Building

TRUST ONE BLOCK

at a time

think & dev

Solidity y su evolución

Buenas prácticas en desarrollo/testing

think
& dev

¿Quiénes somos?

- Somos parte de **Think and Dev**, una software factory basada en Delaware, USA, especializada en proyectos Blockchain de alta complejidad.
  - Alejo Lovallo
    - Desde el 2019 en Blockchain. Primero DevOps, luego Dev. Especializado en DeFi.
    - alejo@thinkanddev.com
  - Lucas Marc
    - Llevo más de 10 años de desarrollo, de los cuales poco más del último año fue en el mundo Blockchain.
    - lucas@thinkanddev.com

Información

- Repositorio:
  - https://github.com/Think-and-Dev/eth-latam-workshop-solidity
- Stack:
  - Vscode
  - Node >= v16
  - Hardhat
  - Hardhat plugins:
    - hardhat-deploy
    - hardhat-abi-exporter
    - hardhat-contract-sizer

Temario

- Evolución de Solidity a lo largo de sus versiones
- Desarrollo
- Testing
- Siguientes pasos
- Preguntas
- Cierre

Evolución de Solidity

think & dev

Solidity Evolución

- Evolución de Solidity a lo largo de sus versiones
  - v0.5
  - v0.6
  - v0.7
  - v0.8

Versión 0.5

- Los constructores ahora deben estar definidos mediante la palabra ***constructor.***
- La palabra ***var*** se encuentra deshabilitada en favor de explicitar el tipado.
- La visibilidad de las funciones es obligatoria.
- Se implementa C99 para el scope de las variables locales de las funciones.
- Para emitir eventos se agrega la palabra reservada ***emit***

```
2   pragma solidity =0.5.16;
3
4   contract LockV5 {
5       uint public unlockTime;
6       //var public a = unlockTime;
7       address payable public owner;
8
9       event Withdrawal(uint amount, uint when);
10
11      constructor(uint _unlockTime) public payable {
12          require(
13              now < _unlockTime,
14              "Unlock time should be in the future"
15          );
16          unlockTime = _unlockTime;
17          owner = msg.sender;
18      }
19
20      function withdraw() public {
21          require(now >= unlockTime, "You can't withdraw yet");
22          require(msg.sender == owner, "You aren't the owner");
23
24          emit Withdrawal(address(this).balance, now);
25
26          owner.transfer(address(this).balance);
27      }
28  }
```

contracts/evolution/v5.sol

Versión 0.6

- **Herencia:**
  - Las funciones (que no pertenezcan a una interfaz) sólo pueden ser sobreescritas si se les agrega la nueva palabra reservada ***virtual***.
  - La función que sobreescribe a otra debe agregarse la nueva palabra reservada **override.**
- Se añade la palabra reservada **abstract** para definir ***contratos abstractos***
- Se divide la función de fallback en dos (y se añade la palabra reservada fallback):
  - Función de fallback convencional
  - Función **receive**

```solidity
pragma solidity ^0.6.0;

interface iLockV6 {
    function getOwner() external view returns(address);
    function getUnlockTime(address) external view returns(uint);
}

abstract contract  abstractLockV6 {
    function withdraw() public virtual;
}

contract LockV6 is iLockV6, abstractLockV6{
    uint public unlockTime;
    address payable private owner;

    event Withdrawal(uint amount, uint when);

    constructor(uint _unlockTime) public payable {
        require(
            block.timestamp < _unlockTime,
            "Unlock time should be in the future"
        );
        unlockTime = _unlockTime;
        owner = msg.sender;
    }

    function getOwner() public view override returns(address){
        return owner;
    }

    function getUnlockTime(address _owner) public view override returns(uint){
        require(owner == _owner,"Only owner could ask for unlockTime");
        return unlockTime;
    }

    function withdraw() public override {
        require(block.timestamp >= unlockTime, "You can't withdraw yet");
        require(msg.sender == owner, "You aren't the owner");

        emit Withdrawal(address(this).balance, block.timestamp);

        owner.transfer(address(this).balance);
    }

}
```

contracts/evolution/v6.sol

Versión 0.7

- Se depreca el uso de la variable global **now** en favor de ***block.timestamp*** para evitar ambigüedades

