Introducción a la tecnología blockchain y smart contracts

Práctica 3

Javier Mulero Martín y Jorge del Valle Vázquez

19 de mayo de 2022

En esta práctica vamos a practicar el almacenamiento de datos con el uso de arrays y mappings en Solidity.

Ejercicios 1 y 2

Los **ejercicios 1 y 2** consistían en programar los contratos

- `PiggyArray.sol` (en anexo A)
- `PiggyMapping.sol` (en anexo B)

con las funciones especificadas, y con arrays y mappings, respectivamente.

Ejercicio 3

En este ejercicio nos pedían observar el consumo de gas al utilizar la función del contrato `addClient` con distintos usuarios y cuentas. Las direcciones de las cuentas que vamos a utilizar durante esta práctica se muestran en la tabla 1.

Name	Address
Juanito	0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB
Jorgito	0x617F2E2fD72FD9D5503197092aC168c91465E7f2
Jaimito	0x17F6AD8Ef982297579C203069C1DbfFE4348c372
Silvestre	0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678

Cuadro 1: *address* de cada usuario

Explica los resultados que has obtenido respecto al código que has programado.

Primero, observamos que en PiggyArray, el primer usuario que se une que es Juanito tiene un coste de ejecución más alto que el resto. Esto se debe a tener que inicializar el array e introducir el primer elemento. Después, vemos como Jorgito cuesta menos, ya

addClient		Execution cost	
Name	**Amount**	**PiggyArray**	**PiggyMapping**
Juanito	10.000	111560	67148
Jorgito	20.000	96990	67148
Jaimito	30.000	99520	67148

Cuadro 2: Costes de ejecutar `addClient` con los distintos usuarios y contratos

que el array estaba inicializado, y Jaimito cuesta más que Jorgito, debido a que tiene que recorrer el array para comprobar que no está incluido en la hucha y luego añadirse.

En cambio, en PiggyMapping, todos los costes son iguales. Esto se debe a que los mapping en Solidity acceden a las "keys" de manera directa, obteniendo los datos que hay en esa posición de memoria.

Ejercicio 4

En este ejercicio, había que observar el coste de ejecución de la función `getBalance` para los distintos usuarios y contratos.

getBalance	Execution cost	
Name	**PiggyArray**	**PiggyMapping**
Juanito	28166	25716
Jaimito	33226	25716
Silvestre	31230	23568

Cuadro 3: Costes de ejecutar la función `getBalance` para los distintos usuarios y contratos

Explica los resultados obtenidos respecto al código que has programado. ¿Qué piensas que ocurriría si tuvieras 100 clientes registrados en las huchas? ¿Y si tuvieras 1000 clientes?

En el caso de PiggyArray, el contrato recorre todos los clientes hasta encontrar al que ha ejecutado la función `getBalance`. Como Juanito está antes que Jaimito, su coste de ejecución es menor. En el caso de Silvestre, se recorre todo el array, y al no estar, el `require` devuelve parte del gas no consumido y no ejecuta el `return`, por eso cuesta un poco menos que Jaimito, que es el último del array.

En PiggyMapping, el coste es igual cuando el usuario está en la hucha, y cuando no se encuentra es menor, debido a que no ejecuta la instrucción `return`.

Por lo tanto, si tuviéramos 100 clientes, en el caso PiggyArray habría que recorrer hasta encontrarlo, y el caso peor es que no esté el usuario en la hucha y tenga que recorrerse

todos, incrementando mucho el coste de ejecución. En el caso de 1000 clientes, sería un coste mucho mayor. Por ello, el caso de PiggyMapping es mejor, ya que el acceso a las claves es directo, independientemente del número de usuarios registrados.

Ejercicio 5

Por último, en este ejercicio nos pedían añadir un array de `address` para poder recorrer el mapping, y así programar la función `checkBalances` que devolvería `true` si la suma de los saldos de todos los clientes de las huchas era igual al saldo del contrato, y `false` en caso contrario.

```
1 function checkBalances()external view returns (bool){
2         uint amount = 0;
3         for(uint i = 0; i < keys.length; ++i){
4             amount += clients[keys[i]].balance;
5         }
6         return amount == address(this).balance;
7     }
8 }
```

El resto del contrato se puede encontrar en el anexo C.

