Examen parcial Programación III – Nov. 2018

- En la web de la asignatura ("Exámenes y Evaluación") tienes los ficheros con los que se va a trabajar en un archivo comprimido. Crea un proyecto en eclipse y descomprime ese fichero en la carpeta src de ese proyecto.
- Para entregar el examen, comprime todo el directorio src, y renombra el fichero (.zip o .rar) con tu dni, nombre y apellido. Asegúrate que tiene los ficheros que has modificado.
- Al acabar el examen, entrega en la tarea correspondiente de ALUD.

1. Planteamiento

Partimos de un pequeño sistema de gestión de seguimiento de unas sesiones realizadas en varios centros escolares. La clase `Main.java`, funcional, lanza una ventana en la que se puede ver la lista de centros, y al pulsar el botón "Carga feedback" se cargan las sesiones realizadas y se calculan los datos agregados (número de sesiones y estudiantes, valoraciones medias, etc.). Clases esenciales:

- **`CentroEd`**. Clase que modela los datos utilizados por cada centro educativo.
- **`Tabla`**. Clase que permite gestionar tablas de datos de cualquier tipo, organizadas por filas y columnas, con valores strings. Permite además visualizarse fácilmente en una *JTable*, al devolver un modelo de datos partiendo de su información.
- **`ConvertibleEnTabla`**. Interfaz que define el funcionamiento de cualquier clase que quiera poder usarse fácilmente como una tabla (al permitir el acceso a sus datos estructurados como columnas).

2. Tareas

Se pide que realices las siguientes tareas (su puntuación sobre 10 entre corchetes):

1. JUnit [2,5(+1)] [`TablaTest.java`, `Tabla.java`, 30-35 líneas de código] Crea una clase de JUnit nueva **`TablaTest`**, para probar el funcionamiento correcto del método de carga de ficheros CSV (método `processCSV`). Para ello debe crear una tabla partiendo del fichero *testTabla1.csv* que verás en el mismo paquete. Tras cargar la tabla la prueba debe comprobar que el número de filas (`size()`) es el correcto, el de columnas (`getWidth()`) es el correcto, y los valores son exactamente estos:

"1", "1", "1", "1"
"2", "2", "2", "2"
"4", "4", "4", "4"
"6", "6", "6", "6"

[+1] Observa que hay otro fichero CSV *testTabla2.csv*, que tiene dos líneas con errores. Añade al test anterior que se compruebe que esta segunda tabla también tiene los mismos valores. Verás que el test falla porque se cargan filas incorrectas (una tiene valores de más, la otra valores de menos). Corrige el método `processCSV` para que las filas incorrectas se borren de la tabla antes de devolverse (tienes un punto marcado con `// T1` para hacer esto), y comprueba que ahora los dos ficheros pasan el test con los mismos valores.

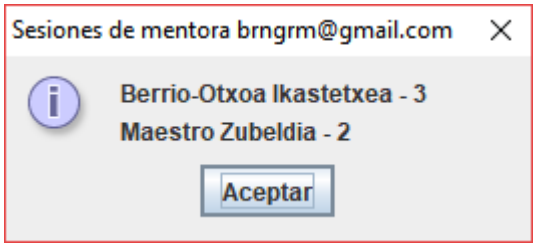
2. Java funcional y loggers [3] [`VentanaDatos.java`, 10-15 líneas de código] Verás un método `procesaErrorLinea` definido al final de la clase ventana. Utiliza ese método para gestionar los dos errores que actualmente sacan información de error a la consola `System.err` (unas líneas más arriba, marcadas con `// T2`). Ambos casos deben hacer su proceso llamando a procesaErrorLinea pasándoles la acción a realizar y la línea donde se ha producido el error. **Debes utilizar Java funcional** para hacer este proceso y completar el cuerpo del método `procesaErrorLinea` para que haga su trabajo.

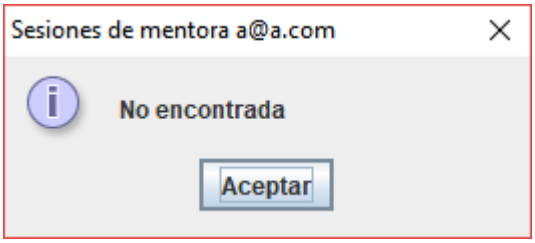
- En el caso del primer error (código de centro incorrecto), el trabajo a hacer es el mismo que se hace ahora pero a la consola de salida (`System.out`) en lugar de a la de error.
- En el segundo error (error en línea de seguimiento), debes sacar el mensaje a un logger en lugar de a la consola. Para ello, define e inicializa un logger estático con salida a un fichero xml (*errores-sesiones.xml*). Puedes utilizar el nivel WARNING para este error.

3. Base de Datos [3,5] [`VentanaDatos.java, BD.java`, 20 líneas de código más los métodos de BD] Observa que hay un botón "Guardar en BD" que no tiene funcionalidad. Prográmalo para que guarde el estado actual de los centros en base de datos. Para ello tienes ya codificados una serie de métodos operativos de base de datos sqlite en la clase BD (tendrás que añadir algunos); también puedes partir de cualquier otra definición de base de datos que ya tengas. Define dos tablas SQL como ves en esa clase BD.java (**líneas 36 a 47**) para guardar centros educativos y estudiantes por sesión. Y al pulsar el botón haz que:

- Se recorran todos los colegios del mapa (*Datos.centros*)
- Se procesen solo aquellos que tienen sesiones (>0)
- Se actualicen los datos de cada centro en la tabla *centroEd* (y si no está, añadirlo)
- Se actualicen los datos de los estudiantes por sesión en la tabla *centroSesion* (si no está, añadirlo) para cada una de las seis sesiones (1-6)

4. Recursividad [3] [`VentanaDatos.java`, 20 líneas] Programa el botón "Buscar mentora" (verás el código ya preparado con el espacio para esta tarea marcado como `// T4`). Al pulsar el botón el usuario puede introducir un email de mentora. La tarea debe *buscar* este email e indicar los centros en las que esa mentora ha tenido sesiones, y el número de sesiones en cada centro en mensajes de `JOptionPane`, por ejemplo con estos dos emails:





El código preparado ya hace el trabajo de cargar las sesiones, y construye una lista de mentoras-centros (definida con objetos `MentoraCentro`) que se visualiza en consola (para que veas más claramente cómo está construida esa lista). La lista **ya está ordenada** por email de mentora y por código de centro (observa que una misma mentora puede hacer sesiones en más de un centro), con lo que te falta por hacer es la búsqueda en esa lista y el recorrido (si procede) de todos los centros de esa mentora. Para ello:

- Debes programar una **búsqueda binaria recursiva** en esa lista que devuelva la posición del email.
- Intenta programar la búsqueda para que la posición devuelta sea la primera vez que el email aparece. De este modo, puedes recorrer las posiciones siguientes mientras el email sea el mismo, para construir una información multilínea como la que aparece en el primer ejemplo (arriba).
- Si el email no se encuentra, muestra el mensaje "No encontrada" (segundo ejemplo).
- Comprueba finalmente que el número de llamadas recursivas es **logarítmico** con respecto a la longitud de la lista (calcula un contador y visualiza en consola ese contador y el tamaño de la lista).