PROGRAMMING 2
PROJECT-BITCOIN SYSTEMS

June-2023

Ayca Su Alparslan
Betul Gul
Malak Matar
Barbara Pianigiani

Sapeienza University-
ACSAI

PROJECT GOALS

We aimed to develop a simple Simulator of Bitcoin and Blockchain systems in Java, in a case of 4 users and one blockchain making an output of the blockchain's hashmap and results of the transactions.

The key of the project is to study of the blockchain system with the help of the SHA-256 algorithm. Including 6 of the required classes for the simulation.

BREIF RECAP OF THE CLASSES

1. The code defines a Transaction class representing a transaction between two agents in a blockchain. It contains the sender's name, recipient's name, the amount sent, and a verification status.

2. The Block class represents a block in the blockchain. It contains a list of transactions, the hash value of the previous block, and a nonce value used in mining.

3. The Blockchain class manages the blockchain and provides methods for adding blocks and processing pending transactions. It also keeps track of pending transactions and maintains a list of blocks.

4. The Agent class represents an agent participating in the blockchain. It has a name and a balance. It can send coins to other agents by creating a transaction and adding it to the pending transactions in the blockchain.

5. The Miner class represents a miner who mines new blocks in the blockchain. It has a name and a reward. The miner can mine a new block by collecting pending transactions, finding a valid nonce, and adding the block to the blockchain.

6. The BitcoinSimulation class is the main class that demonstrates the usage of the blockchain, agents, and miner. It creates a few agents, performs transactions between them, processes pending transactions, and allows the miner to mine a new block. Finally, it prints the hash codes of all blocks in the blockchain.

INPUTS

1. Initialization of Agents:
   - The code initializes three Agent objects: Su, Barbara, and betul.
   - Each agent is initialized with a name (a string) and a balance (a double).
   - For example:

```
public static void main(String[] args) {
    Agent Su = new Agent( name: "Su",  balance: 10.0);
```

2. Initialization of Miner:
   - The code initializes a Miner object: miner.
   - The miner is initialized with a name (a string) and a reward (a double).
   - For example:

```
Miner malak = new Miner( name: "Malak",  reward: 2.0);
```

3. Initialization of Blockchain:
   - The code initializes a Blockchain object: blockchain.
   - The blockchain is initialized with an empty list of blocks and an empty map of pending transactions.
   - For example:

```
Blockchain blockchain = new Blockchain();
```

4. Transaction Initialization and Sending Coins:
   - The code initiates transactions between agents by calling the sendCoins() method on agent objects.
   - The sendCoins() method takes the recipient agent, the amount of coins, and the blockchain as inputs.
   - For example:

```
Su.sendCoins(Barbara,  amount: 3.0, blockchain);
```

5. Processing Pending Transactions:
   - The code calls the processPendingTransactions() method on the blockchain object.
   - This method verifies all pending transactions, adds them to a new block, and adds the block to the blockchain.
   - It returns a boolean indicating whether any verified transactions were processed.
   - [illegible]

```
if (blockchain.processPendingTransactions()) {
    malak.mine(blockchain);
```

6. Mining:
   - If there are verified transactions to be processed (returned true from processPendingTransactions()), the code proceeds with mining a new block.
   - The mine() method is called on the malak object, passing the blockchain as an input.
   - The mine() method adds a new block to the blockchain, including the verified transactions and a reward transaction for the malak.

These are the main inputs in the code that determine the behavior of the blockchain simulation. By modifying the initialization values and transaction amounts, we can observe different scenarios and outcomes in the simulation.

TRANSACTIONS

This class represents a transaction between two agents in the blockchain system

Attributes:
sender: A String variable that stores the name of the sender initiating the transaction.
recipient: A String variable that stores the name of the recipient receiving the transaction.

