

Homework2

Programming a Better Faucet contract

108321033 吳明騰

108321056 唐泳烽

108321062 許恩慈

Using Truffle

Introduction: In this section, we talk step by step how to use truffle to connect with the local blockchain simulation Ganache and public testnet Sepolia. Our code is in this [Github link](#)

Step 1: Download - [Node.js](#)

Step 2: use npm to install truffle, [truffle-hdwallet-provider](#) and [dotenv](#)

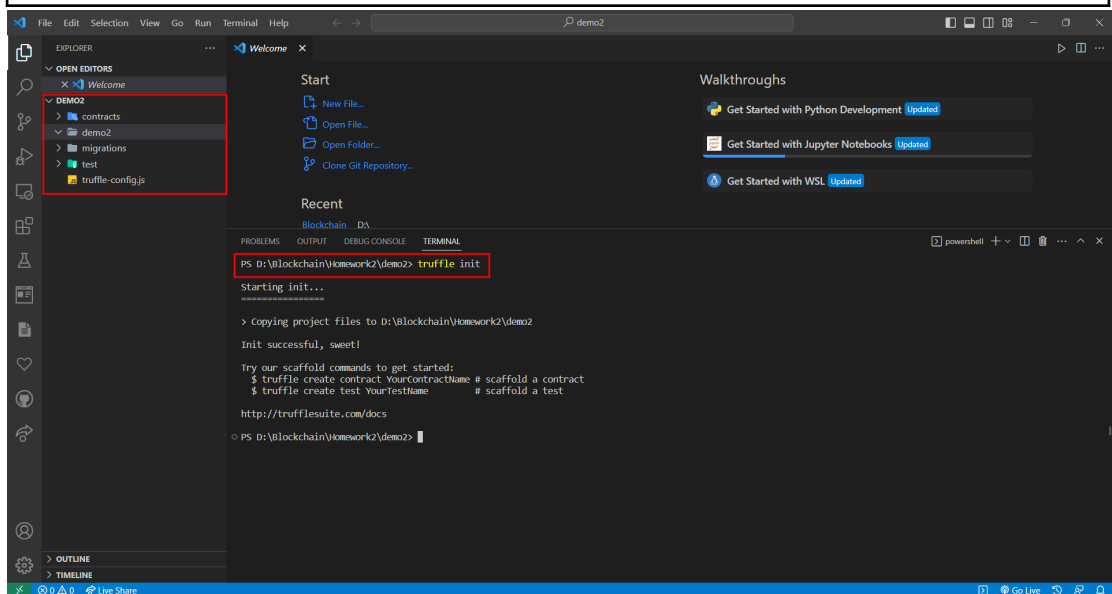
```
$ npm install -g truffle
$ npm install @truffle/hdwallet-provider
$ npm install dotenv
```

Step 3: Solve the problem of "Cannot load because script execution is disabled on this system..." when inputting commands on Windows ([Reference URL](#))

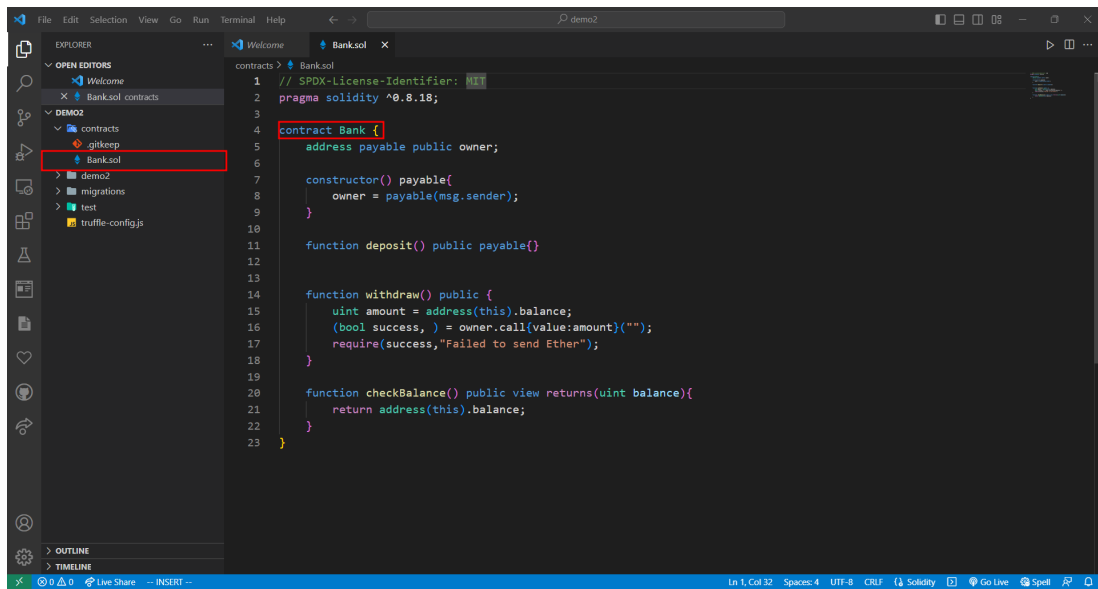
```
$ Get-ExecutionPolicy
$ Set-ExecutionPolicy RemoteSigned
```

