



Proyecto buscaminas. Sprint 5

1 Descripción general de la versión final del juego

En esta versión del proyecto, añadiremos funciones relacionadas con el manejo del ranking de puntuaciones y operaciones de entrada/salida.

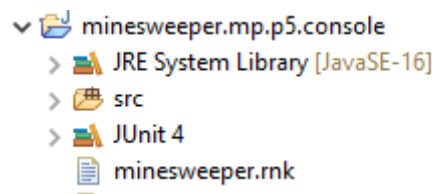
- Existirá siempre y en todo momento una copia persistente de ranking (GameRanking). Se proporciona en la clase Main, un fichero en disco con un nombre y ubicación fijos (fichero por defecto).
- Se ofrece una nueva opción de menú para que el usuario pueda exportar el ranking actual (en memoria) a un fichero (puede ser distinto del fichero por defecto).
- También se ofrece una opción de menú para que el usuario pueda sobrescribir el ranking en memoria con el contenido de un fichero de su elección (que podría ser distinto del fichero por defecto).
- En caso de seleccionar las opciones de menú 2 o 3, las entradas del ranking se mostrarán de forma ordenada.
- Por último, se implementará un log capaz de registrar los sucesos en fichero.

2 Ranking persistente

En la versión anterior, las puntuaciones obtenidas por los usuarios (**game ranking**) se mantienen únicamente en memoria. Es decir, al iniciar una nueva sesión, la estructura de datos que mantiene el **game ranking** se crea de nuevo y, aunque al finalizar una partida el usuario decida guardar su resultado y dicha estructura de datos se actualice, al salir de la aplicación, se pierden todas las puntuaciones conseguidas.

En esta nueva versión, haremos persistente el **game ranking** logrando así que **todos los resultados de todas las partidas estén disponibles para todas las sesiones**.

Para ello, se mantendrá en un fichero una copia persistente del **game ranking** en memoria. Se utilizará un **fichero** de texto que tendrá un nombre y ubicación fijos, es decir, será el mismo fichero **para todas las sesiones: minesweeper.rnk**



Se proporciona un nuevo fichero Main que debe sustituir al que se encuentra en el proyecto Console del Sprint4. En él se encuentran los nombres para los nuevos ficheros necesarios.

Para implementar las operaciones de entrada/salida a fichero, se requiere hacerlo en un paquete (uo.mp.lab.util.file o similar) del proyecto util23.

Es imprescindible que el **game ranking** en memoria y en disco sean consistentes en todo momento, es decir, tengan el mismo contenido. Para lograrlo:



- Al **arrancar la aplicación**, el contenido de `minesweeper.rnk` se carga al **game ranking** en memoria.
- **Cuando se finalice una partida** y siempre que el usuario decida guardar su resultado, no solo se añadirá un nuevo objeto `GameRankingEntry` al **game ranking** en memoria, sino que se actualizará inmediatamente el fichero `minesweeper.rnk` para mantener la consistencia.
- Cada vez que el usuario seleccione la opción de menú que permite importar un nuevo ranking desde un fichero de su elección y, por tanto, se sobrescriba el **game ranking** en memoria, de forma inmediata debe actualizarse también el contenido del fichero `minesweeper.rnk`.

Al tratar de realizar estas operaciones pueden aparecer varios errores que se deberán gestionar mediante el uso de excepciones:

Situación de error	Respuesta del sistema
Al iniciar la aplicación. El fichero <code>minesweeper.rnk</code> no se encuentra en la ubicación esperada.	* El ranking en memoria se inicia sin puntuaciones previas .
Al iniciar la aplicación. Cualquier otra excepción intentando cargar el fichero <code>minesweeper.rnk</code>	* Se tratará como un error de sistema. Se informa al usuario del error y la aplicación termina de forma controlada.
Al tratar de guardar una puntuación nueva. Aparece un error de escritura.	* Se tratará como un error de sistema. Se informa al usuario del error y la aplicación termina de forma controlada.

***IMPORTANTE:** Utiliza el método `showErrorMessage(msg)` del objeto `SessionInteractor` para dar el aviso correspondiente al usuario de un error suyo y `showFatalErrorMessage(msg)` para errores de sistema o de programación.

