THE UNIVERSITY OF SYDNEY

Due: 17 March 2020(Tuesday Week 4) by 11:59 PM

COMP3221 Assignment 1: CS Blockchain

The goal of this project is to implement a Client-Server (CS) Blockchain application in Java which could be able to store messages and detect tampering.

1 Submission Details

The assignment comprises 3 tasks, each can be submitted separately. The final version of your assignment should be submitted electronically via PASTA by 11:59PM on the Tuesday of Week 4 (Hard Deadline). The project is an individual project, and each student has to submit his/her own version.

1.1 Program structure

For task 1, three java files must be submitted. It is recommended to submit by 11:59 PM on the Friday of Week 2 (Soft Deadline).

- A **`Blockchain.java`** file.
- A **`Block.java`** file.
- A **`Transaction.java`** file.

For task 2, four java files must be submitted. It is recommended to submit by 11:59 PM on the Sunday of Week 2 (Soft Deadline).

- A **`Blockchain.java`** file.
- A **`Block.java`** file.
- A **`Transaction.java`** file.
- A **`BlockchainServer.java`** file.

For task 3, five java files must be submitted. It is recommended to submit by 11:59 PM on the Sunday of Week 3 (Soft Deadline).

- A **`Blockchain.java`** file.
- A **`Block.java`** file.

- A **`Transaction.java`** file.
- A **`BlockchainServer.java`** file.
- A **`BlockchainClient.java`** file.

All classes will be stored in the same default package (no package header in java files), and all files should be located in the same **`src`** folder with no subfolders. All present **`.java`** files should be correct and do not forget to remove any dummy files that do not count as source files (E.g. junit test cases, class files). Please zip the **`src`** folder and submit the resulting archive **`src.zip`** by the deadline given above. The program should compile with Java 8. No optional packages that are not part of the default Java 8 JDK can be used.

1.2 Submission system

PASTA will be used to assess that your program correctly implements the Client-Server Blockchain protocol. The archive **`src.zip`** should be submitted at https://soit-app-pro-10.ucc.usyd.edu.au:8443/PASTA/assessments/. To access this website, you will have to be connected to the network of the University of Sydney (physically on campus or through VPN).

PASTA stands for "Programming Assessment Submission and Testing Application" and is a web-based application that automates the compilation, execution and testing of your program. When you submit your **`src.zip`** archive in PASTA, the system enqueues your submission in the shared queue of all assignments to be tested. It may take more time close to the deadline as many students will try to submit at the same time.

1.3 Academic Honesty / Plagiarism

By uploading your submission to PASTA you implicitly agree to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not your work you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

In assessing a piece of submitted work, the School of IT may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

2 Marking

This assignment is worth 10% of your final grade for this unit of study.

The first category of tasks, called *Blockchain: Data Structure*, assesses the quality of the message being stored in the block chain. If you could pass all tests, 4 marks are given.

- Maximum 1 mark for rejecting invalid transactions.
- Maximum 1 mark for storing transaction into pool properly.
- Maximum 2 mark for committing transactions from pool to blockchain correctly.

The second category, called *Blockchain: Server*, assesses the behaviour of the server with respect to protocol requirements. If you could pass all tests, 3 marks are given.

- Maximum 1 mark for server listening to connections continuously.
- Maximum 1 mark for server replying to add transaction request correctly.
- Maximum 1 mark for server replying to print blockchain request correctly.

The third category, called *Blockchain: Client*, assesses the behaviour of the client with respect to protocol requirements. If you could pass all tests, 3 marks are given.

- Maximum 1 mark for client connecting to the server properly.
- Maximum 1 mark for client forwarding requests to the server correctly.
- Maximum 1 mark for client printing replies correctly.

Please make sure previous tasks are implemented correctly, before moving to next task. You may face cascading failures due to improper implementation of previous tasks.

2.1 Feedback

PASTA provides feedback about each individual submission, one at a time. It will output a list of tests and outcomes indicating whether your program passed each visible test successfully or failed. The feedback provided is indicative, intentionally high-level and will not precisely identify the nature of any bug of your program. Please write your personal test cases and thoroughly test all your code before submission.

3 Functionalities of a Client-Server Blockchain

The goal of this project is to implement a Client-Server (CS) Blockchain application. While the original blockchain operates in a distributed and P2P mode, your current CS Blockchain program will only require one client and one server running at the same time. The following tasks indicate the features that should be implemented, one solution for each task is expected to be submitted.

Task 1 Blockchain: Data Structure

In task 1, you are required to build the Blockchain's core data structure, which is a linked list of blocks of transactions. If you would like to have a better view of what we are trying to build, please have a look at Assignment 0.

Transaction. A transaction is defined as the container to store a single message (sender + content).

