/\*  
 1. Name / Date: satya Yoganand Addala / 26-10-2022  
  
 2. Java version used (java -version), if not the official version for the class: 18.0.2  
  
 3. Precise command-line compilation examples / instructions:  
  
 > javac Blockchain.java  
  
 4. Files Used for this process :  
  
 > BlockInput0.txt  
 > BlockInput1.txt  
 > BlockInput2.txt  
  
 5. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java -cp ".:gson-2.8.2.jar" Blockchain.java 0  
 > java -cp ".:gson-2.8.2.jar" Blockchain.java 1  
 > java -cp ".:gson-2.8.2.jar" Blockchain.java 2  
  
  
 All acceptable commands are displayed on the various consoles.  
  
 6. External Libraries Used : GSON Jar- gson-2.8.2.jar  
  
 7. Referenced Programs from Clark Elliott :  
  
 > Blockchain utilities sample program Version J  
 > Blockchain input utilty program,  
 > Sample Work Program  
 > Process Coordination  
  
 8. Notes:  
  
 \* The Blockchain.java file holds various classes and methods that helps to execute the BlockChain  
 \* This file runs three processes in parallel where are currently i am using three input files namely BlockInput0.txt,BlockInput1.txt,BlockInput2.txt.  
 \* Now the process ID will take by the process from commandline and sets up all the required ports with respect to processID.  
 \* Then all the process waits for process 2 to run in which once the process2 Starts running it will multicast a start message to all the running processes.  
 \* Then the other process starts to fetch all the three processes public keys and it will try to read there Input files based on their processID's.  
 \* Once it has been done then all the records present in the files are converted into unverified blocks and are added to the priority Queue for further processing.  
 \* All the Unverified blocks will be shared among all the three processes.  
 \* Then the Work will come into play which induces a simple puzzle to solve and helps to verify the unverified blocks and add them into a blockchain.  
 \* Now at the end of the process user will be given an opportunity to analyze the statistics of the processes by giving an user Input which inturn will give the processes records and credits for all the three processes.  
  
  
 9. Extra Notes :  
  
 The sleep time has been incremented to 10000 milliseconds because when trying to start the second process the first process is not able to pick the pub,ic key which has been settled using increased sleep time  
  
 10.Referenced Web Sources :  
  
 https://mkyong.com/java/how-to-parse-json-with-gson/  
 http://www.java2s.com/Code/Java/Security/SignatureSignAndVerify.htm  
 https://www.mkyong.com/java/java-digital-signatures-example/  
 https://javadigest.wordpress.com/2012/08/26/rsa-encryption-example/  
 https://www.programcreek.com/java-api-examples/index.php?api=java.security.SecureRandom  
 https://www.mkyong.com/java/java-sha-hashing-example/  
 https://stackoverflow.com/questions/19818550/java-retrieve-the-actual-value-of-the-public-key-from-the-keypair-object  
 https://www.java67.com/2014/10/how-to-pad-numbers-with-leading-zeroes-in-Java-example.html  
 https://www.javacodegeeks.com/2013/07/java-priority-queue-priorityqueue-example.html  
  
  
\*/  
  
import com.google.gson.Gson;  
import com.google.gson.GsonBuilder;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.security.\*;  
import java.security.spec.X509EncodedKeySpec;  
import java.util.\*;  
  
  
public class Blockchain {  
  
 /\* ServerName Set to Localhost \*/  
 static String *serverName* = "localhost";  
  
 /\* Total Processes that are being used \*/  
 static int *numProcesses* = 3;  
  
 /\* Initialize Process ID \*/  
 static int *PID* = 0;  
  
 /\* BlockList to hold records of unverified blocks \*/  
 static List<BlockRecord> *blockArr* = new ArrayList<BlockRecord>();  
 /\* Variable to read the file name that is picked with respect to pid \*/  
 private static String *fileName*;  
  
 /\* Initializing an alpha numeric string for creating a random seed \*/  
 private static final String *ALPHA\_NUMERIC\_STRING* = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  
 /\* BlockList to hold records of all the verified blocks \*/  
 public static LinkedList<BlockRecord> *blockChain* = new LinkedList<>();  
 /\* List to hold all the publicKeys with the list Object as PublicKeyObject where the public key and processID has been stored \*/  
 public static final List<PublicKeyObject> *publicKeyArray* = new ArrayList<>();  
 /\* Variable to store private key \*/  
 public static PrivateKey *privateKey*;  
 /\* Start variable to begin the process and is set to default as wait \*/  
 public static String *start* = "wait";  
 /\* Setting first name Index\*/  
 private static final int *fName\_Index* = 0;  
 /\* Setting last name Index \*/  
 private static final int *lName\_Index* = 1;  
 /\* Setting DateOfBirth Index \*/  
 private static final int *dOB\_Index* = 2;  
 /\* Setting SSN Index \*/  
 private static final int *iSSNUM\_Index* = 3;  
 /\* Setting Diag Index \*/  
 private static final int *iDIAG\_Index* = 4;  
 /\* Setting Treatment Index \*/  
 private static final int *treatment\_Index* = 5;  
 /\* Setting iRX Index \*/  
 private static final int *iRX* = 6;  
  
 /\* Creating a publicKeyObject of the type PublicKeyObject to use it in the process \*/  
 public static PublicKeyObject *publicKeyObj* = new PublicKeyObject();  
 public static Comparator<BlockRecord> *blockTSComparator* = new Comparator<BlockRecord>()  
 {  
 static Queue<BlockRecord> *blockPriorityQueue* = new PriorityQueue<>(4, *blockTSComparator*);  
 /\* Overriding the default compare method to do the compare task with respect to timestamps \*/  
 @Override  
 public int compare(BlockRecord r1, BlockRecord r2)  
 {  
 /\* Fetching the time stamps for each block to compare against time \*/  
 String t1 = r1.getTimeStamp();  
 String t2 = r2.getTimeStamp();  
 /\* If both time stamps are equal then return 0 \*/  
 if (t1 == t2) {return 0;}  
 /\* If either t1 or t2 are null then return -1 0r 1 respectively \*/  
 if (t1 == null) {return -1;}  
 if (t2 == null) {return 1;}  
 /\* return the standard compare method with respect to the timestamps \*/  
 return t1.compareTo(t2);  
 }  
 };  
 /\* Setting up the Priority Queue \*/  
 static Queue<BlockRecord> *blockPriorityQueue* = new PriorityQueue<>(4, *blockTSComparator*);  
  
 /\* Hashing the whole block to set the winning hash \*/  
 public static String hashBlock(String blockContents){  
 /\* Setting up the hash Algorithm to SHA-256 \*/  
 String hashAlgorithm = "SHA-256";  
 String SHA256String = "";  
  
 try{  
 /\* Message Digest Class has been used to get an instance of hash Algorithm \*/  
 MessageDigest md = MessageDigest.*getInstance*(hashAlgorithm);  
 md.update (blockContents.getBytes());  
 /\* Converting the contents to bytes \*/  
 byte byteData[] = md.digest();  
  
 /\* Initializing a new String Buffer Class which is thread safe \*/  
 StringBuffer sb = new StringBuffer();  
 /\* Looping on the hash bytes \*/  
 for (int i = 0; i < byteData.length; i++) {  
 /\* Appending the hash values into the string buffer \*/  
 sb.append(Integer.*toString*((byteData[i] & 0xff) + 0x100, 16).substring(1));  
 }  
 /\* Converting appended string buffer hex value to String \*/  
 SHA256String = sb.toString();  
  
 } catch (NoSuchAlgorithmException x) {  
 /\* Printing Stack Trace \*/  
 x.printStackTrace();  
 }  
 /\* Returning the Converted String \*/  
 return SHA256String.toUpperCase();  
  
 }  
 /\* Method for creating the Initial block with dummy data \*/  
 public static LinkedList<BlockRecord> blockChainInit(){  
 /\* Initializing a linked list to hold the blockrecords \*/  
 LinkedList<BlockRecord> blockRecord = new LinkedList<>();  
 /\* Initializing first block \*/  
 BlockRecord block0 = new BlockRecord();  
 /\* Generating the UniqueId with random UUID method and converting it to string \*/  
 String unique\_ID = UUID.*randomUUID*().toString();  
 /\* Setting the blockID for block0 with uniqueId that has been generated \*/  
 block0.setBlockID(unique\_ID);  
 /\* Setting the block number to initial block with is 0 \*/  
 block0.setBlockNum(0);  
 /\* Making the thread to sleep for 1001 milliseconds \*/  
 try{  
 Thread.*sleep*(1001);  
 }  
 catch(InterruptedException ex){  
 /\* Printing Stack Trace \*/  
 ex.getStackTrace();  
 }  
 /\* Initializing the date Variable \*/  
 Date date = new Date();  
 /\* Generating the time stamp with respect to the date variable, and it generates something like year-month-date.hour:minutes:seconds \*/  
 String timeStamp = String.*format*("%1$s %2$tF.%2$tT", "", date);  
 /\* Concatinating the ProcessID with the generated timestamp\*/  
 String timeStampString = timeStamp + "." + Blockchain.*PID*;  
 /\* Setting the TimeStamp to block0 \*/  
 block0.setTimeStamp(timeStampString);  
 /\* Setting the verification ProcessID to 0\*/  
 block0.setVerificationProcessID("0");  
 /\* Setting the previous hash to a random value \*/  
 block0.setPreviousHash("0000");  
 /\* Setting the FirstName,LastName and DateOfBirth to match my details \*/  
 block0.setFname("Satya Yoganand"); block0.setLname("Addala"); block0.setDOB("05-22-1998");  
 /\* Setting some dummy SSN Number \*/  
 block0.setSSNum("123-45-6789");  
 /\* Setting the diag to Blockchain Initial Block \*/  
 block0.setDiag("Blockchain ");  
 /\* Setting RX to Blockchain pills \*/  
 block0.setRx("Blockchain");  
 /\* Setting Treat to Writing more blockChain code \*/  
 block0.setTreat("BlockChain");  
 /\* Setting some random seed \*/  
 block0.setRandomSeed("03L810QS");  
 /\* Combining all the block record data into a single string \*/  
 String combinedData = block0.getTimeStamp() + block0.getBlockNum() + block0.getBlockID() + block0.getSignedID() + block0.getPreviousHash() + block0.getFname() + block0.getLname() + block0.getDOB() + block0.getSSNum() + block0.getVerificationProcessID() + block0.getDiag() + block0.getTreat() + block0.getRx() + block0.getTimeStamp();  
 /\* Generating the block data with the help of random seed \*/  
 String blockData = combinedData + "03L810QS";  
 /\* Setting the Winning hash with the block data \*/  
 block0.setWinningHash(*hashBlock*(blockData));  
 /\* Adding this block0 to blockRecord \*/  
 blockRecord.add(block0);  
 System.*out*.println("Collected Data from Initial Block ........");  
 return blockRecord;  
 }  
 /\* Method for generating public keys for each process \*/  
 public static PublicKeyObject publicKeyInit(int pid) throws Exception {  
  
