BLOCKCHAIN CRYPTOCURRENCY PROJECT

MEMBERS:

NIKHIL KRISHNA 2018A7PS0496H

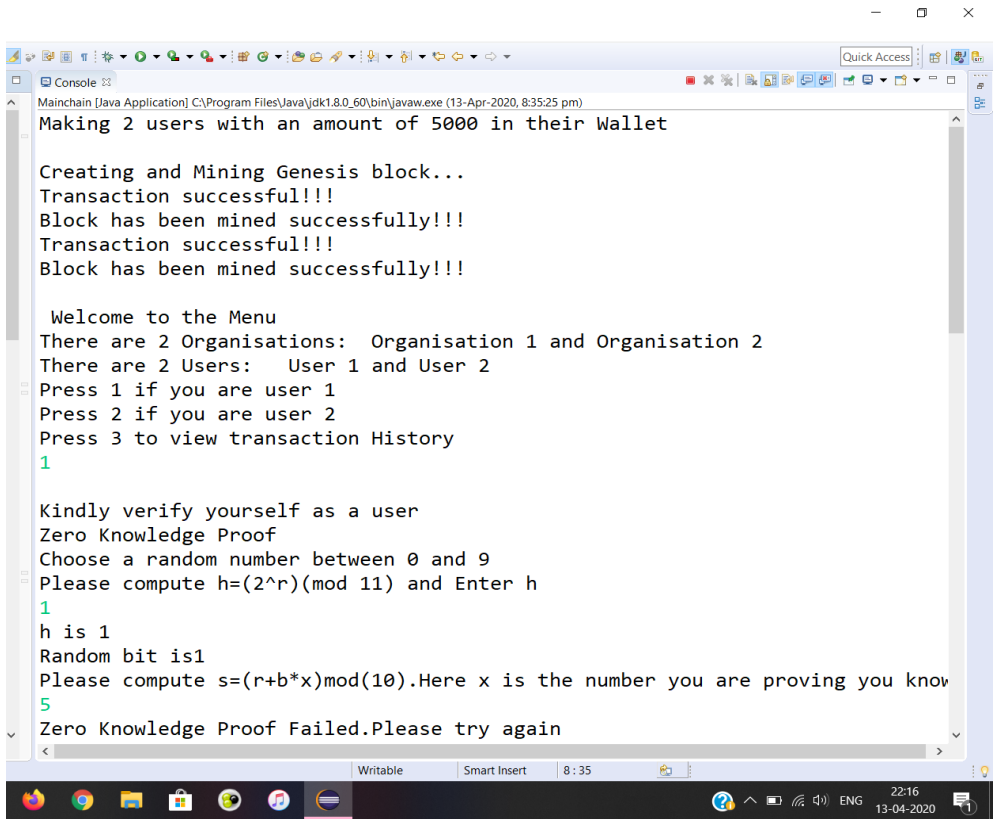
MIHIR BANSAL 2018A7PS0215H

BHAVYESH DESAI 2018A7PS0164H

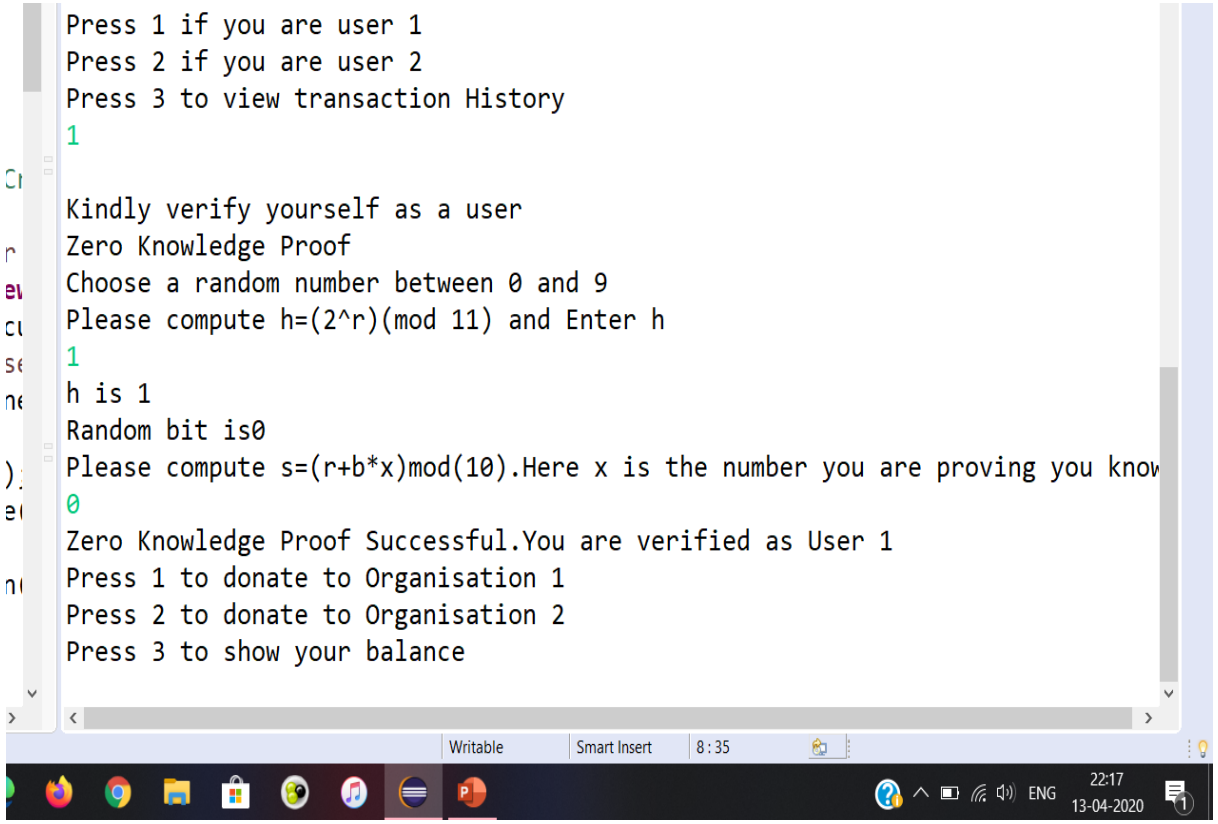
KUSHAGRA LAVANIA 2018A7PS0216H

ADITYA JHAVERI 2018A7PS0209H

STEP 1 : USER AUTHENTICATION



FAILED AUTHENTICATION



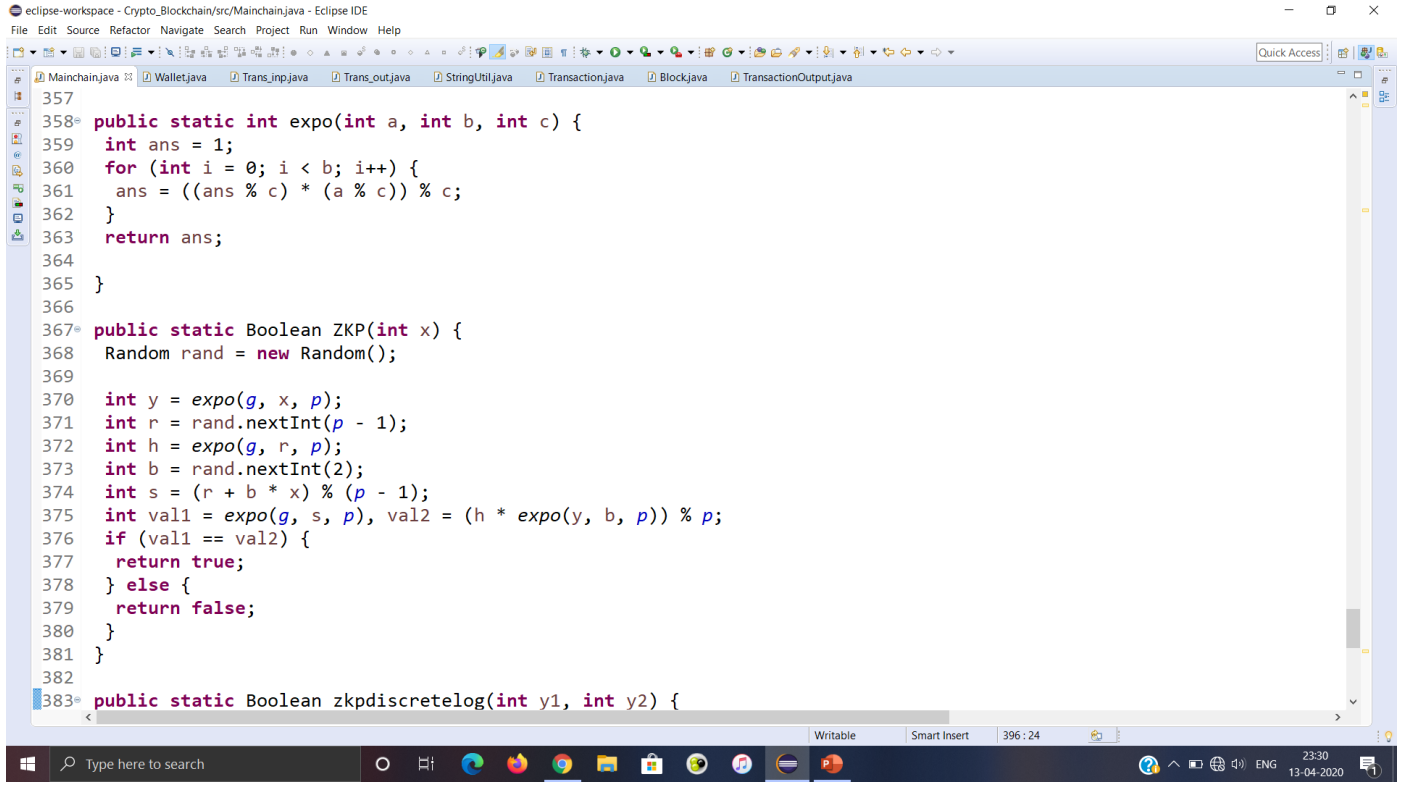
SUCCESSFUL AUTHENTICATION

PROCESS OF USER VERIFICATION

```java
public static Boolean zkpdiscretelog(int y1, int y2) {

 Random rand = new Random();
 Scanner sc = new Scanner(System.in);
 System.out.println("\nKindly verify yourself as a user");
 System.out.println("Zero Knowledge Proof");
 System.out.println("Choose a random number between 0 and 9");
 System.out.println("Please compute h=(2^r)(mod 11) and Enter h");
 int h = sc.nextInt();
 System.out.println("h is " + h);
 int b = rand.nextInt(2);
 System.out.println("Random bit is" + b);
 System.out.println("Please compute s=(r+b*x)mod(10).Here x is the number you are proving you know");
 int s = sc.nextInt();

 int val1 = expo(2, s, 11) % 11;
 int val2 = (h * expo(y1, b, 11)) % 11;
 int val3 = (h * expo(y2, b, 11)) % 11;

 if (val1 == val2) {
  System.out.println("Zero Knowledge Proof Successful.You are verified as User 1");
  return true;
 } else if (val1 == val3) {
  System.out.println("Zero Knowledge Proof Successful.You are verified as User 2");
  return true;
 } else {
  System.out.println("Zero Knowledge Proof Failed.Please try again");
```

This involves solving the Discrete logarithm problem which is the toughest to compute by an attacker. Hence, it leads to a secure blockchain network.

