MÁSTER INTERUNIVERSITARIO EN CIBERSEGURIDAD

# Tecnoloxías de rexistro distribuído e Blockchain
# Práctica 1

Héctor Núñez Fernández

## Ejercicio 1

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract FabricaContract {
    uint idDigits = 16;
    uint idModulus = 10 ** idDigits;

    struct Producto {
        string nombre;
        uint id;
    }

    Producto[] public productos;

    event NuevoProducto(uint ArrayProductoId, string nombre, uint id);

    mapping(uint => address) private productoAPropietario;
    mapping(address => uint) private propietarioProductos;

    function _crearProducto(string memory _nombre, uint _id) private {
        productos.push(Producto(_nombre, _id));
        uint productoId = productos.length - 1;
        Propiedad(productoId);
        emit NuevoProducto(productoId, _nombre, _id);
    }

    function _generarIdAleatorio(string memory _str) private view returns (uint) {
        uint rand = uint(keccak256(abi.encodePacked(_str)));
        return rand % idModulus;
    }

    function crearProductoAleatorio(string memory _nombre) public  {
        uint _randId = _generarIdAleatorio(_nombre);
        _crearProducto(_nombre, _randId);
    }

    function Propiedad(uint productoId) private  {
    // Actualizamos el mapping productoAPropietario para almacenar msg.sender bajo ese
    productoId
    productoAPropietario[productoId] = msg.sender;
    // Aumentamos propietarioProductos para msg.sender
    propietarioProductos[msg.sender]++;
    }

    function getProductosPorPropietario(address _propietario) external view returns (
    uint[] memory) {
        uint contador = 0;
        uint cantidadProductos = propietarioProductos[_propietario];
        uint[] memory resultado = new uint[](cantidadProductos);
        for (uint i = 0; i < productos.length; i++) {
            if (productoAPropietario[i] == _propietario) {
```

```
                resultado[contador] = i;
                contador++;
            }
        }
        return resultado;
    }

}
```

Listing 1: FabricaContract.sol

## Ejercicio 2

```
// SPDX-License-Identifier: Unlicensed
pragma solidity ^0.8.18;

contract TokenContract {

    uint256 public tokenPrice = 5 ether;
    address public owner;

    struct Receivers {
        string name;
        uint256 tokens;
    }

    mapping(address => Receivers) public users;

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    event TokensPurchased(address indexed buyer, uint256 amount);

    constructor() {
        owner = msg.sender;
        users[owner].tokens = 100;
    }

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

    function double(uint _value) public pure returns (uint) {
        return _value * 2;
    }

    function register(string memory _name) public {
        users[msg.sender].name = _name;
    }

    function giveToken(address _receiver, uint256 _amount) public onlyOwner {
        require(users[owner].tokens >= _amount);
        users[owner].tokens -= _amount;
        users[_receiver].tokens += _amount;
    }

    function buyTokens(uint256 _amount) public payable {
        uint256 totalCost = _amount * tokenPrice;
        require(msg.value >= totalCost, "Cantidad de Ether insuficiente para comprar
    tokens");
        require(users[owner].tokens >= _amount, "No hay suficientes tokens disponibles
    para la compra");

        giveToken(msg.sender, _amount);

