CSCM29 - Blockchain, Cryptocurrencies and Smart Contracts

Arnold Beckmann, Anton Setzer

Lab 2 - ScarCoin

9 October 2019

Statement about Academic Integrity By submitting this coursework, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at `https://myuni.swansea.ac.uk/academic-life/academic-misconduct`

Introduction: In the ledger-based model, the central authority – that we call here Scar – receives transactions from users. Your task is to implement the logic used by Scar to check transactions, process them and update the ledger.

You will be provided with the following classes:

- UserAmount.java which specifies a pair consisting of a user (given as a string) and an amount. This represents one arrow going in or out of a transaction.
- UserAmountList.java which specifies a list of inputs or outputs of a transactions. It provides functions to compute the sum of amounts to be used later for checking.
  - UserAmountList.java contains as well a method toLedger() which converts the list into a ledger. This will be used later for checking whether the list of inputs of a transaction can be deducted. It is not enough to check that each individual entry can be deducted in isolation, since there may be several entries for the same user. The solution is to first convert the list of user amounts into a ledger, which tells for each user the sum of amounts to be deducted. Then we check whether the resulting ledger can be deducted.
- A class Transaction.java which provides a transaction consisting of a list of inputs and a list of outputs. It contains as well a method checkTransactionAmountsValid() which checks whether the sum of inputs is greater or equal to the sum of outputs.
- An incomplete version of a class Ledger.java (your task is to complete it), which administrates a ledger. It has
  - Operations for checking and administrating individual balances,

- A method checkLedgerCanBeDeducted to check whether a ledger can be deducted; this ledger will later be obtained from the sum of inputs of a transaction – by first converting it into a ledger before checking we overcome the problem that there might be several entries for the same user, so we need to check that the balance of this user is greater or equal the sum of all these entries.

  Task 1 is to complete this method.
- A method checkUALCanBeDeducted to check whether a list of inputs consisting of users and amounts can be deducted. This is done by first converting this list into a ledger and then checking whether the resulting ledger can be deducted.

  Task 2 is to complete this method.
- Methods addUAL and substractUAL for subtracting and adding a list of user amounts from/to the ledger. This will be used to subtract the inputs from the ledger and then add the outputs.

  Task 3 is to complete this method.
- A Method checkTransactionValid which checks whether a transaction is valid: the sum of outputs needs to be less than or equal the sum of inputs, and the inputs can be deducted from the current ledger.

  Task 4 is to complete this method.
- A Method processTransaction which processes a transaction by deducting the inputs and adding the outputs. A prerequesite is that the transaction is valid (otherwise one might obtain a ledger with negative entries).

  It is Task 5 to complete this method.

Download the environment:

1. Login to your Linux account, open a terminal and switch to your home directory.
2. Enter at prompt:

   ```
   git clone https://github.com/csetzer/CSCM29.git
   ```
3. Change to the subdirectory for this week's lab:

   ```
   cd CSCM29/2019-20/Lab2/
   ```
4. Copy the files in this directory into some other user directory. Doing this will allow you to upate the git repository to the latest version of the code by running

   ```
   git pull
   ```

Note: In the directory for this week's lab (i.e. the Lab2 directory), you will find the following files: UserAmount.java, UserAmountList.java, Transaction.java and a template Ledger.java.

Task 1 - 5: Complete the methods as described above. Note that in some cases the complete method has been commented out so that the whole code compiles. You need to uncomment those methods first.

Task 6: Your last task is to create a test case in Ledger.java, where you can test the following scenario. For each step you want to print a header, the results of any checks done, and the result of the resulting balance:

- Create a ledger for Alice, Bob, Carol, David, initialised with the amount 0 for each user.
- set the balance for Alice to 15,
- set the balance for Bob to 10,
- subtract 5 from the balance of Bob,
- check whether the user amount list giving Alice 10 units, and Alice again 10 units can be deducted,
- check whether the user amount list giving Alice 10 units, and Bob 5 units can be deducted,
- deduct this list from the ledger
- add the list giving Alice twice 10 units to the ledger
- Create a transaction which takes as input for Alice 30 units and gives Bob 5 and Carol 20 units
- Check whether it is valid
- Create a transaction which takes as input for Alice 20 units and gives Bob 5 and Carol 20 units
- Check whether it is valid
- Create a transaction which takes as input for Alice 20 units and gives Bob 5 and Carol 5 units
- Check whether it is valid
- Process the previous transaction
- Create a transaction which takes as input for Alice 5 units, Bob 5 units and gives Carol 5 and David 5 units
- Process that transaction.