Sistema de archivos

Un equipo de desarrollo acaba de implementar un novedoso y revolucionario sistema de archivos: un conjunto de objetos que nos permiten abrir y cerrar archivos (binarios), y leerlos o escribirlos secuencialmente. Sin embargo, no se esmeraron mucho en que la interfaz entrante al sistema sea fácil de usar:

```
public interface LowLevelFileSystem {
  int openFile(String path);

  void closeFile(int fd);
  int syncReadFile(int fd, byte[] bufferBytes, int bufferStart, int bufferEnd);
  void syncWriteFile(int fd, byte[] bufferBytes, int bufferStart, int bufferEnd);
  void asyncReadFile(int fd, byte[] bufferBytes, int bufferStart, int bufferEnd,
      Consumer<Integer> callback);
  void asyncWriteFile(int fd, byte[] bufferBytes, int bufferStart, int bufferEnd,
                      Runnable callback);
}
```

Como ven esta interfaz entrante (representada por una interface Java) cumple lo solicitado, pero presenta un bajo nivel de abstraccion y no es por tanto fácil de usar.

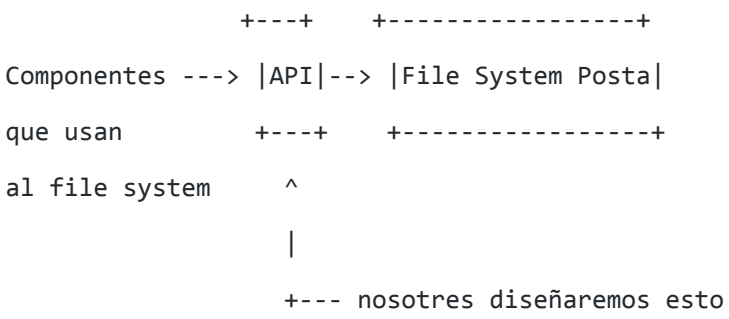
Entonces nos solicitaron que diseñemos una mejor interfaz entrante a este sistema, que pueda usar un programador (una API, application programming interface) que sea de mayor nivel de abstracción y más simple de usar. Esta estará conformada por uno o más componentes (interfaces, objetos, clases, etc) que tendremos que diseñar.

Ademas, nos pidieron que implementemos esta intefaz de forma que todas las operaciones que se hagan contra esta API terminen ejecutando las operaciones de la interfaz de bajo nivel del sistema de archivos.

Sin embargo, los requerimientos son un poco abiertos. Nos han señalado las siguientes cosas:

- Este API debe ser "fácil" de usar para un programador: debería ser clara, aprovechar el paradigma de objetos, ocultar detalles que no nos aportan y presentar buenas abstracciones.
- No es necesario que exponga una única interface tipo fachada.
- Debe usar a la interfaz entrante original que nos dieron, pero no modificarla.
- Debe ser robusta, presentando un buen manejo de errores
- Esta API debe permitr como mínimo:
    - abrir un archivo para lecto escritura

    - cerrar un archivo
    - escribir sincrónicamente un bloque de n bytes de archivo
    - leer sincrónicamente un bloque de n bytes de un archivo
    - leer asincrónicamente un bloque de n bytes de un archivo
    - escribir asincrónicamente un bloque de n bytes de un archivo
- ¡OJO! Es importante que el API tenga un buen manejo de errores
- Esta API que diseñaremos será básicamente un adaptador, es decir, no agregará funcionalidad al sistema de archivos original, sino que tan solo expondrá una mejor interfaz entrante. Es decir, ahora aquella interfaz entrante original será para nosotres la interfaz saliente del pequeño sistema adaptador que vamos a diseñar.



Entonces, tenemos como tarea "embellecer" al sistema de archivos. Y como segunda tarea, dar un ejemplo de uso del API para los siguiente casos:

- Tenemos que leer de un archivo 3 campos: el primero de 4 bytes (C0), el segundo de 1 byte (C1), el tercero de 5 bytes (C2). Y escribir en otro archivo C0, un bloque 0x0, 0x10, 0x0, y luego C1 y C2.
- Tenemos que leer un archivo completo, y escribirlo en otro archivo, en bloques de igual tamaño parametrizable.

Los ejemplos también tenemos que plantearlos nosotres, en forma de tests.

Finalmente, como nuestro cliente es bastante quisquilloso quiere ver formas alternativas de solucionar las lecturas sincrónicas y asincrónicas, para compararlas y ver cuales le gustan más.

Bonus

Opcionalmente, esta interfaz debería permitir:

- Saber si una ruta (path) denota un archivo regular

- Saber si una ruta (path) denota un directorio
- Saber si una ruta (path) existe (sin importar qué sea)