Scala Project Part 3 – Actors, Futures, and Promises

TDT4165 – Programming Languages 2015

1 Exercise Description

Traditional online banking applications are currently experiencing great competition from new players in the market who are offering direct transactions with a few seconds of response time. Banks are therefore looking at possibilities of changing their traditional method which involves batch transactions at given times of day with hours in between. They must now update their software systems to adapt to the current demand, which means transactions must be handled in real-time.

Your overall task for this project is to implement features of a real-time banking transaction system.

In the zipped folder `Exercise` is the source code for Part 3 of the project. Unzip the folder and import its contents to the Scala IDE of your choice. The file structure is presented below.

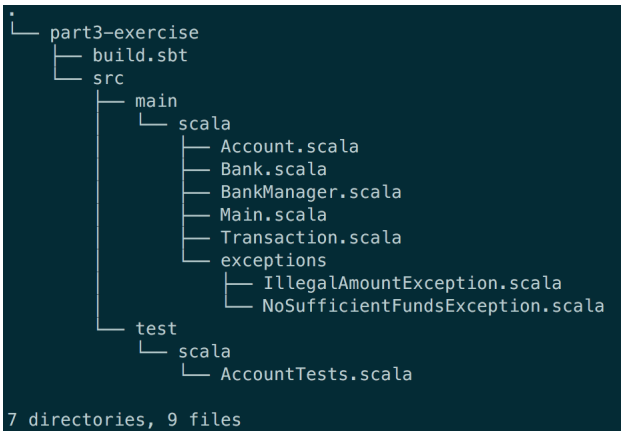

Figure 1: File structure

In the final part of the Scala project, you will still be working with transactions, but this time you will need to support transactions between accounts in different Bank-instances, using actors. It is highly recommended that you read up on Actors before starting this part, if you are not already familiar with it.

You will be working in `Bank.scala` and `Account.scala`. Also provided is `BankManager.scala`, which is the Actor System that you will need when looking up other Banks or Accounts. Below is some important information about what you should implement in this assignment:

- All Accounts should be initialized with a unique AccountID which is exactly 4 characters long, and each Account belongs to exactly one Bank.
- All Banks should be initialized with a unique BankID which is exactly 4 characters long.
- An *account number* is defined as the BankID concatenated with AccountID (e.g. 40012001 - Account with AccountID 2001 belongs to a bank with BankID 4001). Account numbers are represented as Strings in the program, for easier manipulation.
- A transaction works as follows: the sending Account (A) calls the `transferTo`-method. `transferTo` withdraws the correct amount from the A, adds a new `Transaction`-instance (let's call this `t`) to a HashMap internal to A, and forwards `t` to the A's Bank. The Bank should then forward `t` either to a different Bank or an Account, depending on whether `t` is internal or not. If `t` is external and sent to a different Bank, that Bank should forward `t` to the correct Account.

  When the receiving Account (B) has received `t`, B should process `t`, and send a `TransactionRequestReceipt`, saying that `t` succeeded, back to the A, the same way `t` was sent (only backwards).

  If the transaction somehow fails on the way (e.g., if a Bank or Account does not exist), a `TransactionRequestReceipt` saying that `t` failed should be sent back to A from the point of failure.

  When A has received a `TransactionRequestReceipt`, it should update the information about `t` in the HashMap that the transaction was stored in earlier.
- A transaction holds the following information:
  - `from: String`
    The *account number* of the sending Account.
  - `to: String`
    The *account number* of the receiver Account. An internal transaction does not need to include the BankID in the account number (e.g., Account A with account number 40012001 wants to transfer to Account B with account number 40012002, they are in the same Bank and `to` should be 2002; whereas if a transaction from Account A to Account C with account number 50012002 (different Bank), `to` should be 50012002).
  - `amount: Double`
    The amount of money that should be transferred.
  - `status: TransactionStatus.Value`
    An enum representing the current status of the Transaction (`PENDING`, `SUCCESS`, or `FAILED`).
  - `id: String`
    A unique ID.

- `receiptReceived: Boolean`
  When the sending Account receives the receipt from the receiving Account, this value should be `true`.

2 Running the Tests

2.1 IDE

To run the tests within a Scala IDE, simply run `AccountTests.scala`. If you do not have the option of running this file, make sure the `src/test/scala`-directory is marked as a source folder (Eclipse) or a test source root (IntelliJ), and that your project has `scalatest.jar` in its build path (this is not necessary if you are running the tests through sbt).

2.2 Command Line

To run the tests from the command line, `cd` into the `part2-exercise`-directory, and run the `sbt test` command.

If you have not yet installed sbt, visit `www.scala-sbt.org/release/tutorial/Setup.html` and follow the installation instructions for your OS.

3 Deliverables and Deadline

To submit your solution, upload the files `Bank.scala`, `Account.scala`, and any other modified/added files to itslearning before **November 16th**.

Your code should be presented to a TA during lab hours before **November 20th**. For your submission to be approved, 70% test coverage is required; 14 of the 17 tests need to pass (two of the tests will pass by default if you do not change the provided code).

Good luck!