PRACTICA 3 BLOCKCHAIN

Alejandro Ramírez y David Seijas

February 2022

1 Ejercicio 1

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.8.0;

contract piggyArray {

    struct infoClient{
        string name;
        uint balance;
        address addr;
    }

    infoClient[] clients;

    function addCliente(string memory name) external payable{
        require(bytes(name).length > 0, "Nombre vacio NO valido!");
        (bool exist, uint pos) = searchCliente(msg.sender);
        require(!exist, "El cliente ya existe");
        infoClient memory newClient = infoClient(name, msg.value,
            msg.sender);
        clients.push(newClient);
    }

    function searchCliente(address addr) internal view returns(bool
        , uint){
        bool exist = false;
        uint pos = 0;
        for (uint i = 0; i < clients.length && !exist; ++i){
            if(clients[i].addr == addr){
                exist = true;
                pos = i;
            }
        }

        return (exist, pos);
    }

    function deposit() external payable{
        (bool exist, uint pos) = searchCliente(msg.sender);
        require(exist, "No puedes depositar dinero porque no estas
            registrado");
        clients[pos].balance += msg.value;
```

```
    }

    function withdraw(uint amountInWei) external{
        (bool exist, uint pos) = searchCliente(msg.sender);
        require(exist, "No puedes retirar dinero porque no estas registrado");
        require(clients[pos].balance > amountInWei, "No tienes suficiente dinero en la hucha");
        clients[pos].balance -= amountInWei;
        payable(msg.sender).transfer(amountInWei);
    }

    function getBalance() external view returns (uint){
        (bool exist, uint pos) = searchCliente(msg.sender);
        require(exist, "No puedes consultar tu dinero porque no estas registrado");
        return clients[pos].balance;
    }
}
```

2 Ejercicio 2

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.8.0;

contract piggyMapping {

    struct infoClient{
        string name;
        uint balance;
    }

    mapping(address => infoClient) clients;

    function addCliente(string memory name) external payable{
        require(bytes(name).length > 0, "Nombre vacio NO valido!");
        require(bytes(clients[msg.sender].name).length == 0, "El cliente ya existe");
        infoClient memory newClient = infoClient(name, msg.value);
        clients[msg.sender] = newClient;
    }

    //Ahora no necesitamos searchClient por usar mapping

    function deposit() external payable{
        require(bytes(clients[msg.sender].name).length > 0, "No puedes depositar dinero porque no estas registrado");
        clients[msg.sender].balance += msg.value;
    }

    function withdraw(uint amountInWei) external{
        require(bytes(clients[msg.sender].name).length > 0, "No puedes retirar dinero porque no estas registrado");
        require(clients[msg.sender].balance > amountInWei, "No tienes suficiente dinero en la hucha");
```

```
30          clients[msg.sender].balance -= amountInWei;
31          payable(msg.sender).transfer(amountInWei);
32      }
33
34      function getBalance() external view returns (uint){
35          require(bytes(clients[msg.sender].name).length > 0, "No
                puedes consultar tu dinero porque no estas registrado")
                ;
36          return clients[msg.sender].balance;
37      }
38  }
```

3 Ejercicio 3

addClient			Execution cost	
Name	**Amount**	**Address**	**PiggyArray**	**PiggyMapping**
Juanito	10.000	0x617F2E2fD72FD9D5503197092aC168c91465E7f2	111532	67167
Jorgito	20.000	0x17F6AD8Ef982297579C203069C1DbfFE4348c372	96990	67167
Jaimito	30.000	0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678	99548	67167

En PiggyArray se observa que el primer acceso es más costoso porque hemos declarado un array sin tamaño y el push inicial tiene un coste más alto. Además, a partir del segundo cliente el coste experimenta un crecimiento lineal, ya que el array va creciendo y hacemos una búsqueda lineal.

Sin embargo, con los mappings ahorramos coste de ejecución y es siempre el mismo para todos los clientes. Esto se debe a que el acceso a un mapping es constante y no es necesario realizar ninguna búsqueda.

4 Ejercicio 4

getBalance		Execution cost	
Name	**Address**	**PiggyArray**	**PiggyMapping**
Juanito	0x617F2E2fD72FD9D5503197092aC168c91465E7f2	28419	25738
Jorgito	0x17F6AD8Ef982297579C203069C1DbfFE4348c372	30977	25738
Jaimito	0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678	33527	25738
Silvestre	0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7	31267	23600

Silvestre no estaba registrado como usuario, por lo que la llamada a getBalance con su cuenta ha dado error. Para los otros 3 usuarios sí que ha funcionado. Los resultados obtenidos siguen la línea del ejercicio 3.

En PiggyArray los primeros clientes tendrán menor coste que los últimos porque en la búsqueda se recorren menos elementos. Silvestre no está registrado, por lo que su búsqueda requiere recorrer todo el array. Como el require falla, en su getBalance() no se ejecuta el return y el coste es un poco menor que el de Jaimito (en su ejecución si se realiza el return). De nuevo en PiggyMapping el

coste es constante.

Por lo tanto, con 100 clientes el coste en PiggyArray aumenta mucho, aproximadamente 3000/cliente. Por ejemplo, con 1000 clientes, tratando de acceder al usuario más reciente, el coste sería $c \approx 2800 + 3000 * 1000 = 3.000.000$ gas. Por lo tanto, habría que poner un límite de gas de la transacción mayor para que se pueda realizar. En PiggyMapping el coste de la transacción seguiría siendo de 25738 gas.

5 Ejercicio 5

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.7.0 <0.8.0;
3
4  contract piggyMapping2 {
5
6      struct infoClient{
7          string name;
8          uint balance;
9      }
10
11     mapping(address => infoClient) clients;
12     address[] clientsArray;
13
14     function addCliente(string memory name) external payable{
15         require(bytes(name).length > 0, "Nombre vacio NO valido!");
16         require(bytes(clients[msg.sender].name).length == 0, "El
               cliente ya existe");
17         infoClient memory newClient = infoClient(name, msg.value);
18         clients[msg.sender] = newClient;
19         clientsArray.push(msg.sender);
20     }
21
22     //Ahora no necesitamos searchClient por usar mapping
23
24     function deposit() external payable{
25         require(bytes(clients[msg.sender].name).length > 0, "No
               puedes depositar dinero porque no estas registrado");
26         clients[msg.sender].balance += msg.value;
27     }
28
29     function withdraw(uint amountInWei) external{
30         require(bytes(clients[msg.sender].name).length > 0, "No
               puedes retirar dinero porque no estas registrado");
31         require(clients[msg.sender].balance > amountInWei, "No
               tienes suficiente dinero en la hucha");
32         clients[msg.sender].balance -= amountInWei;
33         payable(msg.sender).transfer(amountInWei);
34     }
35
36     function getBalance() external view returns (uint){
37         require(bytes(clients[msg.sender].name).length > 0, "No
               puedes consultar tu dinero porque no estas registrado")
               ;
```

```
38          return clients[msg.sender].balance;
39      }
40
41      function checkBalances() external view returns(bool){
42          uint totalBalance = 0;
43          for(uint i = 0; i < clientsArray.length; ++i){
44              totalBalance += clients[clientsArray[i]].balance;
45          }
46
47          return(address(this).balance == totalBalance);
48      }
49  }
```