Hibernate Applications (HAP)

Exercise HAP.1 – Bank Application

The Setup:

The main purpose of this exercise is to use Hibernate in a larger application, applying the solutions to the problems we discussed in the lecture. We will use the same project as when we first added Spring – The Bank Application.

Start the exercise by downloading the **W2D4-HibernateApp-1.zip** project and add the Hibernate, mysql and log4j dependencies to it (as described in previous exercises).

The Application:

Running the application should create the following output:

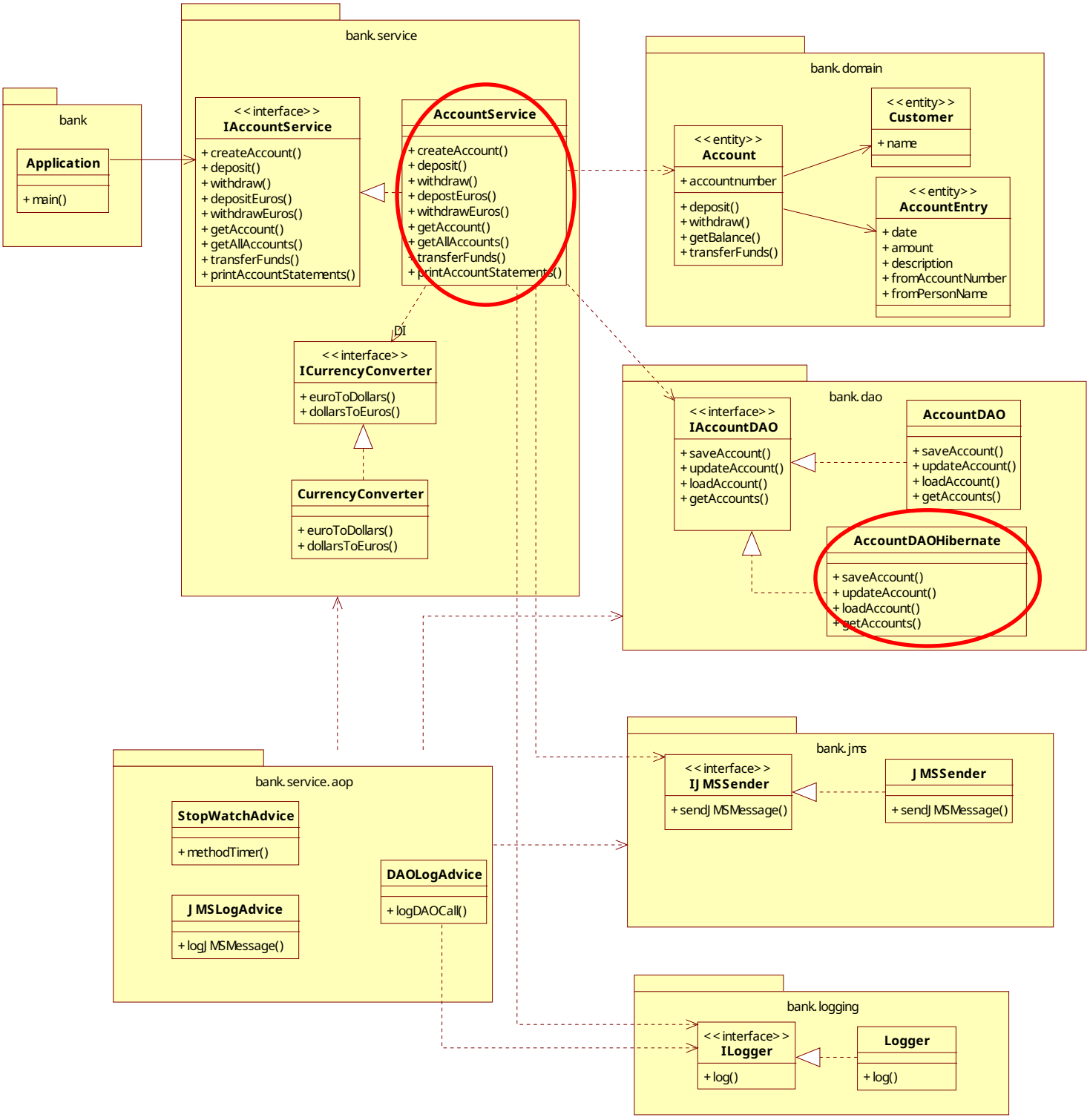
```
Statement for Account: 4253892
Account Holder: John Doe
-Date--------------------------Description-------------------Amount-------------
  Fri May 14 19:46:43 GMT 2010                       deposit            12450.00
  Fri May 14 19:46:43 GMT 2010                       deposit              314.00
  Fri May 14 19:46:43 GMT 2010      payment of invoice 10232             -100.00
-------------------------------------------------------------------------------
                                        Current Balance:            12664.00

Statement for Account: 1263862
Account Holder: Frank Brown
-Date--------------------------Description-------------------Amount-------------
  Fri May 14 19:46:43 GMT 2010                       deposit              240.00
  Fri May 14 19:46:43 GMT 2010                       deposit              529.00
  Fri May 14 19:46:43 GMT 2010                      withdraw             -361.10
  Fri May 14 19:46:43 GMT 2010      payment of invoice 10232              100.00
-------------------------------------------------------------------------------
                                        Current Balance:              507.90
```

