

Télécom Saint-Étienne

TD Middleware - 3h00

Syed Gillani and Julien Subercaze

Objectives

- EJB with persistence data source.
- Transactions with EJB.

Tools Needed

- Eclipse EE (Enterprise Edition) Luna, Kepler etc.
- Glassfish.
- Java EE Sample Project from Glassfish (Downloadable from [here](#))
- Helper Classes (Downloadable from [here](#))

1 Background

Java support dozens of different network protocols, from a simple socket, hessian, burlap, SpringHttpInvoker, SOAP, XML and so on. However, in this TD, we will try to create a very simple client server application with EJB 3 and persistence, using Glassfish 3 as JavaEE Container.

We will leverage Bean-Managed transaction through the Java EE and employ JPA (Derby database) to implement a simple bank transaction system. It won't be too elaborated like what's currently being deployed in the banking system, but it is good enough to illustrate times when elaborated control of transaction is required. Figure 1 shows the diagram of tables, accompanied by the SQL scripts (provided with the helper classes) for the Derby DB that generates the tables which are involved with the transfer of funds from a savings account to a checking account. Both accounts can and will belong to a single customer.

Note that, we will use the same hello-stateless-ejb project and extend it for the bank use case.

2 DB Creation & Connection with GlassFish

Before start the creation of new beans and extending the bean session class, we need to create the tables in the database and configure the connection with the glassfish server. In order to create the database tables, please follow the below mentioned steps.

- Right click **hello-stateless-ejb-bean** project in eclipse, then **New ►Others ►SQL File**. Now we will write our SQL queries and execute them over our database (the one you created for the Derby DB).
- Copy the SQL queries (allQueries.SQL) provided in the helper classes folder. It contains queries to create the table mentioned in the schema (refresh the DB if you cannot see it).
- Right click the SQL file that you have created with the queries and **Execute SQL File** (see Figure. 2). Make sure that your derby server is running before executing the SQL file. That is, you have already executed the `./asadmin start-database`, and you have created a new database connection in Eclipse as well.
- Now you will see a set of tables in your database **USER** Schema.

