Introduction to Blockchain and Smart Contracts

Arda Harman – Group 22

In my paper, I will be covering 5 exercises in total…

Also, at the very end I will share all the screenshots related to my work

The essence of these exercises are for:

* Understanding what does contracts do
* How do they store client information
* Different ways to store clients (mapping vs array)

Before starting to the exercises, since we will always be going to talk about “contracts” we should talk about them.

What is a contract?

A contract is a middle-man which handles Money transfer between 2 people

* These 2 people are people which has money in their accounts (their accounts are in the contract)
* So, it is better to say “money transfer between 2 accounts” (rather than 2 people)

A contract is really similiar to a bank

* Users have accounts in a bank
* When users transfer money, bank makes this transfer

A contract owns all the money which accounts have

And, when users do transfers, it will ensure everything is done correctly and no error occurs

How “contract ensures we don’t get any errors” is by the code we write

This is why we call it “smart” contract

Exercise 1 – Implementing PiggyArray

We use:

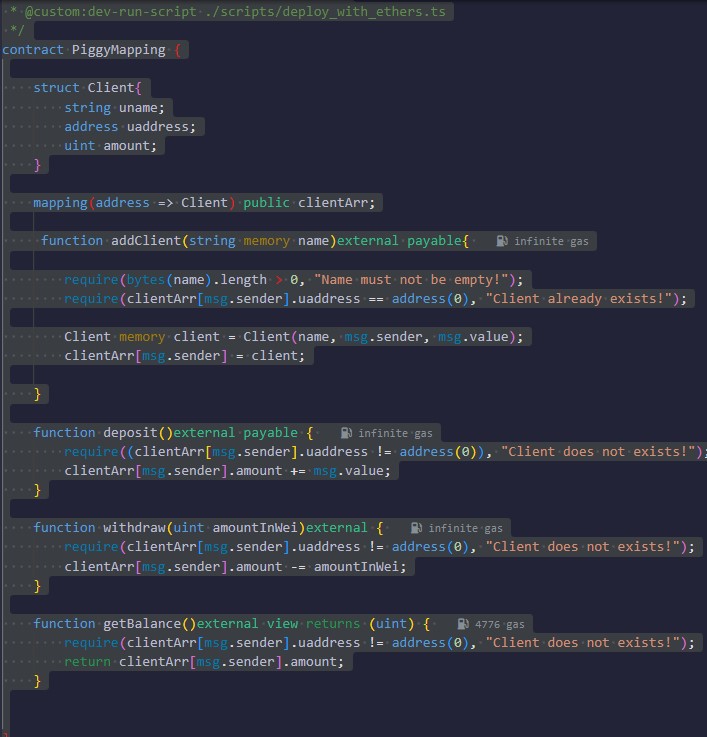
* Client struct, to create “clients” to store them
* clientArr -> to store clients and relative information
* addClient -> for adding clients to the clientArr
  + We check clients doesn’t already exist in our array and we check the name of client isn’t empty
* Deposit -> to put money/deposit to the contract
  + We check that, the user who calls this deposit function really exists in our contract
* Withdraw -> to take money back from contract (taking money which exists in our account)
  + Again, we check that user exists
* getBalance -> Returns the balance of the user (of course if a user like that exists)
* checkAccountExists & searchForAccountIndex-> Check whether given client exists at clientArr



Exercise 2 – Implementing piggyMapping

It has the same functions as piggyArray so it is doing the same things  
Only difference between them is that, this uses mapping while the first one using arrays

* A search in an array is sequential, meaning we have to traverse array to reach the element
* A search in mapping is iimmediate. Because it works with key -> value logic



Exercise 3 – Comparing addClient results

For both exercises, I gave:

* 10 ETH for Huey
* 20 ETH for Dewey
* 30 ETH for Louie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| addClient | | | Execution cost | |
| Name | Amount | Address | PiggyArray | PiggyMapping |
| Huey | 10.000 | 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 | 90.508 | 68.429 |
| Dewey | 20.000 | 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB | 78.462 | 68.429 |
| Louie | 30.000 | 0x617F2E2fD72FD9D5503197092aC168c91465E7f2 | 80.989 | 68.429 |

