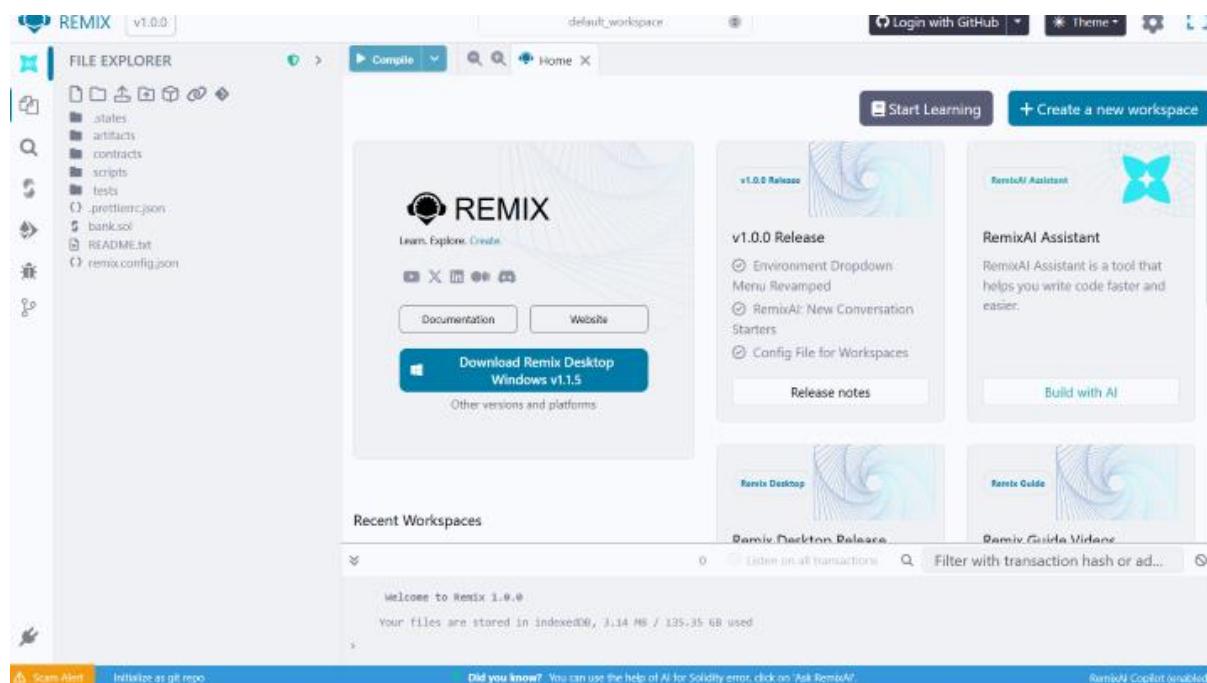


## ASSIGNMENT NO:03

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

- Deposit money
- Withdraw Money
- Show balance

### STEP 1:



### STEP 2:

A screenshot of the Remix IDE interface showing the Solidity code for a smart contract named 'MyBank'. The code defines a mapping of addresses to uint values for balances, sets an owner, and emits events for deposit and withdrawal. It includes a constructor that initializes the owner and emits a deposit event. The 'deposit' function allows deposits from payable senders, checks for valid addresses and overflow, and emits a deposit event. The code is annotated with comments and includes an 'Explain contract' button at the bottom. The bottom status bar shows 'Scan Alert', 'Initialize as git repo', 'Did you know?', and 'RemixAI Copilot (enabled)'.

## STEP 3:

The screenshot shows the Remix IDE interface with the 'Compiled' tab selected. The code editor contains the following Solidity code for the `MyBank` contract:

```
1 // SPDX-License-Identifier: Unlicensed
2 pragma solidity ^0.8.0;
3
4 contract MyBank {
5     mapping(address => uint) private _balances;
6     address public owner;
7
8     event LogDepositMade(address accountHolder, uint amount);
9
10    constructor() {
11        owner = msg.sender;
12        emit LogDepositMade(msg.sender, 1000);
13    }
14
15    function deposit() public payable returns (uint) {
16        require(msg.sender != address(0), "Invalid address");
17        require(_balances[msg.sender] + msg.value) >= _balances[msg.sender], "Overflow");
18
19        _balances[msg.sender] += msg.value;
20        emit LogDepositMade(msg.sender, msg.value);
21        return _balances[msg.sender];
22    }
23 }
```

Below the code editor, the transaction history shows a call to `MyBank.deposit()` from a specific address, resulting in a balance of 1000.

