Exercise Series 3

Banking example: Interoperability and Databases

5 June 2012

The objective of this exercise is to implement the interoperability between banks and the database access of the bank example using the results of series 2 and JDBC for the interaction with a database management system (MySql). *All questions below have to be answered explicitly in your report*, with your own words. Pieces of text literally copied from the lectures notes or books do not qualify for marking. The report with answers to the questions below should clearly describe the solutions produced, making it unnecessary for the teachers to dive into the code. Clarity and readability will be important for the marking of the reports.

The results of this exercise have to be submitted to Blackboard as a *zip file* containing the report in PDF format, all the files of the code you have produced (Java classes, JSP files, description files, Ant scripts, WAR files, etc.). The whole project should be generated, deployed and started using Ant scripts.

1 Banking example: architecture

In Figure 1 we recall the architecture of the banking system, adapted to this exercise series. The banking system has been structured according to the three tiers *presentation*, *business logic* and *resources*. The business logic has been split in two layers (application and banking layers), and the resources tier includes the database access. In the business logic we separated the parts that are present in any banking application, from the parts that are specific for the applications that have to be built. The banking system has two applications, namely *cash dispensing* and *home banking*.

The banking layer comprises a single subsystem for each of the following generic banking functions: transaction processing and authentication. The interfaces supported by these subsystems are Java RMI interfaces and have to be implemented exactly as specified in the exercises series 2 and here in order to guarantee interoperability between the groups. The Inter Bank Registry subsystem has been implemented by the teachers and can be accessed remotely by all the groups.

In this exercise you are asked to implement the interoperability between banks from different groups, and the interactions with the database management system using JDBC.

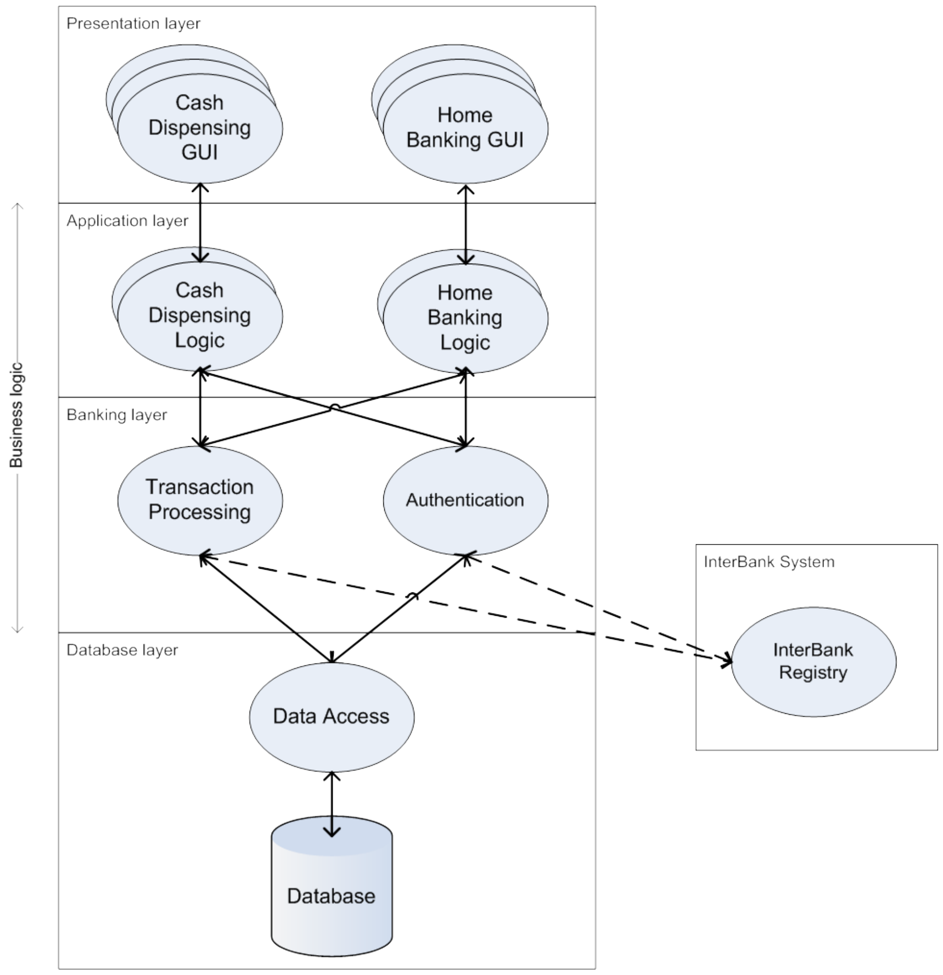


Figure 1. Banking example architecture.

