Proyecto: Buscaminas. Parte 3

1 Descripción general de la nueva versión del juego

En esta versión del proyecto añadiremos **tres nuevas características** a nuestro buscaminas:

- **Descubrimiento inicial de una isla de casillas vacías**.
- **Desacoplamiento del código del juego y de la interfaz de usuario**, posibilitando usar y adaptar distintas interfaces sin modificar las clases que se encargan de la lógica.
- **Junto con lo anterior, se mejorará la entrada y salida por pantalla**.

2 Estructura de la solución

2.1 Nuevos proyectos

El juego se dividirá en tres proyectos distintos.

- **identificación_minesweeper_sprint3** contendrá todo lo necesario para implementar las reglas del juego. No contendrá clase Main ya que no se ejecuta el juego desde aquí, sino desde uno de los proyectos que actúan de interfaz con el juego.

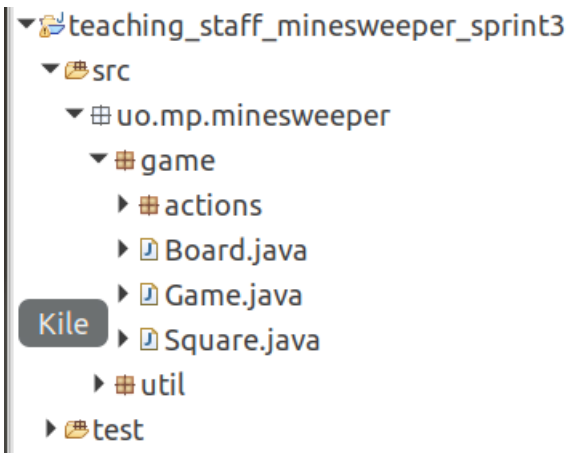
- **minesweeper.mp.p3.console** implementa una interfaz de usuario por consola (modo texto). Deberá ser implementado en parte por los alumnos. Además, será necesario cambiar el nombre, añadiendo la identificación (apellidos_nombre) del alumno.

- **minesweeper.mp.p3.gui** implementa una interfaz de usuario gráfica. Está completamente implementado por los profesores. No es necesario renombrar.

Los dos segundos contendrán clases Main desde las que se podrán ejecutar el juego implementado en el primer proyecto utilizando distintas interfaces. Hacia el final de este documento se explica cómo integrarlos.

2.2 Estructura del proyecto minesweeper_sprint3

Al final de esta práctica, la estructura de paquetes de este proyecto debe ser como sigue:



- paquete **game** contiene el código java que implementa las reglas del juego (Board, Game, Square y otras que el alumno crea necesario)
- paquete **game.actions** con el código que implementa las acciones que se realizan en las distintas casillas al invocar su método `stepOn`, tal como se describe más adelante
- paquete **util**, puede contener clases de utilidad, no relacionadas directamente con las reglas del juego, como el chequeo de argumentos (también puede decidir hacerse en un proyecto separado, en cuyo caso el nombre deberá ser identificaciónAlumno_minesweeper_sprint3_util).

Para el chequeo de argumentos, es necesario una clase como la que sigue, que contenga, al menos, los dos métodos indicados:

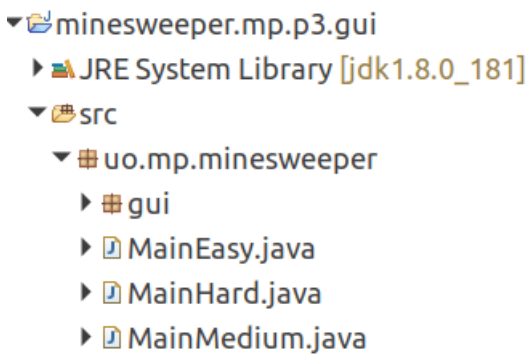
```
public class ArgumentChecks  {

    /**
     * Evaluates condition. If false, throw IllegalArgumentException with message
     * @param condition
     * @param message
     */
    public static void isTrue( boolean condition, String message ) {

    /**
     * Evaluates condition. If false, throw IllegalArgumentException
     * @param condition
     */
    public static void isTrue(boolean condition) {
}
```

Nótese que no hay clase `Main` en este proyecto, ni debe haberla.

2.3 Estructura del proyecto de interfaz gráfica

Este proyecto (**minesweeper.mp.p3.gui**), proporcionado por los profesores, despliega una interfaz gráfica similar a la del buscaminas original, con casillas sobre las que es posible hacer click.