- Se elimina la necesidad de explicitar la visibilidad de los constructores.

contracts/evolution/v7.sol

```
2   pragma solidity ^0.7.6;
3
4   contract LockV7 {
5       uint public unlockTime;
6       address payable public owner;
7
8       event Withdrawal(uint amount, uint when);
9
10      constructor(uint _unlockTime) payable {
11          /**
12          THIS RESULTS IN ERROR
13          require(
14              now < _unlockTime,
15              "Unlock time should be in the future
16          )
17          */
18          require(
19              block.timestamp < _unlockTime,
20              "Unlock time should be in the future"
21          );
22
23          unlockTime = _unlockTime;
24          owner = msg.sender;
25      }
26
27      function withdraw() public {
28          require(block.timestamp >= unlockTime, "You can't withdraw yet");
29          require(msg.sender == owner, "You aren't the owner");
30
31          emit Withdrawal(address(this).balance, block.timestamp);
32
33          owner.transfer(address(this).balance);
34      }
35  }
```

Versión 0.8

- Se añade soporte nativo para las operaciones aritméticas —> Se elimina la necesidad de usar librerías como SafeMath de Oz.
- Se puede operar con el comportamiento anterior si se encierra la operación con el operador ***unchecked***
- Address literals son de tipo address en vez de address payable
- tx.origin y msg.sender también son por defecto de tipo address

contracts/evolution/v8.sol

```
2   pragma solidity ^0.8.9;
3   import "@openzeppelin/contracts/utils/math/SafeMath.sol";
4
5   contract LockV8 {
6       uint256 public unlockTime;
7       address payable public owner;
8
9       event Withdrawal(uint256 amount, uint256 when);
10
11      constructor(uint256 _unlockTime) payable {
12          require(block.timestamp < _unlockTime, "Unlock time should be in the future");
13
14          unlockTime = _unlockTime;
15          owner = payable(msg.sender);
16      }
17
18      function withdraw() public {
19          require(block.timestamp >= unlockTime, "You can't withdraw yet");
20          require(msg.sender == owner, "You aren't the owner");
21
22          emit Withdrawal(address(this).balance, block.timestamp);
23
24          owner.transfer(address(this).balance);
25      }
26
27      function safeSub(uint256 a, uint256 b) external pure returns (uint256) {
28          return a - b;
29      }
30
31      function unsafeSub(uint256 a, uint256 b) external pure returns (uint256) {
32          unchecked {
33              return a - b;
34          }
35      }
36
37      function safeSubWithSafeMath(uint256 a, uint256 b) external pure returns (uint256) {
38          return SafeMath.sub(a, b, "Unsafe sub attempt");
39      }
40  }
```

