

## Laboratorio 3:

### Ejercicio 1:

Implementación de una hucha multicliente con un array:

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

contract PiggyArray {

    struct Client {
        string _name;
        address _add;
        uint _amount;
    }

    Client[] public clients;

    function getAddress() internal view returns (uint) {    ⚠ infinite gas
        uint i = 0;
        while (i < clients.length && msg.sender != clients[i]._add)
            ++i;
        return i;
    }

    function addClient(string memory name) external payable {    ⚠ infinite gas
        require(bytes(name).length > 0 , "empty name");
        uint i = getAddress();
        require(i == clients.length, "client already exists");
        clients.push(Client({ _name : name, _add : msg.sender, _amount : msg.value }));
    }

    function deposit() external payable {    ⚠ infinite gas
        uint i = getAddress();
        require(i != clients.length, "address not found");
        clients[i]._amount += msg.value;
    }

    function withdraw(uint amountInWei) external {    ⚠ infinite gas
        uint i = getAddress();
        require(i != clients.length, "address not found");
        require(amountInWei <= clients[i]._amount, "not enough money in your account");
        clients[i]._amount -= amountInWei;
        payable(msg.sender).transfer(amountInWei);
    }

    function getBalance() external view returns (uint) {    ⚠ infinite gas
        uint i = getAddress();
        require(i != clients.length, "address not found");
        return clients[i]._amount;
    }

}
```

## Ejercicio 2:


Implementación de una hucha multicliente con mapping:


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


contract PiggyMapping {


    struct Client {
        string _name;
        uint _amount;
    }

    mapping (address => Client) public clients;

    function addClient(string memory name) external payable {  infinite gas
        require(bytes(name).length > 0, "empty name");
        require(bytes(clients[msg.sender]._name).length == 0, "client already exists");
        clients[msg.sender] = Client({ _name : name, _amount : msg.value });
    }

    function deposit() external payable {  22079 gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        clients[msg.sender]._amount += msg.value;
    }

    function withdraw(uint amountInWei) external {  infinite gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        require(amountInWei <= clients[msg.sender]._amount, "not enough money in your account");
        clients[msg.sender]._amount -= amountInWei;
        payable(msg.sender).transfer(amountInWei);
    }

    function getBalance() external view returns (uint) {  2052 gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        return clients[msg.sender]._amount;
    }

}
```

### Ejercicio 3:

addClient			Execution Cost	
Name	Amount	Address	PiggyArray	PiggyMapping
Juanito	10.000	0x5B38Da6a701c5685 45dCfcB03FcB875f56b eddC4	90001	45601
Jorgito	20.000	0xAb8483F64d9C6d1 EcF9b849Ae677dD33 15835cb2	75438	45601
Jaimito	30.000	0x4B20993Bc481177e c7E8f571ceCaE8A9e2 2C02db	77975	45601

La diferencia de los costes se debe a cómo se gestionan los datos en los dos contratos, es más costoso en la implementación PiggyArray porque depende de la cantidad de elementos que haya en el array ya que verificar que existe el cliente en la base de datos es lineal respecto al número de clientes. En el PiggyMapping es el mismo en los tres ya que acceder y modificar en un mapa tiene coste constante siendo más eficiente y dando, por lo tanto, menos coste de ejecución.

#### Ejercicio 4:

getBalance	Execution Cost
------------	----------------

Name	Address	PiggyArray	PiggyMapping
Juanito	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4	7196	4652
Jaimito	0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db	12270	4652
Silvestre	0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB	10197	2504

Al igual que en el ejercicio anterior, el PiggyArray es más costoso en ejecución por el coste de verificar que existe en la base de datos del contrato, se puede ver que en los costes de PiggyArray, Juanito es el que tiene menos coste ya que es el primero en el array, mientras que Jaimito es el que tiene más porque se encuentra en la última posición del mismo. Silvestre, que no está registrado como cliente tiene el coste de recorrer todo el array buscándolo, por eso es más costoso que Juanito pero menos que Jaimito ya que él también tiene el coste de verificar su balance.

En PiggyMapping se puede ver que siguen siendo constantes los costes de ejecución, teniendo menos Silvestre porque solo verifica si existe en el mapa, no comprueba el balance.

## Ejercicio 5:

PiggyMapping2 es igual que PiggyMapping solo con las siguientes modificaciones:

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

contract PiggyMapping2 {

    struct Client {
        string _name;
        uint _amount;
    }

    mapping (address => Client) public clients;

    address[] public adds;

    function addClient(string memory name) external payable {    Infinite gas
        require(bytes(name).length > 0, "empty name");
        require(bytes(clients[msg.sender]._name).length == 0, "client already exists");
        clients[msg.sender] = Client({ _name : name, _amount : msg.value });
        adds.push(msg.sender);
    }

    function deposit() external payable {    22079 gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        clients[msg.sender]._amount += msg.value;
    }

    function withdraw(uint amountInWei) external {    Infinite gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        require(amountInWei <= clients[msg.sender]._amount, "not enough money in your account");
        clients[msg.sender]._amount -= amountInWei;
        payable(msg.sender).transfer(amountInWei);
    }

    function getBalance() external view returns (uint) {    2075 gas
        require(bytes(clients[msg.sender]._name).length > 0, "address not found");
        return clients[msg.sender]._amount;
    }

    function checkBalances() external view returns (bool) {    Infinite gas
        uint sum = 0;
        for (uint i = 0; i < adds.length; ++i)
            sum += clients[adds[i]]._amount;
        return (sum == (address(this).balance));
    }
}
```

Se añade `address[] public adds` que guardará las direcciones de los clientes cada vez que uno se registre con la transacción `addClient` y el método `checkBalance` que comprueba en ese array iterativamente que la suma de todo lo depositado sea igual al balance general del contrato.