2.1 Implementación de ranking persistente

Será necesario realizar los siguientes cambios sobre el proyecto resultante de los sprint anteriores:

1. El **constructor de la clase `GameRanking`** debe recibir **un parámetro más**: `String rankingFileName`. Este parámetro indica la ruta en disco al archivo de ranking.
2. **Se debe leer** el ranking al iniciar y **sobre escribir** el ranking cada vez que haya una nueva puntuación que guardar. Ambas **consistirán en serializar o des-serializar** la lista de objetos `GameRankingEntry` de **`GameRanking`** en la ruta en disco recibida en el constructor. Se debe usar la serialización de Java, **no implementes tus propios parsers/serializers**.
3. La clase **`GameRankingEntry`** debe **implementar** la interfaz **`Serializable`** de Java para poder llevar a cabo la operación de serialización.

Más allá de estas 3 indicaciones obligatorias, para culminar esta tarea será necesario con toda seguridad **que añadas nuevos métodos y nuevas clases a tu proyecto**. Añade el contenido que **consideres oportuno** para lograr la funcionalidad indicada. Para ello, cumple con **principios generales sobre calidad de código** en los que hemos insistido durante el curso:

- Máxima encapsulación.
- Mínimo acoplamiento.



- Nombres y clases de métodos con sentido.
- Ajuste a convenciones de nombrado y estilo.
- Ocultación adecuada (sólo son miembros públicos aquellos que es necesario que sean públicos).
- Gestión adecuada de excepciones (cuándo y por qué se generan, dónde se capturan y qué efecto han de tener sobre la aplicación).
- Validación de parámetros en métodos públicos (precondiciones).
- Asignación adecuada de clases a paquetes según su funcionalidad.
- ...

3 Importar y exportar ranking

El menú principal contará con **dos nuevas opciones**, además de las ya implementadas en sprints anteriores: **importar** y **exportar el ranking** de puntuaciones. El menú mostrado mediante `ConsoleSessionInteractor` deberá ofrecer las opciones que mostramos en la *Ilustración 1*.

```
Available options:
1- Play a new game
2- Show my results
3- Show all results
4- Export results
5- Import results
0- Exit
Option? 3
```

Ilustración 1 - Menú de Consola

3.1 Comportamiento de exportar ranking

Al seleccionar esta opción de menú, la aplicación **solicitará un fichero** al usuario. Las **puntuaciones** cargadas en la lista de `GameRanking` **se escribirán** en este fichero.

En este caso, las puntuaciones NO se guardarán usando la serialización de Java, sino en un fichero de texto con el formato mostrado en la *Ilustración 2*.

Cada objeto `GameRankingEntry` se almacenará en **una línea** aparte del fichero que se crea. Cada atributo se representará en un campo, y cada campo estará **separado** del siguiente campo mediante un **carácter ‘;’**.

```
Pepe;08/02/2020;21:05:16;EASY;lost;4
Dani;06/04/2020;18:49:18;EASY;won;45
```

Ilustración 2 - Fichero de texto de puntuaciones

- Campo **0**: **nombre** de usuario.
- Campo **1**: **fecha** (dd/mm/aaaa).
- Campo **2**: **hora** (hh:mm:ss).
- Campo **4**: **nivel** de la partida (EASY, MEDIUM, HIGH).
- Campo **5**: **victoria** (won) o **derrota** (lost).
- Campo **6**: **tiempo** de la partida en segundos.

Para generar un string con la fecha y la hora, en los formatos indicados a partir de un objeto `Date`, se puede hacer según se indicó en el sprint 4 o de la siguiente forma:

```
String date = new SimpleDateFormat("dd/MM/yy").format( GameRankingEntry.getDate() );
String time = new SimpleDateFormat("HH:mm:ss").format( GameRankingEntry.getDate() );
```



3.2 Comportamiento de importar ranking

Al seleccionar esta opción de menú, la aplicación **solicitará** un el nombre de **fichero** al usuario y **se cargarán** en la aplicación las **puntuaciones** que se encuentren en el fichero. El contenido del fichero debe seguir el **formato** indicado en la sección 3.1. Las nuevas puntuaciones cargadas sustituirán a las que tuviera anteriormente el GameRanking.

IMPORTANTE. El ranking en fichero, sea cual sea el momento en que se selecciona esta opción de importar, se sobrescribe con el contenido del nuevo ranking en memoria.

