BANK OF CSJ

# Blockchain Development

## Writing a Simple Smart Contract

A simple smart contract representing a mini bank involving transfer of Ethers, programmed using Solidity and online Remix IDE, based on the Ethereum blockchain platform.

CHINMAY JOSHI

Cryptocurrency is a digital payment system that doesn't rely on banks to verify transactions. It's a peer-to-peer system that can enable anyone anywhere to send and receive payments. Instead of being physical money that is carried around and exchanged in the real world, cryptocurrency payments exist purely as digital entries to an online database that describe specific transactions. When you transfer cryptocurrency funds, the transactions are recorded in a public ledger. You store your cryptocurrency in a digital wallet.

Cryptocurrency got its name because it uses encryption to verify transactions. This means advanced coding is involved in storing and transmitting cryptocurrency data between wallets and to public ledgers. The aim of the encryption is to provide security and safety.

Cryptocurrencies are usually built using blockchain technology. Blockchain describes the way transactions are recorded into "blocks" and time stamped. It's a fairly complex, technical process, but the result is a digital ledger of cryptocurrency transactions that's hard for hackers to tamper with.

In addition, transactions require a two-factor authentication process. For instance, you might be asked to enter a username and password to start. Then, you might have to enter an authentication code that's sent via text to your personal cell phone. While securities are in place, that doesn't mean cryptocurrencies are un-hackable.

# Abstract

I have developed a smart contract for basic banking operations and made sure that, as much as I could, it included all of the functionalities and capabilities that Solidity has. Also, I can show how to send dummy ETH between any account and the contract I developed, and how to restrict the people who can use the relevant function of the smart contract.

A little bit of gist has been given, regarding Ethereum and Solidity, and then I have developed a smart contract that includes all of the functionalities. There are several beautiful methods to developing a smart contract. Of these methods, I have used Remix IDE, a powerful open-source tool that provides the ability to develop a smart contract from a browser.

A blockchain is a growing list of records, called blocks, that are linked together using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree). The timestamp proves that the transaction data existed when the block was published in order to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks. Although blockchain records are not unalterable as forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

The blockchain was invented by a person (or group of people) using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin. The identity of Satoshi Nakamoto remains unknown to date. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server. The bitcoin design has inspired other applications and blockchains that are readable by the public and are widely used by cryptocurrencies.

The blockchain is considered a type of payment rail. Private blockchains have been proposed for business use but Computerworld called the marketing of such privatized blockchains without a proper security model "snake oil". However, others have argued that permissioned blockchains, if carefully designed, may be more decentralized and therefore more secure in practice than permissionless ones.

Ethereum is a decentralized, open-source blockchain with smart contract functionality. Ether (ETH or Ξ) is the native cryptocurrency of the platform. After Bitcoin, it is the largest cryptocurrency by market capitalization. Ethereum is the most actively used blockchain.

Ethereum was proposed in 2013 by programmer Vitalik Buterin. In 2014, development was crowdfunded, and the network went live on 30 July 2015. The platform allows developers to deploy permanent and immutable decentralized applications onto it, with which users can interact.

Decentralized finance (DeFi) applications provide a broad array of financial services without the need for typical financial intermediaries like brokerages, exchanges, or banks, such as allowing cryptocurrency users to borrow against their holdings or lend them out for interest.

Ethereum also allows for the creation and exchange of NFTs, which are non-interchangeable tokens connected to digital works of art or other real-world items and sold as unique digital property. Additionally, many other cryptocurrencies operate as ERC-20 tokens on top of the Ethereum blockchain and have utilized the platform for initial coin offerings.

Ethereum has started implementing a series of upgrades called Ethereum 2.0, which includes a transition to proof of stake and aims to increase transaction throughput using sharding.

# Smart Contracts

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

Smart contracts work by following simple "if/when…then…" statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the "if/when...then…" rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

Then the smart contract can be programmed by a developer – although increasingly, organizations that use blockchain for business provide templates, web interfaces, and other online tools to simplify structuring smart contracts.