Step 4: initialize project

```
$ truffle init
```



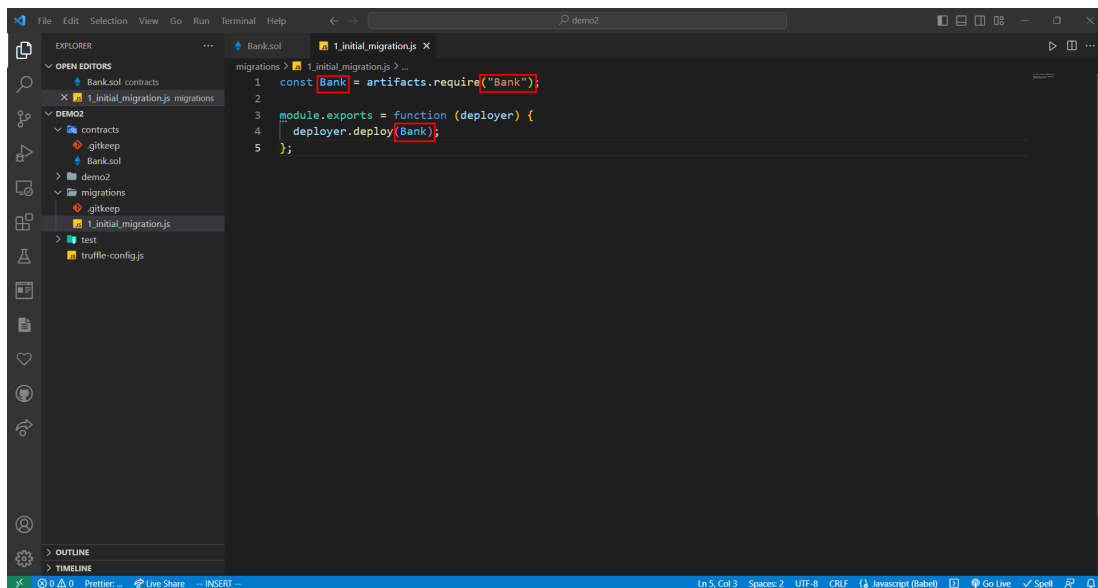
Step 5: Add a smart contract under the ./contract folder, which is a .sol file. Remember that the file name must be consistent with the name of the smart contract



