Grp C: Assignment No 3

Write a smart contract on a test network, for Bank account of a customer for following operations:

- Deposit money
- Withdraw Money
- Show balance

Code:

```
//SPDX-License-Identifier: MIT


//https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f


pragma solidity >=0.7.0 <0.9.0;


contract SimpleBank {

    struct client_account{
        int client_id;   //Keep Client ID
        address client_address; //Keep Client Address
        uint client_balance_in_ether;   ////Keep Client Ether balance
    }

    client_account[] clients;  //Array of all client maintain information

    int clientCounter;
    address payable manager; // payable function to receives ether address is datatype it is 20 byte hash address public key


    modifier onlyManager() { //modifier can check wheather code is executed accouding to condition for manager side
        require(msg.sender == manager, "Only manager can call this!");
// here sender is manager in this case
        // for deposit sender is  == manager and for withdrawal sender == client
        _; // when the function should be executed.
    }
```

```
    modifier onlyClients() { //modifier can check wheather code is
executed accouding to condition for client side
        bool isclient = false;    // intially value of isclient false
        for(uint i=0;i<clients.length;i++){   //check upto to all client
store in array
            if(clients[i].client_address == msg.sender){ //now check
client address matched with sender only that client intiate transaction
                isclient = true; // client address matched with existing
client address in bank database isclient value updated true.
                break;
            }
        }
        require(isclient, "Only clients can call this!"); // isclient
true here so allowed call the transaction.
        _; // when the function should be executed.
    }

    constructor() {
        clientCounter = 0;  // those client join contract assign there
ID intially it set 0
    }

    receive() external payable { } // this allows the smart contract to
receive ether

    function setManager(address managerAddress) public returns(string
memory){ //setManager method will be used to set the manager address to
variables
        // string memory store address of manager account instead of
store data
        manager = payable(managerAddress);// managerAddress is consumed
as a parameter and cast as payable to provide sending ether.
        return ""; // return payable address of manager
    }

    function joinAsClient() public payable returns(string memory){
//joinAsClient method will be used to make sure the client joins the
contract.

        clients.push(client_account(clientCounter++, msg.sender,
address(msg.sender).balance)); // push() array method to add items into
a storage array.
        return ""; // return all client details
    }
```

```
    function deposit() public payable onlyClients{  // deposit == client
to contract by onlyclient
        //deposit method will be used to send ETH from the client
account to the contract.
    // We want this method to be callable only by clients who've joined
the contract, so the onlyClient modifier is used for this restriction.
        payable(address(this)).transfer(msg.value); //transfer methods
belongs to the contract, and it's dedicated to sending an indicated
amount of ETH between addresses.
        // The payable keyword makes receipt of the ETH transfer
possible so the amount of ETH indicated in the msg.value will be
transferred to the contract address.
    }

    function withdraw(uint amount) public payable onlyClients{ //
withdraw == contract to client by onlyclient
        payable(msg.sender).transfer(amount * 1 ether); // The address
of the sender( ie contract ) is held in the msg.sender variable.
        //The withdraw method will be used to send ETH from the contract
to the client account. It sends the unit of ETH indicated in the amount
parameter, from the contract to the client who sent the transaction. We
want this method to be callable only by clients who've joined the
contract either,
        // so the onlyClient modifier is used for this restriction.
    }

    function sendInterest() public payable onlyManager{ //The
sendInterest method will be used to send ETH as interest from the
contract to all clients. can called by only manager
        for(uint i=0;i<clients.length;i++){ // check client in database
            address initialAddress = clients[i].client_address; // check
client address

            payable(initialAddress).transfer(1 ether);


        }
    }

    function getContractBalance() public view returns(uint){
//getContractBalance method will be used to get the balance of the
contract we deployed.
        return address(this).balance;
    }
}


//Output steps
```

```
//Initially All Acount 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 Onother
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii.Copy this address
(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) and paste it in
contract, infront of set Manager button
//       iii. click on set manager button, Now
Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
//Step 4: join as Client: Follow Following instructions
//        i.Select Onother
Address(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)
//       ii.Copy this address
(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) and paste it in
contract, infront of joinAsClient button
//       iii.click on joinAsClient button, Now
Client=0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
//Initially Balanve 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 withdrow button
//       iii. Click on withdrow 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 getcontract balance button to view total balance
in contract
```