Anexos

A. PiggyArray.sol

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.7.0 <0.8.0;
3
4  contract PiggyArray {
5      struct Client {
6          address add;
7          string name;
8          uint balance;
9      }
10     Client[] clients;
11
12     function addClient(string memory name) external payable{
13         require(bytes(name).length > 0, "invalid name");
14         (bool b, ) = findClient(msg.sender);
15         require(!b, "already added");
16         Client memory c = Client(msg.sender,name,msg.value);
17         clients.push(c);
18     }
19     function findClient(address dir) internal view returns (bool, uint) {
20         for(uint i = 0; i < clients.length; ++i){
21             if(clients[i].add == dir)
22                 return (true, i);
23         }
24         return (false,clients.length);
25     }
26     function deposit() external payable{
27         (bool b, uint index) = findClient(msg.sender);
28         require(b, "not found");
29         clients[index].balance += msg.value;
30     }
31     function withdraw(uint amountInWei) external{
32         (bool b, uint index) = findClient(msg.sender);
33         require(b , "not found");
34         require(clients[index].balance > amountInWei, "Sorry Mr's, you
               cant withdraw that much amount of money");
35         payable(msg.sender).transfer(amountInWei);
36     }
37     function getBalance() external view returns (uint){
38         (bool b, uint index) = findClient(msg.sender);
39         require(b, "not found");
40         return clients[index].balance;
41     }
42 }
```

B. PiggyMapping.sol

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.7.0 <0.8.0;
3
4  contract PiggyMapping {
5
6      struct Client {
7          string name;
8          uint balance;
9      }
10
11     mapping (address=> Client) clients;
12
13     function addClient(string memory name) external payable{
14         require(bytes(name).length > 0, "invalid name");
15         require(bytes(clients[msg.sender].name).length == 0, "already
               added");
16         Client memory c = Client(name,msg.value);
17         clients[msg.sender] = c;
18
19     }
20     function deposit() external payable{
21         require(bytes(clients[msg.sender].name).length > 0 ,"not found");
22         clients[msg.sender].balance += msg.value;
23     }
24
25     function withdraw(uint amountInWei) external{
26         require(bytes(clients[msg.sender].name).length > 0 , "not found");
27         require(clients [msg.sender].balance > amountInWei, "Sorry Mr's,
               you cant withdraw that much amount of money");
28         payable(msg.sender).transfer(amountInWei);
29     }
30
31     function getBalance() external view returns (uint){
32         require(bytes(clients[msg.sender].name).length > 0,"not found");
33         return clients[msg.sender].balance;
34     }
35
36 }
```

C. PiggyMapping2.sol

```
1
2  // SPDX-License-Identifier: GPL-3.0
3  pragma solidity >=0.7.0 <0.8.0;
4
5  contract PiggyMapping2 {
6
7      struct Client {
8          string name;
9          uint balance;
10     }
11
12     mapping (address=> Client) clients;
13     address[] keys;
14
15     function addClient(string memory name) external payable{
16         require(bytes(name).length > 0,"invalid name");
17         require(bytes(clients[msg.sender].name).length == 0,"already added
               ");
18         Client memory c = Client(name,msg.value);
19         clients[msg.sender] = c;
20         keys.push(msg.sender);
21     }
22     function deposit() external payable{
23         require(bytes(clients[msg.sender].name).length > 0 ,"not found");
24         clients[msg.sender].balance += msg.value;
25     }
26
27     function withdraw(uint amountInWei) external{
28         require(bytes(clients[msg.sender].name).length > 0 , "not found");
29         require(clients[msg.sender].balance > amountInWei, "Sorry Mr's,
               you cant withdraw that much amount of money");
30         payable(msg.sender).transfer(amountInWei);
31     }
32
33     function getBalance() external view returns (uint){
34         require(bytes(clients[msg.sender].name).length > 0,"not found");
35         return clients[msg.sender].balance;
36     }
37     function checkBalances()external view returns (bool){
38         uint amount = 0;
39         for(uint i = 0; i < keys.length; ++i){
40             amount += clients[keys[i]].balance;
41         }
42         return amount == address(this).balance;
43     }
44 }
```