



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

Overview of Java EE Development

Version 5.0.1

Instructor's Guide



Overview

This course is a bit of a departure from our other Java and Java EE courses, in that it is intended first and foremost for non-developers. A secondary audience that has been well served by this course material consists of experienced developers from outside the Java space who are migrating to Java. Experienced Java programmers may well find this material interesting, and it can be positioned as a lead-off for our technology-driven courses in servlets, JSP, EJB, web services, etc. – but it will not hold the interest of that audience for a full day. See the Timelines section below.

So we're painting with a broad brush, and one benefit is that we can choose to emphasize the inventions that make Java EE what it is. It should be fun for any of the above audiences to focus their attention on big concepts like portability, why containers are important, roles, etc.

While the course does treat Java EE at the highest conceptual level, it trades depth for breadth to some extent. The instructor will have to be well versed in Java, JDBC, web application, JSF, EJB, JMS, and web services development, at least, in order to present effectively.

Because the audience is not expected to have a lot of interest in code, the examples are treated lightly in the book. We show a little code for each major component type or other type of artifact: a `@Stateless` annotation here, a URL mapping there, etc. But the primary audience will probably want to keep the introductory diagrams in view and track through them, sort of like checking the map to see where they are.

It's important for the instructor to learn these applications to some depth, since whatever questions people do have, the answers won't be in the student guide. See the Example Applications section later in this guide for instructions to get up and running with each example. You should find all the examples to be fairly self-explanatory if you are familiar with the technologies that they illustrate. The examples are deployable to Sun's AS as written; if the instructor prefers to illustrate a different Java EE server, some customization will be necessary.





Timeline

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

1 hour	Chapter 1
1 hour	Chapter 2
3½ hours	Chapter 3
½ hour	Chapter 4
1 hour	Chapter 5

Chapter 3 is the big, dense middle of this course, with discussions, diagrams, and concrete examples of all the most popular Java EE technology. It's good to let Chapters 1 and 2 breathe a little – no need to rush them, as they are pretty short. Then, don't worry if Chapter 3 is running you pretty close to your shutdown time. Chapters 4 and 5 are each very short, and are much easier to summarize if that becomes necessary.





Example Applications

In the coursebook, each example is identified in general terms, such as “a dating service.” Specific paths to examples are not provided, since these examples are only meant to be installed on the instructor’s PC and demonstrated from the front of the room. We do give specific filenames and paths, because that’s of some interest as part of studying how a particular technology works; for example one is not really so literate in servlets development if one doesn’t know that the deployment descriptor is named **web.xml**.

So code-review details are provided, but build instructions are missing, except for the Hello example where we actually want to study the concepts of compiler and launcher. This section provides specific instructions for building each example. (You may want to carry these all out before class, in fact. Students don’t do these steps, and you won’t ever need to edit and rebuilt an application.)

Environment

Before class begins, or just before Chapter 3, you need to set some environment variables; start the MySQL database server; and start the GlassFish server.

Environment is as follows:

```
set JAVA_HOME appropriately, as described in the setup guide
set JAVA_EE_HOME appropriately, as described in the setup guide
PATH=%JAVA_HOME%\bin;%JAVA_EE_HOME%\bin;c:\Capstone\JavaEEOverview\Ant
set CC_MODULE=c:\Capstone\JavaEEOverview
```

To start MySQL, open a DOS console in **c:\Capstone\Tools\MySQL5.0\bin**. Run **startup**. You’ll see a banner explaining that this is the MySQL daemon process, listening on port 3306. Leave this console alone for the day; it is the RDBMS, and it will host databases for a few applications.

Copy the file **c:\Capstone\Tools\MySQL5.0\lib\mysql-connector-java-5.0.4-bin.jar** to the directory **c:\Sun\SDK\lib**. This will assure that the JDBC driver is available when called for by certain applications running in GlassFish.

Start GlassFish by opening a DOS console with the above environment in **c:\Capstone\JavaEEOverview\Examples\LoveIsBlind**, and running **ant start-server**. You’ll see verbose output as the server starts up over a minute or so, and then “server startup complete.” You can stop the server by running **ant stop-server** from another console, with the same working directory or any other project directory (one with a **build.xml**).