## STEP 4:

The screenshot shows the Remix IDE interface with the 'Compiled' tab selected. The code editor contains the following Solidity code for the `MyBank` contract, which includes new `withdraw` and `viewBalance` functions:

```
23
24     function withdraw(uint withdrawAmount) public returns (uint) {
25         require(msg.sender != address(0), "Invalid address");
26         require(_balances[msg.sender] >= withdrawAmount, "Insufficient balance");
27
28         _balances[msg.sender] -= withdrawAmount;
29         payable(msg.sender).transfer(withdrawAmount);
30         emit LogDepositMade(msg.sender, withdrawAmount); // You might want to use a different event here
31         return _balances[msg.sender];
32     }
33
34     function viewBalance() public view returns (uint) {
35         return _balances[msg.sender];
36     }
37 }
```

Below the code editor, the transaction history shows a call to `MyBank.viewBalance()` from a specific address, returning a balance of 1000.

## **STEP 5:**

- **Deploy:**

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar displays account information (0x5B3...), gas limit (3000000), and a button to 'Authorize Delegation'. Below that is a 'CONTRACT' section with 'MyBank - BankAccount.sol' selected. The main workspace shows the Solidity code for the MyBank contract, which includes a constructor, a deposit function, and a LogDepositMade event. At the bottom of the code editor, there's an 'Explain contract' button. The bottom right corner features the 'AI copilot' feature.

```
// SPDX-License-Identifier: Unlicensed
pragma solidity ^0.8.0;

contract MyBank {
    mapping(address => uint) private _balances;
    address public owner;

    event LogDepositMade(address accountHolder, uint amount);

    constructor() {
        owner = msg.sender;
        emit LogDepositMade(msg.sender, 1000);
    }

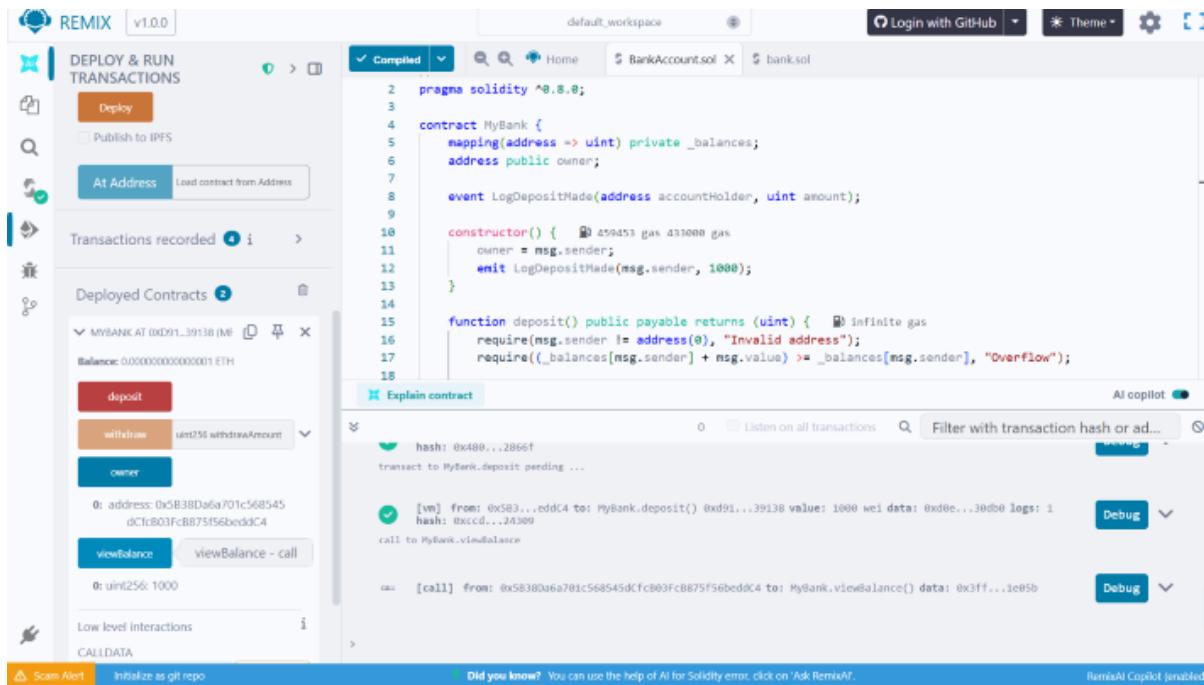
    function deposit() public payable returns (uint) {
        require(msg.sender != address(0), "Invalid address");
        require(_balances[msg.sender] + msg.value) >= _balances[msg.sender], "Overflow");
        _balances[msg.sender] += msg.value;
    }
}
```

