Blockchain with Java. Stage 1/6

Project description ↓

Blockchain essentials

Report a typo

Description

Blockchain has a simple interpretation: it's just a chain of blocks. It represents a sequence of data that you can't break in the middle; you can only append new data at the end of it. All the blocks in the blockchain are chained together.

Check out this great video about the blockchain. It uses a different approach to reach the final result of the project, which is cryptocurrencies, but it explains the blockchain pretty well.

To be called a blockchain, every block must include the **hash of the previous block**. Other fields of the block are optional and can store various information. The hash of a block is **a hash of all fields of a block**. So, you can just create a string containing every element of a block and then get the hash of this string.

Note that if you change one block in the middle, the hash of this block will also change. and the next block in the chain would no longer contain the hash of the previous block. Therefore, it's easy to check that the chain is invalid.

In the first stage, you need to implement such a blockchain. In addition to storing the hash of the previous block, every block should also have a unique identifier. The chain starts with a block whose id = 1. Also, every block should contain a timestamp representing the time the block was created. You can use the following code to get such a timestamp. This represents the number of milliseconds since 1 January 1970.

```
1  long timeStamp = new Date().getTime(); // 1539795682545 represents 17.10.2018, 20:01:22.545
```

By the way, since the first block doesn't have a previous one, its hash of the previous block should be 0.

The class Blockchain should have at least two methods: the first one generates a new block in the blockchain and the second one validates the blockchain and returns true if the blockchain is valid. Of course, the Blockchain should store all it's generated blocks. The validation function should validate all the blocks of this blockchain.

Also, for hashing blocks, you need to choose a good cryptographic hash function that is impossible to reverse-engineer. Insecure hash functions allow hackers to change the information of the block so that the hash of the block stays the same, so the hash function must be secure. A good example of a secure hash function is SHA-256. You can use this implementation of the SHA-256 hashing:

You should create 5 blocks in this stage. After the creation, validate the created blockchain using your validation method.

Example

The example below shows how your output might look. To be tested successfully, the program should output information about the first five blocks of the blockchain. Blocks should be separated by an empty line.

```
Block:
Id: 1
Timestamp: 1539810682545
Hash of the previous block:
0
Hash of the block:
796f0a5106c0e114cef3ee14b5d040ecf331dbf1281cef5a7b43976f5715160d

Block:
Id: 2
Timestamp: 1539810682557
Hash of the previous block:
796f0a5106c0e114cef3ee14b5d040ecf331dbf1281cef5a7b43976f5715160d
Hash of the block:
717242af079ccb7dd44c3f016936a81cf8ab2d4c1901243f30cbb7daa2060a0d

Block:
Id: 3
Timestamp: 1539810682558
Hash of the previous block:
717242af079ccb7dd44c3f016936a81cf8ab2d4c1901243f30cbb7daa2060a0d
Hash of the block:
28a2269bb34abd01dee9cea03400345bc9ea7322d73d3263221a47c6d970404f
```

