Implementation of Blockchain

A Step-by-Step Guide

[BATCH-D]

OBJECT ORIENTED PROGRAMMING(OOPS)

Presented by,

Prisha Gupta [CB.SC.U4AIE23331]

Naksh Singh [CB.SC.U4AIE23361]

Vedant Maheshwari [CB.SC.U4AIE23346]

Sri Krishna Karthikeya [CB.SC.U4AIE23350]

AMRITA VISHWA VIDYAPEETHAM

Introduction

- Purpose:
  - To explain how to implement a simple blockchain and transaction system in Java.

- Overview:
  - We will cover the Block.java, Transaction.java, and BlockchainNotary.java files.

What is a Blockchain?

Definition:

A blockchain is a distributed digital ledger that records transactions across many computers so that the record cannot be altered retroactively without the alteration of all subsequent blocks and the consensus of the network.

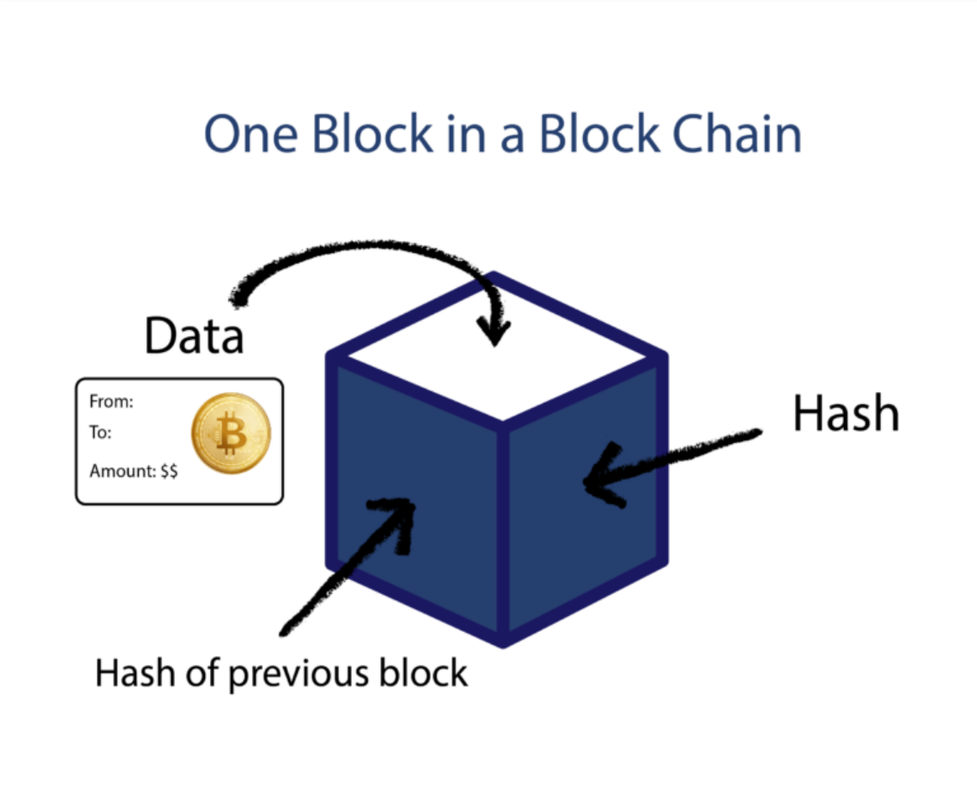
Key Components:

- Block: Contains transaction data.
- Chain: A sequence of blocks linked together.
- Hash: A unique identifier for the block, created using cryptographic hash functions.
- Proof of Work (PoW): A consensus algorithm to validate transactions and create new blocks by solving computational puzzles.

What is a Block?

