Ejercicios Solidity - DelegateCall / Proxies

Codingheroes

El objetivo de estos ejercicio es practicar el uso de *delegatecall* en situaciones reales, principalmente para el uso de Proxies. Todos los ejercicios se basarán en arreglar la implementación de una calculadora. En el ejercicio 1, crearemos nuestro propio proxy para cumplir este objetivo. En el ejercicio 2, utilizaremos el patrón UUPS. Finalmente, en el ejercicio 3 utilizaremos el patrón Transparent Proxy.

1 . El primer ejercicio consistirá en crear un proxy propio, sin utilizar ninguna plantilla ni patrones pre establecidos. Observaremos cómo gracias a los proxies podemos redirigir las llamadas a una nueva implementación. El ejercicio constará de tres contratos:

a. Un contrato llamado **Proxy.** Este contrato deberá tener:
   i. Una variable **implementación**: esta variable será la dirección a la cual redirigiremos las llamadas del proxy. Inicialmente, deberá apuntar al contrato **ImplementaciónV1**.
   ii. Una variable **owner**: será el dueño del contrato
   iii. Una función **upgrade()** que recibirá la dirección de la nueva implementación a la que apuntará el proxy (considera qué restricciones pueden ser importantes para esta función)
   iv. Una función de fallback que rediriga todas las llamadas a la implementación utilizando *delegatecall*
b. Un contracto llamado **ImplementaciónV1** (hecho por nosotros, lo podéis encontrar en el fichero CalculatorV1.sol)**.** Este contrato consiste en una implementación **incorrecta** de una calculadora.
c. Un contrato llamado **ImplementaciónV2**. Este contrato consistirá en arreglar la calculadora para que funcione de forma correcta.

El objetivo de este ejercicio es arreglar el contrato de calculadora incorrecto (ImplementaciónV1), para que el proxy pueda operar con normalidad. Tu test deberá hacer deploy del proxy apuntando inicialmente a la ImplementaciónV1. A continuación, deberás ejecutar el upgrade hacia la ImplementaciónV2. Deberás comprobar que llamar al proxy con las funciones *addition(), substraction(), multiplication() y division()* devuelve resultados correctos debido a que el proxy redirige a ImplementaciónV2 en vez de ImplementaciónV1.

2 . Para el segundo ejercicio, replicarás el ejercicio 1 utilizando el patrón UUPS. Para ello, deberás:

a. Utilizar la plantilla del proxy ERC1967 de OpenZeppelin. Este contrato actúa de la misma que el Proxy del ejercicio 1, sin necesidad de que tengamos que crear nosotros el proxy desde cero. Así, crearemos un contrato **Proxy** que herede de ERC1967, y esto será todo!
b. Las implementaciones de la calculadora deberán ser las mismas que en el ejercicio 1 (**ImplementaciónV1,** hecha por nosotros, e **ImplementaciónV2**, hecha por tí), con una pequeña modificación. Ambas implementaciones deberán heredar la plantilla UUPSUpgreadable.

El objetivo de este ejercicio será practicar la realización de upgrades utilizando UUPS. Recuerda que en el patrón UUPS los upgrades se realizan desde la implementación (no el proxy). El proxy simplemente actúa como un contrato que redirige llamadas a la implementación y que guarda los datos. Las implementaciones son las que contienen la lógica de hacer upgrade mediante la función *upgradeToAndCall()* que podemos encontrar en el código del UUPS. Esta función recibe dos parámetros: *newImplementation (*que será la nueva implementación a la que queramos hacer upgrade*),* y *data (*que deberemos dejar vacía*).* Deberás testear que el upgrade se realiza de forma correcta y la calculadora finalmente devuelve los valores correctos.