The screenshot shows the Visual Studio Code editor with the 'Bank.sol' file open. The Explorer sidebar on the left shows the project structure with 'contracts' and 'Bank.sol' highlighted. The main editor displays the Solidity code for the 'Bank' contract.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.18;
3
4 contract Bank {
5     address payable public owner;
6
7     constructor() payable {
8         owner = payable(msg.sender);
9     }
10
11     function deposit() public payable {}
12
13
14     function withdraw() public {
15         uint amount = address(this).balance;
16         (bool success, ) = owner.call{value:amount}("");
17         require(success, "Failed to send Ether");
18     }
19
20     function checkBalance() public view returns(uint balance){
21         return address(this).balance;
22     }
23 }
```

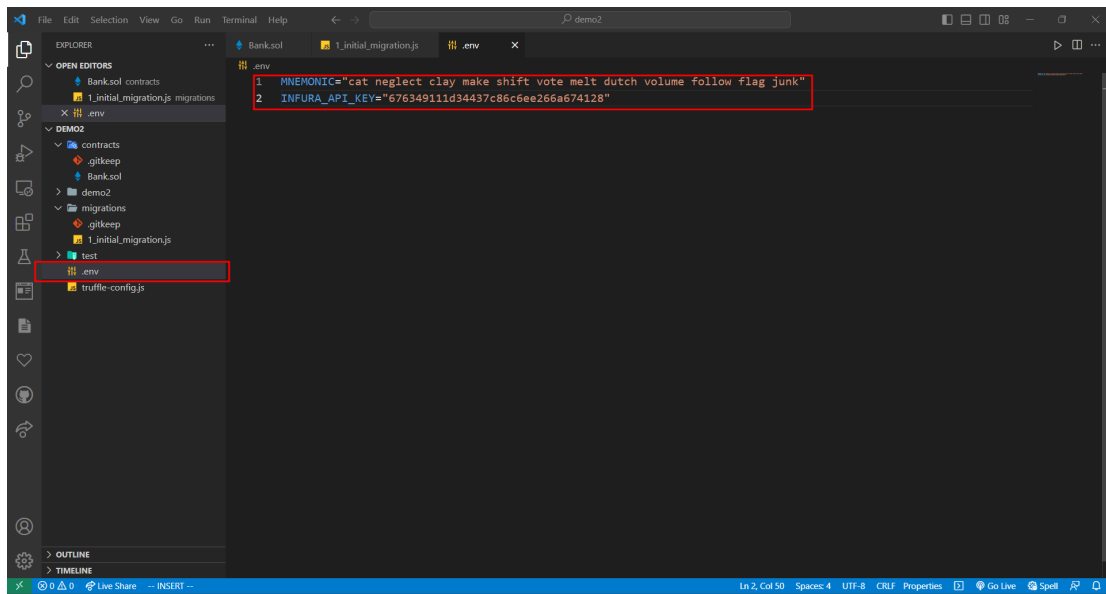
Step 6: Add 1_initial_migration.js to ./migrations and remember to change the smart contract name



The screenshot shows the Visual Studio Code editor with the '1_initial_migration.js' file open. The Explorer sidebar on the left shows the project structure with 'migrations' and '1_initial_migration.js' highlighted. The main editor displays the JavaScript code for the migration.

```
1 const Bank = artifacts.require("Bank");
2
3 module.exports = function (deployer) {
4     deployer.deploy(Bank);
5 };
```

Step 7: Add a new `./env` file, and add your own EOA mnemonic, and infura's api key.



Step 8: Modify `./truffle-config.js`

```
require('dotenv').config();  
var HDWalletProvider = require("@truffle/hdwallet-provider");
```

Establish a connection with local blockchain simulation Ganache.

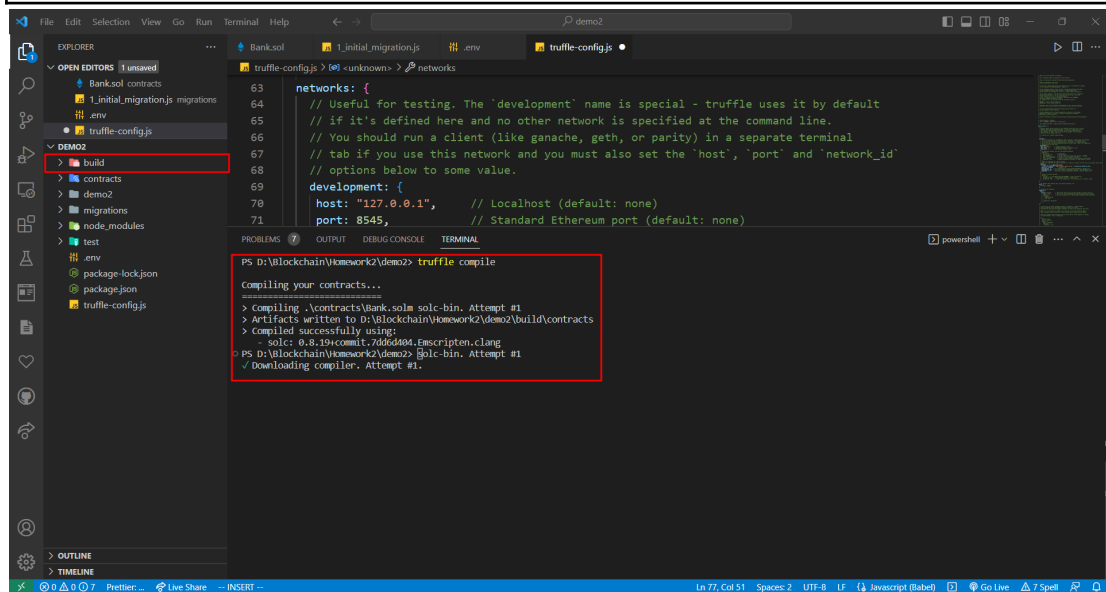
```
development: {  
  host: "127.0.0.1",  
  port: 7545,  
  network_id: "5777",  
},
```

Establish a connection with public testnet Sepolia