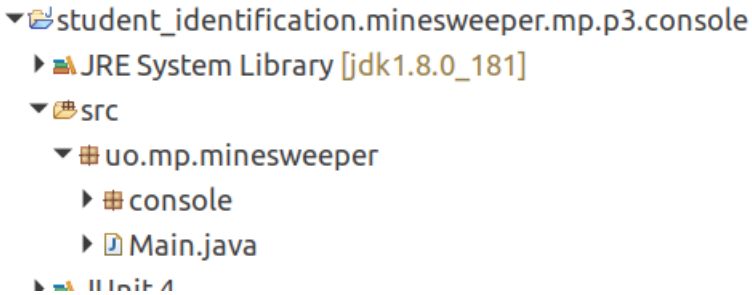
Tiene la siguiente estructura:



- paquete **gui** contiene todo lo necesario para implementar esta interfaz gráfica
- Las clases **MainEasy**, **MainHard** y **MainMedium** son tres clases que podrán ejecutarse para iniciar el juego con distintos niveles de dificultad.

2.4 Estructura del proyecto de interfaz textual

Por último, este proyecto (**minesweeper.mp.p3.console**), que debe ser implementado a medias por el usuario, tiene una clase **Main** para ejecutar el juego con una interfaz modo texto (la que venimos usando hasta ahora) y un paquete **console** donde se debe implementar una interfaz modo texto (`ConsoleGameInteractor`, descrito más adelante en este mismo documento), así como todas aquellas clases auxiliares que el usuario crea necesario.



Deberá renombrarse, antes de entregarlo, para que incluya el nombre del usuario.

Es completamente necesario mantener la estructura de paquetes anterior, así como los nombres de clases y métodos indicados a lo largo de los enunciados.

3 Descubrimiento inicial de una isla de casillas vacías

Las implementaciones más habituales del buscaminas se aseguran de que el primer click del usuario sea sobre una casilla vacía y cause el descubrimiento de una isla.

Puedes comprobar este comportamiento en la versión online del buscaminas disponible en http://buscaminas.eu/. Sea cual sea la casilla que escojas inicialmente, esto siempre provoca el descubrimiento de una isla. De esta forma, se evita que el usuario tenga que hacer clicks aleatorios hasta dar con una.

Implementar este comportamiento implicaría tener en cuenta la primera decisión del usuario para repartir las minas. Esa primera selección de casilla condicionaría la construcción del tablero para asegurar que la casilla escogida pertenece a una isla.

Para **nuestro proyecto** vamos a implementar un **comportamiento similar pero más simple**. Descubriremos una isla de casillas vacías al azar. Es decir, **al iniciar el juego, el tablero no aparecerá con todas las casillas cerradas**, sino que **una de las islas ya habrá sido destapada**. El usuario comenzará a jugar con esa situación de tablero.

3.1 Implementación

Para ello deberás implementar un **nuevo método público en la clase *Board***:

```
void uncoverWelcomeArea()
```

> Busca una casilla vacía al azar y ejecuta *stepOn* sobre ella para descubrir una isla.
> La casilla escogida al azar no podrá ser una esquina del tablero.

Asegúrate de que ***uncoverWelcomeArea()*** se llama una vez **antes de mostrar el tablero** y solicitar el primer comando al usuario.

También son necesarios dos getter públicos en Board

`int rows()`

Retorna el número de filas en el tablero

`int cols()`

Retorna el número de columnas en el tablero

4 Desacoplamiento de Game y de la interfaz de usuario

Cuando hablamos de interfaz de usuario nos referimos a la vía de comunicación utilizada para recoger los comandos y mostrar la información pertinente al usuario.

Actualmente, la interfaz de usuario que usa nuestro buscaminas es la consola (interfaz textual). Se solicitan comandos al usuario y se imprime el estado de la matriz usando una representación con caracteres del estado del tablero.

Es común en cualquier proyecto que las interfaces de usuario se adapten o modifiquen sin que esto afecte a la lógica de la aplicación. Es también común que un mismo programa funcione con distintas interfaces.

Para llevar a cabo este objetivo es **imprescindible que el código que contiene la lógica del programa esté desacoplado del código que sirve para comunicarse con el usuario**. Actualmente, nuestra clase Game contiene código que implementa las reglas del juego mezclado con código de interfaz de usuario. Desde allí se ejecutan sentencias *println* y se leen comandos por teclado.

Modificaremos la clase *Game* para que todas **estas operaciones se soliciten a un objeto** que implemente una interfaz java a la que llamaremos *GameInteractor*.