 /\* Initializing the random variable with the help of Random method \*/  
 Random random = new Random();  
 long randomNum = random.nextInt(1000);  
 /\* Generating the Key Value Pair with the help of randomNum \*/  
 KeyPair keyPair = *generateKeyPair*(randomNum);  
 /\* Fetching the private key through the generated KeyPair \*/  
 *privateKey* = keyPair.getPrivate();  
 System.*out*.println("Key Pair : "+ keyPair);  
 /\* Fetch the public key from the key value pair,and store it in the bytePublicKey variable of the type byte array \*/  
 byte[] bytePublicKey = keyPair.getPublic().getEncoded();  
 /\* By using base64 encoding convert the public key which is in byte array to a string \*/  
 String stringKey = Base64.*getEncoder*().encodeToString(bytePublicKey);  
 /\* Set the converted string to the public key variable in publicKeyObj Class \*/  
 *publicKeyObj*.setPublicKey(stringKey);  
 /\* Set the ProcessId of the process to ProcessId variable in publicKeyObj Class \*/  
 *publicKeyObj*.setProcessID(pid);  
 System.*out*.println("Printing Public Key : "+*publicKeyObj*);  
 /\* Returning the publickey object \*/  
 return *publicKeyObj*;  
 }  
  
 public static boolean verifySig(byte[] data, PublicKey key, byte[] sig) throws Exception {  
 /\* Getting an Instance of Signature Class and using SHA1withRSA Algorithm \*/  
 Signature signer = Signature.*getInstance*("SHA1withRSA");  
 /\* Initializing the data to get signed with the help of public key \*/  
 signer.initVerify(key);  
 signer.update(data);  
 /\* Returning the status of verifying \*/  
 return (signer.verify(sig));  
 }  
  
 /\* Method for generating public and private keys \*/  
 public static KeyPair generateKeyPair(long seed) throws Exception {  
 /\* Fetching a new Instance of KeyPairGenerator and using RSA Algorithm \*/  
 KeyPairGenerator keyGenerator = KeyPairGenerator.*getInstance*("RSA");  
 /\* Fetching a new instance of SHA1PRNG to generate random values \*/  
 SecureRandom rng = SecureRandom.*getInstance*("SHA1PRNG", "SUN");  
 /\* Setting the random seed \*/  
 rng.setSeed(seed);  
 /\* Initializing the key generator with random seed \*/  
 keyGenerator.initialize(1024, rng);  
 /\* Returning the generated Key Pair \*/  
 return (keyGenerator.generateKeyPair());  
 }  
  
 /\* Method for Signing the winning hash \*/  
 public static byte[] signData(byte[] data, PrivateKey key) throws Exception {  
 /\* Initializing the Signature Instance of SHA1withRSA Algorithm \*/  
 Signature signer = Signature.*getInstance*("SHA1withRSA");  
 /\* Initializing signer Object for Signing \*/  
 signer.initSign(key);  
 /\* This updates the data to be Signed \*/  
 signer.update(data);  
 return (signer.sign());  
 }  
 /\* Method for getting randomSeed based on the count variable \*/  
 public static String randomAlphaNumeric(int count) {  
 /\* Initializing String Builder to new for getting different random seeds for different processes \*/  
 StringBuilder builder = new StringBuilder();  
 /\* Running loop based on count so that the random seed has the same number of characters \*/  
 while (count-- != 0) {  
 /\* fetching single character from the Alpha numeric string \*/  
 int character = (int)(Math.*random*()\**ALPHA\_NUMERIC\_STRING*.length());  
 /\* Appending each character to the builder String \*/  
 builder.append(*ALPHA\_NUMERIC\_STRING*.charAt(character));  
 }  
 /\* Returning the random Seed \*/  
 return builder.toString();  
 }  
 public static BlockRecord Work(BlockRecord blockRecord){  
 /\* Initializing the randomSeed to null \*/  
 String randomSeed = "";  
 /\* Initializing dataSeed to null \*/  
 String dataSeed = "";  
 /\* Initializing hash to null \*/  
 String hash = "";  
 /\* Initializing workNumber to 0 \*/  
 int workerNum = 0;  
 /\* Setting the previous Hash with that of the Winning hash of the previous block \*/  
 blockRecord.setPreviousHash(Blockchain.*blockChain*.get(0).getWinningHash());  
 /\* Setting the block number such that it will always be one number higher than that of the previous block number \*/  
 blockRecord.setBlockNum(Blockchain.*blockChain*.get(0).getBlockNum() + 1);  
 /\* Setting verification ProcessID to the current process \*/  
 blockRecord.setVerificationProcessID(Integer.*toString*(Blockchain.*PID*));  
 /\* Concatenating all the above values and stored it in a string \*/  
 String blockData = blockRecord.getTimeStamp() + blockRecord.getBlockNum() + blockRecord.getBlockID() + blockRecord.getSignedID() + blockRecord.getPreviousHash() + blockRecord.getFname() + blockRecord.getLname() + blockRecord.getDOB() + blockRecord.getSSNum() + blockRecord.getVerificationProcessID() + blockRecord.getDiag() + blockRecord.getTreat() + blockRecord.getRx() + blockRecord.getTimeStamp();;  
 try {  
 while (true) {  
 /\* Calling the randomAlphaNumeric Method and setting count to 8 to generate a random seed \*/  
 randomSeed = *randomAlphaNumeric*(8);  
 /\* Combining the BlockData with that of random seed to generate dataSeed \*/  
 dataSeed = blockData + randomSeed;  
 /\* Initializing a message digest algorithm Instance \*/  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 /\* Hashing the dataSeed into bytes and storing ot on a variable \*/  
 byte[] bytesHash = md.digest(dataSeed.getBytes("UTF-8"));  
 /\* Converting the hashbytes to hex using stringBuilder \*/  
 StringBuilder stringBuilder = new StringBuilder();  
 /\* Looping through the bytes \*/  
 for ( byte b : bytesHash) {  
 /\* Appending each byte by converting it using String format to the stringBuilder \*/  
 stringBuilder.append(String.*format*("%02x", b));  
 }  
 hash = stringBuilder.toString();  
 /\* Fetching the first four digits by using hexadecimal number system of the converted hash and store it in a variable \*/  
 System.*out*.println("Hash is: " + hash);  
  
 workerNum = Integer.*parseInt*(hash.substring(0,4),16);  
 System.*out*.println("Printing the first four digits from hash in decimal along with the hex value : " + hash.substring(0,4) +" and " + workerNum);  
 /\* Checking for the puzzle condition that if the worknumber is less than 200000 then the puzzle is solved and the block gets verified. \*/  
 if (!(workerNum < 20000)){  
 System.*out*.format("%d is not less than 20,000 so we did not solve the puzzle\n\n", workerNum);  
 }  
 if (workerNum < 20000){  
 System.*out*.println("block verified! with the value : "+ workerNum);  
 /\* Setting the random seed to the verified block \*/  
 blockRecord.setRandomSeed(randomSeed);  
 /\* Setting the winning hash for the verified block \*/  
 blockRecord.setWinningHash(hash);  
 /\* Signing the Winning hash with the help of private key \*/  
 byte[] signedWinHash = *signData*(bytesHash, *privateKey*);  
 /\* Converting the signed winning hash to string using base64 encoder \*/  
 String signedWinHashStr = Base64.*getEncoder*().encodeToString(signedWinHash);  
 /\* Setting the SignedWinningHash for the current verified block \*/  
 blockRecord.setSignedWinningHash(signedWinHashStr);  
 break;  
 }  
 /\* Looping through blockchain to verify whether any unverified blocks are added to blockchain by any other process \*/  
 for (BlockRecord b: *blockChain*){  
 /\* Condition to check for any matched BlockID's \*/  
 if (b.getBlockID().equals(blockRecord.getBlockID())){  
 /\* If there is any match then the block should be abandoned \*/  
 System.*out*.println("Abandoning block...");  
 /\* Creating a new block record for abandoned Block \*/  
 BlockRecord abandonedBlock = new BlockRecord();  
 /\* Setting the BlockID of that block to Abandoned \*/  
 abandonedBlock.setBlockID("Abandoned");  
 /\* Returning the abandoned block \*/  
 return abandonedBlock;  
 }  
 }  
 /\* Inducing sleep to do work\*/  
 try{  
 Thread.*sleep*(7001);  
 }catch(InterruptedException exec) {  
 throw new RuntimeException(exec);  
 }  
 }  
 }catch(Exception exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 /\* Return a verified block \*/  
 return blockRecord;  
 }  
  
 /\* Method for multicasting publicKeys over different processes \*/  
 public void multiCastPublicKey(PublicKeyObject publicKey) {  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a PrintStream Object \*/  
 PrintStream toServer;  
 /\* Initializing the gson variable with the help of gson library \*/  
 Gson gson = new GsonBuilder().create();  
  
 /\* Fetching PublicKey and converting it to json Object \*/  
 String JSON = gson.toJson(publicKey);  
 try{  
 /\* Looping all the processes \*/  
 for(int i=0; i<*numProcesses*; i++){  
 /\* Creating a new connection to each port here we are using the port number which is spaced out based on the processID \*/  
 /\* Creating a new Socket \*/  
 sock = new Socket(*serverName*, (Ports.*publicKeyServerBase* + (i)));  
 /\* Setting up the Output Stream \*/  
 toServer = new PrintStream(sock.getOutputStream());  
 /\* Sending the public key with that of processID in jSON Format \*/  
 toServer.println(JSON);  
 /\* Flushing the data to server \*/  
 toServer.flush();  
 }  
 }catch (Exception exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace ();  
 }  
 }  
  
 /\* Method for multicasting start message to all the running processes \*/  
 public void multiCastStart(){  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a print stream object \*/  
 PrintStream toServer;  
 try{  
  
 /\* Looping for all the processes \*/  
 for(int i=0; i<*numProcesses*; i++){  
 /\* Creating a new connection to each port here we are using the port number which is spaced out based on the processID \*/  
 sock = new Socket(*serverName*, (Ports.*StartServerBase* + (i)));  
 /\* Setting the Output Print Stream \*/  
 toServer = new PrintStream(sock.getOutputStream());  
 /\* Sending the Start Message to all the other processes \*/  
 toServer.println("go");  
 /\* Flushing the data to server \*/  
 toServer.flush();  
 }  
 }catch (Exception exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace ();  
 }  
 }  
  