```
sepolia: {  
  provider: () => new HDWalletProvider(  
    process.env.MNEMONIC,  
    `https://sepolia.infura.io/v3/` + process.env.INFURA_API_KEY  
  ),  
  network_id: 11155111,  
  confirmations: 2,  
  timeoutBlocks: 200,  
  skipDryRun: true  
},
```

Step 9: Execute the following command, if the compilation is successful, the folder `./build` will be added automatically

```
$ truffle compile
```



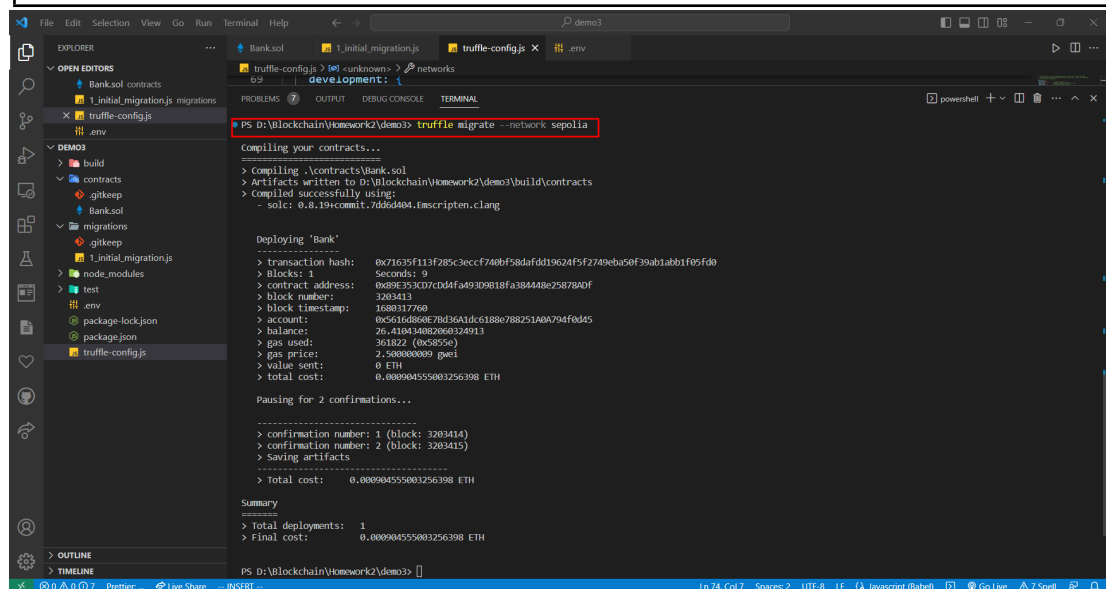
The screenshot shows the VS Code interface with a project named 'demo2'. The Explorer sidebar on the left shows the file structure, including 'contracts', 'migrations', 'test', and 'build' (highlighted with a red box). The main editor displays the 'truffle-config.js' file. The terminal at the bottom shows the command 'truffle compile' being executed, with output indicating successful compilation of contracts to the build directory.

