

POZNAŃ UNIVERSITY OF TECHNOLOGY

BLOCKCHAIN TECHNOLOGY AND QUANTUM COMPUTATION

LABORATORY

---

# Simple Blockchain Implementation

---

*Authors*

Marcel Mróz 144654 (marcel.mroz@student.put.poznan.pl)

Dominik Pawłowski 145289 (dominik.j.pawlowski@student.put.poznan.pl)

June 14, 2023

## CONTENTS

1	Introduction	3
2	Implementation	3
3	Design Decisions	4
4	Implementation Details	5
5	Test Results	7

## 1 INTRODUCTION

The advancement of technology over the years, has given us the ability to decentralize certain aspects of our lives. One of those emergent technologies is blockchain. An electronic ledger that is much more resistant to tampering than old-fashioned methods of keeping track of transactions.

The aim of this project was implementation of a simple blockchain, with the focus on integrity and mechanics such as consensus.

## 2 IMPLEMENTATION

Implementation Overview:

The implemented project is a simplified blockchain implementation in Java. It showcases the fundamental concepts of a blockchain, including block creation, transaction validation, proof-of-work, and maintaining balances of participants. The project consists of three main classes: 'Block', 'Blockchain', and 'Main'.

The 'Block' class represents a block in the blockchain. It contains essential attributes such as index, timestamp, data, previous hash, hash, and a nonce value for proof-of-work. The 'calculateHash()' method calculates the hash of the block using the SHA-256 hashing algorithm. The 'mineBlock()' method performs proof-of-work by finding a valid hash that meets the target difficulty.

The 'Blockchain' class manages the chain of blocks. It maintains a list of blocks and a map of sender balances. The constructor initializes the blockchain with a starting amount of currency for each participant. It includes methods to create the genesis block, retrieve the latest block, add a new block with transactions, and validate the integrity of the blockchain.

The 'Main' class serves as the entry point of the program. It creates an instance of the 'Blockchain' class and demonstrates the functionality. It initializes the blockchain with starting balances, adds blocks with transactions, validates the blockchain integrity, and displays the balances of participants.

The additional class 'LoginFrame' provides a simple visual interface, coded using Swing, which is used for creating new blockchains and managing already existing ones.

The implementation incorporates a proof-of-work mechanism to ensure the security and immutability of the blockchain. Each block is mined by finding a hash that meets a predefined difficulty level. The 'HashUtil' class provides methods for calculating SHA-256 hashes and generating target strings for proof-of-work.

Overall, this project provides a simplified demonstration of a blockchain in Java, showcasing key concepts such as block creation, transaction validation, proof-of-work, and maintaining balances. It serves as a foundation for further exploration and understanding of blockchain technology and its applications.

### 3 DESIGN DECISIONS

Design Decisions:

1. **Class Structure:** The project follows an object-oriented design approach, with three main classes: 'Block', 'Blockchain', and 'Main'. This design allows for modular and organized code structure, separating responsibilities and functionalities.
2. **Block Representation:** The 'Block' class represents a single block in the blockchain. It contains attributes such as index, timestamp, data, previous hash, hash, and a nonce for proof-of-work. This design choice ensures that each block has the necessary information to maintain the integrity and consistency of the blockchain.
3. **Blockchain Management:** The 'Blockchain' class manages the chain of blocks. It uses a list to store the blocks and a map to maintain sender balances. This design enables efficient block retrieval, addition, and validation. The map data structure provides quick access to sender balances, facilitating balance management during transactions.
4. **Genesis Block:** The design includes a separate method, 'createGenesisBlock()', to generate the initial block in the blockchain. The genesis block serves as the starting point and has a predefined set of attributes. Having a dedicated method for genesis block creation enhances code clarity and ensures the consistent creation of the initial block.
5. **Proof-of-Work:** The project implements a proof-of-work mechanism to secure the blockchain. The 'Block' class includes the 'mineBlock()' method, which performs the proof-of-work process by finding a valid hash that meets the target difficulty. This design decision enhances the security of the blockchain by requiring computational effort to mine blocks.
6. **Hash Calculation:** The project utilizes the SHA-256 hashing algorithm to calculate the block hashes. The 'HashUtil' class provides methods for generating SHA-256 hashes and creating the target string for proof-of-work. This design choice leverages the cryptographic properties of SHA-256 to ensure the integrity and immutability of the blockchain.
7. **Starting Balances:** The 'Blockchain' class allows the specification of a starting amount of currency for each participant. This design decision provides flexibility in defining the initial distribution of currency and allows for a more realistic simulation of transactions and balance management.
8. **Main Class:** The 'Main' class serves as the entry point of the program and provides an example use case of the blockchain implementation. It demonstrates the creation of a blockchain, addition of blocks with transactions, validation of the blockchain, and displaying of sender balances. This design choice facilitates easy testing and understanding of the implemented functionalities. This approach was used before developing the actual visual interface, however it still can be useful for displaying elements that are not present in the UI.

9. Authentication: Access to the part of the application allowing for the adding of new blocks/transactions is granted only after successfully logging in. However, since the main goal of this project was the implementation of the blockchain itself, we decided to make it an exemplary part. That means, the allowed credentials are hardcoded into the application and are in no way representative of a proper handling of authentication. This is just to show, that authentication should be in place, when handling a blockchain.