First I declared a contract beside the Solidity version I wanted to use. I have created a user object to keep the user's information, which will join the contract by using the struct element. It stores the user's ID, address, and balance in the contract. Then I created an array in the user_account type to keep the information of all of the users.

I am supposed to assign an ID to each user whenever they join the contract, so we define an int counter and set it to 0 in the constructor of the contract. Also, we need to define an address variable for the manager and a mapping to keep the last interest date of each client. Since we want to restrict the time required to send interest again to any account, it'll be used to check whether enough time has elapsed.

```
pragma solidity ^0.6.6;

contract BankOfCSJ
{
    struct userAccount
    {
        int userID;
        address userAddress;
        uint userBalanceAmountInEthers;
    }

    userAccount[] users;

    int numberOfUsers;
    address payable manager;
    mapping(address => uint) public dateOfInterest;

```

In a smart contract, I may need to restrict the people that can call the relevant method or I may want to allow the execution of the method only specific circumstances. In these kinds of circumstances, the modifier checks the condition I've implemented, and it determines whether the relevant method should be executed. I have to implement two modifiers. Both methods will check the people who call the relevant method and which of the modifiers is used. One of them determines whether the sender is the manager, and the other one determines whether the sender is a user.

The receive function is needed to make the contract receive ether from any address. The receive keyword is new in Solidity 0.6.x, and it's used as a fallback function to receive ether. Since I'll receive ether from the users as a deposit, I need to implement the receive function.

```
    modifier onlyManager()
    {
        require(msg.sender == manager, "The calling permission is only to the manager.");
        _;
    }

    modifier onlyUsers()
    {
        bool isUser = false;
        for(uint i = 0; i < users.length; i++)
        {
            if(users[i].userAddress == msg.sender)
            {
                isUser = true;
                break;
            }
        }
        require(isUser, "The permission to call the function is only to the users.");
        _;
    }

    constructor() public
    {
        numberOfUsers = 0;
    }

    receive() external payable
    {

    }

    function setManager(address managerAddress) public returns(string memory)
    {
        manager = payable(managerAddress);
        return "";
    }
```

After declaring the contract, defining the variables, and implementing the modifiers and receive function, I start developing the other important methods now.
In this contract, I'll develop the following methods:

- The setManager method will be used to set the manager address to variables we've defined. The managerAddress is consumed as a parameter and cast as payable to provide sending ether.
- The joinAsUser method will be used to make sure the user joins the contract. Whenever a user joins the contact, their interest date will be set, and the user information will be added to the user array.
- The deposit method will be used to send ETH from the user account to the contract. We want this method to be callable only by user who've joined the contract, so the onlyUser modifier is used for this restriction.

