SOFTENG 325 Software Architecture
Lab 3

Ian Warren

August 6, 2017

The purpose of this lab is to reinforce week 3's lecture material concerning further development with the JAX-RS framework, and to explore what's involved in writing DAO (data access object) classes for objects that are to be persisted in a relational database.

Tasks

Task 1: Build the JAX-RS Parolee project

Project `softeng325-lab3-parolee` is a complete project that implements the Parolee Web service discussed in lectures this week. It includes a domain model, a DTO (data transmission object) class for `Parolee`, and makes use of the JAXB (Java Architecture for XML Binding) framework for converting between Java objects and XML. `softeng325-lab3-parolee` is a multi-module project comprising two modules:

- `softeng325-lab3-parolee-domain-model`. This module implements the Parolee domain model. It also provides converters (`XmlAdapter`s) for third-party classes (external to the JDK) whose instances can't automatically be mapped to and from XML.
- `softeng325-lab3-parolee-web-service`. This implements the JAX-RS Web service, including the necessary `Application` subclass and resource class (`ParoleeResource`) that handles HTTP requests. In addition it contains the `Parolee` DTO and a mapping class `ParoleeMapper` to convert between `Parolee` domain objects and DTOs. Finally, this project includes an integration test class (`ParoleeWebServiceIT`) that exercises the Web service.

This project should be a useful resource as it illustrates how to use many aspects of the JAX-RS framework and API.

(a) Import the project

Import the project into your Eclipse workspace. As with the multi-module projects you worked with in Lab 1, import the *parent* project `softeng325-lab3-parolee-parent`. The parent project names the 2 modules, and these will be imported automatically.

(b) Build and run the project

Build and run the project by applying a suitable goal, e.g. `verify`, to `softeng325-lab3-parolee-parent`. The POM for `softeng325-lab3-parolee-web-service` is configured to build a WAR file, to run an embedded servlet container that hosts the Web service, and to run the integration tests – just like you did for Lab 2. The integrations tests should pass.

(c) Reflect on the project

Make sure you understand how the projects works. The main difference between this project and the Parolee JAX-RS project from Lab 2 is that this project leverages JAX-RS' capability to automate marshalling/unmarshalling to/from different data formats (in this case XML through MessageBodyReader/Writer implementations that use JAXB). In addition it illustrates the DTO concept.

You may want to refer to this project in the future when working on the main assignment. It shows additional HTTP message processing and how to work with generically-typed objects that are to be marshalled and unmarshalled.

Task 2: Develop the Concert Web service to automate XML marshalling

Further develop the Concert Web service from Lab 2 to allow clients to exchange both XML-based and Java Serialization representations of `Concert`s. Unlike the supplied Parolee project from Lab 2, where XML was hand-crafted, you want to leverage the JAX-RS framework to automate XML marshalling and unmarshalling.

(a) Modify the project artifacts

You simply need to work through the following steps:

- Create a copy of the Lab 2 Concert project, naming it `softeng325-lab3-concert`.
- Add the JAXB dependency to the project's POM file. The artifact you need is RESTEasy's `resteasy-jaxb-provider`. See the POM for `softeng325-lab3-parolee-web-service` – it necessarily includes the dependency.
- Modify the `@Produces`/`@Consumes` annotations in the `Resource` class to add the XML MIME type.
- In the integration test class, duplicate methods `testCreate()`, `testRetrieve()`, and `testRetrieveWithRange()` tests and change them so that they specify HTTP `Accept` and `Content` header values of `application/xml` as opposed to `application/java-serialization`.

In changing the `@Produces`/`@Consumes` annotations, you can specify them at the class level (rather than on individual methods) if you wish – they then apply to all methods in the class. Also, it's good practice to use MIME typed constants rather than string literals, e.g.

```
@Produces(javax.ws.rs.core.MediaType.APPLICATION_XML,
   SerializationMessageBodyReaderAndWriter.APPLICATION_JAVA_SERIALIZED_OBJECT)
```

(a) Build and run the project

Once you've amended the code, build and run the project. With the additional test cases, the integration test should demonstrate that the Web service offers both Java serialization and XML representations of resources. To see the XML in the HTTP message bodies, configure logging output for the namespace `org.apache.http` to `DEBUG` (see Lab 2).

(c) Reflect on the project

Reflect on what you've done. In particular, consider how the JAX-RS framework separates the concerns of resource representation from application logic. What would you need to do to add support for another data format, e.g. JSON? What quality attribute is promoted by the way that JAX-RS manages data formats?

Task 3: Review the JAXB project

Project `softeng325-lab3-jaxb` demonstrates more of JAXB than has been used in the Parolee project from Task 1. Since the Parolee project uses a DTO class, marshalling to and from XML is relatively simple. Project `softeng325-lab3-jaxb` includes an example involving two classes (`Course` and `Lecturer`) that are linked by a bidirectional association. It shows how cycles can be processed, using the `@XmlID` and `@XmlIDREF` annotations, when marshalling objects to XML and unmarshalling the XML to generate a copy of the original object graph. Project `softeng325-lab3-jaxb` also includes the Library example from the lecture notes, and, for interest, shows how an XML schema can be generated and used to validate an XML instance that describes a library of books.