Desarrollo y buenas prácticas

Desarrollo y buenas prácticas

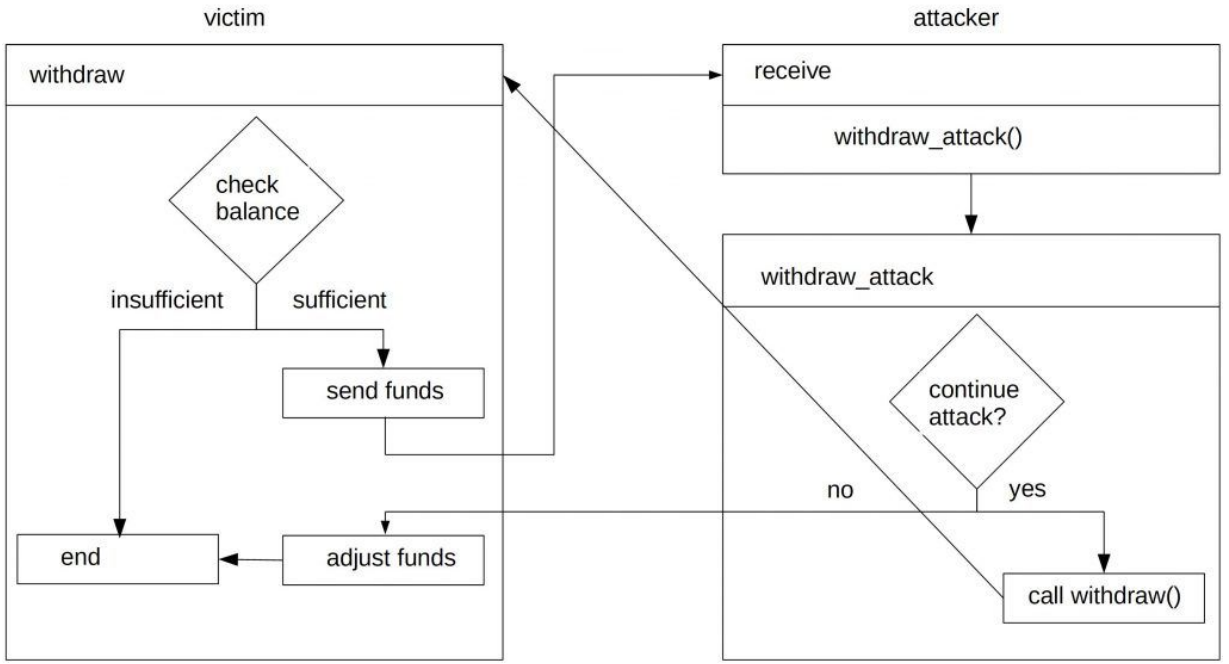
- Re-entrancy
- CHECK - EFFECTS - INTERACTIONS PATTERN
- EMERGENCY STOP PATTERN
- No loops
- Manejo de espacio

Re-entrancy

Check|effects|interactions

Re-entrancy

- Es un tipo de ataque que se puede dar por la forma y orden en el que escribimos nuestras instrucciones en los contratos.
- Veremos con ejemplos de código casos de re-entrancy attacks en la práctica, y cómo podemos defendernos de dichos ataques.
- Finalmente, veremos el uso de ReentrancyGuard de Open Zeppelin
  - https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol



Re-entrancy

- Código del contrato atacante de ejemplo
  - La condición de corte del ataque se maneja con un contador que llega hasta 4, pero podría ser una condición más compleja.
- Se usará el mismo código atacante en todos los ejemplos

contracts/development/reentrancy/re_attacker.sol

```
2  pragma solidity ^0.8.9;
3
4  contract ReentrancyAttacker {
5      address public victim;
6      uint256 public amount;
7      uint256 public counter;
8
9      constructor(address _victim) payable {
10         victim = _victim;
11         amount = msg.value;
12     }
13
14     receive() external payable {
15         counter++;
16         withdrawAttack();
17     }
18
19     function payIn() public returns (bool success) {
20         (success, ) = payable(victim).call{value: amount}(abi.encodeWithSignature("payIn()"));
21     }
22
23     function addAmount() public payable {
24         amount += msg.value;
25     }
26
27     function withdrawAttack() public {
28         if (counter < 4) {
29             payable(victim).call(abi.encodeWithSignature("withdraw()"));
30         }
31     }
32
33     function contractBalance() public view returns (uint256) {
34         return address(this).balance;
35     }
36 }
```

Re-entrancy - unsafe

```solidity
pragma solidity ^0.8.9;

contract ReentrancyVictim {
    //Balance tracking
    mapping(address => uint256) public balances;

    //Declare events
    event Deposit(address indexed _from, uint256 _value);
    event Withdraw(address indexed _from, uint256 _value);

    // Allows the caller to add balance
    function getBalance(address _address) public view returns (uint256 balance) {
        balance = balances[_address];
    }      Lucas Marc, 18 hours ago • chore: reentrancy scripts and example contracts

    // Allows the caller to add balance
    function payIn() public payable {
        emit Deposit(msg.sender, msg.value);
        balances[msg.sender] += msg.value;
    }

    // Withdraws the balance available to the caller
    function withdraw() public payable {
        require(balances[msg.sender] > 0, "Insufficient balance");
        emit Withdraw(msg.sender, balances[msg.sender]);
        (bool success, ) = payable(msg.sender).call{value: balances[msg.sender]}("");
        require(success, "Failed to send funds");
        balances[msg.sender] = 0;
    }

    function contractBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

