Desarrollo Blockchain Ethereum con Solidity

Módulo 3 - Herencia

Herencia

Herencia de contratos

- Al igual que en la programación orientada a objetos, los contratos pueden derivarse o “heredarse”, por lo cual podemos reutilizar el código que escribimos en un contrato en otros contratos.
- Es una buena manera de distribuir el código de forma que no se exceda el máximo establecido para los contratos.
- Se implementa a través de la palabra *is.*
- Se puede heredar de más de un contrato.
- Su finalidad principal es la de aplicar el polimorfismo en los contratos.

Herencia de contratos

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Padre {
    string private _nombre;

    constructor(string memory nombre) {
        _nombre = nombre;
    }
}

contract Hijo is Padre {
    string _apellido;

    constructor (string memory nombre, string memory apellido) Padre (nombre) {
        _apellido = apellido;
    }

}
```

Redefinición de funciones

- Cuando queremos que una función pueda ser reescrita en los contratos derivados debemos usar la palabra **virtual** en los modificadores.
- Del lado del contrato derivado, debemos usar la palabra **override** para indicar que estamos reescribiendo una función del contrato “padre”.
- Desde el contrato derivado podemos llamar a una función del contrato padre con la sentencia **super.<nombre de la función>(parámetros).**
- Para el caso de la herencia múltiple, si existe una función que está definida en más de un contrato, debemos especificar qué contratos son los que estamos redefiniendo por medio de **override(contrato1, contrato2, ...)**
- Es posible también definir a los **modifiers** como **virtual**

Redefinición de funciones

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract Padre {
    string private _nombre;

    constructor(string memory nombre) {
        _nombre = nombre;
    }

    function mostrarDatos() public view virtual returns(string memory) {
        return _nombre;
    }
}

contract Hijo is Padre {
    string _apellido;

    constructor (string memory nombre, string memory apellido) Padre (nombre) {
        _apellido = apellido;
    }

    function mostrarDatos() public view override returns(string memory) {
        return _apellido;
    }

}
```

Contratos abstractos

- Si un contrato contiene al menos una función sin implementación, entonces debe incluir la palabra ***abstract.***
- Los contratos abstractos no se pueden instanciar, sólo se utilizan para ser derivados.
- Pueden incluir variables de estado del contrato.

```
abstract contract Persona {
    function mostrarDatos() public virtual returns (string memory);
}
```

Interfaces

- Son similares a los contratos abstractos, pero ninguna función lleva implementación.
- No tienen variables ni constructor.
- No pueden tener variables de estado ya que no son contratos.
- No pueden ser derivados de un contrato, pero pueden ser derivados de una interface.
- Todas las funciones declaradas tienen que ser de tipo external.

```
interface iPersona {
    function mostrarDatos() external returns (string memory);
}
```

Polimorfismo

Teniendo un tipo de entidad definido (contrato o interface) y sabiendo su dirección en la blockchain podemos utilizar a un contrato dentro de nuestras funciones.

```
pragma solidity ^0.8.5;

contract Implementado {

    function asignarValor(uint) public returns (uint) {}

    function prueba() public pure returns (uint) {}

}

contract Existente  {
    Implementado dc;

    function Existing(address _t) public {
        dc = Implementado(_t);
    }

    function getA() public view returns (uint result) {
        return dc.prueba();
    }

    function setA(uint _val) public returns (uint result) {
        dc.asignarValor(_val);
        return _val;
    }
}
```

¡Muchas gracias!

¡Sigamos trabajando!