- Message sender, e.g., test0001
- Message content, e.g., welcome to comp2121!

You should perform some checks before you consider a message as a valid transaction. The message sender must present and should match a unikey-like form (regex: [a-z]{4}[0-9]{4}). The message content cannot have more than 70 English characters or contain a '|' character. (| is used as delimiter later). If a transaction violates those rules, it should be considered as an invalid transaction.

Please use the skeleton code below to implement your Transaction class.

```
1  public class Transaction {
2    private String sender;
3    private String content;
4
5    // getters and setters
6    public void setSender(String sender) { this.sender = sender; }
7    public void setContent(String content) { this.content = content; }
8    public String getSender() { return sender; }
9    public String getContent() { return content; }
10
11   public String toString() {
12     return String.format("|%s|%70s|\n", sender, content);
13   }
14
15   // implement helper functions here if you need any
16 }
```

Block. A block contains a list of transactions and a hash value of the previous block as the pointer:

- a list of transactions contains the messages;
- a hash pointer stores the hash value of the previous block.

The important thing here is to calculate the hash precisely. The hash function is SHA-256. The first thing gets hashed is the previous block's hash value using **`dos.write()`** method, then each transaction in the list gets hashed using the **`dos.writeUTF()`** method. **`writeUTF()`** method expects string as input. Therefore, you should convert each transaction to **`tx|<sender>|<content>`** string format (E.g. tx|test0001|welcome to comp2121!) before writing to **`DataOutputStream`**. If you do not know how to achieve this, please have a look at Assignment 0 of this course. The hashing algorithm is shown as follows.

```
H = SHA-256(H,tx1,tx2,tx3)
```

Please use the skeleton code below to implement your Block class.

```
public class Block {

  private Block previousBlock;
  private byte[] previousHash;
  private ArrayList<Transaction> transactions;

  public Block() { transactions = new ArrayList<>(); }

  // getters and setters
  public Block getPreviousBlock() { return previousBlock; }
  public byte[] getPreviousHash() { return previousHash; }
  public ArrayList<Transaction> getTransactions() { return transactions; }
  public void setPreviousBlock(Block previousBlock) { this.previousBlock = previousBlock; }
  public void setPreviousHash(byte[] previousHash) { this.previousHash = previousHash; }
  public void setTransactions(ArrayList<Transaction> transactions) {
    this.transactions = transactions;
  }

  public String toString() {
    String cutOffRule = new String(new char[81]).replace("\0", "-") + "\n";
    String prevHashString = String.format("|PreviousHash:|%65s|\n",
      Base64.getEncoder().encodeToString(previousHash));
    String hashString = String.format("|CurrentHash:|%66s|\n",
      Base64.getEncoder().encodeToString(calculateHash()));
    String transactionsString = "";
    for (Transaction tx : transactions) {
      transactionsString += tx.toString();
    }
    return "Block:\n"
      + cutOffRule
      + hashString
      + cutOffRule
      + transactionsString
      + cutOffRule
      + prevHashString
      + cutOffRule;
    }

  // to calculate the hash of current block.
  public byte[] calculateHash() {
    // implement your code here.
  }

  // implement helper functions here if you need any.
}
```

Blockchain. A blockchain contains the blockchain itself to store committed transactions and

a pool to store uncommitted transactions. More precisely, it contains:

- a list of uncommitted transactions as a pool;
- the head (latest block) of the blockchain;
- the length of the blockchain.

The blockchain has a **`public int addTransaction(String txString)`** method for the user to add a transaction. The **`txString`** is in **`tx|<sender>|<content>`** format. You need to check each part of the **`txString`** following the standard as suggested before. Only valid **`txString`** is converted into a transaction and then added to the pool. Adding invalid transaction will cause this method to terminate and return 0. The pool should have an upper limit of 3. Once the limit is reached, all uncommitted transactions in the pool are committed to a new block, and that block is added to the blockchain (it becomes the new head of the blockchain). If adding a new transaction causes a new block being generated, the length of the blockchain should increase by 1 and this method should return 2. Otherwise, it returns 1. The root hash value is 0, this means the genesis block's hash pointer has value 0.