```
PS D:\Blockchain\homework2\demo2> truffle compile
Compiling your contracts...
=====
> Compiling ./contracts/Bank.solm solc-bin, Attempt #1
> Artifacts written to D:\Blockchain\homework2\demo2\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd5d404.Emscripten.clang
PS D:\Blockchain\homework2\demo2> [solc-bin, Attempt #1
✓ Downloading compiler, Attempt #1.
```

Step 10: Execute the following command to deploy a smart contract. The first one is for ganache and the other one is for Sepolia.

```
$ truffle migrate
```

```
$ truffle migrate --network sepolia
```



The screenshot shows the VS Code interface with a project named 'demo3'. The Explorer sidebar on the left shows the file structure, including 'contracts', 'migrations', 'test', and 'build' (highlighted with a red box). The main editor displays the 'truffle-config.js' file. The terminal at the bottom shows the command 'truffle migrate --network sepolia' being executed, with output indicating successful compilation and deployment of the 'Bank' contract to the Sepolia network.

```
PS D:\Blockchain\homework2\demo3> truffle migrate --network sepolia
Compiling your contracts...
=====
> Compiling ./contracts/Bank.sol
> Artifacts written to D:\Blockchain\homework2\demo3\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd5d404.Emscripten.clang

Deploying 'Bank'
-----
> transaction hash: 0x71635f113f285c3ecf740bf58d4fd19624f5f2749eba5ef39ab1abb1f5fd8
> Blocks: 1
> contract address: 0x89e353cd7cd4fa493d9818fa38448e25878d0f
> block number: 3283413
> block timestamp: 1680317760
> account: 0x561d860e78d36a1dc6188e788251a0a794f0d45
> balance: 26.410434082060324913
> gas used: 361822 (0xc855a)
> gas price: 2.500000000 gwei
> value sent: 0 ETH
> total cost: 0.000904555003256398 ETH

Pausing for 2 confirmations...

> confirmation number: 1 (block: 3203414)
> confirmation number: 2 (block: 3203415)
> Saving artifacts
-----
> Total cost: 0.000904555003256398 ETH

Summary
-----
> Total deployments: 1
> Final cost: 0.000904555003256398 ETH

PS D:\Blockchain\homework2\demo3>
```

Small Project

Introduction: In this section, we'll explain how to use javascript to interact with smart contracts and introduce our small project.

Truffle Console

The below commands are to Enter the truffle console. The first one is for ganache and the other one is for Sepolia.

```
$ truffle console
```

```
$ truffle console --network sepolia
```

We just need to type javascript code in the truffle console to call the smart contract. However, instead of using this approach, we write a javascript file [run.js](#) and use the below commands to run it in the truffle console. The first one is for ganache and the other one is for Sepolia.

```
$ truffle exec ./run.js    (truffle exec ./run.js --network sepolia)
Using network 'development'.
```

```
<Caller>
```

```
Smart contract address: 0x342B197aaa0317bf476A3363C074672eEC96F3FA
```

```
The owner: 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

```
The balance: 0 ether
```

```
-----
```

```
<Callee>
```

```
Smart contract address: 0xCe2b6B2A957E4a3d71D502F44D0FC6437c0A73A6
```

```
The owner: 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

```
The balance: 0 ether
```

```
=====
```

```
<Caller>
```

```
Deposit 0.3 ether from 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

```
The balance: 0.3 ether
```

```
-----
```

```
<Callee>
```

```
Deposit 0.3 ether from 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

```
The balance: 0.3 ether
```

```
=====
```

```
<Caller>
```

```
Withdraw 0.1 ether to 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

```
The balance: 0.2 ether
```

```
-----
```

```
<Callee>
```

```
Withdraw 0.1 ether to 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
```

The balance: 0.2 ether

Use the smart contract 'caller' to call the smart contract 'callee's' function withdraw_myself to Withdraw 0.1 ether to 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29