```
----------Deploying reentrancy example (ReentrancyVictim)----------
Deployer user address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
RentrancyVictimInstance deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
RentrancyAttackerInstance with 1 ETH deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
----------Deployed reentrancy example contracts----------
   Event Deposit. From: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266. Value: 9
   Event Deposit. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance before attack:  10
Attacker contract total balance before attack:  0
Proceed with attack? (Y,n)
Performing attack
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance after attack:  6
Attacker contract total balance after attack:  4
```

contracts/development/reentrancy/re_1_vulnerable.sol

Re-entrancy - safe

```sol
contract ReentrancyProtected {
    //Balance tracking
    mapping(address => uint256) public balances;

    //Declare events
    event Deposit(address indexed _from, uint256 _value);
    event Withdraw(address indexed _from, uint256 _value);

    // Allows the caller to add balance
    function getBalance(address _address) public view returns (uint256 balance) {
        balance = balances[_address];
    }

    // Allows the caller to add balance
    function payIn() public payable {
        balances[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }

    // Withdraws the balance available to the caller
    function withdraw() public payable {
        require(balances[msg.sender] > 0, "Insufficient balance");
        uint256 balance = balances[msg.sender];
        balances[msg.sender] = 0;
        (bool success, ) = payable(msg.sender).call{value: balance}("");
        require(success, "Failed to send funds");
        emit Withdraw(msg.sender, balance);
    }

    function contractBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

```
----------Deploying reentrancy example (ReentrancyProtected)----------
Deployer user address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
RentrancyVictimInstance deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
RentrancyAttackerInstance with 1 ETH deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
----------Deployed reentrancy example contracts----------
   Event Deposit. From: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266. Value: 9
   Event Deposit. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance before attack:  10
Attacker contract total balance before attack:  0
Proceed with attack? (Y,n)
Performing attack
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance after attack:  9
Attacker contract total balance after attack:  1
```

contracts/development/reentrancy/re_2_final.sol

Re-entrancy - safe with OZ

```sol
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract ReentrancyWithOZ is ReentrancyGuard {
    //Balance tracking
    mapping(address => uint256) public balances;

    //Declare events
    event Deposit(address indexed _from, uint256 _value);
    event Withdraw(address indexed _from, uint256 _value);

    // Allows the caller to add balance
    function getBalance(address _address) public view returns (uint256 balance) {
        balance = balances[_address];
    }

    // Allows the caller to add balance
    function payIn() public payable {
        emit Deposit(msg.sender, msg.value);
        balances[msg.sender] += msg.value;
    }

    // Withdraws the balance available to the caller
    function withdraw() public payable nonReentrant {
        require(balances[msg.sender] > 0, "Insufficiant balance");
        emit Withdraw(msg.sender, balances[msg.sender]);
        (bool success, ) = payable(msg.sender).call{value: balances[msg.sender]}("");
        require(success, "Failed to send funds");
        balances[msg.sender] = 0;
    }

    function contractBalance() public view returns (uint256) {
        return address(this).balance;
    }
}
```

```
----------Deploying reentrancy example (ReentrancyWithOZ)----------
Deployer user address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
RentrancyVictimInstance deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
RentrancyAttackerInstance with 1 ETH deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
----------Deployed reentrancy example contracts----------
   Event Deposit. From: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266. Value: 9
   Event Deposit. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance before attack:  10
Attacker contract total balance before attack:  0
Proceed with attack? (Y,n)
Performing attack
   Event Withdraw. From: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512. Value: 1
Victim contract total balance after attack:  9
Attacker contract total balance after attack:  1
```

