TDT4165 Programming Languages Scala Project Part 2

Ola Lømo Ellingsen

November 9, 2024

Task 1:

See the code provided in the zip-file.

Task 2:

See the code provided in the zip-file.

Task 3: Explain the code

1. The code is designed to represent a bank system where transactions can be processed between accounts in a safe and reliable way, with emphasis on using concurrency. The system is built using object orientation in order to represent the Bank, Accounts, Transactions and their relations in a logical way. The Bank holds a register of Accounts, a pool with awaiting transactions and a pool of completed transactions. The main functions of the Bank is to transfer money between accounts, process all transactions and handle transactions success, retries and failure based on available retries and the account balances. To ensure the system can handle multiple transfers at once, transactions are processed in separate threads to allow for concurrent processing.

The Transaction class represents each transaction with details about the sender's account and destination account, as well as status (pending, failed, success). If a transaction fails, it can be retried three times. TransactionPool manages the queue of the transactions that needs to be processed. The Account represents an individual bank account with a unique code and the correct balance. Each account has thread-safe methods for deposit and withdrawal, and these methods returns a new instance of the Account object, making each Account object immutable.

2. The features of the Account and Transaction classes where the easiest to implement because I quickly understood the concepts of the functionality (e.g. deposit and withdraw), and how these could be implemented. How to implement the queue for the TransactionPool and use basic methods to read and write to this data structure was also straight forward, as this is basic object orientation.

3. The more difficult part to implement was the main Bank class, as this had to manage all accounts and transactions in a concurrent way. After a while I understood that the transfer method should be synchronized, allowing only one thread to add a transaction to the pool at a time. It was difficult to implement the **processTransactions** method because I didn't

understand the idea behind the "workers" in the beginning. I then realized I had to iterate over the queue and filter it based on pending transactions, before starting each thread. In the Transaction class, I had to implement a mutable variable and a method to decrement it in order to track the number of remaining retries each transaction had. This made it easier to implement the logic in the **processSingleTransaction** method, where the number of remaining retries decided if the transaction should be marked as failed or pending.

4. For the tests to pass, I needed to implement the transfer method as a synchronized method. In order for the list of threads (workers) to work, I needed to call the **iterator** before filtering and then call the **toList** method in the end. I used the **forEach** method on the lists of succeeded and failed transactions, and I used the **getRemainingRetries** method implemented in the Transaction class in order to check if the transaction should be marked as failed and added to the completed Transactions.

In the **processSingleTransaction**, I had to make a new thread to process the transaction. Pattern matching is used to check if the accounts involved exists, and if balance and amount are valid before making the transaction. If valid, it updates the account balances and marks the transaction as successful. If invalid, it decrements the retry count and updates the transaction status accordingly. If either account does not exist, it marks the transaction as failed.