Please use the skeleton code below to implement your Blockchain class.

```
1  public class Blockchain {
2
3    private Block head;
4    private ArrayList<Transaction> pool;
5    private int length;
6
7    private final int poolLimit = 3;
8
9    public Blockchain() {
10     pool = new ArrayList<>();
11     length = 0;
12   }
13
14   // getters and setters
15   public Block getHead() { return head; }
16   public ArrayList<Transaction> getPool() { return pool; }
17   public int getLength() { return length; }
18   public void setHead(Block head) { this.head = head; }
19   public void setPool(ArrayList<Transaction> pool) { this.pool = pool; }
20   public void setLength(int length) { this.length = length; }
21
22   // add a transaction
23   public int addTransaction(String txString) {
24     // implement you code here.
25   }
26
27   public String toString() {
28     String cutOffRule = new String(new char[81]).replace("\0", "-") + "\n";
29     String poolString = "";
30     for (Transaction tx : pool) {
```

```
31        poolString += tx.toString();
32      }
33
34      String blockString = "";
35      Block bl = head;
36      while (bl != null) {
37        blockString += bl.toString();
38        bl = bl.getPreviousBlock();
39      }
40
41      return "Pool:\n"
42        + cutOffRule
43        + poolString
44        + cutOffRule
45        + blockString;
46    }
47
48    // implement helper functions here if you need any.
49  }
```

Task 2 Blockchain: Server

The Blockchain server should build a **`ServerSocket`** that keeps listening to a specific port provided by the user. Once a client connects, the server should accept the connection, and pass the **`InputStream`** and **`OutputStream`** of the accepted socket to the **`serverHandler()`** function. **`serverHandler()`** method implements the core logic about how to interact with user requests. Your server should keep listening for connections unless you terminated the process explicitly like **`"kill -9 <processID>"`**. There are three types of requests.

- tx request. "tx|<sender>|<content>"
- pb request. "pb"
- cc request. "cc"

For tx (Transaction) request, the server should try to add the transaction message to the blockchain and reply whether the transaction is in correct format. If the transaction is valid, server should reply with **`"Accepted\n\n"`**. Otherwise, it should reply with **`"Rejected\n\n"`**. Note, since Mac, Windows and Linux all utilizes different newline characters, In this assignment, if there is no special indication, **`print()`** method is used by default rather than **`println()`** method, and we manually add **`"\n"`** when a new line is required.

For pb (Print Blockchain) request, the server should reply with string representation of the blockchain with an additional **`"\n"`**. (blockchain.toString() + **`"\n"`**).

For cc (Close Connection) request, the server should not reply anything and close socket gracefully. (Note, only end the connection, don't terminate the server).

For any other unrecognized request, the server should reply with **"Error\n\n"**.

Please use the skeleton code below to implement your BlockchainServer class.

```
public class BlockchainServer {

  private Blockchain blockchain;

  public BlockchainServer() { blockchain = new Blockchain(); }

  // getters and setters
  public void setBlockchain(Blockchain blockchain) { this.blockchain = blockchain; }
  public Blockchain getBlockchain() { return blockchain; }

  public static void main(String[] args) {
    if (args.length != 1) {
      return;
    }
    int portNumber = Integer.parseInt(args[0]);
    BlockchainServer bcs = new BlockchainServer();

    // implement your code here.
  }

  public void serverHandler(InputStream clientInputStream, OutputStream clientOutputStream) {

    BufferedReader inputReader = new BufferedReader(
      new InputStreamReader(clientInputStream));
    PrintWriter outWriter = new PrintWriter(clientOutputStream, true);

    // implement your code here.

  }

  // implement helper functions here if you need any.
}
```

You should test this part using Linux telnet program as taught in the tutorial before.

For example, if you choose to use command line to compile your code, please issue command **"javac -d bin src/*.java"** for compilation.

To start up your server, please issue command **"java -cp bin BlockchainServer 8333 &"**, and then start a connection to your server with command **"telnet localhost 8333"** in the terminal.

