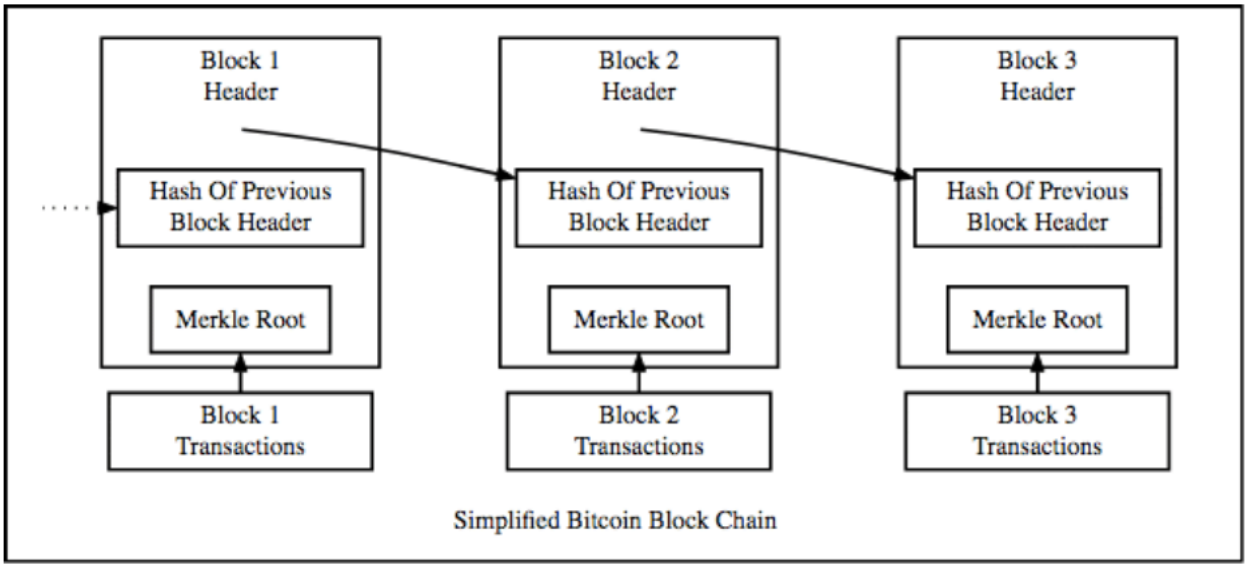
Program 3: Blockchain



Objectives:

- Hashing
- Blocks
- Blockchain
- Proof of Work
- Merkle Trees
- Merkle Root

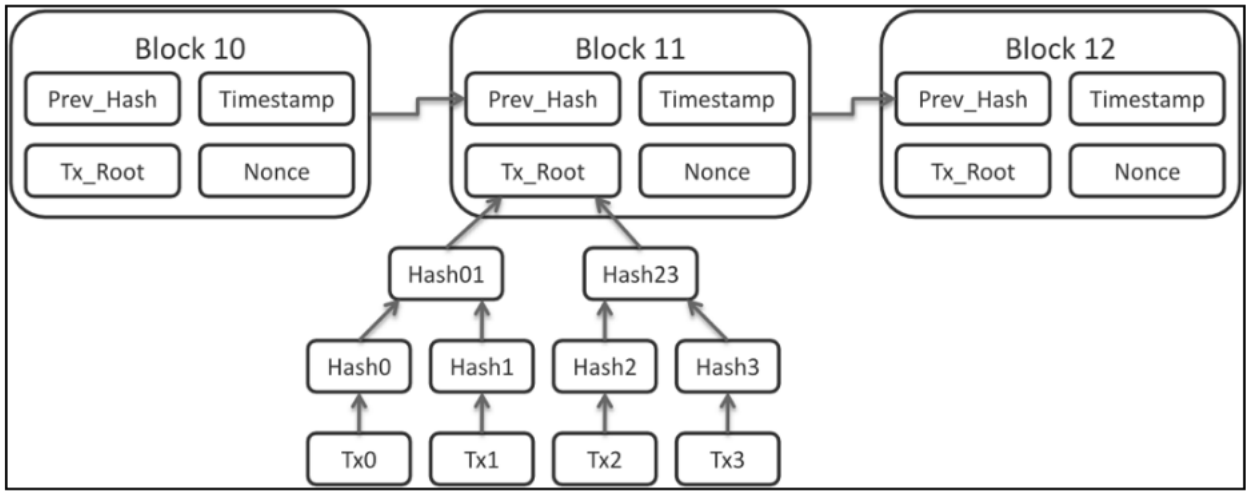
In simple terms, **hashing** means taking an input string of any length and giving out an output of a fixed length. In the context of cryptocurrencies like Bitcoin, the transactions are taken as an input and run through a hashing algorithm (Bitcoin uses SHA-256) which gives an output of a fixed length (256 bits).