- **Deposit money:**

The screenshot shows the Remix IDE interface with the following details:

- Deploy & Run Transactions** tab is active.
- Deploy** button is visible.
- Publish to IPFS** checkbox is unchecked.
- At Address** and **Load contract from Address** buttons are present.
- Transactions recorded**: 0 recorded.
- Deployed Contracts**: MYBANK AT 0xD91...3913B (MF)
- Balance**: 0.000000000000001 ETH
- Contract Methods**:
  - deposit**: deposit - transact (payable)
  - withdraw**: uint256 withdrawAmount
  - owner**
- Events**: creation of MyBank pending...
- Logs**:
  - [vm] from: 0x5B3...eddC4 to: MyBank.(constructor) value: 0 wei data: 0x000...e0033 logs: 1 hash: 0x400...3B06f
  - transact to MyBank.deposit pending...
  - [vm] from: 0x5B3...eddC4 to: MyBank.deposit() 0xd91...3913B value: 1000 wei data: 0xd0e...30db0 logs: 1 hash: 0xccc...7A309
- Low level interactions**: CALLDATA
- RemixAI Copilot** is enabled.

- Show balance after Deposit money:

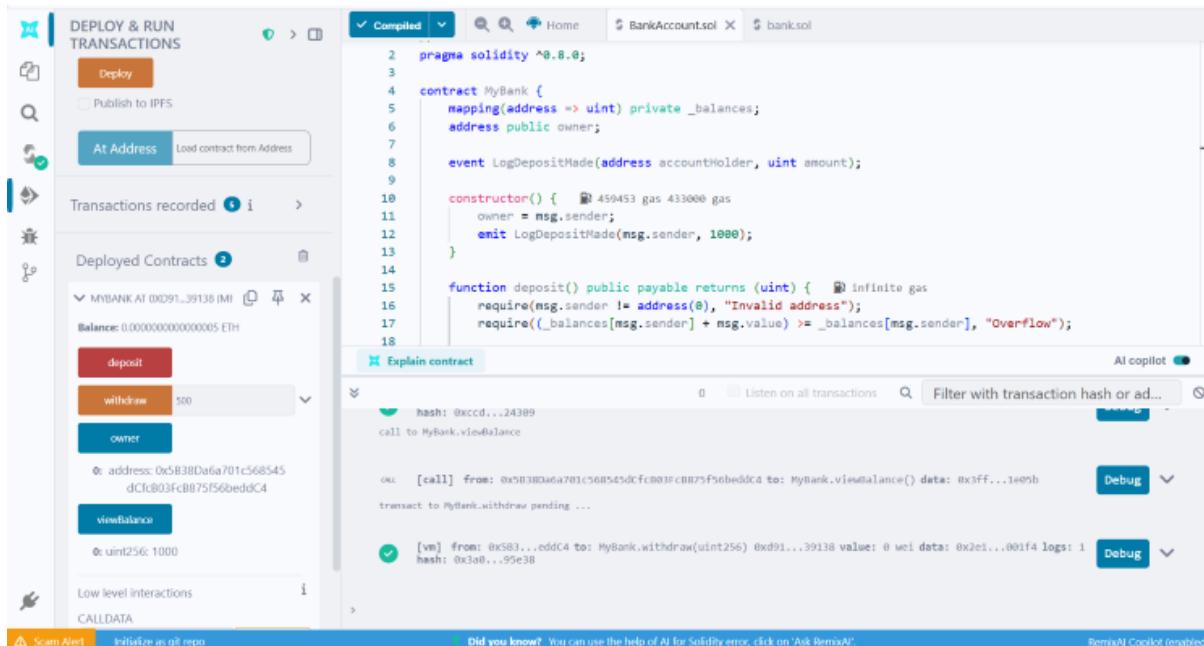


The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar has 'At Address' selected. In the center, the 'BankAccount.sol' file is open, showing the Solidity code for the MyBank contract. On the right, the transaction history shows a successful deposit transaction. The transaction details are as follows:

- hash: 0x480...2866f
- from: 0x583...edc4 to: MyBank.deposit() 0xd91...39138 value: 1000 wei
- data: 0x0e...30db0 logs: 1
- call to MyBank.deposit()

The contract's balance is shown as 0.0000000000000001 ETH.

