EVALUACIÓN DEL MÓDULO #4

PROYECTO: GESTOR INTELIGENTE DE CLIENTES (GIC)

1. Situación Inicial

Unidad solicitante: Empresa Solution Tech (Startup de tecnología) .

La empresa SolutionTech ha detectado ineficiencias y problemas de seguridad en su gestión actual . Si bien el almacenamiento definitivo se resolverá en futuras etapas (Bases de Datos), la gerencia exige urgentemente un sistema que audite y registre las operaciones que ocurren en la plataforma para asegurar la trazabilidad.

La propuesta: Desarrollar el núcleo lógico de una plataforma de gestión de clientes en Python, basada estrictamente en Programación Orientada a Objetos (POO). El sistema debe permitir la gestión de clientes en memoria y, fundamentalmente, generar un reporte automático (Log) en un archivo de texto (`.txt`) que registre cada acción realizada (creación, eliminación o errores), garantizando así un historial de actividad .

2. Nuestro Objetivo

Desarrollar una aplicación de consola en Python que gestione clientes aplicando reglas de negocio y persistencia de logs, asegurando una estructura modular y robusta ante fallos.

Objetivos Específicos:

- Diseñar un modelo UML detallado del sistema .
- Implementar un sistema de gestión aplicando herencia, polimorfismo y encapsulación .
- Incorporar validaciones de datos y manejo de excepciones personalizadas .
- Implementar Manejo de Archivos: Crear un sistema de registro de actividad que escriba en un archivo externo (`registro.txt`) cada operación exitosa o error capturado .

3. Requerimientos Funcionales

El sistema debe cumplir con las siguientes funcionalidades esenciales:

A. Gestión de Clientes (Lógica POO):

- Gestión: Creación, modificación y listado de clientes en memoria .
- Tipos de Clientes (Herencia): Implementar una clase padre `Cliente` y subclases (ej. `ClienteRegular`, `ClienteCorporativo`) con atributos diferenciados .
- Métodos: Uso de `__str__` para mostrar datos y encapsulamiento en atributos sensibles .

B. Registro de Actividad (Persistencia en Archivos):

- Logs de Operación: Cada vez que se crea o elimina un cliente, el sistema debe agregar una línea en un archivo `registro.txt` indicando la acción.
    - *Ejemplo:* `"Cliente Juan Pérez agregado exitosamente."`
- Logs de Errores: Si ocurre un error (ej. validación fallida), debe guardarse en el mismo archivo.
    - *Ejemplo:* `"ERROR: Intento de crear cliente con RUT inválido."`
- Persistencia: El archivo no debe sobrescribirse en cada ejecución, debe acumular el historial (modo `append`).

C. Validaciones y Errores:

- Validar datos de entrada (ej. formato de email, teléfono, RUT) .
- Utilizar bloques `try-except` para evitar que el programa se cierre inesperadamente .

4. Especificaciones Técnicas

- Lenguaje: Python 3 .
- Paradigma: Orientación a Objetos (Clases, Instancias, Herencia) .
- Manejo de Archivos: Uso de funciones nativas `open()`, `write()`, y manejo de contextos `with` para la generación del `.txt` .
- Interfaz: Menú de opciones en consola (CLI)..
- Estructura: Código organizado y limpio, siguiendo buenas prácticas (PEP8).

5. ¿Qué vamos a validar? (Criterios de Evaluación)

Se evaluará la correcta aplicación de los contenidos vistos en el Módulo 4:

- Lógica de Objetos: Correcta definición de clases y herencia .
- Coherencia UML: Que el código corresponda al diagrama diseñado .
- Manejo de Archivos (Logs): Que el archivo `registro.txt` se cree y actualice correctamente con las acciones del usuario .
- Robustez: Que el sistema capture excepciones y las registre en el log en lugar de fallar .

6. Entregables

El estudiante deberá entregar un archivo comprimido o repositorio conteniendo:

1. Código Fuente: Scripts `.py` con la solución (separando lógica de clases y ejecución).
2. Archivo de Log generado: Un ejemplo del archivo `registro.txt` creado por su programa.
3. Diagrama de Clases (UML): Esquema visual de la estructura del software .
4. Breve Informe en video: Explicación de las validaciones implementadas y cómo leer el log de actividad.