

## Group C

### Assignment No: 3

**Title of the Assignment:** Write a smart contract on a test network, for Bank account of a customer for following operations: Deposit money Withdraw Money Show balance

**Objective of the Assignment:** Students should be able to learn about smart contract for banking application and able to perform some operation like Deposit money Withdraw Money Show balance

**Prerequisite:**

1. Basic knowledge of smart contract
2. Remix IDE
3. Basic knowledge of solidity

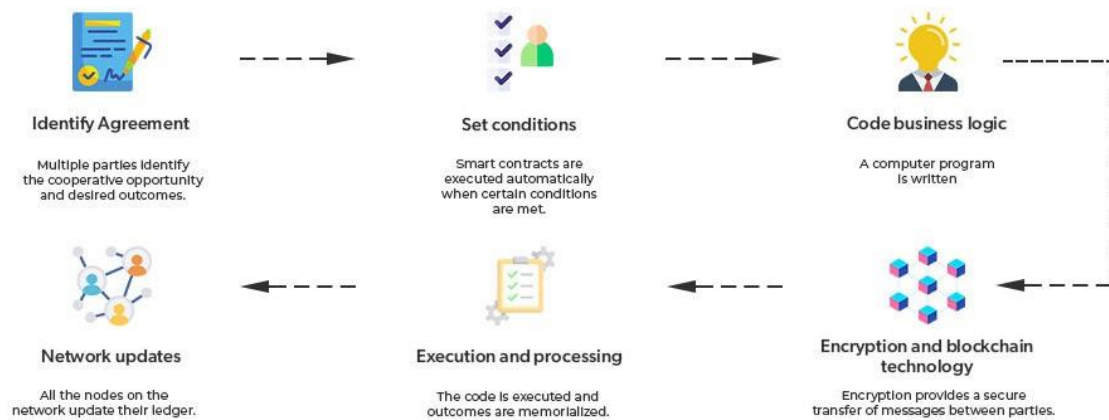
**Contents for Theory:**

1. Smart contract
2. Remix IDE
3. Solidity Basics
4. Ether transaction
- 5.

### Introduction to Smart Contract

- Smart contracts are self-executing lines of code with the terms of an agreement between buyer and seller automatically verified and executed via a computer network.
- Nick Szabo, an American computer scientist who invented a virtual currency called "Bit Gold" in 1998,1 defined smart contracts as computerized transaction protocols that execute terms of a contract.2
- Smart contracts deployed to blockchains render transactions traceable, transparent, and irreversible.

## How does a Smart Contract Work?

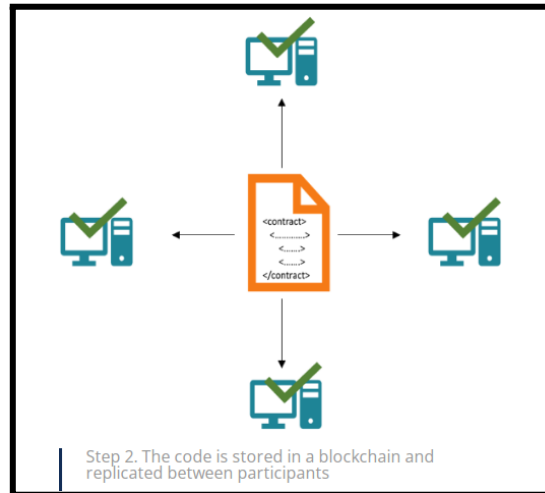


### How does Smart Contract work?

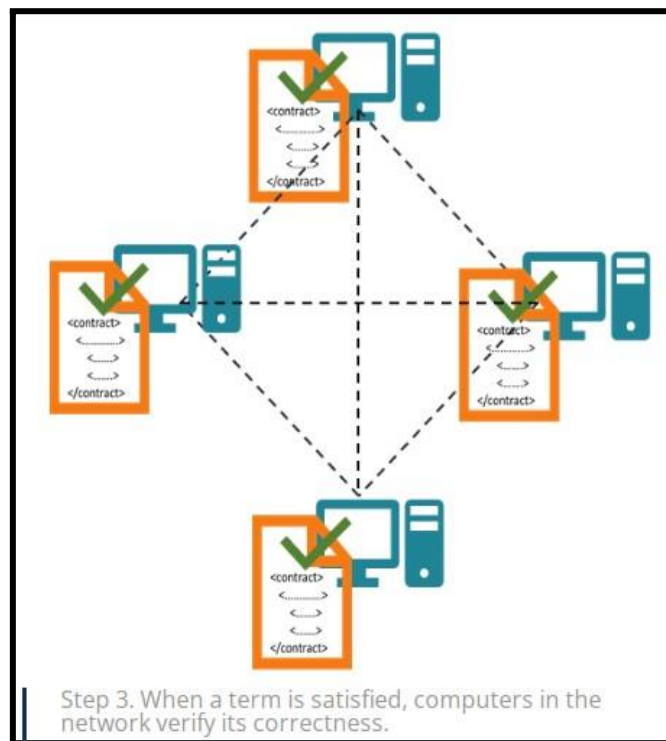
- First, the contractual parties should determine the terms of the contract. After the contractual terms are finalized, they are translated into programming code. Basically, the code represents a number of different conditional statements that describe the possible scenarios of a future transaction.