The **blockchain** is a digital concept to store data. This data comes in blocks, so imagine blocks of digital data. These blocks are chained together, and this makes their data immutable. When a block of data is chained to the other blocks, its data can never be changed again. It will be publicly available to anyone who wants to see it ever again, in exactly the way it was once added to the blockchain. That is quite revolutionary, because it allows us to keep track records of pretty much anything we can think of (to name some: property rights, identities, money balances, medical records), without being at risk of someone tampering with those records.

It is essentially a linked list which contains data and a hash pointer pointing to its previous block, hence creating the chain. A hash pointer is similar to a pointer, but instead of just containing the address of the previous block, it contains the hash of the data inside the previous block. This one small tweak is what makes blockchains so amazingly reliable and trailblazing.

Imagine this for a second, a hacker attacks block 3 and tries to change the data. Because of the properties of hash functions, a slight change in data will change the hash drastically. This means that any slight changes made in block 3, will change the previous hash which is stored in block 4, now that in turn will change the data and the hash of block 4 which will result in changes in block 5 and so on and so forth. This will completely change the chain, which is impossible. This is exactly how blockchains attain immutability.

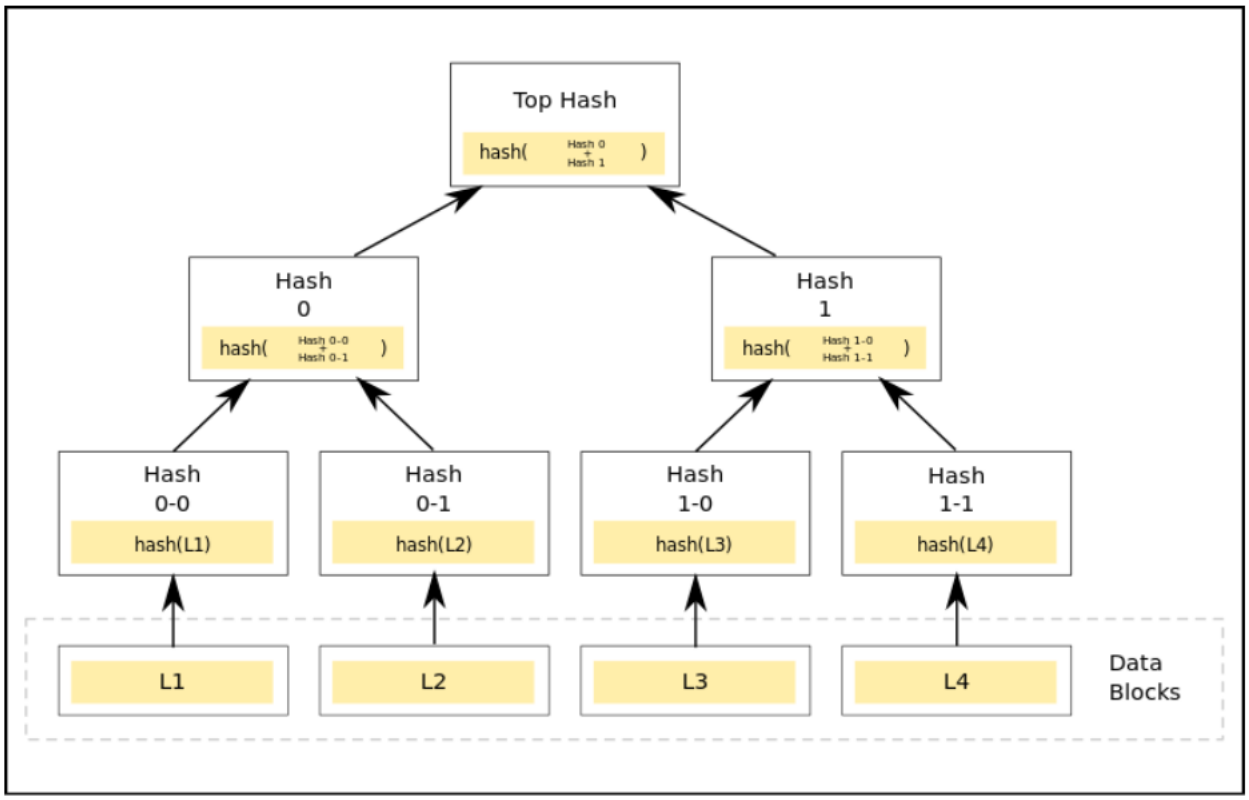
So, what is inside of a single Block?



A block contains:

- Time: the current timestamp.
- Block Number: The current block number.
- Merkle Tree (to obtain a Merkle Root & the number of transactions within the block)
- The current difficulty target hash. (More on this later).
- The previous block (to obtain the previous block hash)
- Nonce (more on this later).
- Hash (created by concatenating the time stamp, block number, Merkle root, difficulty, & previous hash and then generating a SHA256 hash)

The Merkle Root is the root of a Block's Merkle Tree:



The above diagram shows what a Merkle tree looks like. In a Merkle tree, each non-leaf node is the hash of the values of their child nodes.