 /\* Method for multicasting unverified blocks to all the processes \*/  
 public void multiCastUB(BlockRecord blockRecord, int serverBase){  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a PrintStream Object \*/  
 PrintStream toServer;  
  
 try{  
 /\* Initializing new gson variable \*/  
 Gson gson = new GsonBuilder().create();  
 /\* Converting the BlockRecord object to json object \*/  
 String jSon = gson.toJson(blockRecord);  
 /\* Looping for all the processes \*/  
 for(int i=0; i< *numProcesses*; i++){  
 /\* Creating a new Socket connection with the port number which is spaced out based on ProcessID \*/  
 sock = new Socket(*serverName*, (serverBase + (i)));  
 /\* Setting up Output Print Stream \*/  
 toServer = new PrintStream(sock.getOutputStream());  
 /\* Sending the JSON Object along with the processID over all the processes \*/  
 toServer.println(jSon);  
 /\* Flushing the data to server \*/  
 toServer.flush();  
 }  
  
 }catch (Exception exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace ();  
 }  
 }  
  
 /\* Method for Multicasting BlockChain to all the other processes \*/  
 public void multiCastBC(LinkedList<BlockRecord>bc, int serverBase){  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a PrintStream Object \*/  
 PrintStream toServer;  
  
 try{  
 /\* Converting the blockchain record into a JSon using the method blockchain\_JSon\_Converter \*/  
 String jSon = *blockChain\_JSon\_Converter*(bc);  
 /\* Loop for all the running processes \*/  
 for(int i=0; i< *numProcesses*; i++){  
 /\* Creating a new Socket connection with the port which is based on the processID \*/  
 sock = new Socket(*serverName*, (serverBase + (i)));  
 /\* Setting up an Output PrintStream \*/  
 toServer = new PrintStream(sock.getOutputStream());  
 /\* Send the JSON to all the other processes \*/  
 toServer.println(jSon);  
 /\* Flushing the data to server \*/  
 toServer.flush();  
 }  
  
 }catch (Exception exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace ();  
 }  
 }  
  
 /\* Method for reading an INPUT File and Converting it to block record \*/  
 public static List<BlockRecord> readFile(int pid){  
 /\* Record list of the type BlockRecord to contain the records in the file \*/  
 List<BlockRecord> recordList = new ArrayList<>();  
  
 /\* Switch Condition to pick the appropriate file based on ProcessID \*/  
 switch(pid){  
 /\* Read BlockInput1 for processID 1 \*/  
 case 1: *fileName* = "BlockInput1.txt";  
 break;  
 /\* Read BlockInput2 for processID 2 \*/  
 case 2: *fileName* = "BlockInput2.txt";  
 break;  
 /\* Read BlockInput3 for processID 0 \*/  
 default: *fileName*= "BlockInput0.txt";  
 break;  
 }  
  
 System.*out*.println("Processing BlockInput"+pid+".txt File...");  
 /\* Buffered Reader to go through the contents of the file \*/  
 try {  
 BufferedReader br = new BufferedReader(new FileReader(*fileName*));  
  
 String[] tokens;  
 String inputLineStr;  
 String unique\_ID;  
  
 /\* Variable to keep track of the records present in a file \*/  
 int recordCount = 0;  
  
 while ((inputLineStr = br.readLine()) != null) {  
 /\* Creating new Record for each line in the file \*/  
 BlockRecord blockRecord = new BlockRecord();  
 /\* Initializing a thread sleep so that it will give some time to read the data of each line \*/  
 try{  
 Thread.*sleep*(1001);  
 }catch(InterruptedException exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 /\* Creating a new date variable to generate the timestamp \*/  
 Date date = new Date();  
 /\* Generating the TimeStamp using string format \*/  
 String T1 = String.*format*("%1$s %2$tF.%2$tT", "", date);  
 /\* Adding the processID to the generated TimeStamp \*/  
 String TimeStampString = T1 + "." + pid;  
 /\* Setting the timeStamp for the blockRecord \*/  
 blockRecord.setTimeStamp(TimeStampString);  
 /\* Setting the recordCount to the BlockNumber \*/  
 blockRecord.setBlockNum(recordCount);  
  
 /\* Generating a UniqueID using randomUUID call \*/  
 unique\_ID = new String(UUID.*randomUUID*().toString());  
 /\* Setting the UniqueID to BlockID \*/  
 blockRecord.setBlockID(unique\_ID);  
 /\* Signing the UniqueID with the privateKey \*/  
 byte[] digitalSignature = *signData*(unique\_ID.getBytes(), *privateKey*);  
 /\* Converting the Signed Variable to String using Base64 Encoding \*/  
 String SignedSHA256ID = Base64.*getEncoder*().encodeToString(digitalSignature);  
 /\* Setting the Signed ID \*/  
 blockRecord.setSignedID(SignedSHA256ID);  
 /\* Splitting the contents of the line and setting up the data \*/  
 tokens = inputLineStr.split(" +");  
 /\* Setting the firstname \*/  
 blockRecord.setFname(tokens[*fName\_Index*]);  
 /\* Setting the lastname \*/  
 blockRecord.setLname(tokens[*lName\_Index*]);  
 /\* Setting SSN Number \*/  
 blockRecord.setSSNum(tokens[*iSSNUM\_Index*]);  
 /\* Setting dateOfBirth \*/  
 blockRecord.setDOB(tokens[*dOB\_Index*]);  
 /\* Setting diag \*/  
 blockRecord.setDiag(tokens[*iDIAG\_Index*]);  
 /\* Setting Treat \*/  
 blockRecord.setTreat(tokens[*treatment\_Index*]);  
 /\* Setting Rx \*/  
 blockRecord.setRx(tokens[*iRX*]);  
 /\* Fetching VerificationID \*/  
 blockRecord.setVerificationProcessID(Integer.*toString*(Blockchain.*PID*));  
 /\* Add the block record to the recordList \*/  
 recordList.add(blockRecord);  
 /\* Incrementing the record Count \*/  
 recordCount++;  
 }  
 } catch (Exception exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 /\* Return the unverified blocks \*/  
 return recordList;  
 }  
  
 /\* Method for write the verified blocks to a file \*/  
 public static void writeToFile(LinkedList<BlockRecord> bc){  
 /\* For printing the Json in a readable format we use gson pretty \*/  
 Gson gsonPretty = new GsonBuilder().setPrettyPrinting().create();  
 /\* The JSON Write variable initialized to open square bracket as the Json starts with this \*/  
 String JSONWrite = "[";  
 /\* Looping each block in block Chain \*/  
 for (BlockRecord block: bc){  
 JSONWrite += gsonPretty.toJson(block);  
 /\* Adding a comma separator between each block \*/  
 if (bc.indexOf(block) != bc.size() - 1)  
 JSONWrite += ",";  
 }  
 /\* Setting the closing square bracket at the end of each block \*/  
 JSONWrite = JSONWrite + "]";  
 /\* Writing all the data to a file \*/  
 try (FileWriter writer = new FileWriter("BlockchainLedger.json", false)) {  
 writer.write(JSONWrite);  
 } catch (IOException e) {  
 /\* Printing Stack Trace \*/  
 e.printStackTrace();  
 }  
 }  
  
 /\* Method for converting BlockRecord object to a json Object \*/  
 public static String blockChain\_JSon\_Converter(LinkedList<BlockRecord> bc){  
 /\* Created a new GsonBuilder Instance \*/  
 Gson gson = new GsonBuilder().create();  
 /\* Initializing the Json Object to open Square bracket \*/  
 String jSON = "[";  
 /\* looping through block record \*/  
 for (BlockRecord blockRecord: bc){  
 jSON += gson.toJson(blockRecord);  
  
 /\* Adding commas between the values \*/  
 if (bc.indexOf(blockRecord) != bc.size() - 1)  
 jSON += ",";  
 }  
 /\* Closing the Json with a closed Square Bracket \*/  
 jSON = jSON + "]";  
 /\* Return the converted Json \*/  
 return jSON;  
 }  
  
 public static void main(String args[]) throws Exception {  
  
 /\* Checking for the Arguments that are received from command line \*/  
 if (args.length < 1)  
 *PID* = 0;  
 else if (Integer.*parseInt*(args[0]) > 2){  
 System.*out*.println("Process numbers are 0, 1, or 2");  
 throw new IllegalArgumentException();  
 }  
 else  
 *PID* = Integer.*parseInt*(args[0]);  
  
 System.*out*.println("Satya Yoganand's Blockchain program. Ctl-c to quit\n");  
 System.*out*.println("Using processID " + *PID* + "\n");  
  
  
 /\* Setting up ports for the current process with respect to processID \*/  
 new Ports().setPorts();  
  
 /\* Creating a blockChain new Variable \*/  
 Blockchain blockChain1 = new Blockchain();  
 /\* Initializing the publicKey with respect to porocess ID \*/  
 PublicKeyObject publicKey = *publicKeyInit*(*PID*);  
  
 System.*out*.println("PublicKey for processID : "+*PID*+" ----- "+ publicKey.getPublicKey());  
  
 /\* Generating a new Start Server Thread \*/  
 new Thread(new StartServer()).start();  
 /\* Generating a new PublickeyServer Thread \*/  
 new Thread(new PublicKeyServer()).start();  
 /\* Generating a new UnverifiedBlockServer Thread \*/  
 new Thread(new UnverifiedBlockServer()).start();  
 /\* Generating a new BlockChainServer Thread \*/  
 new Thread(new BlockchainServer()).start();  
 System.*out*.println("All the Servers have been set, waiting to receive start message");  
 /\* If the processID is 2 then multicast the start message to all the running processes \*/  
 if (*PID* == 2){  
 new Blockchain().multiCastStart();  
 }  
 /\* Inducing sleep such that the start message will be multicasted then all the processes start doing the blockchain work \*/  
 System.*out*.println("Before Start Status : "+ *start*);  
 try{Thread.*sleep*(12000);} catch(InterruptedException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 /\* After receiving start message the blockChain process will Start \*/  
  
 if (*start*.equals("go")){  
 /\* Multicasting the public key \*/  
 blockChain1.multiCastPublicKey(publicKey);  
 /\* Inducing sleep so that all the processes will receive all the publicKeys \*/  
 try{Thread.*sleep*(12000);}catch(InterruptedException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 /\* Printing all the public keys on the console \*/  
 String publicKeyProcess = "";  
 for (PublicKeyObject pubK: *publicKeyArray*){  
 publicKeyProcess = "ProcessID "+pubK.getProcessID() + " Using : " + pubK.getPublicKey();  
 System.*out*.println(publicKeyProcess);  
 }  
 System.*out*.println("----------------------------------------------");  
 /\* Initializing blockchain with a dummy block data \*/  
 *blockChain* = *blockChainInit*();  
 /\* Read the files with respect to the processID and store the blocks in a block array \*/  
 *blockArr* = *readFile*(*PID*);  
 /\* Looping through blockArray so that all the blocks will be marked as Unverified and then added to priority Queue for further processing \*/  
 for (BlockRecord block: *blockArr*)  
 new Blockchain().multiCastUB(block, Ports.*UnverifiedBlockServerPortBase*);  
 System.*out*.println("Unverified Blocks sent to Priority Queue...");  
 /\* Inducing sleep so that all the process will fetch all their unverified blocks \*/  
 try{Thread.*sleep*(4000);}catch(InterruptedException e){  
 /\* Printing Stack Trace \*/  
 e.printStackTrace();  
 }  
  
