Exercises week 11

Last update: 2023/11/01

Goal of the exercises

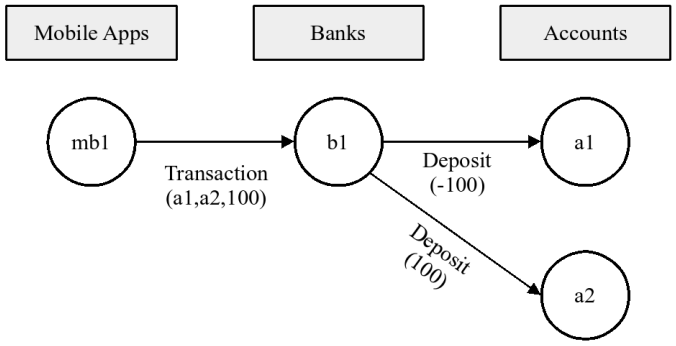
The goals of this week's exercises are:

- Design an Actor system.
- Design the communication protocol between actors.
- Implement the design in Akka.

Exercise 11.1 Your task in this exercise is to implement a Mobile Payment system using Akka. The system consists of three types of actors:

- *Mobile App*: These actors send transactions to the bank corresponding to mobile payments.
- *Bank*: These actors are responsible for executing the transactions received from the mobile app. That is, substracting the money from the payer account and adding it to the payee account.
- *Account*: This actor type models a single bank account. It contains the balance of the account. Also, banks should be able to send positive deposits and negative deposits (withdrawals) in the account.

The directory `code-exercises/week11exercises` contains a source code skeleton for the system that you might find helpful.



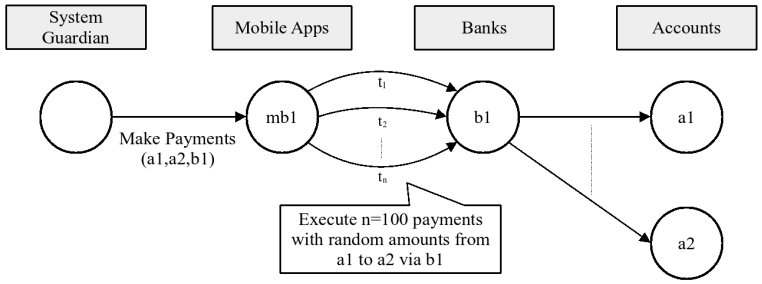
The figure above shows an example of the behavior. In this example, there is a mobile app, $mb1$, a bank, $b1$, and two accounts, $a1$ and $a2$. The arrow from $mb1$ to $b1$ models $mb1$ sending a transaction to $b1$ indicating to transfer 100 DKK from $a1$ to $a2$. Likewise, the arrows from $b1$ to $a1$ and $a2$ model the sending of deposits to the corresponding accounts to realise the transaction—the negative deposit can be seen as a withdrawal.

Mandatory

1. Design and implement the guardian actor (in `Guardian.java`) and complete the `Main.java` class to start the system. The `Main` class must send a kick-off message to the guardian. For now, when the guardian receives the kick-off message, it should spawn an MobileApp actor. Finally, explain the design of the guardian, e.g., state (if any), purpose of messages, etc. Also, briefly explain the purpose of the program statements added to the `Main.java` to start off the actor system.

   Note: In this exercise you should only modify the files `Main.java` and `Guardian.java`. The code skeleton already contains the minimal actor code to start the MobileApp actor. If your implementation is correct, you will observe a message `INFO mobilepaymentsolution.MobileApp - Mobile app XXX started!` or similar when running the system.

2. Design and implement the Account actor (see the file `Account.java` in the skeleton), including the messages it must handle. Explain your design choices, e.g., elements of the state, how it is initialized, purpose of messages, etc.
3. Design and implement the Bank actor (see the file `Bank.java` in the skeleton), including the messages it must handle. Explain your design choices, e.g., elements of the state, how it is initialized, purpose of messages, etc.
4. Design and implement the Mobile App actor (see the file `MobileApp.java` in the skeleton), including the messages it must handle. Explain your design choices, e.g., elements of the state, how it is initialized, purpose of messages, etc.
5. Update the guardian so that it starts 2 mobile apps, 2 banks, and 2 accounts. The guardian must be able to send a message to mobile app actors to execute a set of payments between 2 accounts in a specified bank. Finally, the guardian must send two payment messages: 1) from $a1$ to $a2$ via $b1$, and 2) from $a2$ to $a1$ via $b2$. The amount in these payments is a constant of your choice.
6. Modify the mobile app actor so that, when it receives a "make payments" message from the guardian, it sends 100 transactions between the specified accounts and bank with a random amount. Hint: You may use `Random::ints` to generate a stream of random integers. The figure below illustrates the computation.



7. Update the Account actor so that it can handle a message `PrintBalance`. The message handler should print the current balance in the target account. Also, update the guardian actor so that it sends `PrintBalance` messages to accounts 1 and 2 *after* sending the make payments messages in the previous item.

   What balance will be printed? The one before all payments are made, or the one after all payments are made, or anything in between, or anything else? Explain your answer.
8. Consider a case where two different bank actors send two deposit exactly at the same time to the same account actor. Can data races occur when updating the balance? Explain your answer.

Challenging

9. Modify the system above so that Account actors are associated with a bank, and ensure that negative deposits are only executed if they are sent by that bank. In other words, only the account's bank is allowed to withdraw money. Note that positive deposits may be received from any bank.
10. Modify the system so that an account's balance cannot be below 0. If an account actor receives a negative deposit that reduces the balance below 0, the deposit should be rejected. In such a case, the bank should be informed and the positive deposit for the payee should not be performed.