

## **Experiment No : 3**

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

Deposit Money

Withdraw Money

Show Balance

### **Aim:**

To create a secure and efficient smart contract for customer bank account management on a test network.

### **Objectives:**

Develop a Solidity smart contract for deposit, withdrawal, and balance check.

Deploy the contract on a test network.

Integrate a user interface for customer interactions.

### **Requirements:**

Solidity development environment (Remix, Truffle, or similar).

Access to a blockchain test network (Ropsten, Rinkeby, etc.).

Basic web development tools for the user interface.

Understanding of Ethereum and blockchain concepts.

Ethereum wallet (e.g., MetaMask) for interacting with the smart contract on the test network.

### **Theory:**

## **Step 1: Set Up Your Development Environment**

Before we begin, make sure you have a development environment set up for Ethereum smart contract development. You can use Remix, Truffle, or another Solidity IDE. Here we are going to use Remix IDE;

Setting up Remix for Solidity development is relatively straightforward, and it's a web-based IDE, which means you don't need to install anything locally. Here are the steps to set up Remix for beginners:

### 1. Access Remix:

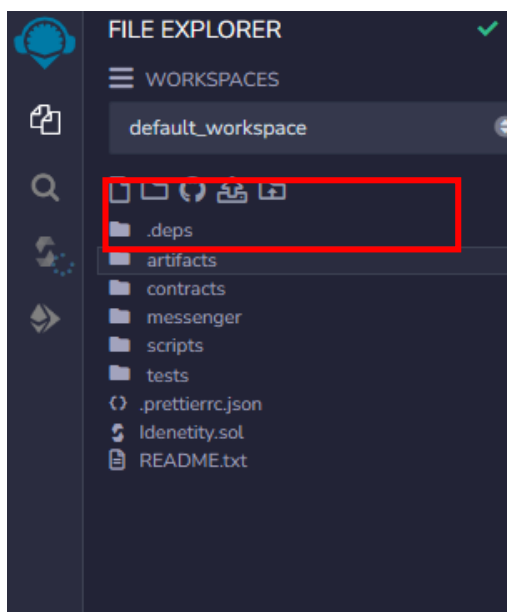
- Open your web browser and navigate to Remix's official website:  
<https://remix.ethereum.org/>

### 2. Choose Environment:

- On the Remix website, you'll see different environments to choose from. For beginners, you can start with the "Remix - Solidity IDE" environment.

### 3. Create a New File:

- Remix will open with a file named "Untitled.sol." You can start coding in this file or create a new one by clicking the file button at the top-left corner and selecting "Solidity."



### 4. Write Your Solidity Code:

- You can now start writing your Solidity code in the code editor. You can also paste existing code or templates into the editor.

Remix is a powerful tool for Solidity development and is beginner-friendly. It provides code highlighting, error checking, and a user-friendly interface for deploying and interacting with smart contracts. You can experiment with Solidity and Ethereum development right in your web browser without the need for a local setup.

## Step 2: Create a New Solidity Contract

Certainly! The provided Solidity code defines a smart contract named `BankAccount` with the ability to deposit, withdraw, and check the balance. Here's a step-by-step explanation of the code:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.4.16 < 0.9.0;

contract BankAccount {

    address public owner;
    uint256 public balance;

    constructor(address _owner) {
        owner = _owner;
        balance = 0;
    }

    modifier onlyOwner {
        require(msg.sender == owner, "Only the owner can call this
function.");
        _;
    }

    function deposit(uint256 amount) public onlyOwner {
        balance += amount;
    }

    function withdraw(uint256 amount) public onlyOwner {
        require(balance >= amount, "Insufficient balance.");
        balance -= amount;
    }

    function showBalance() public view returns (uint256) {
        return balance;
    }
}
```

### 1. Pragma Directive:

```
// SPDX-License-Identifier: GPL-3.0  
pragma solidity >= 0.4.16 < 0.9.0;
```

- This section specifies the Solidity compiler version and the license for the smart contract. In this case, it's designed to work with Solidity versions greater than or equal to 0.4.16 and less than 0.9.0.

### 2. Contract Declaration:

```
contract BankAccount {
```

- This defines the start of the `BankAccount` smart contract.

### 3. State Variables:

```
address public owner;  
uint256 public balance;
```

- These are state variables of the contract:

- `owner`: An Ethereum address that will represent the owner of the bank account.

- `balance`: A `uint256` variable that will store the account balance. The `public` keyword allows external access to these variables.

### 4. Constructor:

```
constructor(address owner) {  
    owner = _owner;  
    balance = 0;  
}
```

- This is the constructor function. It initializes the contract with an address (the owner) and sets the initial balance to 0.

- The `constructor` is executed only once during contract deployment.

### 5. Modifier:

```
modifier onlyOwner {  
    require(msg.sender == owner, "Only the owner can call this  
function.");  
    ;  
}
```

- This is a custom modifier called `onlyOwner`.

- It checks if the caller of a function is the owner of the contract (as specified in `owner`).
- If the condition is met, the `\_` underscore allows the function that uses this modifier to execute. If the condition is not met, it throws an error with the provided message.

#### 6. Deposit Function:

```
function deposit(uint256 amount) public onlyOwner {  
    balance += amount;  
}
```

- This function allows the owner to deposit an amount of Ether (`amount`) into the contract's balance.
- It uses the `onlyOwner` modifier to ensure only the owner can deposit funds.

#### 7. Withdraw Function:

```
function withdraw(uint256 amount) public onlyOwner {  