FIG. 1 – Bank Account Transfer Database Tables

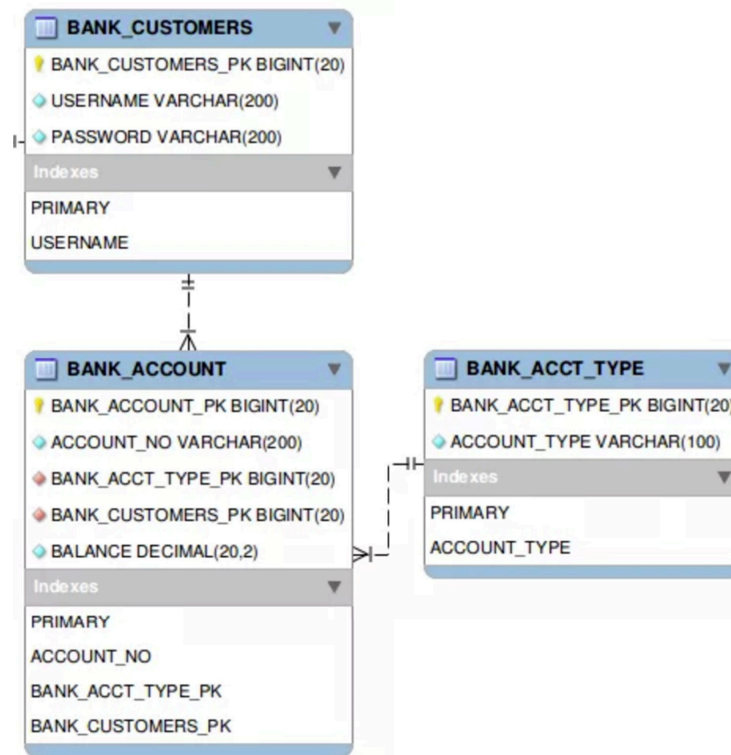


FIG. 2 – SQL Query Execution

```

5  PASSWORD VARCHAR(200) NOT NULL,
6  PRIMARY KEY(BANK_CUSTOMERS_PK),
7  UNIQUE(USERNAME)
8  );
9
10 INSERT INTO BANK_CUSTOMERS(USERNAME, PASSWORD) VALUES ('ModMax', 'passwd123');
11
12 CREATE TABLE BANK_ACCT_TYPE
13 (
14  BANK_ACCT_TYPE_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
15  ACCOUNT_TYPE VARCHAR(100) NOT NULL,
16  PRIMARY KEY(BANK_ACCT_TYPE_PK),
17  UNIQUE(ACCOUNT_TYPE)
18 );
19
20 INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('SAVINGS');
21 INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('CHECKING');
22
23 CREATE TABLE BANK_ACCOUNT
24 (
25  BANK_ACCOUNT_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
26  ACCOUNT_NO VARCHAR(200) NOT NULL,
27  BANK_ACCT_TYPE_PK BIGINT NOT NULL,
28  BANK_CUSTOMERS_PK BIGINT NOT NULL,
29  BALANCE DECIMAL(20, 2) NOT NULL,
30  PRIMARY KEY(BANK_ACCOUNT_PK),
31  UNIQUE(ACCOUNT_NO),
32  FOREIGN KEY(BANK_CUSTOMERS_PK) REFERENCES BANK_CUSTOMERS(BANK_CUSTOMERS_PK)
33 );
34
35 INSERT INTO BANK_ACCOUNT(ACCOUNT_NO, BANK_ACCT_TYPE_PK, BANK_CUSTOMERS_PK, BALANCE) VALUES
36 (
37  'BK-001-09',
38  (SELECT BANK_ACCT_TYPE_PK FROM BANK_ACCT_TYPE WHERE ACCOUNT_TYPE = 'SAVINGS'),
39  (SELECT BANK_CUSTOMERS_PK FROM BANK_CUSTOMERS WHERE USERNAME = 'ModMax'),
40  99999999
41 );
42
43

```

The next part is to configure the connection with the glassfish server and the derby database. Please follow the steps described below (you probably have done this in your Web-Service TD).

- Add JDBC Connection pool for derby with the properties described in Figure 3 and 4. (you can also refer to this [link](#). However, now we are working with Derby, instead of SQL).
- Due to a bug you might get an error at glassfish server, such as output stream is already being used. To get around this error, go to the domain.xml file (/glassfish4/glassfish/domain/config/). Copy the existing JDBC connection and change the name and add your new properties, finally restart the glassfish server. Now you will be able to see the new JDBC connection.
- The most important thing is the URL for your derby database, which you will get from

- the connection you created in Eclipse. (your-db-name ►properties).
- After configuring the connection pool, go to JDBC Resources on Eclipse Web portal, and then to the jdbc/___default. Change the pool name: the one you just created in the earlier step (see Figure 5).
- After this you must have a connection with the Derby database.

FIG. 3 – *Glassfish config-1*

Glassfish™ Server Open Source Edition

Common Tasks

Domain

server (Admin Server)

Clusters

Standalone Instances

Nodes

Applications

Lifecycle Modules

Monitoring Data

Resources

Concurrent Resources

Connectors

JDBC

JDBC Resources

JDBC Connection Pools

DerbyPool

MDT

TimerPool

JMS Resources

JNDI

JavaMail Sessions

Resource Adapter Configs

Configurations

default-config

Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

Load Defaults Flush Ping

Save Cancel

* Indicates required field

General Settings

Pool Name: MDT

Resource Type: javax.sql.DataSource

Datasource Classname: org.apache.derby.jdbc.ClientDataSource

Driver Classname:

Ping: Enabled

Deployment Order: 100

Description:

Pool Settings

Initial and Minimum Pool Size: 8 Connections

FIG. 4 – *Glassfish Config2*

Glassfish™ Server Open Source Edition

Common Tasks

Domain

server (Admin Server)

Clusters

Standalone Instances

Nodes

Applications

Lifecycle Modules

Monitoring Data

Resources

Concurrent Resources

Connectors

JDBC

JDBC Resources

JDBC Connection Pools

DerbyPool

MDT

TimerPool

JMS Resources

JNDI

JavaMail Sessions

Resource Adapter Configs

Configurations

default-config

server-config

Update Tool

Edit JDBC Connection Pool Properties

Modify properties of an existing JDBC connection pool.

