Consigna 3

Motivación:

Esta actividad tiene la finalidad de definir clases con relaciones entre ellas, de modo de resolver un requerimiento sencillo e implementarla en lenguaje C++, con la restricción de integrar lo nuevo con clases preexistentes, propias y/o de terceros.

Un aspecto a considerar en el desarrollo de software orientado a objetos, es el uso (habitual) de clases auxiliares.

Entre ellas, las clases destinada a registrar y/o administrar mensajes del sistema, errores, advertencias, información de depuración, etc.

En general, esta clase se denomina Logger (o registrador) y es usada como herramienta base para tareas de depuración y/o de optimización.

Una característica de la clase es la de definir diferentes niveles de registro, en función del tipo de mensaje y/o evento que se registra. Por ejemplo:

- DEBUG – Para mensajes de depuración detallada.
- INFO – Para mensajes informativos generales.
- WARNING – Para advertencias sobre situaciones inesperadas pero no críticas.
- ERROR – Para errores graves que no detienen la aplicación.
- CRITICAL – Para errores críticos que pueden detener el sistema.

Otro aspecto del desarrollo a considerar, es el de aprovechar las referencias al conocimiento ya adquirido y las buenas prácticas ya probadas. Algo de esto se encuentra detallado en los patrones de software.

Un patrón (pattern) es una solución ya probada y aplicable a un problema que se presenta una y otra vez en el desarrollo de distintas aplicaciones y en distintos contextos. Es importante destacar que un patrón no es en general una solución en forma de código directamente "listo para usar", sino más bien una descripción de cómo resolver el problema y de ante qué circunstancias es aplicable. Podría decirse que, un patrón es una regla de tres partes, la cual expresa una relación entre un cierto contexto, un problema y una solución.

Los patrones suelen agruparse en colecciones, según diferentes criterios (según la etapa o ámbito del desarrollo, según su finalidad, según el autor u organización inicial).

Una de las categorías son los patrones de diseño. Estos, ayudan a diseñar soluciones de software, definiendo comportamientos comunes que suelen encontrarse en distintos componentes de software

Particularmente, el patrón Singleton es un patrón de diseño que restringe la instanciación de una clase a un único objeto. Esto es útil cuando se necesita exactamente un objeto para coordinar las acciones de todo el sistema.

Requerimientos:

Se necesita integrar una clase auxiliar, que permita realizar un registro de los eventos que ocurran durante la ejecución de programas, almacenando los datos en un archivo de texto.

Para ello, otros integrantes del equipo de desarrollo han provisto de una clase Logger.

Para el sistema en curso, se desea reutilizar el código ya desarrollado para el primer requerimiento (clase File_NNN).

Para el sistema futuro, se sabe que, además de los eventos internos, para los cuales se requiere conocer en qué modulo se produjo (y eventualmente también en qué línea de programa) deberá registrar eventos que surgen de peticiones externas, para los cuales se requiere conocer un ID de usuario y un ID de dispositivo/aplicación remoto desde donde se solicitan servicios.

Esta clase auxiliar debe ser capaz también de ofrecer la información almacenada en formato CSV, JSON o XML según se necesite, ya sea de manera completa, por rango de fechas en que ocurrieron los eventos, y/o por tipo de evento y/o por usuario.

Para el requerimiento actual, usar un módulo principal capaz de construir los objetos que se requieran y realizar una secuencia de prueba significativa.

La implementación Orientada a Objetos realizada debe separar la capa de presentación/control de la de modelo.

Es conveniente que la clase implementada utilice excepciones para gestionar los errores y/o devolver mensajes destinados al usuario.

Procedimiento general:

1) Diseñar en Umbrello un modelo OO usando UML, que muestre y defina la/s clase/s fundamentales. Realice las mismas consideraciones que las utilizadas en consignas previas.

2) Dado que se debe aprovechar el modelo y las definiciones para la clase archivo desarrollada anteriormente, analice al menos 3 relaciones diferentes entre aquella y la clase Logger provista.

3) Generar los módulos que correspondan, con el código base en lenguaje C++, para uno de los modelos propuestos.

4) Editar y completar la implementación de modo de obtener una aplicación funcional que resuelva el requerimiento.

5) Realizar un informe similar a lo ya pautado, incluyendo sus comentarios al análisis de relaciones realizado y el modelo adoptado.

6) A partir del modelo implementado, analice y agregue al informe un detalle con los cambios que debería hacer en la clase para que cumpla con el patrón Singleton (la implementación de esta característica es opcional)

7) Preparar una carpeta para entrega, similar a lo ya pautado.

Recursos complementarios (disponibles en el aula virtual)**:**

- Apuntes de clase.
- Referencias de las consignas previas.
- Código de ejemplo (singleton_demo)
- Libro: Patrones de Diseño de Gamma, Helm, Johnson y Vlissides