

Creating Blockchain with Java

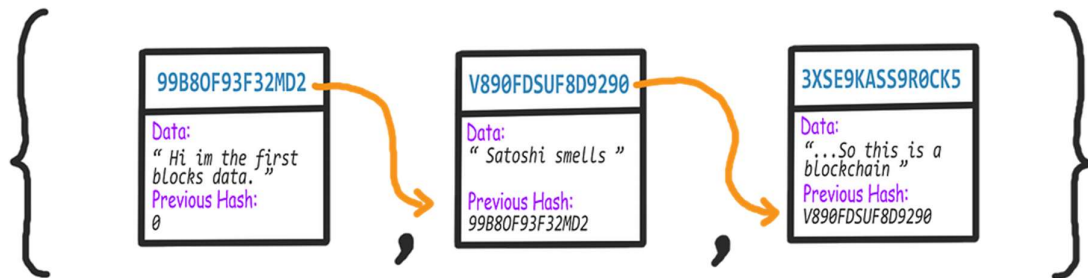
In this project we will try to understand blockchain technology by developing one in java.

Here are the steps that we follow:

1. Create your first (very) basic 'blockchain'.
2. Implement a simple proof of work (mining) system.
3. Marvel at the possibilities.

Making the Blockchain.

A blockchain is just a chain/list of blocks. Each block in the blockchain will have its own digital fingerprint, contain digital fingerprint of the previous block, and have some data (this data could be transactions for example).



[Hash = Digital Fingerprint]

Each block doesn't just contain the hash of the block before it, but its own hash is in part, calculated from the previous hash. If the previous block's data is changed then the previous block's hash will change (since it is calculated in part, by the data) in turn affecting all the hashes of the blocks there after. Calculating and comparing the hashes allow us to see if a blockchain is invalid.

What does this mean ? ...Changing any data in this list, will change the signature and break the chain.

So Firsts lets create class Block that make up the blockchain:

```
import java.util.Date;

public class Block {

    public String hash;

    public String previousHash;

    private String data; //our data will be a simple message.

    private long timeStamp; //as number of milliseconds since 1/1/1970.

    //Block Constructor.

    public Block(String data,String previousHash ) {

        this.data = data;

        this.previousHash = previousHash;

        this.timeStamp = new Date().getTime();

    }

}
```

As weu can see our basic Block contains a String hash that will hold our digital signature. The variable previousHash to hold the previous block's hash and String data to hold our block data.

Next we will need a way to generate a digital fingerprint,

There are many cryptographic algorithms you can choose from, however SHA256 fits just fine for this example. We can import java.security.MessageDigest; to get access to the SHA256 algorithm.

We need to use SHA256 later down the line so lets create a handy helper method in a new StringUtil 'utility' class :

```
import java.security.MessageDigest;

public class StringUtil {

    //Applies Sha256 to a string and returns the result.

    public static String applySha256(String input){
```

```

        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            //Applies sha256 to our input,
            byte[] hash = digest.digest(input.getBytes("UTF-8"));
            StringBuffer hexString = new StringBuffer(); // This will contain hash as
hexidecimal

            for (int i = 0; i < hash.length; i++) {
                String hex = Integer.toHexString(0xff & hash[i]);
                if(hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        }
        catch(Exception e) {
            throw new RuntimeException(e);
        }
    }
}

```

Now lets use our applySha256 helper, in a new method in the Block class, to calculate the hash. We must calculate the hash from all parts of the block we don't want to be tampered with. So for our block we will include the previousHash, the data and timeStamp.

```

public String calculateHash() {
    String calculatedhash = StringUtil.applySha256(
        previousHash +
        Long.toString(timeStamp) +
        data
    );
    return calculatedhash;
}

```

The first block is called the genesis block, and because there is no previous block we will just enter "0" as the previous hash.

Each block now has its own digital signature based on its information and the signature of the previous block.

