Lab 4 – Herencia, visibilidad y modificadores en Solidity

En esta sesión vamos a continuar estudiando las funcionalidades que nos ofrece Solidity como lenguaje de programación de alto nivel. Al ser similar a los lenguajes de programación orientada a objetos que conocemos, como Java, podemos definir *interfaces* y *contratos abstractos*. También veremos las distintas opciones de visibilidad y modificadores que existen en Solidity. Recuerda todo lo que hemos visto hasta ahora ya que lo seguiremos utilizando.

1. ¡A votar!

Una de las aplicaciones más comunes de las tecnologías blockchain son los sistemas de votaciones. En Ethereum son muy utilizados en las DAOs (*Decentralized Autonomous Organizations*). En este ejercicio vamos a modelar un sistema básico de votación. Para ello vamos a tener que crear contratos para representar la información relativa al sistema de votación (sustituiremos el método D'Hondt que se utiliza en España por un mecanismo proporcional muy sencillo), el mecanismo básico de votación que simule una sede electoral dirigida por un presidente de mesa y finalmente la creación de unas elecciones.

Vamos a definir el contrato `DhondtElectionRegion` que almacenará la información relativa al método de contabilización de votos de una región, pues en muchos sistemas electorales no todos los votos tienen el mismo valor. Las sedes y mesas electorales estarán representadas por el contrato abstracto `PollingStation`. El contrato `DhondtPollingStation` hereda de los contratos `PollingStation` y `DhondtElectionRegion` e implementará la sede de votación en cada una de las regiones (vamos a simplificar suponiendo que cada región tiene una sola sede electoral con un solo presidente de mesa). Por último el contrato `Election` creará las distintas sedes electorales y nos permitirá votar y hacer un recuento de los resultados.

Los partidos políticos estarán representados por un número entero sin signo desde 0 hasta el número de partidos menos uno. Las regiones estarán representadas por un número entero sin signo.

1.1. Contrato `DhondtElectionRegion`

Este contrato contiene el método de contabilización de votos. Aunque por el nombre se sugiere que aplica la ley D'Hondt, realmente es un mecanismo mucho más sencillo: a partir del identificador de región, asigna un peso que se aplicará a cada voto (las regiones con más población como Madrid tienen un peso menor; este es todo el parecido con el método D'Hondt). Para ello, debes crear en este contrato un mapping privado `mapping(uint => uint)private weights` que contendrá los pesos asignados a cada región. Puedes utilizar la siguiente función privada para darle valores de ejemplo desde el constructor:

```
function savedRegionInfo() private{
    weights[28] = 1; // Madrid
    weights[8]  = 1; // Barcelona
    weights[41] = 1; // Sevilla
    weights[44] = 5; // Teruel
    weights[42] = 5; // Soria
    weights[49] = 4; // Zamora
    weights[9]  = 4; // Burgos
    weights[29] = 2; // Malaga
}
```

Además, debes definir las siguientes variables de estado:

- **`uint regionId`**: valor constante que representa el identificador de región y que se da valor en el constructor.
- **`weights`**: mapping descrito antes.
- **`uint[] results`**: array interno con los resultados de la votación por cada partido. Debe tener tantos elementos como partidos se presentan a la elección.

Este contrato debe contener las siguientes funciones:

- **`constructor`**: debe recibir como parámetros el número de partidos que se presentan a la elección y el identificador de región.
- **`registerVote`**: función **interna** que registra un voto en el array `results`. Debe recibir como parámetro el partido al que se vota (**`uint`**). Debe comprobar que el identificador de partido es válido. A continuación debe acumular en `results` el resultado de aplicar un voto al peso fijado para la región que representa el contrato. Debe devolver un valor Booleano indicando si el partido al que se vota es válido o no.

1.2. Contrato `PollingStation`

Este es un contrato abstracto que representa una sede electoral (para simplificar, en nuestro sistema una sede electoral está formada por una única "urna" y un presidente de mesa). Este contrato es genérico y no tiene todas las funciones implementadas, pero debe incluir el código para la apertura y cierre de la sede electoral por parte del presidente. Debe tener las siguientes variables de estado:

- **`votingFinished`**: es una variable **pública** de tipo **`bool`** que contiene el valor **`true`** si el presidente ha cerrado la votación.
- **`votingOpen`**: es una variable **privada** de tipo **`bool`** que contiene el valor **`true`** si el presidente ha abierto la votación y todavía no la ha cerrado. Esta variable **no tiene el significado contrario a la anterior:** cuando se crea una sede electoral, ambas variables tienen el mismo valor **`false`**.
- Además, debes añadir variables de estado para almacenar quién es el presidente de la mesa.

Respecto a las funciones, deben implementarse las siguientes:

- **`constructor`**: debe recibir como parámetro el `address` del presidente de la mesa.
- **`openVoting`**: función externa sin parámetros que solo puede ejecutar el presidente de la mesa. Una mesa solo puede recibir votos si está abierta.
- **`closeVoting`**: función externa sin parámetros que solo puede ejecutar el presidente de la mesa. Con esta función se indica que ya no se pueden recibir más votos y que ha terminado la votación.

Además, este contrato abstracto debe declarar (sin implementar) las siguientes funciones:

- **`castVote`**: función externa para depositar un voto. Recibe como parámetro el identificador del partido al que se vota. Solo puede ejecutarse si la votación está abierta.
- **`getResults`**: función externa sin parámetros que devuelve los resultados de la votación (un array de **`uint`**)

Importante: Debes utilizar modificadores para comprobar precondiciones en estas funciones: por ejemplo, si la votación está abierta, o si el emisor de la transacción es el presidente.