Finally, note that one application in particular seems to give Sun AS real memory problems. The Flights application uses RichFaces, which in turn uses a code-generation tool called CGLIB, and that library is known to drain the PermGen memory space quickly. We haven't seen this with the latest version of the course, but it's been a problem in the past. So it's a good idea to undeploy this application when done – see instructions below – and you may need to re-start GlassFish at some point during the day. If you notice the server responding very sluggishly to web requests or deploy/undeploy commands, this is usually the reason, and soon after you notice this, the server may freeze completely. Stop and start using the Ant targets described above; or you may find that this is ineffective and you actually have to go and kill the Java process from the Windows Task Manager.

Chapter 1 – Hello, Java! (Hello)

Examples/Hello. This one is dead-simple and actually described in detail in the coursebook.

Chapter 1 – A Java GUI Application (WordGame)

Examples/WordGame. Just **build** and **run** this one, like the previous example.

Chapter 2 – EJB Metadata (LandUse)

See Chapter 3, A Modern Web Application, as this is the same application, just shown for different reasons.





Chapter 3 – A Classic Web Application (LoveIsBlind)

Examples/LoveIsBlind. From this point forward all applications build and deploy with **ant**. But each has different setup requirements. This application needs a database, so run these commands in sequence:

```
ant init-DB
ant
```

Then visit the application at the URL shown in the coursebook.

Chapter 3 – A Modern Web Application (LandUse)

Examples/LandUse. This is an example of JSF and EJB3. It is a little more intricate to set up, because it needs a database but the DB script only sets up the tables; then a Java application uses JPA to populate the DB. (It's an interesting point to raise during this example, showing that JPA can indeed be used separately from EJB and an EJB container.)

To get this one up and running, do the following:

```
ant init-DB
ant build
run PrimeWithData
run ListAll
ant deploy
```

The ListAll application is just for your own confirmation that the DB now has data in it; PrimeWithData can't be skipped.





Chapter 3 – An Ajax Application (Flights)

Examples/Flights. This one's easy to deploy – just run **ant** and visit the URL shown in the coursebook. But one additional trick might be worthwhile. From **Examples/HTTPSniffer**, in a separate console, **build** and **sniff**. This DOS window will now show all HTTP traffic coming into port 8079, while forwarding it automatically to port 8080. You can then visit **http://localhost:8079/Flights**. The same things will happen in the browser (a bit more slowly), and you'll see everything over the wire. You can do this for any of the web applications, actually, but it's probably most interesting here, because you can highlight the difference between full-page requests and Ajax requests.

As mentioned earlier, it's a good idea to **ant undeploy** this application before moving on. Sun AS has trouble more quickly than most servers with the CGLIB bug that drains the PermGen memory space.

Chapter 3 – A JMS Application (Auction)

Examples/Auction. There are no code listings for this one, as it seems especially hard to do just a little JMS and not dig all the way into how the API works. But the runtime behaviors tell a pretty good story. Most of the steps are spelled out in the coursebook. Beforehand, just run **ant**. Also not shown, but a good idea, is to run **tearDown** afterwards; this clears the JMS queues and connection factories from the server.





Chapter 3 – A Web Service (Tracking)

Examples/Tracking. Build the service from the **Service** directory by running **ant**; then build the client from the **Client** directory, again with **ant**. Nothing very tricky. If there's interest, you might enhance this demonstration in few couple ways. Run the HTTPSniffer mentioned earlier, and then

```
run check 2 http://localhost:8079/Tracking/Track
```

This will push the SOAP traffic through the sniffer console, so you can show students the real-time interactions and inspect the SOAP. Nice to make the point that all the interactions over the wire are XML text.

You can also **build** and **run** from **Examples/SOAPPad**. Use the GUI to direct a SOAP request to the right URL: "localhost", "8080", and "/Tracking/Track", with SOAP content copied from **Examples/Tracking/SOAP/Request.txt** and pasted into the request text area. This really drives home the point that anyone who can put XML out in an HTTP request can participate in web services – and that could be software written in Java or some other language, or, in odd cases, interactive tools like this one, maybe for debugging purposes.

Finally, you might spend a little extra time with the XML Schema and WSDL descriptor.

Chapter 4 – How Portable is Portable? (LevelsBlind)

See Chapter 3, A Classic Web Application, as this is the same example showing different files.