Definition:

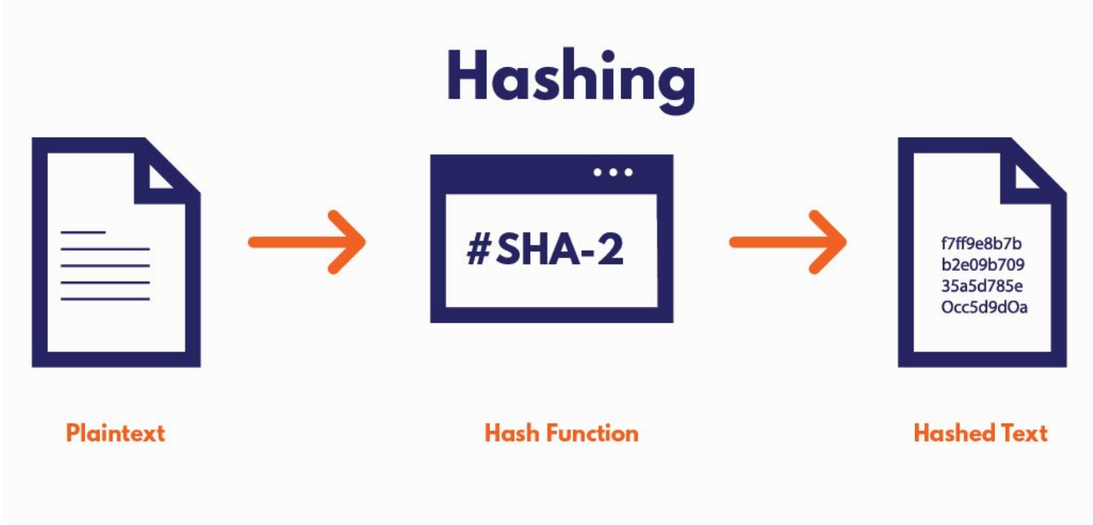
A Block is a primary unit of blockchain. In the blockchain, a block is a collection of data or information.
A block consists of Block number, nonce, data, previous hash, and hash.
A block has only one parent. The first block of a blockchain is called the ‘genesis block’. The ‘genesis block’ has no parent block.



