Convert the persistence layer of a simple application to Java Persistence API (JPA)

What this exercise is about 1
Lab requirements 1
What you should be able to do 2
Introduction 2
Exercise instructions 2
Part 1: Create project and become familiar with the sample application 4
Part 2: Exchange plain JDBC model with JPA 6
Part 3: Create a new Embeddable class and use a one-to-one relationship 11
Part 4: Update Embeddable class to a many-to-one relationship 12
What you did in this exercise 13

What this exercise is about

This lab shows you how to convert the persistence layer of a simple application from plain JDBC to JPA. You will then extend the application to implement new JPA 2.0 features.

This lab is provided **AS-IS**, with no formal IBM support.

Lab requirements

List of system and software required to complete the lab. Although other versions may function properly, the specified versions were used to create and test this sample application.

- WebSphere Application Server Version 7.0
- WebSphere Application Server Feature Pack for OSGi Applications and Java Persistence API (JPA) 2.0
    - For this lab, only the thin client is needed. The thin client is a JAR file that, when included in the PATH variable, allows applications to use the provided JPA 2.0 functionality and extensions without the need to install and run the Application Server.
- Java (J2SE) V1.6 – included with WebSphere Application Server V7
- Eclipse Version 3.5 (other IDEs will work, but the instructions are specific to Eclipse)
- Apache Derby database – included with WebSphere Application Server V7
- The lab source code files

What you should be able to do

At the end of this lab you should be able to:

- Convert an existing application to use JPA.
- Create a new JPA application.
- Run a JPA 2.0 application.
- Identify how JPA creates tables based on one-to-one and one-to-many relationships.

Introduction

Object-relational persistence is a key developer requirement for many application developer scenarios. JPA is the Java EE standard for object-relational persistence and was first introduced as part of Java EE 5. As part of the Java EE 6 standards, JPA 2.0 (JSR-317) updates object-relational capabilities with important developer APIs and enhancements.

The WebSphere Application Server JPA implementation is based on Apache OpenJPA, a leading open source Java persistence framework. This feature pack provides the Apache OpenJPA 2.0 implementation with IBM enhancements to benefit integration with WebSphere Application Server. The Apache OpenJPA 2.0 implementation includes improvements and benefits over previous releases and even beyond the JPA 2.0 specification.

Useful links:

- OpenJPA JavaDoc: http://openjpa.apache.org/builds/2.0.0-beta2/apache-openjpa-2.0.0-beta2/docs/javadoc/index.html
- OpenJPA Properties (can be specified in the persistence.xml file): http://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/ref_guide_conf_openjpa.html
- Java EE 6 javax.persistence package JavaDoc (includes list of Annotations): http://java.sun.com/javaee/6/docs/api/javax/persistence/package-summary.html