contracts/development/reentrancy/re_3_oz_modifier.sol

Re-entrancy
OZ ReentrancyGuard

- Código del contrato de ReentrancyGuard de Open Zeppelin
  - Como se puede ver, el modificador de ***nonReentrant*** es un sistema de exclusión mutua, que evita el ingreso de nuevos llamados si hay uno en curso

```
22  abstract contract ReentrancyGuard {
23      uint256 private constant _NOT_ENTERED = 1;
24      uint256 private constant _ENTERED = 2;
25
26      uint256 private _status;
27
28      constructor() {
29          _status = _NOT_ENTERED;
30      }
31
32      /**
33       * @dev Prevents a contract from calling itself, directly or indirectly.
34       * Calling a `nonReentrant` function from another `nonReentrant`
35       * function is not supported. It is possible to prevent this from happening
36       * by making the `nonReentrant` function external, and making it call a
37       * `private` function that does the actual work.
38       */
39      modifier nonReentrant() {
40          // On the first call to nonReentrant, _notEntered will be true
41          require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
42
43          // Any calls to nonReentrant after this point will fail
44          _status = _ENTERED;
45
46          _;
47
48          // By storing the original value once again, a refund is triggered (see
49          // https://eips.ethereum.org/EIPS/eip-2200)
50          _status = _NOT_ENTERED;
51      }
52  }
53
```

CHECK-EFFECTS-INTERACTIONS

- Es un patrón básico que busca prevenir la ejecución inesperada de un contrato. Verifica una serie de requisitos antes de ejecutar cierta acción dentro de una función.

- La máxima que enuncia este patrón es:

  “Cualquier modificación de estado dentro de una función debe suceder

  antes de que se realice una llamada externa”

CHECK-EFFECTS-INTERACTIONS

1. CHECK
   Implementar validaciones necesarias para asegurar que los argumentos, condiciones sean válidas y la función se encuentra lista para ser ejecutada

2. EFFECTS
   Realizar los cambios que afecten y/o modifiquen las variables de estado y por consiguiente el estado del contrato.

3. INTERACTIONS
   Sólo y sólo luego de haber realizado los pasos uno y dos se permiten interacciones con otros contratos. Se incluyen todas las llamadas externas a otros contratos.

Emergency stop pattern

think & dev

EMERGENCY STOP PATTERN

Patrón que detiene la ejecución de ciertas funciones de un contrato

Motivos:

- Seguridad
- Posible bug crítico
- Terminación del contrato por algún motivo

Home made solution

- Modificadores
  - ***haltInEmergency***: Se pausa la ejecución cuando contractStopped = true
  - ***enableInEmergency***: Se permite la ejecución sólo cuando el contrato se encuentra detenido

```
2   pragma solidity ^0.8.9;
3
4   contract StoppablePattern {
5     address public owner;
6     bool public contractStopped = false;
7
8       modifier haltInEmergency {
9           require(!contractStopped);
10          _;
11      }
12
13      modifier enableInEmergency {
14          require(contractStopped);
15          _;
16      }
17
18      modifier onlyOwner{
19          require(msg.sender == owner);
20          _;
21      }
22
23      constructor(address _owner){
24          require(_owner != address(0), "OWNER CAN NOT BE NULL");
25          owner = _owner;
26      }
27
28      function toggleContractStopped() public onlyOwner {
29          contractStopped = !contractStopped;
30      }
31
32      function deposit() public payable haltInEmergency {
33          // some code
34      }
35
36      function withdraw() public haltInEmergency{
37          //some code
38      }
39
40      function emergencyWithdraw() public view enableInEmergency {
41          // some code
42      }
```

Oz solution

- Herencia del contrato Pausable.sol
- Modifcadores
  - ***whenNotPaused***
  - ***whenPaused***

```
3   import "@openzeppelin/contracts/security/Pausable.sol";
4
5   pragma solidity ^0.8.9;
6
7   contract OzStoppablePattern is Pausable {
8
9       function deposit() public payable whenNotPaused {
10          // some code
11      }
12
13      function withdraw() public view {
14          // some code
15      }
16
17      function onlyPaused() public view whenPaused returns (bool) {
18          return this.paused();
19      }
20  }
```

No loops

NO LOOPS

- El uso de instrucciones repetitivas dentro de los contratos se suele desaconsejar por las siguientes razones:
  - Imprevisibilidad de los límites de la instrucción
    - for (i = 0; i < ??? ; i++)
  - Mal uso de gas
  - Posible error de ejecución de la función por quedarse sin gas ***(block.gasLimit)***
  - Es un ***anti-patrón***

NO LOOPS

- Posibles soluciones | consejos:
  - Evitar el ordenamiento
  - Devolver de a una fila a la vez
  - ¿Realizar operación sobre toda una estructura? → Delegar la responsabilidad del lado del usuario
  - ¿El ordenamiento es inevitable? → Off-chain