- Withdraw Money

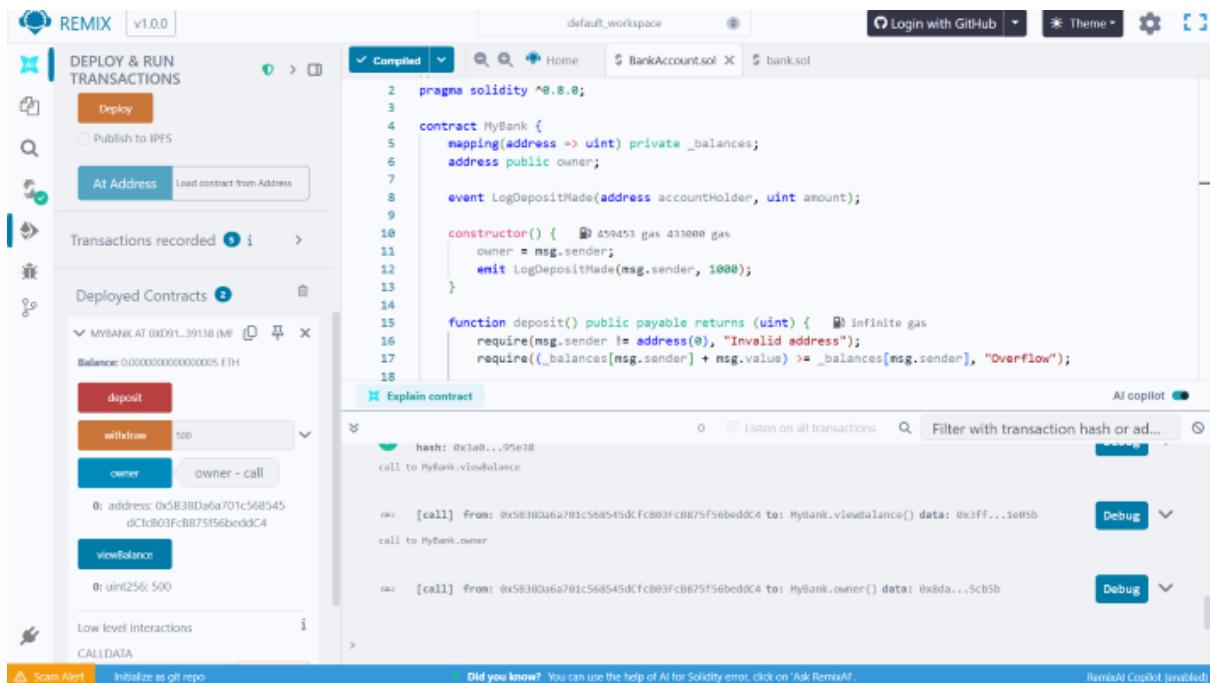


The screenshot shows the Remix IDE interface. The transaction history now includes a withdrawal transaction. The transaction details are as follows:

- hash: 0xccd...24309
- from: 0x583...edc4 to: MyBank.withdraw() 0xd91...39138 value: 500 wei
- data: 0x3ff...1e05b
- call to MyBank.withdraw()

The contract's balance is shown as 0.0000000000000005 ETH.

