a maintenance revision, with several minor fixes and enhancements – mostly for the sake of clarity. A few more significant changes are below:

- We had no immediate exercise in the for-each loop in Chapter 4. Short sections have been added to each of Labs 4A and 4B for this purpose.
- Added short discussion of break and continue vs. structured programming to Chapter 4 - hoping to make students aware of potential issues with this coding practice without preaching any particular philosophy.
- New example of static imports in Chapter 8.
- Converted DNA example to use `StringBuilder` rather than `StringBuffer` in Chapter 8.
- Fixed maintenance-induced bug in PigLatin code by which the project wouldn't build - it had been missing an import statement.
- DOS build and run scripts now run from any working directory, not just from their own. This is mostly for the convenience of Eclipse and other IDE users, because the scripts can now be invoked from an IDE project tree, if one desires to build and/or test from the command line for the sake of comparison with the IDE build and test.



Revision 5.0 overhauls the course for Java 5.0. Major changes include:

- New coverage of native enumerated types: introduced in Chapter 3 and then studied in depth in Chapter 8.
- New coverage of varargs.
- New coverage of generics, in Chapter 10, which is now called "Generics, Collections, and Algorithms."
- Removed chapters on threading and reflection - partly to make room for new material and partly because these are generally considered advanced features and many classrooms were skipping them anyway. Look for these in a new version of our Advanced Java course soon.
- New coverage of the J2SE logging API, added to Chapter 11 on exceptions.
- New final chapter on testing with JUnit.
- Chapter 6 on Classes and Objects has been reorganized a bit, to get better flow and especially to get to some lab work earlier. Also the first lab has been broken up into stages, so the whole chapter is a bit less of a brick.

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.

Revision 1.4.1 is a maintenance release, with various fixes for book and code.

Revision 1.4 reorganizes the course in several major ways, the most important of which is a consolidation of the previous five-module structure into a single, integrated course.

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.





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.





Errata

The following issues have been reported since the latest course release. These will be addressed in the next revision.

- In Chapter 6, on page 194, the example of name collisions in imports is incorrect. If these were two wildcard imports it would be correct, but single-type imports are given precedence over wildcard imports, so the compiler would resolve **List** successfully (though probably not correctly, and there'd be other problems).
- In Chapter 17, on page 503, the code listing should include the **@Test** annotation on the method.





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.

