Laboratorio Modulo 1 Desarrollo Full Stack

POO y Patrones de Diseño

Docente: Julián Felipe Latorre

Nombre: _____

Documento: _____

Objetivo del Laboratorio:

Evaluar los conocimientos teóricos y prácticos de los estudiantes en los siguientes temas:

- Principios de Programación Orientada a Objetos (POO)
- Implementación de POO en JavaScript
- Diseño y modelado de clases y objetos
- Patrones de diseño en POO

Parte 1: Principios de POO

Definición y Principios de POO

1. Define qué es la Programación Orientada a Objetos (POO).
2. Explica los siguientes principios fundamentales de la POO:
    - Encapsulamiento
    - Herencia
    - Polimorfismo
    - Abstracción

Ejemplo de POO en un Lenguaje Teórico

3. Describe mediante texto y usando pseudo-código cómo implementarías una clase `Animal` con las siguientes propiedades y métodos:
    - Propiedades: `nombre` (string), `edad` (número)
    - Métodos: `hablar()` (método abstracto)

    Luego, proporciona una implementación concreta para las clases `Perro` y `Gato`, donde `hablar()` retorna "Guau" y "Miau" respectivamente.

Parte 2: Implementación de POO en JavaScript

Implementación Básica

1. ¿Cómo se define una clase en JavaScript? Proporciona un ejemplo de una clase `Persona` con propiedades `nombre` y `edad`, y un método `presentarse()` que imprima una presentación en la consola. Propón como implementar la funcionalidad para conversar e iniciar una conversación entre dos `Personas.`

Creación de Clases y Objetos

2. Implementa una clase `Vehiculo` en JavaScript con las siguientes características:
    - Propiedades: `marca`, `modelo`, `año`
    - Método: `detalles()` que retorne una cadena con los detalles del vehículo.

   Luego, crea dos instancias de `Vehiculo` diferentes y llama al método `detalles()` para ambas.

3. Crea una clase abstracta `Animal` con propiedades `nombre` y `edad`, y métodos abstractos `hablar()` y `moverse()`. Luego, implementa las subclases `Perro` y `Gato` que hereden de `Animal`, donde `hablar()` retorna "Guau" para `Perro` y "Miau" para `Gato`, y `moverse()` retorna "El perro está corriendo" y "El gato está caminando sigilosamente" respectivamente. Instancia objetos de ambas clases y muestra los resultados en un documento HTML de llamar a `hablar()` y `moverse()`.

Parte 3: Diseño y Modelado de Clases y Objetos

Diseño y Modelado

1. Explica la importancia del diseño y modelado de clases y objetos en la POO. ¿Qué beneficios aporta un buen diseño de clases y objetos?

Creación de Diagramas UML

2. Dibuja un diagrama UML para el siguiente sistema:
    - Clases: `Libro`, `Autor`, `Editorial`
    - Relaciones:
        - Un `Autor` puede escribir varios `Libros`.
        - Un `Libro` es publicado por una única `Editorial`.
        - Una `Editorial` puede publicar varios `Libros`.

   Asegúrate de incluir las propiedades y métodos relevantes para cada clase.

Principios SOLID

3. Implementa una clase `Empleado` que cumpla con el principio de responsabilidad única (SRP) al separar las responsabilidades de cálculo de salario y generación de reportes en clases distintas; aplica el principio abierto/cerrado (OCP) al permitir que la clase `Empleado` pueda ser extendida para diferentes tipos de empleados (como `EmpleadoTiempoCompleto` y

`EmpleadoMedioTiempo`) sin modificar su código base; utiliza el principio de sustitución de Liskov (LSP) asegurando que las subclases puedan reemplazar a la clase base `Empleado` sin alterar el funcionamiento del programa; emplea el principio de segregación de interfaces (ISP) creando interfaces específicas para cada tipo de empleado, evitando interfaces monolíticas; y finalmente, sigue el principio de inversión de dependencia (DIP) inyectando dependencias a través del constructor para facilitar la prueba y el mantenimiento del código.

Parte 4: Patrones de Diseño en POO

1. Define qué es un patrón de diseño en la POO y menciona tres ejemplos comunes.

Implementación de un Patrón de Diseño

2. Elige uno de los siguientes patrones de diseño e implementa un ejemplo en JavaScript:
    - **Singleton:** Implementa una clase `Configuracion` que solo permita una única instancia para manejar la configuración de una aplicación.
    - **Factory:** Implementa una clase `FabricaDeHerramientas` que cree instancias de `Pica` o `Pala` basadas en un parámetro de entrada.
    - **Observer:** Implementa un sistema de notificación donde múltiples `Observadores` pueden suscribirse a cambios en un `Sujeto`.

Conclusión y Reflexión

1. Reflexiona sobre lo aprendido en este laboratorio. ¿Qué conceptos encontraste más desafiantes y por qué?
2. ¿Cómo aplicarías estos conceptos y patrones de diseño en el proyecto del curso?

Recursos y Material de Apoyo:

- **Documentación de JavaScript:** Referencia oficial de Mozilla Developer Network (MDN) sobre clases y objetos en JavaScript. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
- **Herramientas UML:** Herramientas en línea como Lucidchart (https://www.lucidchart.com/), draw.io (https://app.diagrams.net/), Plan Text (https://www.planttext.com/) o software como Visual Paradigm para la creación de diagramas UML
- **Ejemplos de Patrones de Diseño:** Libros y recursos en línea sobre patrones de diseño, como "Dive into Design Patterns" (**Shvets, Alexander.** Refactoring.Guru, 2019) (https://refactoring.guru/)