Blockchain with Java. Stage 2/6

Project description ↓

A proof of work concept

Report a typo

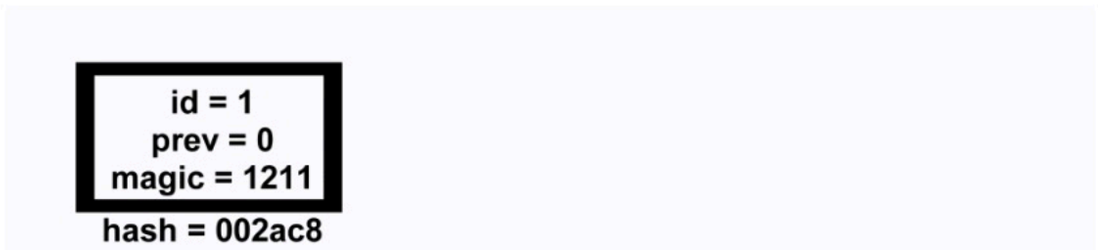
Description

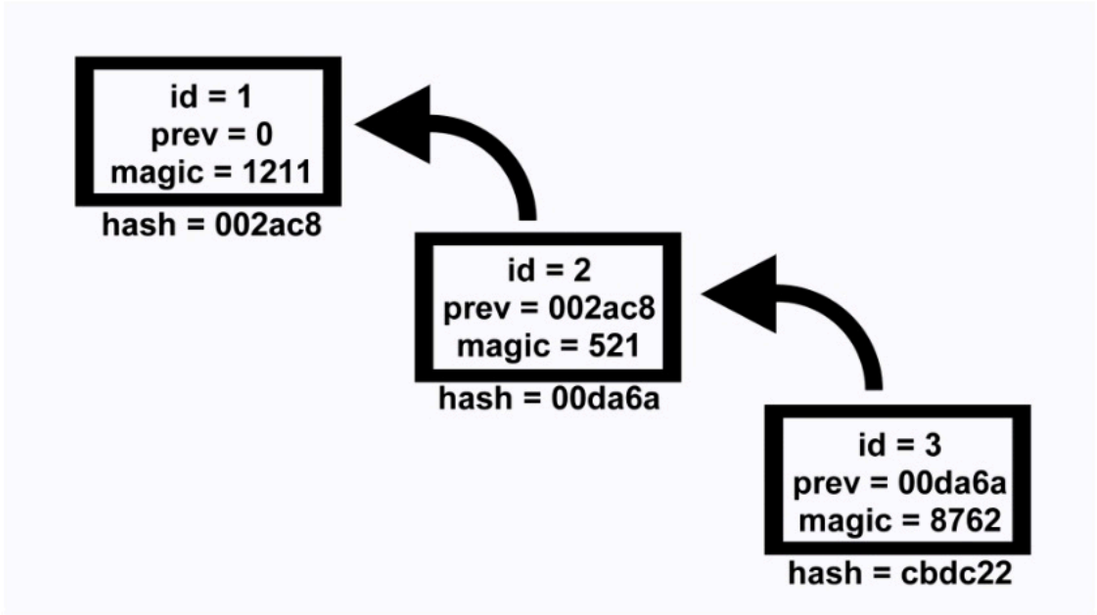
The security of our blockchain is pretty low. You can't just change some information in the middle of a blockchain, because the hash of this block will also be changed. And the next block still keeps the old hash value of the previous block. But can't we replace the old hash value with the new hash value so everything will be ok? No, because when you change the value of the previous hash in the block, the hash of this block will also be changed! To fix this, you need to change the value of the previous hash in the block after it. To solve this problem, you need to fix hash values in all the blocks until the last block of the blockchain!

This seems to be a pretty hard task to execute, doesn't it? If the time it takes to fix the hash value of the previous block is less than time to create a new block, we suddenly would be fixing blocks faster than the system can create them and eventually we will fix them all. The problem is that fixing the hash values is easy to do. The blockchain becomes useless if it is possible to change information in it.

The solution to this is called **proof of work**. This means that creating new blocks and fixing hash values in the existing ones should take time and shouldn't be instant. The time should depend on the amount of computational work put into it. This way, the hacker must have more computational resources than the rest of the computers of the system put together.

The main goal is that the hash of the block shouldn't be random. It should start with some amount of zeros. To achieve that, the block should contain an additional field: a **magic number**. Of course, this number should take part in calculating the hash of this block. With one magic number, and with another, the hashes would be totally different even though the other part of the block stays the same. But with the help of probability theory, we can say that there exist some magic numbers, with which the hash of the block starts with some number of zeros. The only way to find one of them is to make random guesses until we found one of them. For a computer, this means that the only way to find the solution is to brute force it: try 1, 2, 3, and so on. The better solution would be to brute force with random numbers, not with the increasing from 1 to N where N is the solution. You can see this algorithm in the animation below:





Obviously, the more zeros you need at the start of the block hash, the harder this task will become. And finally, if the hacker wants to change some information in the middle of the blockchain, the hash of the modified block would be changed and it won't start with zeros, so the hacker would be forced to find another magic number to create a block with a hash which starts with zeros. Note that the hacker must find magic numbers for all of the blocks until the end of the blockchain, which seems like a pretty impossible task, considering that the blockchain will grow faster.

It's said that the block is **proved** if it has a hash which starts with some number of zeros. The information inside it is impossible to change even though the information itself is open and easy to edit in the text editor. The result of the edit is a changed hash of the block, no longer containing zeros at the start, so this block suddenly becomes **unproved** after the edit. And since the blockchain must consist of only proved blocks, the whole blockchain becomes invalid. This is the power of the proof of work concept.

In this stage, you need to improve the blockchain. It should generate new blocks only with hashes that start with N zeros. The number N should be input from the keyboard.

