/\* BlockInputG.java

Name : ROHAN DHOYDA

Version 3.0 2020-02-24, combines BlockingPriorityQueue.

To compile and run (may have to update the file name):

javac -cp "gson-2.8.2.jar" BlockInputG.java

java -cp ".;gson-2.8.2.jar" BlockInputG

RunBlockInput.bat:

java -cp ".;gson-2.8.2.jar" m %1

Example for process two:

> RunBlockInput 2

Author: Clark Elliott, with ample help from the below web sources.

You are free to use this code in your assignment, but you MUST add

your own comments. Leave in the web source references.

This utility program shows one method of reading data into a linked list of unverified blocks from an input data file.

The specific data file / Process ID is determined by argment passed to Java at runtime.

The list is shuffled. Blocks are also written into a priority queue with TimeStamp priority which

demonstrates how the priority queue works.

The shuffled list is marshaled (written) to disk in JSON format.

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import java.io.\*;

/\* CDE: The encryption needed for signing the hash: \*/

**import** com.google.gson.Gson;

**import** com.google.gson.GsonBuilder;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.security.NoSuchAlgorithmException;

// Ah, heck:

// Produces a 64-bye string representing 256 bits of the hash output. 4 bits per character

**import** java.security.MessageDigest; // To produce the SHA-256 hash.

/\* CDE Some other uitilities: \*/

**import** java.util.\*;

**import** **static** java.nio.charset.StandardCharsets.UTF\_8;

//-------------------------------------------------------------------------------

// Firstly we will create a block structure

**class** BlockRecord{

/\* Here I have defined examples of block field\*/

**public** String BlID;

**public** String TSP;

**public** String VPID;

**public** String PrevHash;

**public** String uuid;

String Fname;

String Lname;

String SSNum;

String DOB;

String RandomSeed;

**public** String WinningHash;

String Diag;

String Treat;

String Rx;

// public String data;

**public** **int** blockNumber;

**public** **int** processID;

**public** String hash;

**public** String getBlockID() {**return** BlID;}

**public** **void** setBlockID(String BID){**this**.BlID = BID;}

**public** String getTimeStamp() {**return** TSP;}

**public** **void** setTimeStamp(String TS){**this**.TSP = TS;}

**public** String getVerificationProcessID() {**return** VPID;}

**public** **void** setVerificationProcessID(String VID){**this**.VPID = VID;}

**public** String getPreviousHash() {**return** **this**.PrevHash;}

**public** **void** setPreviousHash (String PH){**this**.PrevHash = PH;}

**public** String getUUID() {**return** uuid;}

**public** **void** setUUID (String ud){**this**.uuid = ud;}

**public** String getLname() {**return** Lname;}

**public** **void** setLname (String LN){**this**.Lname = LN;}

**public** String getFname() {**return** Fname;}

**public** **void** setFname (String FN){**this**.Fname = FN;}

**public** String getSSNum() {**return** SSNum;}

**public** **void** setSSNum (String SS){**this**.SSNum = SS;}

**public** String getDOB() {**return** DOB;}

**public** **void** setDOB (String RS){**this**.DOB = RS;}

**public** String getDiag() {**return** Diag;}

**public** **void** setDiag (String D){**this**.Diag = D;}

**public** String getTreat() {**return** Treat;}

**public** **void** setTreat (String Tr){**this**.Treat = Tr;}

**public** String getRx() {**return** Rx;}

**public** **void** setRx (String Rx){**this**.Rx = Rx;}

**public** String getRandomSeed() {**return** RandomSeed;}

**public** **void** setRandomSeed (String RS){**this**.RandomSeed = RS;}

**public** String getWinningHash() {**return** WinningHash;}

**public** **void** setWinningHash (String WH){**this**.WinningHash = WH;}

}

**public** **class** Blockchain {

**public** **static** **int** PRCID; //we have defined integer variable for process ID

**public** **static** List<BlockRecord> blockchainList = **new** ArrayList<>(); //blockchainList is an instance of the ArrayList class that can hold objects of type BlockRecord

// Each elements of the list is representing a block record which contains all details like data, timestamp, hash value

//We have now set all this variables with an index value

**private** **static** **final** **int** iFNAME = 0; //We have now set all this variables with an index value

**private** **static** **final** **int** iLNAME = 1; //We have set the index LastName Variable to 1

**private** **static** **final** **int** iDOB = 2; //We have set the index Date of Birth Variable to 2

**private** **static** **final** **int** iSSNUM = 3;//We have set the index Social Security Number Variable to 3

**private** **static** **final** **int** iDIAG = 4; //We have set the index DIAG Variable to 4

**private** **static** **final** **int** iTREAT = 5; //We have set the index TREAT Variable to 5

**private** **static** **final** **int** iRX = 6; //We have set the index RX Variable to 6

**private** **static** String FILENAME; // We have defined string variable for FILENAME

**public** **static** **void** main(String[] argv) {

Blockchain.PRCID = Integer.parseInt(argv[0]);

System.out.println("Hello from Process " + Blockchain.PRCID); //Whenever the code will run it will firstly display this message

**if**(PRCID ==0)

{ blockchainList.add(ArtificialBlock(PRCID));}

blockchainList.addAll(Blockchain.createBlocks(Blockchain.PRCID, argv));

Gson gson = **new** GsonBuilder().setPrettyPrinting().create();

// We will firstly convert the Java object to a JSON String:

String json = gson.toJson(blockchainList);

System.out.println("\n JSON String list is: " + json);

// We will now write the JSON object to a file:

**try** (FileWriter writer = **new** FileWriter("myList.json")) {

gson.toJson(blockchainList, writer);

} **catch** (IOException e) {e.printStackTrace();}

}

// I have defined dummy block name as Artificial block

**private** **static** BlockRecord ArtificialBlock(**int** id) {

Date date = **new** Date(); //Class Date is a part of the Java standard library

// Whenever a new Date object is created with no arguments, it represents the current date and time.