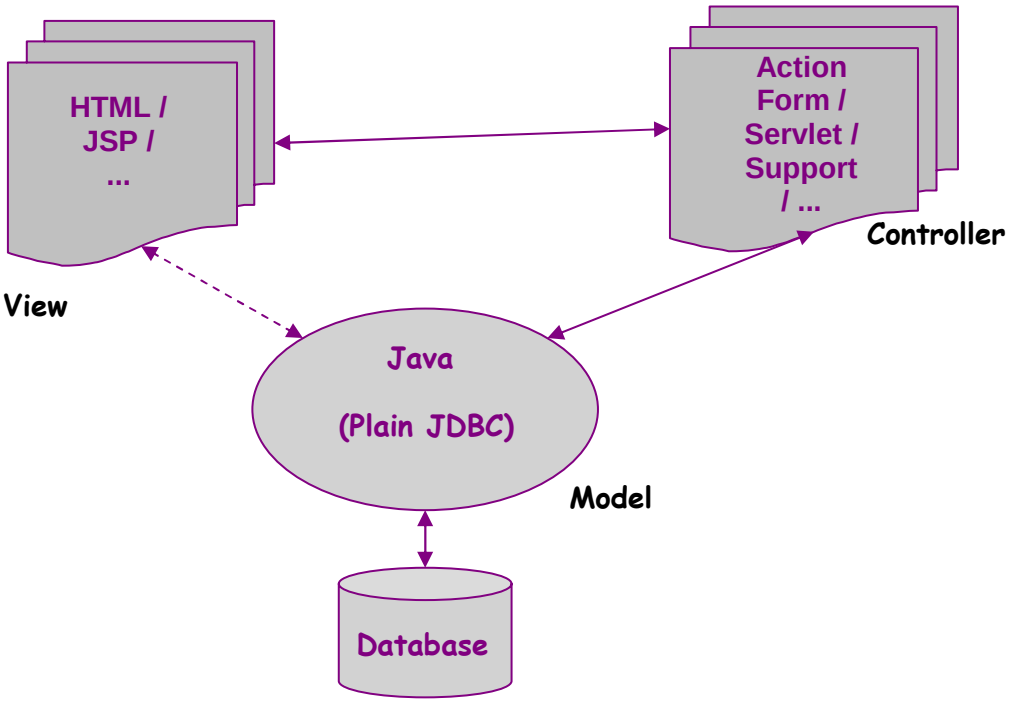
Exercise instructions

Some instructions in this lab are Windows® operating-system specific. If you plan on running the lab on an operating-system other than Windows, you will need to run the appropriate commands and use appropriate files (.sh or .bat) for your operating system. The directory locations are specified in the lab instructions using symbolic references, as follows:

Reference variable	Description
<WAS_HOME>	WebSphere Application Server home directory
<LAB_FILES>	Directory where the downloaded lab files have been unzipped. It contains a “src” directory and some utility “.bat” files.
<JAVA_HOME>	Java home directory. Could be <WAS_HOME>\java.
<DERBY_HOME>	Apache Derby home directory. <WAS_HOME>\derby
<PROJECT_NAME>	The name you choose for the project.

Part 1: Create project and become familiar with the sample application

The Eclipse project, jpa_dv_START, is a simple MVC-like application. To keep this task as simple as possible, it does not require an application server to run. Therefore, the *View* is displayed using a command-line instead of a web browser.



____ 1. Import the project into Eclipse

__ a. Go to File > New > Project… > Java > Java Project.
__ b. Under Contents, select the “Create project from existing source” radio button.
__ c. Browse to find the <LAB_FILES> directory. Highlight it and click OK
__ d. Keep the default or modify the project name. <PROJECT_NAME> will be used in future references.
__ e. Click Next.
__ f. On the Java Settings window, set the output directory for compiled classes.
1) Click the Source tab
2) Near the bottom of the window, ensure that Default output folder is set to <PROJECT_NAME>/bin
__ g. On the Java Settings window, add the appropriate links to your <JAVA_HOME> directory and the JPA 2.0 and Derby JAR files.
1) Click the Libraries tab
2) Click Add External JARs…
3) Browse to <WAS_HOME>\feature_packs\jpa\runtimes, select com.ibm.ws.jpa.thinclient_JPA2FEP1.0.0.jar, and click Open
4) Click Add External JARs… again
5) Browse to <DERBY_HOME>\lib, select derby.jar and derbytools.jar (both can be selected at once by holding the Ctrl key or they can be selected separately), and click Open

6) If a JRE System Library was already chosen, ensure it is pointing to Java 6. If one was not created, click Add Library… on the Java Settings / Libraries tab, select JRE System Library, and click Next. Select one of the radio buttons and specify the location of your JRE, if needed. Click Finish to close the JRE System Library window.

__ h. Click Finish to close the New Java Project window.

____ 2. Review the source files. The MVC structure and source files can be viewed in: <PROJECT_NAME>/src/simpleJPA

__ a. The *Model* files are:
Client.java
Participant.java
DB_Accessor.java

__ b. The *View* files are:
InputClientForm_JSP.java
ListClients_JSP.java

__ c. The *Controller* files are:
AddClient.java
ListClients.java

____ 3. Run Derby scripts

__ a. In Eclipse, locate <PROJECT_NAME>/refreshDB.bat. Right-click it and go to Open With > Text Editor. Update the directories specified for the `JAVA_HOME`, `DERBY_HOME`, and `PROJECT_HOME` variables. Make sure there are no spaces before or after the equal sign. Save the changes and close the file.

__ b. Right-click refreshDB.bat again and go to Open With > Default Editor. This will run a script to delete any existing tables and to create and populate the client table. If the script does not run successfully, it may be necessary to open the file again and remove the REM remark tag from lines 9 and 10 so that the PATH and CLASSPATH.

