**Taller Práctico # 1**

**Tipo actividad:**

**Taller Práctico: Implementación de Patrones de Diseño en Contratos Inteligentes**

En el contexto de la tecnología blockchain, los contratos inteligentes son elementos fundamentales que permiten la ejecución automatizada y transparente de acuerdos digitales. Sin embargo, para desarrollar contratos inteligentes eficientes y seguros, es crucial comprender y aplicar los patrones de diseño adecuados. Este taller práctico se centra en la implementación concreta de estos patrones en contratos inteligentes, consolidando la comprensión teórica y desarrollando habilidades prácticas para aplicaciones reales.

A lo largo de cuatro horas, se explorarán ejercicios prácticos diseñados para abordar desafíos comunes en el diseño de contratos inteligentes. Se analizará la implementación de patrones clave, como el Factory Pattern, el Proxy Pattern y el State Machine Pattern, mediante ejemplos concretos y discusiones interactivas. Este enfoque permitirá comprender cómo estos patrones estructuran y optimizan el desarrollo de contratos inteligentes, mejorando su eficiencia y seguridad.

**Estructura del Taller: Instalación de Entorno de Trabajo**

**1. Introducción al Taller:**

**- Breve Repaso de los Patrones de Diseño:** Se realizará una revisión rápida de los principales patrones de diseño abordados en la sesión previa, destacando su importancia en el desarrollo de contratos inteligentes. Esto permitirá a los participantes refrescar sus conocimientos y establecer una base sólida para el taller práctico.

**- Objetivos y Estructura del Taller:** Se explicarán los objetivos del taller y la estructura general que seguirá. Los participantes comprenderán lo que se espera de ellos durante el taller y cómo se organizará el tiempo para lograr los mejores resultados.

**- Presentación del Entorno de Desarrollo:** Se realizará una presentación de las herramientas y plataformas que se utilizarán durante el taller. Esto incluirá la instalación y configuración de IDEs, frameworks y otras herramientas esenciales para el desarrollo de contratos inteligentes. Los participantes recibirán orientación detallada sobre cómo acceder y utilizar cada una de estas herramientas.

En el contexto del desarrollo de contratos inteligentes y aplicaciones blockchain, existen varios Entornos de Desarrollo Integrados (IDEs) que son ampliamente utilizados debido a sus características específicas y compatibilidad con los lenguajes de programación utilizados en blockchain. Algunos de los IDEs más populares incluyen:

* **Remix:** Es un IDE en línea que permite desarrollar, desplegar y probar contratos inteligentes en la red Ethereum. Ofrece una interfaz de usuario intuitiva y funciones de depuración integradas que facilitan el desarrollo de contratos inteligentes.
* **Visual Studio Code (VS Code) con extensiones para blockchain:** VS Code es un editor de código altamente personalizable que cuenta con una amplia gama de extensiones para el desarrollo en blockchain. Estas extensiones incluyen soporte para lenguajes como Solidity (utilizado en Ethereum) y herramientas para la interacción con redes blockchain.
* **Truffle Suite:** Truffle es un conjunto de herramientas que incluye un framework de desarrollo, un entorno de pruebas y una biblioteca de activos para Ethereum. Se integra bien con Visual Studio Code y ofrece características como la compilación, migración y prueba de contratos inteligentes de manera eficiente.
* **Ganache:** Ganache es un entorno de prueba personalizado que simula una red blockchain local para el desarrollo y la depuración de contratos inteligentes. Se puede integrar fácilmente con Truffle Suite y ofrece características como la visualización de registros, el control del estado de la cadena y la gestión de cuentas.
* **Embark:** Embark es un framework de desarrollo para Ethereum que proporciona herramientas para el desarrollo, despliegue y pruebas de aplicaciones descentralizadas y contratos inteligentes. Incluye un servidor de desarrollo integrado y funciones de compilación y despliegue automatizadas.

**Instalación de algunos IDEs:**

### **Instalación en Windows:**

#### **Ganache:**

1. Visita el sitio web oficial de Ganache (<https://www.trufflesuite.com/ganache>) y descarga la versión compatible con Windows.
2. Una vez completada la descarga, haz doble clic en el archivo de instalación para comenzar el proceso de instalación.
3. Sigue las instrucciones en pantalla para completar la instalación de Ganache en tu sistema.

