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-stateles-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*

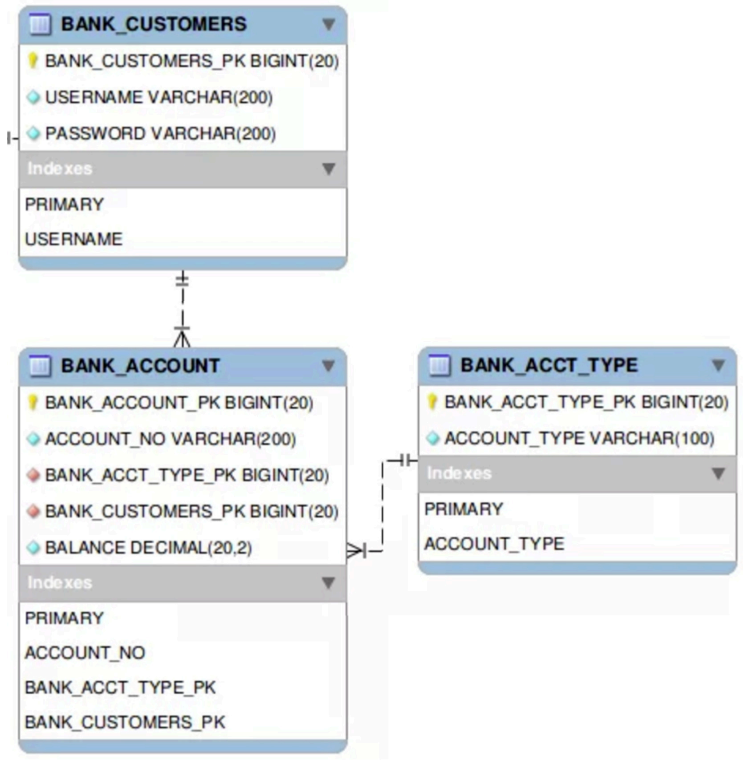
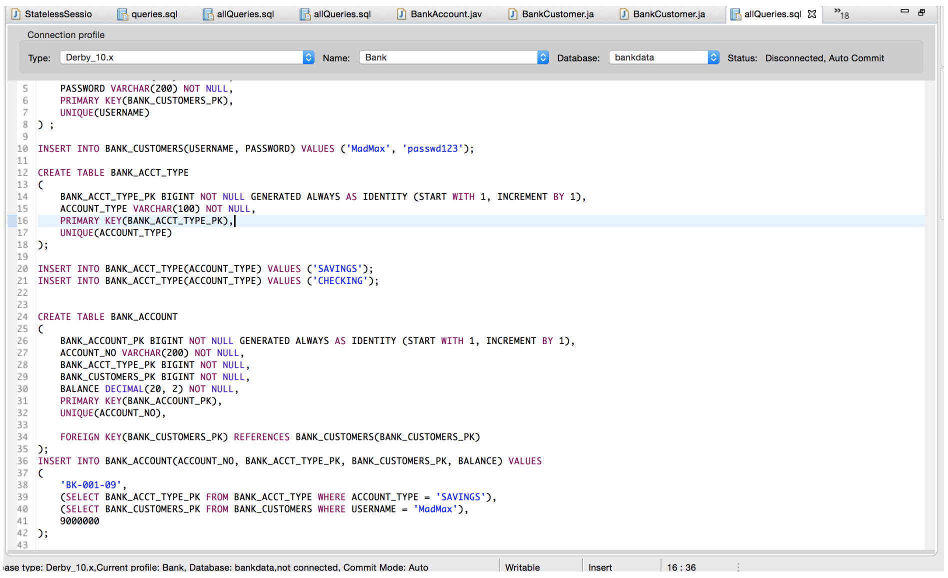