__ c. Repeat a. and b. for <PROJECT_NAME>/showTables.bat. This script will list and describe any existing tables. You can ignore any warning or error messages.

__ d. These files can be run at any time to clear or view tables. Once the Open With option is Default Editor, you can just double-click the scripts to run them.

____ 4. Run *View* classes

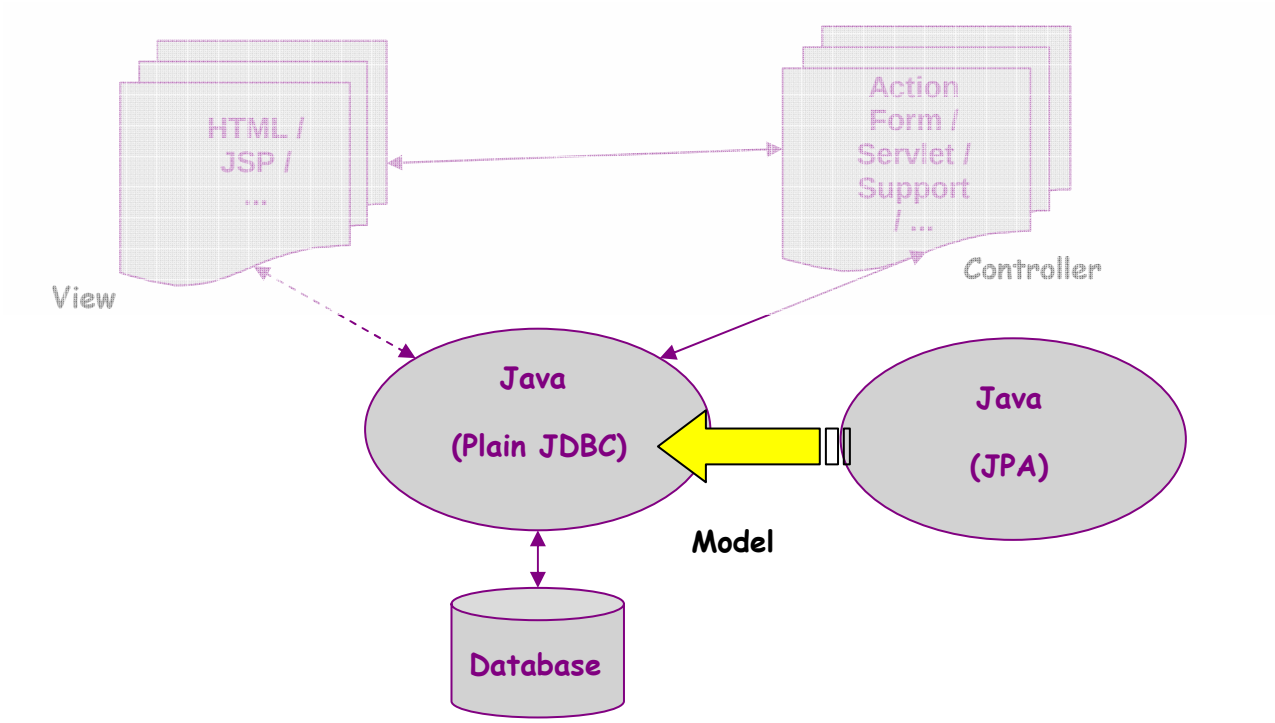
__ a. Right-click **ListClients_JSP.java**, go to Run As>Java Application. The output will show in the Eclipse console view at the bottom of the window.

__ b. Right-click **InputClientForm_JSP.java**, go to Run As>Java Application. Input client information when prompted. You can run **ListClients_JSP.java** again to see your new client.

__ c. These *view* classes should be run throughout the scenario to input data and verify input. You may be able to use the following shortcut: Ctrl+F11.

Part 2: Exchange plain JDBC model with JPA

The goal of this task is to identify, modify, or add files in order to change the entity classes and database interaction to use JPA instead of plain JDBC. In order to do so, make no changes to View and Controller, add annotations to Java Objects, simplify DB interaction code, and make some configuration updates (for example, persistence.xml).