Save Cancel

Pool Name: MDT

Additional Properties (8)

Add Property Delete Properties

Select	Name	Value	Description
<input type="checkbox"/>	PortNumber	1527	
<input type="checkbox"/>	Password	user	
<input type="checkbox"/>	User	user	
<input type="checkbox"/>	serverName	localhost	
<input type="checkbox"/>	DatabaseName	banktables	
<input type="checkbox"/>	connectionAttributes	create=true	
<input type="checkbox"/>	URL	jdbc:derby://localhost:1527/banktables	
<input type="checkbox"/>	driverClass	org.apache.derby.jdbc.ClientDriver	

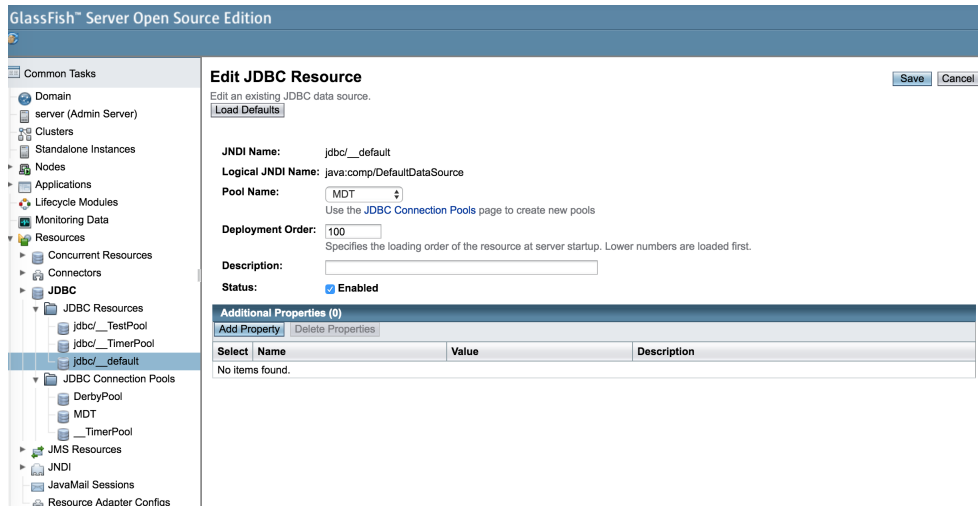
3 Creating Entities from the Tables

The next part is to create the Entity classes from the database tables you have just created. For this step either you can copy the classes provided in the helper classes (that you have downloaded or you can use Eclipse to create it for you). Follow the following steps:

Note that you can either create a new maven project and then convert it to EJB and JPA project, or use the existing hello-stateless-ejb-bean project. To create a new project, first create a new maven project, then right click the project, go to properties ►Project Facet ►check EJB and JPA.

Furthermore, if you guys are using the VM – as provided by Julien – then everything is configured in it for the JPA. This means you can skip the red green

FIG. 5 – *Glassfish Config2*



parts in the following instruction.

- Convert the hello-stateless-ejb-bean project into a JPA project, thus you can access the database from your beans.
- Right click your project **Configure** ► (skip this part if you're using the VM) **Convert to JPA project** (see Figure 6). **During this step Eclipse may ask you to add the user modules for JPA. If click on add new module and give the path of glassfish modules (/glassfish4/glassfish/modules, add all the jars).**
- Now in your project you must have a resource folder `src/main/resources`, which must contain a folder name `META-INF` and a file name `persistence.xml` (see Figure 7). If it's not there, create such folder and copy the `persistence.xml` from the helper classes folder. be careful with the `persistence.xml`, this file provides the connection from EJB to the DB.
- In the end, you must have the structure of the project shown in the Figure 8.

