Experiment No.6
To design a smart contract implementing the concepts of wallet and transaction using solidity
Date of Performance:
Date of Submission:

AIM: To design a smart contract implementing the concept of wallet and transaction using solidity

Objective: To develop a program using solidity to create a demonstration of wallet and perform transaction on the wallet

Theory:

A blockchain wallet is a cryptocurrency wallet that allows users to manage different kinds of cryptocurrencies—for example, Bitcoin or Ethereum. A blockchain wallet helps someone exchange funds easily. Transactions are secure, as they are cryptographically signed. The wallet is accessible from web devices, including mobile ones, and the privacy and identity of the user are maintained. So a blockchain wallet provides all the features that are necessary for safe and secure transfers and exchanges of funds between different parties.

Helps in Exchange of funds
Privacy is maintained
Blockchain wallet
Transactions are secure
Accessible from web or mobile devices

Fig, 6.1. Illustrative view of Wallet

Blockchain Wallet Types

There are two types of blockchain wallets based on private keys: hot wallets and cold wallets. Hot wallets are like normal wallets that we carry for day-to-day transactions, and these wallets are user-friendly. Cold wallets are similar to a vault; they store cryptocurrencies with a high level of security.

Hot Wallets and Cold Wallets

Hot wallets are online wallets through which cryptocurrencies can be transferred quickly. They are available online. Examples are Coinbase and Blockchain.info. Cold wallets are digital offline wallets where the transactions are signed offline and then disclosed online. They are not maintained in the cloud on the internet; they are maintained offline to have high security. Examples of cold wallets are Trezor and Ledger.

With hot wallets, private keys are stored in the cloud for faster transfer. With cold wallets, private keys are stored in separate hardware that is not connected to the internet or the cloud, or they are stored on a paper document. Hot wallets are easy to access online 24/7 and can be accessed via a desktop or mobile device, but there is the risk of unrecoverable theft if hacked. With cold wallets, the method of the transaction helps in protecting the wallet from unauthorized access (hacking and other online vulnerabilities).

Process:

Step 1. Open Remix **IDE** by typing URL https://remix.ethereum.org/.

Step 2. In Remix IDE select 'Solidity' plugins

Step 3. Click on File Explorer

Step 4. In the default workspace click on 'create new file'

Step 5. Give suitable name to the *file* with extension .sol

Step 6. Type the (Wallet and transaction) code in the editor section of the Remix IDE for newly created file (.sol)

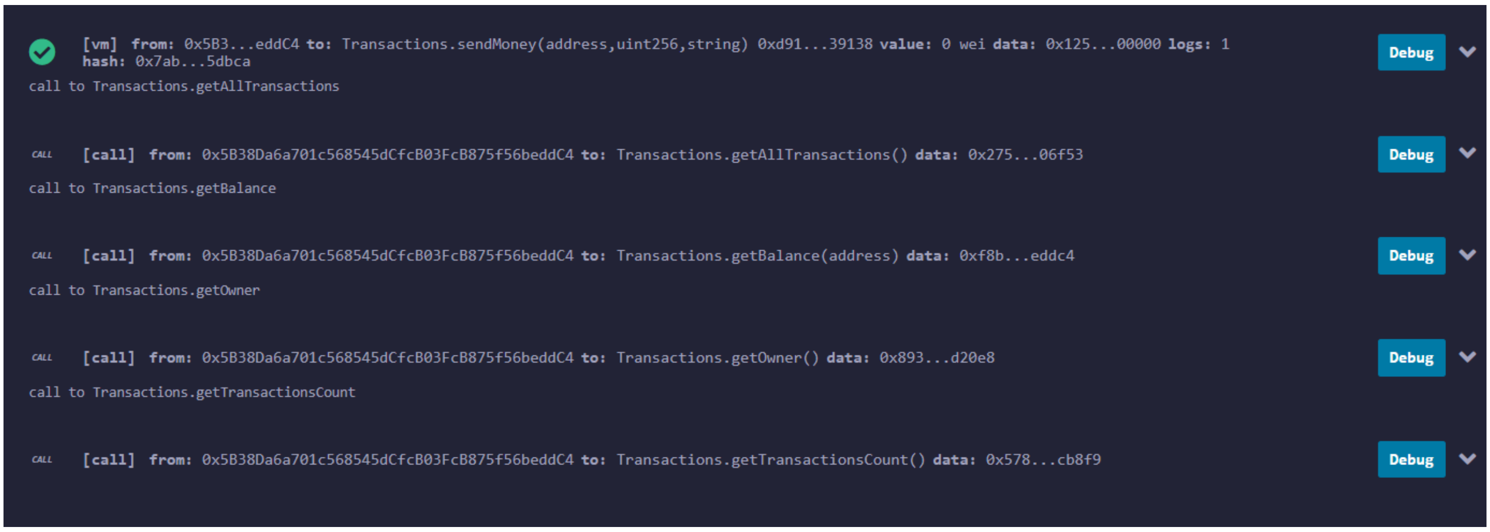
Step 7. After typing the code for the smart contract in the newly creates .sol file,click on the compiler option available and then compile the file