- When the code is created, it is stored in the blockchain network and is replicated among the participants in the blockchain.



- Then, the code is run and executed by all computers in the network. If a term of the contract is satisfied and it is verified by all participants of the blockchain network, then the relevant transaction is executed.

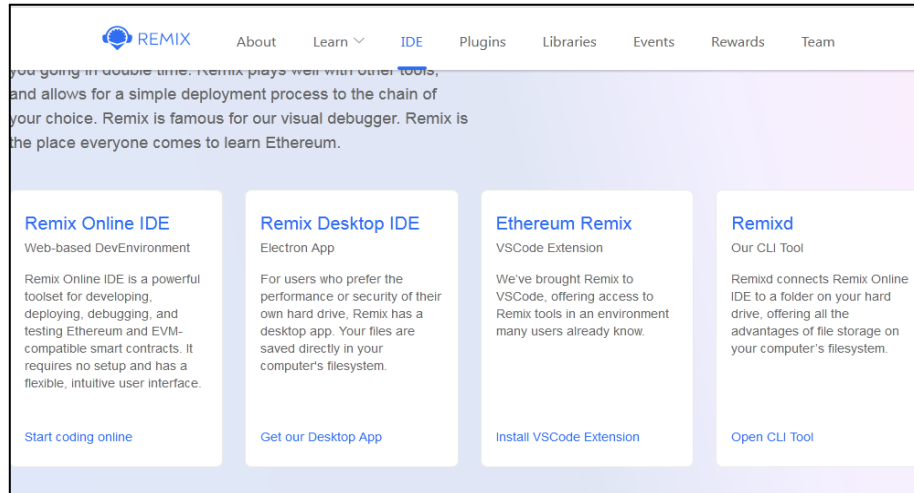


### Remix IDE : Installation steps

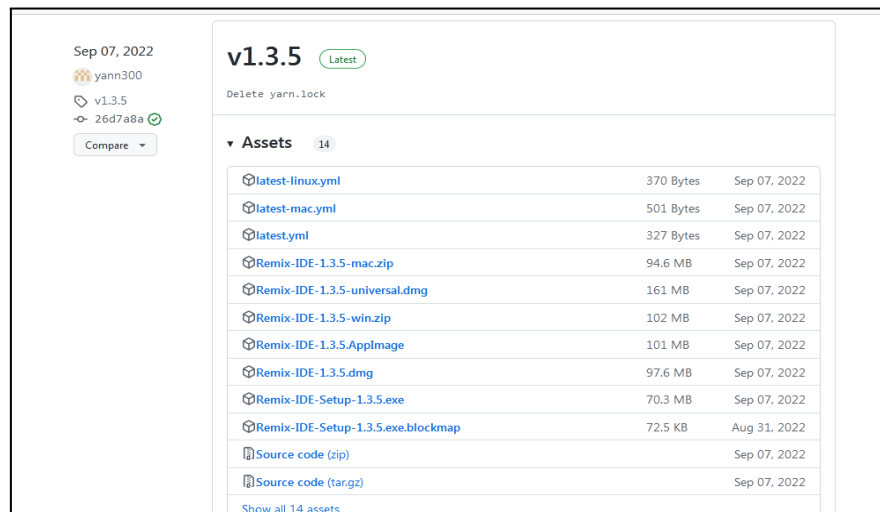
- First and foremost, head on over to the release page of the official Remix Desktop repository and grab the binary suited for your host system. This will be the .exe file for Windows users,

the .dmg for macOS and the .deb for Debian-derivative GNU/Linux systems. For Ubuntu and other AppImage setups, the .AppImage will be what you are looking for.

- Go to <https://remix-project.org/>
- Go to IDE option .Select Desktop IDE



- Select The file As per your choice



- Install Executable file,by double clicking on file(Applicable to .exe file Windows OS)

## What is Solidity?

- Solidity is a rather simple language deliberately created for a simplistic approach to tackle real world solutions. Gavin Wood initially proposed it in August of 2014. Several developers of the Ethereum chain such as Christian Reitwiessner, Alex Beregszaszi, Liana Husikyan,

Yoichi Hirai and many more contributed to creating the language. The Solidity language can be executed on the Ethereum platform, that is a primary Virtual Machine implementing the blockchain network to develop decentralized public ledgers to create smart contract systems.

### **Solidity Basics**

- To get started with the language and learn the basics let's dive into coding. We will begin by understanding the syntax and general data types, along with the variable data types. Solidity supports the generic value types, namely:
- Booleans: Returns value as either true or false. The logical operators returning Boolean data types are as follows:
- ! Logical negation
- && logical conjunction, "and"
- || logical disjunction, "or"
- == equality
- != inequality

**Integers:** Solidity supports int/unit for both signed and unsigned integers respectively. These storage allocations can be of various sizes. Keywords such as uint8 and uint256 can be used to allocate a storage size of 8 bits to 256 bits respectively. By default, the allocation is 256 bits. That is, uint and int can be used in place of uint256 and int256. The operators compatible with integer data types are:

- Comparisons: <=, <, ==, !=, >=, >. These are used to evaluate to bool.
- Bit operators: &, |, ^ bitwise exclusive 'or', ~ bitwise negation, "not".
- Arithmetic operators: +, -, unary -, unary +, \*, /, % remainder, \*\* exponentiation, << left shift, >> right shift.