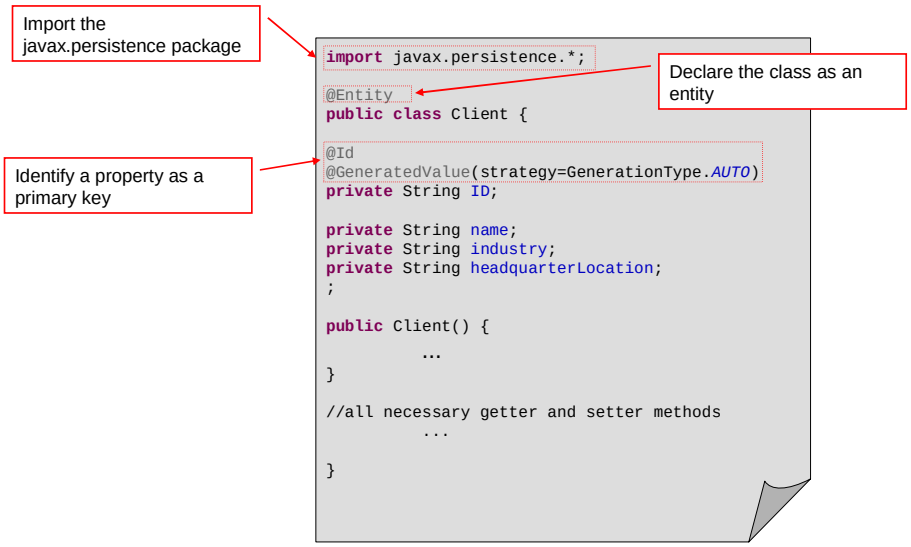


____ 5. Add annotations to Java objects (entity classes)

__ a. Open Client.java

1) Import javax.persistence.*
2) Add the @Entity annotation before the class definition
3) Add @Id before the ID attribute. As shown below, also adding @GeneratedValue(strategy=GenerationType.AUTO) will automatically create a value for ID.
4) Save changes (Eclipse can configured to automatically build your project).