Step 8. If no error, then click on the 'Deploy and Run' to execute the smart contract

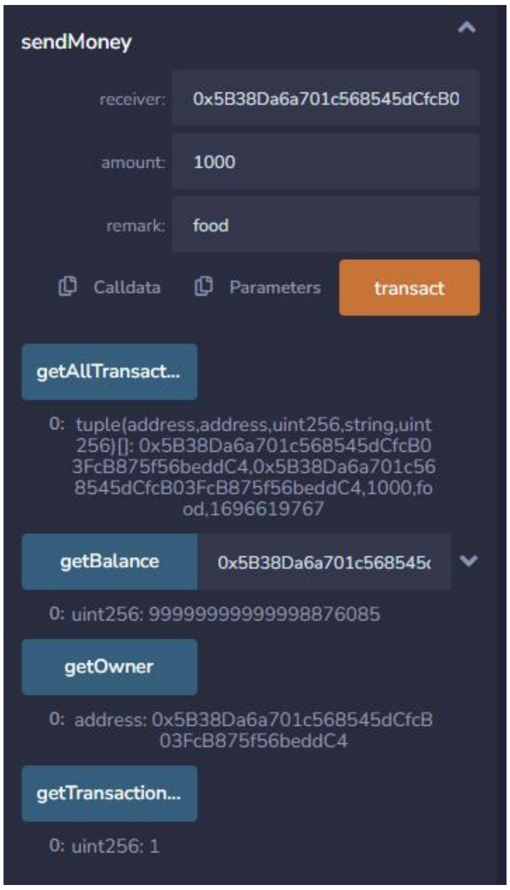
Code:

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

contract Transactions {
    address private owner;
    uint256 transactionCounts;
    mapping (address => uint) balanceOf;

     event Transfer(address indexed sender, address indexed receiver, uint256 amount, string
remark, uint256 timestamp);

    struct TransferStruct {
        address sender;
        address receiver;
        uint256 amount;
        string remark;
        uint256 timestamp;
    }

    TransferStruct[] transactions;

    constructor() {
        owner = msg.sender;
        balanceOf[tx.origin] = msg.sender.balance;
    }

    function getOwner() public view returns (address) {
        return owner;
    }
    function sendMoney(address payable receiver, uint256 amount, string memory remark)
public returns(bool success) {
        if (balanceOf[owner] < amount) return false;
        balanceOf[owner] -= amount;
        balanceOf[receiver] += amount;

        transactionCounts += 1;
        transactions.push(
            TransferStruct(
                owner,
                receiver,
                amount,
                remark,
                block.timestamp
            )
        );
        emit Transfer(msg.sender, receiver, amount, remark, block.timestamp);
        return true;
    }
    function getBalance(address addr) public view returns(uint) {
        return balanceOf[addr];
    }

    function getAllTransactions() public view returns(TransferStruct[] memory) {
        return transactions;
    }
    function getTransactionsCount() public view returns(uint256) {
        return transactionCounts;
    }
}
```

Output:



```
[vm] from: 0x5B3...eddC4 to: Transactions.sendMoney(address,uint256,string) 0xd91...39138 value: 0 wei data: 0x125...00000 logs: 1
hash: 0x7ab...5dbca
call to Transactions.getAllTransactions

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Transactions.getAllTransactions() data: 0x275...06f53
call to Transactions.getBalance

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Transactions.getBalance(address) data: 0xf8b...eddc4
call to Transactions.getOwner

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Transactions.getOwner() data: 0x893...d20e8
call to Transactions.getTransactionsCount

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Transactions.getTransactionsCount() data: 0x578...cb8f9
```



```
sendMoney
receiver: 0x5B38Da6a701c568545dCfcB0
amount: 1000
remark: food
Calldata   Parameters   transact

getAllTransact...
0: tuple(address,address,uint256,string,uint
   256)[]: 0x5B38Da6a701c568545dCfcB0
   3FcB875f56beddC4,0x5B38Da6a701c56
   8545dCfcB03FcB875f56beddC4,1000,fo
   od,1696619767

getBalance   0x5B38Da6a701c568545
0: uint256: 99999999999998876085

getOwner
0: address: 0x5B38Da6a701c568545dCfcB
   03FcB875f56beddC4

getTransaction...
0: uint256: 1
```

Conclusion:

The provided Solidity smart contract demonstrates the application of features to perform transactions on the Ethereum blockchain:

1. Constructor: Initializes the contract and sets the owner's address and balance.
2. Events: Logs important activities on the blockchain, such as transactions.
3. Structs: Defines a structured data type to store transaction details.
4. Mappings: Stores balances associated with Ethereum addresses.

Functions:

1. getOwner(): Allows reading the contract owner's address.
2. sendMoney(): Lets the contract owner send Ether to others.
3. getBalance(): Retrieves the balance of a specific address.
4. getAllTransactions(): Provides a history of all transactions.
5. getTransactionsCount(): Counts the total number of transactions.

Overall, this contract illustrates how Solidity features are used to create a transaction management system with access control, event logging, and data querying capabilities.