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

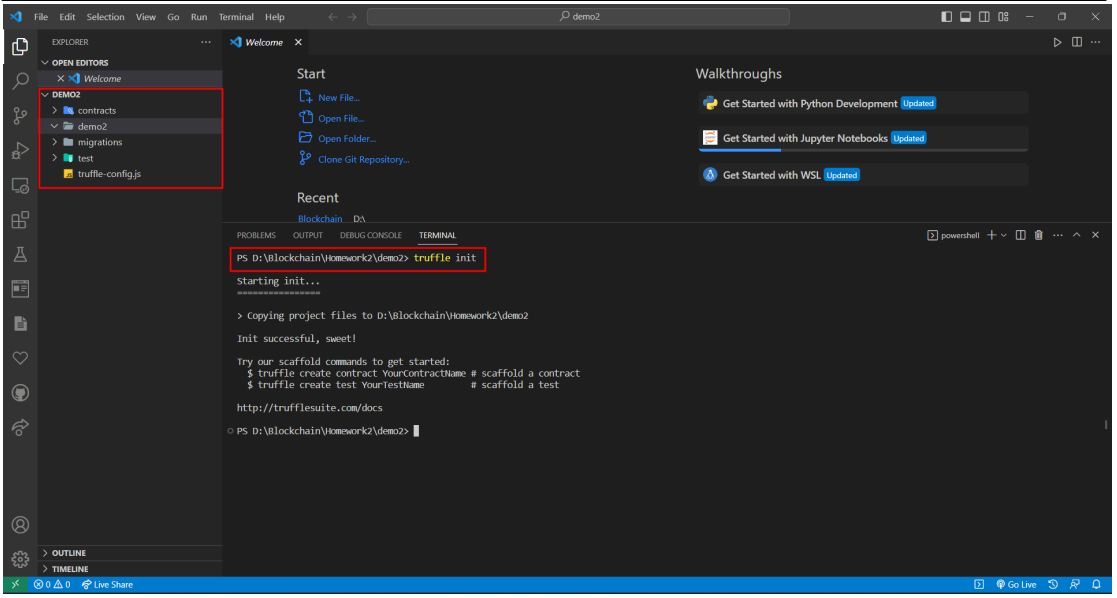
```
$ npm install -g truffle
$ npm install @truffle/hdwallet-provider
$ npm install dotenv
```

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

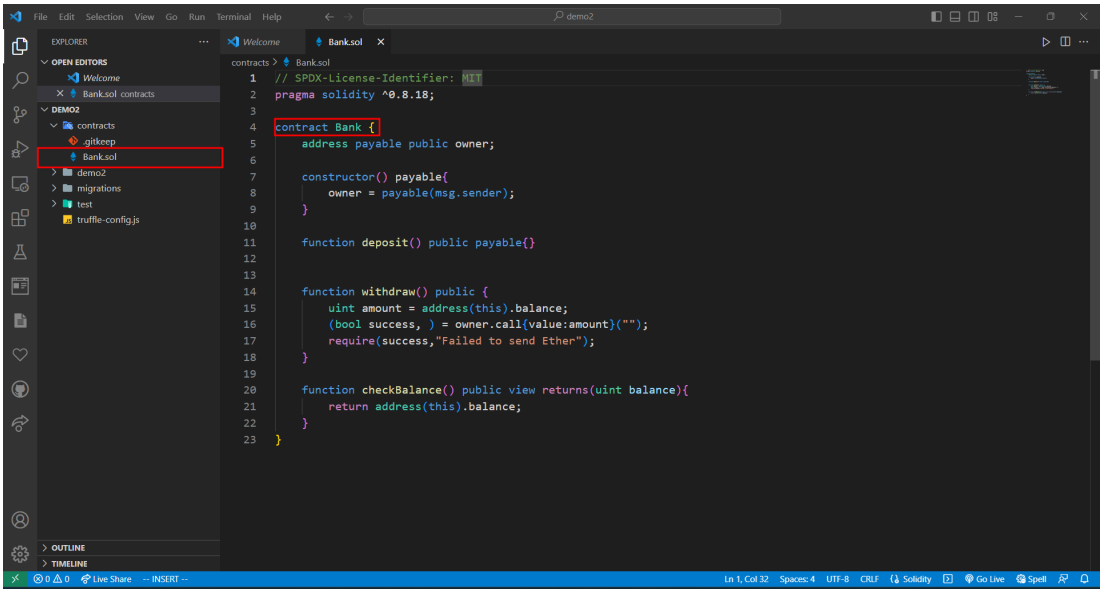
```
$ Get-ExecutionPolicy
$ Set-ExecutionPolicy RemoteSigned
```