* PiggyArray results are always bigger than PiggyMapping because piggyArray searches for clients until end of the array. But on the other hand, piggyMapping always gives the key and it can understand whether a user exists or not in O(1)
* PiggyMapping results are always same because checking user is done with trying to giving a key… meanwhile it is always different in piggyArray because size of the array is client
  + Array had different amounts of clients when checked for the user

Exercise 4 – Comparing getBalance results

* I gave 40 ETH to Sylvester

|  |  |  |  |
| --- | --- | --- | --- |
| getBalance | | Execution cost | |
| Name | Address | piggyArray | piggyMapping |
| Huey | 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 | 7743 | 4766 |
| Dewey | 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB | 13.851 | 4766 |
| Louie | 0x617F2E2fD72FD9D5503197092aC168c91465E7f2 | 16.905 | 4766 |
| Sylvester | 0x27C67393E8d74177dDF009AC37DF9199d60d2D45 | 23.013 | 4766 |

* PiggyArray is always bigger because it has to traverse to find the user. Contrary to that, piggyMapping always finds user in O(1)
* Also, what I noticed is, in piggyArray,as users tend to have more in their account, the cost of the getBalance is getting bigger

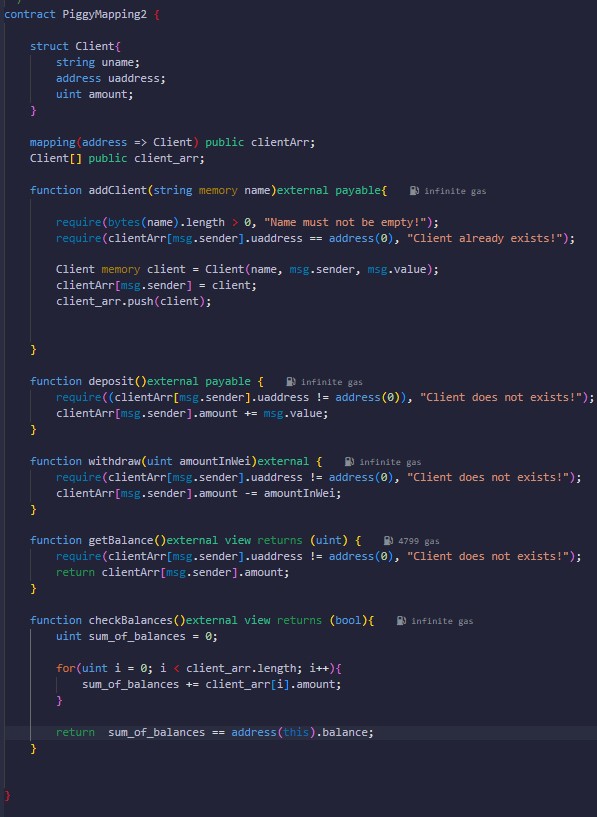
Exercise 5 – Adding “checkBalances to PiggyMapping2”

It is the exact same contract except now we have added a “client\_arr” array and a “checkBalance” function

Whatever element a clientArr has, client\_arr also has it

* Because, when we add an element to clientArr we also add to clien-arr at “addClient”