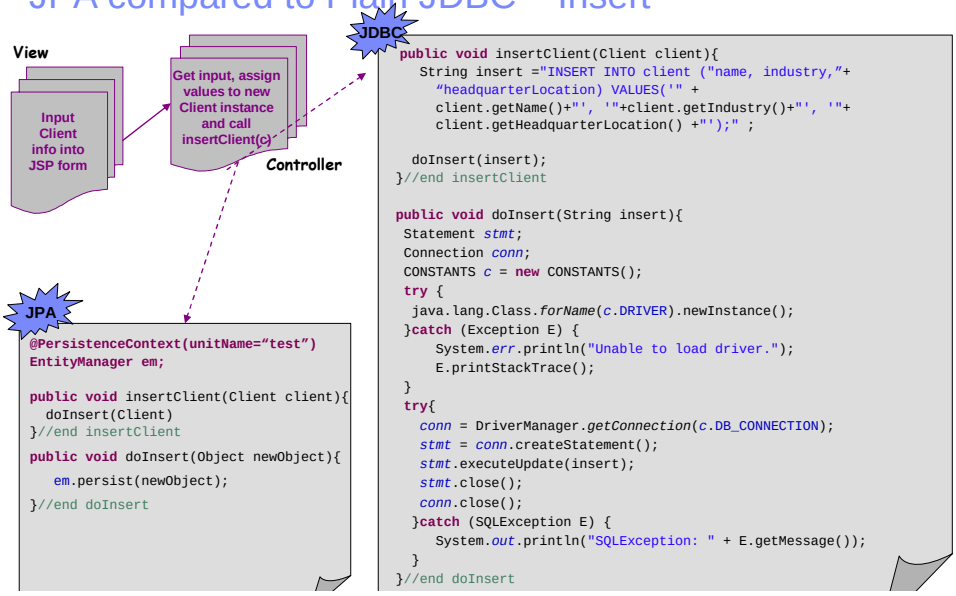
Annotate Entity Object Class



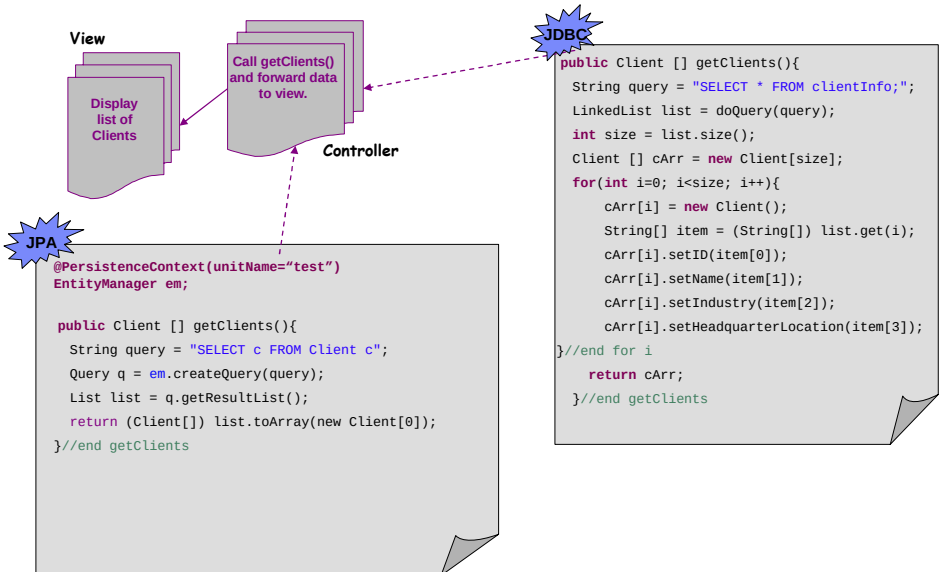
____ 6. Simplify DB interaction code

__ a. Delete (or move to a folder outside of the project) DB_Accessor.java
__ b. Delete (or move to a folder outside of the project) CONSTANTS.java
__ c. Right-click DB_Accessor.java.jpa, go to Refactor>Rename and remove ".jpa"
__ d. This new class has added an EntityManager and shows that the source code has been greatly simplified for insert() and get() methods (see following two images).

JPA compared to Plain JDBC – Insert



JPA compared to Plain JDBC – Retrieve



____ 7. You will need to create a persistence.xml file.

__ a. Create a META-INF directory in <PROJECT_HOME>/src/simpleJPA

__ b. Create a persistence.xml (the name of this file is not optional) file in the META-INF directory.

__ c. Copy and paste the following text into the persistence.xml file. The "openjpa.log" property can be removed if you do not prefer to see the SQL commands that JPA creates and runs. For a complete list of properties that can be defined in this file, go to the "OpenJPA Properties" link listed on Page 2 in the "Useful Links" section.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       version="1.0">
    <persistence-unit name="simplejpa" transaction-
       type="RESOURCE_LOCAL">
       <class>simpleJPA.model.Client</class>
        <properties>
            <property name="openjpa.jbdc.Schema" value="simplejpa" />
            <property name="openjpa.jdbc.SynchronizeMappings"
                   value="buildSchema"/>
             <property name="openjpa.ConnectionURL"
                   value="jdbc:derby:simplejpa-db;create=true"/>
              <property name="openjpa.ConnectionDriverName"
                   value="org.apache.derby.jdbc.EmbeddedDriver"/>
              <property name="openjpa.ConnectionUserName" value=""/>
              <property name="openjpa.ConnectionPassword" value=""/>
              <property name="openjpa.Log"
                   value="DefaultLevel=WARN,SQL=TRACE"/>
          </properties>
    </persistence-unit>
</persistence>
```

____ 8. Before the code can run, the Client class byte codes must be *enhanced* to allow for efficient interaction with the JPA runtime. This step can be set to start automatically by adding a Java agent and specifying the JPA thin client library:

__ a. Right-click the **InputClientForm_JSP.java** file in the Package Explorer and select Run As > Run Configurations…

Tip: Make sure the correct file is highlighted in the left-side navigation. If the file name does not appear, you may need to close the window, attempt to run InputClientForm_JSP.java and then repeat step (a).

__ b. Select the Arguments tab (make sure InputClientForm_JSP.java is highlighted on the left) and add the following line in the VM arguments text area:

-javaagent:<WAS_HOME>\feature_packs\jpa\runtimes\com.ibm.ws.jpa.thinclient_JPA2FEP1.0.0.jar

__ c. Repeat for **ListClients_JSP.java**

____ 9. Run InputClientForm_JSP, select option 2, and input the requested information.

____ 10. Run ListClientForm_JSP to see your new data.

Troubleshooting Tips:

1. Problem:

The META-INF/persistence.xml file is not in the class path if you receive an error similar to:

```
16  WARN   [main] openjpa.Runtime - The configuration property named
"openjpa.Id" was not recognized and will be ignored, although the name
closely matches a valid property called "openjpa.Id".
78  INFO   [main] openjpa.Runtime - Starting OpenJPA 2.0.0-SNAPSHOT
Exception in thread "main" <openjpa-2.0.0-SNAPSHOT-r422266:922441 fatal
user error> org.apache.openjpa.persistence.ArgumentException: A JDBC
Driver or DataSource class name must be specified in the
ConnectionDriverName property.
                                . . .
```

Suggestion:

If META-INF/persistence.xml is in the \src directory, but did not get copied to the output directory, then the simplest way to fix this problem is to copy META-INF/persistence.xml into the output directory, typically \bin or \classes.

2. Problem

The –javaagent argument has an error if you receive an error similar to:

```
2656  simplejpa  WARN   [main] openjpa.Enhance - This configuration
disallows runtime optimization, but the following listed types were not
enhanced at build time or at class load time with a javaagent: "
simpleJPA.model.Client".
Exception in thread "main" <openjpa-2.0.0-SNAPSHOT-r422266:922441 nonfatal
user error> org.apache.openjpa.persistence.ArgumentException: An error
occurred while parsing the query filter "SELECT c FROM Client c". Error
message: The name "Client" is not a recognized entity or identifier. Known
entity names: []
    at
org.apache.openjpa.kernel.exps.AbstractExpressionBuilder.parseException(Ab
stractExpressionBuilder.java:119)
                            . . .
```

Suggestion:

Check for spelling errors. Make sure there are no spaces. Make sure the directory containing the thin client is correct.

Part 3: Create a new Embeddable class and use a one-to-one relationship

Employee.java has been created without any JPA annotations. Currently, no participant related data has been (or can be) persisted using JPA. Add annotations to the Participant entity class such that a one-to-one relationship exists with the Client entity and such that participant data does not exist without a related client.

____ 1. Add annotations to Java objects (entity classes)

__ a. Open Employee.java

1) Import javax.persistence.*
2) Add the @Embeddable annotation before the class definition. This type of class does not require an @Id annotation because it cannot exist by itself and uses the ID of the class in which it gets embedded.

__ b. Open Client.java

1) Remove the comment slashes, "//" from line 19 to enable the @Embedded annotation.

____ 2. Update the persistence.xml file to recognize the newly annotated Employee class.

__ a. Open persistence.xml

1) Add the following line below the other class definition:
<class>simpleJPA.model.Employee</class>

____ 3. Save all of your changes.

____ 4. You may want to run refreshDB.bat (double-click) to reset the tables.

____ 5. Run InputClientForm_JSP, select option 3, and input the requested information.

____ 6. Run ListClientForm_JSP to see your new data.

____ 7. Double-click showTables.bat to see how JPA has updated the client table.

Part 4: Update Embeddable class to a many-to-one relationship

Update your relationship between Client and Employee to be one-to-many.

____ 1. Open Client.java

__ a. Comment out line 19, //@Embedded

__ b. Remove the comment slashes, "//" from lines 22 and 23 to enable the @ ElementCollection and @ CollectionTable annotations

____ 2. Save all of your changes.

____ 3. Run refreshDB.bat (double-click) to reset the tables.

____ 4. Run InputClientForm_JSP, select option 3, and input the requested information.

____ 5. Run ListClientForm_JSP to see your new data.

____ 6. Double-click showTables.bat to see what tables were created or modified.

What you did in this exercise

You converted the persistence layer of a simple application from plain JDBC to JPA. You then explored the functionality of a few basic 2.0 features. You should now understand the fundamental differences between plain JDBC and the object-relational persistence support provided by JPA. You should also be able to identify the minimal set of required files, property definitions, annotations, and so on to create and run an application that takes advantage of JPA.

This page is left intentionally blank.