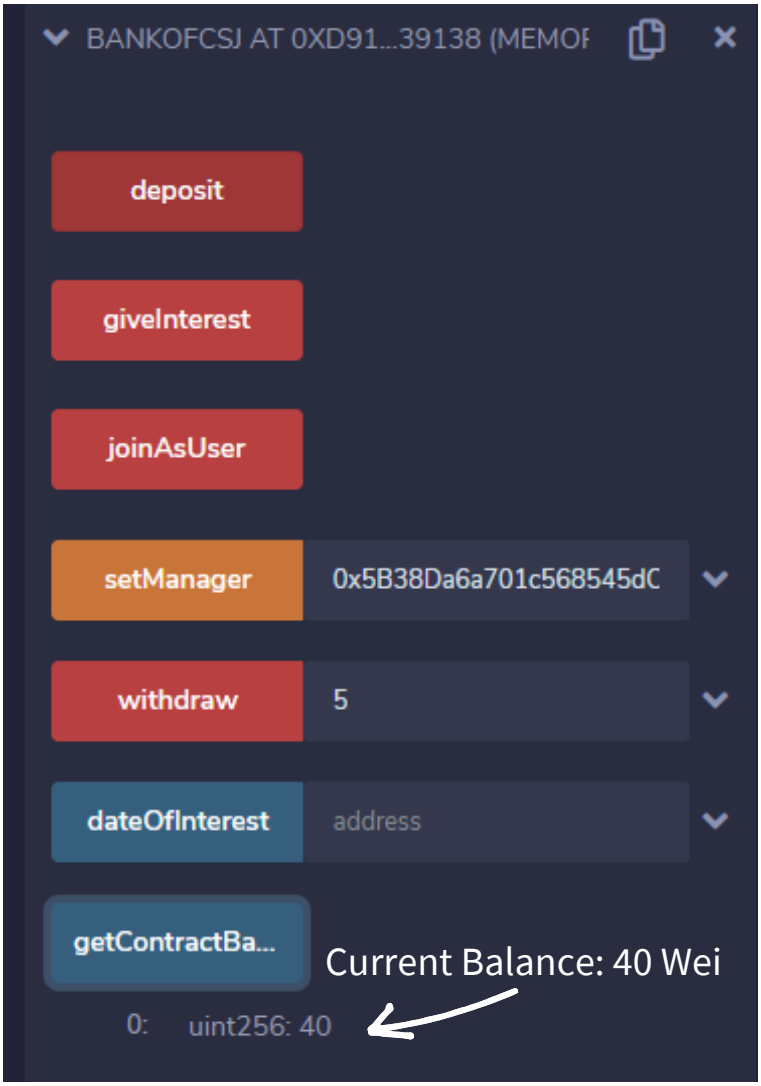
- The withdraw function will be used to send ETH from the contract to the user account. It sends the unit of ETH indicated in the amount parameter, from the contract to the user who sent the transaction. We want this method to be callable only by user who've joined the contract either, so the onlyUser modifier is used for this restriction.
- The giveInterest method will be used to send ETH as interest from the contract to all user. We want this method to be callable only by the manager, so the onlyManager modifier is used for this restriction. Here, the last date when the relevant user takes the interest will be checked for all of the users, and the interest will be sent if the specific time period has elapsed. Finally, the new interest date is reset for the relevant user into the dateOfInterest array if the new interest is sent.
- The getContractBalance method will be used to get the balance of the contract we deployed.