 /\* Waiting to accept all the connections \*/  
 while (true){  
 /\* Inducing sleep on the thread so that the work will be done on each unverified block \*/  
 try{Thread.*sleep*(2001);}catch(InterruptedException e){  
 /\* Printing Stack Trace \*/  
 e.printStackTrace();  
 }  
  
 System.*out*.println(*blockPriorityQueue*.size() + " unverified blocks remaining");  
 /\* Checking for the items in priorityQueue if null then break the loop \*/  
 BlockRecord tempBlock = *blockPriorityQueue*.poll();  
 if (tempBlock == null)  
 break;  
 BlockRecord verifiedBlock = new BlockRecord();  
 /\* Setting up a flag for block which refers whether the block is present in the blockchain or not \*/  
 boolean blockExists = false;  
 /\* Initializing a temporary public key variable \*/  
 String tempPubKey = "";  
 for (PublicKeyObject pub: *publicKeyArray*){  
 if (Integer.*toString*(pub.getProcessID()).equals(tempBlock.getVerificationProcessID())){  
 /\* Fetch the public key for the current process and store it in the temp variable \*/  
 tempPubKey = pub.getPublicKey();  
 System.*out*.println("Using the public key from process: " + pub.getProcessID());  
 }  
 }  
 /\* Converting this public key into bytes \*/  
 byte[] publicKeyInBytes = Base64.*getDecoder*().decode(tempPubKey);  
 /\* converting the signed blockID into bytes to get ready for verification \*/  
 byte[] idSignature = Base64.*getDecoder*().decode(tempBlock.getSignedID());  
 /\* converting the bytes of the public key into a public key object reference \*/  
 X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(publicKeyInBytes);  
 KeyFactory keyFactory = KeyFactory.*getInstance*("RSA");  
 PublicKey RestoredKey = keyFactory.generatePublic(publicKeySpec);  
 /\* Now by using verify Sig method with the key and both byte arrays, verify if it is signed \*/  
 boolean verified = *verifySig*(tempBlock.getBlockID().getBytes(), RestoredKey, idSignature);  
 if(!verified){  
 System.*out*.println("This block is not signed by the correct owner of the private key, moving on...");  
 }  
 else {  
  
 try{Thread.*sleep*(1000);}catch(InterruptedException e){}  
 /\* Condition for whether the block exists in the blockchain \*/  
 for (BlockRecord b: *blockChain*){  
 if (b.getBlockID().equals(tempBlock.getBlockID())){  
 blockExists = true;  
 System.*out*.println("Block already in blockchain");  
 }  
 }  
 /\* If the block is not present in the blockchain then start verifying it \*/  
 while (!blockExists){  
 System.*out*.println("Attempting to verify block");  
 /\* performing work for the current record \*/  
 verifiedBlock = *Work*(tempBlock);  
 /\* Condition to check if the blockchain was changed \*/  
 String previousHash = *blockChain*.get(0).getWinningHash();  
 /\* If the block is an abandoned block then process the next block in the PriorityQueue \*/  
 if (verifiedBlock.getBlockID().equals("Abandoned"))  
 break;  
 if (!(verifiedBlock.getBlockID().equals("Abandoned"))){  
 /\* if the blockChain was not changed then add it to the blockchain \*/  
 if (verifiedBlock.getPreviousHash().equals(previousHash)){  
 System.*out*.println("Block has been verified and added to the blockchain and started multicasting to all the other running processes");  
 /\* Multicasting the modified blockchain to all the other processes \*/  
 *blockChain*.addFirst(verifiedBlock);  
 /\* Multicasting the modified blockchain to all the other processes \*/  
 blockChain1.multiCastBC(*blockChain*, Ports.*BlockchainServerPortBase*);  
 /\* Change the status of the blockExists to true \*/  
 blockExists = true;  
 }  
 /\* If the block is modified \*/  
 else {  
 /\* Checking to see again if the block is in the blockchain, if it exists then start processing the next block in PriorityQueue \*/  
 for (BlockRecord b: *blockChain*){  
 if (b.getBlockID().equals(verifiedBlock.getBlockID())){  
 blockExists = true;  
 }  
 }  
 System.*out*.println("processing to rework on block again...");  
 }  
 }  
 }  
 }  
 }  
 /\* The program is done with processing all the unverified blocks and adding them into the blockChainlist is complete \*/  
 System.*out*.println("BLOCKCHAIN COMPLETE");  
 }  
 }  
}  
class Ports{  
 /\* Setting the StartServerBase to 4600 \*/  
 public static int *StartServerBase* = 4600;  
  
 /\* Setting the UnverifiedBlockServerPortBase to 4820 \*/  
 public static int *UnverifiedBlockServerPortBase* = 4820;  
 /\* Setting the BlockchainServerPortBase to 4930 \*/  
 public static int *BlockchainServerPortBase* = 4930;  
  
 /\* Setting the publicKeyServerBase to 4710 \*/  
 public static int *publicKeyServerBase* = 4710;  
 public static int *publicKeyServerPort*;  
 public static int *StartServerPort*;  
 public static int *UnverifiedBlockServerPort*;  
 public static int *BlockchainServerPort*;  
 /\* Methods to set ports for all the running processID's \*/  
 public void setPorts(){  
 /\* Setting the StartServerPort to StartServerBase incremented with processID \*/  
 *StartServerPort* = *StartServerBase* + (Blockchain.*PID*);  
 /\* Setting the publicKeyServerPort to publicKeyServerBase incremented with processID \*/  
 *publicKeyServerPort* = *publicKeyServerBase* + (Blockchain.*PID*);  
 /\* Setting the UnverifiedBlockServerPort to UnverifiedBlockServerBase incremented with processID \*/  
 *UnverifiedBlockServerPort* = *UnverifiedBlockServerPortBase* + (Blockchain.*PID*);  
 /\* Setting the BlockChainServerPort to BlockChainServerBase incremented with processID \*/  
 *BlockchainServerPort* = *BlockchainServerPortBase* + (Blockchain.*PID*);  
 }  
}  
  
class BlockRecord{  
 /\* Block ID for the current Block \*/  
 String BlockID;  
 /\* Block Number for the current Block \*/  
 int blockNum;  
 /\* Time Stamp for the block \*/  
 String TimeStamp;  
 /\* VerificationID for the process \*/  
 String VerificationProcessID;  
 /\* PreviousHash for the block which is the winning hash of previous block \*/  
 String PreviousHash;  
 /\* Unique ID \*/  
 UUID uuid;  
  
 /\* First Name for the current Block \*/  
 String fName;  
  
 /\* Last Name for the current Block \*/  
 String lName;  
  
 /\* SSN for the current Block \*/  
 String SSNum;  
 /\* DateOfBirth for the current Block \*/  
 String DOB;  
 /\* Random Seed is our guess \*/  
 String RandomSeed;  
 String winningHash;  
 String signedID;  
 String signedWinningHash;  
 /\* Creating an object for diag \*/  
 String Diag;  
 /\* Creating an object for Treatment \*/  
 String Treat;  
 /\* Creating an object for Rx \*/  
 String Rx;  
  
 /\* Initializing setters and getters for all the variables defined \*/  
 public String getBlockID() {return BlockID;}  
 public void setBlockID(String BID){this.BlockID = BID;}  
  
 public int getBlockNum() {return blockNum;}  
 public void setBlockNum(int blockNum){this.blockNum = blockNum;}  
  
 public String getTimeStamp() {return TimeStamp;}  
 public void setTimeStamp(String TS){this.TimeStamp = TS;}  
  
 public String getVerificationProcessID() {return VerificationProcessID;}  
 public void setVerificationProcessID(String VID){this.VerificationProcessID = VID;}  
  
 public String getPreviousHash() {return this.PreviousHash;}  
 public void setPreviousHash (String PH){this.PreviousHash = PH;}  
  
 public UUID getUUID() {return uuid;}  
 public void setUUID (UUID ud){this.uuid = ud;}  
  
 public String getSignedID() {return signedID;}  
 public void setSignedID (String signedID){this.signedID = signedID;}  
  
 public String getLname() {return lName;}  
 public void setLname (String lName){this.lName = lName;}  
  
 public String getFname() {return fName;}  
 public void setFname (String fName){this.fName = fName;}  
  
 public String getSSNum() {return SSNum;}  
 public void setSSNum (String ssNum){this.SSNum = ssNum;}  
  
 public String getDOB() {return DOB;}  
 public void setDOB (String dOB){this.DOB = dOB;}  
  
 public String getDiag() {return Diag;}  
 public void setDiag (String diag){this.Diag = diag;}  
  
 public String getTreat() {return Treat;}  
 public void setTreat (String treat){this.Treat = treat;}  
  
 public String getRx() {return Rx;}  
 public void setRx (String rx){this.Rx = rx;}  
  
 public String getRandomSeed() {return RandomSeed;}  
 public void setRandomSeed (String randomSeed){this.RandomSeed = randomSeed;}  
  
 public String getWinningHash() {return winningHash;}  
 public void setWinningHash (String winningHash){this.winningHash = winningHash;}  
  
 public String getSignedWinningHash() {return signedWinningHash;}  
 public void setSignedWinningHash (String signedWinningHash){this.signedWinningHash = signedWinningHash;}  
  
  
}  
  
class PublicKeyObject {  
 String publicKey;  
 int processID;  
 /\* Initializing get method for public key \*/  
 public String getPublicKey(){return this.publicKey;}  
 /\* Initializing set method for public key \*/  
 public void setPublicKey(String pk){this.publicKey = pk;}  
 /\* Initializing get method for processID \*/  
 public int getProcessID(){return this.processID;}  
 /\* Initializing set method for processID \*/  
 public void setProcessID(int id){this.processID = id;}  
}  
  
/\* Worker thread for processing incoming public keys \*/  
class PublicKeyWorker extends Thread {  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Initializing a new gson variable \*/  
 Gson gson = new Gson();  
 PublicKeyWorker (Socket s) {sock = s;}  
 public void run(){  
 try{  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Storing the received input \*/  
 String input = in.readLine ();  
  