Currently it's not much of a blockchain, so let's store our blocks in an ArrayList and also

```
import gson to view it as Json.
```

```
import java.util.ArrayList;
```

```
import com.google.gson.GsonBuilder;
```

```
public class NoobChain {
```

```
    public static ArrayList<Block> blockchain = new ArrayList<Block>();
```

```
    public static int difficulty = 5;
```

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

```
        //add our blocks to the blockchain ArrayList:
```

```
        blockchain.add(new Block("Hi im the first block", "0"));
```

```
        System.out.println("Trying to Mine block 1... ");
```

```
        blockchain.get(0).mineBlock(difficulty);
```

```
        blockchain.add(new Block("Yo im the second block",blockchain.get(blockchain.size()-1).hash));
```

```
        System.out.println("Trying to Mine block 2... ");
```

```
        blockchain.get(1).mineBlock(difficulty);
```

```
        blockchain.add(new Block("Hey im the third block",blockchain.get(blockchain.size()-1).hash));
```

```
        System.out.println("Trying to Mine block 3... ");
```

```
        blockchain.get(2).mineBlock(difficulty);
```

```
        System.out.println("\nBlockchain is Valid: " + isChainValid());
```

```

        String blockchainJson = new
GsonBuilder().setPrettyPrinting().create().toJson(blockchain);

        System.out.println("\nThe block chain: ");

        System.out.println(blockchainJson);

    }
}

```

Now we need a way to check the integrity of our blockchain.

Lets create an `isChainValid()` Boolean method in the `NoobChain` class, that will loop through all blocks in the chain and compare the hashes. This method will need to check the hash variable is actually equal to the calculated hash, and the previous block's hash is equal to the `previousHash` variable.

```

public static Boolean isChainValid() {

    Block currentBlock;

    Block previousBlock;

    String hashTarget = new String(new char[difficulty]).replace('\0', '0');

    //loop through blockchain to check hashes:
    for(int i=1; i < blockchain.size(); i++) {

        currentBlock = blockchain.get(i);

        previousBlock = blockchain.get(i-1);

        //compare registered hash and calculated hash:
        if(!currentBlock.hash.equals(currentBlock.calculateHash())) {

            System.out.println("Current Hashes not equal");

            return false;

        }

        //compare previous hash and registered previous hash
        if(!previousBlock.hash.equals(currentBlock.previousHash) ) {

            System.out.println("Previous Hashes not equal");

            return false;

        }

        //check if hash is solved
        if(!currentBlock.hash.substring( 0, difficulty).equals(hashTarget)) {

```

```

        System.out.println("This block hasn't been mined");
        return false;
    }
}
return true;
}

```

Any change to the blockchain's blocks will cause this method to return false.

On the bitcoin network nodes share their blockchains and the longest valid chain is accepted by the network. What's to stop someone tampering with data in an old block then creating a whole new longer blockchain and presenting that to the network ? Proof of work. The hashcash proof of work system means it takes considerable time and computational power to create new blocks. Hence the attacker would need more computational power than the rest of the peers combined.

Lets start mining blocks !!!

We will require miners to do proof-of-work by trying different variable values in the block until its hash starts with a certain number of 0's.

Lets add an int called nonce to be included in our calculateHash() method, and the much needed mineBlock() method :

Nonce: In the context of blockchain mining, a nonce is a number that miners incrementally change in order to find a hash value that meets certain criteria. This criteria often includes having a hash with a certain number of leading zeros, which is indicative of proof of work.

```

public void mineBlock(int difficulty) {
    String target = new String(new char[difficulty]).replace('\0', '0'); //Create a string with
difficulty * "0"
    while(!hash.substring( 0, difficulty).equals(target)) {
        nonce ++;
        hash = calculateHash();
    }
    System.out.println("Block Mined!!! : " + hash);
}