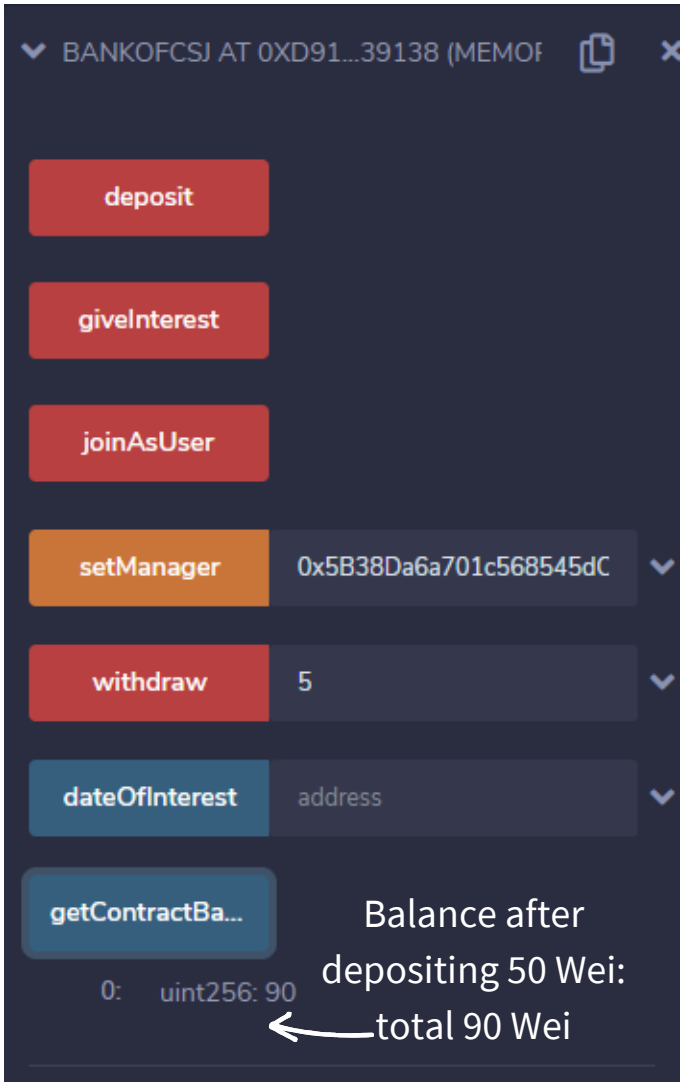
```
function joinAsUser() public payable returns(string memory)
{
    dateOfInterest[msg.sender] = now;
    users.push(userAccount(numberOfUsers++, msg.sender, address(msg.sender).balance));
    return "";
}

function deposit() public payable onlyUsers
{
    payable(address(this)).transfer(msg.value);
}

function withdraw(uint amount) public payable onlyUsers
{
    msg.sender.transfer(amount * 1 ether);
}

function giveInterest() public payable onlyManager
{
    for(uint i = 0; i < users.length; i++)
    {
        address previousAddress = users[i].userAddress;
        uint lastDateOfInterest = dateOfInterest[previousAddress];
        if(now < lastDateOfInterest + 10 seconds)
        {
            revert("It's just been less than 10 seconds!");
        }
        payable(previousAddress).transfer(1 ether);
        dateOfInterest[previousAddress] = now;
    }
}
```

```
pragma solidity ^0.6.6;

contract BankOfCSJ
{
    struct userAccount
    {
        int userID;
        address userAddress;
        uint userBalanceAmountInEthers;
    }

    userAccount[] users;

    int numberOfUsers;
    address payable manager;
    mapping(address => uint) public dateOfInterest;

    modifier onlyManager()
    {
        require(msg.sender == manager, "The calling permission is only to the manager.");
        _;
    }

    modifier onlyUsers()
    {
        bool isUser = false;
        for(uint i = 0; i < users.length; i++)
        {
            if(users[i].userAddress == msg.sender)
            {
                isUser = true;
                break;
            }
        }
        require(isUser, "The permission to call the function is only to the users.");
        _;
    }

    constructor() public
    {
        numberOfUsers = 0;
    }

    receive() external payable
    {

    }
```

```
receive() external payable
{

}

function setManager(address managerAddress) public returns(string memory)
{
    manager = payable(managerAddress);
    return "";
}

function joinAsUser() public payable returns(string memory)
{
    dateOfInterest[msg.sender] = now;
    users.push(userAccount(numberOfUsers++, msg.sender, address(msg.sender).balance));
    return "";
}

function deposit() public payable onlyUsers
{
    payable(address(this)).transfer(msg.value);
}

function withdraw(uint amount) public payable onlyUsers
{
    msg.sender.transfer(amount * 1 ether);
}

function giveInterest() public payable onlyManager
{
    for(uint i = 0; i < users.length; i++)
    {
        address previousAddress = users[i].userAddress;
        uint lastDateOfInterest = dateOfInterest[previousAddress];
        if(now < lastDateOfInterest + 10 seconds)
        {
            revert("It's just been less than 10 seconds!");
        }
        payable(previousAddress).transfer(1 ether);
        dateOfInterest[previousAddress] = now;
    }
}

function getContractBalance() public view returns(uint)
{
    return address(this).balance;
}
```

# Deployment Result and Running Transactions

# Conclusion

Thus in this report I have demonstrated how a simple banking code can be written in the Ethereum network using solidity language. Remix IDE is really good for this purpose and for testing out different smart contracts.

Smart contracts are immutable, that means when they are deployed, they can't be changed. A new contract with 0 value has to be created again and that will occupy more block space. Hence remix IDE is helpful for practicing smart contracts, before deploying the real ones on the Ethereum network and before playing with real ethers.

As for practical solutions of blockchain, there are plenty. If the agricultural and irrigation system of a farming town can be digitalized using blockchain software, and some of the local IT people of that town become miners, then our rain-fed farming systems can be completely revolutionized for example. Although for now, blockchain is mostly used by financial firms like JPMC to reduce settlement times when compliance queries are to be made in transactions.

**References:**


- Dapp University
- Wikipedia
- Solidity Language Documentation