<Caller>

The balance: 0.1 ether

<Callee>

The balance: 0.1 ether

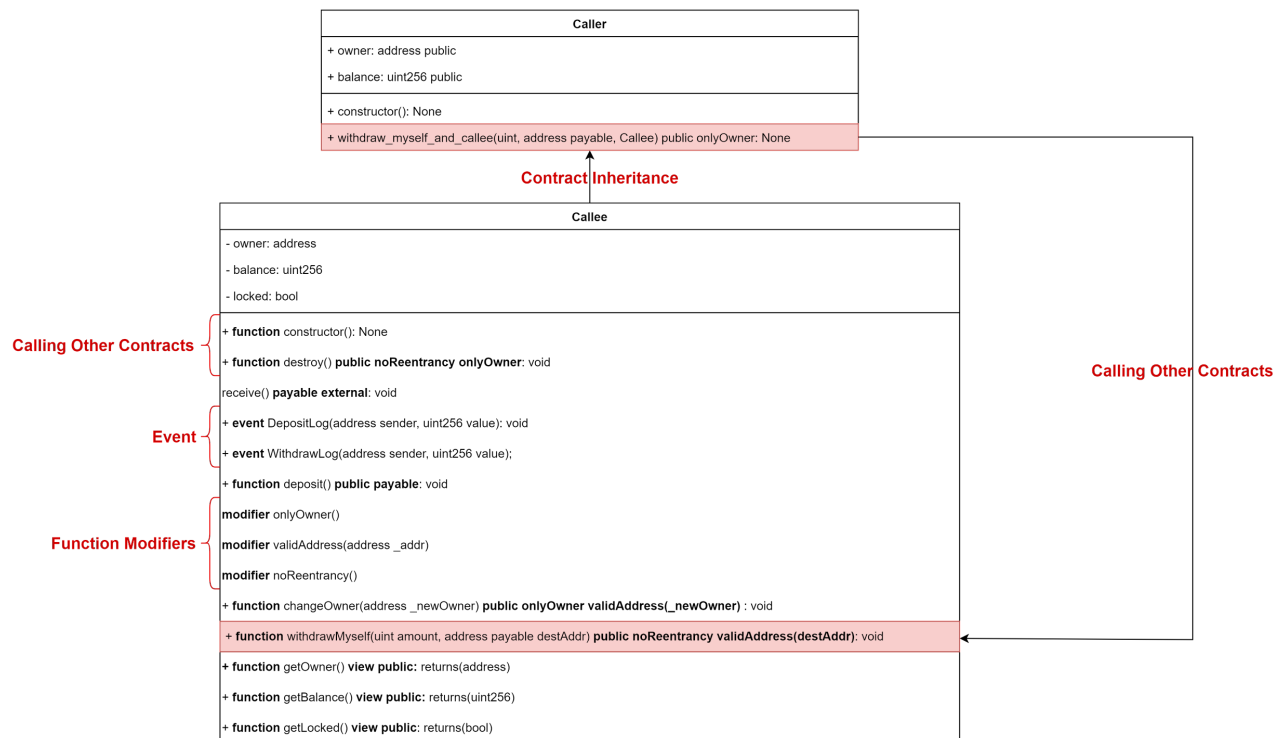
These are the tractions for the run.js.

Ganache			
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS
CURRENT BLOCK 318	GAS PRICE 20000000000	GAS LIMIT 7721973	WASMFORM MERGE
NETWORK ID 5777	RPC SERVERS HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE WIDE-SETTINGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES			
SWITCH			
TX HASH			
0xb973d408c61809e02df72348a95474d04fb830b1a79afc71b3d00dd29458b30f			
FROM ADDRESS 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29	TO CONTRACT ADDRESS Caller	GAS USED 62431	VALUE 0
CONTRACT CALL			
TX HASH			
0x0568810bd8a53706fbfbdafdc4bac2a10e71d388ad610f66a35adc3d2b2f8			
FROM ADDRESS 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29	TO CONTRACT ADDRESS Callee	GAS USED 47037	VALUE 0
CONTRACT CALL			
TX HASH			
0xc01de696755a23c976b382315b0bd82c358987cfe3543d6306bdf68055d357c			
FROM ADDRESS 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29	TO CONTRACT ADDRESS Caller	GAS USED 47037	VALUE 0
CONTRACT CALL			
TX HASH			
0xab56c999b0590a07f25b2ee9ec8275e344e46e57e57942c91a71f55d1e505081			
FROM ADDRESS 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29	TO CONTRACT ADDRESS Callee	GAS USED 45161	VALUE 3000000000000000000
CONTRACT CALL			
TX HASH			
0x17c0990eb05aef2c404c085dc96c23b03ef481248d7a12cdd659c37e4025b56			
FROM ADDRESS 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29	TO CONTRACT ADDRESS Caller	GAS USED 45161	VALUE 3000000000000000000
CONTRACT CALL			

These are the events for the run.js.