Manejo de espacio y gas

think & dev

Manejo de espacio y gas

- Tamaño límite de un contrato: 24 KB [EIP 170]
- Consejos y/o buenas prácticas
  - No guardes información de más —> Eventos
  - Usar require-strings cortos
  - Uso adecuado de los modificadores de funciones ***internal*** y ***external***
  - Pack (agrupa) tus variables
- Tip para ahorro de gas
  - Usar ***constant*** ó ***immutable*** cuando sea posible

No guardar información demás

Problema:

- El contrato está almacenando un registro por cada transacción realizada

```
2   pragma solidity ^0.8.9;
3
4   contract MisUseOfStorage {
5       //Balances
6       mapping(address => uint) s_balances;
7
8       // Historico de transferencias
9       mapping(address => uint256[]) s_transfers;
10
11      function transfer(address payable receiver, uint256 amount) public payable{
12          require(s_balances[address(this)]>= amount, "Not enough balance");
13
14          transferBalance(address(this),receiver, amount);
15
16          //SEND ETH
17          (bool sent,) = receiver.call{value: amount}("");
18          require(sent, "Failed to send Ether");
19
20          //MAL USO DE STORAGE: GUARDO QUE LE ENVIE ETH
21          s_transfers[receiver].push(amount);
22      }
23
24      function transferBalance(address from, address to, uint amount) internal{
25          //Transfer code
26      }
27
28  }
```

No guardar información demás

Solución:

- Emitir eventos
- Ahorro de gas además de storage:
  - ~40%

```
36  contract GoodUseOfStorage {
37      //Balances
38      mapping(address => uint) s_balances;
39
40      //Transfer Event
41      event Transfer(address indexed sender, address indexed receiver, uint256 amount);
42
43      function transfer(address payable receiver, uint256 amount) public payable{
44          require(s_balances[address(this)]>= amount, "Not enough balance");
45
46          transferBalance(address(this),receiver, amount);
47
48          //SEND ETH
49          (bool sent,) = receiver.call{value: amount}("");
50          require(sent, "Failed to send Ether");
51
52          emit Transfer(address(this), receiver, amount);
53      }
54
55      function transferBalance(address from, address to, uint amount) internal{
56          //Transfer code
57      }
58
59  }
```

Require strings

- Cualquier require-string ocupa al menos 32 bytes

```
2  pragma solidity ^0.8.9;
3
4  contract ReasonStrings {
5      uint256 balance;
6      uint256 amount;
7
8      modifier badReasonString {
9          require(balance >= amount, "To whomsoever it may concern.
10         I am writing this error message to let you know that the amount you are trying to
11         transfer is unfortunately more than your current balance. Perhaps you made a typo or
12         you are just trying to be a hacker boi. In any case, this transaction is going to revert.
13         Please try again with a lower amount. Warm regards, EVM");
14         _;
15     }
16
17     modifier goodReasonString {
18         require(balance >= amount, "Insufficient balance");
19     }
20
21     modifier possibleOptionForLongStrings {
22         require(balance >= amount, "CODE ERROR: 20, please refer to www....");
23     }
24
25 }
```

Visibility modifiers

- Internal
  - Solo dentro del mismo contrato, y de contratos hijos
- External
  - Ahorra gas
  - Solo para funciones
- Public
  - Tanto para propiedades (getter automático generado por el compilador) como para funciones
- Private
  - Solo para uso interno desde el mismo contrato

Agrupa tus variables

- Cada storage-slot es de 32 bytes
- Se almacenan de forma secuencial
- Cuando una nueva variable declarada no cabe dentro de un slot, se abre uno nuevo

```
2   pragma solidity ^0.8.9;
3
4   contract VariablesPacking {
5
6       struct RegistroSinPacking {
7           address variableA;
8           address variableB;
9           uint96  variableC;
10          uint96  variableD;
11      }
12      /**
13          variableA = 20/32 bytes --> SLOT 1 = 20bytes
14          variableB = 20/32 bytes --> SLOT 2 = 20 bytes
15          variableC = 12/32 bytes --> SLOT 2 = 20 + 12 = 32 bytes
16          variableD = 12/32 bytes --> SLOT 3 = 12 bytes
17      TOTAL = 3 SLOTS
18      */
19
20      struct RegistroConPacking {
21          address variableA;
22          uint96  variableC;
23          address variableB;
24          uint96  variableD;
25      }
26
27      /**
28          variableA = 20/32 bytes --> SLOT 1 = 20bytes
29          variableC = 12/32 bytes --> SLOT 1 = 20 + 12 = 32 bytes
30          variableB = 20/32 bytes --> SLOT 2 = 20 bytes
31          variableD = 12/32 bytes --> SLOT 2 = 20 + 12 bytes = 32 bytes
32      ¡ TOTAL = 2 SLOTS !
33       */
34
35  }
```