#### **Truffle Suite:**

1. Abre tu terminal o línea de comandos y asegúrate de tener Node.js y npm instalados en tu sistema.
2. Ejecuta el siguiente comando para instalar Truffle globalmente:

npm install -g truffle

Tras la instalación, verifica que Truffle se haya instalado correctamente ejecutando el siguiente comando en tu terminal:

truffle version

###

### **Instalación en Linux:**

#### **Ganache:**

1. Visita el sitio web oficial de Ganache ( <https://www.trufflesuite.com/ganache> ) y descarga la versión compatible con Linux.
2. Abre tu terminal y navega hasta la ubicación donde se descargó el archivo de instalación.
3. Otorga permisos de ejecución al archivo descargado con el siguiente comando:

chmod +x ganache-<version>-linux-x86\_64.AppImage

Ejecuta Ganache con el siguiente comando:

./ganache-<version>-linux-x86\_64.AppImage

#### **Truffle Suite:**

1. Abre tu terminal y asegúrate de tener Node.js y npm instalados en tu sistema.
2. Ejecuta el siguiente comando para instalar Truffle globalmente:

npm install -g truffle

Verifica que Truffle se haya instalado correctamente ejecutando el siguiente comando en tu terminal:

truffle version

Con estos pasos, deberías tener Ganache y Truffle Suite instalados y listos para su uso en tu sistema Windows o Linux.

### **Ejercicio 1: Implementación del Factory Pattern**

En este primer ejercicio, los participantes trabajarán en grupos para implementar el Factory Pattern en un contrato inteligente específico. El objetivo es comprender cómo el Factory Pattern puede utilizarse para crear instancias eficientes de contratos inteligentes según sea necesario.

#### **Descripción del Ejercicio:**

Los participantes serán presentados con un escenario práctico donde se requiere la creación eficiente de instancias de contratos inteligentes. Por ejemplo, podrían enfrentarse a la necesidad de crear múltiples tipos de tokens en una plataforma de financiación descentralizada (DeFi).

#### **Pasos a Seguir:**

* **Formación de Grupos:** Los participantes se organizarán en grupos pequeños para facilitar la colaboración y el intercambio de ideas.
* **Análisis del Escenario:** Se presentará a los grupos un escenario práctico donde se requiere la creación eficiente de instancias de contratos inteligentes. Se discutirán los requisitos y las características del escenario para comprender completamente el problema a resolver.
* **Diseño del Contrato Inteligente:** Cada grupo diseñará el contrato inteligente utilizando el Factory Pattern para crear instancias según sea necesario. Se identificarán las funciones y variables necesarias, así como la lógica de implementación del Factory Pattern.
* **Desarrollo del Contrato Inteligente:** Los participantes utilizarán herramientas como Remix, Truffle o cualquier otro IDE de su elección para codificar el contrato inteligente. Se fomentará la colaboración y el intercambio de conocimientos entre los miembros del grupo.
* **Pruebas y Depuración:** Una vez desarrollado el contrato inteligente, los grupos realizarán pruebas exhaustivas para garantizar su funcionalidad y seguridad. Se identificarán y corregirán errores utilizando técnicas de depuración.
* **Presentación de Resultados:** Cada grupo presentará su contrato inteligente implementado y explicará cómo aplicaron el Factory Pattern para crear instancias de manera eficiente. Se facilitará una discusión abierta para compartir experiencias y aprendizajes entre todos los participantes.

### **Ejercicio 2: Aplicación del Proxy Pattern**

En este segundo ejercicio, los participantes implementarán el Proxy Pattern para controlar el acceso y la gestión de permisos en un contrato inteligente. El objetivo es comprender cómo el Proxy Pattern puede utilizarse para gestionar de manera segura y eficiente los permisos de acceso en un contrato inteligente.

#### **Descripción del Ejercicio:**

Los participantes serán presentados con un caso de uso donde se requiere restringir el acceso a ciertas funciones del contrato inteligente. Por ejemplo, podrían enfrentarse a la necesidad de permitir que solo ciertos usuarios autenticados ejecuten transacciones críticas.

#### **Pasos a Seguir:**

* **Presentación del Caso de Uso:** Se presentará a los participantes un caso de uso donde se requiere restringir el acceso a ciertas funciones del contrato inteligente. Se discutirán los requisitos y las características del caso de uso para comprender completamente el problema a resolver.

**Caso de Uso: Sistema de Votación Electrónica**

En un sistema de votación electrónica, es fundamental garantizar que solo los votantes autorizados puedan emitir sus votos y que estos votos sean registrados de manera segura y verificable. En este contexto, se necesita un contrato inteligente que gestione el proceso de votación y restrinja el acceso a las funciones de emitir votos y contar los resultados únicamente a los votantes registrados.

**Requisitos y Características del Caso de Uso:**

**Registro de Votantes:** El contrato inteligente debe permitir el registro de votantes autorizados, verificando su identidad y almacenando sus credenciales de manera segura en la blockchain.

**Emisión de Votos:** Solo los votantes registrados deben poder emitir votos válidos. El contrato inteligente debe verificar la identidad del votante y permitirle emitir un voto solo una vez. Además, el voto debe ser registrado de forma inmutable en la blockchain para garantizar su integridad.

**Conteo de Votos:** Una vez finalizado el período de votación, el contrato inteligente debe contar los votos emitidos y determinar los resultados de la elección. Solo las autoridades electorales designadas deben tener acceso a esta función para evitar manipulaciones o fraudes en el proceso de conteo.