- Show Balance after Withdraw Money



The screenshot shows the REMIX IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar shows a deployed contract named 'MYBANK AT 0xD91...39138'. The balance is listed as 0.0000000000000005 ETH. Below the balance are three buttons: 'deposit', 'withdraw 500', and 'owner'. The 'withdraw' button is highlighted. On the right, the 'BankAccount.sol' file is displayed with the following Solidity code:

```

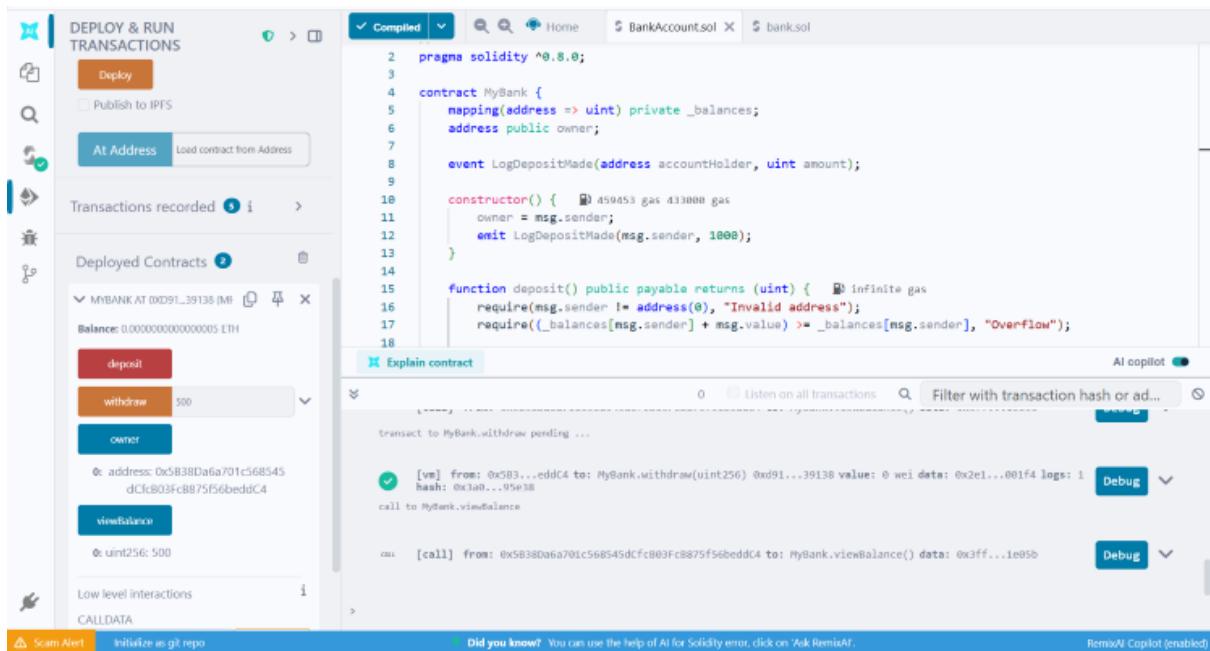
2 pragma solidity ^0.8.0;
3
4 contract MyBank {
5     mapping(address => uint) private _balances;
6     address public owner;
7
8     event LogDepositMade(address accountHolder, uint amount);
9
10    constructor() { 0x459453 gas 433000 gas
11        owner = msg.sender;
12        emit LogDepositMade(msg.sender, 1000);
13    }
14
15    function deposit() public payable returns (uint) { 0x infinite gas
16        require(msg.sender != address(0), "Invalid address");
17        require({_balances[msg.sender] + msg.value} >= {_balances[msg.sender]}, "Overflow");
18    }

```

The 'Explain contract' section shows the transaction details for the withdraw operation:

- hash: 0x3ab...95e38
- call to MyBank.viewBalance
- out [call] from: 0x5B38Da6a701c568545dCfcB03FcB75f56beddC4 to: MyBank.viewBalance() data: 0x3ff...1e05b
- call to MyBank.owner
- out [call] from: 0x5B38Da6a701c568545dCfcB03FcB75f56beddC4 to: MyBank.owner() data: 0x8da...5cb5b

- Check Owner:



This screenshot is identical to the one above, showing the REMIX IDE interface with the MyBank contract deployed at 0xD91...39138. The 'withdraw' button is highlighted. The 'Explain contract' section now highlights the transaction for the withdraw call, which is pending:

- transact to MyBank.withdraw pending ...
- [vm] from: 0x5B3...eddC4 to: MyBank.withdraw(uint256) 0xd91...39138 value: 0 wei data: 0x2e1...001f4 logs: 1
- hash: 0x3ab...95e38
- call to MyBank.viewBalance
- out [call] from: 0x5B38Da6a701c568545dCfcB03FcB75f56beddC4 to: MyBank.viewBalance() data: 0x3ff...1e05b