Modeling Bitcoin Circulation in a Decentralized Network: Java Implementation

June 21, 2023

Abstract:

This report presents a detailed description of a Java implementation that models the circulation of bitcoins in a decentralized network. The model aims to simulate key features of blockchain technology, specifically focusing on preventing double spending of coins. Simplifying assumptions have been made to manage the complexity of real-life scenarios. The system allows for simulations involving the transfer of bitcoins between agents, transaction validation by competing miners, emergence of different blockchains, and testing the robustness against malicious agents and miners.

1. *Introduction*: The implemented model simulates a simplified version of a decentralized cryptocurrency system inspired by Bitcoin. It assumes a network of independent agents and miners without a central authority overseeing transactions. The primary objective is to prevent double spending through the use of a blockchain and competitive mining.

1. *System Description*: The system comprises the following components:

a) Agents: Each agent represents an independent entity in the network that possesses bitcoins and can initiate transactions with other agents. The agent maintains its own digital wallet, which stores the agent's bitcoin balance.

b) Miners: Miners validate transactions and construct new blocks in the blockchain. They compete to solve a cryptographic puzzle, with the winner adding their block to the blockchain. In the implemented model, miners utilize a proof-of-work mechanism for transaction validation.

c) Blockchain: The blockchain is a decentralized ledger that records all valid blocks in chronological order. It provides transparency, immutability, and security to the system. In the model, the blockchain is represented as a linked list of blocks, with each block containing a set of validated transactions.

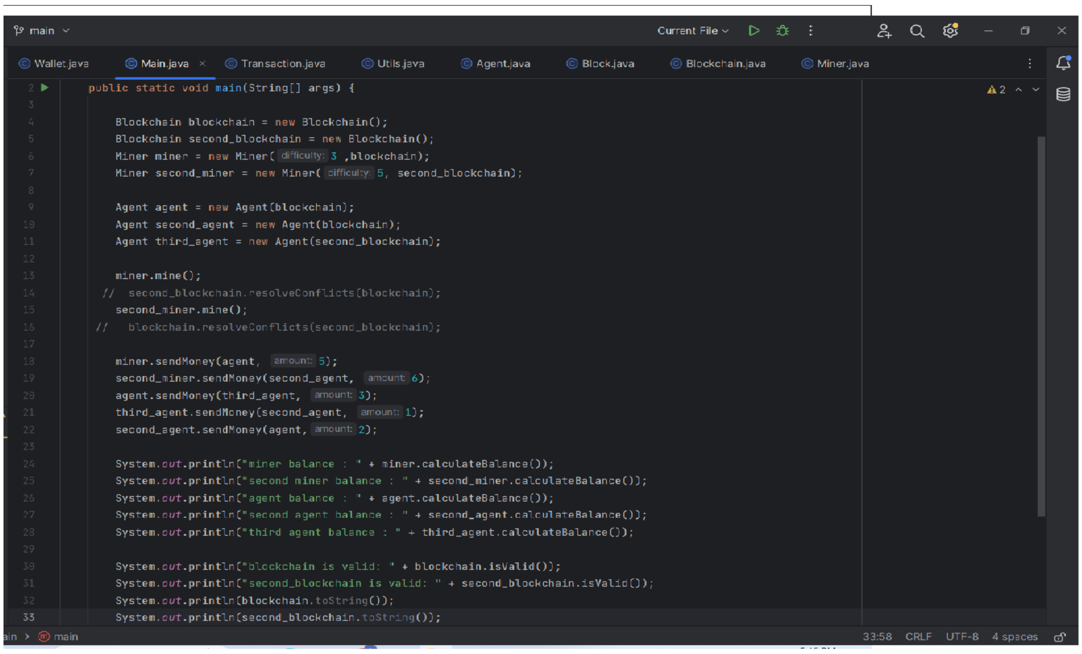
d) Block: The block class consists of methods for adding transactions and calculating the hash of a block, also a block is a list of transactions that has been signed and verified by the transaction object.

e) Transaction: it consists of methods for verifying and signing transactions also for calculating hash of new transactions.

f) Wallet: it is a class consisting of methods for checking that agent has enough money and also preventing double spending. Moreover, there are methods for calculating agents' balance and sending money to others.

