CSIS 294: PROJECT 1
"Griffin Blockchain"

1. SUMMARY

This project will involve the creation of a Blockchain miner that does all the following:

- Receives transactions from user or remote node.
- Sends transactions to a remote node.
- Manages incoming remote node messages using a Queue.
- Generates a Merkle Tree and an eventual Merkle Root as part of Transaction processing.
- Creates Blockchain Blocks using Proof of Work algorithm.

For this project, you will create the main functionality inside of an existing code base called **BlockchainBasics**. The BlockchainBasics code that you will begin with is provided along with this document.

BlockchainBasics, once completed, will allow you to run two instances of this app on two different ports that pass transactions to each other and mine blocks.

This app doesn't save the blocks into a blockchain, but the central functionality you code into this app will be able to be plugged in directly into a larger Blockchain application (the code for this will be provided in the coming weeks) that fully manages a Blockchain, connects to multiple networked nodes, and allows you to trade items on the network.

However, you will only be turning in the BlockchainBasics code as specified at the end of this doc.

This project will involve the following:
- Socket programming
- Queue
- Multithreading
- Hashing: SHA-256
- Merkle Trees
- Blockchain Proof of Work

2. DETAILS

a. Download BlockchainBasics.zip

i. This includes the following classes:
1. BlockchainBasics.java
2. Miner.java
3. Block.java
4. MerkleNode.java
5. BlockchainUtil.java
6. P2PServer.java
7. P2PMessageQueue.java
8. P2PMessage.java
9. P2PUtil.java

b. You only add code where you see this:

```
####################
### ADD CODE HERE ###
####################
```

c. CLASSES where you add code:

d. Miner

i. doProofOfWork method: The logic behind this will be reviewed in class.
   1. This method should first create a string that has as many zeros in it as oBlock.getDifficulty() equals.
      a. Hint: use a while loop here that keeps looping while the length of this string you're creating is less than the target difficulty length.

   2. Then, this method should have another while loop that keeps looping until a valid nonce is found.
      a. There needs to be an if --- else in this while loop.

         i. If bAbortPoW (an instance variable you'll see at the top of the class) equals true, then this means another miner solved the block and this miner received it, so you need to:
            1. Set bAbortPoW back to false.
            2. Print out something to the user like:
               "[miner] Aborted mining block, probably because another confirmed block received."
            3. Return false so the method ends.

         ii. Else perform another attempt at a valid nonce:
            1. The nonce should start at 0 before the while loop and increment each loop.
            2. A valid nonce, after it's added to the block's properties and the block is hashed, should result in a hash that has as many leading zeros as the string you created above (use String's startsWith method for this check).
               a. So remember that on oBlock you have to call:
                  i. Set nonce.
                  ii. Compute hash.
                  iii. Set hash to whatever came back from compute hash.
               b. Then check if hash on oBlock now begins with your leading zeroes string you created above.
                  i. If it does, then return true.

e. Block

i. computeMerkleRoot:

1. Choose which approach you want to take:
   a. The **90% approach** is much easier and is based on what we've done in class but your max score for the project can only reach 90%.
   b. The **100% approach** is much trickier, and you must figure out on your own.

2. **90% Project Score Logic**: This logic needs to account for possibly **2 or 4 items** in lstItems that are passed in.
   a. This means your code should have **2 or 4 Merkle Tree leaves** depending on how many items are passed in.
3. **100% Project Score Logic**: You can only get 90% on this project if everything else is done perfectly. To get 100%, you must make this method flexible to accept any power of 2 items. So lstItems could be 2,4,8,16,32, etc. and your code would be able to compute the Merkle Root.

4. **NOTE**: There is a main method at the bottom of this class that you can use to test 2 and 4 items simply by right clicking on the tab for this class and selecting "Run Block.main()"

ii. populateMerkleNode:
1. This method will work just as we did in class… set the left and right nodes and then call generateHash in the BlockchainUtil class and set the node's hash.

f. **BlockchainUtil**

i. generateHash:
1. This method generates a SHA-256 hash of the string passed in.
2. Code for this was reviewed in class and doesn't need to be changed.

g. **P2PMessageQueue**

i. enqueue:
1. This adds a node to the queue.
2. Code will be reviewed in class.

ii. dequeue:
1. This removes and returns a node from the queue.
2. Code will be reviewed in class.

iii. hasNodes:
1. This returns true or false depending on if any nodes exist in queue.
2. Code will be reviewed in class.

h. P2PUtil

i. connectForOneMessage:
   1. This method will connect to a given server via Socket for a one-time sending of a message to that server and it will return the reply from the server and disconnect.
   2. Code for the Socket communication will be reviewed in class.

3. Submitting a SCREENSHOT.

1. **TAKE A SCREENSHOT** when finished running it without connecting to another miner.
2. **INSTRUCTIONS:**
   a. Run the main method in BlockchainBasics, but when asked…
   "Send transactions to another miner (y,n)?"
   Enter n (instead of y)
   b. Then after entering in transactions and the block gets mined, screenshot the print out so I can see the last lines printed out.
   c. NOTE 1: It doesn't have to show all the print out of the mined block, just the last few lines.
   d. NOTE 2: **IF YOU CAN'T RUN your app, then no need to do the screenshot**.
   e. NOTE 3: In Windows, you can use the "**snipping tool**" (included with Windows)… easiest way to get a screenshot.

---

4. Testing Your Code

Approach 1 (used for screenshot)

a. You can test your code just in the IDE by choosing not to connect to another miner when prompted.
b. This is the only way some of you can test your code, since some run into problems running Approach 2.

Approach 2: Run two instances of this app so that they can communicate with each other.
(This gives you experience with creating a JAR file and running two instances that call each other.)

c. One app can be run from your IDE like you normally do.
d. The other app must be run from a PowerShell window or other command line tool as explained below.

e. **Running a separate instance of your app:**
   i. **Jar file generation**
      1. You first need to build your code into a JAR file.
      2. Instructions are included with the project in canvas and will be reviewed in class.
   ii. Running your JAR file
      1. Navigate to your JAR file in file explorer.
         a. Shortcut if using IntelliJ:
            i. Find it in your left side file explorer in your IntelliJ IDE:
               1. …out > artifacts > BlockchainBasics_jar > BlockchainBasics.jar
            ii. Right click on it and pic "Open in Explorer."
      2. Make sure that no file is highlighted by LEFT clicking in white space.
      3. SHIFT + RIGHT click in white space as shown in class to get extra menu item of "Open PowerShell Window here"
      4. In PowerShell or other command line tool, type the following to run:
         java -jar BlockchainBasics.jar
         (NOTE: if error says **java command not recognized**,

see jar instructions file in canvas project 1 module.)

5. Then make sure a different port is chosen for your IDE instance and your command line instance:
   a. For instance, choose 8000 for one and 8888 for the other.

---

5. **NOT PART OF PROJECT GRADING: Incorporating your code into larger in-class Griffin Blockchain app.**
   a. All of the code you added to this BlockchainBasics project can also now be added to the Griffin Blockchain app to make it functional and partake in full Blockchain networking with others in class.
   b. Details will be provided in class, and this app's code will be made available later.

TURNING IN PROJECT:

IMPORTANT: **DO NOT ZIP** your files please.

- **5 .java files**: Turn in just the **5 .java files** that you added code to.
- **Screenshot**: ALSO, if your code is functional, **include the screenshot.**
- **JAR file**: If you made a runnable **.jar file**, include that as well**.**