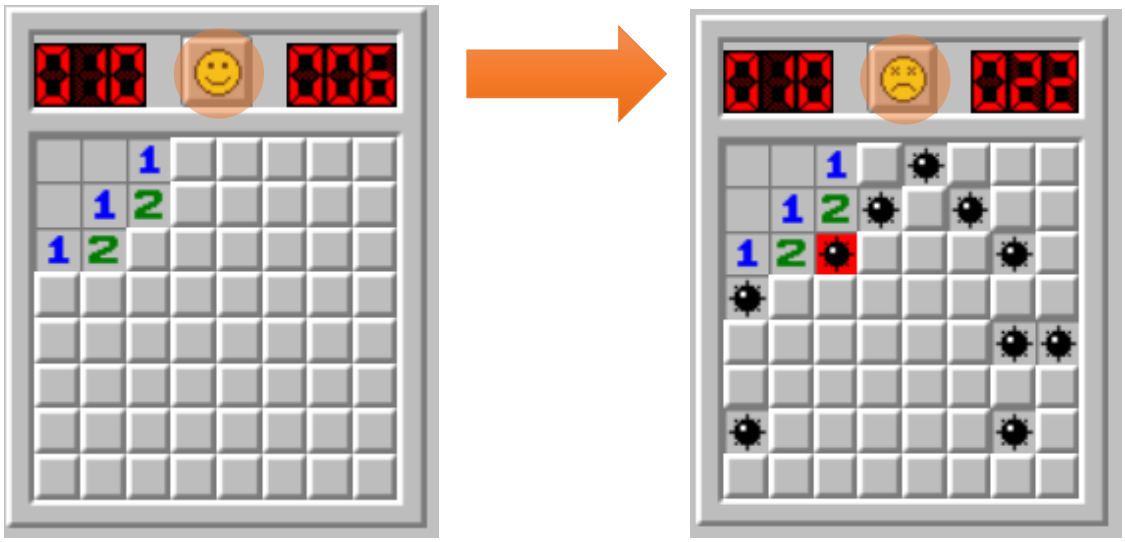
Esto quiere decir que, dentro de la clase *Game*, donde antes teníamos algo como:

```
System.out.println("BOOOOMMMM!!!!")
```

Ahora tendremos una llamada a un objeto *interactor* de tipo *GameInteractor* como:

```
interactor.showBooommm();
```

Cada objeto *interactor* ejecutará a su manera esta instrucción (polimorfismo). Para un método como *showBooommm()*, un interactor que maneje la consola ejecutará *System.out.println("BOOOOMMMM!!!!")*. En cambio, otro interactor conectado a **una interfaz gráfica como la del buscaminas online podría modificar el aspecto de un icono**.



4.1 Implementación

Para desacoplar *Game* de la interfaz de usuario se deben realizar las siguientes acciones.

4.1.1 Clase GameMove

Esta clase deberá localizarse en el **paquete *uo.mp.minesweeper.game***

GameMove consiste en un simple **contenedor de datos** para recoger una acción de usuario completa: comando, fila y columna. *GameMove* debe contener los siguientes métodos públicos:

```
GameMove(char operation, int row, int column)
```

> **Constructor**. Recibe un carácter ('s', 'f', o 'u') representando un comando de usuario sobre el juego, y dos enteros *row* y *column* que indican las coordenadas de la casilla donde aplicar el comando.

```
char getOperation()
```

> Devuelve el carácter que representa el comando de usuario.

```
int getRow()
```

> Devuelve la coordenada de fila.

```
int getColumn()
```

> Devuevle la coordenada de columna.

```
String toString()
```

> Devuelve una representación textual del comando completo con el siguiente formato (sustituir los campos en negrita por los datos que procedan en cada caso):
>
> ```
> GameMove [operation=s, row=1, column=1 ]
> ```

4.1.2 Interfaz GameInteractor

Esta interfaz deberá localizarse en el **paquete *uo.mp.minesweeper.game*.**

La interfaz *GameInteractor* ofrece todas las **órdenes** que se pueden generar desde el juego para **enviar** o **solicitar información** al usuario.