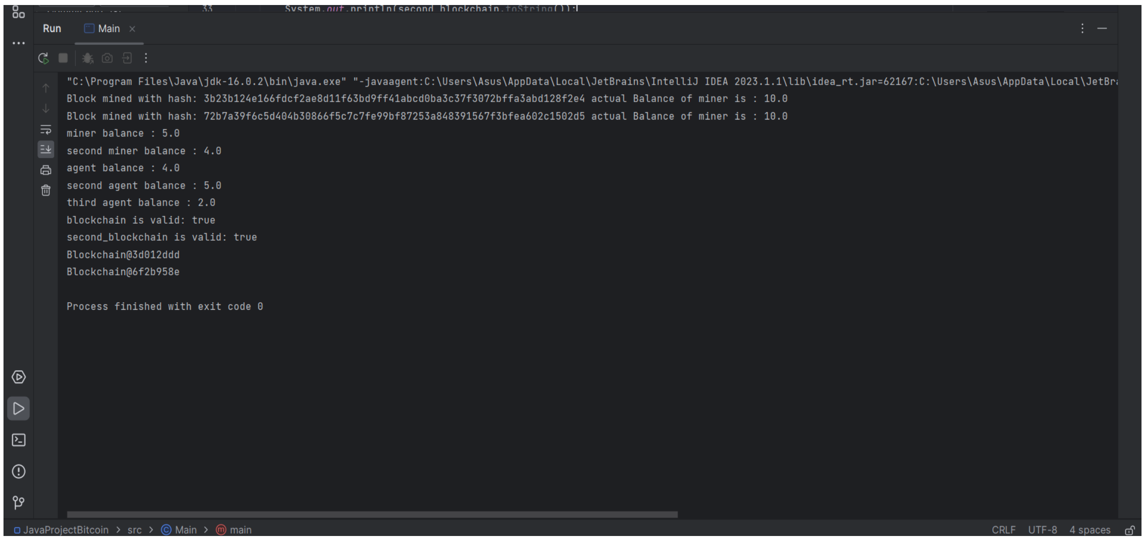
0.1 Testing the code:

1. You can find the main class and its code in the following picture:



```java
    public static void main(String[] args) {

        Blockchain blockchain = new Blockchain();
        Blockchain second_blockchain = new Blockchain();
        Miner miner = new Miner( difficulty: 3 ,blockchain);
        Miner second_miner = new Miner( difficulty: 5, second_blockchain);

        Agent agent = new Agent(blockchain);
        Agent second_agent = new Agent(blockchain);
        Agent third_agent = new Agent(second_blockchain);

        miner.mine();
  //    second_blockchain.resolveConflicts(blockchain);
        second_miner.mine();
 //     blockchain.resolveConflicts(second_blockchain);

        miner.sendMoney(agent,  amount: 5);
        second_miner.sendMoney(second_agent,  amount 6);
        agent.sendMoney(third_agent, amount: 3);
        third_agent.sendMoney(second_agent,  amount: 1);
        second_agent.sendMoney(agent, amount: 2);

        System.out.println("miner balance : " + miner.calculateBalance());
        System.out.println("second miner balance : " + second_miner.calculateBalance());
        System.out.println("agent balance : " + agent.calculateBalance());
        System.out.println("second agent balance : " + second_agent.calculateBalance());
        System.out.println("third agent balance : " + third_agent.calculateBalance());

        System.out.println("blockchain is valid: " + blockchain.isValid());
        System.out.println("second_blockchain is valid: " + second_blockchain.isValid());
        System.out.println(blockchain.toString());
        System.out.println(second_blockchain.toString());
```