Examples

The example below shows how your output might look. Output information about a few first blocks of the blockchain. Also, output the time that was needed to create a block. Your results and time measurements can be totally different than in the example! To be tested successfully, the program should output information about the first five blocks of the blockchain. Blocks should be separated by an empty line.

```
Enter how many zeros the hash must start with: 5

Block:
Id: 1
Timestamp: 1539827383396
Magic number: 24672386
Hash of the previous block:
0
Hash of the block:
00000a3fe20573b5bb358d2291165e15662a5b057240e954c573fb1f2a6d0cb8
Block was generating for 12 seconds

Block:
Id: 2
Timestamp: 1539827385414
Magic number: 87453465
Hash of the previous block:
00000a3fe20573b5bb358d2291165e15662a5b057240e954c573fb1f2a6d0cb8
Hash of the block:
000002e0ddd3c11e85466be0fa3dc5cb112daa7a3126e680c7d4f5716c0c6f9c
Block was generating for 21 seconds

Block:
Id: 3
Timestamp: 1539827387961
Magic number: 32734621
Hash of the previous block:
000002e0ddd3c11e85466be0fa3dc5cb112daa7a3126e680c7d4f5716c0c6f9c
Hash of the block:
000006edc10682ac3d511175b54192a7d36459af6e23671275c2c6879ab1c412
Block was generating for 18 seconds
```

Blockchain with Java. Stage 3/6

Project description ↓

Miner mania

Report a typo

Description

The blockchain itself shouldn't create new blocks. The blockchain just keeps the chain valid and accepts the new blocks from outside. In the outside world, there are a lot of computers that try to create a new block. All they do is search for a magic number to create a block whose hash starts with some zeros. The first computer to do so is a winner, the blockchain accepts this new block, and then all these computers try to find a magic number for the next block.

There is a special word for this: **mining**. The process of mining blocks is hard work for computers, like the process of mining minerals in real life is hard work. Computers that perform this task are called **miners**.

Note that if there are more miners, the new blocks will be mined faster. But the problem is that we want to create new blocks with a stable frequency. For this reason, the blockchain should regulate the number N: the number of zeros at the start of a hash of the new block. If suddenly there are so many miners that the new block is created in a matter of seconds, the complexity of the next block should be increased by increasing the number N. On the other hand, if there are so few miners that process of creating a new block takes longer than a minute, the number N should be lowered.

In this stage, you should create a lot of threads with miners, and every one of them should contain the same blockchain. The miners should mine new blocks and the blockchain should regulate the number N. The blockchain should check the validity of the incoming block (ensure that the previous hash equals the hash of the last block of the blockchain and the hash of this new block starts with N zeros). At the start, the number N equals 0 and should be increased by 1 / decreased by 1 / stays the same after the creation of the new block based on the time of its creation.

Example

To be tested successfully, program should output information about first five blocks of the blockchain. Blocks should be separated by an empty line.

```
Block:
Created by miner # 9
Id: 1
Timestamp: 1539866031047
Magic number: 23462876
Hash of the previous block:
0
Hash of the block:
1d12cbbb5bfa278734285d261051f5484807120032cf6adcca5b9a3dbf0e7bb3
Block was generating for 0 seconds
N was increased to 1

Block:
Created by miner # 7
Id: 2
Timestamp: 1539866031062
Magic number: 63576287
Hash of the previous block:
1d12cbbb5bfa278734285d261051f5484807120032cf6adcca5b9a3dbf0e7bb3
Hash of the block:
04a6735424357bf9af5a1467f8335e9427af714c0fb138595226d53beca5a05e
Block was generating for 0 seconds
N was increased to 2

Block:
Created by miner # 1
Id: 3
Timestamp: 1539866031063
Magic number: 57875299
Hash of the previous block:
04a6735424357bf9af5a1467f8335e9427af714c0fb138595226d53beca5a05e
Hash of the block:
0061924d48d5ce30e97cfc4297f3a40bc94dfac6af42d7bf366d236007c0b9d3
Block was generating for 0 seconds
N was increased to 3
```

Blockchain with Java. Stage 4/6

Project description ↓

You've got a message

Report a typo

Description

For now, we are mining blocks to create a blockchain, but just the blockchain itself is not particularly useful. The most useful information in the blockchain is the data that every block stores. The information can be anything. Let's create a simple chat based on the blockchain. If this blockchain works on the internet, it would be a world-wide chat. Everyone can add a line to this blockchain, but no one can edit it afterward. Every message would be visible to anyone.

In this stage, you need to upgrade the blockchain. A block should contain messages that the blockchain received during the creation of the previous block. When the block was created, all new messages should become a part of the new block, and all the miners should start to search for a magic number for this block. New messages, which were sent after this moment, shouldn't be included in this new block. Don't forget about thread synchronization as there is a lot of shared data.

You don't need any network connections as this is only a simulation of the blockchain. Use single blockchain and different clients that can send the message to the blockchain just invoking one method of the blockchain.

