Lab 2: 30 pts Group Lab: “MERKLE MADNESS”

Skills covered in this lab that were reviewed in class:

* SHA256 Hashing
* Merkle trees
* Multithreading

**TIP**: Remember, don't be overwhelmed by all the details. Simply focus on one line at a time as the instructions walk you through the creation of the app.

**IMPORTANT GROUP GUIDANCE**:

There are many classes in this lab – one approach is to divide up the classes initially and then share with each other your code in the “Files” section of your group page, and then help each other figure out what parts are correct and who needs help with their code.

Be patient with each other and remember that if you happen to be a group member who is understanding things better than the others than this is an opportunity for you to help the others reach that level of understanding as well – this will only help you in the long run because it makes you think about how to explain things to someone else. This is a normal process that happens on every developer team out in the industry.

**Each member of the group will be including in the group submission a doc with a paragraph** about which part of the code was the most challenging part and why.

**IMPORTANT**: Read bottom of this doc to understand **where to turn things in for a group lab**!

1. **SUMMARY**

\*\*\* REMINDER: You can watch the video accompanying this lab as a reminder of how the game works from a user’s standpoint.

The user first enters in the merkle root of four words the user will try to give to the merkle thread... this merkle root can be manually figured out by the user from the "NOTE" description on the top of page 2.

Then a multithreaded portion of the app allows user submission of Strings on the main thread, and each of those Strings is grabbed by a Merkle or Rogue background thread depending on the user’s timing.

If the Merkle thread gets enough words (4), then it will create a merkle tree and produce a merkle root string – and if it matches the initial merkle root entered by the user, the user wins … if it doesn’t match the user loses.

Meanwhile, every word the rogue thread gets is a “strike.” 3 strikes and the user loses.

The two background threads will sleep a random number of seconds which challenges the user to time their entry just right so that the Merkle thread gets the word and not the Rogue thread.

A third Monitor thread in the background will check on progress and end the app once lost or won.

So it's just a simple game to allow you to get some coding experience with multithreading and Merkle roots.

1. **DETAILS**
   1. You’re creating a console app.
   2. Remember that it’s easier to prompt the user for input using the JOptionPane, which you’ll create a method for near the end in the Util class. This is because if you use System.in from the console text, your typing gets mixed in with the threads waking up and typing stuff to the console window.
   3. Classes
      1. **MerkleManager\_Test**
         1. **AT TOP IN COMMENTS: ALL MEMBER NAMES IN GROUP**
         2. Will have main method and will instantiate instance of MerkleManager and call manage function... that's it.
      2. **MerkleManager**
         1. Instance variables:
            1. Public & static & volatile String for User’s entered word.
            2. Public & static String for user entered expected merkle root.
            3. Public & static String for merkle root set to null.
            4. Public & static int for strikes set to 0.
         2. Method: A public "manage" method and this will be called by the MerkleManager\_Test class's main method.
            1. No inputs or outputs.
            2. Use Util class (that you’ll create later on below) to prompt user for the expected merkle root (The merkle root of the four words the user will enter later as specified down below in the UI Menu Loop).

Put user entered text into the instance variable to hold for end of game comparison.

**NOTE**: Use <https://www.xorbin.com/tools/sha256-hash-calculator> to figure out merkle root of the four words you expect to use just like we do in the code:

EXAMPLE:

H6: Merkle root

/ \

H4 H5

/ \ / \

H0 H1 H2 H3

| | | |

“word1” “word2” “word3” “word4”

if your words were “word1”, “word2”, “word3”, “word4”, then you would enter “word1” into the above web page and get the hash (H0),

Then do the same with the others.

Then combine each pair of hashes, so the hash of “word1” (H0) together with the hash of “word2” (H1), and get the hash of that long string (H4). You’ll do that twice, once for the first two (H0 & H1) and once for the second two (H2 & H3). So you now have H4 & H5.

Finally, then combine those resulting two hashes (H4 & H5) into one string in the web page entry to produce another hash – this is your merkle root hash.

Copy this merkle root from the web page and paste into your user entry when prompted in the console.

* + - * 1. Start 3 separate threads: Instantiate and start new threads for…

MerkleThread, RogueThead, MonitorThread.

* + - * 1. Begin UI question Loop:

Just use while(true) to make an eternal loop (because the monitor will eventually close the app).

Ask user for a word and put it into instance variable for user’s word.

* + - 1. Method: A public & static & synchronized method called grabWord.
         1. No inputs but returns a String.
         2. Puts instance variable of user’s word into a temp String variable and then makes the instance variable null.
         3. Then returns the temp variable.
    1. **MerkleThread**
       1. This class must implement Runnable.
       2. Instance variables:
          1. Public & static & volatile ArrayList<String> that holds all the grabbed words.

Example:

public static volatile ArrayList<String> **lstWords**;

* + - * 1. Private int called **iMerkleTreeInputs** for how many words to wait for before creating a merkle tree.