Fig. 2 – *SQL Query Execution*



Connection profile

Type: Derby_10.x Name: Bank Database: bankdata Status: Disconnected, Auto Commit

```
    PASSWORD VARCHAR(200) NOT NULL,
    PRIMARY KEY(BANK_CUSTOMERS_PK),
    UNIQUE(USERNAME)
) ;

INSERT INTO BANK_CUSTOMERS(USERNAME, PASSWORD) VALUES ('MadMax', 'passwd123');

CREATE TABLE BANK_ACCT_TYPE
(
    BANK_ACCT_TYPE_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    ACCOUNT_TYPE VARCHAR(100) NOT NULL,
    PRIMARY KEY(BANK_ACCT_TYPE_PK),
    UNIQUE(ACCOUNT_TYPE)
);

INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('SAVINGS');
INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('CHECKING');


CREATE TABLE BANK_ACCOUNT
(
    BANK_ACCOUNT_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
    ACCOUNT_NO VARCHAR(200) NOT NULL,
    BANK_ACCT_TYPE_PK BIGINT NOT NULL,
    BANK_CUSTOMERS_PK BIGINT NOT NULL,
    BALANCE DECIMAL(20, 2) NOT NULL,
    PRIMARY KEY(BANK_ACCOUNT_PK),
    UNIQUE(ACCOUNT_NO),

    FOREIGN KEY(BANK_CUSTOMERS_PK) REFERENCES BANK_CUSTOMERS(BANK_CUSTOMERS_PK)
);
INSERT INTO BANK_ACCOUNT(ACCOUNT_NO, BANK_ACCT_TYPE_PK, BANK_CUSTOMERS_PK, BALANCE) VALUES
(
    'BK-001-09',
    (SELECT BANK_ACCT_TYPE_PK FROM BANK_ACCT_TYPE WHERE ACCOUNT_TYPE = 'SAVINGS'),
    (SELECT BANK_CUSTOMERS_PK FROM BANK_CUSTOMERS WHERE USERNAME = 'MadMax'),
    9000000
);

```

ase type: Derby_10.x,Current profile: Bank, Database: bankdata,not connected, Commit Mode: Auto | Writable | Insert | 16 : 36

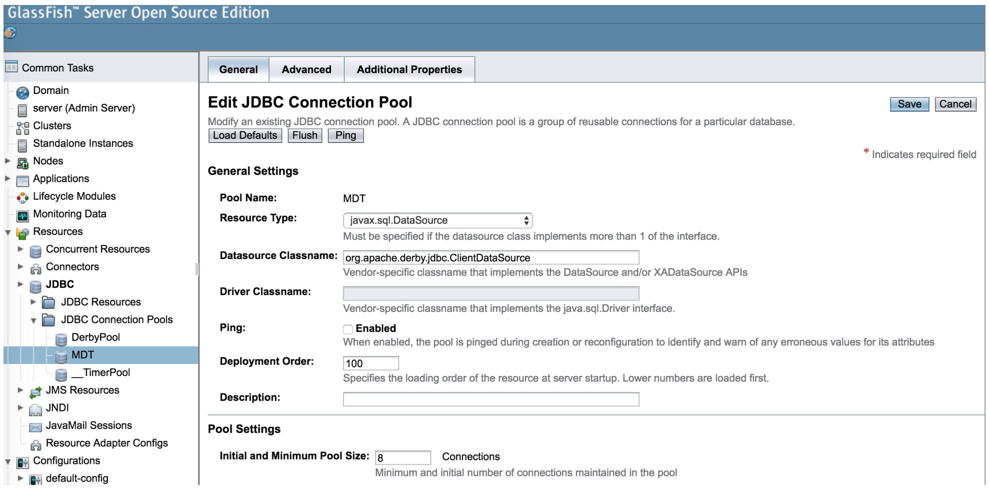
The next part is to configure the connection with the glassfish server and the derby database. Please follow the steps described below (you probably have done this in your Web-Service TD).

– 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 Eclispe. (`your-db-name` ►`properties`).

– After configuring the connection pool, go to JDBC Resources on Eclispe 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*