At “checkBalance” function, we just traverse for the array

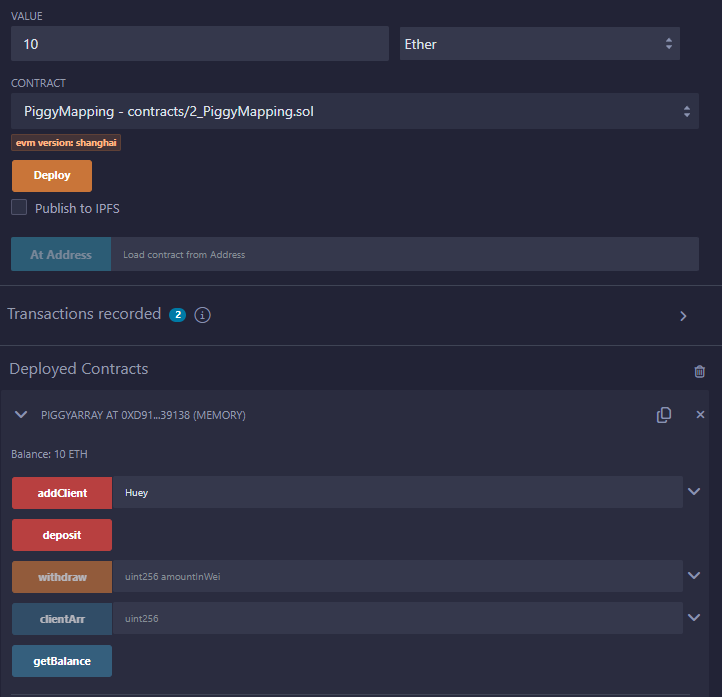


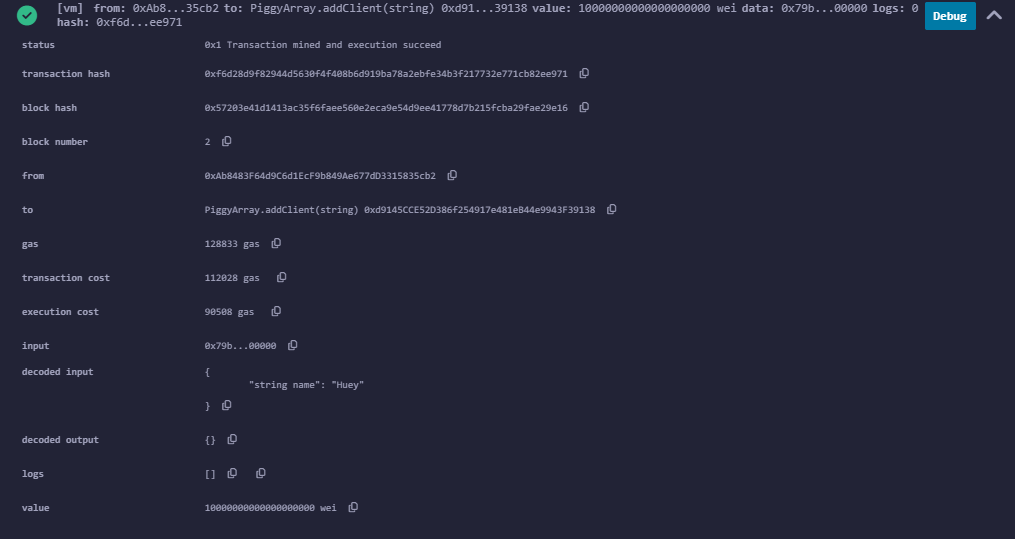
ALL THE NECESARY SCREENSHOTS FOR EXERCISES

