Bitcoin and Cryptocurrency Technologies
Assignment 1: ScroogeCoin
TMC-Winter 2018

Group No. : 2

In this assignment we are implementing the scroogecoin functionality with this set of classes:

- Crypto.java
- Transaction.java
- TxHandler.java
- UTXO.java
- UTXOPool.java

Where each class have significant role starting from the trasaction.java where its object and represents the basic transaction in which a transaction which is a set of input values and set of output values.

Transaction.java:

Transaction class that represents a ScroogeCoin transaction and has inner classes Transaction.Output and Transaction.Input

A transaction output consists of a value and a public key to which it is being paid. For the public keys, we use the built-in Java PublicKey class.

A transaction input consists of the hash of the transaction that contains the corresponding output, the index of this output in that transaction (indices are simply integers starting from 0), and a digital signature. For the input to be valid, the signature it contains must be a valid signature over the current transaction with the public key in the spent output.

Crypto.java:

To verify a signature, you will use the verifySignature() method included in the provided file .
public static boolean verifySignature(PublicKey pubKey, byte[] message, byte[] signature)
This method takes a public key, a message and a signature, and returns true if and only signature correctly verifies over message with the public key pubKey .

UTXO.java

UTXO class that represents an unspent transaction output. A UTXO contains the hash of the transaction from which it originates as well as its index within that transaction. We have included equals , hashCode , and compareTo functions in UTXO that allow the testing of equality and comparison between two UTXO s based on their indices and the contents of their txHash arrays.

UTXOpool.java

UTXOPool class that represents the current set of outstanding UTXO s and contains a map from each UTXO to its corresponding transaction output.

TxHandler.java(Explanation of Code)

A transaction consists of a list of inputs, a list of outputs and ID.
In the isValid function, we want to test whether the transaction is a valid transaction or not.

The transaction is valid if the following 5 conditions are satisfied:
(1) all outputs claimed by {@code tx} are in the current UTXO pool,
(2) the signatures on each input of {@code tx} are valid,
(3) no UTXO is claimed multiple times by {@code tx},
(4) all of {@code tx}s output values are non-negative, and
(5) the sum of {@code tx}'s input values is greater than or equal to the sum of its output values; and false otherwise.
Steps in which we have achieved this is :

a. Looping over all the inputs in transaction
b. passing the hash of previous transaction whose output is being used and the output index used in previous transactions so as to get the input UTXO
c. Checking all outputs claimed by {@code tx} are in the current UTXO pool
d. Obtaining output which is actually the output of previous utxo which implies {@code tx} input values
e. Checking the signatures on each input of {@code tx} are valid
f. To check that no UTXO is claimed multiple times by {@code tx}, we maintain a list of SeenUTXOs.
g. Finally we check all of {@code tx}s output values are non-negative and the sum of {@code tx}s input values is greater than or equal to the sum of its output values and false otherwise.

handleTxs() function handles each epoch by receiving an unordered array of proposed transactions, checking each transaction for correctness, returning a mutually valid array of accepted transactions, and updating the current UTXO pool as appropriate.

We achieve this in following steps:

a. If the transaction is valid using the above function, proceed else skip.
b. We form a mutually valid array of accepted transactions by removing the previous UTXO and updating the UTXOPool.
c. This is done by removeUTXO() and addUTXO() functions.
d. Finally we return the valid Transactions in array of type Transaction.

We run our code from the main where we create two demo transaction tx0 and tx1

//after running Test.java //eclipse