        emit TokensPurchased(msg.sender, _amount);
    }
}
```

Listing 2: TokenContract.sol

## Ejercicio 3

En primer lugar se crea la wallet en MetaMask 1:

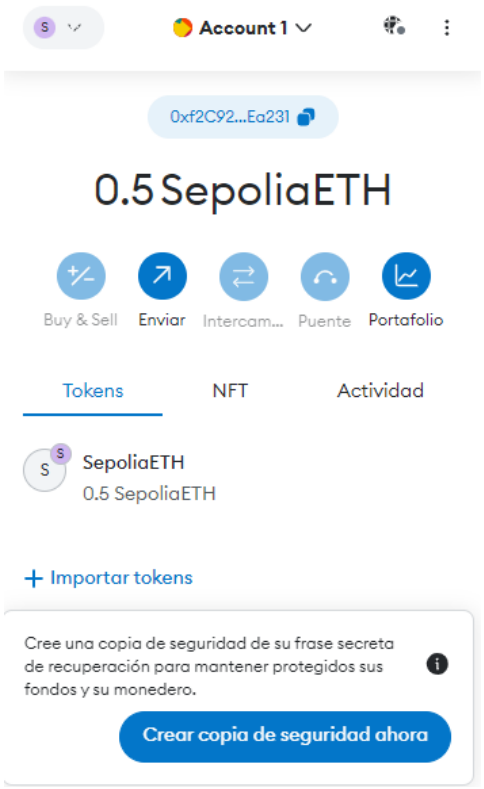


Figure 1: Wallet.

Luego, se envía algo de ETH y en https://sepolia.etherscan.io/ se puede realizar el seguimiento de la transacción 2:

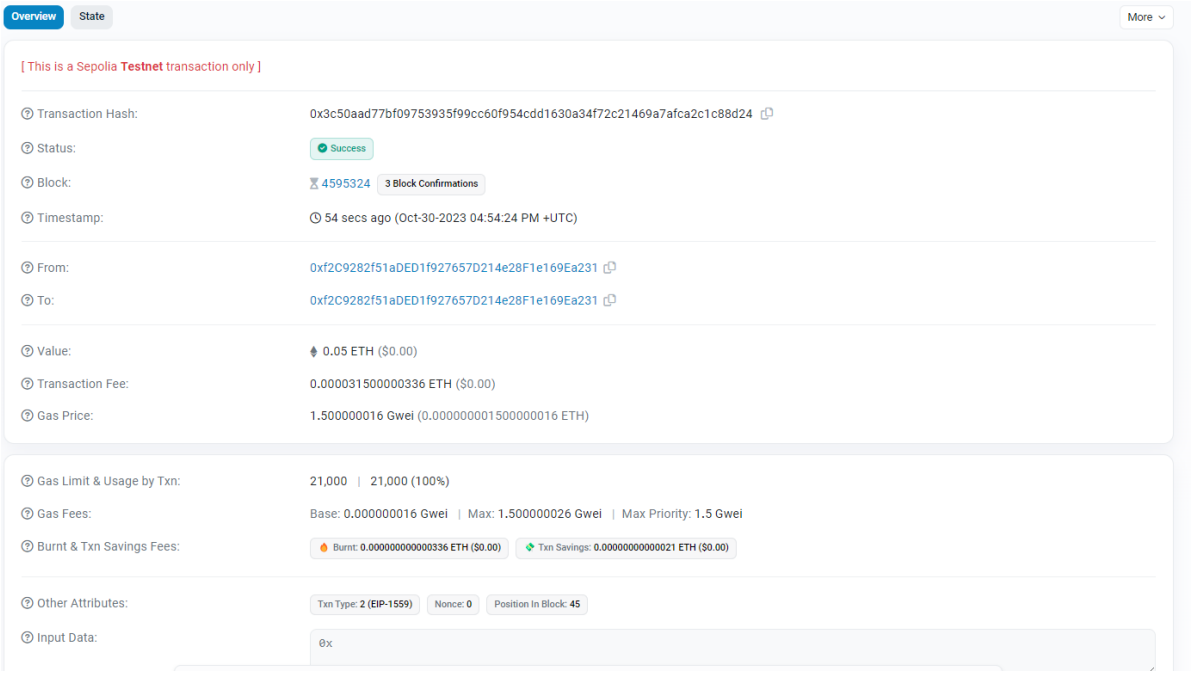


Figure 2: Transaction details.

También se puede ver el saldo de las cuentas implicadas antes y después de la transacción, como me he enviado los ETH a mi mismo, tengo casi el mismo balance, lo que falta es lo que se ha gastado en la transacción 3:

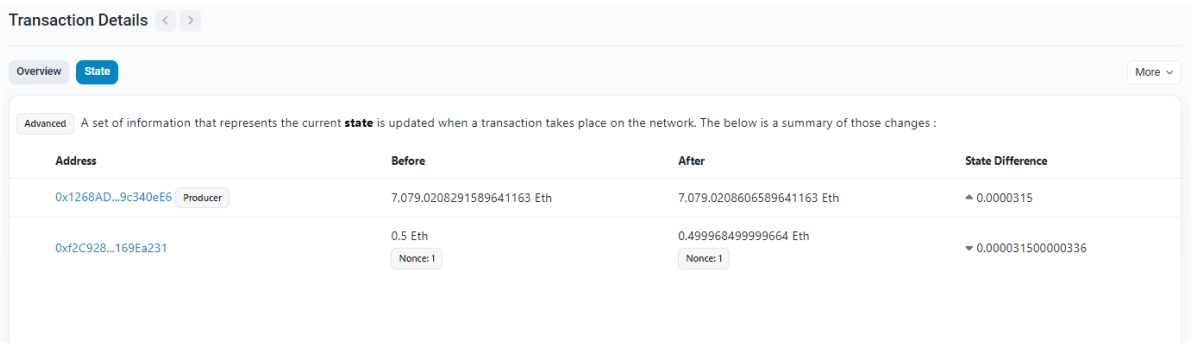


Figure 3: Before and after addresses.

## Ejercicio 4

Tras completar CryptoZombies, el código resultante es el siguiente:

```
contract ERC721 {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function transfer(address _to, uint256 _tokenId) public;
  function approve(address _to, uint256 _tokenId) public;
  function takeOwnership(uint256 _tokenId) public;
}
```

Listing 3: erc721.sol

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>CryptoZombies front-end</title>
    <script language="javascript" type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script language="javascript" type="text/javascript" src="web3.min.js"></script>
    <script language="javascript" type="text/javascript" src="cryptozombies_abi.js"></script>
  </head>
  <body>
    <div id="txStatus"></div>
    <div id="zombies"></div>
    <script>
      var cryptoZombies;
      var userAccount;
      function startApp() {
        var cryptoZombiesAddress = "YOUR_CONTRACT_ADDRESS";
        cryptoZombies = new web3js.eth.Contract(cryptoZombiesABI, cryptoZombiesAddress);
        var accountInterval = setInterval(function() {
          // Comprobar si la cuenta ha sido cambiada
          if (web3.eth.accounts[0] !== userAccount) {
            userAccount = web3.eth.accounts[0];
            // Llamar la funci n que va a updatear la UI with de la nueva cuenta
            getZombiesByOwner(userAccount)
            .then(displayZombies);
          }
        }, 100);
        // Empieza aqu
      }
      function displayZombies(ids) {
        $("#zombies").empty();
        for (id of ids) {
          // Obtener los detalles del zombi de nuestro contrato. Devuelve un objeto `zombie`
          getZombieDetails(id)
          .then(function(zombie) {
            // Se usa las "template literals" (plantillas literales) de ES6 para inyectar variables al HTML.
            // Se adjunta cada uno a nuestro div #zombies
            $("#zombies").append(`<div class="zombie">
              <ul>
                <li>Name: ${zombie.name}</li>
                <li>DNA: ${zombie.dna}</li>
                <li>Level: ${zombie.level}</li>
                <li>Wins: ${zombie.winCount}</li>
                <li>Losses: ${zombie.lossCount}</li>
                <li>Ready Time: ${zombie.readyTime}</li>
              </ul>
            </div>`);
          });
        }
      }
      function createRandomZombie(name) {
        // Esto va a tardar un rato, as  que vamos a updatear la UI para que el usuario
        // sepa que la transacci n se ha hecho correctamente
        $("#txStatus").text("Creating new zombie on the blockchain. This may take a while...");
        // Enviar el texto hacia nuestro contrato:
        return cryptoZombies.methods.createRandomZombie(name)
        .send({ from: userAccount })
        .on("receipt", function(receipt) {
          $("#txStatus").text("Successfully created " + name + "!");
          // La transacci n ha sido aceptada por la blockchain, as  que vamos a redibujar la UI
          getZombiesByOwner(userAccount).then(displayZombies);
        })
        .on("error", function(error) {
          // Se avisa al usuario de que su transacci n no ha sido completada con
```

```
     xito
65          $("#txStatus").text(error);
66        });
67      }
68      function feedOnKitty(zombieId, kittyId) {
69        $("#txStatus").text("Eating a kitty. This may take a while...");
70        return cryptoZombies.methods.feedOnKitty(zombieId, kittyId)
71        .send({ from: userAccount })
72        .on("receipt", function(receipt) {
73          $("#txStatus").text("Ate a kitty and spawned a new Zombie!");
74          getZombiesByOwner(userAccount).then(displayZombies);
75        })
76        .on("error", function(error) {
77          $("#txStatus").text(error);
78        });
79      }
80      function levelUp(zombieId) {
81        $("#txStatus").text("Leveling up your zombie...");
82        return cryptoZombies.methods.levelUp(zombieId)
83        .send({ from: userAccount, value: web3.utils.toWei("0.001", "ether") })
84        .on("receipt", function(receipt) {
85          $("#txStatus").text("Power overwhelming! Zombie successfully leveled up");
86        })
87        .on("error", function(error) {
88          $("#txStatus").text(error);
89        });
90      }
91      function getZombieDetails(id) {
92        return cryptoZombies.methods.zombies(id).call()
93      }
94      function zombieToOwner(id) {
95        return cryptoZombies.methods.zombieToOwner(id).call()
96      }
97      function getZombiesByOwner(owner) {
98        return cryptoZombies.methods.getZombiesByOwner(owner).call()
99      }
100     window.addEventListener('load', function() {
101       // Aqu  se comprueba si Web3.js ha sido inyecto por el navegador (Mist/
    MetaMask)
102       if (typeof web3 !== 'undefined') {
103         // Usar el proveedor Mist/MetaMask
104         web3js = new Web3(web3.currentProvider);
105       } else {
106         // Aqu  se podr a poner algo para avisar al usuario de que no tiene
    Metamask o Mist instalado
107         // Probablemente mostrarle un mensake pidi ndole que se lo instale
108       }
109       // Ahora ya puedes acceder libremente a tu DApp y usar Web3:
110       startApp()
111     })
112   </script>
113 </body>
114 </html>
```