Constant e Immutable

- Ejemplo 1
  - Contrato que NO utiliza constantes

```
2   pragma solidity ^0.8.9;
3
4   contract NoConstants {
5       address public TOKEN;
6
7       constructor (address _token){
8           TOKEN = _token;
9       }
10  }
```

- Ejemplo 2
  - Contrato que hace uso de ***constant***

```
20  contract Constant {
21      address public constant TOKEN = 0x153073310327
22
23      constructor(){
24
25      }
26
27  }
```

Constant e Immutable

- Ejemplo 3:
  - Contrato que hace uso de ***immutable***

```
12  contract Immutable {
13      address public immutable TOKEN;
14
15      constructor(address _token){
16          TOKEN = _token;
17      }
18  }
```

Testing

Etapa de pruebas

- Gas estimation plugin (ej: hardhat-gas-reporter)
- Contract sizer plugin (ej: hardhat-contract-sizer)
- Coverage plugin (ej: solidity-coverage)

Gas Estimation

Solc version: 0.8.9	Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas		
Methods					
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
OzStoppablePattern	withdraw	–	–	33820	7	–
Deployments					**% of limit**	
Constant		–	–	86501	0.3 %	–
Immutable		–	–	90857	0.3 %	–
NoConstants		–	–	109202	0.4 %	–

Contract sizer

Contract Name	Size (KiB)	Change (KiB)
Constant	0.149	
Immutable	0.161	
List	1.011	
LockV4	0.483	
LockV5	0.605	
LockV6	0.656	
LockV7	0.489	
LockV8	0.471	
NoConstants	0.142	
OzStoppablePattern	0.525	
StoppablePattern	0.420	

Test coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
development/	8.7	0	16.67	7.69	
OzStoppable.sol	0	100	0	0	18
constants.sol	100	100	100	100	
list.sol	0	100	0	0	... 57,58,60,61
stoppablePattern.sol	0	0	0	0	... 20,24,25,29
evolution/	18.42	18.75	16.67	18.42	
v4.sol	0	0	0	0	... 18,19,21,23
v5.sol	0	0	0	0	... 21,22,24,26
v6.sol	0	0	0	0	... 38,39,41,43
v7.sol	0	0	0	0	... 28,29,31,33
v8.sol	100	100	100	100	
All files	14.75	15	16.67	14.06	

Test coverage

all files development/

8.7% Statements 2/23 **0%** Branches 0/8 **16.67%** Functions 3/18 **7.69%** Lines 2/26

File		Statements		Branches		Functions		Lines	
OzStoppable.sol		0%	0/1	100%	0/0	0%	0/3	0%	0/1
constants.sol		100%	2/2	100%	0/0	100%	3/3	100%	2/2
list.sol		0%	0/14	100%	0/0	0%	0/4	0%	0/14
stoppablePattern.sol		0%	0/6	0%	0/8	0%	0/8	0%	0/9

