Kass Follow
pineapple express, star trek, late night programming, Desperados (with lime) & looking beyond 22338618.
Dec 16, 2017 · 6 min read

Creating Your First Blockchain with Java. Part 1.

The aim of this tutorial series, is to help you build a picture of how one could develop blockchain technology.

In this tutorial we will :

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

(I will assume you have a basic understanding of Object Oriented Programming)

It's worth noting that this wont be a fully functioning, ready for production block chain. Instead this is a proof of concept implementation to help you understand what a blockchain is for future tutorials.

You can support this and future tutorials :)
btc: 17svYzRv4XJ1Sfi1TSThp3NBFnh7Xsi6fu

. . .

Setting Up.

We will be using Java but you should be able to follow along in any OOP language. I'll be using Eclipse but you can use any new fancy text editor (though you'll miss out on a lot of good bloat).

You will need:

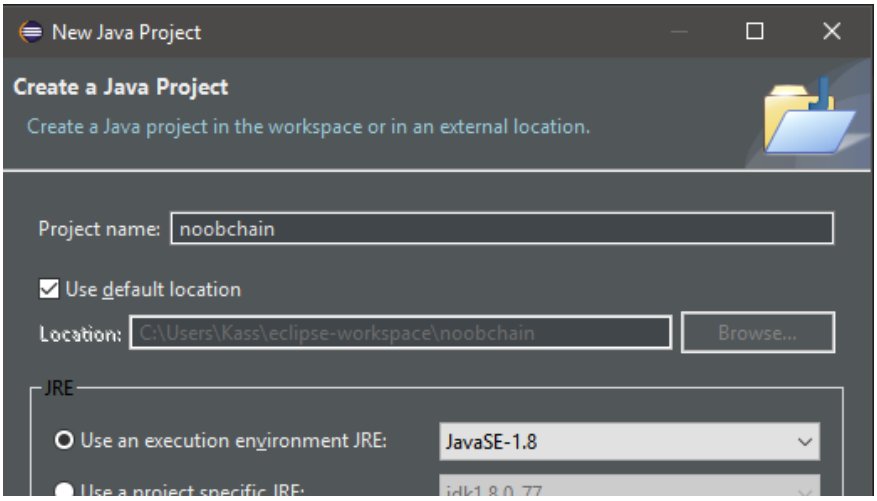
- Java and JDK installed. (duh).

- Eclipse (or another IDE/Text Editor).

Don't worry if your eclipse looks different to mine. I'll be using a dark theme in eclipse because ^

Optionally, you can grab GSON library by google (*who are they ???*). This will allow us to turn an object into Json \o/. It's a super useful library that we will also be using further down the line for peer2peer stuff, but feel free to use an alternate method.

In Eclipse create a (file > new >) Java project. I'll call my Project "**noobchain**" and create a new *Class* by the same name (**NoobChain**).



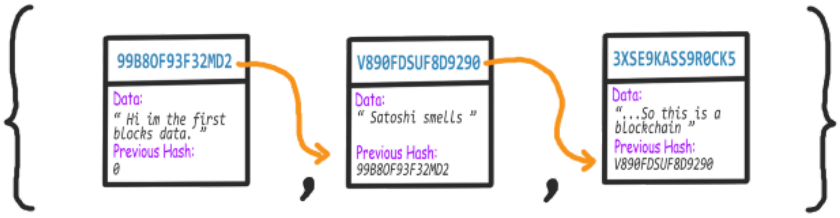
Don't be copying my project name now (ᵔ◡ᵔ)

Now you're good to go :)

. . .

Making the Blockchain.

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



I sure hope Nakamoto never sees this.

> **Hash = Digital Signature.**

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:

```
1  import java.util.Date;
2
3  public class Block {
4
5          public String hash;
6          public String previousHash;
7          private String data; //our data will be a simple me
8          private long timeStamp; //as number of milliseconds
9
10         //Block Constructor.
```

As you 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 signature,

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* :

```
1  import java.security.MessageDigest;
2
3  public class StringUtil {
4          //Applies Sha256 to a string and returns the result
5          public static String applySha256(String input){
6                  try {
7                          MessageDigest digest = MessageDiges
8                          //Applies sha256 to our input,
9                          byte[] hash = digest.digest(input.g
10                         StringBuffer hexString = new String
11                         for (int i = 0; i < hash.length; i+
12                                 String hex = Integer.toHexS
13                                 if(hex.length() == 1) hexSt
14                                 hexString.append(hex);
```

This is mostly a carbon copy of the http://www.baeldung.com/sha-256-hashing-java

Don't worry too much if you don't understand the contents of this helper method, *all you need to know is that it takes a string and applies SHA256 algorithm to it, and returns the generated signature as a string.*

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` .