```

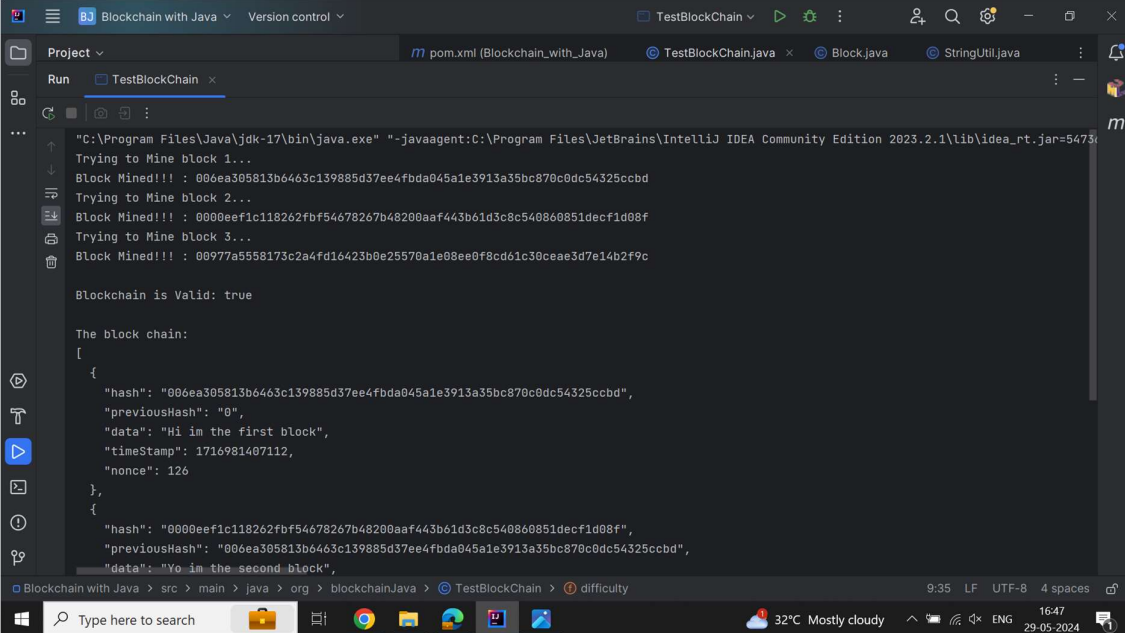
The mineBlock() method takes in an int called difficulty, this is the number of 0's they must solve for. Low difficulty like 1 or 2 can be solved nearly instantly on most computers, i'd suggest something around 4–6 for testing.

Lets add the difficulty as a static variable to the NoobChain class :

```
public static int difficulty = 5;
```

We should update the NoobChain class to trigger the mineBlock() method for each new block. The isValid() Boolean should also check if each block has a solved (by mining) hash.

OUTPUT:



```
C:\Program Files\Java\jdk-17\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.1\lib\idea_rt.jar=5473:..."
Trying to Mine block 1...
Block Mined!!! : 006ea305813b6463c139885d37ee4fbda045a1e3913a35bc870c0dc54325ccbd
Trying to Mine block 2...
Block Mined!!! : 0000eef1c118262fbf54678267b48200aaf443b61d3c8c540860851decf1d08f
Trying to Mine block 3...
Block Mined!!! : 00977a5558173c2a4fd16423b0e25570a1e08ee0f8cd61c30ceae3d7e14b2f9c

Blockchain is Valid: true

The block chain:
[
  {
    "hash": "006ea305813b6463c139885d37ee4fbda045a1e3913a35bc870c0dc54325ccbd",
    "previousHash": "0",
    "data": "Hi im the first block",
    "timeStamp": 1716981407112,
    "nonce": 126
  },
  {
    "hash": "0000eef1c118262fbf54678267b48200aaf443b61d3c8c540860851decf1d08f",
    "previousHash": "006ea305813b6463c139885d37ee4fbda045a1e3913a35bc870c0dc54325ccbd",
    "data": "Yo im the second block",
```