EXERCISE 3

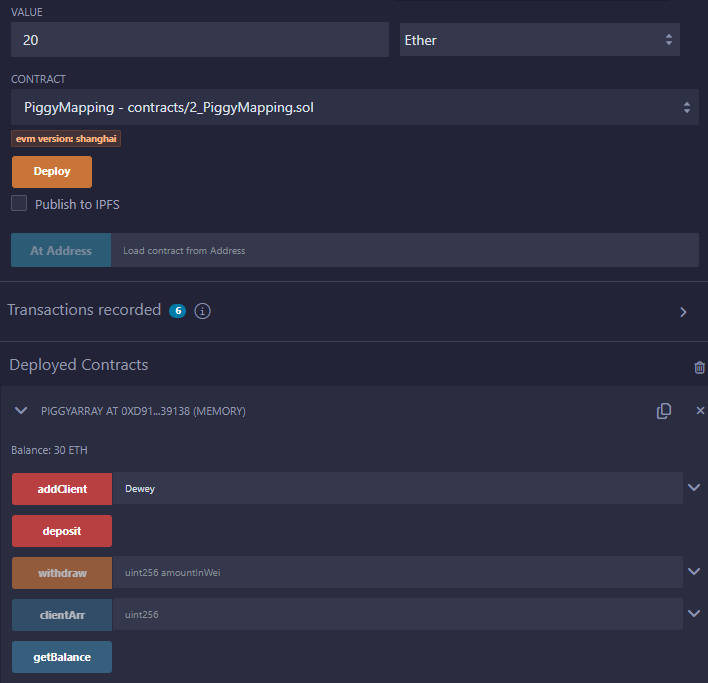
piggyArray – addingAccount

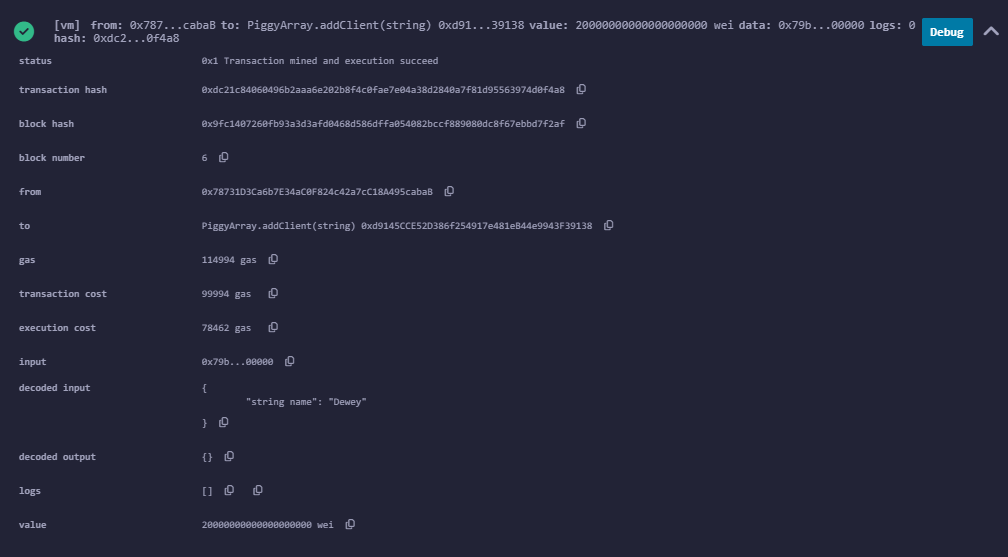
* Huey



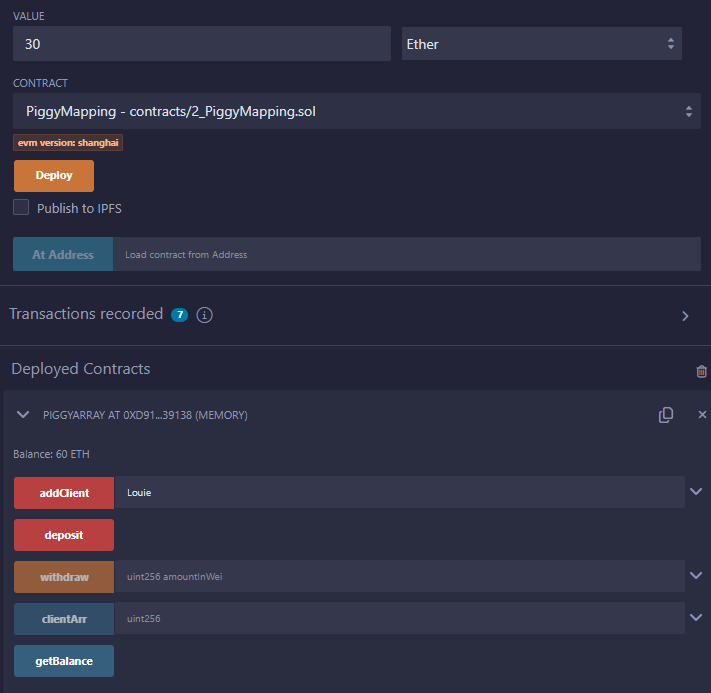


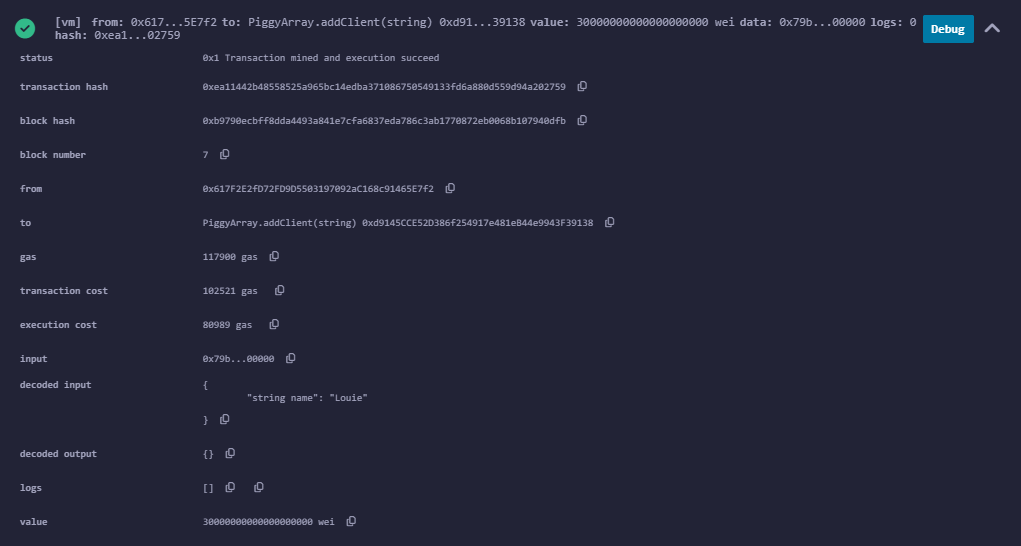