Step 4: initialize project

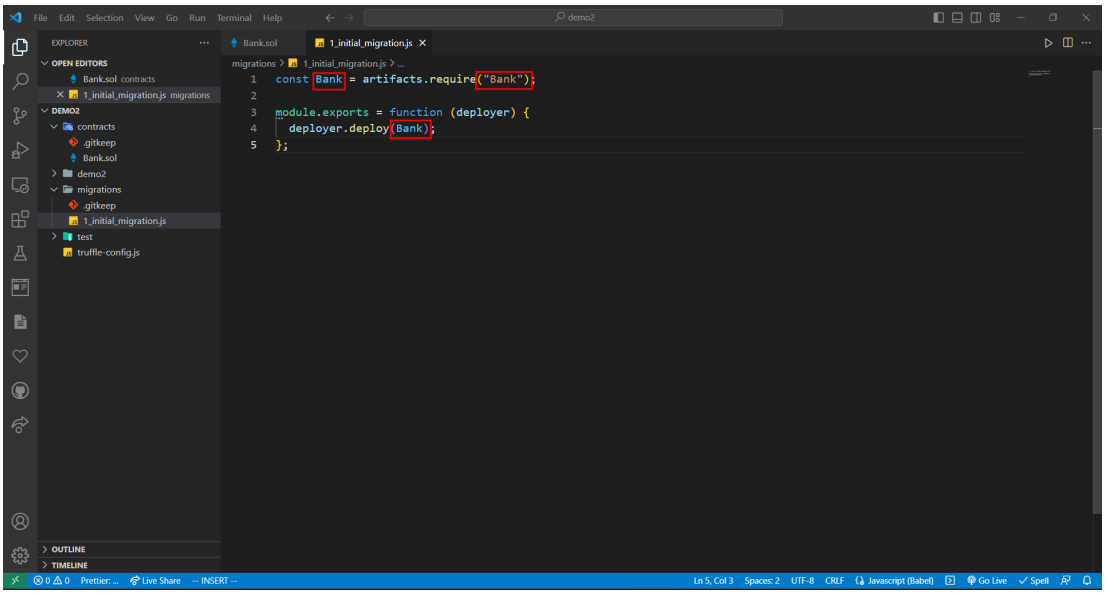
```
$ truffle init
```



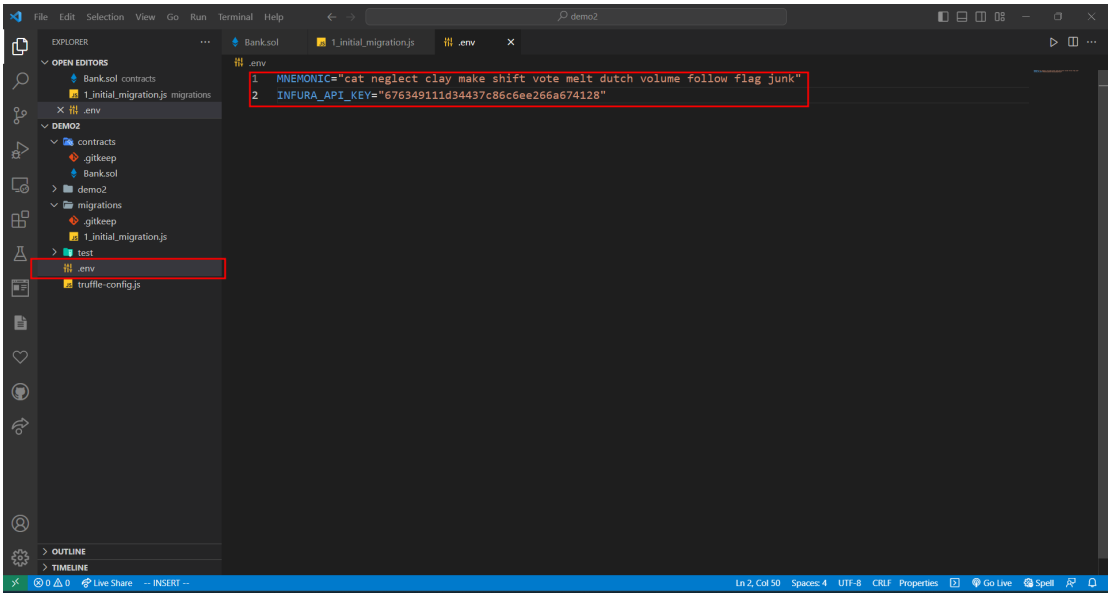
Step 5: Add a smart contract under the ./contract folder, which is a .sol file. Remember that the file name must be consistent with the name of the smart contract



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



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.