2 Transaction processing

In this series the scenarios involving banks from different groups have to be implemented in addition to the scenarios implemented in series 2. Interfaces `Transaction` and `TransactionProcessing` remain the same as in series 2.

Operation `getBalance(accountId:String)` is responsible for obtaining the balance of account `accountId`. In case the account is managed by another bank *B*, the `TransactionProcessing` subsystem should delegate the operation to the `TransactionProcessing` subsystem of bank *B*. The Inter Bank System implements the `Interbank` RMI interface, which allows a bank to locate the `TransactionProcessing` subsystem of other banks. Section 5 discusses the Inter Bank System.

Operation `transfer()` tries to transfer some amount of money *M* from one account (debit account *D*), to another account (credit account *C*). The rules for allowing a transfer to take place described in series 2 still hold here.

Figure 2 shows how operation `transfer()` should perform an atomic money transfer using subsystems `TransactionProcessing` and `DataAccess` in case the debit and credit accounts are managed by distinct bank systems. This solution applies a two-phase commit protocol, where the `TransactionProcessing` subsystem on which the `transfer()` operation is invoked coordinates the distributed transaction. The two phase commit protocol consists of a prepare phase followed by a commit phase. During the prepare phase, the transaction coordinator requests each `TransactionProcessing` subsystem to perform the required modifications on the bank's account database, by invoking methods `prepareDebit()` and `prepareCredit()`. In case both succeed, the transaction coordinator starts the commit phase by requesting both `TransactionProcessing` subsystems to commit to the modifications (i.e., save them and make them final). This is done by invoking method `commit()` on both subsystems (both banks).

Figure 2 shows a successful money transfer in which the debit account is managed by bank system *A* and the credit account is managed by bank system *B*.

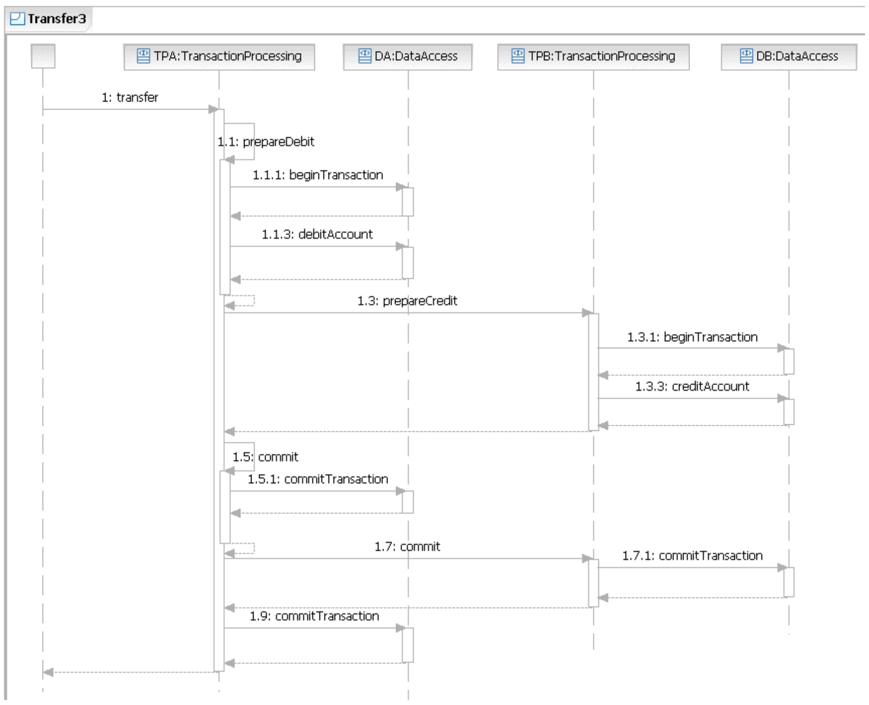


Figure 2. Transfer money to an account of another bank.