* Dewey





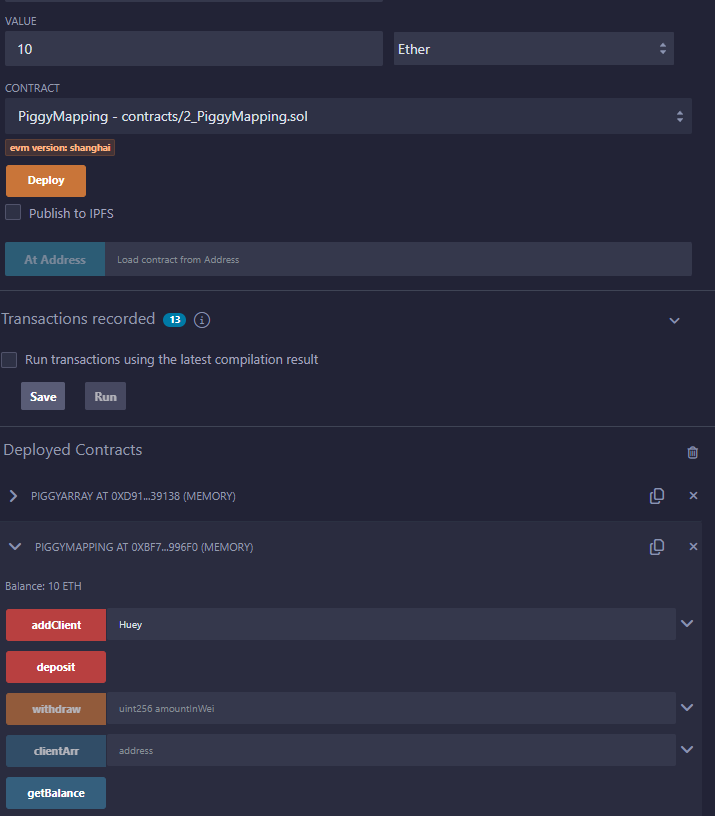
* Louie

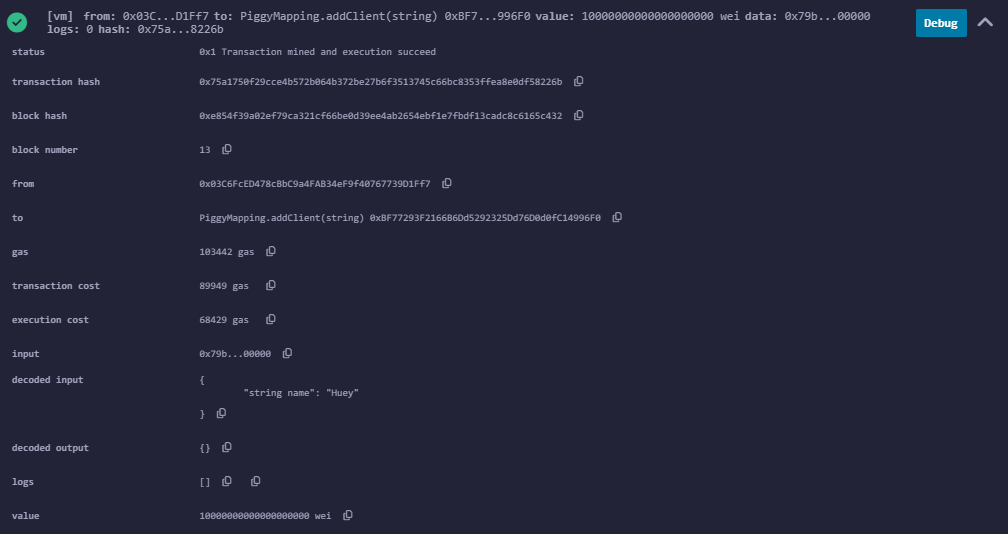




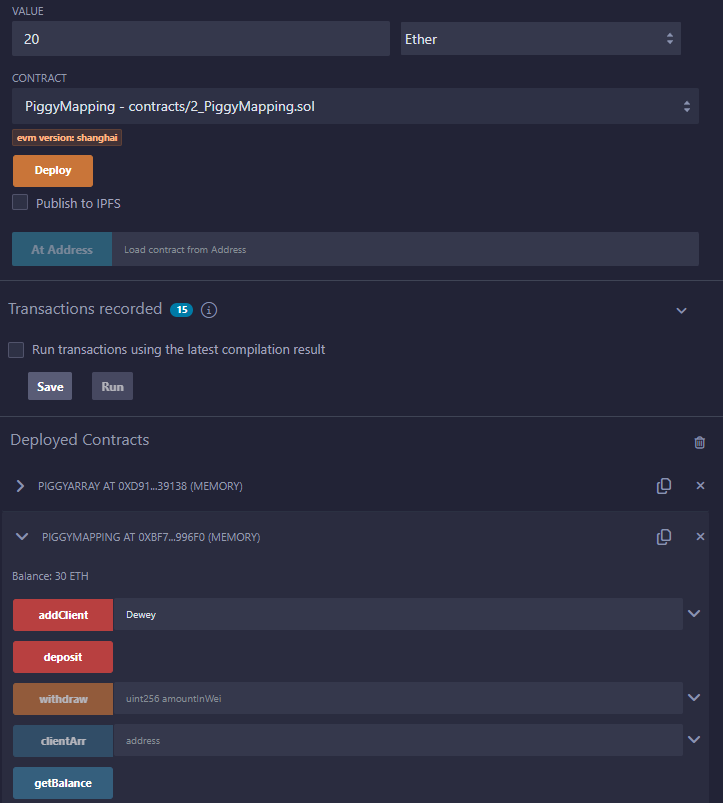
piggyMapping- addingAccount