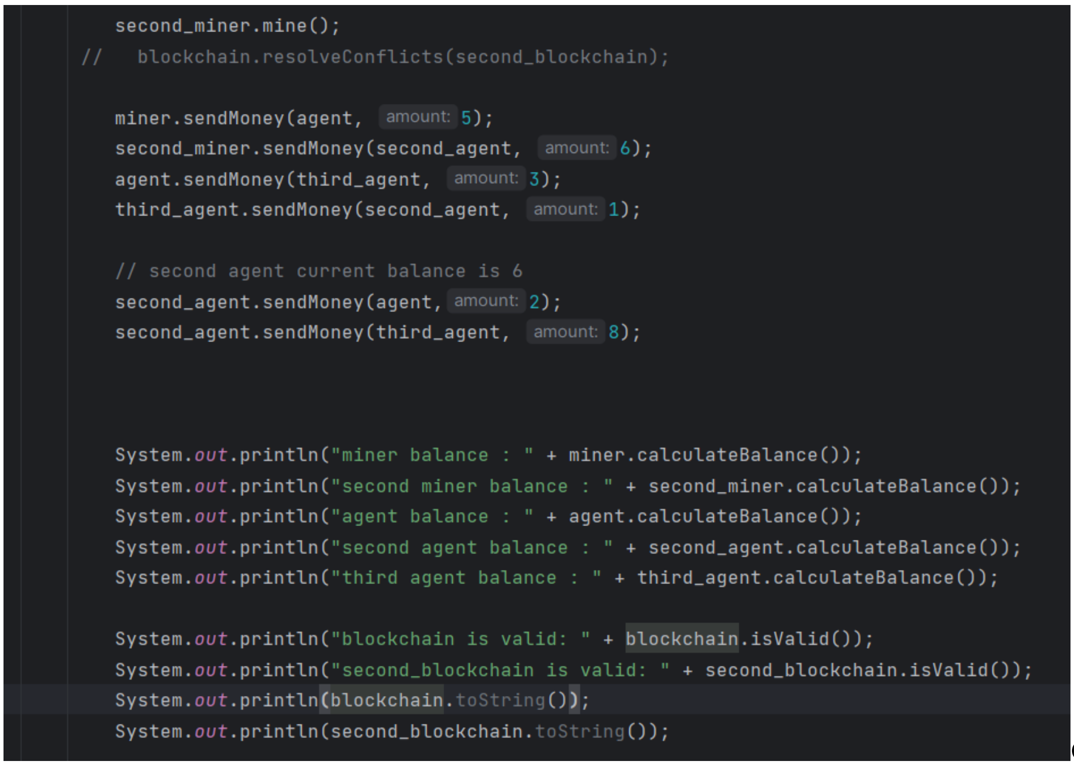
2. Here you can find the result of the code above:



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Users\Asus\AppData\Local\JetBrains\IntelliJ IDEA 2023.1.1\lib\idea_rt.jar=62167:C:\Users\Asus\AppData\Local\JetBra
Block mined with hash: 3b23b124e166fdcf2ae8d11f63bd9ff41abcd0ba3c37f3072bffa3abd128f2e4 actual Balance of miner is : 10.0
Block mined with hash: 72b7a39f6c5d404b30866f5c7c7fe99bf87253a848391567f3bfea602c1502d5 actual Balance of miner is : 10.0
miner balance : 5.0
second miner balance : 4.0
agent balance : 4.0
second agent balance : 5.0
third agent balance : 2.0
blockchain is valid: true
second_blockchain is valid: true
Blockchain@3d012ddd
Blockchain@6f2b958e

Process finished with exit code 0
```

Taking care of some possible errors:

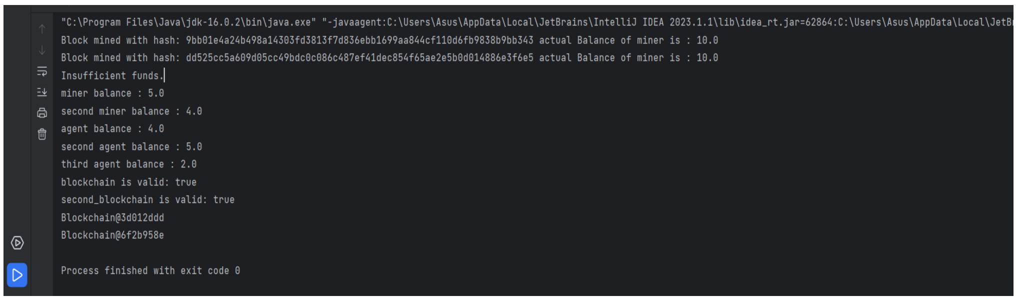
1. in this section you can find some possible errors on having not enough money and the agent tries to do the transaction:

1. 