In case one or both subsystems fail after the prepare phase (i.e., after the `prepareDebit()` and `prepareCredit()` methods have been called), the coordinator orders both `TransactionProcessing` subsystems to rollback, i.e., to recover the situation at the beginning of the prepare phase, by invoking method `rollback()` on both subsystems.

Figure 3 shows a money transfer failure in which the crediting subsystem *B* has not been able to complete the transaction.

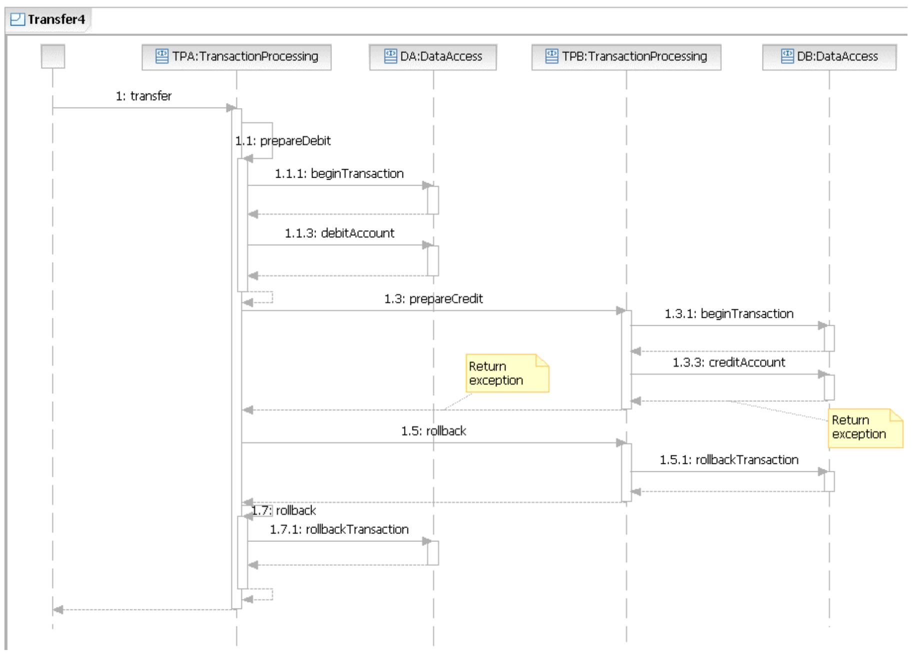


Figure 3. Money transfer failure between two banks.

3 Authentication

Analogously to the `TransactionProcessing` subsystem, the `Authentication` subsystem should delegate the authenticate operations `authenticatClient()` and `authenticateCard()` to the `Authentication` subsystem of another bank *B*, in case the client to be authenticated is a client of bank *B*. The Inter Bank System allows a bank to locate the `Authentication` subsystem of another bank (see Section 5).

4 Database interactions

The operations supported by the `DataAccess` interface and simulated in series 2 have to be supported by interactions with a database management system, for which we have chosen MySql.

Transactional support is necessary to implement the `DataAccess` interface. Transactional support can be obtained in MySql by assigning tables to the Innodb storage engine, for example, as in

```
CREATE TABLE t (i INT) ENGINE = INNODB
```

The following functions have to be used in order to perform transactions:

- `BEGIN`: marks the start of a transaction;
- `COMMIT`: marks the end of a transaction. All modifications on the database that have been made since the preceding `BEGIN` are saved;
- `ROLLBACK`: marks the end of a transaction. All modifications on the database that have been made since the preceding `BEGIN` are undone.

These functions should correspond to the operations of the `DataAccess` interface that have the same functionality.

Table 1 gives the relation between JDBC operations and these functions.

Table 1. MySql commands and corresponding JDBC methods.