    require(balance >= amount, "Insufficient balance.");  
    balance -= amount;  
}
```

- This function allows the owner to withdraw an amount of Ether (`amount`) from the contract's balance.
- It checks if the contract's balance is sufficient to cover the withdrawal amount and then subtracts the withdrawn amount from the balance.

#### 8. Show Balance Function:

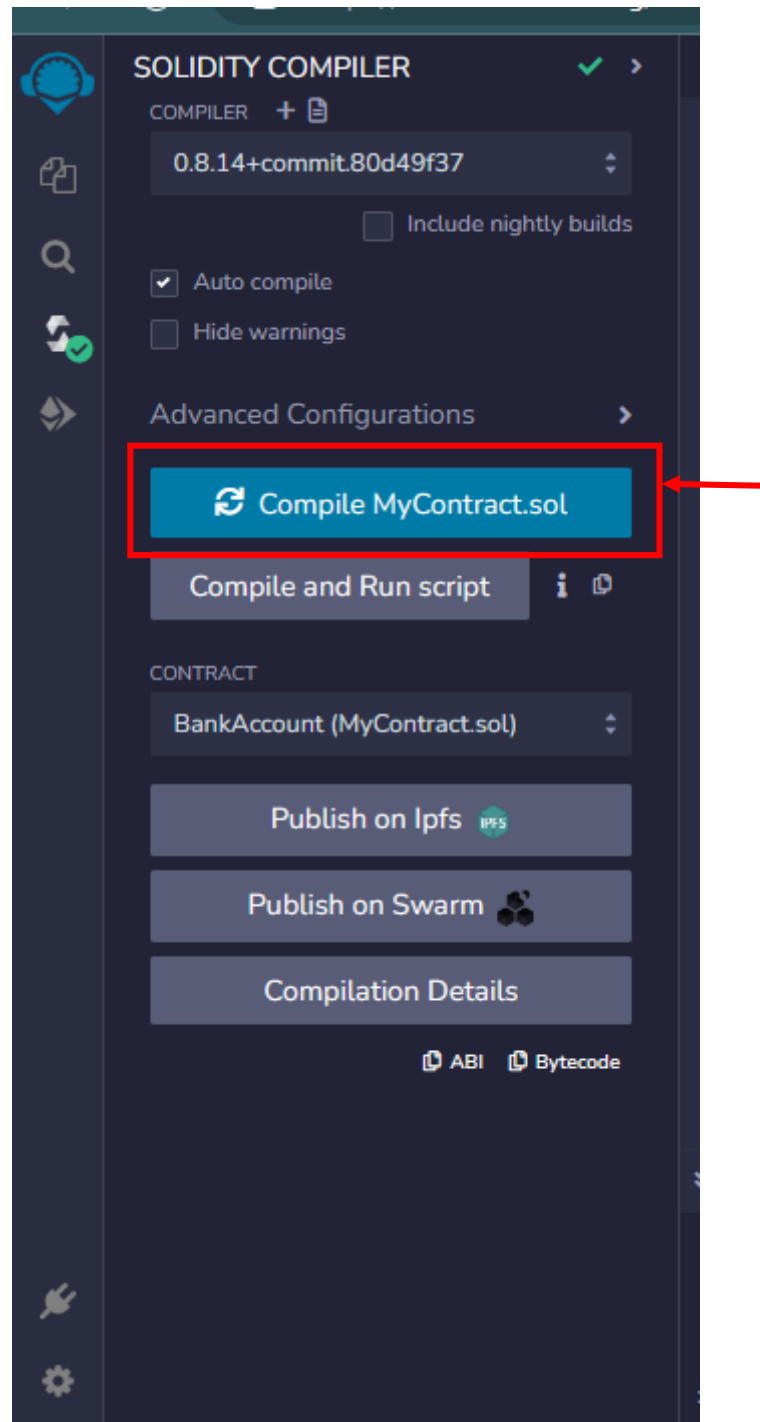
```
function showBalance() public view returns (uint256) {  
    return balance;  
}
```

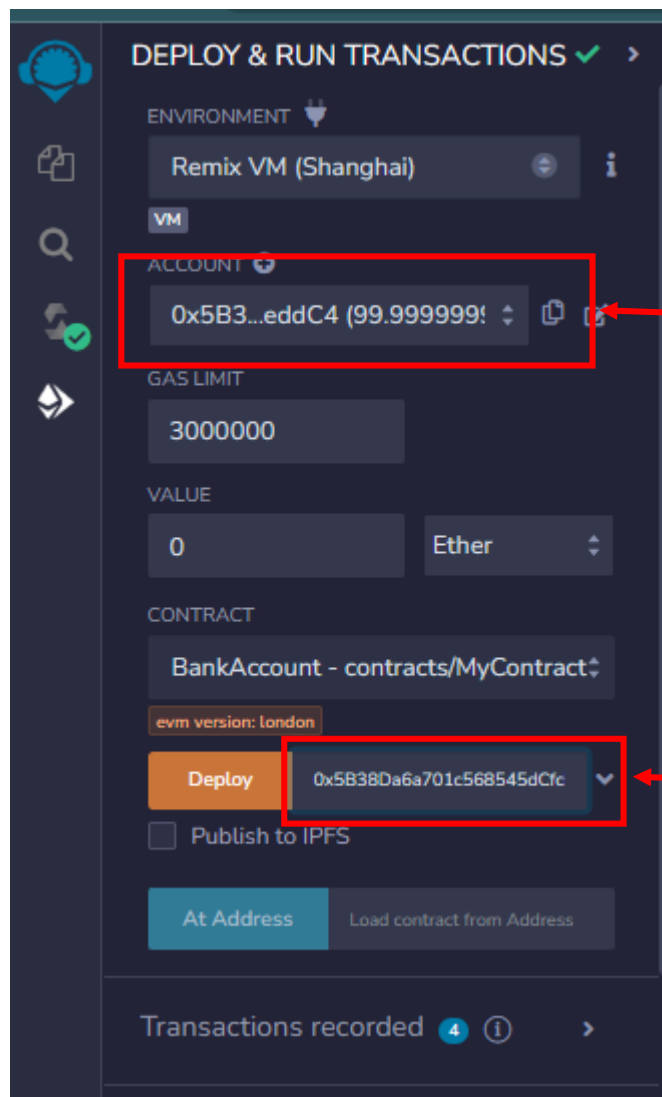
- This function allows anyone to query and retrieve the current balance. It's a `view` function, meaning it doesn't modify the state and is read-only.

The contract essentially creates a simple bank account where the owner can deposit and withdraw funds while checking the account balance. The use of modifiers ensures that only the owner can perform deposit and withdrawal operations.

### Step 3: Compile and Deploy

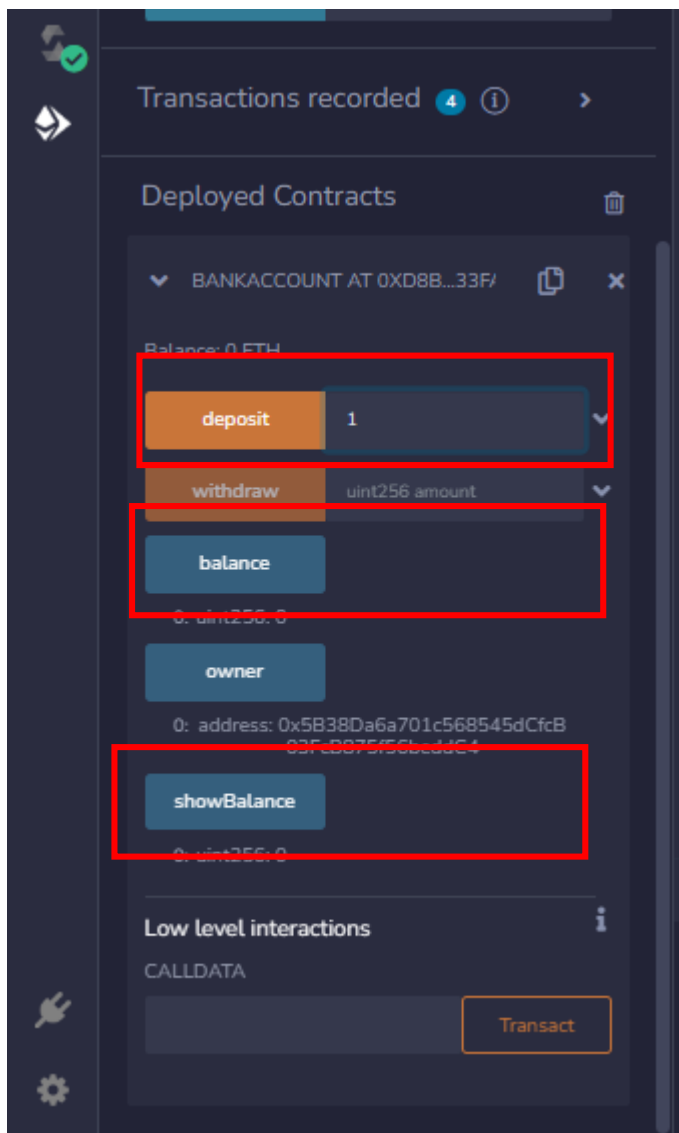
Compile your Solidity code and deploy the contract on a test network. You'll need some test Ether for deployment. Which will already given to you by RemixIDE.





Copy The Address of above field from above field and paste it in field beside Deploy. Before Deploying.

#### Step 4: Interact with the Contract



You can check balance, owners address, and deposit and withdraw money from this section just by clicking on it. Fill the data provided before clicking the buttons.