The EVM returns a Runtime Exception when the modulus operator is applied to the zero of a "divide by zero" operation.

**Address:** An address can hold a 20 byte value that is equivalent to the size of an Ethereum address. These address types are backed up with members that serve as the contract base.

**String Literals:** String literals can be represented using either single or double quotes (for example, "foo" or 'bar'). Unlike in the C language, string literals in Solidity do imply trailing value zeroes. For instance, "bar" will represent a three byte element instead of four. Similarly, in the case of integer literals, the literals are convertible inherently using the corresponding fit, that is, byte or string.

**Modifier:** In a smart contract, modifiers are used to ensure the coherence of the conditions defined before executing the code.

Solidity provides basic arrays, enums, operators, and hash values to create a data structure known as “mappings.” These mappings are used to return values associated with a given storage location. An Array is a contiguous memory allocation of a size defined by the programmer where if the size is initialized as K, and the type of element is instantiated as T, the array can be written as T[k].

Arrays can also be dynamically instantiated using the notation `uint[][6]`. Here the notation initializes a dynamic array with six contiguous memory allocations. Similarly, a two dimensional array can be initialized as `arr[2][4]`, where the two indices point towards the dimensions of the matrix.

We will begin our programming venture with a simple structure of a contract. Consider the following code:

```
pragma solidity^0.4.0;
contract StorageBasic {
    uint storedValue;
    function set(uint var) {
        storedValue= var;
    }
    function get() constant returns (uint) {
        return storedValue;
    }
}
```

- The word “**Pragma**” refers to the instructions given to a compiler to sequentially execute the source code.
- Solidity is statically typed language. Therefore, each variable type irrespective of their scope can be instantiated at compile time. These elementary types can be further combined to create

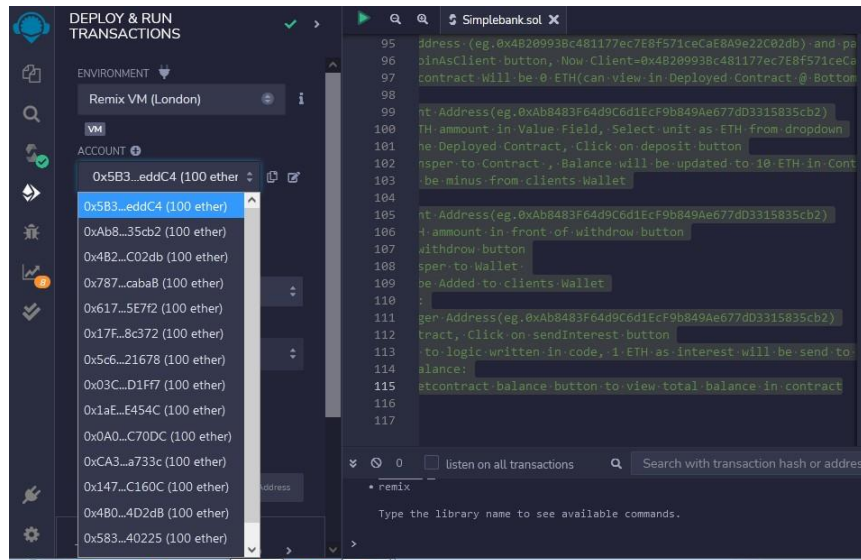
complex data types. These complex data types then synchronize with each other according to their respective preferences.