THE DISCRETE LOGARITHM PROBLEM



In the mathematics of the real numbers, the logarithm $\log_b a$ is a number x such that $b^x = a$, for given numbers a and b. Analogously, in any group G, powers b^k can be defined for all integers k, and the discrete logarithm $\log_b a$ is an integer k such that $b^k = a$. In number theory, the more commonly used term is index: we can write $x = \mathrm{ind}_r\, a \pmod{m}$ (read the index of a to the base r modulo m) for $r^x \equiv a \pmod{m}$ if r is a primitive root of m and $\gcd(a,m) = 1$

BLOCK CREATION

- The user’s data such as Aadhar Number and PAN Number are stored in a block in the form of a String.
- The variable ‘epochTime’ stores time elapsed since 1/1/1970.
- Nonce is a arbitrary number that is used once.
- Genesis block is the first block of the Blockchain. It does not have a previous block so we set "0" as the previous hash.
- A block is created on calling the constructor of Block.java class which takes previous hash value and user’s userid as arguments.

MINING A BLOCK

- Bitcoin mining is the process of adding transaction records to Bitcoin's public ledger of past transactions or blockchain.
- It is done using function blockMine() inside Block.java class by using a merkle tree structure concept.
- Any node in the merkle tree is the hash of all it's children
- Hence the merkle root is the hash of the hashes of all transactions in the Block
- Adding or removing even one transaction changes the value of all parent's of the transaction
- Hence addding or removing a transaction also changes the merkle root hash
- So we can use the merkle root as a value to verify the integrity of all transactions under it
- In doing so, we do not need the body of the transactions

GENERATING KEY PAIRS

- Elliptic curve Cryptography is used to generate each user’s private and public keys in a KeyPair.
- The Elliptic Curve Digital Signature Algorithm is implemented in the generateKeyPair() method inside Wallet.java
- The private key of each user is kept secret and is used for generating the digital signature of every user after each transaction.
- To view a transaction, the public key of the recipient is used to decrypt the hashed transaction block inside the merkle tree of every user.
- The public key of every user is made public to all whereas the private key is kept secret with himself.

TRANSACTION PROCESSING

- The user is asked if he wants to make a donation to Organization 1 or Organization 2.

```
Press 1 to donate to Organisation 1
Press 2 to donate to Organisation 2
Press 3 to show your balance
1
Enter amount to donate
450
Transaction successful!!!
Block has been mined successfully!!!
Now Your balance is 3750.0
Press 1 if you are user 1
Press 2 if you are user 2
Press 3 to view transaction History
3

User1's  balance is 3750.0
User2's  balance is 5000.0
Organization 1's  balance is 1250.0
Organization 2's  balance is 0.0
Press 1 to view user 1 Transaction History
Press 2 to view user 2 Transaction History
Press 3 to exit
```

TRANSACTION VERIFICATION

- The user is asked if he wants to make a donation to Organization 1 or Organization 2.
- On selecting the appropriate option, the transaction is hashed with the public key of the organization and the transaction block is added to the history of transactions under the user authenticated.
- This is done using sendfunds() function inside the Wallet.java class.
- The transaction is signed by the sender’s digital signature which is hashed using private key of the sender.
- Inside StringUtil.java, toApplyECDSASig(PrivateKey privateKey,String input) applies ECDSA Signature to the input and the sender's private key and returns the result as bytes.
- In StringUtil.java, toVerifyECDSASig(PublicKey publicKey, String data, **byte[] signature)** will verify the signature using the public key and the string data. It returns True or False if the signature is valid
- Only the public key of the recipient will be able to successfully verify the transaction.