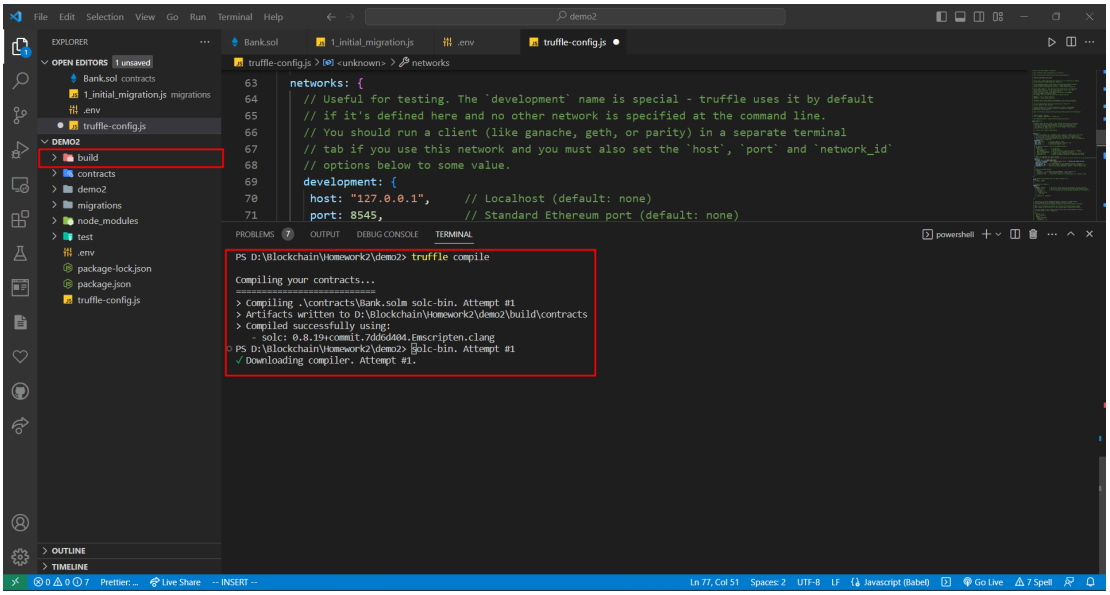
```
development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "5777",
},
```