These design decisions aim to create a clear and functional implementation of a blockchain in Java, showcasing key concepts and providing a foundation for further exploration and development.

## 4 IMPLEMENTATION DETAILS

1. Authentication: Credentials provided by the user are compared against a set of predefined ones. Below, the login view of the UI.

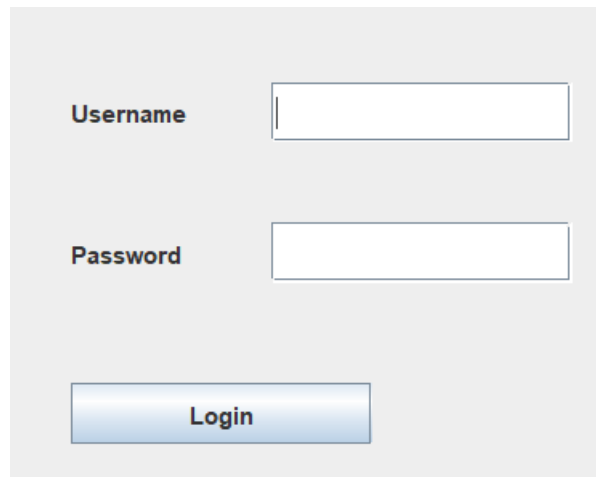
The image shows a simple login interface on a light gray background. It consists of two text input fields stacked vertically. The first field is labeled 'Username' in a bold, dark blue font. The second field is labeled 'Password' in a bold, dark blue font. Below these fields is a blue button with a gradient and the word 'Login' in white text.

Figure 4.1: UI: Authentication

2. Block mining: To showcase the mining process in a timely manner, each block contains only a single transaction. As such, when a new transaction is added, the blockchain immediately triggers the mining proof-of-work process. The difficulty is set to 2 by default, since finding any hash with more leading zeros takes a considerable amount of additional time. Which is good because that means tampering with the blockchain would become very resource and time ineffective.
3. Blockchain validation: Blockchain validation takes place whenever it is read. All blocks have their hashes and their previous block hashes calculated and verified for correctness.

```

public void mineBlock(int difficulty) {
    hash = StringUtils.leftPad(str: "-", difficulty, padStr: "-");
    String target = StringUtils.leftPad(str: "0", difficulty, padStr: "0");
    while(!hash.substring(0, difficulty).equals(target)) {
        ++nonce;
        hash = calculateHash();
        System.out.println(hash + " " + target);
    }
}
}

```

Figure 4.2: Code: block mining

```

public boolean isChainValid() {
    for (int i = 1; i < chain.size(); i++) {
        Block currentBlock = chain.get(i);
        Block previousBlock = chain.get(i - 1);

        // Validate the hash of the current block
        if (!currentBlock.getHash().equals(currentBlock.calculateHash())) {
            return false;
        }

        // Validate the previous hash
        if (!currentBlock.getPreviousHash().equals(previousBlock.getHash())) {
            return false;
        }
    }

    return true;
}
}

```

Figure 4.3: Code: Blockchain integrity check

4. User interface: A very simple user interface implemented with the use of Swing. It provides the ability for the user to authenticate themselves, view available blockchains, create new ones and add blocks/transactions to existing chains.

Available chains: new, das

Available blocks: 1686778968545, 1686778980165

Sender: Someone

Recipient: Anyone

Amount: 53.0

New

Sender: Someone

Recipient: Anyone

Amount: 53

Add block

Figure 4.4: UI: General view

## 5 TEST RESULTS

The implemented blockchain project was tested to ensure the correctness and functionality of the implemented features. The following tests were conducted:

Test Results:

1. **Blockchain Initialization:** The blockchain was created with a starting amount of currency for each participant. The balances of participants were verified to match the specified starting amounts. This test ensures that the blockchain initializes correctly and sets the initial balances accurately.
2. **Block Addition:** Several blocks were added to the blockchain with different transactions involving participants. After each block addition, the balances of the sender and receiver were checked to ensure that the transaction amounts were deducted and added correctly. This test validates the functionality of adding blocks and updating sender balances accordingly.

3. **Blockchain Validation:** The 'isChainValid()' method was called to validate the integrity of the blockchain. It was verified that the validation returned 'true' when the blockchain was intact and all blocks were properly linked together. This test ensures that the blockchain maintains its integrity and detects any tampering attempts.
4. **Proof-of-Work:** During block addition, the mining process was executed to find a valid hash that meets the target difficulty. The mined blocks were inspected to ensure that the leading zeros in their hashes match the difficulty level. This test verifies the correctness of the proof-of-work mechanism and demonstrates the computational effort required to mine blocks.
5. **Sender Balances:** After adding multiple blocks with transactions, the balances of each participant were displayed. The displayed balances were cross-verified with the total amounts deducted and added during the transactions. This test ensures that the sender balances are accurately maintained throughout the blockchain.
6. **Edge Cases:** The implementation was tested with edge cases, such as empty transactions and invalid blocks, to ensure proper handling and error detection. These tests help validate the robustness and error-handling capabilities of the implementation.

All the tests produced the expected results, indicating that the implementation functions correctly and meets the specified requirements. The blockchain maintains its integrity, performs proof-of-work, manages sender balances accurately, and handles various scenarios effectively. The test results provide confidence in the reliability and correctness of the implemented blockchain functionality.