Make sure you can reproduce the transcript below.

```
Trying ::1...
Connected to localhost.
Escape character is '^]'.
```

```
pb
Pool:
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

tx|test0000|1
Accepted

pb
Pool:
--------------------------------------------------------------------------------
|test0000|                                                                    1|
--------------------------------------------------------------------------------

tx|test0000|2
Accepted

pb
Pool:
--------------------------------------------------------------------------------
|test0000|                                                                    1|
|test0000|                                                                    2|
--------------------------------------------------------------------------------

tx|test0000|3
Accepted

pb
Pool:
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Block:
--------------------------------------------------------------------------------
|CurrentHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
--------------------------------------------------------------------------------
|test0000|                                                                    1|
|test0000|                                                                    2|
|test0000|                                                                    3|
--------------------------------------------------------------------------------
|PreviousHash:|                   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
--------------------------------------------------------------------------------

tx|test0000|4
Accepted

pb
Pool:
--------------------------------------------------------------------------------
|test0000|                                                                    4|
--------------------------------------------------------------------------------
```

```
Block:
--------------------------------------------------------------------------------
|CurrentHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
--------------------------------------------------------------------------------
|test0000|                                                                    1|
|test0000|                                                                    2|
|test0000|                                                                    3|
--------------------------------------------------------------------------------
|PreviousHash:|                   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
--------------------------------------------------------------------------------

tx|test0000|5
Accepted

pb
Pool:
--------------------------------------------------------------------------------
|test0000|                                                                    4|
|test0000|                                                                    5|
--------------------------------------------------------------------------------
Block:
--------------------------------------------------------------------------------
|CurrentHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
--------------------------------------------------------------------------------
|test0000|                                                                    1|
|test0000|                                                                    2|
|test0000|                                                                    3|
--------------------------------------------------------------------------------
|PreviousHash:|                   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
--------------------------------------------------------------------------------

tx|test0000|6
Accepted

pb
Pool:
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Block:
--------------------------------------------------------------------------------
|CurrentHash:|                    Lakir/jIQFUGnf+UUnRbiuYsNDOcGXekM+2cKXVmyRw=|
--------------------------------------------------------------------------------
|test0000|                                                                    4|
|test0000|                                                                    5|
|test0000|                                                                    6|
--------------------------------------------------------------------------------
|PreviousHash:|                   jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
--------------------------------------------------------------------------------
Block:
--------------------------------------------------------------------------------
```

```
|CurrentHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
-------------------------------------------------------------------------------
|test0000|                                                                  1|
|test0000|                                                                  2|
|test0000|                                                                  3|
-------------------------------------------------------------------------------
|PreviousHash:|                  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
-------------------------------------------------------------------------------

tx|test0000|7
Accepted

pb
Pool:
-------------------------------------------------------------------------------
|test0000|                                                                  7|
-------------------------------------------------------------------------------
Block:
-------------------------------------------------------------------------------
|CurrentHash:|                    Lakir/jIQFUGnf+UUnRbiuYsNDOcGXekM+2cKXVmyRw=|
-------------------------------------------------------------------------------
|test0000|                                                                  4|
|test0000|                                                                  5|
|test0000|                                                                  6|
-------------------------------------------------------------------------------
|PreviousHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
-------------------------------------------------------------------------------
Block:
-------------------------------------------------------------------------------
|CurrentHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
-------------------------------------------------------------------------------
|test0000|                                                                  1|
|test0000|                                                                  2|
|test0000|                                                                  3|
-------------------------------------------------------------------------------
|PreviousHash:|                  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
-------------------------------------------------------------------------------

tx|wrong|wrong
Rejected

pb
Pool:
-------------------------------------------------------------------------------
|test0000|                                                                  7|
-------------------------------------------------------------------------------
Block:
-------------------------------------------------------------------------------
|CurrentHash:|                    Lakir/jIQFUGnf+UUnRbiuYsNDOcGXekM+2cKXVmyRw=|
-------------------------------------------------------------------------------
```

```
|test0000|                                                                    4|
|test0000|                                                                    5|
|test0000|                                                                    6|
-------------------------------------------------------------------------------
|PreviousHash:|                    jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
-------------------------------------------------------------------------------
Block:
-------------------------------------------------------------------------------
|CurrentHash:|                     jWuaYc5TOawKJew+B+tYuLZT0NDsTo6NDKEJdmgJyBk=|
-------------------------------------------------------------------------------
|test0000|                                                                    1|
|test0000|                                                                    2|
|test0000|                                                                    3|
-------------------------------------------------------------------------------
|PreviousHash:|                    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=|
-------------------------------------------------------------------------------

cc
Connection closed by foreign host.
```

Task 3 Blockchain: Client

The last task is to build the client program. It should try to create a socket to connect to the specific server and port number, and pass socket's **`InputStream`** and **`OutputStream`** to the **`clientHandler()`** method. The **`clientHandler()`** method implements the core logic of the client, which uses a scanner to listen to user inputs, and forward user requests to the server. If the request is **`"cc"`**, the client should terminate. Otherwise, it keeps waiting for new user inputs. All server replies should be correctly printed on the screen. The client program should work exactly the same as the telnet program does, except for lines like

```
Trying ::1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.
```

Please use the skeleton code below to implement your BlockchainClient class.

```
public class BlockchainClient {

  public static void main(String[] args) {

    if (args.length != 2) {
      return;
    }
    String serverName = args[0];
    int portNumber = Integer.parseInt(args[1]);
    BlockchainClient bcc = new BlockchainClient();

    // implement your code here.
```

```
13    }
14
15    public void clientHandler(InputStream serverInputStream, OutputStream serverOutputStream) {
16      BufferedReader inputReader = new BufferedReader(
17        new InputStreamReader(serverInputStream));
18      PrintWriter outWriter = new PrintWriter(serverOutputStream, true);
19
20      Scanner sc = new Scanner(System.in);
21      while (sc.hasNextLine()) {
22        // implement your code here
23      }
24    }
25
26    // implement helper functions here if you need any.
27  }
```

While no code should be shared, note that you are allowed to test with others after you have finished your assignment.