Edit JDBC Connection Pool

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

* Indicates required field

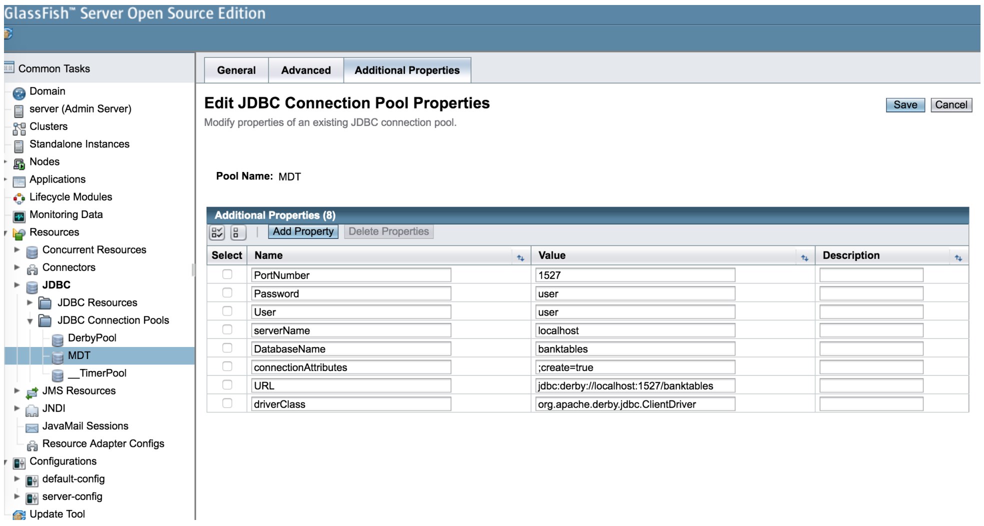
General Settings

Pool Name:	MDT
Resource Type:	javax.sql.DataSource Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:	org.apache.derby.jdbc.ClientDataSource Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:	 Vendor-specific classname that implements the java.sql.Driver interface.
Ping:	Enabled When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Deployment Order:	100 Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:	

Pool Settings

Initial and Minimum Pool Size:	8 Connections Minimum and initial number of connections maintained in the pool

Fig. 4 – *Glassfish Config2*



Edit JDBC Connection Pool Properties

Modify properties of an existing JDBC connection pool.

Pool Name: MDT

Additional Properties (8)

Select	Name	Value	Description
	PortNumber	1527	
	Password	user	
	User	user	
	serverName	localhost	
	DatabaseName	banktables	
	connectionAttributes	;create=true	
	URL	jdbc:derby://localhost:1527/banktables	
	driverClass	org.apache.derby.jdbc.ClientDriver	

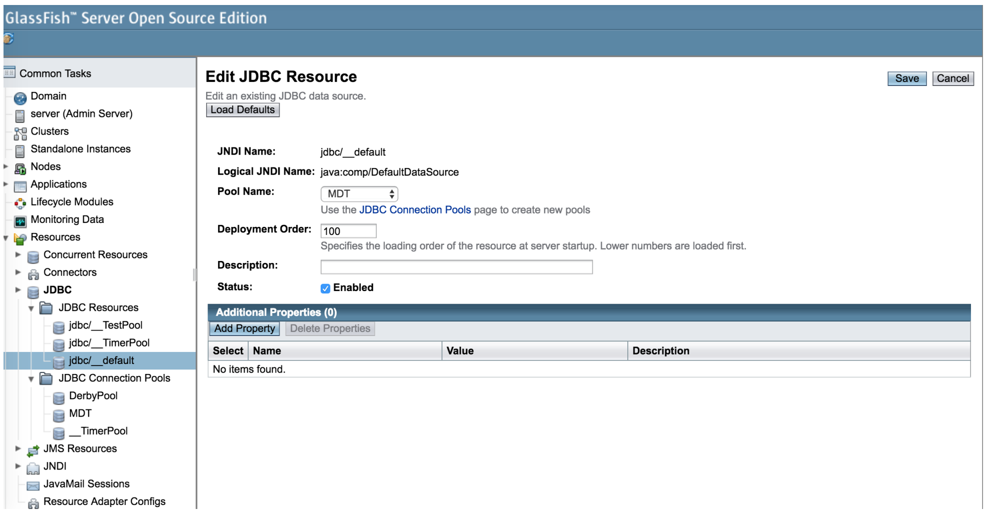
3 Creating Entities from the Tables

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.

– Convert the hello-stateless-ejb-bean project into a JPA project, thus you can access the database from your beans.

Fig. 5 – *Glassfish Config2*



Edit JDBC Resource

Edit an existing JDBC data source.