 System.*out*.println("Public Key Received: " + input);  
 /\* Fetching the publicKey,ProcessID from PublicKeyObject using from JSon \*/  
 PublicKeyObject publicKey = gson.fromJson(input, PublicKeyObject.class);  
 /\* Adding the obtained publicKeyObject to publicKeyArray \*/  
 Blockchain.*publicKeyArray*.add(publicKey);  
 System.*out*.println("PublicKeyArray : "+Blockchain.*publicKeyArray*);  
 /\* Closing the Socket \*/  
 sock.close();  
 } catch (IOException ex){  
 /\* Printing Stack Trace \*/  
 ex.printStackTrace();  
 }  
 }  
}  
  
class PublicKeyServer implements Runnable {  
 /\* Allowed number of connections from client to server at the same time \*/  
 int q\_len = 6;  
 Socket sock;  
  
 public void run(){  
  
 try{  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 ServerSocket servSock = new ServerSocket(Ports.*publicKeyServerPort*, q\_len);  
 /\* Accepting all the connections \*/  
 while (true) {  
 /\* This is an accept method for listening from the client \*/  
 sock = servSock.accept();  
 /\* This Start method will invoke the run method in PublicKeyWorker class \*/  
 new PublicKeyWorker (sock).start();  
 }  
 }catch (IOException ioExec) {  
 /\* Printing Stack Trace \*/  
 ioExec.printStackTrace();  
 }  
 }  
}  
  
class UnverifiedBlockWorker extends Thread {  
  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a new gson Object \*/  
 Gson gson = new Gson();  
 public UnverifiedBlockWorker(Socket s){  
 this.sock = s;  
 }  
 public void run(){  
  
 try{  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Blocking Call - the server code will pauses at this point and starts listening to the input from client \*/  
 String input = in.readLine();  
 /\* Converting it to BlockRecord object which holds all the Unverified blocks \*/  
 BlockRecord recordData = gson.fromJson(input, BlockRecord.class);  
 /\* Adding the block to priorityQueue \*/  
 Blockchain.*blockPriorityQueue*.add(recordData);  
 }catch(IOException exec){  
 /\* Printing Stack Trace \*/  
 System.*out*.print(exec);  
 }  
 }  
}  
  
class UnverifiedBlockServer implements Runnable {  
 /\* Allowed number of connections from client to server at the same time \*/  
 int q\_len = 6;  
 /\* Create a Socket Object \*/  
 Socket sock;  
  
 public void run() {  
 try {  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 ServerSocket servSock = new ServerSocket(Ports.*UnverifiedBlockServerPort*, q\_len);  
 /\* Accepting all the connections \*/  
 while (true) {  
 /\* This is an accept method for listening from the client \*/  
 sock = servSock.accept();  
 /\* This Start method will invoke the run method in UnverifiedBlockWorker class \*/  
 new UnverifiedBlockWorker(sock).start();  
 }  
 } catch (IOException exec) {  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 }  
}  
  
class BlockChainWorker extends Thread {  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\* Create a new gson Object \*/  
 Gson gson = new Gson();  
 /\* Constructor to assign the incoming sock to the local variable \*/  
 BlockChainWorker (Socket s) {sock = s;}  
 public void run(){  
  
 try{  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Blocking Call - the server code will pauses at this point and starts listening to the input from client \*/  
 String input = in.readLine();  
 /\* Converting the String input to array of BlockRecord through fromJson \*/  
 BlockRecord[] blockRecordData = gson.fromJson(input, BlockRecord[].class);  
 /\* Empty the Blockchain list first \*/  
 Blockchain.*blockChain*.clear();  
 /\* Now Adding all the blocks to blockchain \*/  
 for (BlockRecord record: blockRecordData){  
 Blockchain.*blockChain*.add(record);  
 }  
 /\* If the process 0 hears any new update it will rewrite it to a file \*/  
 if (Blockchain.*PID* == 0){  
 /\* Write method will be called to update the blockchain \*/  
 Blockchain.*writeToFile*(Blockchain.*blockChain*);  
 }  
 } catch (IOException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 }  
}  
  
class BlockchainServer implements Runnable {  
 /\* Allowed number of connections from client to server at the same time \*/  
 int q\_len = 6;  
 /\* Create a Socket Object \*/  
 Socket sock;  
  
 public void run(){  
 try{  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 ServerSocket servSock = new ServerSocket(Ports.*BlockchainServerPort*, q\_len);  
 /\* Accepting all the connections \*/  
 while (true) {  
 /\* This is an accept method for listening from the client \*/  
 sock = servSock.accept();  
 /\* This Start method will invoke the run method in BlockChainWorker class \*/  
 new BlockChainWorker(sock).start();  
 }  
 }catch(IOException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 }  
}  
  
class StartWorker extends Thread {  
 /\* Create a Socket Object \*/  
 Socket sock;  
 /\*constructor for assigning incoming connection to a local variable \*/  
 public StartWorker(Socket s){  
 this.sock = s;  
 }  
 public void run(){  
 try{  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Setting up the Global start variable in BlockChain class as per the incoming output typically "go" \*/  
 Blockchain.*start* = in.readLine();  
  
 }catch(IOException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 }  
}  
  
class StartServer implements Runnable {  
 /\* Allowed number of connections from client to server at the same time \*/  
 int q\_len = 6;  
 /\* Create a Socket Object \*/  
 Socket sock;  
  
 public void run(){  
 try{  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 ServerSocket servSock = new ServerSocket(Ports.*StartServerPort*, q\_len);  
 /\* Accepting all the connections \*/  
 while (true) {  
 /\* This is an accept method for listening from the client \*/  
 sock = servSock.accept();  
 /\* This Start method will invoke the run method in StartWorker class \*/  
 new StartWorker(sock).start();  
 }  
 }catch(IOException exec){  
 /\* Printing Stack Trace \*/  
 exec.printStackTrace();  
 }  
 }  
}