4 Adding Server Code

It's time to change the `StatelessSession` and `StatelessSessionBean` classes. Open the classes given in the helper classes and copy the content into your classes. In these classes, we have two main methods: `getBankCustomer` uses the username provided by the client and query it from the database (see also the content of your `BankCustomer` Entity class), while the `transferFund` class, use the bank customer queried earlier and update its bank balance using a transaction into his account. Carefully examines these methods and see how they are working.

```
@Override
public BankCustomer getBankCustomer(String user) {

    // Get the Customer that matches to a given name
    Query query = em.createNamedQuery( "BankCustomer.findByUsername" );
    query.setParameter( "username", user );
    //Execute the query
    BankCustomer bc= ( BankCustomer ) query.getSingleResult();
    return bc;

}
```

FIG. 6 – *Convert to JPA project*

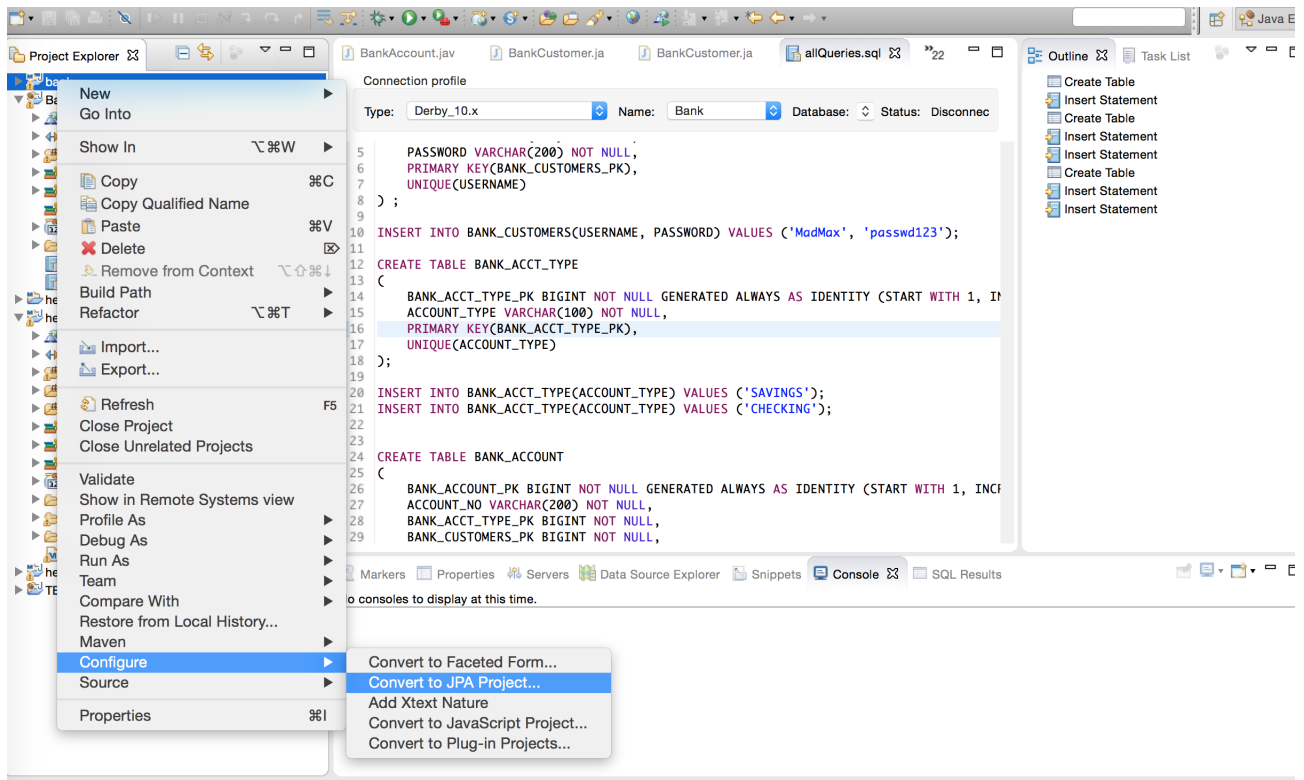


FIG. 7 – *JPA persistence class*

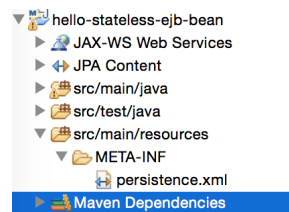
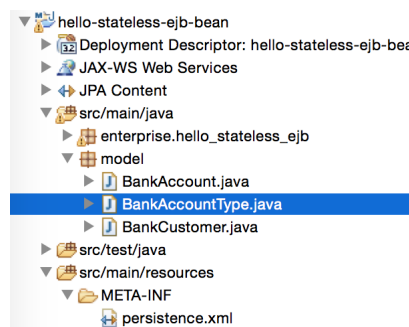


FIG. 8 – *JPA persistence class*



```
public void transferFunds( BankCustomer bankCustomerEntity, ..... ) {
    UserTransaction utx = context.getUserTransaction();
```

```

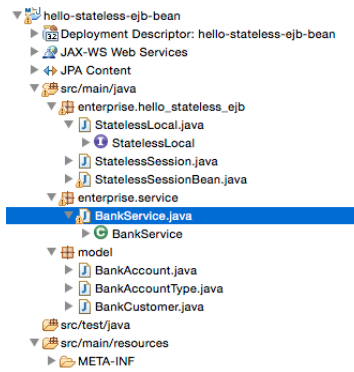
try
{
    //Get source bank account entity
    Query query = em.createNamedQuery( "BankAccountEntity.findByAccountNo" );
    query.setParameter( "accountNo", fromAccountNo );
    BankAccount fromBankAccountEntity = null;
    .....
}

```

5 Configuring the Tester Client

Since the server-code is done, we need to create a simple web service client to test our code. Thus create a new package (`enterprise.service`) in your `hello-stateless-ejb-bean` project. Then add the `BankService` class from the helper classes that you have downloaded earlier. This service class contains two methods: `getUserInfo` and `transferMoney`. The first method is used to get the name of the customer for a bank, while the second is used to transfer money from one account to another. Both of these classes employ the session bean methods.

FIG. 9 – *View Endpoints*