Listing 4: index.html

```
1  /**
2   * @title Ownable
3   * @dev The Ownable contract has an owner address, and provides basic authorization
      control
4   * functions, this simplifies the implementation of "user permissions".
5   */
6  contract Ownable {
7    address public owner;
8    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
9    /**
10    * @dev The Ownable constructor sets the original `owner` of the contract to the
      sender
11    * account.
12    */
13   function Ownable() public {
14     owner = msg.sender;
```

```
  }
  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }
  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
}
```

Listing 5: ownable.sol

```
pragma solidity ^0.4.18;
/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }
  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }
  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
    minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }
  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}
```

Listing 6: safemath.sol

```
pragma solidity ^0.4.19;
import "./zombiehelper.sol";
contract ZombieAttack is ZombieHelper {
  uint randNonce = 0;
  uint attackVictoryProbability = 70;
```

```
  function randMod(uint _modulus) internal returns(uint) {
    randNonce++;
    return uint(keccak256(now, msg.sender, randNonce)) % _modulus;
  }
  function attack(uint _zombieId, uint _targetId) external onlyOwnerOf(_zombieId) {
    Zombie storage myZombie = zombies[_zombieId];
    Zombie storage enemyZombie = zombies[_targetId];
    uint rand = randMod(100);
    if (rand <= attackVictoryProbability) {
      myZombie.winCount++;
      myZombie.level++;
      enemyZombie.lossCount++;
      feedAndMultiply(_zombieId, enemyZombie.dna, "zombie");
    } else {
      myZombie.lossCount++;
      enemyZombie.winCount++;
      _triggerCooldown(myZombie);
    }
  }
}
```

Listing 7: zombieattack.sol

```
pragma solidity ^0.4.19;
import "./ownable.sol";
import "./safemath.sol";
contract ZombieFactory is Ownable {
  using SafeMath for uint256;
  event NewZombie(uint zombieId, string name, uint dna);
  uint dnaDigits = 16;
  uint dnaModulus = 10 ** dnaDigits;
  uint cooldownTime = 1 days;
  struct Zombie {
    string name;
    uint dna;
    uint32 level;
    uint32 readyTime;
    uint16 winCount;
    uint16 lossCount;
  }
  Zombie[] public zombies;
  mapping (uint => address) public zombieToOwner;
  mapping (address => uint) ownerZombieCount;
  function _createZombie(string _name, uint _dna) internal {
    uint id = zombies.push(Zombie(_name, _dna, 1, uint32(now + cooldownTime), 0, 0)) -
    1;
    zombieToOwner[id] = msg.sender;
    ownerZombieCount[msg.sender]++;
    NewZombie(id, _name, _dna);
  }
  function _generateRandomDna(string _str) private view returns (uint) {
    uint rand = uint(keccak256(_str));
    return rand % dnaModulus;
  }
  function createRandomZombie(string _name) public {
    require(ownerZombieCount[msg.sender] == 0);
    uint randDna = _generateRandomDna(_name);
    randDna = randDna - randDna % 100;
    _createZombie(_name, randDna);
  }
}
```

