Lab 1.2 – Using an Entity Class

Lab

- **Overview**: In this lab we will use the `EntityManager` to lookup `MusicItem` entities from data stored in a database
  - We provide a simple test program to do this
  - We will need to add JPA code into the main method
- **Objectives**: Use a JPA entity class to work with stored data
- **Builds on previous labs**: Lab 1.1
  - Continue working in your **Lab01.1** directory for this lab
- **Approximate Time**: 25-35 minutes

Notes:

- It's a stylistic choice as to where you put your JPA code
  - Generally, the entity classes will contain the annotations (unless you decide to do it with XML mapping files), but you still need to decide where entity manager code should go
- For now, we'll just put it in a test program – a simple program with a main method
  - In the future we'll show some other choices – Data Access Objects, EJB, etc

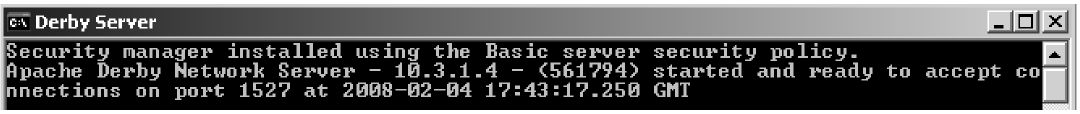
Setup / Create the Database

Lab

- We're using JavaDB – which is actually the open source Derby DB

Tasks to Perform

- Look at the scripts we provide to work with the Derby DB, in ***C:\StudentWork\JPA\Derby***
- Start the Derby database server
  - Execute ***dbStart.cmd*** (you can double-click on it)
  - This starts a standalone Derby server that can accept network connections to the database



```
Security manager installed using the Basic server security policy.
Apache Derby Network Server - 10.3.1.4 - (561794) started and ready to accept co
nnections on port 1527 at 2008-02-04 17:43:17.250 GMT
```

- Create the database
  - Execute ***dbCreate.cmd*** (you can double-click on it)
  - This creates the JavaTunesDB database
  - (you can ignore any DROP TABLE errors, if you see them)

Notes:

- JavaDB is included with Java 6
  - We include it in the setup for those not using Java 6
  - It is just the open source Derby database

- *dbCreate.cmd* generates DROP TABLE errors the first time it is run, because it drops the Item and other tables, which don't exist the first time, then creates them
  - The database for this course consists only of the Item table.
  - If you ever want a "fresh" database, simply rerun *dbCreate.cmd*.

- *dbSQL.cmd* launches the ij program , a command line SQL tool.
  - All commands in ij are terminated with a semicolon (`;`). For help, type `help;`. To exit, type `exit;`.

- To stop the database server, run the *dbStop.cmd* program
  - But leave it running for now

Using an SQL Command Tool

Lab

- **ij** is Derby's SQL command line tool
  - It allows us to create database objects and view and manipulate database data, without having to write code to do so
  - Most database packages provide such a tool

Tasks to Perform

- Execute ***`dbSQL.cmd`*** to run ij (you can double-click on it)
  - This command file is set up to connect to the correct DB
  - Using ij, you can browse the DB
  - Execute the query shown below to see the items in the DB

```
ij> select * from item;
    ... Lots of output ...
ij> exit;
```

Notes:

- All ij commands are terminated with a semicolon (`;`).

- Many ij commands require single quotes (`'`).

- For a list of commands, type `help;`.

- ij can connect to any database server if you know have the driver and know URL. You just need to specify the driver and hostname, as shown below for Derby and the JavaTunes DB

```
– DRIVER 'com.ibm.db2.jcc.DB2Driver';
– connect 'jdbc:derby:net://localhost:1527/JavaTunesDB'
      USER 'guest' PASSWORD 'password' ;
```

persistence.xml

Lab

Tasks to Perform

- Open *persistence.xml* - located in the ***META-INF*** directory
  - Look for the TODOs
  - Set the name of the persistence unit to ***javatunes***
  - Set the transaction type to ***RESOURCE_LOCAL***
- Note the Hibernate specific properties in *persistence.xml*, e.g.

```
<property name="hibernate.dialect"
    value="org.hibernate.dialect.DerbyDialect"/>
```

  - This particular property tells Hibernate that we are using the Derby database

- There are many other Hibernate Specific properties
  - The next slides cover what is available, and how to configure the database connection using them

Notes:

- We also have another property that turns on basic logging of the SQL that Hibernate uses

```
<property name="hibernate.show_sql" value="true"/>
```

  - We'll use this in a later lab to see how to optimize querying using fetch joins
  - For now, it's useful to see what SQL Hibernate is generating

Finish persistence.xml

Lab

Tasks to Perform

- Complete the other hibernate properties for a DB connection as shown below (look for the TODOs in the file)
  - You need to fill in the username, password, and the name of the database in the connection url (JavaTunesDB)

```
  <!-- Database Connection Settings -->
  <property name="hibernate.connection.username">guest</property>
  <property name="hibernate.connection.password">password</property>
  <property name="hibernate.connection.url">
     jdbc:derby://localhost:1527/JavaTunesDB</property>
  <property name="hibernate.connection.driver_class">
     org.apache.derby.jdbc.ClientDriver</property>

 <!-- SQL Dialect -->
  <property name="hibernate.dialect">
     org.hibernate.dialect.DerbyDialect</property>

 <!-- Other Hibernate specific configuration not shown -->
```

Notes:

- The first four property elements contain the necessary configuration for the JDBC connection
  - The dialect property element specifies the particular SQL variant Hibernate generates.
  - If you're working in a J2EE environment, you can specify a datasource with standard JPA elements, and don't need to use the Hibernate elements

- Since Hibernate is the persistence provider, it is the layer in your application which connects to this database, so it needs connection information
  - The connections are made through a JDBC connection pool, which we also have to configure
  - The Hibernate distribution contains several open source JDBC connection pooling tools, but this example uses the Hibernate built-in connection pool.
  - Note that you have to copy the required library into your classpath and use different connection pooling settings if you want to use a production-quality third party JDBC pooling software
  - [Hibernate 3.2.2 Reference Documentation, Section 1.2.3]

Finish the Test Class

Lab

Tasks to Perform

- Open *JPATest.java* (located in the package `com.javatunes.persist.client`)
  - We'll use this to write some fairly straightforward code to work with JPA, and we'll add complexity later
- Do the following in the main method
  - Create an `EntityManagerFactory` and `EntityManager`
    - You'll need to use the name of the persistence unit from *persistence.xml* (javatunes)
  - Use the `EntityManager.find()` method to get an instance of `MusicItem` using the id of 1, and print it out (you can just use `System.out.println()` with an item to output it)
  - See the earlier manual slides for examples
  - Don't forget to add in any imports

Notes:

- In the last lab, you put annotations on `MusicItem` to make it an entity class
  - In this lab, you'll write code to use the `EntityManager` and the `MusicItem` class go do to the database