Contendrá los siguientes **métodos públicos**:

```
GameMove askMove(int heigth, int width);
```

Solicita al usuario un comando (s, f, u) y unas coordenadas de fila y columna. Devuelve un objeto de tipo *GameMove* que contiene la información facilitada por el usuario. Los parámetros *heigth* y *width* marcan el tamaño máximo de fila y columna respectivamente que el usuario puede introducir. El método ha de asegurarse de que tanto el carácter de comando como las coordenadas devueltas son valores válidos (dentro de los rangos permitidos).

```
void showGame(long elapsedTime, Board board);
```

Muestra el estado del juego al usuario. Esto incluye estado del tablero, tiempo empleado y banderas que quedan por colocar. *elapsedTime* contiene el tiempo empleado, y *board* es una referencia al objeto tablero cuya información se va a mostrar.

```
void showGameFinished();
```

Informa al usuario de que el juego ha terminado.

```
void showCongratulations(long elapsedTime);
```

Informa al usuario de que ha ganado el juego. Le muestra el tiempo que ha empleado, que se recibe en el parámetro *elapsedTime*.

```
void showBooommm();
```

Informa al usuario de que ha pisado una casilla de mina.

```
void showReadyToStart();
```

Informa al usuario de que el juego está listo para comenzar.

4.1.3 Adaptación del código de Game

. Se añadirá a la clase *Game* el siguiente **método público**:

```
void setInteractor ( GameInteractor  interactor )
```

Asigna a *Game* el objeto *GameInteractor* recibido como parámetro.

Al construir un objeto *Game*, se deberá llamar a ***setInteractor()*** **antes** de ejecutar ningún método que implique **entrada o salida de información** desde *Game*.

Es necesario **sustituir en *Game* todos los *System.out.println()*** y todas las sentencias destinadas a solicitar una **entrada de datos** al usuario por una llamada al método que proceda de su **objeto *interactor***.

5 Mejora de la entrada y salida por pantalla

Se proporcionarán dos formas nuevas de interacción con el usuario. Una de ellas, proporcionada por los profesores, realizará la interacción con el usuario de forma gráfica (GUI). La otra, a implementar por los estudiantes, lo hará a través de consola.

Ambas deben implementar la interfaz `GameInteractor`.

5.1 Implementación de un ConsoleGameInteractor

`ConsoleGameInteractor`, que implementa la interfaz `GameInteractor`, deberá estar en el proyecto ***minesweeper.mp.p3.console***

Esta nueva clase **mejorará el aspecto de la salida por pantalla** y la forma de **interaccionar con el usuario**.

- Se **preguntará** al usuario **por separado** por un **movimiento** (**s**tepOn, **f**lag o **u**nflag), por una **fila** y por una **columna**.
- La matriz de texto que muestra la situación de juego no consistirá únicamente en un carácter por cada casilla representando su estado, sino que tratará de dibujar un tablero de forma más clara con caracteres puramente ornamentales**.** Además, incluirá **números de guía fuera de la matriz** para que el usuario no tenga que contar filas y columnas para indicar una coordenada**.**

En la siguiente captura puedes ver un ejemplo de tablero mejorado y de la forma de solicitar comandos al usuario:

```
Ready to start
Time: 0 seconds
Flags left: 9
      0   1   2   3   4   5   6   7   8
   +---+---+---+---+---+---+---+---+---+
 0 | # | # | # | # | # | # | 1 |   |   |
   +---+---+---+---+---+---+---+---+---+
 1 | # | # | # | 2 | 2 | 1 | 1 |   |   |
   +---+---+---+---+---+---+---+---+---+
 2 | # | # | # | 1 |   |   |   | 1 | 1 |
   +---+---+---+---+---+---+---+---+---+
 3 | 2 | 3 | 2 | 1 |   | 1 | 1 | 2 | # |
   +---+---+---+---+---+---+---+---+---+
 4 |   |   |   |   |   | 2 | # | 3 | 1 |
   +---+---+---+---+---+---+---+---+---+
 5 |   |   |   |   |   | 2 | # | 2 |   |
   +---+---+---+---+---+---+---+---+---+
 6 |   |   |   |   |   | 1 | 1 | 1 |   |
   +---+---+---+---+---+---+---+---+---+
 7 |   | 1 | 1 | 1 |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 8 |   | 1 | # | 1 |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+

movement (s | f | u)? s
row? 1
column? 1
```

El nuevo formato de tablero no tiene por qué ser idéntico al mostrado en la captura, pero sí ajustarse lo más posible, **incluyendo, obligatoriamente, los siguientes elementos**:

- **Caracteres que separen unas celdas de otras**.

- **Números de guía** en abscisas y ordenadas para que sea más sencillo introducir las coordenadas de una casilla.