Constructor:
The constructor takes three parameters: sender, recipient, and amount.
It initializes the sender, recipient, and amount attributes based on the provided values.
The verified attribute is set to false by default, indicating that the transaction is initially unverified.
Getter methods: getSender: Returns the name of the sender involved in the transaction.
getRecipient: Returns the name of the recipient involved in the transaction.
getAmount: Returns the amount of coins being transferred in the transaction.
isVerified: Returns the verification status of the transaction.
Setter method: setVerified: Sets the verification status of the transaction.

BLOCK CLASS

Attributes transactions A List of Transaction objects representing the transactions included in the block.

previousHash: A String representing the hash value of the previous block in the blockchain.

Constructor:
This takes three parameters: transactions, previousHash, and nonce.
This method calculates the hash value of the block using the SHA-256 hashing algorithm.
It concatenates the relevant data of all the transactions in the block, along with the previousHash and nonce values.
It returns the calculated hash value as a String.

Getter methods: getTransactions: Returns the list of transactions included in the block.
The Block class encapsulates a set of transactions and provides the functionality to calculate the hash value of the block.

BLOCKCHAIN CLASS

The `Blockchain` class represents a blockchain, which is a data structure that contains a sequence of blocks linked together.
Attributes

- `blocks`: A `List` of `Block` objects representing the blocks in the blockchain.
- `pendingTransactions`: A `Map` that stores pending transactions waiting for verification.

Constructor:

- The constructor initializes the `blocks` attribute by creating an empty list.
- This method adds a transaction to the `pendingTransactions` map.
- This method processes the pending transactions and adds them to a new block if they are verified.
- It iterates over the pending transactions stored in the`pendingTransactions` map.
- For each transaction, it calls the `verifyTransaction` method to check if the transaction is valid.
- If a transaction is verified, it adds the transaction to a list of verified transactions.
- This method verifies a transaction by checking if the sender has sufficient balance to perform the transaction.
- It retrieves the sender's balance by calling the `getBalance` method.
- This method calculates the balance of an agent by iterating over all blocks in the blockchain.
- It checks each transaction in each block and adjusts the balance based on the sender and recipient of the transaction.
- This method returns the `blocks` list, providing access to all the blocks in the blockchain.

AGENTS

The `Agent` class represents an agent participating in the blockchain network.

Attributes
- `name`: A `String` representing the name of the agent.

Constructor
- The constructor initializes the `name` and `balance` attributes based on the provided arguments.
- This method returns the name of the agent.
- This method returns the balance of the agent.
- This method allows the agent to send coins to another agent.
- It takes three arguments: the `recipient` agent, the `amount` of coins to send, and the `blockchain` in which the transaction will be recorded.
- It first checks if the agent's balance is sufficient to perform the transaction.
- If the balance is sufficient, it deducts the `amount` from the agent's balance and adds it to the recipient's balance.
- It creates a new `Transaction` object with the agent as the sender, the recipient as the recipient, and the specified amount.