[{  
 "BlockID": "61ded3de-bf41-421c-8263-fc9d5fc7d35d",  
 "blockNum": 12,  
 "TimeStamp": " 2022-11-01.23:09:23.2",  
 "VerificationProcessID": "0",  
 "PreviousHash": "177c84a43b713a30417529598f674bcd95ebcd06e577623641e32c62c22a5fc2",  
 "fName": "Joe",  
 "lName": "DiMaggio",  
 "SSNum": "111-22-3333",  
 "DOB": "1914.11.25",  
 "RandomSeed": "AHMIDHOH",  
 "winningHash": "3a8a8f91e1cb86ca68495f8c53dc1c7a005e1e91627c6b42a343305823b1af74",  
 "signedID": "izDdNyBcI7YR2C0e6XbVBFvT+KCjNvwdIWdNRT6Qv7k3asrXz6yHbng39TtHSiEzdSvFrNg8yiU2ABu4Kf2bbTx/qIkmvwpTakjmtmYMvsIMaGnncBb8L976WSEFAxzUQMgv6ytKFNTD1OJfjRrWnK41gdWHXTPSwrNmkt0vlVc\u003d",  
 "signedWinningHash": "Lxvf4CgKu2y6OUBTZarMriNWNHp049O2xyJFmAALgrjoTyX+fuGSGNlnSe4al0B+bJyzUcCsTo7OoTSG06orIc6a7c+UMTeOzXqZPFwA1mia3e4QhXXgVA4taL2k0vzaEvXeOrL2aGJaW425ia08MLWomV2N3dGnejaRwkzUWqI\u003d",  
 "Diag": "SoreKnees",  
 "Treat": "RestFromSports",  
 "Rx": "Aspirin"  
},{  
 "BlockID": "82978e3d-3bf9-4a80-b727-b5845d25439b",  
 "blockNum": 11,  
 "TimeStamp": " 2022-11-01.23:09:22.2",  
 "VerificationProcessID": "1",  
 "PreviousHash": "467b45ae3408d6d8862d5a870d55a24b8662baf7a77e9f554979936db02b7d74",  
 "fName": "John",  
 "lName": "Kennedy",  
 "SSNum": "333-45-6999",  
 "DOB": "1917.05.29",  
 "RandomSeed": "QTGIXS8Y",  
 "winningHash": "177c84a43b713a30417529598f674bcd95ebcd06e577623641e32c62c22a5fc2",  
 "signedID": "Xdshk93gOoi+8oGYi/FQ1VYaCnfuIWuA9fa2fNaKUlVMGdevB89ATzhU6ucehd/rI++B0kE9mBj2eXyJF7Q8xS8sRWFQdBR3KPvWRnHpHmbGvMJtLm+2nnFRk0L+qSYm3no0aRYVV9w0PcEzw9CcxgoYPPR6EkcSYCFH5vXb/4Q\u003d",  
 "signedWinningHash": "bqbt1E8HmkzpiaIzFOkPYyf8nTDOGQjXInpHjKPu5hMD03KKRPEvgpKBwHc0JGPjoTu5c+92t+FHOMeAKioKM63WfyayCBOpvzL/uhPrlqogZ4iMMhQs5q3Xx+FQQIxgTjtm6Pf9hveX/n9C53D4kHYWq/GaXECLE8XmeKvJj00\u003d",  
 "Diag": "AddisonsDisease",  
 "Treat": "DrugTherapy",  
 "Rx": "Steroids"  
},{  
 "BlockID": "f1ca2f0b-2c2a-4cb1-87cd-532a1dcdb15a",  
 "blockNum": 10,  
 "TimeStamp": " 2022-11-01.23:09:21.2",  
 "VerificationProcessID": "0",  
 "PreviousHash": "4770d6271025d66c423a4532f60f7890b1c42765dc238feff45caf1780253099",  
 "fName": "Abraham",  
 "lName": "Lincoln",  
 "SSNum": "444-45-6888",  
 "DOB": "1809.02.12",  
 "RandomSeed": "QQUOKHD3",  
 "winningHash": "467b45ae3408d6d8862d5a870d55a24b8662baf7a77e9f554979936db02b7d74",  
 "signedID": "JmAvszAWJzKuTOnajoo8s2yY7fi0POF6fTLgj0D1M9FWSwTDRHnRbNoXVLHPQEWitXnxXwNT7y25cpQ/wr7zd9ALFIH3/iJyzIMN6T5uhuwbNNlcqAEyop11mGf4wp+DKCXbS9JYwRvrXwSM2oqqFHrr1QtVCuBrvA1AxVC9Egg\u003d",  
 "signedWinningHash": "QcNf7As4b97MvyN+aJTd/wVgk4DJjFsseHbCJupec4ZDZN9+PY33gPHZro4FJqkETe/7pfRaXmdti4MRo+AA6aT+S0wBoB6ZbnfvQ+9hpRTXSFkB0PDk5DYBRdxvkdHNUIuuRLP6w2YJ4vskt7CS90okn/YA1DFE6Yr4WFiDdzo\u003d",  
 "Diag": "GreviousWound",  
 "Treat": "Surgery",  
 "Rx": "Whiskey"  
},{  
 "BlockID": "f79f23bf-aa6c-432f-b16d-f6cdab358ac1",  
 "blockNum": 9,  
 "TimeStamp": " 2022-11-01.23:09:20.2",  
 "VerificationProcessID": "1",  
 "PreviousHash": "1bf7e0f877d5643ad9283c6102c4e9d37a5176dabb961dd38653bb04cabc8ce7",  
 "fName": "Helen",  
 "lName": "Keller",  
 "SSNum": "666-45-6789",  
 "DOB": "1880.06.27",  
 "RandomSeed": "48EM8FBZ",  
 "winningHash": "4770d6271025d66c423a4532f60f7890b1c42765dc238feff45caf1780253099",  
 "signedID": "aDcEks95+i6p8qITnTTPvqZSqpqsGw6bHEf6BaRRKwwh3/yB7JKPv7MEM71AXFSCUZBdRd1KzTso5CoXZ0FUCSp4TOBGmaqK9plA/QpApenFNNdyVLTCfIsb6UZH93TeGrdoNGVMe8oTW4Uj84HVI/rOX/Je9Ruqee9B0TyLoj0\u003d",  
 "signedWinningHash": "W9/LWHjEr2ZQKFFaT7WEbxsFHrBPDzKfPNrdWtbbUfpBVaUoKxxIJjeNN6GIFWsBkNdvbQz77tEBxWSPvC61lUszuF8Gxw+p+C1b5LaeqkxyqBNZjtuIwbIQl05+wTw0kFj1lRIej29TR5ywokPajhYhF10+mNFjJDYYl/vRteQ\u003d",  
 "Diag": "Arthritis",  
 "Treat": "WarmCloths",  
 "Rx": "Aspirin"  
},{  
 "BlockID": "8610b059-1abd-4b3f-866a-123920fce4e9",  
 "blockNum": 8,  
 "TimeStamp": " 2022-11-01.23:09:20.1",  
 "VerificationProcessID": "2",  
 "PreviousHash": "06a03e20618588528d4a6ce460a06960741cbb8c1c2f67534fadf7e895c41e3b",  
 "fName": "Bruce",  
 "lName": "Lee",  
 "SSNum": "456-789-123",  
 "DOB": "1940.11.27",  
 "RandomSeed": "7C9OQ6L5",  
 "winningHash": "1bf7e0f877d5643ad9283c6102c4e9d37a5176dabb961dd38653bb04cabc8ce7",  
 "signedID": "DA8ISXe8Z0v3iPl4HVeyYlL/qAmdB6ErAaA87w24w/b5Bhfu8G843Yy7CznoVF657/RAnpmgYF9bUPPJJ6tdPFpxVH/XWLKnXHgxFYpVF7SBH6ns9+ni1xl1oyBGxm9PshD1bOuUKmHfve578eATLGuYRxVK4vh5bZv5FYrKvRI\u003d",  
 "signedWinningHash": "kDjgQAg5hbICo0n1J2yISCuzdy29r5k04Ca0a6YdVGrrmihF+b2Zo4c7u+4fYn1u2J3BhEeccbXfklUVEGREjDcy8TVnl3hRbLJQH7IVzhQchdY5RLp295oQwdcOIoIxmAetaRnjUl1G2ChLItYOJpEBIF06WdNhoBr+tsQfsLg\u003d",  
 "Diag": "SoreStomach",  
 "Treat": "LessCombat",  
 "Rx": "Vicodine"  
},{  
 "BlockID": "09e352ea-e662-4774-bca1-45285ccc62af",  
 "blockNum": 7,  
 "TimeStamp": " 2022-11-01.23:09:19.1",  
 "VerificationProcessID": "0",  
 "PreviousHash": "029abc9c8a9058b4ba500cff14f410a1741593e39faf83447d874c0b5dde0c9b",  
 "fName": "Sally",  
 "lName": "McCutty",  
 "SSNum": "123-456-999",  
 "DOB": "1970.01.01",  
 "RandomSeed": "LQ50RQ6R",  
 "winningHash": "06a03e20618588528d4a6ce460a06960741cbb8c1c2f67534fadf7e895c41e3b",  
 "signedID": "YCHkpmapdoDHuzP9+nun5yX7R9VDG70nBluAnMOpjVdqyIxuNFpzZfZS19FXyYaSoqdlILhEqewLpxBwkO3angI6CaqlvKtlWl5fJw9pxHWjm6UhmoEYVel3n267tRu5pz5VQg0jh5j1Okg3deCq0nnqdXKQUMIpcmG0w0gXRIM\u003d",  
 "signedWinningHash": "cOp5P5M71sKRSx5YvyZzH2qYe3H4AvbwyCL0Ro6220Egwq0i/+7Ym0wRxPi1F9npylxV660pHiPEbWsKzlnykQtOJXThXIPgrx/nklfQUFxRLqnOtzEz43bNgAidhpUH5Bw8qVdI5cotOvIThVfs4Gy0fHtzpOZD8Sav0pQYqLA\u003d",  
 "Diag": "Migraine",  
 "Treat": "IcePack",  
 "Rx": "Almotriptan"  
},{  
 "BlockID": "8cf263ec-1924-46f2-bce4-123c30bd3622",  
 "blockNum": 6,  
 "TimeStamp": " 2022-11-01.23:09:18.1",  
 "VerificationProcessID": "1",  
 "PreviousHash": "44ffe1ff1ac0def92edae8ff58fef5444be04c04fcd9c07deba40098b03ffae4",  
 "fName": "Wei",  
 "lName": "Xu",  
 "SSNum": "123-456-333",  
 "DOB": "1996.03.22",  
 "RandomSeed": "S4NJ3PHR",  
 "winningHash": "029abc9c8a9058b4ba500cff14f410a1741593e39faf83447d874c0b5dde0c9b",  
 "signedID": "RBCv/alHwTe23lwpP39OM1Q6zr5GOT3rsjpWPbxaviaUb38r8GVZzkFtC24iHPZviCJpDRwXR0bNNGJ6bOIf7YX65L7nAKMeL4kWqE42eukq6QFaQpA/7qUJSja/GneWQWhOlHTLr5CSL1xfW0wsPeG3J1gkBoA4JjNjfPB9ypU\u003d",  
 "signedWinningHash": "gJ3EXiSHCAoSscXnOFNSB3JF2HjPW2yfRysxexPsEP2X3hLANv6VGFY5Eb4PjwqCOAoIeutqi5BXwaEZOS0xTbnwAgmq1E9Mi4ptmz5EeM585uPT3Gpc4rQcM64KcnhXkaZ77BTRTWWJWmSs/gmCUeQCzjin6yVmP5vy4hVl9gk\u003d",  
 "Diag": "Shingles",  
 "Treat": "WaitForRelief",  
 "Rx": "Zovirax"  
},{  
 "BlockID": "a1f277de-8cb4-40f1-984f-2ddc613628a9",  
 "blockNum": 5,  
 "TimeStamp": " 2022-11-01.23:09:18.0",  
 "VerificationProcessID": "2",  
 "PreviousHash": "25ad9b5af477a3f8b2fe6afc665a95068e76398ded1ed2f6eca2aea7db42434b",  
 "fName": "Wayne",  
 "lName": "Blaine",  
 "SSNum": "123-45-6777",  
 "DOB": "1942.07.07",  
 "RandomSeed": "LCKD00X8",  
 "winningHash": "44ffe1ff1ac0def92edae8ff58fef5444be04c04fcd9c07deba40098b03ffae4",  
 "signedID": "Oz/rb7eszQY3QunJ8eD08AjNJV+YG5C0JuKnXSrlpU3g0g2HOwpoTS1Ng0MPmiHnSqlJdO/oCZ/jwECQYoiTITSElscVHQzU5LP5NK/YVTVZdo+i7rnOnzh+WawfjJKVjspqZNx94tTnnZurahshVKVggF7D+WCEaGhFKtZV7i8\u003d",  
 "signedWinningHash": "dLchedMFkd+CsfYeaMZ/itaWyHWAefWdcdvfO3VK4AMZiV09XrMdiRQoZeXoJlfHFWOJSpy2AOEdgshooUAGBRCkKT9wUJSiC9aoDsSQkV867SkKuwijLUPjisAmLfRQlV7Zdd+euqIhplOj7SuyAWDVFQHjOMcbzUMiHAJFoLE\u003d",  
 "Diag": "Measles",  
 "Treat": "WaitToGetBetter",  
 "Rx": "CodLiverOil"  
},{  
 "BlockID": "ab0318e7-cc05-4438-8a08-055aa5add737",  
 "blockNum": 4,  
 "TimeStamp": " 2022-11-01.23:09:16.0",  
 "VerificationProcessID": "0",  
 "PreviousHash": "06dc21b75322b96e0f066b70a9e93c70980996dee8d51e21b65f4a00464aa1d7",  
 "fName": "Joe",  
 "lName": "Blow",  
 "SSNum": "123-45-6888",  
 "DOB": "1996.03.07",  
 "RandomSeed": "5BED4FB2",  
 "winningHash": "25ad9b5af477a3f8b2fe6afc665a95068e76398ded1ed2f6eca2aea7db42434b",  
 "signedID": "fCPAvnadsHFNsSIIegpzI+xRF9q3UxvbU58tEmvIhq/iNUN2BA/lncM3cJcj5lHuH20JuByO/fJuPUWs1Ghbw5bmQbQdPkR5uiXq5F7igUJRlCM5VNGh9lp3fmKP1Hcn3Og9eC9KeQ6xm8gFGcCR1E7rzCKFUX72iIs+Eqiwt4s\u003d",  
 "signedWinningHash": "C9/TwzQRsVg/sQOOunhscU4dhzkpVE1XXanZ0DKAvfuIl0zd+lUcmAM4UAvIVwV6Av72QTPIFFUttdFaRjDXOALiDY1Dz8up10SAeWCstHQq5oE14L0kZrFmPe+YyFLRp/Qz5c8OncgYS99N6XSqAqHSGtXmIFXCNCcZAnUJtfs\u003d",  
 "Diag": "Smallpox",  
 "Treat": "BedRest",  
 "Rx": "Whiskey"  
},{  
 "BlockID": "7a1c762c-28c6-4329-b143-141034347afc",  
 "blockNum": 3,  
 "TimeStamp": " 2022-11-01.23:09:17.0",  
 "VerificationProcessID": "1",  
 "PreviousHash": "109ae0901d49f7058acef4834876ff0e2173f8e1ebfbe43ddaef9f1aeb644fcc",  
 "fName": "Julie",  
 "lName": "Wilson",  
 "SSNum": "123-45-6999",  
 "DOB": "1996.03.07",  
 "RandomSeed": "ATNRBQX7",  
 "winningHash": "06dc21b75322b96e0f066b70a9e93c70980996dee8d51e21b65f4a00464aa1d7",  
 "signedID": "aYVvxd92RKdQnC6XR0eIp2krSUuaVNhV9JVFQlQ+ZV2hDyVtE/veKBfZGGlU57pf3doC7uAQWxM7Zn2sYcIAZ8GxGP1vwdOSqMF/0fZKlZyNdEGUlj5qqvMEerfiAPoox3+fviMc2R54wT/A1KOo+wCUeOxmhiI+DPgsMxNJC9Y\u003d",  
 "signedWinningHash": "YtYMsZX6Q5VP4ZRxhj7fVP6zJpb/Ofbj6xE6azMuXuHWwh66RKFWX/dWT9Lg5aN2oLbEmM/jTIfv/fia54uWD8tdnCXr+a9WeKBt14fsbGoaQbUZJmD+wraCke5u3oY2pg5gNco/LY/0/RfDgZASXp2on4V0YkKnxR/sFpcFAMg\u003d",  
 "Diag": "Insomnia",  
 "Treat": "Exercise",  
 "Rx": "HotPeppers"  
},{  
 "BlockID": "ac01b37d-98a4-41de-9814-fb6f63297c65",  
 "blockNum": 2,  
 "TimeStamp": " 2022-11-01.23:09:17.1",  
 "VerificationProcessID": "2",  
 "PreviousHash": "0c7194144c3541cb957a0ee37b92743425ea0b8a76f08b27d021ae4b2b032414",  
 "fName": "Rita",  
 "lName": "Vita",  
 "SSNum": "999-456-789",  
 "DOB": "1992.01.31",  
 "RandomSeed": "KQIP02OZ",  
 "winningHash": "109ae0901d49f7058acef4834876ff0e2173f8e1ebfbe43ddaef9f1aeb644fcc",  
 "signedID": "VHXT6+x6TlRrxEgnBtauBsfnb1anmJRdiE3l3q2RXQh1TAQd43J2McDn9MG/rvGLVKtuLNVzYVB1lOzrtc2cf8VmXprku7GrhgHQVA3ybBiW/NV/MBHUhFH6FxovreBqcZ86P2etB38i3MfI6zXkALOMhQHltTTQupvVxxrGNrI\u003d",  
 "signedWinningHash": "VUjre7adE7Z6tJXmxtI3DWOw3ews851+R6SrH8ORqdaRgKmuq/FWiTCT7/r1k+qw6TycFbonxkyxqsgnmQZuw21lVwExqLuYEPhjEg/JpWcP5WbVpYWdFrvTxg7HTK4qD2jQOIEdT3IEVtxnlsd73lXqJAmgX07Azox98aXueW4\u003d",  
 "Diag": "ObessivePersonality",  
 "Treat": "TryToRelax",  
 "Rx": "Ibuprofen"  
},{  
 "BlockID": "d76cb33c-93f6-457d-a152-3dac33d2f198",  
 "blockNum": 1,  
 "TimeStamp": " 2022-11-01.23:09:15.0",  
 "VerificationProcessID": "0",  
 "PreviousHash": "9EF956ACD8CC3D48B6058E34773FC1FC0CE7E234F782CDEE5C98D4D6C50093E5",  
 "fName": "John",  
 "lName": "Smith",  
 "SSNum": "123-45-6789",  
 "DOB": "1996.03.07",  
 "RandomSeed": "DPQFLPDN",  
 "winningHash": "0c7194144c3541cb957a0ee37b92743425ea0b8a76f08b27d021ae4b2b032414",  
 "signedID": "ArIhMAowAsafQbzYu423ZEokeRnSUdx4+mMrCRNoiZVazTCrdHhBJ3EWJZkA7hBsnpyL2+CBwplkbEirsLO1y6VVSXS3Tqm7JXhkoHivp7zQUKwFZoKDnh5MHTVUS5mmrMbJLKTsA4vaNgaAllMYNB1aJjWW8m8771J5ALFXQ/A\u003d",  
 "signedWinningHash": "a73k4ZXFxloGh6BULil+tfsgviDO1ou4605IT5WIVO/hs42jxjRAWlQXFMBHDKSWTS9AAXzVGU/x1/Z5Cd2zr86MuFH8hegMrzwQSNFyE0hBNv2Ky9eIcLm7qNtVp1wB+b1nI7wvKlo6idphe57vovS+iYzWNqB5k+CsenIWP6U\u003d",  
 "Diag": "Chickenpox",  
 "Treat": "BedRest",  
 "Rx": "aspirin"  
},{  
 "BlockID": "e1ed2a77-0a43-49ff-9be3-1a68defe8f25",  
 "blockNum": 0,  
 "TimeStamp": " 2022-11-01.23:09:14.0",  
 "VerificationProcessID": "0",  
 "PreviousHash": "0000",  
 "fName": "Satya Yoganand",  
 "lName": "Addala",  
 "SSNum": "123-45-6789",  
 "DOB": "05-22-1998",  
 "RandomSeed": "03L810QS",  
 "winningHash": "9EF956ACD8CC3D48B6058E34773FC1FC0CE7E234F782CDEE5C98D4D6C50093E5",  
 "Diag": "Blockchain ",  
 "Treat": "BlockChain",  
 "Rx": "Blockchain"  
}]

