Ethereum
Blockchain Development
Week 17

Introduction

The intention of this lab is to look at functions and how variables are stored. All the exercises are completed in Remix IDE.

Code Completion

Writing code in a new language can be a steep learning curve. The approach here is to provide some code with underscores (_) that you are required to complete. These underscores are there to help you. By completing these exercises you will be improving your skills and knowledge of Solidity.

Each exercise starts on a new page. The red numbers in the right-hand margin are estimated minutes you should spend on each exercise. The length of the exercise depends on the bandwidth available to you. If the bandwidth availble to you is low then it is advised that you complete §**??** first.

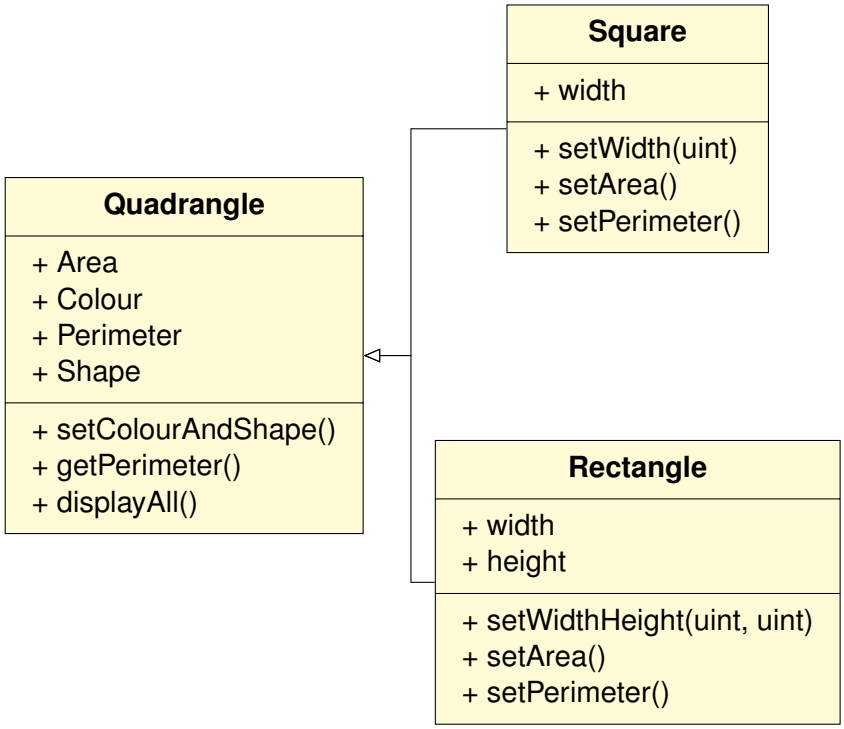

Figure 1: The class diagram for shapes

1 Class Inheritance

25–30

Examine the class diagram in Fig. 1. Write 3 classes as contracts and execute in a fourth class, `Test`. In the class instance of `Test` provide an object rectangle and square and calculate the areas and perimeters of each shape using the functions indicated in the class diagram in Fig.1. Finally, add an event in the `Quadrangle` class and emit this event in the `displayAll` function, then make a call to this function from `Test`.

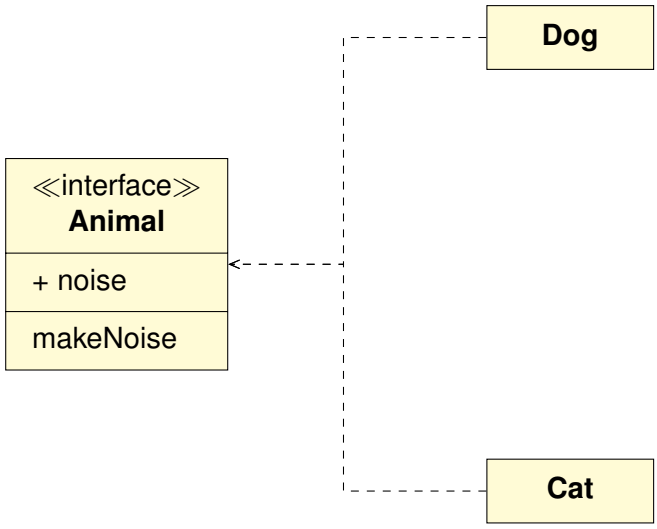

Figure 2: The class diagram for Animal Noises. This illustrates the use of an interface.

2 Interface: Introduction

15-20

Examine the class diagram in Fig. 2. Open the remix editor and add a new file. Write in solidity the interface as a contract. Then implement the two other classes that use the interface. The Dog should return a noise “woof” and the cat contract returns “miaow”. Write this and implement in the remix editor.

3 Private Networks

3.1 Download VM

5

The VM is downloadable from smerf server[1]. This VM has the capability to setup a private network. Download the VM on your personal laptop, and install using Oracle Virtual Box Manager. Further information about VM and their implementation can be found in Learning Week 1. The passphase for this VM is “4125”.