JNDI Name: jdbc/__default

Logical JNDI Name: java:comp/DefaultDataSource

Pool Name: MDT

Use the JDBC Connection Pools page to create new pools

Deployment Order: 100

Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Status: ☑ Enabled

Additional Properties (0)

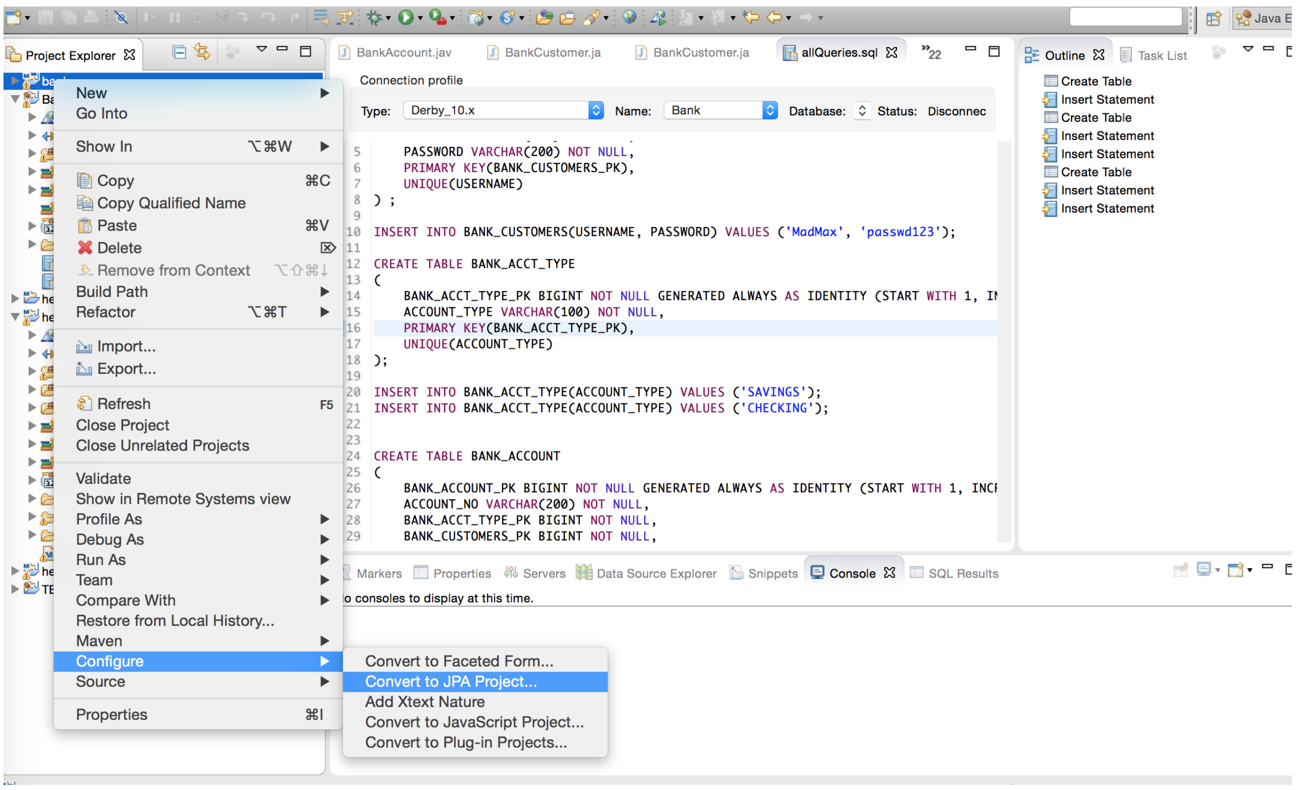
Select	Name	Value	Description
No items found.			

– Right click your project `Configure` ►`Convert to JPA project` (see Figure 6). During this step Eclipe 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 persistece.xml from the helper classes folder. becareful 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.

