EECS 395 BLOCKCHAIN AND CRYPTOCURRENCY

Assignment 1

Problem Description

In this assignment, you will implement the logic to process transactions and produce the ledger. Transactions are organized into time periods or blocks. In each block, you will receive a list of transactions, validate the transactions you receive, and publish a list of validated transactions.

Note that a transaction can reference another in the same block. Also, among the transactions received in a single block, more than one transaction may spend the same output. This would of course be a double-spend, and hence invalid. This means that transactions can't be validated in isolation; it is a tricky problem to choose a subset of transactions that are *together* valid.

You will be provided with a `Transaction` class that represents a coin 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.

More specifically, the raw data that is signed is obtained from the `getRawDataToSign(int index)` method. To verify a signature, you will use the `verifySignature()` method included in the provided file Crypto.java:

```
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`.

Note that you are only given code to verify signatures, and this is all that you will need for this assignment. The computation of signatures is done outside the Transaction class by an entity that knows the appropriate private keys.

A transaction consists of a list of inputs, a list of outputs and a unique ID (see the `getRawTx()` method). The class also contains methods to add and remove an input, add an output, compute digests to sign/hash, add a signature to an input, and compute and store the hash of the transaction once all inputs/outputs/signatures have been added.

You will also be provided with a 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 UTXOs based on their indices and the contents of their txHash arrays.

Further, you will be provided with a `UTXOPool` class that represents the current set of outstanding UTXOs and contains a map from each UTXO to its corresponding transaction output. This class contains constructors to create a new empty `UTXOPool` or a copy of a given `UTXOPool`, and methods to add and remove UTXOs from the pool, get the output corresponding to a given UTXO, check if a UTXO is in the pool, and get a list of all UTXOs in the pool.

You will be responsible for creating a file called `TxHandler.java` that implements the following API:

```java
public class TxHandler {

    /** Creates a public ledger whose current UTXOPool (collection of unspent
     * transaction outputs) is utxoPool. This should make a defensive copy of
     * utxoPool by using the UTXOPool(UTXOPool uPool) constructor.
     */
    public TxHandler(UTXOPool utxoPool);

    /** Returns true if
     * (1) all outputs claimed by tx are in the current UTXO pool,
     * (2) the signatures on each input of tx are valid,
     * (3) no UTXO is claimed multiple times by tx,
     * (4) all of tx’s output values are non-negative, and
     * (5) the sum of tx’s input values is greater than or equal to the sum of
           its output values; and false otherwise.
     */
    public boolean isValidTx(Transaction tx);

    /** 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.
     */
    public Transaction[] handleTxs(Transaction[] possibleTxs);
}
```

Your implementation of `handleTxs()` should return a mutually valid transaction set of maximal size (one that can't be enlarged simply by adding more transactions). It need not compute a set of maximum size (one for which there is no larger mutually valid transaction set).

Extra Credit: Create a second file called MaxFeeTxHandler.java whose handleTxs() method finds a set of transactions with maximum total transaction fees -- i.e. maximize the sum over all transactions in the set of (sum of input values - sum of output values)).

Submission Format

The starter code is given in folder "starterCodeAssignment1". The starter code has satisfied requirement "(1) all outputs claimed by tx are in the current UTXO pool". **Basically what you need to do is to complete function "isValidTx" in "starterCodeAssignment1/assignment1/ TxHandler.java" ands satisfy the other 4 requirements:**

(2) the signatures on each input of tx are valid,
(3) no UTXO is claimed multiple times by tx,
(4) all of tx's output values are non-negative, and
(5) the sum of tx's input values is greater than or equal to the sum of its output values; and false otherwise.

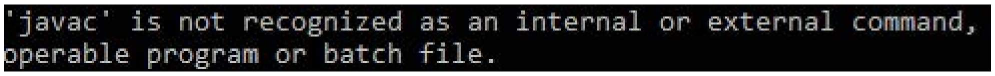
The entire assignment is worth 10 points. You will receive 2 points by satisfying each of the 5 requirements. The extra credit is worth 2 points.

To compile and test your code,

1. Open your command line, and input

```
javac -version
```

If it shows the version number, you can skip to next step. Otherwise, if it shows



```
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

Please install JDK according to
(for windows)
https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_jdk_install.html
(for Mac)
https://docs.oracle.com/javase/8/docs/technotes/guides/install/mac_jdk.html

2. Open your command line, and input

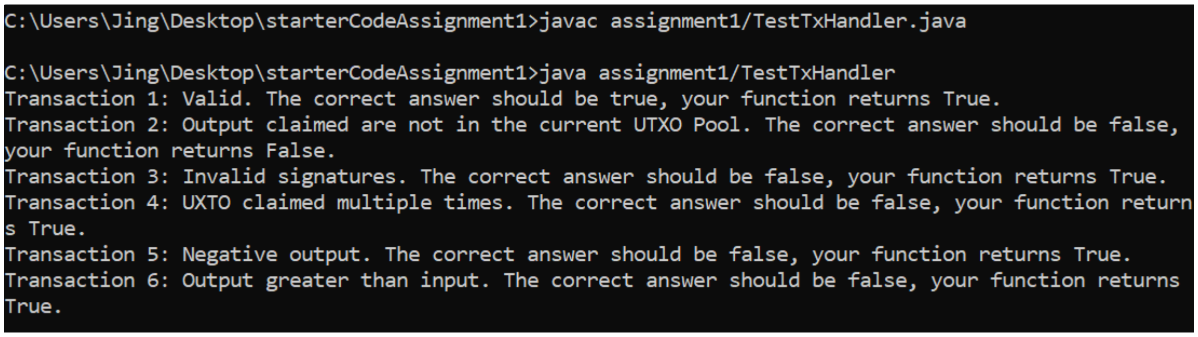
```
java -version
```

If it shows the version number, you can skip to next step. Otherwise, install JRE according to
(for windows)
https://www.java.com/en/download/help/windows_manual_download.xml
(for mac)
https://www.java.com/en/download/help/mac_install.xml

3. Open your command line, navigate to directory "starterCodeAssigment1", and execute

```
javac /assignment1/TestTxHandler.java
java /assignment1/TestTxHandler
```

You will see the result of running 6 test transactions.



```
C:\Users\Jing\Desktop\starterCodeAssignment1>javac assignment1/TestTxHandler.java

C:\Users\Jing\Desktop\starterCodeAssignment1>java assignment1/TestTxHandler
Transaction 1: Valid. The correct answer should be true, your function returns True.
Transaction 2: Output claimed are not in the current UTXO Pool. The correct answer should be false,
your function returns False.
Transaction 3: Invalid signatures. The correct answer should be false, your function returns True.
Transaction 4: UXTO claimed multiple times. The correct answer should be false, your function return
s True.
Transaction 5: Negative output. The correct answer should be false, your function returns True.
Transaction 6: Output greater than input. The correct answer should be false, your function returns
True.
```

Currently the starter code passes 2 out of the 6 test transactions. After filling in "isValidTx" function, you can run

```
javac /assignment1/TestTxHandler.java
java /assignment1/TestTxHandler
```

again, and check whether your code is running correctly with these 6 test transactions.

Passing all these 6 test cases doesn't guarantee getting 10 points, since your work will be graded with a much larger test set.

When submitting, please **rename file "TxHandler.java" (the file you worked on) to "Firstname_Lastname_TxHandler.java" and upload this file (instead of the entire folder) to Canvas.** The "Firstname" and "Lastname" here should be your own first name and last name. For the extra credit problem, please upload it to Canvas as "Firstname_Lastname_MaxFeeTxHandler.java".