### Steps to Run Banking Application

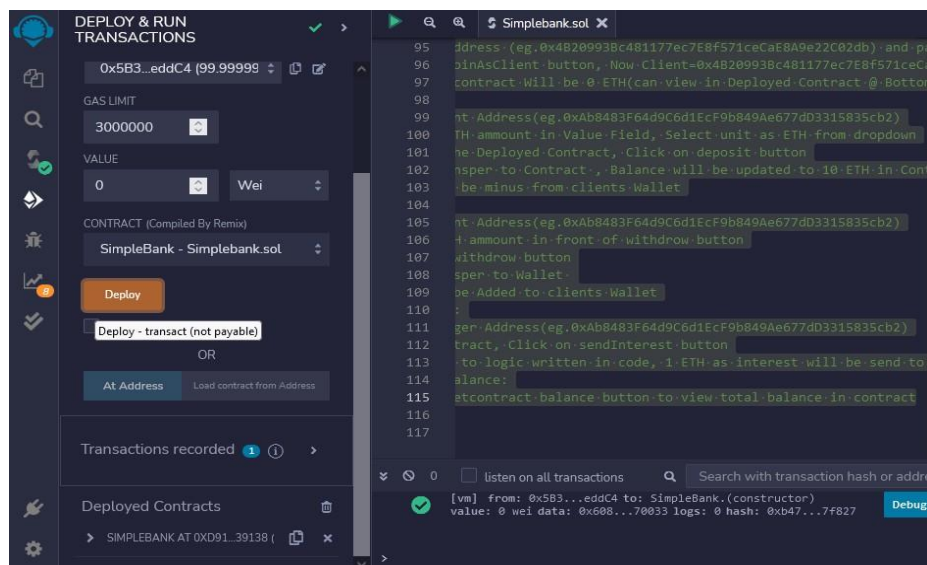
1. //Output steps

//Initially All Account has 100 fake ether

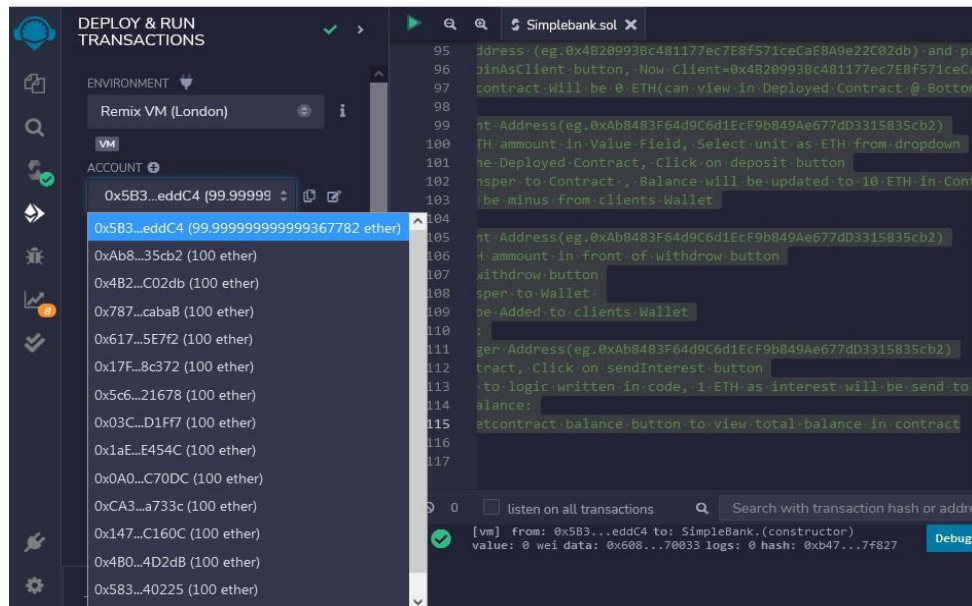
//Step 1: Select first Address (eg.0x5B38Da6a701c568545dCfcB03FcB875f56beddC4)



//Step 2: Click on Deploy button(Contract Created,Can view under Deployed Contract)



//After deploying contract 100 ETH turns to 99.99999.... ETH



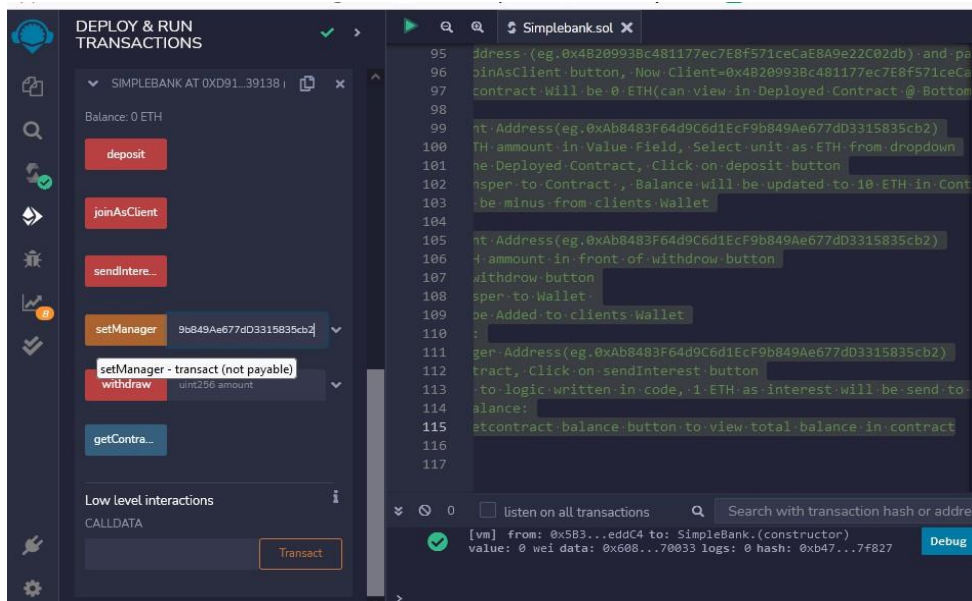
//Step 3: Set Manager: Follow Following instructions

// i. Select Another Address (eg. 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

// ii. Copy this address (eg. 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) and paste it in contract, in front of set Manager button

// iii. click on set manager button, Now

Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2



//Step 4: join as Client: Follow Following instructions

// i. Select Another Address (eg. 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)