The Exercise:

a) Create a **EntityManagerHelper** class similar to what is shown on the slides.

- Replace the AccountDAO object with a JPAAccountDAO, and put persistence annotations on the domain classes (Account, Customer, and AccountEntry). Feel free to add ID properties – you may want to use accountnumber as an assigned @Id for account..
- Make the Service level methods use the Hibernate DAOs and programmatically specify where the transactions should begin and commit inside the service methods
- Create a **persistence.xml** configuration file inside the META-INF directory in resources.
- Run the Application to see if it works and check the SQL it creates to retrieve the entities.


bank
Application
+ main()
bank.service
<<interface>>
IAccountService
+ createAccount()
+ deposit()
+ withdraw()
+ depositEuros()
+ withdrawEuros()
+ getAccount()
+ getAllAccounts()
+ transferFunds()
+ printAccountStatements()
AccountService
+ createAccount()
+ deposit()
+ withdraw()
+ depostEuros()
+ withdrawEuros()
+ getAccount()
+ getAllAccounts()
+ transferFunds()
+ printAccountStatements()
DI
<<interface>>
ICurrencyConverter
+ euroToDollars()
+ dollarsToEuros()
CurrencyConverter
+ euroToDollars()
+ dollarsToEuros()
bank.domain
<<entity>>
Account
+ accountnumber
+ deposit()
+ withdraw()
+ getBalance()
+ transferFunds()
<<entity>>
Customer
+ name
<<entity>>
AccountEntry
+ date
+ amount
+ description
+ fromAccountNumber
+ fromPersonName
bank.dao
<<interface>>
IAccountDAO
+ saveAccount()
+ updateAccount()
+ loadAccount()
+ getAccounts()
AccountDAO
+ saveAccount()
+ updateAccount()
+ loadAccount()
+ getAccounts()
AccountDAOHibernate
+ saveAccount()
+ updateAccount()
+ loadAccount()
+ getAccounts()
bank.service.aop
StopWatchAdvice
+ methodTimer()
JMSLogAdvice
+ logJMSMessage()
DAOLogAdvice
+ logDAOCall()
bank.jms
<<interface>>
IJMSSender
+ sendJMSMessage()
JMSSender
+ sendJMSMessage()
bank.logging
<<interface>>
ILogger
+ log()
Logger
+ log()

Exercise HAP.2 – Open Session in View

The Setup:

The main goal of this exercise is to have you to practice using Hibernate in a web application.