1.3. Contrato `DhondtPollingStation`

Este contrato debe heredar de los dos contratos descritos anteriormente, `PollingStation` y `DhondtElectionRegion`. Representa un tipo concreto de sedes electorales que son las constituidas para una elección basada en el método D'Hondt. Debe implementar las siguientes funciones:

- **`constructor`**: debe recibir tres parámetros: el `address` del presidente de la mesa, el número de partidos que participan en la elección y el identificador de región (**`uint`**) de esta sede.
- **`castVote`**: implementa la función del mismo nombre declarada en `PollingStation`. Debe registrar el voto con la función heredada de `DhondtElectionRegion`. Si no es posible, debe revertir la ejecución con un texto explicativo.
- **`getResults`**: Debe devolver los resultados de la votación en esa sede electoral. Si no ha terminado la votación, debe revertir la ejecución con un texto explicativo.

1.4. Contrato `Election`

Por último, el contrato `Election` representa un proceso electoral. Debe almacenar lo siguiente: las sedes electorales que se creen para el proceso electoral; un registro de todos los votantes que han ejercido su derecho al voto; y por último el `address` de la autoridad que crea el contrato `Election`. Debes utilizar las estructuras de datos que consideres más eficientes. Recuerda que puedes crear estructuras de datos que contengan otros contratos.

Para comprobar que las funciones se pueden ejecutar, debes crear los siguientes modificadores:

- **`onlyAuthority`**: debe comprobar que el emisor de la transacción es la autoridad que crea el contrato.
- **`freshId(uint regionId)`**: dado un parámetro que contiene un identificador de región, debe comprobar que no se ha creado ninguna sede electoral con ese identificador. Solo vamos a permitir crear una sede electoral por cada región.
- **`validId(uint regionId)`**: dado un parámetro que contiene un identificador de región, debe comprobar que ya se ha creado una sede electoral con ese identificador.

Este contrato debe tener las siguientes funciones:

- **`constructor`**: debe recibir como parámetro el número de partidos que se presentan a las elecciones. Debe registrar el `address` de la autoridad administrativa que crea el contrato.
- **`createPollingStation`**: función externa que recibe un identificador de región y un `address` del presidente de mesa de esa sede. Esta función debe crear una sede electoral. Antes de ejecutarse debe comprobar que no hay ninguna sede ya creada para esa región, y que el emisor de la transacción es la autoridad que ha creado el contrato `Election`. Esta función debe devolver la dirección del contrato creado para la sede electoral.
- **`castVote`**: función externa para que cualquier votante deposite un voto. Recibe dos parámetros: el identificador de región del votante (que es el emisor de la transacción) y el partido al que vota. Debe comprobar que el identificador de la región es válido (es decir, que se ha creado una sede electoral para esa región) y que el votante no ha votado antes. Si las comprobaciones son correctas, debe depositar el voto utilizando la función correspondiente del contrato `DhondtPollingStation` de la región que se ha pasado como parámetro.
- **`getResults`**: función externa sin parámetros que calcula el resultado consolidado de las elecciones y solo puede ser ejecutada por la autoridad administrativa. Para ello, debe acumular los resultados de cada una de las sedes electorales. Si alguna de las sedes no ha terminado su votación, debe revertir la ejecución con un texto explicativo. Si todas las sedes han terminado la votación, debe devolver los resultados consolidados (un array **`uint`**`[]` con tantos elementos como partidos que se presentan a las elecciones, y como contenido el valor acumulado de todos los votos).

2. Ejemplo de uso

Una vez implementado nuestro sistema de votación, vamos a comprobar que funciona correctamente.

1. Asigna direcciones a los siguientes roles:

Role	Address
`Authority`	
`President1`	
`President2`	
`Voter1`	
`Voter2`	
`Voter3`	
`Voter4`	
`Voter5`	

2. Despliega el contrato `Elections` con la dirección `Authority`. La votación se creará con 3 partidos.

3. Crea dos sedes electorales (polling station) mediante la función `createPollingStation` y direcciones de `President1` y `President2` respectivamente. La primera sede corresponderá a Madrid mientras que la segunda a Teruel.

4. Llama con cada uno de los presidentes a la función `openVoting`.

5. Emite un voto con cada una de las siguientes direcciones mediante la función `castVote`:

   - `Voter1`: Partido 1 en Madrid.
   - `Voter2`: Partido 0 en Teruel.
   - `Voter3`: Partido 2 en Madrid.
   - `Voter4`: Partido 1 en Madrid.
   - `President1`: Partido 1 en Madrid.
   - `Voter5`: Partido 0 en Teruel.
   - `President2`: Partido 1 en Teruel.

6. Llama a la función `getResults` con la dirección de la autoridad administrativa. ¿Qué mensaje se obtiene al lanzar la ejecución?

7. Cierra las votaciones con ambos presidentes con la función `closeVoting`.

8. Llama a la función `getResults` con la dirección de uno de los presidentes. ¿Qué mensaje se obtiene?

9. Llama a la función `getResults` con la dirección de la autoridad administrativa. ¿Qué resultados se obtienen?

Partido	Resultado
Partido 0	
Partido 1	
Partido 2	