After deployment, you can interact with the contract using a web3 library or a blockchain explorer. You can call the `deposit`, `withdraw`, and `getBalance` functions to test the contract's functionality.

## Step 6: Testing

Test each function to ensure that deposits, withdrawals, and balance checks work as expected.

## Step 7: Verify on a Block Explorer

Use an Ethereum block explorer (e.g., Etherscan for the testnet you deployed on) to verify the transactions and the updated account balance.

Remember that this example is for educational purposes and should only be used on test networks. Real-world financial applications require extensive security and regulatory considerations.



Etherscan

Home

Blockchain

Tokens

NFTs

Resources

Developers

More

Sign In


Transaction Details

Buy

Exchange

Play

Gaming

Sponsored:  bc.game - Free Bonus Up To 5 BTC Everyday! Live casino & 20k slots [Play Now](#)

Overview

State

Comments

More

Transaction Hash:

0xe98beca8e32b679d7bc56525502c37b8faf9510306b763f12541fecd8a453ae3

Status:

Success

Block:

18390569 4 Block Confirmations

Timestamp:

47 secs ago (Oct-20-2023 09:13:35 AM +UTC)

Transaction Action:

Transfer 4.029691476938594566 ETH To 0x9c595f...2c748726

This example provides a basic smart contract for a bank account on a test network. Depending on your use case, you may need to add more features, such as access control, event logging, and more robust error handling. Be sure to thoroughly test your contract on the test network before deploying it to the mainnet.