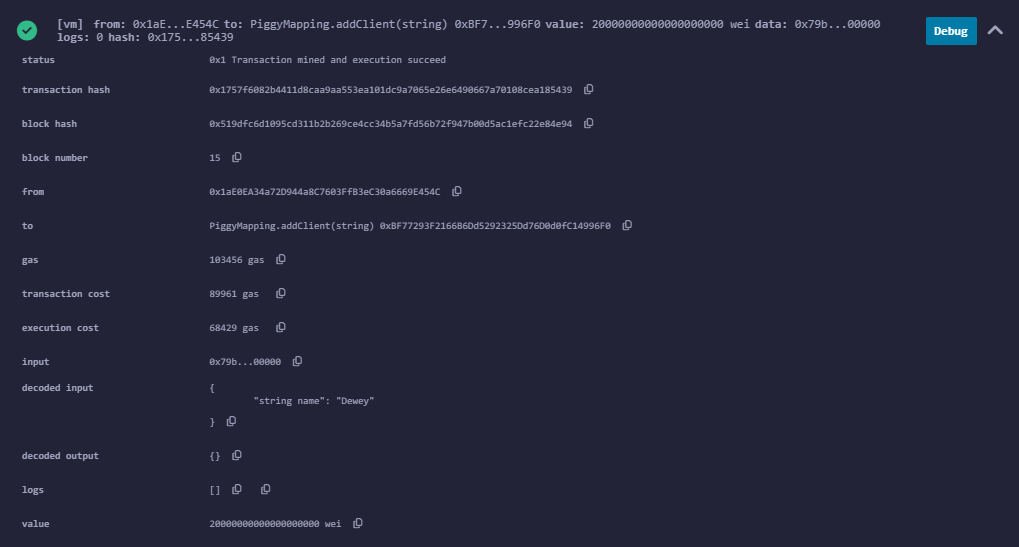
* Huey



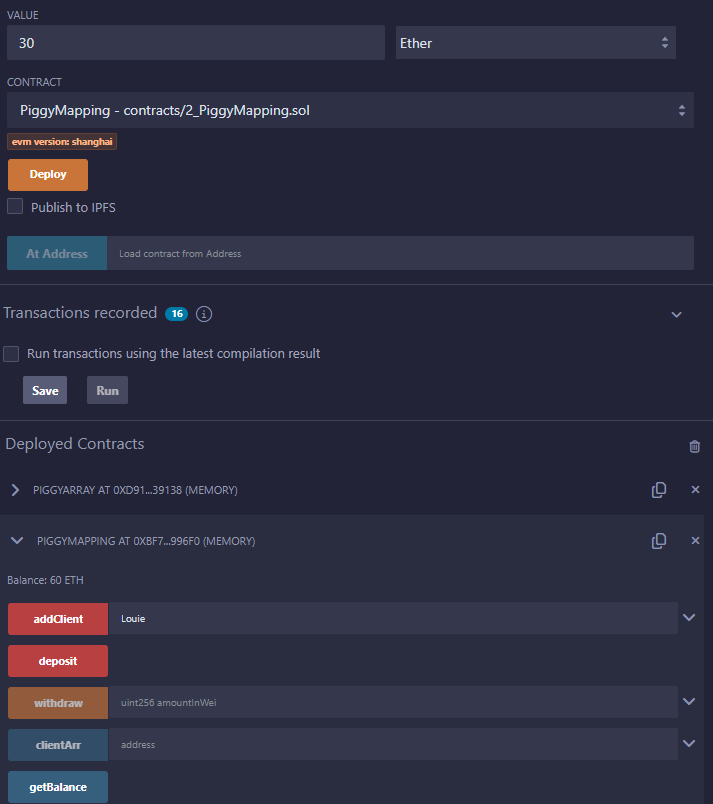


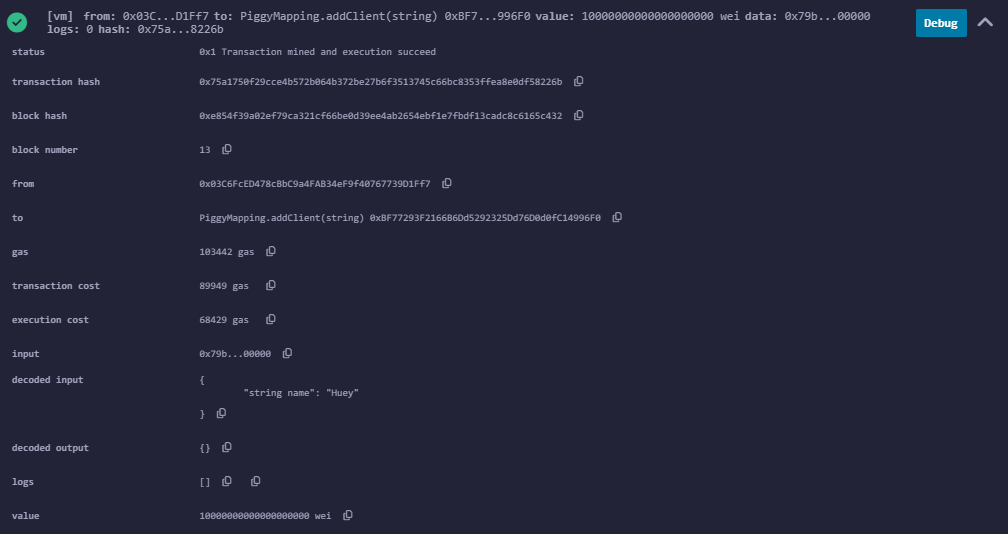
* Dewey





* Louie

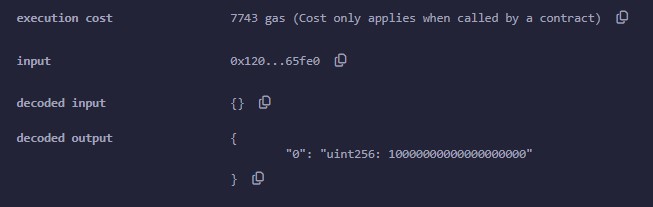