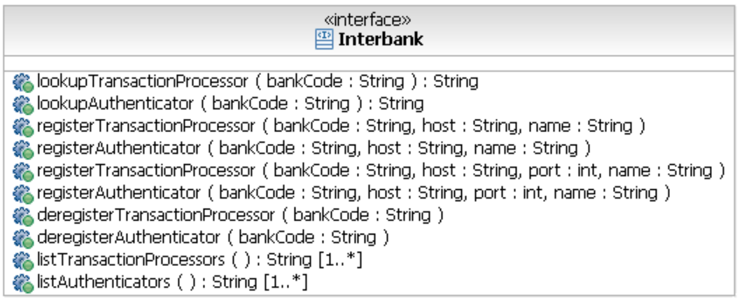
MySql command	`java.sql.Connection` *method*
`BEGIN`	-
`COMMIT`	`commit()`
`ROLLBACK`	`rollback()`
`SET AUTOCOMMIT = 0/1`	`setAutoCommit(false/true)`

5 Inter Bank system

The purpose of the Inter Bank System is to enable interaction between `TransactionProcessing` and `Authentication` subsystems of different bank implementations. The Inter Bank System provides the `Interbank` RMI interface, which allows a bank system to register references to its `TransactionProcessing` and `Authentication` subsystems, and other banks to look up these references.

The Inter Bank System has been implemented and is currently running in host `ewi887.ewi.utwente.nl`. A reference to interface `Interbank` of this instance can be obtained via the RMI registry in the same host at the default port 1099, using the service name `"InterBank"`.

Figure 4 shows the methods of the `Interbank` interface.



```
«interface»
Interbank

lookupTransactionProcessor ( bankCode : String ) : String
lookupAuthenticator ( bankCode : String ) : String
registerTransactionProcessor ( bankCode : String, host : String, name : String )
registerAuthenticator ( bankCode : String, host : String, name : String )
registerTransactionProcessor ( bankCode : String, host : String, port : int, name : String )
registerAuthenticator ( bankCode : String, host : String, port : int, name : String )
deregisterTransactionProcessor ( bankCode : String )
deregisterAuthenticator ( bankCode : String )
listTransactionProcessors ( ) : String [1..*]
listAuthenticators ( ) : String [1..*]
```

Figure 4. `InterBank` interface.

The Inter Bank System is a bank registry that stores addressing information on the `TransactionProcessing` and `Authentication` subsystems by using the RMI URL format `rmi://host:port/name`, where `host` denotes the host (remote or local) where the registry is located, `port` is the port number on which the registry accepts calls, and `name` is a `String` that denotes a remote object. The RMI URL is handy to interact with the registry via the `java.rmi.Naming` interface.

For example, when registering a Transaction processor `tp1` (remote object that implements the `TransactionProcessing` interface) that runs on machine `mickey.example.com`, for the bank id `003` (implemented by group 3), we have to call

```
registerTransactionProcessing (“003”, “mickey.example.com”, 1099, “tp1”)
```

In this example, we consider that the RMI registry in `mickey.example.com` is available at port 1099. The InterBankSystem assigns the URL `rmi://mickey.example.com:1099/tp1` to the bank identified as `003`.

6 Bank identification

The bank identification schema has already been given in exercise series 2, but is repeated here for convenience.

A bank that manages some account can be recognized by the first part of the account identifier. An account identifier consists of two parts: a *bank code*, which uniquely represents the bank that manages the account, and an *account code*, which uniquely identifies the account within this bank. An account identifier is encoded as an ASCII string of length 10, such that the first 3 characters are used for the bank code and the remaining 7 characters are used for the account code.

Each bank implementation built by a group is represented as the `String 00X`, where `X` denotes the group number. For example, the implementation of group 1 is identified as `001`, group2 as `002`, etc. An example of an account id is `0030000012`, which is account `0000012` of bank `003` (bank implemented by group 3).

Question 1
Draw UML class diagrams that represent the classes of your system. Discuss the classes you implemented in addition to the classes implemented in series 2, and the modifications you made in your classes of series 2.

Question 2
Discuss how your implementation of method `transfer()` deals with problems (errors, exceptions, etc.) during the commit phase and during a rollback.

Hint: it may be impossible to implement `transfer()` in such a way that it is able to recover from all possible problems. Therefore, indicate which problems your implementation can handle, and which problems it cannot. In case your implementation is not able to solve some problem, can you think of other ways to solve it?

Question 3
Describe your solution to set up and interact with the MySql database management system.

Question 4
Describe the approach you took in order to test your application. How did you test the application as a whole? How can you be sure the system works? Describe both the testing architecture (implementation-under test and testing components) and the testing strategy (test cases) that you applied, and the results produced in these tests.