Project `softeng325-lab3-jaxb` isn't examinable in itself. However, you should be aware of the issues with marshalling and unmarshalling XML, and the role of XML IDs and XML ID references (as discussed in lectures). Depending on how you tackle the main assignment, project `softeng325-lab3-jaxb` may be a useful resource to refer back to.

Task 4: Complete a DAO class to manage persistence of Concerts

Project `softeng325-lab3-database` implements a partially complete DAO (data access object) class. A DAO is one that enables instances of particular classes to be retrieved and stored in some form of persistent storage (e.g. a relational database). This project doesn't use any ORM technology – it implements the DAO "from scratch" with direct access to a relational database.

The project is a simple Maven project that includes the following key entities:

- Domain classes `Concert` and `Performer`. A `Concert` has a unique ID, a title, a date and one `Performer`. Each `Performer` has a unique ID, a name, image file and genre. A `Performer` can feature in many `Concert`s, hence there's a one-to-many relationship between `Performer` and `Concert`. The relationship is unidirectional, from `Concert` to `Performer`.
- Interface `ConcertDAO` and the associated `DAOException` class. `ConcertDAO` offers methods to store, retrieve and delete `Concert` instances.
- A `ConcertDAO` implementation named `JDBCConcertDAO`. This class implements `ConcertDAO` using Java's JDBC (Java Database Connectivity) API for working with relational databases.
- Class `ConcertDAOTest`, a unit test that tests the `JDBCConcertDAO` implementation.
- `db-init.sql`, a database initialisation script that includes SQL DDL and DML statements to create tables for `Concert` and `Performer`, and to populate the tables with data.

Similarly to the Web application projects using an embedded servlet container, this project uses an embedded database. As with an embedded container, an embedded database greatly simplifies testing as there's no need to have a separate database server up and running. At construction time, a `JDBCConcertDAO` object connects to an embedded database that persists data on the local file system. The database used in this case is H2.

(a) Complete class `JDBCConcertDAO`

`JDBCConcertDAO` is largely implemented, except for methods `getById()` and `getAll()`. Read the Javadoc comments for these methods, in the `ConcertDAO` interface, and implement them accordingly.

With knowledge of the relational database model, Java's JDBC API is quite intuitive and straightforward. Prior to making any database request, a connection is required. `JDBCConcertDAO`'s constructors acquire a connection to the H2 database. Once connected, the connection can be used to acquire `Statement` and `PreparedStatement` objects. Instances of these classes are used to represent SQL requests; a `Statement` is a literal request whereas a `PreparedStatement` includes

template parameters whose values can subsequently be set. A SQL query returns a `ResultSet` object that can be used to navigate through the rows returned by the query.

Study the source code for `JDBCConcertDAO` to familiarise yourself with how the other methods are implemented and how the JDBC API is used; proceed to implement the required two methods.

(b) Run the unit tests

Once you've completed `JDBCConcertDAO`, run the unit tests. Debug your implementation as necessary to pass the tests.

(c) Reflect on the DAO class

In reflecting on this task, consider:

- How useful is an implementation that conforms to the `ConcertDAO` interface specification? Would you reasonably expect a DAO for `Concert` to behave differently? If so, how?
- Which aspects of the paradigm mismatch, presented in lectures, are manifested in the `JDBCConcertDAO` class?
- What would be involved in developing a DAO for a different application?

Use of H2

Class `JDBCConcertDAO` configures the H2 database connection to ensure that any changes to the database are persisted to the local file system (the connection could have been configured such that the database exists only in memory, in which case its data would be lost once the JVM running the database shuts down). Having the data persisted on disk is convenient as H2 includes a console application that you can use to access the database used by `JDBCConcertDAO`. Using the H2 Console, you can interact with the database, e.g. to run queries and see their results.

The H2 Console runs in a Web browser. To run the application, use Windows' *Start* menu to navigate to the H2 folder, and select H2 Console. The default username and password for connecting is `sa` and `sa`.

When configured as described above, there can be at most one connection to the database. If you are using the H2 Console and are connected, you cannot run the test cases. When the `JDBCConcertDAO` object tries to connect to the database, a connection exception will be thrown:

```
Database may be already in use:  "Locked by another process".
```

You should first disconnect from the H2 console, by clicking the `Disconnect` button (at the top left of the H2 console). If you have disconnected but the exception is still thrown, you will need to forcefully close any lingering connections. To do this from the H2 Console, having clicked the `Disconnect` button, click on the `Preferences` link in the `Login` screen. Under `Active Sessions` click the `Shutdown` button.

The POM for project `softeng325-lab3-database` includes a dependency on the H2 library.

Resources

Useful resources for H2 include the H2 website:

```
http://www.h2database.com/html/main.html
```

From here, you can download the H2 Console for your own machines. The website also has useful information, e.g. the SQL grammar for H2, should you need it when specifying the queries for `JDBCConcertDAO`'s `getById()` and `getAll()` methods.

Assessment and submission

Run Maven's `clean` goal on the Task 2 and 4 projects to clear all generated code. Zip up the projects and upload the archive to the Assignment Drop Box (https://adb.auckland.ac.nz).

The submission deadline is 18:00 on Friday 18 August. Participating in this lab is worth 1% of your SOFTENG 325 mark.