3.2 Private Network

5

Create a private network for ethereum using Ganache and desktop remix-ide by opening a file manager in the GUI. This should reveal a “home” directory. Click on the “home” directory and you should find the “Ganache” and “Remix” directories. In each of these there is an `*.AppImage` file, double-click on each of these and wait - be patient this takes some time to set up - and the desktop remix-ide and Ganache should appear. For Ganache click on the "Quickstart" button. Familiarise yourself with the Ganache environment.

3.3 Contract

25–30

Write a contract that will send money from one account to another on Ganache. Implement and run this contract using the `send` and `transfer` methods. Create an event that emits the amount being transferred and the balance of the sender. Test the functions and what happens when the amount sent exceeds the account balance? Write a function that will check the funds prior to the transaction, and only proceed with the contract if there is sufficient funds in the account. Use events to emit an appropriate message.

To help you with this task complete the code in Fig. **??** on the next page.

Compile, deploy and test this contract.

[1] The time spent on this exercise depends on the bandwidth available to you.

Code Completion

```sol
//SPDX-License-Identifier: GPL2.0
pragma solidity ^0.8.0;
contract account{
    _______ (address __ uint) accountBalance;

    event sentMsg( address, address, uint, bool);
    event transferMsg( address, address, uint, string);

    function setAccount() public {
        accountBalance[__________] = msg.sender.balance;
    }

    function setAccount(address _account) public {
        accountBalance[_account] = ________.balance;
    }

    function getAddress() public ____ returns(address){
        return msg.sender;
    }

    function getBalance() public ____ returns(uint){
        return (accountBalance[msg.sender]);
    }

    function getBalanace(address _account) public ____ returns(uint){
        return (_______________[________]);
    }

    function sendEther (address _______ _to) public _______ {
        bool sent;
        uint amount = ___._____;
        if (amount < accountBalance[___._______]) {
            sent = ___.send(amount);
            if (sent){
                accountBalance[___._______]-=amount;
                accountBalance[___]+=amount;
            }
        }
        emit sentMsg(___._______, _to, amount, sent);
    }

    function transferEther(address _______ _to) public _______ {
        uint amount = ___._____;
        string memory eventMsg = "Transferred Failed";
        if (amount < accountBalance[___._______]) {
            ___.transfer(amount);
            accountBalance[___._______]-=amount;
            accountBalance[___]+=amount;
            eventMsg = "Transfer Completed";
        }
        emit transferMsg(___._______, _to, amount, eventMsg);
    }
}
```

Figure 3: Listing for `ex3.sol`, account for an Ethereum contract

Code Completion

```
1 //SPDX-License-Identifier: GPL2.0
2 pragma solidity ^0.8.0;
3
4 _________ SimpleToken{
5     function createAccount(address _account) ________;
6     function unfreezeAccount(address _account) ________;
7     function freezeAccount(address _account) ________;
8     function transfer(address _to, uint _amount) ________;
9 }
```

Figure 4: Solidity listing for an interface for a SimpleCoin application

4 Building a Contract from an Interface

25-30

Open the Ethereum VM and start up Ganache and Remix. Ensure that this exercise is deployed to Ganache, as shown in the previous exercise.

Complete the code in Fig.3 that shows a listing of an interface for a `SimpleToken` and the contract.

There are four functions for this simple application: `freezeAccount, unfreezeAccount, transfer, & createAccount`.

Create a contract as shown in fig. 4, which inherits the interface `SimpleToken`. Develop, compile and implement the following functions:

freezeAccount : sets boolean value to true for the provided address in the mapping. This prevents the transfer of coins.

unfreezeAccount : set a boolean value to false for the provided address in the mapping. This allows the transfer of coins.

transfer : transfers token (not ethereum) and debits and credits accounts of sender and receiver, respectively.

createAccount : similar to a constructor. Adds an account to the hashmap, with a default balance set to the `MAX_LIMIT` and adds an account to the `freezeMap` mapping and sets the default to `false`.

Code Completion

```
contract MDXToken is SimpleToken{
    _______ (address __ uint) public balanceMap;
    _______ (address __ bool) public frozenMap;
    uint ________ MAX_LIMIT = 1000000000;

    _____ transferMsg(address, uint, string);

    ________ createAccount (address _account) public ________ {
        __________[_account] = MAX_LIMIT;
        _________[_account] = false;
    }

    ________ transfer(address _to, uint _amount) public ________ {
        string ______ eventMsg="Transfer failed";
        if( (!frozenMap[__________]) && (_amount < balanceMap[__________])){
            balanceMap[msg.sender] __ _amount;
            balanceMap[_to] __ _amount;
            eventMsg="Transfer completed";
        }
        ____ transferMsg(_to, _amount, ________);
    }

    ________ unfreezeAccount(address _to) public ________{
        _________[_to] = false;
    }

    ________ freezeAccount(address _to) public ________ {
        _________[_to] = true;
    }
}
```

Figure 5: Solidity listing for a Contract using the interface for a SimpleCoin application

5 Reading

Read chapters 6 & 7 in [1]

References

[1] Ritesh Modi. *Solidity Programming Essentials*. Packt, 2018.