Ganache			
ACCOUNTS	BLOCKS	TRANSACTIONS	EVENTS
CURRENT BLOCK 318	GAS PRICE 20000000000	GAS LIMIT 7721973	WASMFORM MERGE
NETWORK ID 5777	RPC SERVERS HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE WIDE-SETTINGS
SEARCH FOR BLOCK NUMBERS OR TX HASHES			
SWITCH			
EVENT NAME			
DepositLog			
CONTRACT Callee	TX HASH 0xb973d408c61809e02df72348a95474d04fb830b1a79afc71b3d00dd29458b30f	LOG INDEX 0	BLOCK TIME 2023-04-10 13:48:16
EVENT NAME			
DepositLog			
CONTRACT Callee	TX HASH 0x0568810bd8a53706fbfbdafdc4bac2a10e71d388ad610f66a35adc3d2b2f8	LOG INDEX 0	BLOCK TIME 2023-04-10 13:48:16
EVENT NAME			
DepositLog			
CONTRACT Caller	TX HASH 0xc01de696755a23c976b382315b0bd82c358987cfe3543d6306bdf68055d357c	LOG INDEX 0	BLOCK TIME 2023-04-10 13:48:16
EVENT NAME			
DepositLog			
CONTRACT Callee	TX HASH 0xab56c999b0590a07f25b2ee9ec8275e344e46e57e57942c91a71f55d1e505081	LOG INDEX 0	BLOCK TIME 2023-04-10 13:48:16
EVENT NAME			
DepositLog			
CONTRACT Caller	TX HASH 0x17c0990eb05aef2c404c085dc96c23b03ef481248d7a12cdd659c37e4025b56	LOG INDEX 0	BLOCK TIME 2023-04-10 13:48:16

Project Structure: Our code is in this [Github link](#)



Constructor and Selfdestruct

Constructor is only called once when deploying the contract, and is used to deploy the initial value. Here you can use it to view the account that deployed the smart contract. ([Github Link](#))

```

constructor() {
    owner = msg.sender; // store information who deployed contract
}

```

Selfdestruct is used to delete the contract, and can also send ether to any address. This address can be a contract, or there is no fallback function. ([Github Link](#))

```

// Contract destructor
function destroy() public noReentrancy onlyOwner {
    (bool sent, ) = msg.sender.call{value: balance}("");
    require(sent, "Failed to send all Ether");
    // selfdestruct(payable(owner));
}

```


Function Modifiers: Function Modifiers can be used to reduce the code for checking the apartment and make the code more concise. It can also be used to modify the code after return. If the require in Function Modifiers is established, it will be executed in the next line, and _ means to use the function of the modifier. ([Github Link](#))

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

modifier validAddress(address _addr) {
    require(_addr != address(0), "Not valid address");
    _;
}

modifier noReentrancy() {
    require(!locked, "No reentrancy");
    locked = true;
    _;
    locked = false;
}
```

Contract Inheritance: If I have two contracts, one of which uses most of the function code of the other, but there are some parts that need to be customized, I can use Inheritance to implement. ([Github Link](#))

```
contract Callee {
    .
    .
    .
}
```

```
contract Caller is Callee {
    .
    .
    .
}
```

Events: When an event is sent (called), it triggers the parameter to be stored in the transaction's log (a special data structure on the blockchain). These logs are associated with the address of the contract and recorded to the blockchain. ([Github Link](#))

```
event DepositLog(address sender, uint256 value);
event WithdrawLog(address sender, uint256 value);
```

Calling Other Contracts: A kind of method to call another contract. ([Github Link](#))

```
function withdraw_myself_and_callee(
    uint amount,
    address payable destAddr,
    Callee _callee
) public onlyOwner {
    require(msg.sender == owner, "Only owner can withdraw");
    require(amount <= balance, "Insufficient funds");

    (bool success, ) = destAddr.call{gas: 1000000, value:amount}("");
    balance -= amount;
    require(success, "Failed to send Ether");

    _callee.withdrawMyself(amount, destAddr);
}
```