So, the algorithm of adding messages is the following:

1. The first block doesn't contain any messages. Miners should find the magic number of this block.
2. During the search of the current block, the users can send the messages to the blockchain. The blockchain should keep them in a list until miners find a magic number and a new block would be created.
3. After the creation of the new block, all new messages that were sent during the creation should be included in the new block and deleted from the list.
4. After that, no more changes should be made to this block apart from the magic number. All new messages should be included in a list for the next block. The algorithm repeats from step 2.

Blockchain with Java. Stage 5/6

Project description ↓

Matters of security

Report a typo

Description

How safe is your messaging system at the moment? Anyone can add a message to the blockchain. But can anyone impersonate you and send a message using your name? Without encryption, this is totally possible. There needs to be a method to verify that it is actually you who sent this message. Note that the registration/authorization method is bad because there is no server to check for a valid login/password pair. And if there is, it can be cracked by the hackers who can steal your password. There needs to be a whole new level of security.

Asymmetric cryptography solves this problem. With this, you can sign the message and let the signature be a special part of the message. You can generate a pair of keys: a public key and a private key. The message should be signed with a private key. And anyone can verify that the message and the signature pair is valid using a public key. The private key should be only on your computer, so no one from the internet can steal it. If you think that someone can steal your computer to get the private key, you can delete it from the computer and keep it in your head—that would be an example of maximum safety!

Please take a look at http://www.mkyong.com/java/java-digital-signatures-example/ for code examples for creating private and public keys and signing and verifying the message.
Now there is another problem. A hacker can't just take any message and sign it like it is your message, but he can take an already signed message and paste it into the blockchain again; the signature of this message stays the same, doesn't it? For this reason, all messages should contain a unique identifier, and all these identifiers should be in ascending order in the blockchain.

To get a unique identifier you should implement a method in the Blockchain class that always returns different numbers in ascending order starting from number 1.

In this stage, you need to upgrade the messages. The message should include the text of the message, the signature of this message, a unique identifier (remember to include a unique identifier when creating a signature), and a public key so everyone can check that this message is valid. Don't forget to check every message when checking that the blockchain is valid! The blockchain should reject the messages with identifier less than the maximum identifier in the block in which miners looking for the magic number. Also, when validating the blockchain you should check that every message has an identifier greater than the maximum identifier of the previous block.

Blockchain with Java. Stage 6/6

Project description ↓

Local currency

Report a typo

Description

Today, the most common application of blockchains is cryptocurrencies. A cryptocurrency's blockchain contains a list of transactions: everyone can see the transactions but no one is able to change them. In addition, no one can send a transaction as another person; this is possible using digital signatures. You have actually implemented all of this functionality in the previous stages.

A miner who creates a new block should be awarded some virtual money, for example, 100 virtual coins. This can be remembered in the blockchain if the block stores information about the miner who created this block. Of course, this message also should be proved, so the miner adds this information to the blockchain before starting a search for a magic number.

After that, a miner can spend these 100 virtual coins by giving them to someone else. In the real world, he can buy things and pay for them using these virtual coins instead of real money. These virtual coins go to the company that sells the things, and the company can pay salaries with these virtual coins. The circulation of these coins starts here and suddenly the virtual coins become more popular than real money!

To check how many coins a person has, you need to check all of his transactions and all of the transactions to him, assuming that the person started with zero virtual coins. The transaction should be rejected when the person tries to spend more money than he has at the moment. Create a special method that returns how many coins the person has.

In this stage, you need to implement transactions like this instead of text messages like in the previous stage. For testing reasons you can assume that everyone starts with 100 virtual coins, not 0. But as described above, all the money of the blockchain is initially awards for creating blocks of the blockchain.

Example

In the output example, VC stands for Virtual Coins. To be tested successfully, program should output information about first **fifteen** blocks of the blockchain. Blocks should be separated by an empty line.

```
Block:
Created by: miner9
miner9 gets 100 VC
Id: 1
Timestamp: 1539866031047
Magic number: 76384756
Hash of the previous block:
0
Hash of the block:
1d12cbbb5bfa278734285d261051f5484807120032cf6adcca5b9a3dbf0e7bb3
Block data:
No transactions
Block was generating for 0 seconds
N was increased to 1

Block:
Created by: miner7
miner7 gets 100 VC
Id: 2
Timestamp: 1539866031062
Magic number: 92347234
Hash of the previous block:
1d12cbbb5bfa278734285d261051f5484807120032cf6adcca5b9a3dbf0e7bb3
Hash of the block:
04a6735424357bf9af5a1467f8335e9427af714c0fb138595226d53beca5a05e
Block data:
miner9 sent 30 VC to miner1
miner9 sent 30 VC to miner2
miner9 sent 30 VC to Nick
Block was generating for 0 seconds
N was increased to 2
```