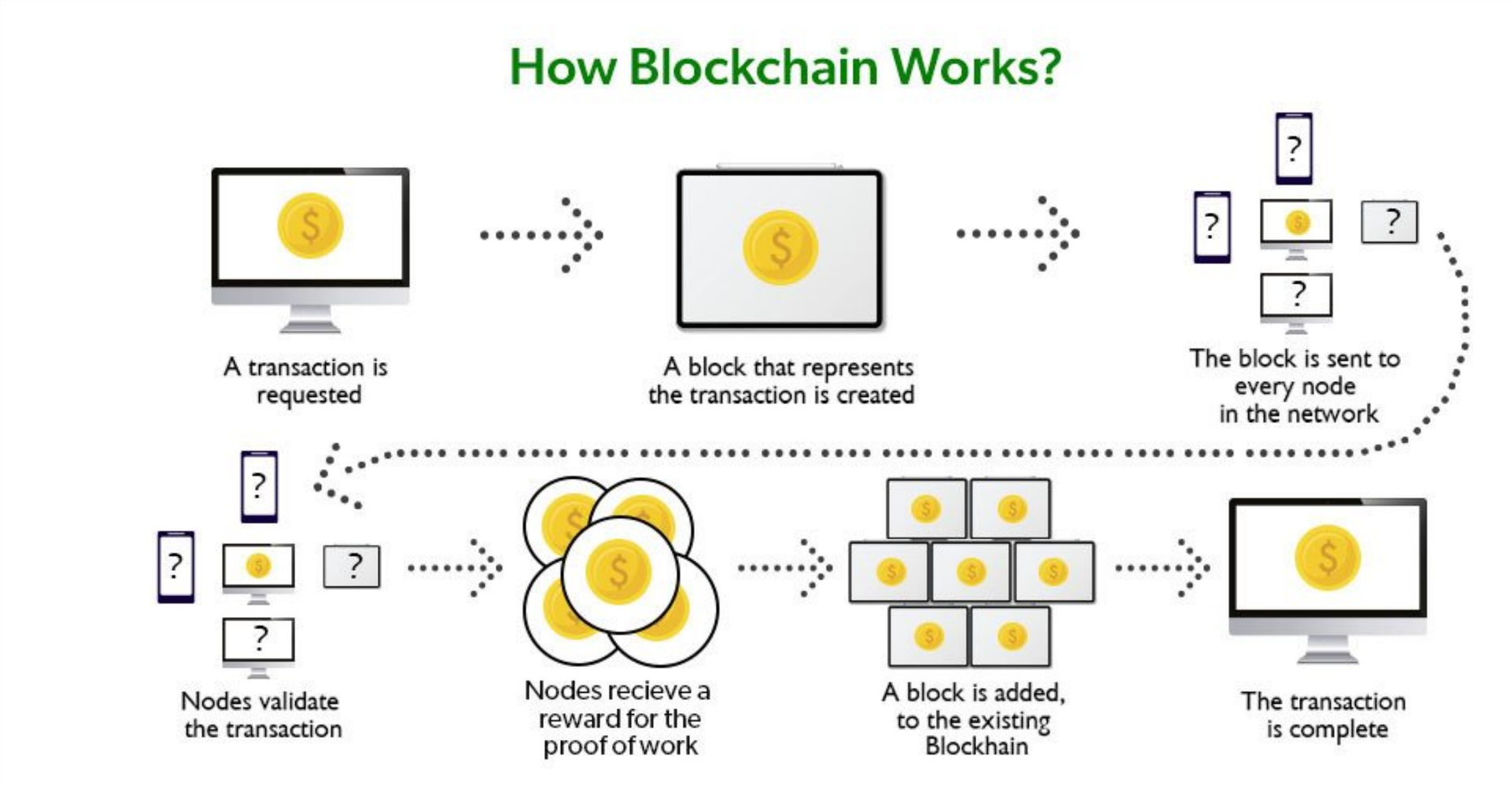
Key Blockchain Components

HASH

A hash is a cryptographic function that converts input data into a fixed-size string of characters. It ensures data integrity by producing a unique output for each unique input—even a slight change in the input data results in a vastly different hash. Hashes link blocks together, maintaining the security and immutability of the blockchain.



Working of Blockchain



How Transactions Work in a Blockchain

- **Transactions:**
  Records of data transfer (e.g., transfer of value).

- **Blocks:**
  Each block contains a list of transactions, a timestamp, a nonce, the hash of the previous block, and its own hash.

- **Linking Blocks:**
  Each block's hash is generated from its contents, including the hash of the previous block, thus creating a chain of blocks.

Overview of the Implementation

- Structure:
  - *Block.java:* Represents individual blocks.
  - *Blockchain.java:* Represents a blockchain.
  - *Transaction.java:* Represents individual transactions.
  - *BlockchainNotary.java:* Managestheblockchain.

- Flow:
  - Step 1: Transactions are created and represented as strings.
  - Step 2: Transactions are included in a block.
  - Step 3: Blocks are mined (proof of work) and added to the blockchain.
  - Step 4: The integrity of the blockchain is maintained by linking each block to the previous block via cryptographic hashes.

Block Class (*Block.java*)

Purpose:

- Represents a single block in the blockchain.

- Key Components:
  - Variables:
    - *hash:* The SHA-256 hash of the block.
    - *previousHash:* The hash of the previous block in the chain.
    - *data:* The transaction data stored in the block.
    - *timeStamp*: The time when the block was created.
    - *nonce:* A number used once to vary the input of the hash function.
  - Constructor:
    - Initializes the block with its data, the hash of the previous block,the current timestamp, and calculates the initial hash of the block.
  - Methods:
    - `calculateHash()`: Combines the block's properties and generates a SHA-256 hash.
    - `mineBlock(int difficulty)`: Adjusts the nonce until the block's hash meets a difficulty requirement (e.g., starts with a certain number of zeros).
    - `applySha256(String input)`: A static method that returns the SHA-256 hash of the input string.

Block Class - *Key Concepts*

- Hashing:
  - A unique hash identifies the block and its contents. Including the previousHash ensures that each block is cryptographically linked to the preceding block.

- Mining:
  - Mining is the process of finding a hash that meets the difficulty requirement. It ensures security and integrity by requiring computational work.

- Nonce:
  - A nonce is varied during mining to find a valid hash, ensuring different hashes for the same block content until a valid one is found.

Blockchain Class (*Blockchain.java*)

Purpose:

- To manage a chain of blocks in a blockchain system.
- Ensure the integrity and validity of the blockchain by validating each block before adding it to the chain.