In the end you will have all the classes for the entity beans, session bean and service to test your beans (see Figure 9). Now deploy your project over the glassfish server. Then click on the deployed project ►view endpoints. This will open the tester client (see Figure 10).

Note that when we created tables using the SQL command, we also added a user and some accounts: we will use these information to perform a simple test on our application. Therefore, add the parameter values provided in Figure 11 .

6 Extending it Further

Now you can play around with the implementation and can extend it with further functions, such as adding new users, bank accounts and transfer balance between them. Pay attention to the code provided in `StatelessSessionBean` and the Entity class (such as `BankCustomer`) to add additional queries for the database.

FIG. 10 – *View Endpoints*

Common Tasks

- Domain
 - server (Admin Server)
- Clusters
 - Standalone Instances
- Nodes
- Applications**
 - hello-stateless-ejb-bean
- Lifecycle Modules
- Monitoring Data
- Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
- Configurations
 - default-config
 - server-config
- Update Tool

GeneralDescriptor

Edit Application

SaveCancel

Modify an existing application or module.

Name:hello-stateless-ejb-bean

Status:☒ Enabled

Implicit CDI☒ Enabled
Implicit discovery of CDI beans

Location:\${com.sun.aas.instanceRootURI}/eclipseApps/hello-stateless-ejb-bean/

Deployment Order:100
A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The default is 100.

Libraries:

Description:

Modules and Components (3)

Module Name	Engines	Component Name	Type	Action
hello-stateless-ejb-bean	[ejb, jpa, webservices, weld]	-----	-----	
hello-stateless-ejb-bean		BankService	StatelessSessionBean	View Endpoint
hello-stateless-ejb-bean		BK	StatelessSessionBean	

FIG. 11 – *Test Endpoints*

BankServiceService Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

```
public abstract java.lang.String enterprise.service.BankService.getUserInfo(java.lang.String)
```

```
public abstract java.lang.String enterprise.service.BankService.transferMoney(java.lang.String java.math.BigDecimal java.lang.String java.lang.String) throws enterprise.service.Exception_Exception
```