--- > Output for Process 0  
  
(base) satyayoganandaddala@Satyas-MacBook-Air BlockChain\_Assignment % java -cp ".:gson-2.8.2.jar" Blockchain.java 0  
Satya Yoganand's Blockchain program. Ctl-c to quit  
  
Using processID 0  
  
Key Pair : java.security.KeyPair@150ab4ed  
Printing Public Key : PublicKeyObject@3a62c01e  
PublicKey for processID : 0 ----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB  
All the Servers have been set, waiting to receive start message  
Before Start Status : wait  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB","processID":0}  
PublicKeyArray : [PublicKeyObject@72e011bc]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB","processID":1}  
PublicKeyArray : [PublicKeyObject@72e011bc, PublicKeyObject@1d57c354]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB","processID":2}  
PublicKeyArray : [PublicKeyObject@72e011bc, PublicKeyObject@1d57c354, PublicKeyObject@29a62b50]  
ProcessID 0 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB  
ProcessID 1 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB  
ProcessID 2 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB  
----------------------------------------------  
Collected Data from Initial Block ........  
Processing BlockInput0.txt File...  
Unverified Blocks sent to Priority Queue...  
11 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: fa717e663de841a5f7d8b59117534f3ae88f2871a8c3284b50ec075b6343aacb  
Printing the first four digits from hash in decimal along with the hex value : fa71 and 64113  
64113 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 4be9660d0c63849b44b8aa91834e0b448518548937147ef6170b5f18cb5402f4  
Printing the first four digits from hash in decimal along with the hex value : 4be9 and 19433  
block verified! with the value : 19433  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
10 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 62d9e94d683d5fbe1e977331e358a4e00d679662ccf7ac48f06845a8578580df  
Printing the first four digits from hash in decimal along with the hex value : 62d9 and 25305  
25305 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 02cbd9d20a4ec8a930e940cd82ded23bc5d1747c40cb114113b75f3ab4026903  
Printing the first four digits from hash in decimal along with the hex value : 02cb and 715  
block verified! with the value : 715  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
9 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 8eaf05fde3f88b1d5b5a580dfbc06a1a557bba88e7a153f0a932d9115225b765  
Printing the first four digits from hash in decimal along with the hex value : 8eaf and 36527  
36527 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 47d6d210be9d1c64edef4abc000ea0529e9283887521ce18f92c9aea6b8b9fe0  
Printing the first four digits from hash in decimal along with the hex value : 47d6 and 18390  
block verified! with the value : 18390  
processing to rework on block again...  
8 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: e83b6cb6a889b7713cdca5381c7b8c9e9469e0f9aa0cbf6ec35e2e18af8112e3  
Printing the first four digits from hash in decimal along with the hex value : e83b and 59451  
59451 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: a472fb71a33d3ce540a8dbb5ba04ca90da0cc1df13a11441f49bde017a908544  
Printing the first four digits from hash in decimal along with the hex value : a472 and 42098  
42098 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 94debe272ac656989f5a00116cd37e9332053b9a14f7ebf9cf911bed12600e56  
Printing the first four digits from hash in decimal along with the hex value : 94de and 38110  
38110 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
7 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 212c028bf703a1d5c50adfa96f36275292ed719a5e0643e390797442f62e2dc7  
Printing the first four digits from hash in decimal along with the hex value : 212c and 8492  
block verified! with the value : 8492  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
6 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: 24afb88e1c045ccae0c436d9243a04a0527b55c693f63f467ae5dadfd364680b  
Printing the first four digits from hash in decimal along with the hex value : 24af and 9391  
block verified! with the value : 9391  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
5 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: 33b8788fc61191bbfb923724e2eb3245286e807dba7578b6602aafa3ba96e0f1  
Printing the first four digits from hash in decimal along with the hex value : 33b8 and 13240  
block verified! with the value : 13240  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
4 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: 1f7dd7efb143c9a4e89d36a58b595cc7e88a3b9c6df25e8fabdec9a028528844  
Printing the first four digits from hash in decimal along with the hex value : 1f7d and 8061  
block verified! with the value : 8061  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
3 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: 8d1fc26b7f3714556877224133375c89ebfc2eddb5515c6900b5b2d1ae886168  
Printing the first four digits from hash in decimal along with the hex value : 8d1f and 36127  
36127 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 41360c65f4f86aa9ba89fc696fdad006b789009e2c132e5af16ec829cfb67206  
Printing the first four digits from hash in decimal along with the hex value : 4136 and 16694  
block verified! with the value : 16694  
processing to rework on block again...  
2 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: 9a34f040a6903a77bc8683a8c98bf3ffcb48472c4788e7f49e07466bb27ba80a  
Printing the first four digits from hash in decimal along with the hex value : 9a34 and 39476  
39476 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: e8c438bc7555755904697dd61ca933f713c0ccf975dc94c86b410f53f3dcff68  
Printing the first four digits from hash in decimal along with the hex value : e8c4 and 59588  
59588 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
1 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: d3880b1b25598a9b2c05fd9bb84edbb2f603ac8f2886d623407d923cf51aeccb  
Printing the first four digits from hash in decimal along with the hex value : d388 and 54152  
54152 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 051843812f3805edb7c96251328ac14a6957808537ddfdbc65a1b939e63847b3  
Printing the first four digits from hash in decimal along with the hex value : 0518 and 1304  
block verified! with the value : 1304  
processing to rework on block again...  
0 unverified blocks remaining  
BLOCKCHAIN COMPLETE  
  
----> Output for process 1  
  
(base) satyayoganandaddala@Satyas-MacBook-Air BlockChain\_Assignment % java -cp ".:gson-2.8.2.jar" Blockchain.java 1  
Satya Yoganand's Blockchain program. Ctl-c to quit  
  