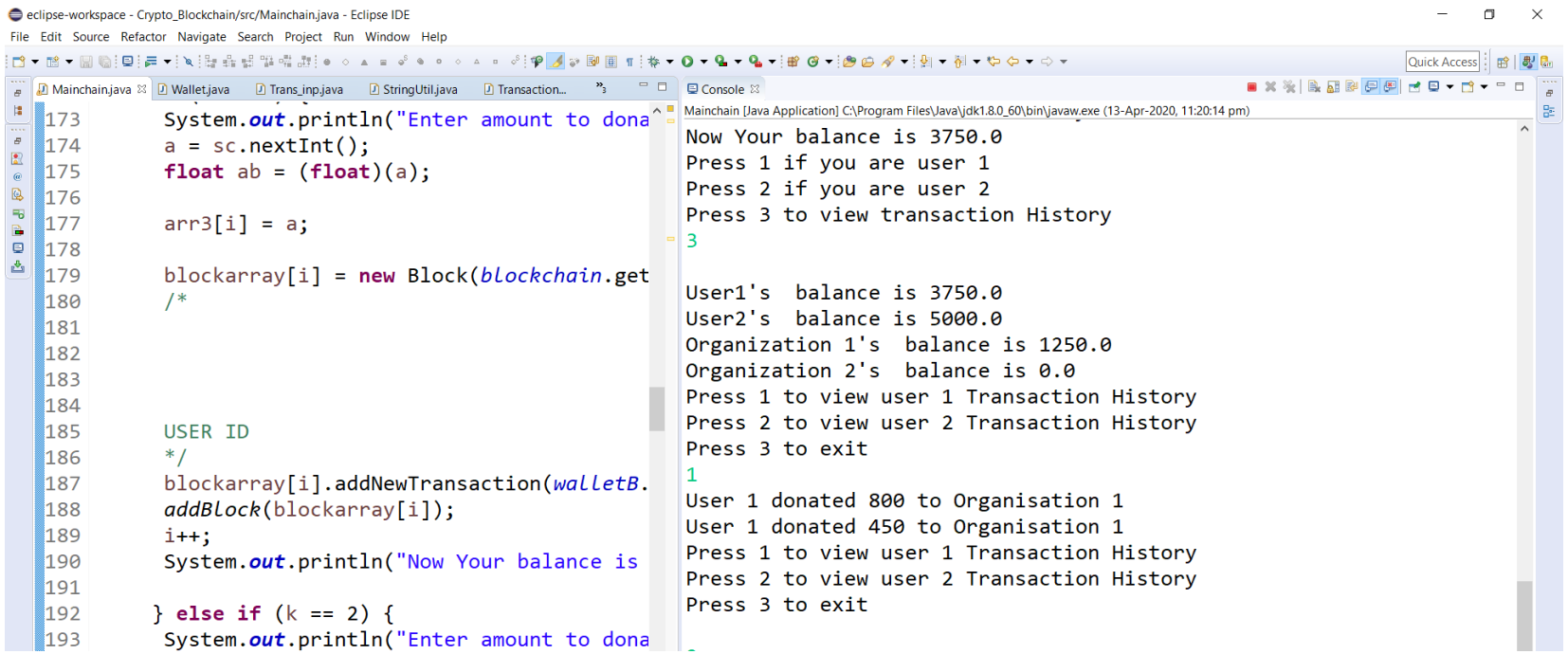
SHA256 ALGORITHM

- Inside StringUtil.java, generate256Sign(String input) applies the SHA256 algorithm to a string and returns the result.
- It generates a 256 bit signature for a text.
- A hash cannot be decrypted back to the original text.
- **SHA-256** is one of the successor hash functions to SHA-1 (collectively referred to as SHA-2), and is one of the strongest hash functions available. SHA-256 is not much more complex to code than SHA-1, and has not yet been compromised in any way.
- The 256-bit key makes it a good partner-function for AES.

VIEW USER’S TRANSACTION HISTORY

- The details of the user can be received by opting for menu no. 3, which displays all the user’s transaction details.
- The user’s wallet balance can be checked by decrypting the block which has the transaction details in the merkle tree after a transaction is made successfully.
- The sum of all the unspent transaction outputs addressed to the user is the wallet balance
- This is done using getBalance() function inside the Wallet.java class which decrypts the hashed block of the specified receiver of transaction using its public key using isMine() function.

VIEWING USER'S TRANSACTION HISTORY



USER 1'S TRANSACTION HISTORY