Battleship sprint 5

1 Descripción general de la versión final del juego

En este sprint añadiremos funciones relacionadas con el manejo del ranking de puntuaciones y operaciones de entrada/salida.

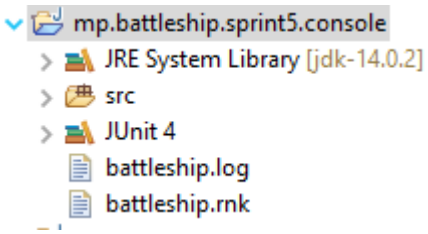
- Existirá siempre y en todo momento una copia persistente de ranking (GameRanking). Estará en un fichero en disco con un nombre y ubicación fijos.
- En caso de seleccionar las opciones de menú 2 o 3, las entradas del ranking se mostrarán de forma ordenada.
- Por último, se implementará un log capaz de registrar los sucesos en fichero.

2 Ranking persistente

En el anterior sprint, 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 este sprint 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: battleship.rnk.**



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

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 battleship.rnk se carga al game ranking en memoria.
- **S**iempre que el usuario decida guardar su resultado, no solo se añadirá un nuevo objeto Score al game ranking en memoria, sino que se actualizará inmediatamente el fichero battleship.rnk para mantener la consistencia.

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 battleship.rnk 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 battleship.rnk	* Se tratará como un error de sistema. Se informa al usuario del error y la aplicación termina ordenadamente.
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 ordenadamente

***IMPORTANTE: Utiliza el método showFatalErrorMessage(msg)** del objeto **SessionInteractor** para dar el aviso correspondiente al usuario.

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 **sobreescribir el ranking** cada vez que haya una nueva puntuación que guardar. Ambas **consistirán en serializar o des-serializar la lista de scores de GameRanking** en la ruta en disco recibida en el constructor. **IMPORTANTE**: Usa serialización de Java, **no implementes tus propios parsers/serializers**.
3. La clase **Score** 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 precondiciones en métodos públicos.
- Asignación adecuada de clases a paquetes según su funcionalidad.
- …

3 Ordenación de Ranking

Cuando el usuario seleccione las opciones para **mostrar puntuaciones** (2 o 3), 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.

3.1 Implementación de ordenación

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

4 Log a fichero

Desarrolla una nueva **clase FileLogger que implemente la interfaz Logger** 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 battleship.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 cerraras el flujo de fichero. Recuerda abrir el flujo en m*odo append*, es decir, asegúrate de **no borrar el contenido antiguo del fichero cada vez que escribes una línea nueva**.

```
new FileOutputStream(logFile, FOR_APPENDING)
```

- **Ubicación**. El fichero battleship.log debe estar localizado en el mismo path que battleship.rnk.

IMPORTANTE: **cuando inicies la aplicación**, recuerda hacer que el **logger** sea una **instancia de la clase que has implementado en este apartado** en lugar del BasicSimpeLogger del sprint 4.