//Output steps

//Initially All Acount 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 Onother Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
// ii.Copy this address (eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) and paste it in contract, infront of set Manager button
// iii. click on set manager button, Now Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
//Step 4: join as Client: Follow Following instructions
// i.Select Onother Address(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)
// ii.Copy this address (eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db) and paste it in contract, infront of joinAsClient button
// iii.click on joinAsClient button, Now Client=0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
//Initially Balanve 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 withdrow button

// iii. Click on withdrow 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 getcontract balance button to view total balance

Code in format

```
//SPDX-License-Identifier: MIT


//https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f


pragma solidity >=0.7.0 <0.9.0;


contract SimpleBank {

    struct client_account{
        int client_id;   //Keep Client ID
        address client_address; //Keep Client Address
        uint client_balance_in_ether;   ////Keep Client Ether balance
    }

    client_account[] clients;  //Array of all client maintain information

    int clientCounter;
    address payable manager; // payable function to receives ether address is datatype it is 20 byte hash address public key


    modifier onlyManager() { //modifier can check wheather code is executed accouding to condition for manager side
        require(msg.sender == manager, "Only manager can call this!"); // here sender is manager in this case
        // for deposit sender is  == manager and for withdrawal sender  == client
```

```
        _; // when the function should be executed.
    }

    modifier onlyClients() { //modifier can check wheather code is executed accouding to condition for client side
        bool isclient = false;    // intially value of isclient false
        for(uint i=0;i<clients.length;i++){   //check upto to all client store in array
            if(clients[i].client_address == msg.sender){ //now check client address matched with sender only that client intiate transaction
                isclient = true; // client address matched with existing client address in bank database isclient value updated true.
                break;
            }
        }
        require(isclient, "Only clients can call this!"); // isclient true here so allowed call the transaction.
        _; // when the function should be executed.
    }

    constructor() {
        clientCounter = 0;  // those client join contract assign there ID intially it set 0
    }

    receive() external payable { } // this allows the smart contract to receive ether

    function setManager(address managerAddress) public returns(string memory){ //setManager method will be used to set the manager address to variables
        // string memory store address of manager account instead of store data
        manager = payable(managerAddress);// managerAddress is consumed as a parameter and cast as payable to provide sending ether.
        return ""; // return payable address of manager
    }

    function joinAsClient() public payable returns(string memory){ //joinAsClient method will be used to make sure the client joins the contract.

        clients.push(client_account(clientCounter++, msg.sender, address(msg.sender).balance)); // push() array method to add items into a storage array.
        return ""; // return all client details
    }

    function deposit() public payable onlyClients{  // deposit == client to contract by onlyclient
        //deposit method will be used to send ETH from the client account to the contract.
```

```
    // We want this method to be callable only by clients who've joined the contract, so the onlyClient modifier is used for this restriction.
        payable(address(this)).transfer(msg.value); //transfer methods belongs to the contract, and it's dedicated to sending an indicated amount of ETH between addresses.
        // The payable keyword makes receipt of the ETH transfer possible so the amount of ETH indicated in the msg.value will be transferred to the contract address.
    }

    function withdraw(uint amount) public payable onlyClients{ // withdraw == contract to client by onlyclient
        payable(msg.sender).transfer(amount * 1 ether); // The address of the sender( ie contract ) is held in the msg.sender variable.
        //The withdraw method will be used to send ETH from the contract to the client account. It sends the unit of ETH indicated in the amount parameter, from the contract to the client who sent the transaction. We want this method to be callable only by clients who've joined the contract either,
        // so the onlyClient modifier is used for this restriction.
    }

    function sendInterest() public payable onlyManager{ //The sendInterest method will be used to send ETH as interest from the contract to all clients. can called by only manager
        for(uint i=0;i<clients.length;i++){ // check client in database
            address initialAddress = clients[i].client_address; // check client address

            payable(initialAddress).transfer(1 ether);

        }
    }

    function getContractBalance() public view returns(uint){ //getContractBalance method will be used to get the balance of the contract we deployed.
        return address(this).balance;
    }
}


//Output steps
//Initially All Acount 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 Onother
Address(eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
//       ii.Copy this address (eg.0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2)
and paste it in contract, infront of set Manager button
//       iii. click on set manager button, Now
Manager=0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
//Step 4: join as Client: Follow Following instructions
//       i.Select Onother
Address(eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)
//       ii.Copy this address (eg.0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db)
and paste it in contract, infront of joinAsClient button
//       iii.click on joinAsClient button, Now
Client=0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
//Initially Balanve 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 withdrow button
//       iii. Click on withdrow 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 getcontract balance button to view total balance in contract
```