Listing 8: zombiefactory.sol

```
pragma solidity ^0.4.19;
import "./zombiefactory.sol";
contract KittyInterface {
  function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
```

```
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
  );
}
contract ZombieFeeding is ZombieFactory {
  KittyInterface kittyContract;
  modifier onlyOwnerOf(uint _zombieId) {
    require(msg.sender == zombieToOwner[_zombieId]);
    _;
  }
  function setKittyContractAddress(address _address) external onlyOwner {
    kittyContract = KittyInterface(_address);
  }
  function _triggerCooldown(Zombie storage _zombie) internal {
    _zombie.readyTime = uint32(now + cooldownTime);
  }
  function _isReady(Zombie storage _zombie) internal view returns (bool) {
      return (_zombie.readyTime <= now);
  }
  function feedAndMultiply(uint _zombieId, uint _targetDna, string _species) internal
    onlyOwnerOf(_zombieId) {
    Zombie storage myZombie = zombies[_zombieId];
    require(_isReady(myZombie));
    _targetDna = _targetDna % dnaModulus;
    uint newDna = (myZombie.dna + _targetDna) / 2;
    if (keccak256(_species) == keccak256("kitty")) {
      newDna = newDna - newDna % 100 + 99;
    }
    _createZombie("NoName", newDna);
    _triggerCooldown(myZombie);
  }
  function feedOnKitty(uint _zombieId, uint _kittyId) public {
    uint kittyDna;
    (,,,,,,,,,kittyDna) = kittyContract.getKitty(_kittyId);
    feedAndMultiply(_zombieId, kittyDna, "kitty");
  }
}
```

Listing 9: zombiefeeding.sol

```
pragma solidity ^0.4.19;
import "./zombiefeeding.sol";
contract ZombieHelper is ZombieFeeding {
  uint levelUpFee = 0.001 ether;
  modifier aboveLevel(uint _level, uint _zombieId) {
    require(zombies[_zombieId].level >= _level);
    _;
  }
  function withdraw() external onlyOwner {
    owner.transfer(this.balance);
  }
  function setLevelUpFee(uint _fee) external onlyOwner {
    levelUpFee = _fee;
  }
  function levelUp(uint _zombieId) external payable {
    require(msg.value == levelUpFee);
    zombies[_zombieId].level++;
  }
  function changeName(uint _zombieId, string _newName) external aboveLevel(2, _zombieId
    ) onlyOwnerOf(_zombieId) {
    zombies[_zombieId].name = _newName;
  }
  function changeDna(uint _zombieId, uint _newDna) external aboveLevel(20, _zombieId)
    onlyOwnerOf(_zombieId) {
    zombies[_zombieId].dna = _newDna;
  }
  function getZombiesByOwner(address _owner) external view returns(uint[]) {
    uint[] memory result = new uint[](ownerZombieCount[_owner]);
```

```
    uint counter = 0;
    for (uint i = 0; i < zombies.length; i++) {
      if (zombieToOwner[i] == _owner) {
        result[counter] = i;
        counter++;
      }
    }
    return result;
  }
}
```

Listing 10: zombiehelper.sol

```
pragma solidity ^0.4.19;
import "./zombieattack.sol";
import "./erc721.sol";
import "./safemath.sol";
/// @title Un contrato que gestiona la transferencia de la propiedad de un zombi.
/// @author H ctor  N   ez  Fern ndez
/// @dev Compatible con la implementaci n de OpenZeppelin borrador de la
    especificaci n ERC721
contract ZombieOwnership is ZombieAttack, ERC721 {
  using SafeMath for uint256;
  mapping (uint => address) zombieApprovals;
  function balanceOf(address _owner) public view returns (uint256 _balance) {
    return ownerZombieCount[_owner];
  }
  function ownerOf(uint256 _tokenId) public view returns (address _owner) {
    return zombieToOwner[_tokenId];
  }
  function _transfer(address _from, address _to, uint256 _tokenId) private {
    ownerZombieCount[_to] = ownerZombieCount[_to].add(1);
    ownerZombieCount[msg.sender] = ownerZombieCount[msg.sender].sub(1);
    zombieToOwner[_tokenId] = _to;
    Transfer(_from, _to, _tokenId);
  }
  function transfer(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    _transfer(msg.sender, _to, _tokenId);
  }
  function approve(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    zombieApprovals[_tokenId] = _to;
    Approval(msg.sender, _to, _tokenId);
  }
  function takeOwnership(uint256 _tokenId) public {
    require(zombieApprovals[_tokenId] == msg.sender);
    address owner = ownerOf(_tokenId);
    _transfer(owner, msg.sender, _tokenId);
  }
}
```