Start the exercise by downloading the **W2D4-HibernateApp-1.zip** project and add the hibernate, mysql and log4j dependencies. Also add the following web dependencies:

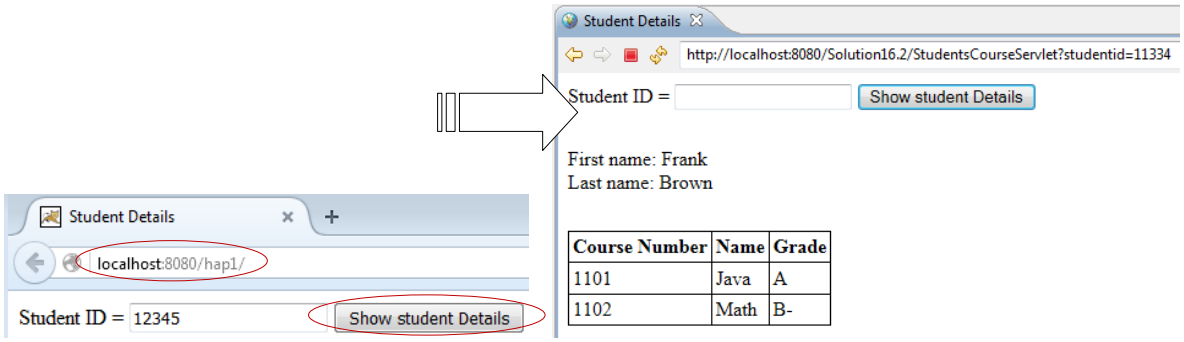
```
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
    <version>3.0.1</version>
</dependency>
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.2</version>
</dependency>
```

This project requires that a web server is setup, see second exercise in W1D1-Course-Overview for instructions on how to setup Apache Tomcat.

The Application:

The application is a small student - course application that lets you to lookup students by studentId, after which it will display all courses taken by that student, and the grades received.

Use **mvn package** to create a **.war** file for this project and deploy it on Tomcat as **hap2.war**. You should then be able to open a browser and go to http://localhost:8080/hap2/ where you can enter the Student Id **12345** and press on Show Student Details to see the next screen:



The Exercise:

The code currently does not use a database, instead the Students and their courses are stored in mock DAO objects.

a) Update the code to use Hibernate to store the Student and Course objects in MySQL.

- Make entities out of the Student and Course classes. The easiest way forward is to make coursenumber and studentId application assigned identifiers (not @GeneratedValue).
- Update the StudentDao to use Hibernate with MySQL instead of the ArrayList
- Create an EntityManagerHelper similar to the one shown in the slides.
- Instead of initializing the data inside the StudentDao constructor, have hibernate initialize the data by creating an **`import.sql`** file in the resources folder containing the following SQL insert statements.

```
INSERT INTO Student VALUES(12345, 'Frank', 'Brown');
INSERT INTO Course VALUES(1101, 'A', 'Java');
INSERT INTO Course VALUES(1102, 'B+', 'Math');
INSERT INTO Student_Course VALUES(12345, 1101);
INSERT INTO Student_Course VALUES(12345, 1102);
```

- Update the StudentService to begin and commit the transaction at the beginning and end of the getStudent() method.
- Update the StudentDao to retrieve the student from the database based on the provided studentId.
- Test to make sure that everything works

b) Did you notice that with this technology stack we did not have to create an OpenEntityManagerInView filter in order to load the related courses in the view?

- For the sake of practice, update the StudentService to close the EntityManager after the transaction has committed. Make sure that line 12 of the EntityManagerHelper says: `if (em == null || !em.isOpen()) {`
- Next update the StudentDao to use join fetch or an EntityGraph to load all the related courses when the student object is loaded.
- Test to make sure that everything works (checking the in the debug console that Hibernate is no longer making a separate select statement to get the courses).