BlockRecord BR = **new** BlockRecord(); // This will create a new block record

String T1 = String.format("%1$s %2$tF.%2$tT", "", date);

String TimeStampString = T1 + "." + id; // There will be no timestamp collisions!

BR.setTimeStamp(TimeStampString); // Will be able to priority sort by TimeStamp

String[] tokens;

String suuid = UUID.randomUUID().toString();

BR.setBlockID(suuid);

// In below code we have defined data for dummy block but in our case its an Artificial block

BR.setFname("ROHAN"); // The FNAME will be ROHAN in the dummy block

BR.setLname("DHOYDA"); // The LNAME will be DHOYDA in the dummy block

BR.setSSNum("668-775-869"); // The serial Number will be 668-775-869 in the dummy block

BR.setDOB("03-19-1995"); // The DOB will be 03-19-1995 in the dummy block

BR.setDiag(" Captain America"); // The Diag will be Captain America in the dummy block

BR.setTreat("Avengers"); // The Treat will be Avengers in the dummy block

BR.setRx("scd");

BR.setWinningHash("1721"); // The winning hash will be 1721 in the dummy block

**return** BR;

}

**public** **static** List<BlockRecord> createBlocks(**int** id, String[] args) {

// File file = new File("BlockInput" + id + ".txt");

List<BlockRecord> list = **new** ArrayList<>();

**try** {

**if** (args.length < 1) PRCID = 0;

**else** **if** (args[0].equals("0")) PRCID = 0;

**else** **if** (args[0].equals("1")) PRCID= 1;

**else** **if** (args[0].equals("2")) PRCID = 2;

**else** PRCID = 0; /\* Default for badly formed argument \*/

**int** UBP = 4710 + PRCID;

**int** BlCP = 4820 + PRCID;

System.out.println("Process number: " + PRCID + " Ports: " + UBP + " " +

BlCP + "\n");

**switch**(PRCID){

**case** 1: FILENAME = "BlockInput1.txt"; **break**;

**case** 2: FILENAME = "BlockInput2.txt"; **break**;

**default**: FILENAME= "BlockInput0.txt"; **break**;

}

System.out.println("Using input file: " + FILENAME);

BufferedReader br = **new** BufferedReader(**new** FileReader(FILENAME));

String[] tokens = **new** String[10];

String InputLineStr;

String suuid;

UUID idA;

BlockRecord tempRec;

/\*

int randval = 27; // Just some unimportant initial value

int tenths = 0;

Random r = new Random();

for (int i=0; i<1000; i++){ // safety upper limit of 1000

Thread.sleep(100); // not really work because can be defeated, but OK for our purposes.

randval = r.nextInt(100); // Higher val = more work

System.out.print(".");

if (randval < 1000) { // Lower threshold = more work

tenths = i;

break;

\*/

**while** ((InputLineStr = br.readLine()) != **null**) {

Date date = **new** Date();

BlockRecord BR = **new** BlockRecord(); // Careful

//String T1 = String.format("%1$s %2$tF.%2$tT", "Timestamp:", date);

String T1 = String.format("%1$s %2$tF.%2$tT", "", date);

String TimeStampString = T1 + "." + PRCID; // No timestamp collisions!

// System.out.println("Timestamp: " + TimeStampString);

BR.setTimeStamp(TimeStampString); // Will be able to priority sort by TimeStamp

/\* CDE: Generate a unique blockID. This would also be signed by creating process: \*/

suuid = **new** String(UUID.randomUUID().toString());

BR.setBlockID(suuid);

/\* CDE put the file data into the block record: \*/

tokens = InputLineStr.split(" +"); // Tokenize the input

BR.setFname(tokens[iFNAME]);

BR.setLname(tokens[iLNAME]);

BR.setSSNum(tokens[iSSNUM]);

BR.setDOB(tokens[iDOB]);

BR.setDiag(tokens[iDIAG]);

BR.setTreat(tokens[iTREAT]);

BR.setRx(tokens[iRX]);

String blockRecStr = BR.getBlockID() + BR.getFname() + BR.getLname() +

BR.getSSNum() + BR.getDOB() + BR.getDiag() +

BR.getTreat() + BR.getRx() ;

String hash = calculateBlockHash(blockRecStr);

BR.setWinningHash(hash);

list.add(BR);

}

br.close();

// fr.close();

**return** list;

} **catch** (IOException e) {

System.out.println("An error occurred while reading the file: " + e.getMessage());

}

**return** **null**;

}

String value;

// The block hash calculates the previous blocks SHA256, the current data and a random string that solves the puzzle and that particular block is verified

**public** **static** String calculateBlockHash(String value) {

MessageDigest digest = **null**;

**byte**[] bytes = **null**;

**try** {

digest = MessageDigest.getInstance("SHA-256");

bytes = digest.digest(value.getBytes(UTF\_8));

} **catch** (NoSuchAlgorithmException ex) {

ex.printStackTrace();

}

StringBuffer buffer = **new** StringBuffer();

**for** (**byte** b : bytes) {

buffer.append(String.format("%02x", b));

}

**return** buffer.toString();

}

}