

The Java Message Service

Version 1.1

Instructor's Guide

Overview

JMS does not place too many conceptual hurdles in the student's path. Of the various J2EE technologies – thinking here of servlets, JSP, EJB, and web services especially – it is perhaps the simplest and easiest for a Java (SE) programmer to grasp quickly. There is no container/contained-object relationship to learn, no tricky lifecycle or context issues. It's really pretty straightforward technology, and most of the hard stuff is in the service implementation.

The flow of this course is similarly un-challenging. About the only thing that students may get curious about before the course is really ready to teach it is reliability: what does that mean, and how does JMS guarantee it? You may need to counsel patience a bit on this subject, but otherwise you should find that the information and the exercises build up in a nice linear way, each one adding a bit more to the overall picture.

Timeline

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

Day 1

3 hours	Chapter 1
1 hour	Chapter 2
2.5 hours	Chapter 3

Unless students are all very sharp Java programmers, taking all six labs in this course will probably run you overtime. Though none are marked as optional, you can be quick to skip Lab 2A. Conversely Labs 1B and 3B are probably the most high-value, and should definitely be covered. The other three are somewhere in the middle. If you don't cover Lab 1A, be sure to review the use of the **PointToPointClient** class, which is integral to just about every successive exercise.

Tools Deployed with the Lab Software

This course's software requires separate setups of a Java SDK, the Crimson text editor, and/or Eclipse 3.x; otherwise it is self-contained, as the lab installer sets up not only the lab software but necessary tools: in this case just a small RDBMS, which is Apache Derby.

Derby lives in **c:\Capstone\Tools\Derby10.0** and is used entirely as JARs in the classpath. The builds for the Energy case study include Derby in the runtime classpath for the Processor component, and the **SetUp** script automates the creation of a Derby database along with the creation of the three necessary queues for the case study. The only pitfall to know with Derby is that as we are using it in embedded mode, it allows only one connection at a time. This should not be an issue at all in the normal run of the case study. The only issue would be if two Processor components were running at the same time – which would be a bad idea anyway since they'd be consuming messages from the same queue – or if you tried to re-create the database while the Processor were running – so don't!

Ant Build Process

Though most students will be happy to let the **ant** command take care of things, some students (and most instructors) will want to understand the inner workings here a little better. Each project in the course has its own **build.xml** and **build.properties** files; these rely on the master environment variable **CC_MODULE** to import targets and properties in a central directory, **%CC_MODULE%\Ant** which is typically **c:\Capstone\JMS\Ant**. Information here defines a routine for building a Hibernate application.

For the most part, the Ant build just compiles Java classes and makes sure that the J2EE library is in the class path, so that JNDI and JMS types are found. The various provided scripts simplifies the launching of a target application class by wrapping a call to **ant** with the target **run** and application class and command-line arguments defined as properties. Ant targets are also behind the scripts to create and remove queues and topics from the running J2EE application server.

Eclipse Overlays

Capstone Courseware provides an optional package of workspace and project files for Eclipse 3.2 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\JMS\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.

Notes on the overlay for this course:

- The source files in this course's projects are in a subdirectory **src** of the main project directory. To make Eclipse happy in attaching to this directory structure, we need the **JavaSource** folder that you see in each Eclipse project – this attaches to the **src** directory. But then, to see the remaining resources in the exercise, we need a second folder that attaches to the root directory of the exercise: this is the **Resources** folder. This is not intuitive, and students will need some guidance. Especially, the **Resources** folder is important because it gives access to the **build.xml** which can and should be used to build and deploy the project using Ant.
- The **J2EE** and **Derby** folders are only there to hold JARs which are then placed in the classpath.

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 chapter establishes the role of JMS and the basic concepts of connections and sessions, queues and topics. A few simple examples illustrate the basic behaviors of each, and there are some interesting contrasts to draw between synchronous and asynchronous message consumption, and between queues and topics. As usual, the first example is also designed to prove out the environment, and so often this one will require a little extra time to stamp out any setup problems: missing tools, mistyped executable path, missing environment variables, etc. Other than creating a few more queues, topics, and factories, there's nothing that's needed for any of the later exercises and isn't checked here.

Chapter 2

This chapter is very short and adds some tools to the toolbox: polymorphism under the main message interface, message headers, properties, and message selectors. Again, Lab 2A is probably the least important one in the course, and it's usually most sensible to skip it or just review the answer code.

Chapter 3

What conceptual challenges there are in JMS are all about reliability. Acknowledgement and redelivery are simple enough, although it can be a surprise to find out that you have to manually recover the session in some situations but not others. It's the matter of transactions that has the most potential to throw students for a loop. They're really very simple beasts, but if students are at all accustomed to database transactions and propagation of transaction contexts between objects, clients and servers, etc., it can be a hard assumption to break. Students may well figure that a message send will block waiting for the receiver to succeed with transactional operations – which is of course impossible, it would break the whole idea of asynchronous messaging. But if that's not it, then what does a JMS transaction really do? It can be surprising, and even a letdown, to find out that the only real impact of a rollback is failure to acknowledge and refusal to send. But this is a powerful tool, and hopefully the two case studies – one demo and one lab – will prove that.

Revision History

Version 1.1 is the initial release of the 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

At this time there are no errata for the course.

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.