Contrato de Distribución

Interfaz de IERC20 y Contexto

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
interface IERC20Metadata is IERC20 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
}


abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return payable(msg.sender);
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this;
        return msg.data;
    }
}
```

Función de Ownership (Propiedad de Contrato)

- Owner() → Retorna el dueño del contrato
- onlyOwner() → Modificador que determina que todas las funciones con el mismo solo podrán ser llamadas por el Owner()
- trasnferOwnership(newOwner) → Toma un address como entrada, transfiere Owner() a la nueva billetera.

```
contract Ownable is Context {
    address private _owner;
    address private _previousOwner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

```
}
```

Declaracion de Variables

- token → Variable tipo IERC20 que hace referencia al token BEP20 que se usara para las distribuciones (inicializado al lanzar el contrato con la direccion de BUSD, pero puede ser cambiado)
- tokenHolder → Variable que guarda el address de la billetera “spender”, es decir, la billetera que tiene el toke para ser distribuido
- distributionPercentages → Variable del tipo mapping que guarda el porcentaje de distribucion de cada wallet
- arrayPosition → Variable del tipo mapping que guarda la posicion de cada billetera en la lista distributionAddresses (variable de uso interno)
- exists → variable del tipo mapping que guarda la existencia de una direccion o billetera en la lista distributionAddresses (variable de uso interno)
- distributionAddresses → Lista de todas las direcciones que forman parte de la distribucion
- percentages → Lista de todos los porcentajes de distribucion de cada billetera. La posicion 1 en la lista corresponde a la billetera 1 en distributionAddresses, y asi sucesivamente.
- masterSum → Usada para debug, no usada en el contrato final.

```
contract DistributionContract is Context, Ownable {
    IERC20 public token;
    address payable public tokenHolder;

    mapping (address => uint) distributionPercentages;
    mapping (address => uint) arrayPosition;
    mapping(address => bool) exists;
    address[] distributionAddresses;
    uint[] percentages;
    uint masterSum;

    event Validate(uint sum, bool result);
    event Sent(address from, address to, uint amount );
```

Constructor

- tokenHolder es iniciado al wallet que tendra los tokens para distribucion
- token es iniciado a la direccion del token que sera usado para distribucion

```
constructor(address payable _tokenHolder, address _token) {
        tokenHolder = _tokenHolder;
        token = IERC20(_token);
    }
```

Funciones

- changeTokenHolder(_netTokenHolder) ONLYOWNER
  - Toma una direccion de billetera como argumento
  - Cambia la billetera que posee los tokens para distribucion a la nueva billetera
- addAddressToDistribution(address, _percentAmount) ONLYOWNER