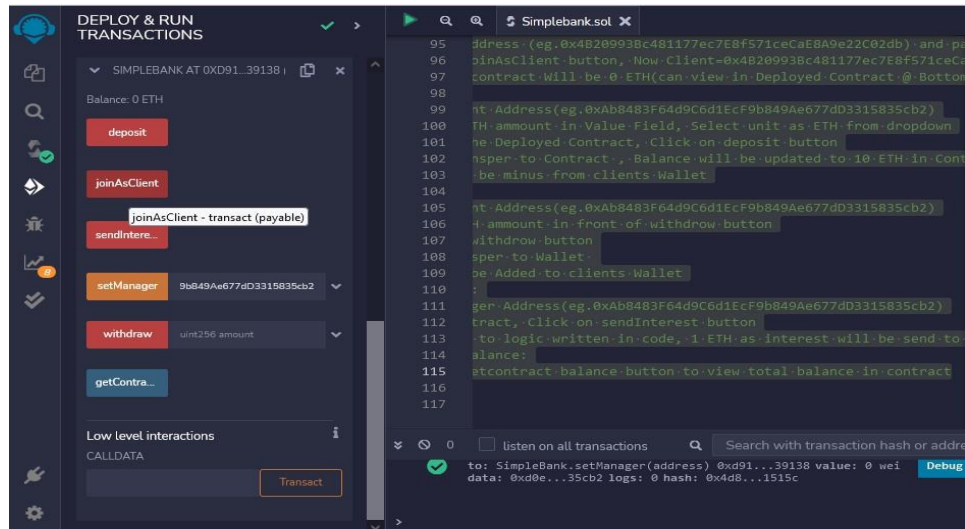
// ii. Copy this address (eg. 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) and paste it in



contract, in front of joinAsClient button

// iii.click on joinAsClient button, Now

Client=0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db



//Initially Balance in contract Will be 0 ETH(can view in Deployed Contract @ Bottom)

//Step 5: Deposit:

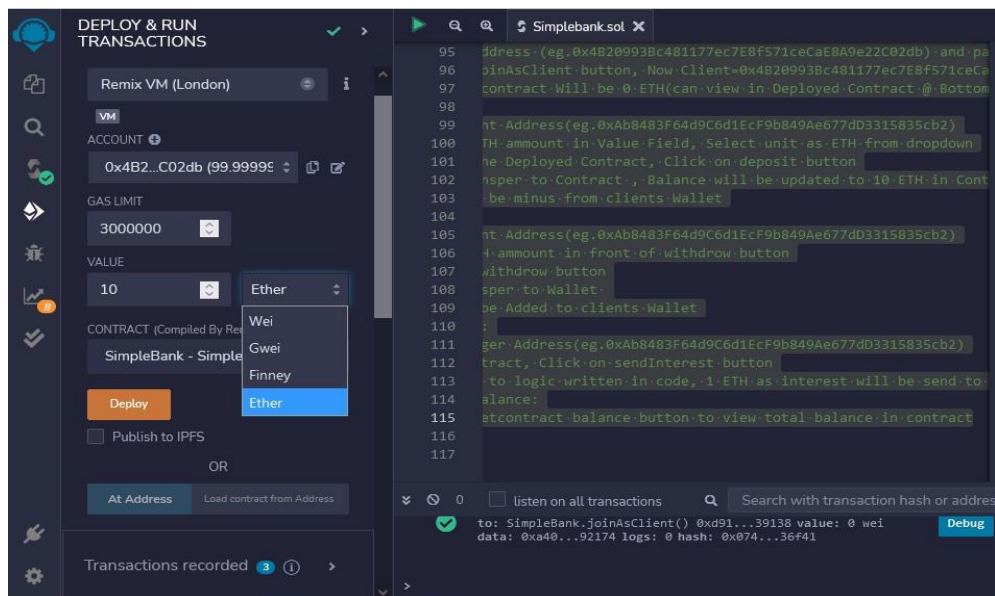
// i.Select Client Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)

