Enfoque adoptado a la resolución de la Tarea 2

Prueba que releva las nociones de procesado de datos adquiridas en la tarea 1, continuamos con acceso a ficheros.

En este caso vamos a trabajar con otro fichero en formato tabular, pero esta vez será algo más denso (64685 líneas para ser más exacto). Cada línea tiene una extensión al principio, que puede ser XXXX_CAB o XXXX_DAT, CAB significa “cabecera” y aquí va el nombre de cada columna, mientras que en DAT (que significa “datos”) van todos los datos que van en las cabeceras, por eso decimos que este fichero tiene tantos bloques como cabeceras contenga.

Primeras cosas hechas:

He reciclado algunas partes del código de tarea1 como el método buscar(), con esto también dejo claro que aunque mi enfoque sobre POO aún no está 100% adaptado va a seguir por este camino y en la medida de lo posible sufrirá mejoras.

También he creado otros dos métodos llamados contarCabeceras() y contarExtension(), contarCabeceras() funciona sobre el objeto sin ningún parámetro pasado, cuenta todas las líneas con extensión XXXX_CAB que encuentre en el fichero para comprobar los bloques existentes en ese fichero. Por otro lado contarExtensión(String) si funciona sobre el objeto con un parámetro, este parámetro es exactamente lo que hay después de XXXX_ (ya sea CAB, DAT o cualquier otro si existiera). Cabe mencionar también que el método toString que hasta ahora nos ha servido para imprimir el fichero por pantalla ha sufrido un traspaso y ahora para imprimir el fichero uso el método imprime(nombre_objeto).

Enfoque aplicado para los siguientes procedimientos:

Ahora debo de hacer que cada bloque (cabecera CAB con sus datos DAT) de ese objeto se almacene en un fichero de texto independiente. Con esto también debo de entender que mi enfoque sobre POO en el cual los objetos eran el nombre del fichero no me valía como tal, así que ahí fue cuando implementé el uso de la clase File, leyendo y testeando algunas de sus opciones he aprendido cosas como: Crear carpetas (también comprobar si no existen para así crearlas), asociar un fichero con un objeto tipo File y así también entender la representación de los objetos File en la sintaxis Java, uso del FileWriter y el buffer por medio de BufferedWriter para su escritura.

Por otra parte los ficheros deben llevar un nombre prediseñado que indica el bloque seguido de la fecha actual (FACT_20180403 por ejemplo) en formato “aaaammdd”, así que una vez más indagando implementé unas nuevas clases llamadas “java.util.Date” y “java.text.DateFormat” con **Date** creo un objeto con la fecha actual y con **DateFormat** le doy el formato que previamente mencionamos (aaaammdd), con dateFormat.format(date) hacemos de esa fecha un String donde queramos usarlo.

<u>El método funciona de la siguiente manera:</u>

Leo línea por línea (gracias al buffer) el fichero, como el fichero tiene una estructura bastante bien marcada y como cada línea cumplen el mismo criterio, con linea**.substring(5, 8)** he tomado una delimitación de lo que necesito de la línea tomada en ese momento en

cuestión y hemos obtenido lo que nos interesa que es la **extensión** de esa línea **(CAB o DAT),** esto, pasando por una condición específica para cada una de las extensiones y aplicando las correspondientes sentencias de escritura mediante buffer de la línea obtenida del fichero original en un fichero que previamente ha sido creado y clasificado por su nombre.