    - Toma una direccion de billetera y un porcentaje (DEBE ser un numero natural entre el 1 y el 100. No se aceptan decimales)
    - Verifica si la direccion ya existe en la lista
    - De no existir, crea la direccion en la lista y agrega el porcentaje de distribucion
- removeAddressFromDistribution(address) ONLYOWNER
    - Toma una direccion de billetera como argumento
    - Verifica que la direccion exista en las listas
    - Quita la direccion de la lista, cambiandola por el “zero address” (0x0) y poniendo su porcentaje de distribucion a 0.
- editAddressFromDistribution(_address, _newPercentAmount) ONLYOWNER
    - Toma una direccion de billetera existente en las listas y un porcentaje (DEBE ser un numero natural entre el 1 y el 100. No se aceptan decimales)
    - Verifica si la direccion ya existe en la lista
    - De existir, actualiza el porcentaje de distribucion para esta billetera
- cleanDistributionArray() ONLYOWNER
    - Borra todas las direcciones y porcentajes de distribucion
    - Util para “Empezar de nuevo” con una lista completamente vacia.
- addAddressesFromArray(address[], uint[]) ONLYOWNER
    - Toma una lista de direcciones de billetera y una lista de porcentajes (DEBEN ser numeros naturales entre el 1 y el 100. No se aceptan decimales. La suma debe ser igual a 100)
        - Las listas tienen el siguiente formato: [”direccion1”,”direccion2”] [porcentaje1,porcentaje2]
        - Es decir, las direcciones van entre comillas y separadas por comas, dentro de parentesis cuadrados. Los porcentajes entre parentesis cuadrados, separados con comas, sin comillas.
    - Verifica que el largo de ambas listas es igual
    - Verifica si las direcciones ya existen en la lista
    - De no existir, crea la direccion en la lista y agrega el porcentaje de distribucion
- addAddressToDistributionNoUpdate() INTERNAL
    - Funcion de uso interno para complementar a addAddressesFromArray

```
function changeTokenHolder(address payable _newTokenHolder) external onlyOwner(){
        tokenHolder = _newTokenHolder;
    }

    function addAddressToDistribution(address _address, uint _percentAmount) external onlyOwner(){
        require(exists[_address] == false, "the address already exists on the list. Please delete it or replace its value using the appropi
        distributionAddresses.push(_address);
        distributionPercentages[_address]=_percentAmount;
        exists[_address] = true;
        updatePercentages();
    }

    function removeAddressFromDistribution(address _address) external onlyOwner(){
        require(exists[_address] == true, "the address does not exist");
        delete distributionAddresses[arrayPosition[_address]];
        distributionPercentages[_address]=0;
        exists[_address] = false;
        updatePercentages();
    }

    function editAddressFromDistribution(address _address, uint _newPercentAmount) external onlyOwner(){
        require(exists[_address] == true, "the address does not exist");
        distributionPercentages[_address]=_newPercentAmount;
        updatePercentages();
    }
```

```
    function cleanDistributionArray() external onlyOwner(){
        uint i = 0;

        while (i<distributionAddresses.length) {
            exists[distributionAddresses[i]] = false;
            arrayPosition[distributionAddresses[i]] = 0;
            distributionPercentages[distributionAddresses[i]] = 0;
            i++;
        }
        delete distributionAddresses;
        delete percentages;
    }


    function addAddressesFromArray(address[] memory _addresses, uint[] memory _percentAmounts) external onlyOwner(){
        require(_addresses.length == _percentAmounts.length, "Array lenghts do not match");

        uint i = 0;

        while (i<_addresses.length) {
            addAddressToDistributionNoUpdate(_addresses[i], _percentAmounts[i]);
            i++;
        }
        updatePercentages();
    }

    function addAddressToDistributionNoUpdate(address _address, uint _percentAmount) internal{
        require(exists[_address] == false, "the address already exists on the list. Please delete it or replace its value using the appropi
        distributionAddresses.push(_address);
        distributionPercentages[_address]=_percentAmount;
        exists[_address] = true;
    }
```

- updatePercentages() INTERNAL
  - Funcion de uso interno para mantener las listas correctamente estructuradas
- getListOftAddresses() PUBLIC
  - Funcion retorna la lista de todas las direcciones que recibiran dividendos
- getListOfAddressesAndPercentages() PUBLIC
  - Funcion retorna la lista de todas las direcciones que recibiran dividendos, y una segunda lista de los porcentajes de distribucion.
- validate() INTERNAL
  - Funcion de uso interno que valida que la suma de todos los porcentajes sea igual a 100%. Es llamada antes de realizar la distribucion de tokens, si el resultado no es igual a 100% la funcion falla.
- getBalanceFromHolder() PUBLIC
  - La funcion retorna el balance de token del tokenHolder
- transact() INTERNAL
  - Funcion que realiza el envio de tokens entre el tokenHolder y el destinatario final.
  - Funciona en conjunto con distributeTokens
- calculateAmountToDistribute() INTERNAL
  - Funcion que realiza el calculo de tokens para distribuir a cada wallet basado en los porcentajes de distribucion
  - Funciona en conjunto con distributeTokens
- distributeTokens(amount) ONLYOWNER
  - Funcion que toma un amount (numero de tokens) para distribuir y los distribuye a todas las direcciones que reciben dividendos segun su porcentaje de distribucion

