Module 16: Integrating Hibernate

Exercise 16.1 – The 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 with regards to Session Management, Transactions, and Exception Handling. Start the exercise by importing the project from **C:\CS544\exercises\exercise16_1** and add the **Hibernate dependencies** to it (as described in the Hibernate exercises during the first week).

The Application:

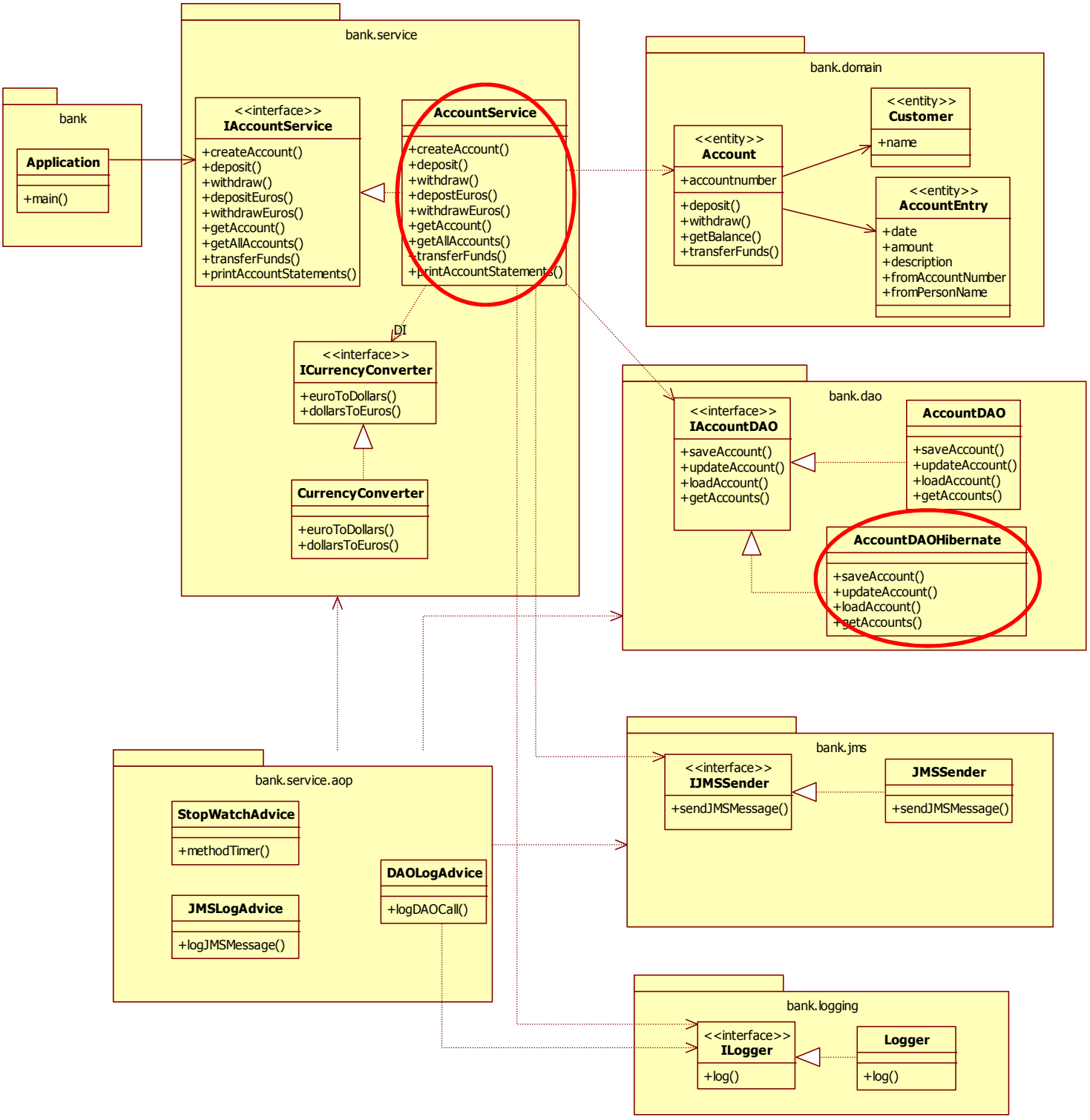
It is the same bank application as before, without any of the Spring elements added to it. Running the application should create the following output:

```
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:

1. Replace the DAO objects with Hibernate DAOs, and put persistence annotations on the domain classes (Account, Customer, and AccountEntry).
2. Make the Service level methods use the Hibernate DAOs and programmatically specify where the transactions start and stop inside the service methods
3. Create a Hibernate configuration, and be sure to use the **Thread Local** Pattern
4. Handle any exceptions Hibernate might throw in the controller (in our cause inside main).
5. Although the 'view' for this application only consists of a couple of print line statements, you will still need to have all entities loaded before the view can be printed. Because the transactions are ended in the service layer, entities cannot be loaded during the view (lazy initialization exception). You can avoid this by either using a join-fetch-query to pre-load the entities, or by explicitly pre-loading them using Hibernate.initialize().


bank.service
bank
Application
+main()
<<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()