```
    second_miner.mine();
//   blockchain.resolveConflicts(second_blockchain);

    miner.sendMoney(agent,  amount: 5);
    second_miner.sendMoney(second_agent,  amount: 6);
    agent.sendMoney(third_agent,  amount: 3);
    third_agent.sendMoney(second_agent,  amount: 1);

    // second agent current balance is 6
    second_agent.sendMoney(agent, amount: 2);
    second_agent.sendMoney(third_agent,  amount: 8);



    System.out.println("miner balance : " + miner.calculateBalance());
    System.out.println("second miner balance : " + second_miner.calculateBalance());
    System.out.println("agent balance : " + agent.calculateBalance());
    System.out.println("second agent balance : " + second_agent.calculateBalance());
    System.out.println("third agent balance : " + third_agent.calculateBalance());

    System.out.println("blockchain is valid: " + blockchain.isValid());
    System.out.println("second_blockchain is valid: " + second_blockchain.isValid());
    System.out.println(blockchain.toString());
    System.out.println(second_blockchain.toString());
```

Code:

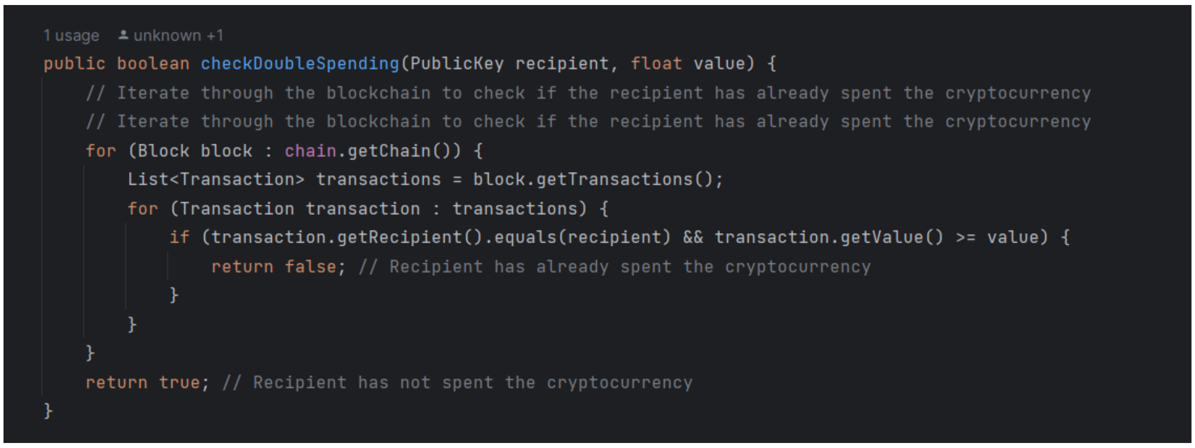
1. Result of error: you can see the error message as indicated “Insufficient funds".



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Users\Asus\AppData\Local\JetBrains\IntelliJ IDEA 2023.1.1\lib\idea_rt.jar=62864:C:\Users\Asus\AppData\Local\JetB
Block mined with hash: 9bb01e4a24b498a14303fd3813f7d836ebb1699aa844cf110d6fb9838b9bb343 actual Balance of miner is : 10.0
Block mined with hash: dd525cc5a609d05cc49bdc0c086c487ef41dec854f65ae2e5b0d014886e3f6e5 actual Balance of miner is : 10.0
Insufficient funds.
miner balance : 5.0
second miner balance : 4.0
agent balance : 4.0
second agent balance : 5.0
third agent balance : 2.0
blockchain is valid: true
second_blockchain is valid: true
Blockchain@3d012ddd
Blockchain@6f2b958e

Process finished with exit code 0
```

b) Double Spending Prevention: The implemented model can simulate scenarios where an agent attempts to double spend their cryptocoins. By running the simulation, it becomes evident that the blockchain's consensus mechanism prevents the successful double spending of coins.

