Clase 3-solidity

martes, 15 de octubre de 2024 14:25

Lenguaje para escribir smart contracts en ethereum
? Primitivos:
? Booleanos: bool (con valores true o false).
? Enteros: int y uint (para números enteros con y sin signo, respectivamente). Ambos pueden tener tamaños múltiples de 8 bits hasta 256 bits, como int8, int16... int256 y uint8, uint16... uint256.
? Direcciones:
? address: Representa una dirección Ethereum de 20 bytes.
? address payable: Similar a address, pero permite recibir y enviar ether.
? Bytes:
? bytes1, bytes2,..., bytes32 (secuencias de bytes de longitud fija).
? Tipos de Punto Flotante:
? No hay soporte nativo para tipos de punto flotante en Solidity debido a complicaciones con la precisión y la determinística en la EVM
Arrays:
? Arrays de longitud fija y dinámica, por ejemplo, uint[5] (fijo) y uint[] (dinámico).
? Bytes Dinámicos: bytes (similares a bytes[], pero de longitud dinámica).
? Strings: string (representa una cadena de caracteres UTF-8 de longitud dinámica).
? Maps: mapping(key => value). Funcionan como diccionarios o tablas hash, donde key puede ser casi cualquier tipo excepto un mapping, y value puede ser cualquier tipo, incluido otro mapping.
? Structs: Definen estructuras personalizadas.
? Enums: Se utilizan para definir variables que pueden tener uno de los valores predefinidos

Solidity: definición de un contrato

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract SimpleContract {
    uint256 public storedData;

    constructor(uint256 initialValue) {   infinite gas 76600 gas
        storedData = initialValue;
    }

    function set(uint256 x) public {   22520 gas
        storedData = x;
    }

    function get() public view returns (uint256) {   2459 gas
        return storedData;
    }
}
```

- State function: modifican estado del contrato (le cambian alguna variable). Consumen gas y las transacciones que las invocan tienen que validarse en un bloque.
- View function: no modifican el contrato, solo leen valores. No consumen gas al ser llamadas externamente. MODIFICADOR VIEW.
- Pure function: parecido a view pero restringen más. No leen ni modifican el estado del contrato. Retornan un valor basado solo en sus argumentos
- Payable functions: funciones que pueden recibir ether.
- Internal functions: solo se pueden llamar desde el propio contrato o desde derivados. No los pueden llamar transacciones externas. Se usan para dividir la lógica del contrato en partes más chicas, reutilizar código.

- External functions: destinada a ser llamada desde transacciones externas y NO desde otras funciones dentro del contrado. Pueden ser más eficientes en términos de gas cuando se llaman desde el exterior en comparación con las funciones public

- Fallback: antes, si un contrato recibia ether sin que ninguna función se llamase (a transacción de Ether a la dirección del contrato sin datos adicionales) y si el contrato no tendía una función payable entonces se ejecutaba fallback.
- Receive: se usa ahora, es una variabnte de fallback que se ejecuta cuando no se llama a ninguna otra función y no se dan datos

No entendí esto de fallback y receive :)

Modificadores: --> decorator
Para cambiar el comportamiento de funciones o verificar condicions antes de ejecutar una función. Agregar lógica extra a la ejecución de una función. Se usan para verificar permisos, restricciones o estados del contrato.

```
// Modificador para restringir el acceso solo al propietario
modifier onlyOwner() {
    require(msg.sender == owner, "Solo el propietario puede ejecutar esto");
    _;  // Placeholder: aquí es donde se insertará el código de la función modificada
}
```

La herencia se escribe con is. Hay herencia múltiple. Un contrato puede heredar de varios. S'importa el orden por performance y demás.

Si un contrato tiene un contructor, hay que llamarlo en el contructor del contrato derivado

```
pragma solidity ^0.8.0;

import "./EjemploFunciones.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract Example is EjemploFunciones, ERC20{

    address public owner;

    constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_){
        owner = msg.sender;
    }

    // Modificador para restringir el acceso solo al propietario
```

Crea un nuevo constructor y llama al constructor de lo que hereda

Modificadores de tipos de datos:

memory: Indica que la variable se almacena temporalmente y desaparecerá entre llamadas de función externa.
storage: Indica que la variable se almacena de forma persistente en el almacenamiento del contrato
calldata: Es similar a memory, pero es inmutable y se utiliza principalmente para parámetros de funciones externas.
constant / immutable:Estos modificadores se utilizan para variables de estado que no cambiarán después de que se asignen. Mientras que constant se usaba para variables cuyo valor se conoce en tiempo de compilación, immutable es para variables que se asignan una vez durante la construcción y no cambian después

palabras clave o reservadas
Algunas importantes, aparte de las ya vistas:
`abstract`: Indica que un contrato no puede ser desplegado por sí mismo y debe ser heredado por otro contrato.
`delete`: Usado para borrar contenido en storage o para resetear una variable a su valor predeterminado.
`library`: Define una biblioteca, que es similar a un contrato pero no tiene variables de estado.
`override`: Indica que una función sobrescribe una función heredada.
`this`: Refiere a la instancia actual del contrato.
`try-catch`: Utilizado para manejar fallos en llamadas externas.
`type`: Obtiene el tipo de una variable.
`using`: Indica que un contrato usa una biblioteca específica

Evento

Para disparar información a los clientes. Se puede escuchar asincrónicamente y hacer algo cuando recibo el mensaje

```
contract Storage {

    event DataChanged(address indexed by, uint256 oldValue, uint256 newValue);
    uint256 public data;

    function setData(uint256 _data) public {
        emit DataChanged(msg.sender, data, _data);  // Emitir el evento
        data = _data;  // Cambiar la data
    }
```