**Restricción de Acceso:** Las funciones de emitir votos y contar los resultados deben estar restringidas únicamente a los votantes registrados y a las autoridades electorales designadas. El contrato inteligente debe implementar mecanismos de control de acceso para garantizar que solo las partes autorizadas puedan ejecutar estas funciones.

* **Diseño del Contrato Inteligente:** Cada participante diseñará e implementará el contrato inteligente utilizando el Proxy Pattern para gestionar de manera segura y eficiente los permisos de acceso. Se identificarán las funciones que requieren restricciones de acceso y se diseñará la lógica de implementación del Proxy Pattern.
* **Desarrollo del Contrato Inteligente:** Los participantes utilizarán herramientas como Remix, Truffle o cualquier otro IDE de su elección para codificar el contrato inteligente. Se fomentará la colaboración y el intercambio de conocimientos entre los participantes.
* **Pruebas y Depuración:** Una vez desarrollado el contrato inteligente, los participantes realizarán pruebas exhaustivas para garantizar su funcionalidad y seguridad. Se identificarán y corregirán errores utilizando técnicas de depuración.
* **Presentación de Resultados:** Cada participante presentará su contrato inteligente implementado y explicará cómo aplicaron el Proxy Pattern para controlar el acceso y la gestión de permisos. Se facilitará una discusión abierta para compartir experiencias y aprendizajes entre todos los participantes.

**Ejercicio 3: Integración del State Machine Pattern**

En este ejercicio, los participantes trabajarán en la integración del State Machine Pattern en un contrato inteligente que involucra múltiples estados y transiciones. El objetivo es modelar estados y transiciones de manera clara y eficiente para manejar la lógica empresarial y los estados complejos.

**Descripción del Ejercicio:**

* **Casos de Uso Complejos:** Se presentará a los participantes un caso de uso complejo que requiere el manejo de múltiples estados y transiciones en un contrato inteligente. Por ejemplo, podría ser un contrato de alquiler de propiedades que incluya estados como "Disponible", "Reservado", "En Uso" y "Terminado", con transiciones entre estos estados según las acciones realizadas por el arrendador y el arrendatario.

**Caso de Uso: Contrato de Alquiler de Propiedades**

En este caso de uso, se aborda la implementación de un contrato inteligente para el alquiler de propiedades, que implica el manejo de múltiples estados y transiciones en función de las acciones realizadas por el arrendador y el arrendatario.

**Estados:**

**Disponible:** El estado inicial de la propiedad cuando está disponible para ser alquilada.

**Reservado:** Cuando un posible arrendatario muestra interés en la propiedad y la reserva para su futura ocupación.

**En Uso:** Una vez que el arrendatario ha firmado el contrato de arrendamiento y ha comenzado a utilizar la propiedad.

**Terminado:** Al finalizar el período de alquiler, la propiedad vuelve al estado de Disponible para futuros arrendatarios.

**Transiciones:**

**Reserva:** Transición de Disponible a Reservado cuando un arrendatario muestra interés en la propiedad y la reserva.

**Confirmación de Contrato:** Transición de Reservado a En Uso cuando el arrendatario confirma y firma el contrato de arrendamiento.

**Terminación del Contrato:** Transición de En Uso a Terminado al finalizar el período de alquiler.

**Acciones del Arrendador:**

**Aprobar Reserva:** El arrendador puede aprobar o rechazar la solicitud de reserva de un posible arrendatario.

**Firmar Contrato:** Después de la aprobación, el arrendador firma el contrato de arrendamiento con el arrendatario.

**Finalizar Contrato:** Al finalizar el período de alquiler, el arrendador confirma la terminación del contrato y la propiedad vuelve a estar disponible.

**Acciones del Arrendatario:**

**Reservar Propiedad:** El arrendatario muestra interés en la propiedad y la reserva para su futura ocupación.

**Confirmar Contrato:** Después de la aprobación del arrendador, el arrendatario firma el contrato de arrendamiento.

**Entregar la Propiedad:** Durante el período de alquiler, el arrendatario ocupa y utiliza la propiedad según lo acordado en el contrato.

* **Análisis de Requisitos:** Los participantes analizarán detalladamente los requisitos del caso de uso para comprender los estados involucrados, las transiciones entre ellos y las reglas de negocio asociadas. Esto les permitirá definir claramente la estructura del State Machine Pattern a implementar en el contrato inteligente.
* **Diseño e Implementación:** Con base en los requisitos analizados, los participantes diseñarán e implementarán el contrato inteligente utilizando el State Machine Pattern. Utilizarán un enfoque modular para definir los diferentes estados, las transiciones entre ellos y las condiciones que gobiernan dichas transiciones. Se centrarán en crear un contrato inteligente, claro, legible y fácil de mantener.
* **Pruebas y Validación:** Una vez completada la implementación, los participantes realizarán pruebas exhaustivas del contrato inteligente para garantizar su correcto funcionamiento en diferentes escenarios. Se validarán las transiciones entre estados y se verificará que la lógica empresarial se aplique correctamente en cada etapa del proceso.