

Migrating to Java 6

Version 6

Instructor's Guide

Overview

This course is intended to give intermediate-level Java-1.4 programmers a digest of what's new in Java 5 and Java 6, and how to program effectively using the new language features. The pace is fast, relative to our other Java programming courses, because we try to stay focused on only the pieces that are new for that audience. For example, we don't teach the Collections API; we show how it's affected by generics. Some features are completely new (enums, annotations, etc.), and these get heavy treatment, but even there we're pitching this to a target group that, for example, has done enough with Java to know the frustrations of using a pile of static final fields instead of a proper enum, and so hopefully will have an immediate appreciation for native enumerations. And similarly with other features.

The course does focus on language features, and does not put a lot into studying new APIs or packages. In the second module, we start to lean over that line a bit, as we do work with the JAXB and JAX-WS; but, unlike many new packages in Java 5 and 6, these two also involve tools built into the JDK and code-generation, which affects the build process itself. Contrast something like the concurrency API, which is lovely to have but isn't so fundamental as language features or tools.

Timeline

Day 1

2½ hours	Module 1, Chapter 1
2 hours	Module 1, Chapter 2
2½ hours	Module 1, Chapter 3

Day 2

1½ hours	Module 1, Chapter 4
1 hour	Module 1, Chapter 5
2 hours	Module 2, Chapter 1
2 hours	Module 2, Chapter 2

There is intentionally more material in the book and lab set than students could cover in two days. Most audiences will not be deeply fascinated with all the dark corners of every feature we cover here – especially, we go fairly deep on enumerations, generics, and annotations and pluggable annotation processors. Times shown above represent a general “squeeze” on all the chapters to fit two days, but the best delivery will usually be one in which one or two of the chapters mentioned above are done in their full detail, with all the demos and labs, and where otherwise some labs are skipped and some sections covered lightly. This will suit most groups just fine, and it’s usually just a matter of discovering where their interests lie.

Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse Galileo 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 thoroughly with the core course material, 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, and for those interested in Java EE development using Eclipse.

While the coursebook contains no references to Eclipse and no IDE-specific demo or lab instructions, it is generally intuitive to map the coursebook instructions (“build,” “run,” “deploy,” etc.) to the appropriate actions in the IDE. Since the courseware is written to work with or without the IDE overlays, there are some things that are not as intuitive, and we’ve documented those in a set of files bundled into the workspace itself. Follow the pointers given in `c:\Capstone\Java6\Eclipse\ReadMe.html` (which appears front-and-center when the workspace is first opened) to find notes on how to use the Eclipse overlays for Capstone courses generally, and on the particulars of this workspace for this course. Please read through those documents in detail before jumping into using the workspace in class, and 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, this IDE layer of the courseware is an optional piece, and while we’ve taken care to test it fully, we can only offer complete technical support on the standard course. If a given exercise is giving trouble, please (a) check the troubleshooting notes both in the workspace itself and in this instructor’s guide, and (b) be certain to build and run it from the command line, using the simpler and more predictable command-line builds as prescribed in the coursebook, before contacting Capstone.

Teaching Notes – Module 1

Chapter 1

After a quick overview of the whole course, this chapter gets right down to brass tacks, with sections on for loops, varargs, formatting, and static imports. Each of these should be welcome news to the Java-1.4 developer, and each is pretty easy to grasp. So this chapter should hum right along.

The theme here is ease of use, and practically speaking one of the most effective ease-of-use enhancements to the language is missing: I'm thinking of auto-boxing. But it doesn't make a lot of sense to cover that now, as the potency of that feature will be much clearer in the context of generics.

Chapter 2

In a pre-release version of this course, enums were rolled into Chapter 1. But there's really so much there, and it seems to make more sense to give them some room to breathe. The basic usage could be covered in about five minutes. But the ins and outs do take a little longer, and taking full advantage of the stateful and behavior features likewise.

Not everyone will be convinced of the goodness of stateful and especially behavioral enumerated values. In fact this is the first of several points during the course at which you're somewhat likely to encounter skepticism of Java 5: after all, even if it seems like old hat today, this was an ambitious and sweeping set of changes to a language that many students are comfortable with as it was. It ain't broke, so don't fix it, is a sentiment I've heard more than once, especially when showing off some of the more startling and tricky pieces of syntax (nested generics, wildcards, annotations within annotations, ...).

It's usually a good idea to establish a dividing line between usage that's simple and effective, and the more geeky stuff that may indeed have a place but can also be set aside without compromising the whole value proposition of Java 5. In other words, give the skeptics less to shoot at. Then, let those discussions roll, and have some fun with the questions of why a given feature was needed, why it looks so darned strange, etc. – without worrying that the underlying question might be, “why are we studying this at all?” There's plenty of obviously good stuff here, we can at least rely on that.