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

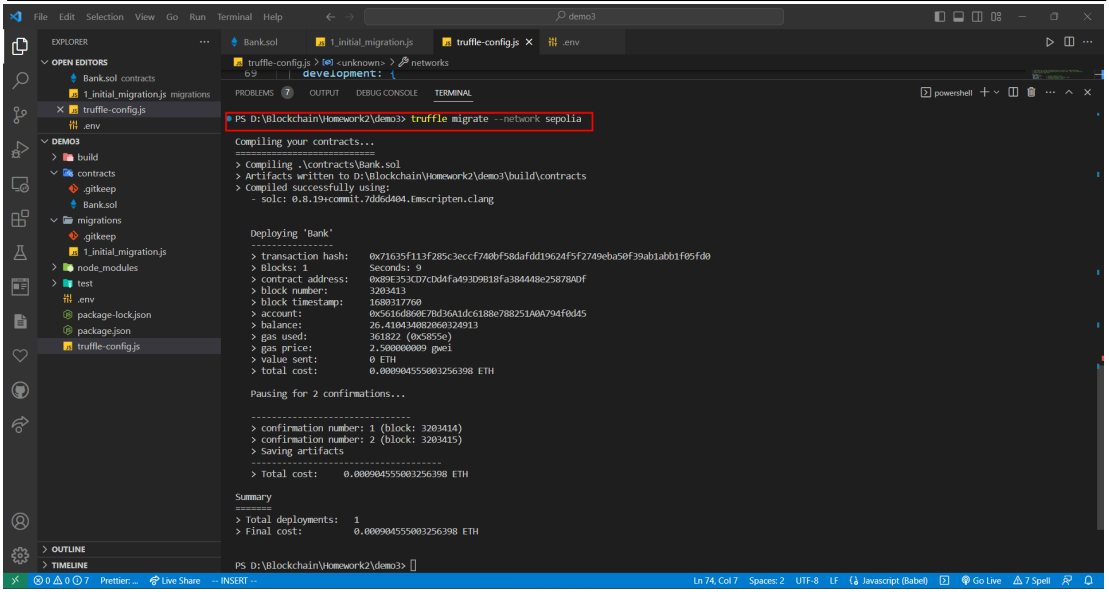
```
$ truffle compile
```



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
```



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
```

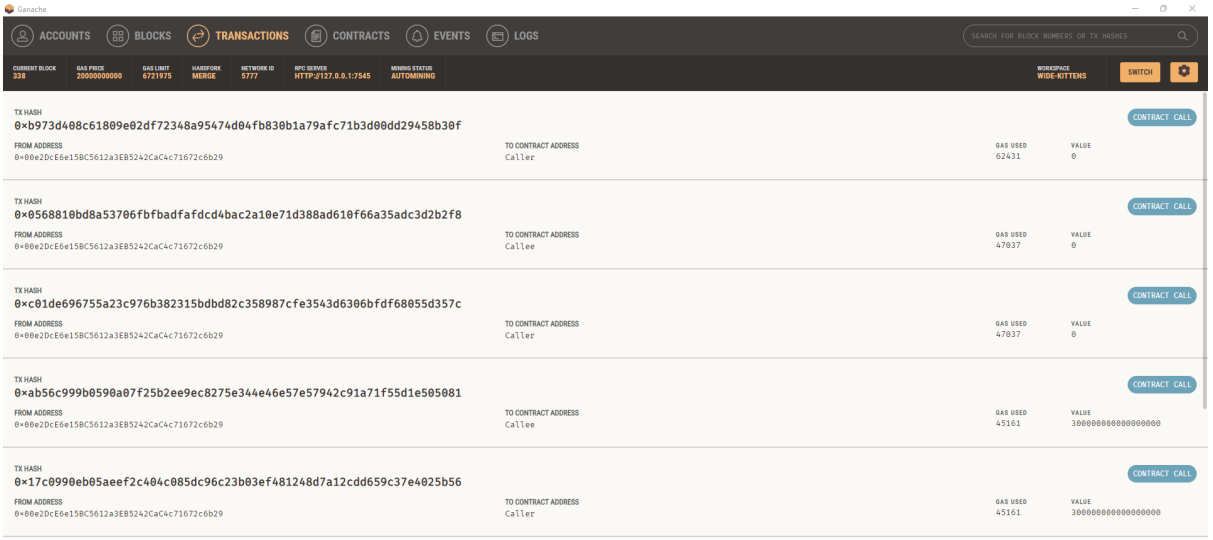
The balance: 0.2 ether

==========

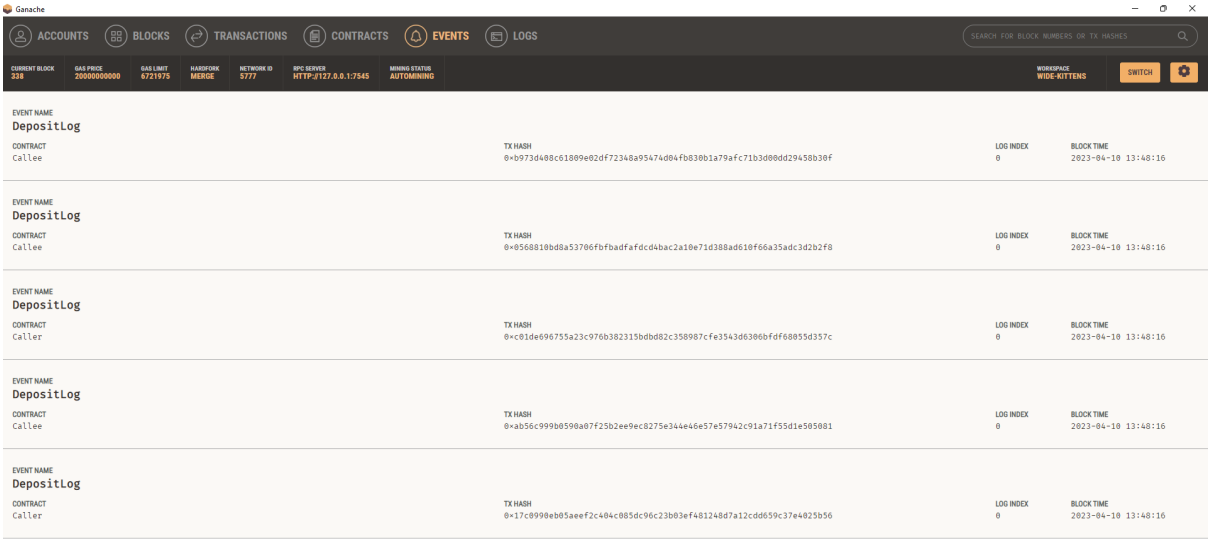
Use the smart contract 'caller' to call the smart contract 'callee's' function withdraw_myself to Withdraw 0.1 ether to 0x00e2DcE6e15BC5612a3EB5242CaC4c71672c6b29
<Caller>
The balance: 0.1 ether