Using processID 1  
  
Key Pair : java.security.KeyPair@150ab4ed  
Printing Public Key : PublicKeyObject@3a62c01e  
PublicKey for processID : 1 ----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB  
All the Servers have been set, waiting to receive start message  
Before Start Status : wait  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB","processID":0}  
PublicKeyArray : [PublicKeyObject@5484353e]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB","processID":1}  
PublicKeyArray : [PublicKeyObject@5484353e, PublicKeyObject@1d57c354]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB","processID":2}  
PublicKeyArray : [PublicKeyObject@5484353e, PublicKeyObject@1d57c354, PublicKeyObject@29a62b50]  
ProcessID 0 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB  
ProcessID 1 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB  
ProcessID 2 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB  
----------------------------------------------  
Collected Data from Initial Block ........  
Processing BlockInput1.txt File...  
Unverified Blocks sent to Priority Queue...  
12 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: edfe693c99d4ba911edcbfa1f3d29f2c56b0ca3a8d0f29141bb8d1c6b94b0990  
Printing the first four digits from hash in decimal along with the hex value : edfe and 60926  
60926 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: f30b90e10ab92eb6fa3d48cd7cf7a38aad6c37cb58a1a2c72878cb4864cefd6b  
Printing the first four digits from hash in decimal along with the hex value : f30b and 62219  
62219 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
11 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: b29f288f7881494f990c886714afd6cdd0af11661805b1132a131c320e6d7338  
Printing the first four digits from hash in decimal along with the hex value : b29f and 45727  
45727 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 44439a375fcf294cf2b84ce030e0a450c43c0dffa6c5a8a6ac6eacc6fa5b22f8  
Printing the first four digits from hash in decimal along with the hex value : 4443 and 17475  
block verified! with the value : 17475  
processing to rework on block again...  
10 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 78f654dd4a23452fcabcf10ce79093920e7fb47cfa2e9a0439c6e0a1ce7b576d  
Printing the first four digits from hash in decimal along with the hex value : 78f6 and 30966  
30966 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 857d65b2644b2edde202939108c664e08417bc7d0d8faba693ec0364267962fa  
Printing the first four digits from hash in decimal along with the hex value : 857d and 34173  
34173 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
9 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: d9a7978dbc48d1d7e54aa88932e44327478738aec11f7e17f53598513cbd07dc  
Printing the first four digits from hash in decimal along with the hex value : d9a7 and 55719  
55719 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 4b9000a4a34ef4e6eaf5970a13a95e34e61cb97c43deb9a9677c72d504984a2a  
Printing the first four digits from hash in decimal along with the hex value : 4b90 and 19344  
block verified! with the value : 19344  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
8 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 52d3c39fc01c620c98d581def70872c844552071b76c68c005f874e0e9b7a937  
Printing the first four digits from hash in decimal along with the hex value : 52d3 and 21203  
21203 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 9b546a6c7a0a9e7960e551d085864b5cdfecd9165d61f55b29f04a0b3953e602  
Printing the first four digits from hash in decimal along with the hex value : 9b54 and 39764  
39764 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
7 unverified blocks remaining  
Using the public key from process: 1  
Block already in blockchain  
6 unverified blocks remaining  
Using the public key from process: 1  
Block already in blockchain  
5 unverified blocks remaining  
Using the public key from process: 2  
Block already in blockchain  
4 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: 44e8d10fc24acb218f13b4b4768535f2cb13258dc241a4da438f86d4f986f83b  
Printing the first four digits from hash in decimal along with the hex value : 44e8 and 17640  
block verified! with the value : 17640  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
3 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: dc4906c39ff4f8d622c359b27b616788290f4e30f73a84a73c816871788115b9  
Printing the first four digits from hash in decimal along with the hex value : dc49 and 56393  
56393 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 82caa71b1eaba746c239961e1530cc773a071a77017ad23f815423a7604b5890  
Printing the first four digits from hash in decimal along with the hex value : 82ca and 33482  
33482 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
2 unverified blocks remaining  
Using the public key from process: 2  
Block already in blockchain  
1 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: ea4888b9c68a0306787c3db15d321af141a0873d33af9b1eab38302d2fafb0c3  
Printing the first four digits from hash in decimal along with the hex value : ea48 and 59976  
59976 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 193806b9a7cf23a0eebeea19ea600a10bec8b1dd7f303ccaf4e6536bf20e6abc  
Printing the first four digits from hash in decimal along with the hex value : 1938 and 6456  
block verified! with the value : 6456  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
0 unverified blocks remaining  
BLOCKCHAIN COMPLETE  
  
----> Output for Process 3  
  
(base) satyayoganandaddala@Satyas-MacBook-Air BlockChain\_Assignment % java -cp ".:gson-2.8.2.jar" Blockchain.java 2  
Satya Yoganand's Blockchain program. Ctl-c to quit  
  
Using processID 2  
  
Key Pair : java.security.KeyPair@150ab4ed  
Printing Public Key : PublicKeyObject@3a62c01e  
PublicKey for processID : 2 ----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB  
All the Servers have been set, waiting to receive start message  
Before Start Status : wait  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB","processID":0}  
PublicKeyArray : [PublicKeyObject@2ea3e2a]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB","processID":1}  
PublicKeyArray : [PublicKeyObject@2ea3e2a, PublicKeyObject@77642f4]  
Public Key Received: {"publicKey":"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB","processID":2}  
PublicKeyArray : [PublicKeyObject@2ea3e2a, PublicKeyObject@77642f4, PublicKeyObject@1d57c354]  
ProcessID 0 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0I3YsBJIHSNSVQxEgkAPEFsiIgCuiY2tzPZtHYhHIdK1X2Uea7hUURuunK972SQJn/drc4oG6TSGXroMNRLz/RA5XKf2ELoPcYOWynldkib49fU88TFfNGXaXX875rgtWE4jtHK7JlKAFzRjHni+UdYd5h0C06f6rYZ22GS2WnQIDAQAB  
ProcessID 1 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC3y8VY7mIIowaNlSey0hTvgu0MGJ/YAS+GievLcIlJTcFgCpmHCFirdYU8E+XFNy3pVnisSZRd/ZRonnOeX3KeAjtj59YWKE8Mu0LZPc1LfI1bi35KTPF/Z+jdpws39SImuDSV+nE8PHtCfZ9kqJBmjQcWfhV8vvagMCkQemmPvQIDAQAB  
ProcessID 2 Using : MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNQhnWtEpWS9tRrzHa0IyBQsQcB7xzX0kZgM++7lzAvL0xcQa2c/rU1MJUcxZMtlNcyQBNOxw2CLIj8T74KT/bRSfFfamIKNwbdvcKVRsHspHEV+8+cq8cf9wT8PUOEVftqGICFbCVOx825Nnrrm8Vtur6qqRkgSGLrhxqReLZ9wIDAQAB  
----------------------------------------------  
Collected Data from Initial Block ........  
Processing BlockInput2.txt File...  
Unverified Blocks sent to Priority Queue...  
11 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: f1784348e5e177cb6256b3632080bd962350a3368eb0b8f0ff4b126aa07c9897  
Printing the first four digits from hash in decimal along with the hex value : f178 and 61816  
61816 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: d936ff819c8366327a1e412197ee7cb57ef8568b37e71cbc0e9d1f0fa303b372  
Printing the first four digits from hash in decimal along with the hex value : d936 and 55606  
55606 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
10 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 7dc8d94d870ccc2214f47f6cd47eeab37f638fe5b22132cbc63bee362e6fdda9  
Printing the first four digits from hash in decimal along with the hex value : 7dc8 and 32200  
32200 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: df2ac7c0288f2cebf8b9ec05248811204397a9c0f178a3a5a521bcf1b3ec7dea  
Printing the first four digits from hash in decimal along with the hex value : df2a and 57130  
57130 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
9 unverified blocks remaining  
Using the public key from process: 0  
Attempting to verify block  
Hash is: 486058b3a172f7dbc5a322627c93ecfa8b177a682d9993a8b7170f3a8adbcf00  
Printing the first four digits from hash in decimal along with the hex value : 4860 and 18528  
block verified! with the value : 18528  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
8 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: bbbbc9281a960fa1f6c08e776eded385c724d54397eade765b83715099eb78c0  
Printing the first four digits from hash in decimal along with the hex value : bbbb and 48059  
48059 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 82277606d08da5baf1ee2b81d2245a1cf2fbeaf02b6f0a029632657193b6d079  
Printing the first four digits from hash in decimal along with the hex value : 8227 and 33319  
33319 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 3d22db10f306dcc4fc3e602d42585e594060133c607e8c3f90d519f081e78449  
Printing the first four digits from hash in decimal along with the hex value : 3d22 and 15650  
block verified! with the value : 15650  
processing to rework on block again...  
7 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: cf0cef95127c96b65621ef2f94e56a5298a224d69543c9a4fba3cdb60a2badbd  
Printing the first four digits from hash in decimal along with the hex value : cf0c and 53004  
53004 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 277168c07d625838a39d9595571ef3df205d59aa0611ea01a3a9e23670121d0f  
Printing the first four digits from hash in decimal along with the hex value : 2771 and 10097  
block verified! with the value : 10097  
processing to rework on block again...  
6 unverified blocks remaining  
Using the public key from process: 1  
Block already in blockchain  
5 unverified blocks remaining  
Using the public key from process: 2  
Block already in blockchain  
4 unverified blocks remaining  
Using the public key from process: 1  
Attempting to verify block  
Hash is: 50cd566cf799ba9e74ec5bbd25f8b6bf8064d85ea3b4c217efed5a75dc9685fc  
Printing the first four digits from hash in decimal along with the hex value : 50cd and 20685  
20685 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: 7293c305b0c65a6e3c5834497f92efb09b5b2f03507bddc31fbd4561376b1692  
Printing the first four digits from hash in decimal along with the hex value : 7293 and 29331  
29331 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
3 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: 450e555cc17432f974b1d4e67e97853a2c447c9201d9b0650772af53a8e26be3  
Printing the first four digits from hash in decimal along with the hex value : 450e and 17678  
block verified! with the value : 17678  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
2 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: 29d2fc6c7eb9d982837cb20f57ce127f95bbb1276b8b245a23612197f8bbc46b  
Printing the first four digits from hash in decimal along with the hex value : 29d2 and 10706  
block verified! with the value : 10706  
Block has been verified and added to the blockchain and started multicasting to all the other running processes  
1 unverified blocks remaining  
Using the public key from process: 2  
Attempting to verify block  
Hash is: 890bf497ee0be9845d6675bcca3fad8071a1bd63989dc6e38ae980638d979a05  
Printing the first four digits from hash in decimal along with the hex value : 890b and 35083  
35083 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: f510aa5660761740344e3d53d42a62d6e7f9ce43931afca52b43c9ee977ce2ae  
Printing the first four digits from hash in decimal along with the hex value : f510 and 62736  
62736 is not less than 20,000 so we did not solve the puzzle  
  
Hash is: f484caad143a1ff2b636af46839424c8a59216b3433d7c7464a0b7261f7521ea  
Printing the first four digits from hash in decimal along with the hex value : f484 and 62596  
62596 is not less than 20,000 so we did not solve the puzzle  
  
Abandoning block...  
0 unverified blocks remaining  
BLOCKCHAIN COMPLETE