Teaching Notes

Chapter 1

As always, one must read one's audience when presenting history and background. Some groups eat it up, and some just really want to get to the details of what's going on right now. This course probably draws a lot more of the former type of participant, though, and offerings to date have borne this out. I've found that placing Java in the context of languages that come before it is a great way to explain what's unique about it, and this in turn provides a natural segue into the evolving role of Java EE.

Chapter 2

The following chapter is really the heart of the course, but this chapter sets up some key concepts that will inform most or all of the technology that's covered in detail in that next chapter. The trick is to speak compellingly about these big ideas before we've had a chance to dig into a specific API or a specific example. The most reliable part of the chapter, in this respect, is the container/component concept: there are a lot of specifics to convey, even if they are architectural and not implementation details; and the features pages toward the end of the chapter (remoteness, scalability, availability, etc.), illustrate the Java EE value proposition and make the case for containers. There's also a final, concrete example – actually sort of a preview of an example that's covered more fully in Chapter 3, but here it's meant to illustrate more clearly the distinction between annotations and XML for metadata.





Chapter 3

Now it's time to walk through the major pieces of the typical Java EE application. Each technology is presented in summary form, and most of these diagrams could account for five minutes of lecture time or thirty, depending on student interest.

One issue with this chapter is that, given the state of the art right now, we really have to cover both the old and new ways of doing things – especially in the area of web applications. By contrast, we don't touch EJB2 at all. But we can't really leave out servlets and JSP, nor can we ignore JSF – both are in common practice right now. So, we have little choice but to cover, in sequence: JDBC, servlets, JSP, JNDI, JSF, EJB, JPA, Ajax, JMS, and JAX-WS/JAXB. There's a real risk here that fatigue will set in, even for the more interested participants.

To address this, we've broken the chapter into three sections – though this division may not be obvious from the coursebook, which is why we mention it here. Try to consider what we're loosely terming "classic" web applications; then "modern" applications w/ JSF and EJB3; then JMS and web services as stragglers. The two big web-app examples are positioned to make this a natural partitioning. It's a good idea to set this up for students at the top of the chapter, and even manage the clock so that they're taking breaks between the sections. It should help keep everyone's head clear, so that JSF isn't blurring with JSP, JPA with JDBC, etc.

If there's time, remember to try the "sniffer" application as suggested in the Example Applications section for the Ajax and web-service examples. It takes a few more minutes, but participants always find it engaging to watch the HTTP traffic in real time.





Chapter 4

In this chapter we catch up on some real-world issues. The previous two chapters are practical enough, but they illustrate textbook designs and implementations for the reference implementation server. In this chapter we connect that theory to information about commonly used tools. Also it's important to explain what the portability promise of Java EE really is; especially in light of Java SE some people are surprised that the "100% Pure Java" mantra doesn't really apply to Java EE, or not as they expected.

One simple metric you might offer is this: to create a variant of this course for another server (WebLogic 10 as it happens), we ported all the working examples that they've seen at this point, and this took one experienced developer about 4 hours, including initial testing. This should reassure anyone who needs reassuring that the non-portable parts are small and that practically speaking the portability of Java EE apps is of real value, even if it doesn't cover 100% of all artifacts.

Chapter 5

Often there will be little time left at the end of the day, and this chapter can be skipped completely if necessary, without compromising the value of the training; the other chapters are compelling without this one. If possible, do try to get to the part about the formation of WARs, EJB JARs, and EARs, and the deployment process. It's quick to cover and fairly high-value, even to those students who would yawn their way through the administration-oriented stuff at the very end.





Revision History

Version 5.0.1 updates the course to use the Java EE 5 SDK, Update 7, and GlassFish 2.1. The code changes were trivial and all in the **Ant** directory: all the example code is untouched.

Version 5.0 updates the course to Java EE 5. Specific changes:

- Added discussion of AOP and annotations to Chapter 2.
- Added coverage of JSF, EJB3, and JPA to Chapter 3.
- Added a quick section on Ajax to Chapter 3.
- Added a concrete example on JMS, which had been a sort of 2nd-class citizen before, with only a brief writeup and no working code.
- Expanded the web-service example a bit, showing the key metadata that makes it fly as a JAX-WS service.
- Cut the case-study stuff at the beginning of Chapter 5. This was partly to keep the course in a reasonable one-day timeline, as we'd added so much to Chapter 3. This part never seemed all that compelling, and the remainder of the chapter lives just fine without it.

Version 1.4 is the initial release of the course, covering J2EE 1.4 practice.





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.