<Callee>
The balance: 0.1 ether

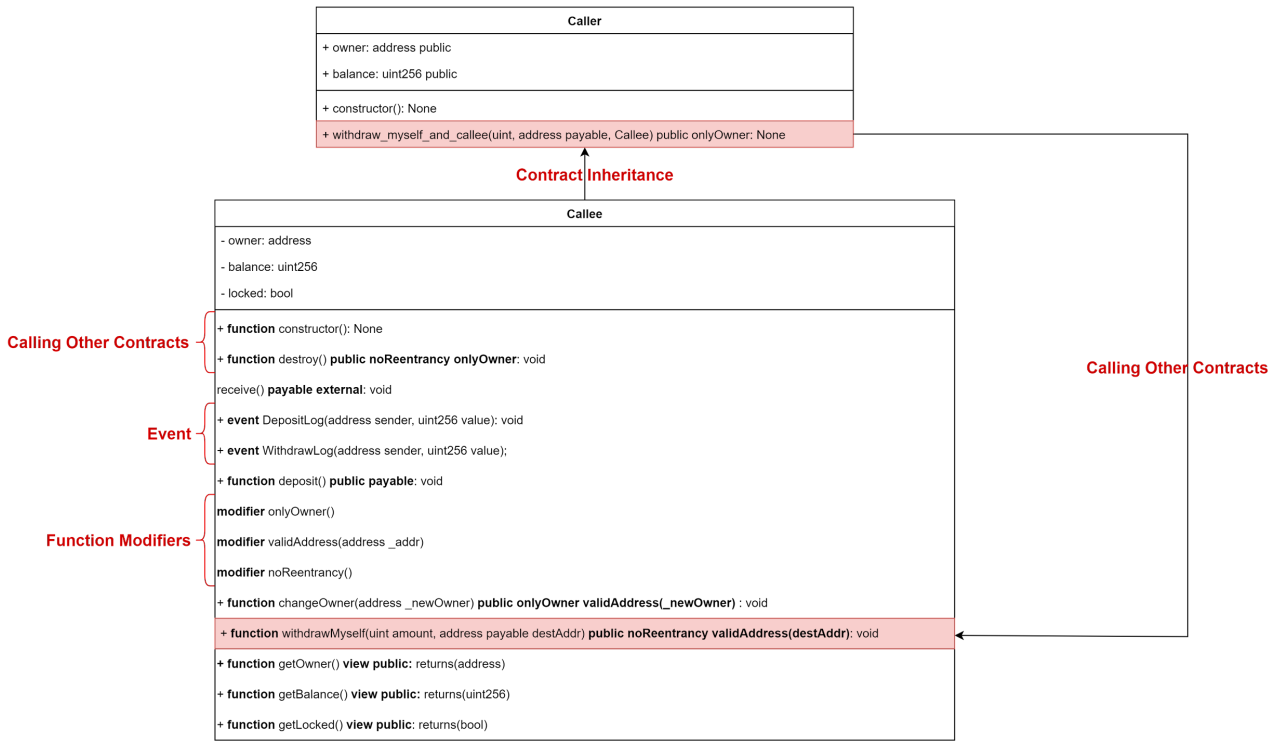
These are the tractions for the run.js.



These are the events for the run.js.



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);
    }
```