Fig. 6 – *Convert to JPA project*



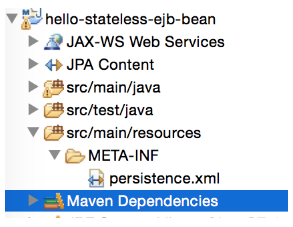
Connection profile

Type: Derby_10.x Name: Bank Database: Status: Disconnec

```
    PASSWORD VARCHAR(200) NOT NULL,
    PRIMARY KEY(BANK_CUSTOMERS_PK),
    UNIQUE(USERNAME)
) ;

INSERT INTO BANK_CUSTOMERS(USERNAME, PASSWORD) VALUES ('MadMax', 'passwd123');

CREATE TABLE BANK_ACCT_TYPE
(
    BANK_ACCT_TYPE_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, IN
    ACCOUNT_TYPE VARCHAR(100) NOT NULL,
    PRIMARY KEY(BANK_ACCT_TYPE_PK),
    UNIQUE(ACCOUNT_TYPE)
);

INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('SAVINGS');
INSERT INTO BANK_ACCT_TYPE(ACCOUNT_TYPE) VALUES ('CHECKING');


CREATE TABLE BANK_ACCOUNT
(
    BANK_ACCOUNT_PK BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 1, INCR
    ACCOUNT_NO VARCHAR(200) NOT NULL,
    BANK_ACCT_TYPE_PK BIGINT NOT NULL,
    BANK_CUSTOMERS_PK BIGINT NOT NULL,
```

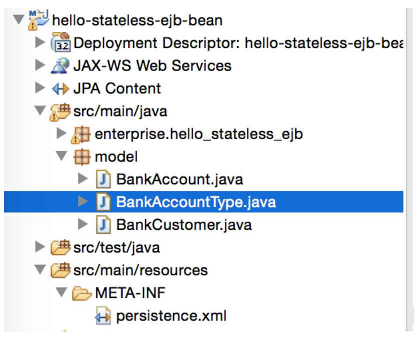
o consoles to display at this time.

Fig. 7 – *JPA persistence class*



```
▼ hello-stateless-ejb-bean
  ▶ JAX-WS Web Services
  ▶ JPA Content
  ▶ src/main/java
  ▶ src/test/java
  ▼ src/main/resources
    ▼ META-INF
        persistence.xml
  ▶ Maven Dependencies
```

Fig. 8 – *JPA persistence class*



```
▼ hello-stateless-ejb-bean
  ▶ Deployment Descriptor: hello-stateless-ejb-bea
  ▶ JAX-WS Web Services
  ▶ JPA Content
  ▼ src/main/java
    ▶ enterprise.hello_stateless_ejb
    ▼ model
      ▶ BankAccount.java
      ▶ BankAccountType.java
      ▶ BankCustomer.java
  ▶ src/test/java
  ▼ src/main/resources
    ▼ META-INF
        persistence.xml
```

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, usea 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.

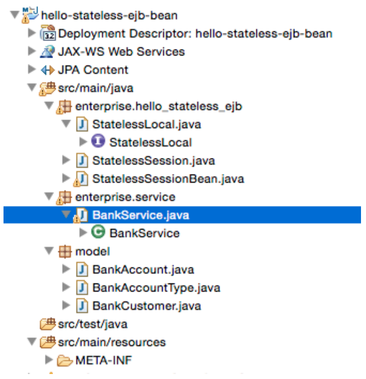
```
        @Override
public BankCustomer getBankCustomer(String user) {

// 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;

}
```

```
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;
            ....................................
}
```