Para realizar el proceso contrario, generar un objeto Date a partir de un string con la fecha y la hora en los formatos indicados, se puede hacer de la siguiente forma:

```
String str = strDate + " " + strTime;  
Date date = new SimpleDateFormat("dd/MM/yy HH:mm:ss").parse(str);
```

3.3 Posibles situaciones de error al importar/exportar fichero

Situación de error	Respuesta del sistema
Al importar, el fichero no existe.	* Se indica al usuario que no se puede acceder al fichero y se descarta la operación de importación.
Al importar, alguna de las líneas contiene errores.	* Se registra en el log el error detectado en cada línea con error y se descarta la línea. El resto de las líneas se siguen procesando para crear objetos GameRankingEntry.
Al importar/exportar, ocurre algún otro error de lectura o escritura.	* Se registra en el log el mensaje de error generado por la excepción. Además, se le indica al usuario que la operación no se ha podido llevar a cabo por un error inesperado y se termina la ejecución.

***IMPORTANTE:** Utiliza el método `showErrorMessage(msg)` del objeto `SessionInteractor` para dar el aviso correspondiente al usuario de un error suyo y `showFatalErrorMessage(msg)` para errores de sistema o de programación.

3.4 Implementación de importación y exportación

La **Interfaz SessionInteractor** debe incluir un **nuevo método**:

```
String askFileName()
```

Pide al usuario el nombre de un archivo. Devuelve el nombre mediante un String.

La clase **ConsoleSessionInteractor** debe **implementar este método** para solicitar el nombre por consola.

Más allá de este cambio, **introduce las nuevas clases y métodos que consideres necesarias** (parsers, serializadores, código de utilidades, excepciones...) respetando los principios generales de calidad de código.

Como medida excepcional los warnings derivados de aplicar un cast tras las de-serialización, pueden suprimirse con `@SupressWarning`.

4 Ordenación de Ranking

Cuando el usuario seleccione las opciones para **mostrar puntuaciones** (opciones 2 o 3 del menú), la **lista** debe aparecer **ordenada**. Los **criterios** de ordenación a seguir son los siguientes:



- **Nivel:** aparecerán más arriba las partidas de mayor nivel de dificultad. HIGH > MEDIUM > EASY. El método `compareTo` de los enumerados Java ordena los valores en el orden en que estén escritos en el tipo enumerado.
- **Tiempo.** Si se empata en lo anterior, aparecerán más arriba las partidas con una duración menor.
- **Fecha:** Si se empata en lo anterior, aparecerán más arriba las partidas más antiguas.

4.1 Implementación de ordenación

Implementa la ordenación entre objetos `GameRankingEntry` con una **nueva clase comparadora** que implemente la interfaz **`Comparator<GameRankingEntry>`**. **Añade** además las **clases** y **métodos** que consideres necesarios para completar esta tarea.

5 Log a fichero

Implementa una nueva **clase `FileLogger`** que implemente la **interfaz `SimpleLogger`** y que, en lugar de escribir información en la salida de error estándar, añada los **mensajes** que le llegan a un **fichero de texto** llamado **`minesweeper.log`**.

Indicaciones:

- **Formato:** los ficheros de log suelen contener una línea por mensaje recibido. Dicha línea suele contener en primer lugar la **fecha** en que se produjo el mensaje y, en segundo lugar, **el propio mensaje** recibido. Ejemplo de formato completo:

[dd/mm/aaaa – hh:mm:ss] : Error de I/O. “wrong.txt” no encontrado.
- **Modo `append`:** presumiblemente, cada vez que hagas una operación de log abrirás un flujo de fichero, escribirás el propio mensaje, y cerrarás el flujo de fichero. Recuerda abrir el flujo en *modo `append`*, es decir, asegúrate de **no borrar el contenido antiguo del fichero cada vez que escribes una línea nueva**.
- **Ubicación.** El fichero `minesweeper.log` debe estar localizado en el mismo path que `minesweeper.rnk`.

6 Test

Prueba los siguientes métodos públicos:

1. **`RankingParser::parse(List<String> lines)`**. *Casos principales:* lista vacía, lista con alguna línea en blanco, lista con alguna **línea incorrecta** (diferentes errores), lista con todas las líneas correctas (diferentes combinaciones de campos válidos),
2. **`RankingSerializer::serialize(List< GameRankingEntry > gameRankingEntries)`**. *Casos principales:* lista vacía, lista con objetos con diferentes campos (victoria/derrota, nivel de dificultad...),