Capstone Courseware, LLC
33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Intermediate Java Programming

Version 5.0.2

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 designed as an alternative to Course 103, Java Programming, and is intended for students who have already developed a strong understanding of Java as a structured language but who are unfamiliar with it's object-oriented nature. Our primary intent here is to make possible a training path that begins with Course 102, Introduction to Java Programming, thus taking the introductory-level student to the same ending state as Course 103 would do, but in two weeks instead of one and letting Course 102 develop some of the basic concepts over a longer stretch of time.

To this end, the first chapter of this course has been developed as a sort of speed-through of the first four chapters of Course 103. For some audiences this material will not need to be covered at all; for some it will make sense to work through it over a couple hours at the beginning of the week, perhaps taking one of the labs as a refresher on procedural Java. (If a group seems to need more than a couple hours on that first chapter, they would probably do well to take Course 102 whole before moving into this course; that not being possible once you've started your class, consider re-setting expectations for the week and slowing everything down considerably.)

Timeline

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

Day 1	
Chapter 1	Review of Java Fundamentals
Chapter 2	Object-Oriented Software
Chapter 3	Classes and Objects (runs to Day 2)
Day 2	
Chapter 3	Classes and Objects
Chapter 4	Inheritance and Polymorphism
Chapter 5	Using Classes Effectively
Day 3	
Chapter 6	Interfaces and Abstract Classes
Chapter 7	Generics, Collections, and Algorithms
Chapter 8	Exception Handling and Logging
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
Chapter 14	Automated Unit Testing with JUnit

Eclipse Overlays

Capstone Courseware 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 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\JavaInter\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.

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

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 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 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 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 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. 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 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. 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 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.

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 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. 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 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. 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 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

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 14

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.

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 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:

- New example of static imports in Chapter 5.
- Converted DNA example to use StringBuilder rather than StringBuffer in Chapter 5.
- 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 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.

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

Since the latest release of the course, the following issues have been discovered. These will be corrected in the next release.

- Chapter 14, page 433: the correct working directory for this demo is **Demos\JUnit**.

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.