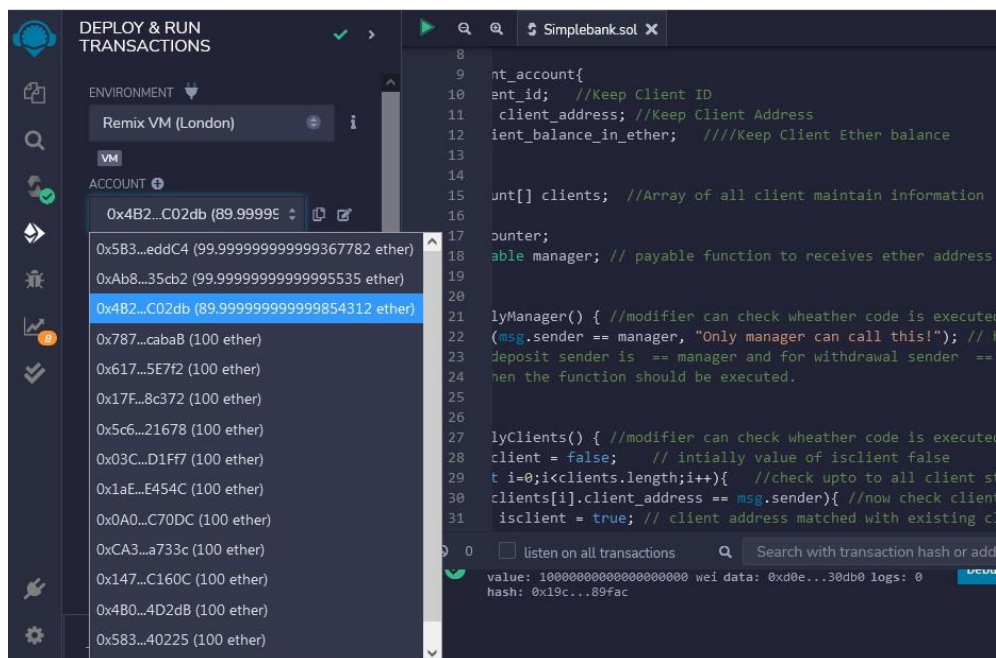
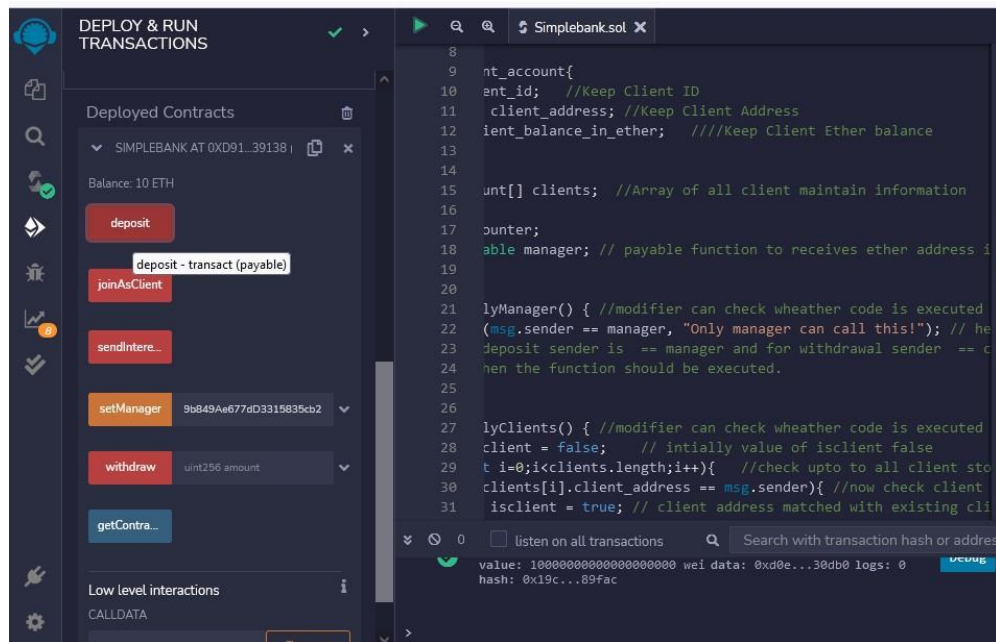
// ii. Enter 10 ETH ammount in Value Field, Select unit as ETH from dropdown

// iii. Come to the Deployed Contract, Click on deposit button

// iv. 10 ETH transper to Contract , Balance will be updated to 10 ETH in Contract

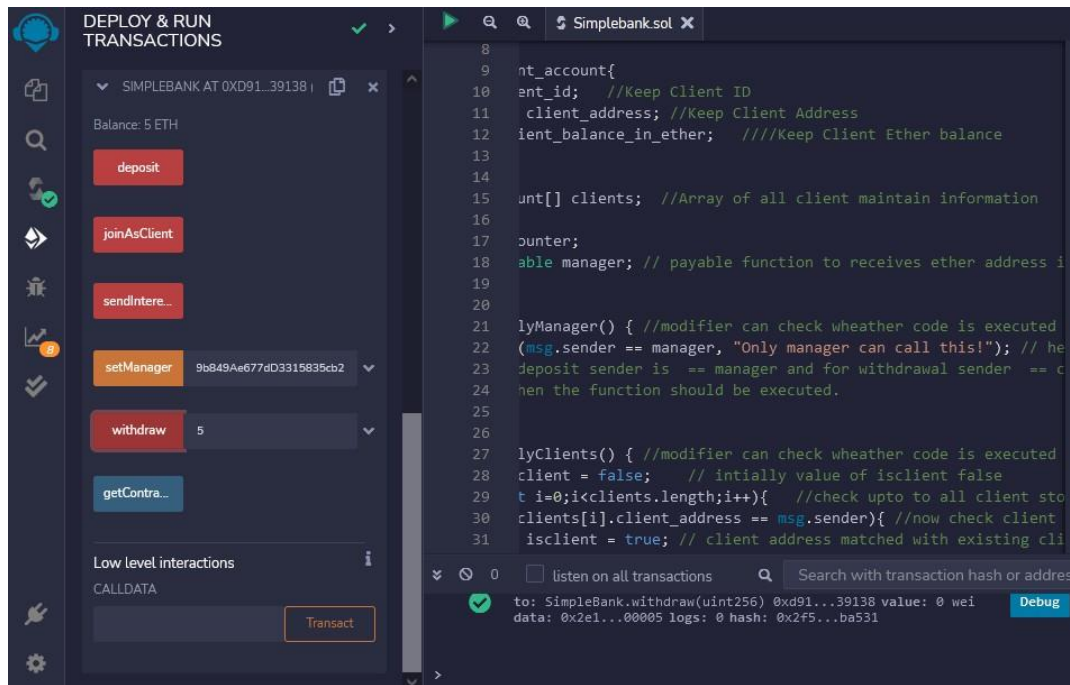
// V. 10 ETH will be minus from clients Wallet