Chapter 3

Generics get a great deal of press when talking about Java 5, and there is an awful lot to this feature of Java. It's another area in which the distance between simple, effective usage (such as basic parameterized collections – a list of strings, a set of integers) and what you can conceivably do (generic types parameterized by their own anticipated subtypes, wildcards, flag parameters, etc.) is vast. More than the other chapters, this is the one where it may make the most sense to just leave some of the later sections out – partly hedging against skepticism here, but also just guarding against outright fatigue. There are many facets to generics, and by the time you've covered basic usage, wildcards and why they exist, and generic methods, students heads may be spinning a bit. We've tried to chop this chapter up with lots of little exercises, and then a couple of longer but not too challenging labs, to let the syntax and the concepts sink in.

One extra note that you might give, after the `MultiIterator` example, is that any iterator can easily be wrapped in a generic type that provides the `Iterable<E>` functionality. I've done this in a few applications, calling the class `Loopable<E>`, and it really is just a wrapper around any delegate `Iterator<E>` that implements the `iterator` method to hand over that iterator. Then we can say

```
for (MyClass x : new Loopable<MyClass> (myIterator)) { ... }
```

Chapter 4

Annotations are one of the less well-understood features of Java 5. 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. If your group knows J2EE 1.4 at all, even the quick sampler of Java-EE-5 annotations at the end of the chapter should really punch it up and give it a more concrete feel.

The lab is necessarily hypothetical, but it should give students some sense of the extensibility that annotations bring to the type model. To do anything more realistic we'd have to get into building annotation processors – or we'd have to start bringing Java EE servers into play, and that seems too heavyweight for this course.

Chapter 5

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.

Teaching Notes – Module 2

Chapter 1

This chapter really helps to make the annotations chapter in the previous module feel more real, because we're starting to do some more practical things with annotations, and not just talking about their syntax and some common uses. Most coding exercises in this chapter are extensions of the lab from that earlier chapter. It also foreshadows the next chapter, and we start bringing the whole annotations story together as we start to see applications such as JAXB and JAX-WS, and it's pretty clear without even knowing how those tools are built that they function as standard annotation processors.

Unless this is one of the chapters on which you've decided to "go deep," as discussed above in the Timeline section, a good way to make this one more manageable is to treat the final demonstration as a lab exercise, and the lab as an optional final code example. The demo is pretty chewy all by itself; and the lab is pretty challenging – straightforward in terms of technique but still it's a lot of complexity to ask students to wade through.

Chapter 2

As mentioned in the overview, the topics here are more like APIs than fundamental features of the sort we've been studying up to now. But they do involve command-line tools that are baked into the JDK, and so they're certainly not purely utility APIs, but somewhere in-between. And, as mentioned above, the segue from the previous chapter should be satisfying, as JAXB and JAX-WS both serve as further (standards-based) examples of the sort of processing we do in that prior chapter.

Both APIs seem worth some time to cover, and yet, without taking days to study them, we have little choice but to scratch the surface, probably leave students with a lot of questions, and move on. The aim is that they will at least have seen JAXB and JAX-WS in action, and understand their role in Java SE and Java EE to a level of detail deeper than if they'd just had a discussion and seen non-working examples.

The JAXB section pointedly leaves a lot out in the interests of time. One big piece that does appear in the JAX-WS demo later, and is not explained, is the use of custom binding files to tweak the mappings from XML to Java. In most presentations this won't show, and students won't ask. But it is there, and you might want to look over the bindings files to get familiar with this part of the code before presenting this chapter. This is more fully explained in Course 561 (whose longer Restaurant case study is the basis for the shorter version we use here), and if you would like some material for further reading or for in-class presentation, please contact Capstone and we'll be happy to get it to you.

The final demo on web services brings a lot of new infrastructure to the lab setup; in fact, without it, the lab installer would be about 8% of its actual size. This is because we start to involve Ant, Tomcat, and the JAX-WS RI – as mentioned in the coursebook. The service side is necessarily opaque; we don't have nearly enough time to dive into how that was built, how it's hosted, etc. So we've made that a simple cookbook, and we spend a little more time (not much) on the client side.

Revision History

Revision 6.0 adds a module to the course focused on Java 6, leaving the Java-5 module exactly as it was in the previous release.

Revision 5.0 is the initial release of the course. For a couple years it was available in a pre-release form as Course Java_0008. This version is built almost from scratch, drawing some of the same sections from other courses but bending them to a coherent storyline; and several new exercises have been added just for the what's-new audience, as opposed to the students who need to learn Java itself at the 1.5 level.

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.

There really aren't too many other things that can go wrong with this one. If the executable path brings up the Java compiler and launcher for an instance of JDK 5, you should be golden. The final chapter does bring some new environmental challenges, so that's when certain things will break if environment variables aren't right. Ant builds will fail if **CC_MODULE** or **JAVA_HOME** aren't set correctly.

Tomcat is usually pretty resilient, but one thing we run into now and again is a latent **CATALINA_HOME** setting. Tomcat 6 doesn't need this variable to be set; but if it is, it will change the way the server starts. Often the result will be that the server will start, but it will start with a different home directory and then your WAR deployment will be ineffective, and you'll get 404 errors when testing. Remove the **CATALINA_HOME** variable (or, in DOS, set it to nothing), and restart, and things should clean up.

Errata

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

- No errata at this time.

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.