        - Importante: La funcion falla si no se ha aprobado esta cantidad para ser enviada (approve transaction by sender → Debe ser aprobada usando la funcion nativa del token INSTRUCCIONES PARA BUSD ABAJO)
        - La funcion falla si la suma de los porcentajes no es igual al 100%
        - La funcion falla si el tokenHolder no posee suficientes tokens para realizar la transaccion
        - El valor de “amount” es el valor en WEI de el token. Es decir, el valor con la maxima cantidad de decimales. El numero de decimales estandar en la mayoria de los tokens ERC20/BEP20 es 18, pero puede variar. Verificar
            - Es decir, si se quiere enviar 100 token, y el token tiene 18 decimales, se debe escribir 100 + 18 ceros, 100000000000000000000 wei = 100 token INSTRUCCIONES PARA BUSD ABAJO
    - Si las condiciones son correctas, se divide el monto entre todos los socios, segun sus porcentajes, y se envian los tokens.
- changeDistributionToken(_tokenAddress) ONLYOWNER
    - Toma una direccion tipo IERC20 y cambia la direccion del token declarado en el constructor a la nueva direccion.
- getDistributionToken() PUBLIC
    - Retorna la direccion del token de distribucion

```
function updatePercentages() internal{
        delete percentages;
        uint i = 0;

        while (i<distributionAddresses.length) {
            percentages.push(distributionPercentages[distributionAddresses[i]]);
            i++;
        }

        uint j = 0;

        while (j<distributionAddresses.length) {
            arrayPosition[distributionAddresses[j]] = j;
            j++;
        }

    }

    function getListOfAddresses() public view returns (address[] memory){
        return distributionAddresses;
    }

    function getListOfAddressesAndPercentages() public view returns (address[] memory, uint[] memory){
        return (distributionAddresses,percentages);
    }

    function validate() internal view returns (bool){
        uint i = 0;
        uint sum = 0;

        while (i<percentages.length) {
            sum = sum + percentages[i];
            i++;
        }

        if (sum == 100){
            return true;
        }
        else{
            return false;
        }

    }

    function getBalanceFromHolder() public view returns(uint) {
        uint balance_ = token.balanceOf(tokenHolder);
        return balance_;
    }

    function transact(uint amount, address recipient) internal {
        token.transferFrom(msg.sender, recipient, amount);
    }
```

```
    function calculateAmountToDistribute(uint amount, uint percent) internal returns(uint){
        uint amountToDistribute = amount*percent/100;
        return amountToDistribute;
    }

    function distributeTokens(uint amount) external onlyOwner(){
        require(amount <= token.balanceOf(tokenHolder), "you don't have enough balance.");
        bool validated = validate();
        require(validated == true, "Sum of percentages not equal to 100, can't execute transaction");

        uint i = 0;
        address payable transferAddress;
        uint amountToDistribute;


        while (i<distributionAddresses.length) {
            transferAddress = payable(distributionAddresses[i]);
            amountToDistribute = calculateAmountToDistribute(amount, percentages[i]);
            transact(amountToDistribute,transferAddress);
            i++;
        }

    }

    function changeDistributionToken(IERC20 _tokenAddress) external onlyOwner(){
        token = _tokenAddress;
    }

    function getDistributionToken() public view returns(IERC20){
        return token;
    }

}
```

Instrucciones para BUSD

- Las transacciones puede ser preaprobadadas aca: https://bscscan.com/token/0xe9e7cea3dedca5984780bafc599bd69add087d56#readContract
    - Conectar la billetera usando “connect to web3” y usar funcion de “approve”
    - Ingresar la direccion del contrato, una vez lanzado en la red, como “spender” (el que gastara los tokens)
    - Poner de Amount la cantidad de tokens a ser “gastados” (es decir, distribuidos)
    - Cualquier transaccion de esa cantidad o menor sera aprobada automaticamente
    - Se podria aprobar un valor bastante alto (100000000000000000000000000) de manera de “preaprobar” todas las transacciones. El riesgo de seguridad es bajo ya que solo el dueño del contrato tiene acceso a esas funciones.
- BUSD tiene 18 decimales. Es decir, 100BUSD (100$) es equivalente a 100000000000000000000 wei (100 + 18 ceros). Cada vez que se introduzca un “amount”, este debe ser seguido de 18 ceros.
- la variable token para BUSD seria iniciada a este valor: 0xe9e7cea3dedca5984780bafc599bd69add087d56