Recuerda, la salida por pantalla ya no se producirá desde la clase *Game*. Se implementará una clase *ConsoleGameInteractor*

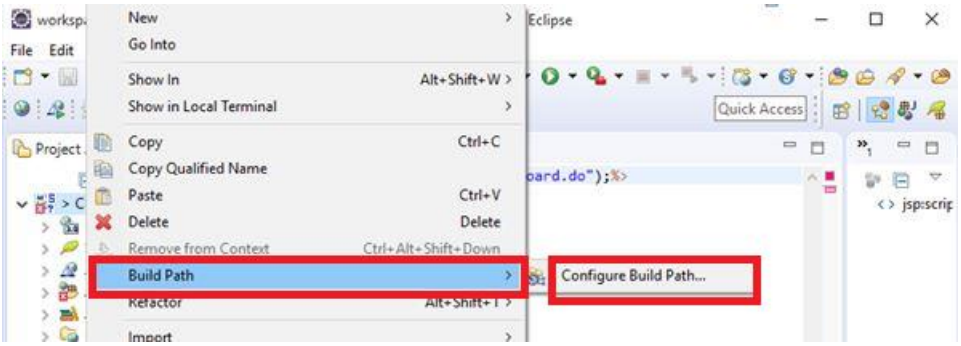
6 Integración de los distintos proyectos.

En el material de este sprint se proporciona un **proyecto** llamado ***minesweeper.mp.p3.gui***. Y otro llamado ***minesweeper.mp.p3.console*** que deberá ser renombrado para contener el nombre del alumno.

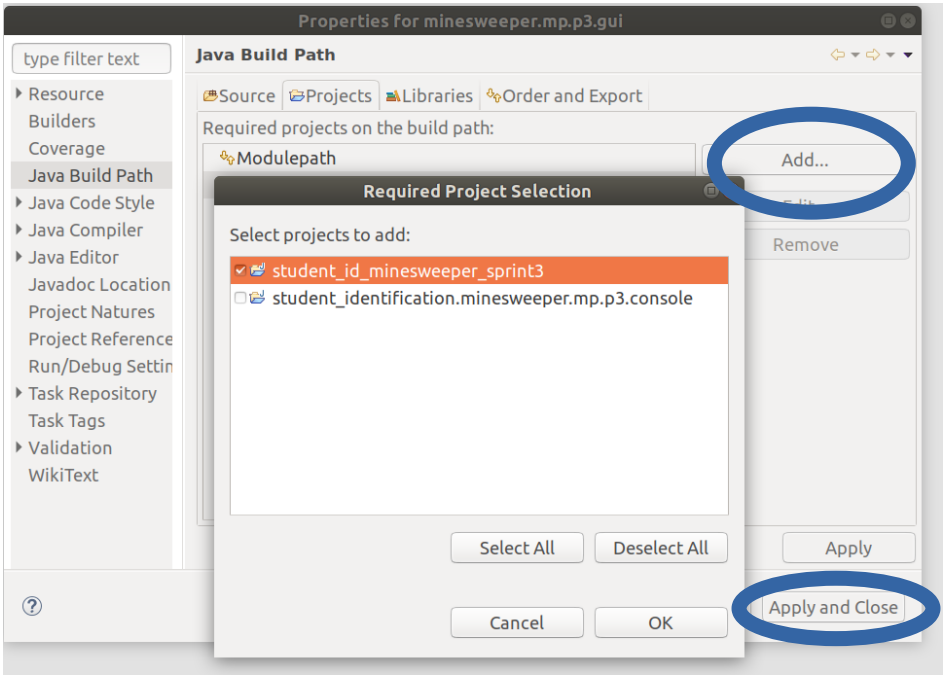
Ambos tienen clases Main desde las que ejecutar el juego. El proyecto de interfaz gráfico contiene 3 clases Main para poder seleccionar la dificultad del juego.

A continuación, se indica cómo proceder (en ambos):

1. Importa el proyecto en tu espacio de trabajo
2. En el nuevo proyecto importado



3. En la pestaña **Projects**, añadir el proyecto principal (**Add**), y seleccionar **OK** seguido de **Apply and Close**



6.1 Cómo ejecutar el juego

Prueba a ejecutar tu aplicación desde cualquiera de las interfaces. Simplemente ejecuta una de las cuatro clases Main que tendrás ahora disponibles. Cada una ejecutará el juego con una interfaz o nivel de dificultad diferente.

7 Test

Deberás añadir **tests** para el método ***`uncoverWelcomeArea()`*** **de la clase** ***`Board`***.