Leaf Node: The leaf nodes are the nodes in the lowest tier of the tree. So, from the diagram above, the leaf nodes will be L1, L2, L3 and L4.

Child Nodes: For a node, the nodes below its tier which are feeding into it are its child nodes. From the diagram, the nodes labeled “Hash 0-0” and “Hash 0-1” are the child nodes of the node labeled “Hash 0”.

Root Node: The single node on the highest tier labeled “Top Hash” is the root node. Also known as the **Merkle Root.**

Each block contains thousands and thousands of transactions. It will be very time inefficient to store all the data inside each block as a series. Doing so will make finding any particular transaction extremely cumbersome and time-consuming. If you use a Merkle tree, however, you will greatly cut down the time required to find out whether a particular block has been tampered with or not.

Mining A Block:

When the Bitcoin mining software wants to add a new block to the blockchain, this is the procedure it follows. Whenever a new block arrives, all the contents of the blocks are first hashed. If the hash is equal to the difficulty target hash (a String of 0’s, with the size equal to the difficulty level), then it is added to the blockchain and everyone in the community acknowledges the new block.

However, it is not as simple as that. You will have to be extremely lucky to get a new block just like that. This is where the nonce comes in. The **nonce** is an arbitrary string which is concatenated with the hash of the block. After that, this concatenated string is hashed again and compared to the difficulty target hash. If it is not equal to the difficulty target hash, then the nonce is changed and this keeps on repeating until finally, the requirements are met. When that happens, the block is added to the blockchain.

The entire process is completely random, there is no thought process behind the selection of the nonces. It is just pure brute-force where the software keeps on randomly generating strings until they reach their goal. The entire process follows the **Proof of Work** protocol which basically means:

- The puzzle solving should be difficult.
- Checking the answer should, however, be easy for everyone. This is done to make sure that no underhanded methods were used to solve the problem.

Helpful Links:

- https://blockgeeks.com/guides/what-is-hashing/
- https://blog.goodaudience.com/blockchain-for-beginners-what-is-blockchain-519db8c6677a
- https://hackernoon.com/merkle-trees-181cb4bc30b4

Part I: The Merkle Tree

Using the provided comments as a guide, complete the MerkleTree class. This class will store our list of transactions and generate the Merkle Root needed for our Block class. Normally, a Merkle Tree would also have the ability to store all computed hashes and perform quick tree searches in order to verify that a specific transaction exists within a block, however, for our purposes, we are just interested in validating the entire block via the Merkle Root (although feel free to implement any additional features for testing purposes). ArrayList<String> getMerkleRoot(ArrayList<String> hashTranList) is a recursive method that will hash pairs of transactions and store these hashes in a temporary ArrayList, until we only have a single item (The Merkle Root) remaining.

You can use the getSHA() method within the SHA256 class to perform your hashes. You can check your answers by using a SHA256 calculator.

Part II: The Block

Complete the Block class. This class represents a single Block in the Blockchain and stores the following information: The Block Number, A Merkle Tree (to obtain the Merkle Root & number of transactions), The Previous Block (to obtain the previous block hash), a difficulty level, a timestamp, and a nonce (number only used once). The Block's hash is calculated by concatenating the timestamp, the difficulty, the previous hash, the block number, and the Merkle Root together and then performing a SHA256 hash.

A block is "Mined" by utilizing the Proof of Work concept. The first n hex digits of the Block's hash will be compared to a target hash (a String of n 0's), with n being equal to the difficulty level. The difficulty level should be between 1 – 5. At a 1, a Block will be mined almost instantaneously, however, the greater the difficulty level entered, the longer it will take to mine a block. If a subset of the hash is not equal to the target hash, a new hash will be generated by concatenating the Block's hash with the nonce value and then performing a SHA256 hash. The nonce should be incremented after each check (this is essentially a Brute Force Search). I would suggest starting your tests with a difficulty level of 1, although your program should allow for a difficulty level of up to 5.

Part III: The Blockchain

Complete the Blockchain class (this is also our driver & GUI). An argument should be provided (1-5) to set your difficulty. If no argument is provided, set a default difficulty. You will be instantiating instance variables and completing the functionality of the "Sign", "Merkle Root", and "Mine Block" buttons. While most of the GUI is completed, the JComboBox getSelectedItem() & JTxtAmount getText() methods will be useful here. The "Sign" button will add a transaction to our model and verify the digital signature of our seller (The verifySignature() method within the given DigitalSignature class will be useful here). The "Merkle Root" button will create a new Merkle Tree based on the previously signed transactions. The "Mine Block" button will create a new Block (passing in the appropriate information), mine that block, and then update the previous block (so that we can link new Blocks to the previous Block).

Submission: please submit a zip containing all relevant files, including a README.txt on how you would like the graders to compile & run your program (I included a batch file, so feel free to use that).