Test coverage

```
1    // SPDX-License-Identifier: UNLICENSED
2    pragma solidity ^0.8.9;
3
4    contract LockV8 {
5        uint public unlockTime;
6        address payable public owner;
7
8        event Withdrawal(uint amount, uint when);
9
10       constructor(uint _unlockTime) payable {
11   4×      require(
12               block.timestamp < _unlockTime,
13               "Unlock time should be in the future"
14           );
15
16   3×      unlockTime = _unlockTime;
17   3×      owner = payable(msg.sender);
18       }
19
20       function withdraw() public {
21   5×      require(block.timestamp >= unlockTime, "You can't withdraw yet");
22   4×      require(msg.sender == owner, "You aren't the owner");
23
24   3×      emit Withdrawal(address(this).balance, block.timestamp);
25
26   3×      owner.transfer(address(this).balance);
27       }
28   }
29
```

```
1    // SPDX-License-Identifier: UNLICENSED
2    pragma solidity ^0.7.6;
3
4    contract LockV7 {
5        uint public unlockTime;
6        address payable public owner;
7
8        event Withdrawal(uint amount, uint when);
9
10       constructor(uint _unlockTime) payable {
11           /**
12           THIS RESULTS IN ERROR
13           require(
14               now < _unlockTime,
15               "Unlock time should be in the future
16           )
17            */
18           require(
19               block.timestamp < _unlockTime,
20               "Unlock time should be in the future"
21           );
22
23           unlockTime = _unlockTime;
24           owner = msg.sender;
25       }
26
27       function withdraw() public {
28           require(block.timestamp >= unlockTime, "You can't withdraw yet");
29           require(msg.sender == owner, "You aren't the owner");
30
31           emit Withdrawal(address(this).balance, block.timestamp);
32
33           owner.transfer(address(this).balance);
34       }
35   }
36
```

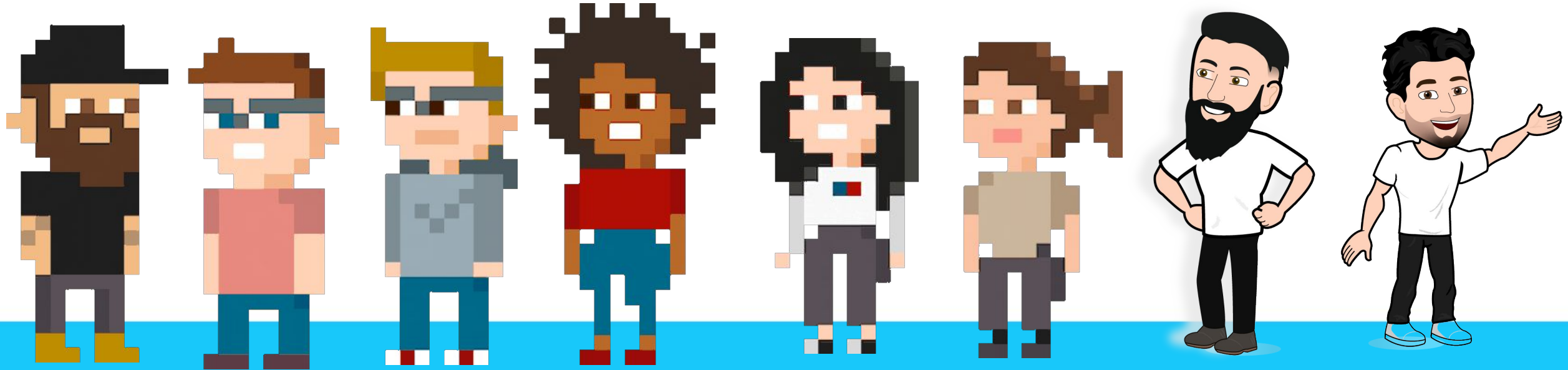
Próximos pasos

- Atomicidad
- Inmutabilidad y **upgradeabilidad (actualización)** de los contratos

Otros patrones:

- Speed bump (Lomos de burro)
  - La idea es agregar un tiempo de espera en el retiro de fondos (También se podría definir una cantidad máxima a retirar)
- Rate limit
  - Limitar la ejecución de llamados continuos
- Mutex
  - Recordar el modificador de nonReentrant del reentrancyGuard de OZ
- Balance limit
  - Definir una balance máximo que puede tener el contrato
- Proxies patterns
  - Storage and implementation layers. Upgradeabilidad

¿Preguntas?



THANK YOU

LET'S TALK! | Thinkanddev.com
https://twitter.com/thinkanddev
https://www.facebook.com/ThinkandDev
https://www.instagram.com/thinkanddevok/
https://www.linkedin.com/company/think-and-dev-llc/
HIRING@THINKANDDEV.COM | HELLO@THINKANDDEV.COM

think & dev