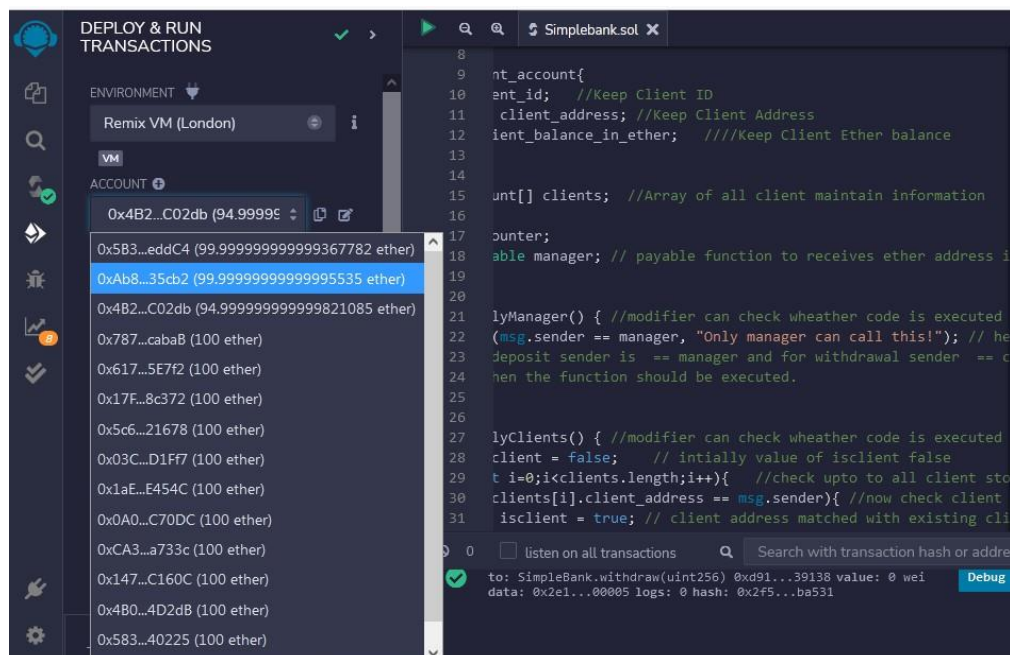
//Step 6: Withdraw:

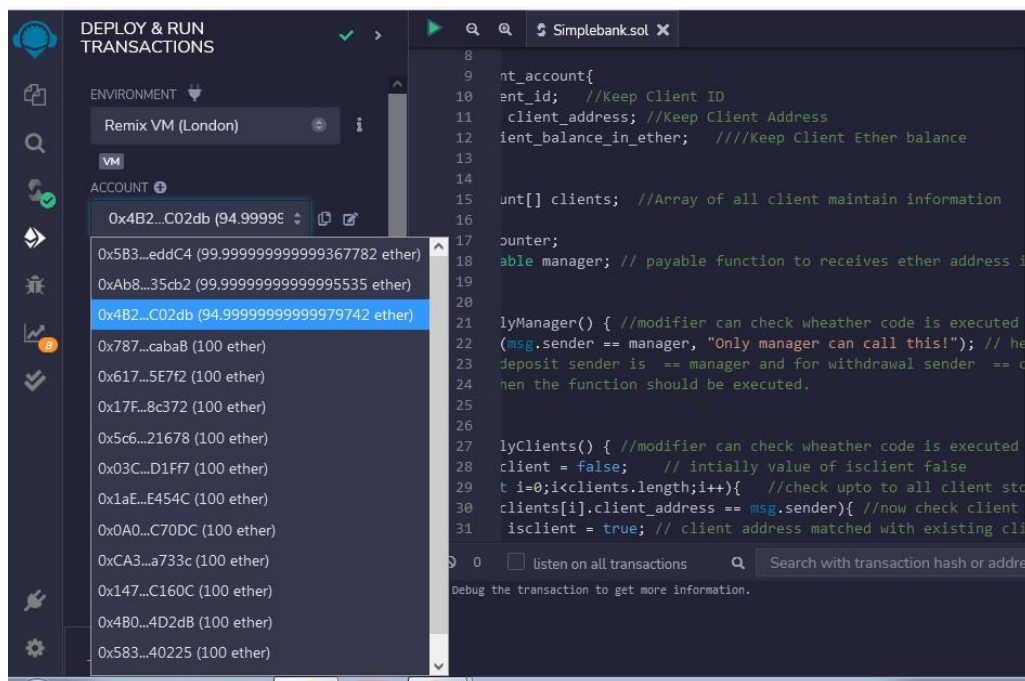
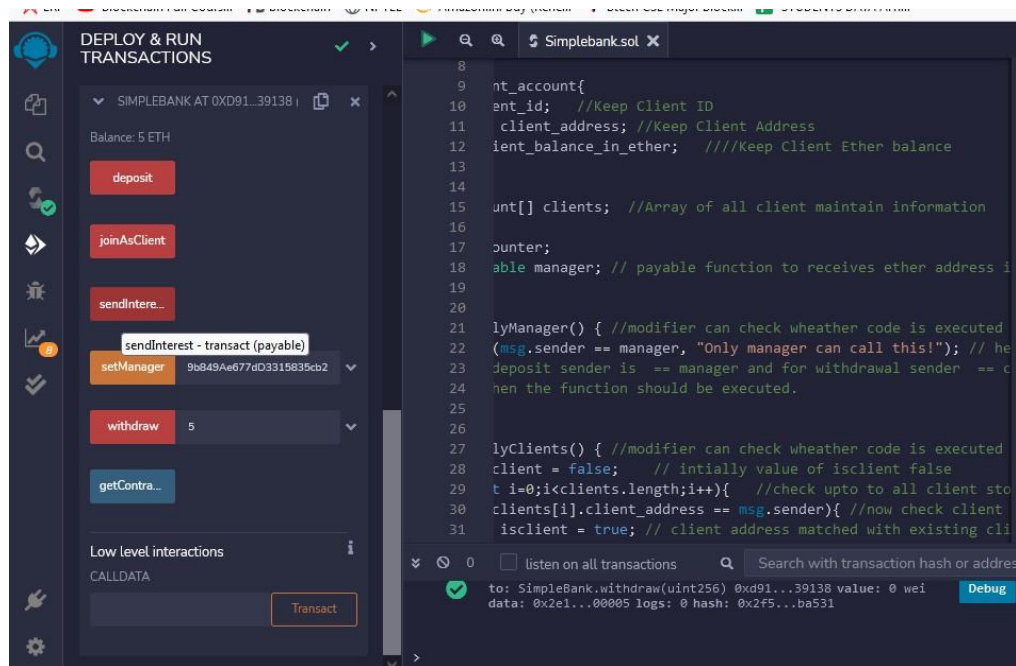
- // i. Select Client Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
- // ii. Enter 5 ETH ammount in front of withdraw button
- // iii. Click on withdraw button
- // iv. 5 ETH transper to Wallet
- // V. 5 ETH will be Added to clients Wallet



//Step 7: Send Interest:

- // i. Select Manager Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
- // ii. Come to contract, Click on sendInterest button
- // iii. According to logic written in code, 1 ETH as interest will be send to Client Wallet

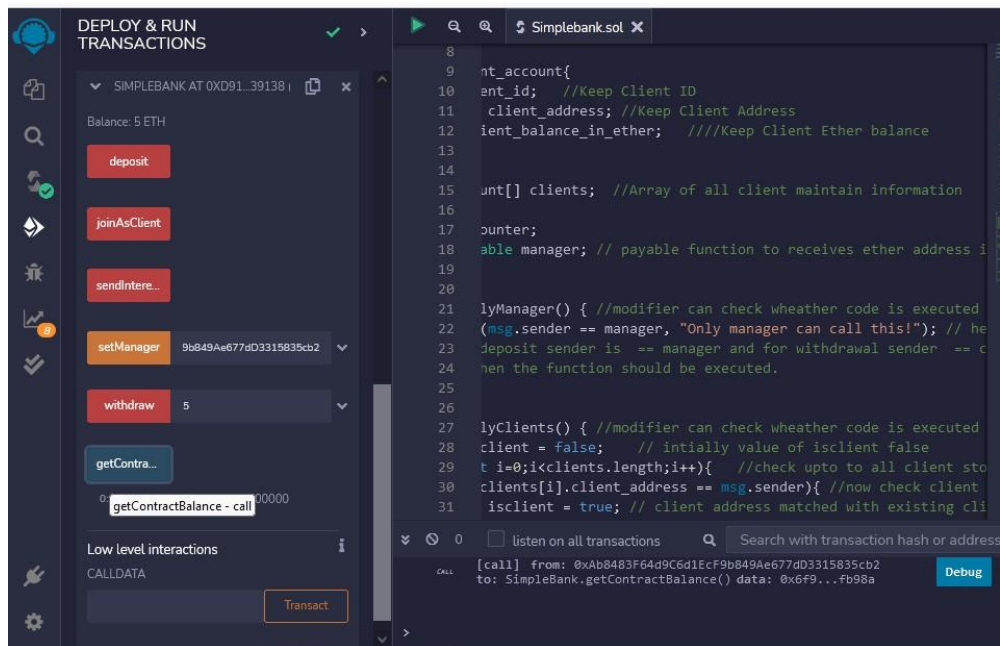




//Step 8: getContract Balance:

// i. Click on get contract balance button to view total balance in contract





**Conclusion-**In this way we have explored Concept u s i n g smart contract on a test network, and mapped with CO6