Amrita Vishwa Vidhyapeetham
Center for Computational Engineering and Networking

Workshop on Blockchain - Plan , Phase 1

15th May , 2021

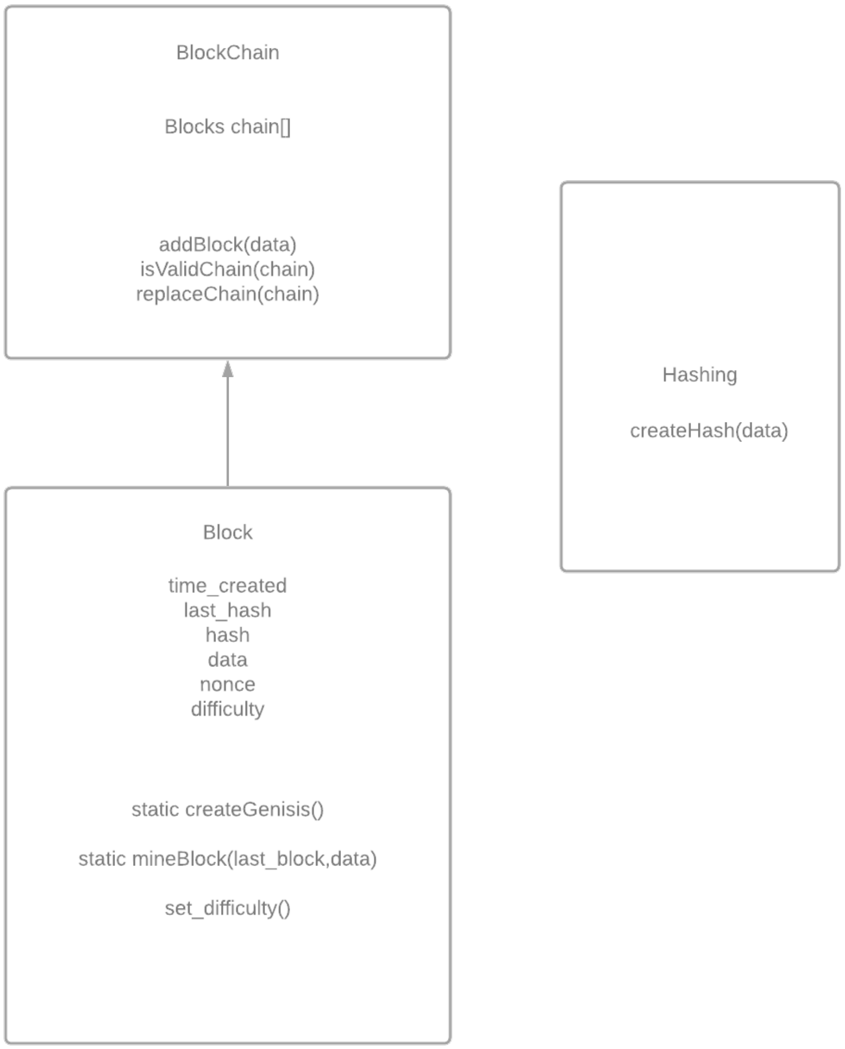
1 Phase 1 Plan

1.1 Hashing class

The Hashing class should contain helper encrypting functions to be used in other classes.one of the most commonly used cryptographic algorithm is SHA-256. We can use SHA-256 as well.

This class, at this phase is expected to contain one function createHash.

String createHash(data) This function should be able to take in variable number of arguments,and give out the SHA-256 encrypted string based on the incoming data. The encryption should be the same irrespective of the order of the data.
String hex2Bin(sting data)
Takes in a hexadecimal string and returns its binary equivalent.
String bin2Hex(sting data)
Takes in a binary string and returns its hexadecimal equivalent .


BlockChain
Blocks chain[]
addBlock(data)
isValidChain(chain)
replaceChain(chain)
Block
time_created
last_hash
hash
data
nonce
difficulty
static createGenisis()
static mineBlock(last_block,data)
set_difficulty()
Hashing
createHash(data)

1.2 Block Class

The Specification of Block class is given below

```
Class Block{

properties:
    time_created
    last_hash
    hash
    data
    nonce
    difficulty

methods:
    Block(data,last_hash,hash,time_created,nonce ,difficulty){
    timestamp = time of creating the block

    last_hash = hash of the previous block.
    THe has of the genisis block can be arbitrary

    hash = Hashing.createHash(time_created,last_hash,data,nonce,difficulty)

    nonce and difficulty must be dynamically adjusted.

    }

    static createGenisis(){
        return  new Block(GENISISDATA)

    }

    static mineBlock(last_block,data){
    a block new block must be returned , if there is Proof of work.
    One simple proof of work algorithm is HashCash.


    }

    static setDifficulty(Block,  time){
    Difficulty must be adjusted based on the mining rate parameter that is to be set.
    }

  }
```

1.2.1 Block class checklist

- Block has all the properties sepsified
- `static genesis()`

  Should return an instance of the block. And the Returned instance should contain a spesific genisis data.
- `static mineBlock(last_block , data)`
  - Should return an block instance
  - Should set the

    `hash`

    property of returned block to the hash of the previous block in the chain.
  - Should set the

    `data`

    property to the incoming data , and the time$_{c}reatedshouldbethecurrenttimeShouldcreateSHA-256hashbasedontheinputs.$

– Hash should also be based on

`difficulty`

– Difficulty should be adjusted.

- `setDifficulty(block,time)`

  .
  - If the time taken to mine the new block is less than the mining rate , then difficulty should be increased. If the time taken to mine the new block is high , hten difficulty should be decreased. The change in difficulty should always be 1
  - the difficulty should not go less than 1

click here to read more about Nonce and difficulty. Also see here for a more deeper dive. THe HashCash algorithm is used in bitcoins network.

1.3 Blockchain Class

The class description is below.

```
ass BlockChain{

ock chain[]; // Dynamic Array of blocks.
Each  Block should contain a reference to its previous block

ockchain(){
ain[0] = Block.genesis()
creates genisis block


dBlock(data){
is should mine a new block ,
sed on the previous block and data.
e mined block should be added in the end of
e chain


atic isValidChain(chain){
ecks if a chain is valid.
lidity criterion is given below



atic replaceChain(chain){

ould check if the incoming chain is valid and
place hte chain of hte blockchain with the incoming chain






```

BlockChain class checklist

- Should contain a

  `Block chain[]`

- Should start with THE genisis block
- `isValidChain()`
  - Chain should start with THE genesis block

  - Should return False if a lasthash reference is tampered
  - Should return false if the block has n invalid feild
  - Should return false if the chain contains a jumped difficulty. (Succesive difficulties should always vary by 1)
  - If all the blocks are valid and none of the above cases hold , then return true

- `replaceChain(chain[])`
  - WHen the incoming chain is shorter than the Block-chain.length , do not replace.
  - WHne the incoming chain is longer than the block chain .length , replace , iff the chain

    `isValid(chain)`