MINER CLASS

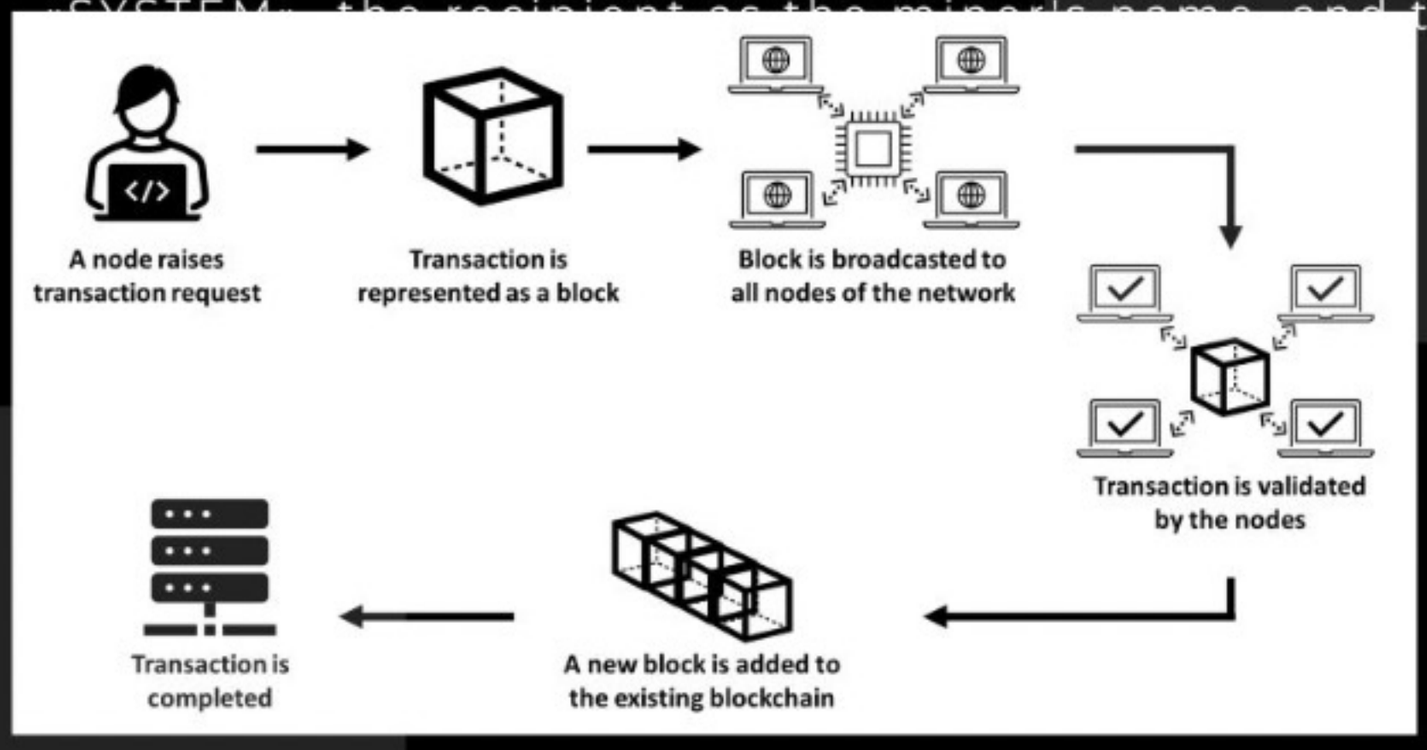
The `Miner` class represents a miner in the blockchain network. Miners are responsible for validating and adding new blocks to the blockchain through a process called mining.

Attributes

- `name`: A `String` representing the name of the miner.

Constructor

- The constructor initializes the `name` and `reward` attributes based on the provided arguments.
- This method returns the name of the miner.
- This method returns the reward amount for the miner.
- This method performs the mining process to add a new block to the blockchain.
- It takes the `blockchain` as an argument to access the pending transactions and add the new block.
- It retrieves the pending transactions from the `pendingTransactions` map of the blockchain.
- It obtains the last block in the blockchain using the `getLastBlock` method.
- It calculates the previous block's hash by calling the `calculateHash` method on the last block.
- It initializes the `nonce` value to 0.
- It creates a new `Block` object with the pending transactions, previous hash, and nonce.
- It calculates the hash of the new block by calling the `calculateHash` method.
- It repeats the process of updating the nonce and recalculating the hash until the hash starts with four leading zeros .
- Once a suitable hash is found, a reward transaction is created with the sender as "SYSTEM", the recipient as the miner's name, and the reward amount.



BITCOIN SIMULATION

The `BitcoinSimulation` class serves as the entry point for the Bitcoin simulation program. It contains the `main` method, which is the starting point of execution.

- This is the main method that gets executed when the program starts.
- It takes an array of `String` arguments as input, although in this case, the `args` array is not used.
- It creates instances of `Agent`, `Miner`, and `Blockchain` classes to simulate the Bitcoin network.
- It performs a series of transactions between agents and adds them as pending transactions to the blockchain.
- It then processes the pending transactions in the blockchain by calling the `processPendingTransactions` method.
- If there are verified transactions, it allows the miner to mine a new block by calling the `mine` method on the `Miner` object.

END OF THE
PRESENTATION