PARTE 1. SOLID

Realizado por: Brais Domínguez Álvarez y Silvia Rodríguez Iglesias.

- **Identifica las diferentes responsabilidades que existen en este programa. Haz una lista con ellas.**
    1. Leer línea(s) de fichero.
    2. Leer línea(s) de un origen.
    3. Escribir línea(s) a fichero.
    4. Escribir línea(s) a un destino.
    5. Pedir 2 rutas de fichero a usuario.
    6. Pedir 1 fichero a usuario.
    7. Transformar 1 línea de tabuladores a XML.
    8. Transformar 1 línea de un formato a otro.
    9. Recorrer fichero de entrada, transformando y escribiendo en destino.
    10. Informar de errores al abrir fichero.

- **Refactoriza la aplicación para que:**
    - **La interfaz de usuario pueda ser reutilizada en un proyecto que haga otra cosa dados dos ficheros (la interfaz se encarga de pedir los ficheros). ¿Qué patrón o patrones SOLID has empleado, cómo y por qué?**

      En este cambio vemos presente el patrón OCP, ya que sin cambiar el código fuente (será reutilizable tal cual en otro proyecto), cambia lo que hacen los módulos. Para añadir nuevas funcionalidades empleamos el código ya existente añadiéndole nuevas clases, es decir, está abierto para extensión pero cerrado para ser modificado.

      Como esta clase es empleada únicamente para pedir dos ficheros, implementa una sola responsabilidad, por lo que cumple el patrón SRP.

    - **El origen de datos (ahora ficheros) pueda ser distinto y/o el destino de los datos también. ¿Qué patrón o patrones SOLID has empleado, cómo y por qué?**

      En este caso también vemos presente los patrones OCP y SRP. OCP ya que al crear una interfaz Writer y Reader para el origen/destino de escritura y lectura, se podrá ampliar código para implementarlo de diferentes formas sin cambiar el código de esas dos clase. Para añadir nuevas funcionalidades empleamos el código ya existente añadiéndole nuevas clases, es decir, está abierto para extensión pero cerrado para ser modificado.

      Como estas clases serán empleadas únicamente para pedir leer y escribir en un origen/destino, implementa una sola responsabilidad, por lo que cumple el patrón SRP.

      También cumple el patrón ISP, ya que se separan en varias interfaces, no siendo necesario emplear todas las partes. De esta manera se puede leer sin tener que escribir.

    - **La transformación que se hace de la entrada pueda ser otra representación basada en texto cualquiera (no a XML). ¿Qué patrón o patrones SOLID has empleado, cómo y por qué?**

      En este caso también vemos presente los patrones OCP y SRP. OCP ya que al crear una interfaz Transformer para la transformación de un formato a otro, se podrá ampliar código para implementarlo de diferentes formas sin cambiar el código de esa clase. Para añadir nuevas funcionalidades empleamos el código ya existente añadiéndole nuevas clases, es decir, está abierto para extensión pero cerrado para ser modificado.

Como esta clase será empleada únicamente para transformar de un formato a otro, implementa una sola responsabilidad, por lo que cumple el patrón SRP.

También cumple el patrón ISP, ya que se separa en una interfaz del resto del código, permitiendo realizar únicamente la transformación.

Otro de los patrones importantes que cumple es DIP, ya que al pasar la responsabilidad de realizar la conversión completa a Converter, el cual hace uso de todas las interfaces, permite que Converter no dependa de los módulos de bajo nivel.

Una vez refactorizada,

- **Elabora una tabla para documentar las responsabilidades de las clases**

Clase	Responsabilidad
FileReader	Leer línea(s) de fichero.
Reader	Leer línea(s) de un origen.
FileWriter	Escribir línea(s) a fichero.
Writer	Escribir línea(s) a destino.
FileUserInterface	Pedir 2 rutas de fichero a usuario.
ConverterApp	Informar de errores al abrir fichero.
FileUserInterface	Pedir 1 fichero a usuario.
TABToXMLTransformer	Transformar 1 línea de tabuladores a XML.
Transformer	Transformar 1 línea de un formato a otro.
Converter	Recorrer fichero de entrada, transformando y escribiendo en destino.

- **Elabora una tabla para documentar el principio OCP (Open Closed Principle) en tu código. Una clase que respeta el principio OCP, está cerrada para modificación pero abierta para extensión, siempre centrándonos en una modificación futura concreta. Por lo tanto, la tabla debe contener estas columnas.**

Clase	Modificación posible	Punto de extensión
Converter	Cambio en el origen de las líneas.	Interfaz Reader
Converter	Cambio en el destino de las líneas.	Interfaz Writer
Converter	Cambio del formato (tanto a transformar como al que se transformará).	Transformer

- **Modifica la aplicación para que la salida se produzca por pantalla y no a fichero. ¿Tuviste que cambiar código existente a mayores que el método *main*? Describe brevemente la modificación.**

Se ha añadido una clase ConsoleWriter que implementa a Writer y, en el método main se crea una instancia a dicha clase en lugar de a FileWriter. En el writer de ConsoleWriter se imprime por pantalla.