Key Components:

- Variables:
  - *List<Block> chain:* Stores the list of blocks in the blockchain.
- Constructor:
  - Initializes the blockchain with a genesis block.
  - Sets initial difficulty and miner address for the genesis block.
- Methods:
  - `addBlock(Block block)`: Adds a new block to the chain if it's valid.
  - `validateBlock(Block block)`: Validates a block by checking its previous hash and hash difficulty.
  - `getChain()`: Returns the list of blocks in the chain.
  - `isChainValid()`: Checks the validity of the entire blockchain by validating each block.

Blockchain Class - *Key Concepts*

- Genesis Block:
  - The first block in the blockchain, setting the foundation for subsequent blocks.

- Block Validation:
  - Ensures previous hash matches the last block.
  - Checks if the block's hash meets difficulty.
  - Verifies the integrity of the block.

- Chain Integrity:
  - Validates the blockchain by checking each block.
  - Ensures continuity and immutability.

Transaction Class (*Transaction.java*)

Purpose:

- Represents a single transaction in the blockchain.

- Key Components:
  - Variables:
    - *sender:* The sender's identifier (e.g., an address or name).
    - *recipitent:* The recipient's identifier.
    - *amount:* The amount being transferred.
  - Constructor:
    - Initializes the transaction with the sender, recipient, and amount.
  - Methods:
    - `toString()`: Returns a string representation of the transaction for inclusion in a block's data field.

Transaction Class - *Key Concepts*

- Transaction:
  - Each transaction records an action, such as the transfer of value from one party to another.

- String Representation:
  - The `toString()` method ensures that transaction data can be easily converted into a format suitable for inclusion in a block's data field.

- Nonce:
  - A nonce is varied during mining to find a valid hash, ensuring different hashes for the same block content until a valid one is found.

BlockchainNotary Class (*BlockchainNotary.java*)

Purpose:

- Manages the entire blockchain, including creating and mining blocks and adding transactions.

- Key Components:
  - Variables:
    - `Blockchain`: A list that stores all the blocks in the chain.
    - `difficulty`: Defines the difficulty level for the mining process.
  - Main method:
    - `Genesis Block`: Initializes the blockchain with the genesis block, the first block in the chain.
    - `Subsequent Blocks`: Creates and mines subsequent blocks that contain transactions.
  - Methods:
    - `addBlock(Block newBlock)`: Mines the new block and adds it to the blockchain.
    - `createBlockWithTransaction(String sender, String recipient, double amount)`: Creates a block containing a transaction.
    - `isChainValid()`: Verifies the integrity of the blockchain.

BlockchainNotary Class - *Key Concepts*

- Genesis Block:
  - The first block, called the genesis block, initializes the blockchain. It has a hardcoded previous hash since there are no preceding blocks.

- Adding Blocks:
  - Blocks are added to the blockchain after being mined. Mining ensures the block meets the network's difficulty requirements, providing security through computational work.

- Transaction Inclusion:
  - Transactions are included in the blocks as string data. This data is then hashed to ensure the integrity and immutability of the block's contents.

- Chain Validation:
  - Ensuring the chain’s validity involves checking that each block’s hash matches its computed hash, the previous hash matches the previous block's hash, and that each block has been properly mined according to the difficulty level.

Results

```
Enter the number of transactions for the new block: 3
Enter ID for transaction 1: 123
Enter data for transaction 1: Transfer $100 to Alice
Enter ID for transaction 2: 456
Enter data for transaction 2: Transfer $50 to Bob
Enter ID for transaction 3: 780
Enter data for transaction 3: Transfer $150 to Carol
Do you want to add another block? (yes/no): no

--- Blockchain Structure ---
Block 1
Previous Hash: 0
Hash: 0b0add0366505d2ceb2165a9174ab57473a98a7bacb9b2b04e2633d9b144debe
Nonce: 3
Timestamp: 2024-04-29 00:23:15
Miner's Address: Genesis Miner
Transactions:

Block 2
Previous Hash: 0b0add0366505d2ceb2165a9174ab57473a98a7bacb9b2b04e2633d9b144debe
Hash: 000d453b6dd5a16dd400f4766b2aeba815fcf9439c35a09bf99ca2a388bdbcbf
Nonce: 2298
Timestamp: 2024-04-29 00:33:57
Miner's Address: Miner-1
```