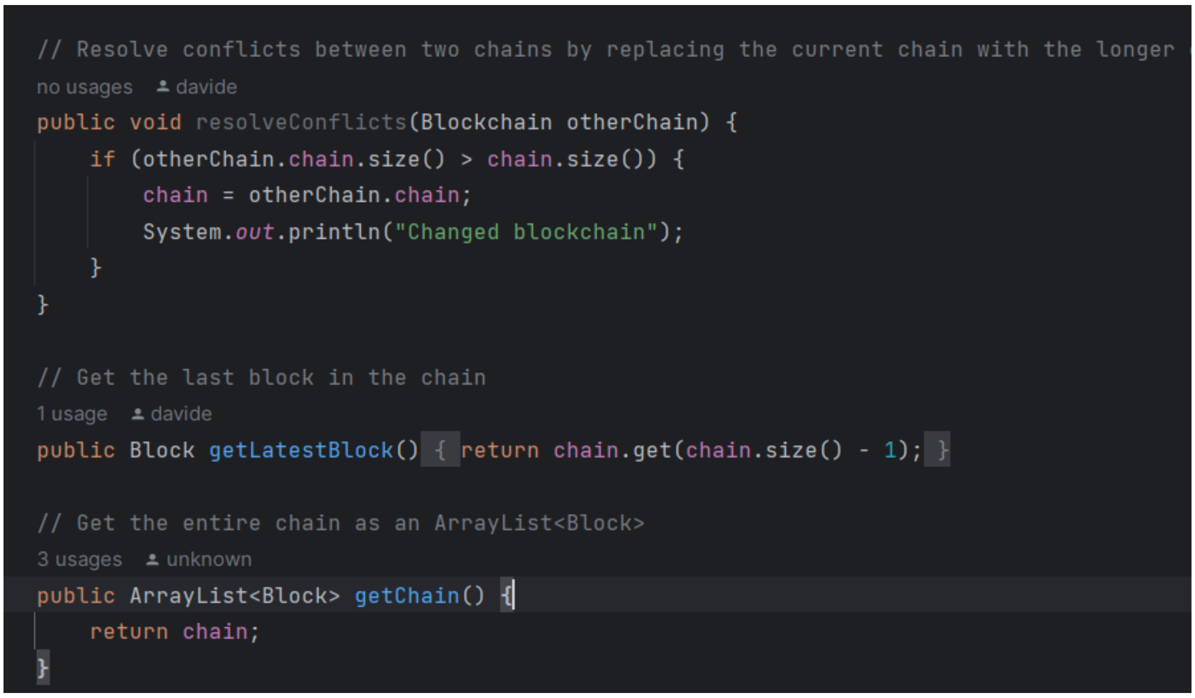
1. Code:



```java
1 usage  unknown +1
public boolean checkDoubleSpending(PublicKey recipient, float value) {
    // Iterate through the blockchain to check if the recipient has already spent the cryptocurrency
    // Iterate through the blockchain to check if the recipient has already spent the cryptocurrency
    for (Block block : chain.getChain()) {
        List<Transaction> transactions = block.getTransactions();
        for (Transaction transaction : transactions) {
            if (transaction.getRecipient().equals(recipient) && transaction.getValue() >= value) {
                return false; // Recipient has already spent the cryptocurrency
            }
        }
    }
    return true; // Recipient has not spent the cryptocurrency
}
```

c) Emergence of Multiple Blockchains: Simulations can illustrate scenarios where multiple competing blockchains emerge due to network delays or conflicts in block creation. The longest chain rule dictates which blockchain ultimately prevails so by the get chain method we can simply control the length of blockchains .

Example code snippet for blockchain emergence:



```java
// Resolve conflicts between two chains by replacing the current chain with the longer
no usages  davide
public void resolveConflicts(Blockchain otherChain) {
    if (otherChain.chain.size() > chain.size()) {
        chain = otherChain.chain;
        System.out.println("Changed blockchain");
    }
}

// Get the last block in the chain
1 usage  davide
public Block getLatestBlock() { return chain.get(chain.size() - 1); }

// Get the entire chain as an ArrayList<Block>
3 usages  unknown
public ArrayList<Block> getChain() {
    return chain;
}
```

0.2 Conclusion:

The implemented Java model provides a simulation of a decentralized cryptocurrency system, incorporating key elements of blockchain technology. Through the adoption of simplifying assumptions, the model demonstrates the prevention of double spending, emergence of multiple blockchains. Further refinement and expansion of the model can capture additional complexities and nuances of real-world cryptocurrency networks.