Set it equal to 4.

* + - 1. Method: A public “run” method that is triggered when “start” is called from MerkleManager.
         1. No inputs or outputs.
         2. Instantiate a Util class object (Util class is defined lower down).
         3. Instantiate the ArrayList instance variable.

Example:

lstWords = new ArrayList();

Create a neverending while loop like this:

while(true){…

In loop, do all the following:

Call sleepRandomTime on util variable.

Cal grabWord on MerkleManager.

Example of how to call static method:

String sNewWord = MerkleManager.grabWord();

Then if sNewWord is not null:

Print out that Merkle grabbed a word.

Add word to lstWords like this: lstWords.add(sNewWord);

Check if lstWords.size() is equal to iMerkleTreeInputs.

And if true, then set MerkleManager.sMerkleRoot to the merkle root generated by the getMerkleRoot method on util class.

* + 1. **RogueThread**
       1. This class must implement Runnable.
       2. Method: A public “run” method.
          1. No inputs or outputs.
          2. Instantiate a Util object.
          3. Create a neverending loop again just like you’ve done before.

Inside loop, do all the following:

Call sleepRandomTime on util variable.

Call grabWord similar to how described in MerkleThread.

If sNewWord is not null:

Increase iStrikes static variable on MerkleManager by 1.

Print out to screen that rogue grabbed a word and mention “STRIKE!”

* + 1. **MonitorThread**
       1. This class must implement Runnable.
       2. Method: “run” method just as in the other Thread classes.
          1. Create endless loop as you’ve done above.

If MerkleManager.sMerkleRoot is not null then…

If the above merkle root equals the initial user-entered merkle root (which you can access the same way as above since they’re both static on MerkleManager):

Then print out “You win: “ followed by the merkle root (which is the above static variable on MerkleManager) and exit the app.

Else if they’re not equal, then tell the user and that the user lost – and exit the app.

To exit:

System.exit(0);

Else if MerkleManager.iStrikes equals 3 then print out “3 strikes: you lost!” or something like that and exit the app as shown in line of code above.

After the if-else statement, then call sleep on util object and sleep for 1 second.

(if you don’t do this, the endless loop never allows updates on that thread to see MerkleManager’s changing values.)

* + 1. **MerkleNode**
       1. Three instance variables:
          1. sHash is a String.
          2. oLeft is a MerkleNode.
          3. oRight is a MerkleNode.
    2. **Util**
       1. This will have helper functions that all the classes can use.
       2. Method: Public method called getMerkleRoot.
          1. Inputs:

ArrayList<String> type for the list of words from MerkleThread.

* + - * 1. Description:

Not much description here since we did this in class.

Creates 7 MerkleNode objects and slowly builds tree to get merkle root.

* + - 1. Method: Private method called populateMerkleNode.
         1. Inputs:

Three MerkleNode types: first is to be populated, second is to be the left node of the first node, and third is to be the right node of the first node.

Desctiption:

Not much description because was reviewed in class.

Basically sets hash, left, and right node values.

* + - 1. Method: generateHash
         1. **NOTE**: This method is given to you on Canvas in this week’s module and in class.
      2. Method: public method called promptUser.
         1. Input: String of question to ask user.
         2. Output: String answer from user.
         3. Description: Use JOptionPane class to get user input so that the user doesn’t have to type while all the other text is being printed out.

Code provided here for getting user input in case you haven’t used JOptionPane before:

//at top of file to allow use of JOptionPane:

import javax.swing.JOptionPane;

//inside class

public String promptUser(String sQuestion){

JOptionPane oQuestion = new JOptionPane();

String sAnswer = oQuestion.showInputDialog(sQuestion);

return sAnswer;

}

* + - 1. Method: public method called sleepRandomTime.
         1. **NOTE**: This whole method is given to you in this week’s module, so you can download it there.
         2. Input: String for thread name.
         3. Output: void.
         4. Description:

Get random int from SecureRandom class between 0 and 5, and then add 3 to it.

Print out how many seconds the thread is going to sleep for.

Call below sleep method passing in the random int (which will range between 3 and 7).

* + - 1. Method: public method called sleep.
         1. Input: int for seconds.
         2. Output: void.
         3. Description:

Make a try-catch statement…

Inside the try, sleep the thread, which means you must multiply the int variable by 1000.

No need to do anything in catch statement, just as demoed in class.

1. **TURNING IN GROUP LAB:** 
   1. DON’T forget to have group member names at top of MekleManager\_Test java file.
   2. You are turning in the following **10/11 files** as a group submission:
      1. **Java files**: All 7 .java files.
      2. **Group Member files**: Each person writes at least a paragraph in a separate doc on which part was the most challenging part of the code for them and why. Each member’s file is included along with the group submission.
   3. IMPORTANT**: DO NOT ZIP** your folders/files and **DO NOT turn in .class files** please.