Listing 11: zombieownership.sol

## Ejercicio 5

Para este ejercicio seleccioné el contrato Voting, de https://docs.soliditylang.org/en/v0.8.13/solidity-by-example.html:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;
/// @title Voting with delegation.
contract Ballot {
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }

    // This is a type for a single proposal.
```

```
    struct Proposal {
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;

    // This declares a state variable that
    // stores a `Voter` struct for each possible address.
    mapping(address => Voter) public voters;

    // A dynamically-sized array of `Proposal` structs.
    Proposal[] public proposals;

    /// Create a new ballot to choose one of `/`.
    constructor(bytes32[] memory proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        // For each of the provided proposal names,
        // create a new proposal object and add it
        // to the end of the array.
        for (uint i = 0; i < proposalNames.length; i++) {
            // `Proposal({...})` creates a temporary
            // Proposal object and `proposals.push(...)`
            // appends it to the end of `proposals`.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }

    // Give `voter` the right to vote on this ballot.
    // May only be called by `chairperson`.
    function giveRightToVote(address voter) external {
        // If the first argument of `require` evaluates
        // to `false`, execution terminates and all
        // changes to the state and to Ether balances
        // are reverted.
        // This used to consume all gas in old EVM versions, but
        // not anymore.
        // It is often a good idea to use `require` to check if
        // functions are called correctly.
        // As a second argument, you can also provide an
        // explanation about what went wrong.
        require(
            msg.sender == chairperson,
            "Only chairperson can give right to vote."
        );
        require(
            !voters[voter].voted,
            "The voter already voted."
        );
        require(voters[voter].weight == 0);
        voters[voter].weight = 1;
    }

    /// Delegate your vote to the voter `to`.
    function delegate(address to) external {
        // assigns reference
        Voter storage sender = voters[msg.sender];
        require(!sender.voted, "You already voted.");

        require(to != msg.sender, "Self-delegation is disallowed.");

        // Forward the delegation as long as
        // `to` also delegated.
        // In general, such loops are very dangerous,
        // because if they run too long, they might
        // need more gas than is available in a block.
        // In this case, the delegation will not be executed,
```

```
        // but in other situations, such loops might
        // cause a contract to get "stuck" completely.
        while (voters[to].delegate != address(0)) {
            to = voters[to].delegate;

            // We found a loop in the delegation, not allowed.
            require(to != msg.sender, "Found loop in delegation.");
        }

        // Since `sender` is a reference, this
        // modifies `voters[msg.sender].voted`
        Voter storage delegate_ = voters[to];

        // Voters cannot delegate to wallets that cannot vote.
        require(delegate_.weight >= 1);
        sender.voted = true;
        sender.delegate = to;
        if (delegate_.voted) {
            // If the delegate already voted,
            // directly add to the number of votes
            proposals[delegate_.vote].voteCount += sender.weight;
        } else {
            // If the delegate did not vote yet,
            // add to her weight.
            delegate_.weight += sender.weight;
        }
    }

    /// Give your vote (including votes delegated to you)
    /// to proposal `proposals[proposal].name`.
    function vote(uint proposal) external {
        Voter storage sender = voters[msg.sender];
        require(sender.weight != 0, "Has no right to vote");
        require(!sender.voted, "Already voted.");
        sender.voted = true;
        sender.vote = proposal;

        // If `proposal` is out of the range of the array,
        // this will throw automatically and revert all
        // changes.
        proposals[proposal].voteCount += sender.weight;
    }

    /// @dev Computes the winning proposal taking all
    /// previous votes into account.
    function winningProposal() public view
            returns (uint winningProposal_)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    // Calls winningProposal() function to get the index
    // of the winner contained in the proposals array and then
    // returns the name of the winner
    function winnerName() external view
            returns (bytes32 winnerName_)
    {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

Listing 12: voting.sol