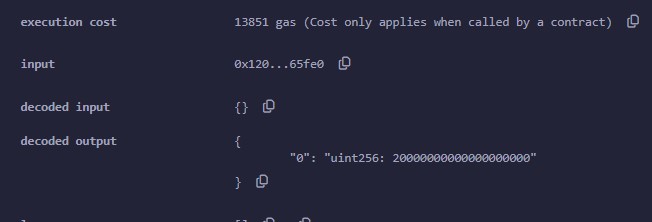
EXERCISE 4

piggyArray

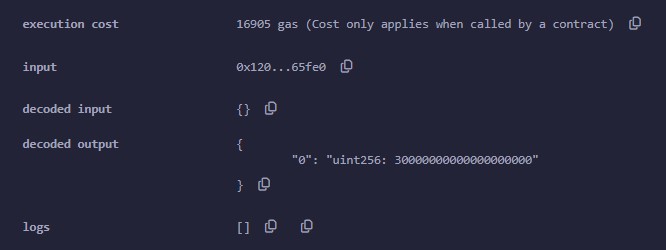
* Huey

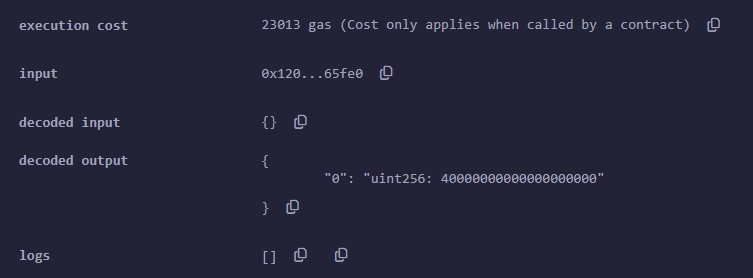


* Dewey