Results

```
Transactions:
   Coinbase: Miner Miner-1, Reward: 50, Timestamp: 2024-04-29 00:27:12
 - ID: 123, Data: Transfer $100 to Alice, Timestamp: 2024-04-29 00:28:07, Signature: Fy6bg2+0rSdMpZawWNEF
                  G081oRPCihE7F0jzTqv/71JwoPiEnKDIgknsWovHLERg2xlRcFg5EJoPrCjymD0sNPsAfjMxJI+Q3JUtyPPfyLT
                  H03poS9JutIuO9gOyKOC56vHn/UllJBFIDwT5FggL/gp0AM8ESDERNIQYCYlcFN/twGbN2g1ShRioVBpOGf5S/5
                  alTQMxiCHm92hHQmsaPrSGL5yY3G0RarMhF5FLjOWxm3oxXWGlBjVlclmqd5i/aPz9vO5fiFDB6X5JM5sGNGopw
                  Tqv9+lCII7QoLD7OMONLSBgSCcrjgkMO/skI1rB6WBf+XcNxVS1di3C/1Vf7w==
 - ID: 456, Data: Transfer $50 to Bob, Timestamp: 2024-04-29 00:33:42, Signature: omADvDr2D4PEjsLrHLfsKZEV
                  zH9i2eqWzLPZbkYSE9MPzNyvHNQ7tWWcHI4LbhOCgwm15L4ib0VoWDj702SuAvdlQENFxUqjCQyw3xBapClOGkJJ
                  sv0PRpCGxaQyZXqPbFl+diQ2THRkjbhzVmKfP0EQ1mFG31KGQx/6cLoL2rvtQQP8jjCg3IOj21MSWI0A+ztwuOeb
                  Ox7qNhfRC734MRpP7RAmjqvRMkVzTWDTnBHqpHXcv6NiPiSlx/T6t499YYzEy85fSzBJzM5j3QFzlgU9+/qTCYhP
                  ryHVw0ZlV15MA7J1bMZqgKOCsvuVpe39RybHfEdAJ42DmpGPMDK2rg==
 - ID: 780, Data: Transfer $150 to Carol, Timestamp: 2024-04-29 00:33:56, Signature: V016L16CUAoBGZp6qax3y
                  qDprd6KS7ByU7cjmy4Cs/6D9K6VMQbMQEFQtULkAz6PEF9UbMLWGlCVx+5U8Hopln9SjK7d4sOrA8cVA6PYAim+P
                  vNxSjRWlvfMUr12f8tEdWZ0DpFAE9gpeJpyIOWqB++GLa123N2+D0thvxUNLf1T3YQ+QBlKWTeYu3YP9CRfBPleI
                  QgdaHLTEz/H62YRfBev6j/JF2XrwkWJAjB+VVzQ+C7sLbiiTtTgmxTDOaVX9VsUHspl7j8y7S6GzZc8RdKoJPmnA
                  3T6GXEWKWcYN9FmkipaJYHTIAz0tgboldCuzz3Mc3wHEM7Gah+180mc2w==

Is the blockchain valid? true
Enter the number of transactions for the new block: 1
Enter ID for transaction 1: 99
Enter data for transaction 1: Transfer $70 to Cam
Do you want to add another block? (yes/no): no
```

Advantages of the Proposed System

Increased Security

The blockchain's decentralized and cryptographic nature enhances data security by making it virtually impossible to tamper with or hack the system.

Faster Transactions

Blockchain technology enables faster and more efficient transactions by eliminating intermediaries and automating processes through smart contracts.

Improved Transparency

The distributed ledger of the blockchain provides transparency, as all transactions are recorded and visible to network participants.

Conclusion and Next Steps

This project has provided a solid understanding of blockchain development and key concepts like consensus mechanisms, immutability, and decentralization by building a basic blockchain using Java.

Future improvements can focus on adding smart contracts, enhancing privacy, and improving scalability. Exploring interoperability between different blockchain networks will also increase the project's usefulness.

Overall, this project has been a valuable learning experience, equipping us with the skills to navigate the evolving blockchain landscape and contribute to developing decentralized systems.