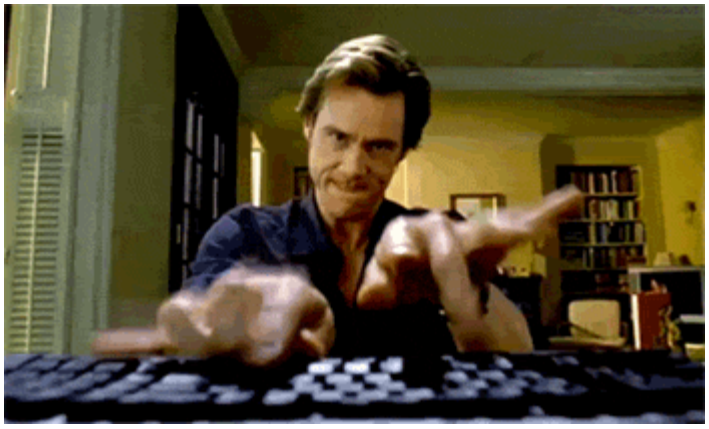
```
public String calculateHash() {
        String calculatedhash = StringUtil.applySha256(
                        previousHash +
                        Long.toString(timeStamp) +
                        data
                        );
```

and lets add this method to the **Block** *constructor*…

```
        public Block(String data,String previousHash ) {
                this.data = data;
                this.previousHash = previousHash;
                this.timeStamp = new Date().getTime();
                this.hash = calculateHash(); //Making sure w
```

Time for some testing…

In our main **NoobChain** class lets create some blocks and print the hashes to the screen to see that everything is in working order.

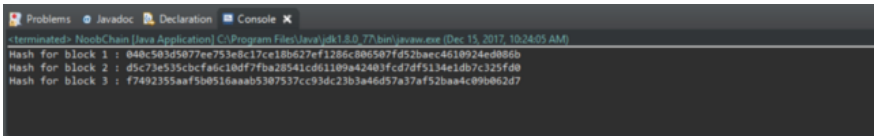


Lets test this…

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

```
1
2  public class NoobChain {
3
4          public static void main(String[] args) {
5
6                  Block genesisBlock = new Block("Hi im the f
7                  System.out.println("Hash for block 1 : " +
8
9                  Block secondBlock = new Block("Yo im the se
10                 System.out.println("Hash for block 2 : " +
```

The output should look similar to this:



Your values will be different because your timestamp will be different.

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 block**chain,** so lets store our blocks in an *ArrayList* and also import gson to view it as Json. *(click here to find out how to import the gson library)*

```
1  import java.util.ArrayList;
2  import com.google.gson.GsonBuilder;
3
4  public class NoobChain {
5
6          public static ArrayList<Block> blockchain = new Arr
7
8          public static void main(String[] args) {
9                  //add our blocks to the blockchain ArrayLis
10                 blockchain.add(new Block("Hi im the first b
11                 blockchain.add(new Block("Yo im the second
12                 blockchain.add(new Block("Hey im the third
```

Now our output should look something closer to what we expect a blockchain to look like.

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.

```
1  public static Boolean isChainValid() {
2          Block currentBlock;
3          Block previousBlock;
4
5          //loop through blockchain to check hashes:
6          for(int i=1; i < blockchain.size(); i++) {
7                  currentBlock = blockchain.get(i);
8                  previousBlock = blockchain.get(i-1);
9                  //compare registered hash and calculated ha
10                 if(!currentBlock.hash.equals(currentBlock.c
11                         System.out.println("Current Hashes
12                         return false;
13                 }
```

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.

hashcash, much wow.