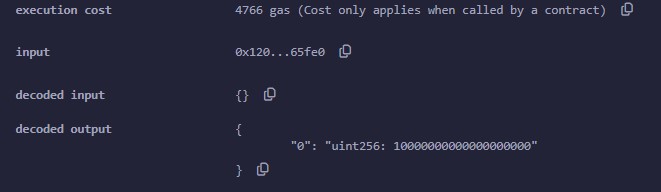
* Louie



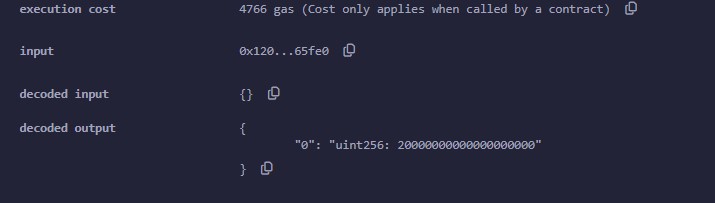
* Slyvester

piggyMapping

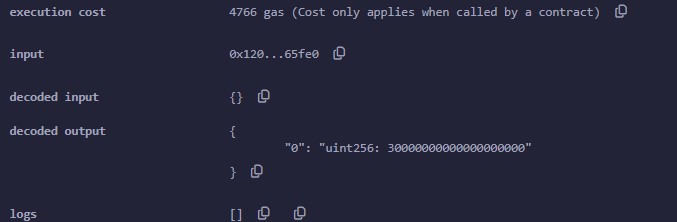
* Huey



* Dewey



* Louie



* Slyvester