4 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*

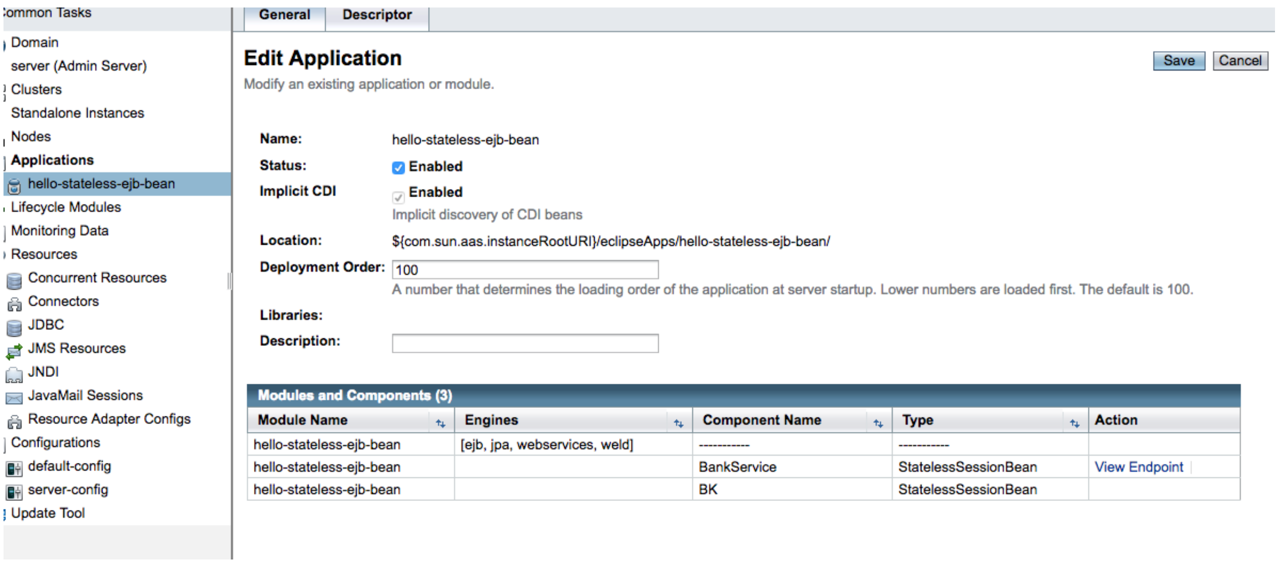


```
▼ hello-stateless-ejb-bean
  ▶ Deployment Descriptor: hello-stateless-ejb-bean
  ▶ JAX-WS Web Services
  ▶ JPA Content
  ▼ src/main/java
    ▼ enterprise.hello_stateless_ejb
      ▼ StatelessLocal.java
        ▶ StatelessLocal
      ▶ StatelessSession.java
      ▶ StatelessSessionBean.java
    ▼ enterprise.service
      ▼ BankService.java
        ▶ BankService
    ▼ model
      ▶ BankAccount.java
      ▶ BankAccountType.java
      ▶ BankCustomer.java
    src/test/java
  ▼ src/main/resources
    ▶ META-INF
```

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 .

Fig. 10 – *View Endpoints*



Edit Application

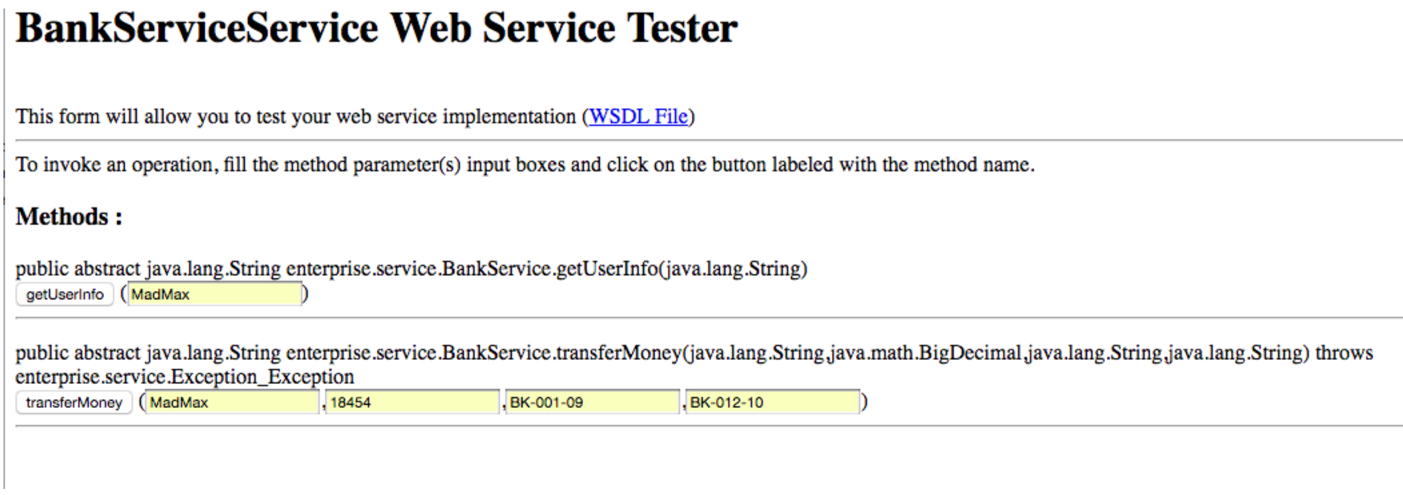
Modify an existing application or module.

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)

getUserInfo (MadMax)

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

transferMoney (MadMax , 18454 , BK-001-09 , BK-012-10)

5 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.