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 :

```
import java.util.Date;

public class Block {

        public String hash;
        public String previousHash;
        private String data; //our data will be a simple me
        private long timeStamp; //as number of milliseconds
        private int nonce;

        //Block Constructor.
        public Block(String data,String previousHash ) {
                this.data = data;
                this.previousHash = previousHash;
                this.timeStamp = new Date().getTime();

                this.hash = calculateHash(); //Making sure
        }

        //Calculate new hash based on blocks contents
        public String calculateHash() {
                String calculatedhash = StringUtil.applySha
                                previousHash +
                                Long.toString(timeStamp) +
```

In reality each miner will start iterating from a random point. Some miners may even try random numbers for nonce. Also it's worth noting that at the harder difficulties solutions may require more than integer.MAX_VALUE, miners can then try changing the timestamp.

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. At the time of writing Litecoin's difficulty is around 442,592.

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

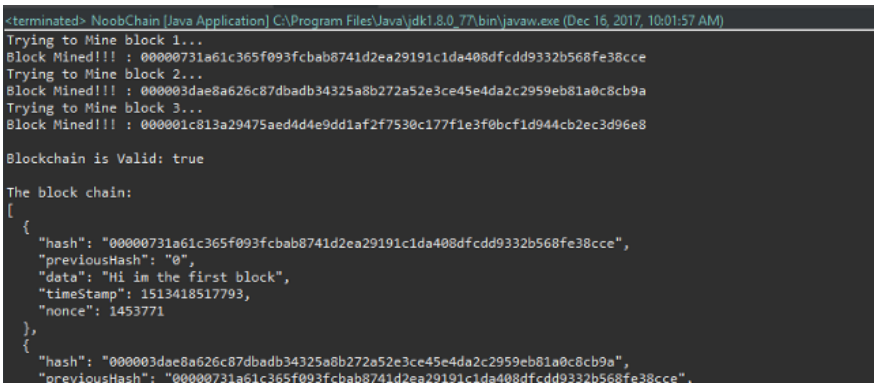
```
public static int difficulty = 5;
```

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

```
import java.util.ArrayList;
import com.google.gson.GsonBuilder;

public class NoobChain {

        public static ArrayList<Block> blockchain = new Arr
        public static int difficulty = 5;

        public static void main(String[] args) {
                //add our blocks to the blockchain ArrayLis

                blockchain.add(new Block("Hi im the first b
                System.out.println("Trying to Mine block 1.
                blockchain.get(0).mineBlock(difficulty);

                blockchain.add(new Block("Yo im the second
                System.out.println("Trying to Mine block 2.
                blockchain.get(1).mineBlock(difficulty);

                blockchain.add(new Block("Hey im the third
                System.out.println("Trying to Mine block 3.
                blockchain.get(2).mineBlock(difficulty);

                System.out.println("\nBlockchain is Valid:

                String blockchainJson = new GsonBuilder().s
                System.out.println("\nThe block chain: ");
                System.out.println(blockchainJson);
        }

        public static Boolean isChainValid() {
                Block currentBlock;
                Block previousBlock;
                String hashTarget = new String(new char[dif
```

Notice we also check and print isChainValid.

Running this your results should look like :



Mining each block took some time! (around 3 seconds) You should mess around with the difficulty value to see how that effects the time it takes to mine each block ;)

If someone were to **tamper** 😒 with the data in your blockchain system:

- Their blockchain would be invalid.
- They would not be able to create a longer blockchain.
- Honest blockchains in your network will have a time advantage on the longest chain.

A tampered blockchain will not be able to catch up with a longer & valid chain. *

*unless they have vastly more computation speed than all other nodes in your network combined. A future quantum computer or something.

You're all done with your basic blockchain!

Go on pat yourself on the back.

Your blockchain:
> Is made up of blocks that store data.
> Has a digital signature that chains your blocks together.
> Requires proof of work mining to validate new blocks.
> Can be check to see if data in it is valid and unchanged.

You can download these project files on Github.

You can **follow to be notified** when next tutorials and other blockchain development articles are posted. Any feedback is also greatly appreciated. Thanks.

Creating Your First Blockchain with Java. Part 2:

We cover **Transactions, Signatures** and **Wallets**.

contact: kassCrypto@gmail.com

Questions*:* https://discord.gg/ZsyQqyk *(I'm on the Blockchain developers Club discord)*