Ejercicio Obligatorio. Sesión 01

Este ejercicio debe estar implementado 48 horas antes de la siguiente clase de laboratorio.

- Cargue el proyecto: student_session01_game
- Renombre el proyecto dentro del Eclipse (usando `Refactor- Rename`) como sigue: **nombre_apellido1_apellido2_session01_task_game**, en letras minúsculas y sin acentos (por ejemplo, un estudiante llamado Pablo Peláez Nuño quedaría renombrado como **pablo_pelaez_nuno_session01_task_game**).
- Para entregar la tarea es necesario exportar el proyecto y comprimirlo en **formato ZIP**

Enunciado

Este enunciado está basado en el juego 2048 **con algunas modificaciones**. **Se realizará en dos partes**. En esta sesión se realizan las operaciones definidas aquí y en la próxima se ampliará con otras operaciones.

2048 es un juego en línea creado en 2014 por Gabriele Cirulli (con 19 años), cuyo objetivo es mover los números del tablero para combinarlos hasta conseguir en una celda del tablero el número 2048, 4096, … Se puede probar en: http://juego2048.es/.

Consta de un tablero cuadrado en el que partiendo de un 2 se trata de sumar números iguales. El usuario realiza movimientos con las flechas: *derecha*, *izquierda*, *arriba* o *abajo* para mover los números. Cada movimiento, sea posible o no, añade un nuevo número 2 al tablero. La partida termina cuando se llena el tablero y no es posible realizar más sumas.

Para la implementación del juego utilizaremos un tablero 3x3 ya que nos facilita hacer las pruebas. Cuando la aplicación esté funcionando se puede cambiar la dimensión.

Este proyecto se va a implementar en las dos primeras sesiones.

En la sesión 1 se deberá completar el proyecto proporcionado (student_sesion1_game) realizando solo desplazamientos.

Por ejemplo, al comienzo de una nueva partida la interfaz de la aplicación puede mostrar:

```
JUEGO 2048
0 0 0
2 0 0
0 0 0

Mueve los números en una dirección [r R]/[l L]/[u U]/[d D]:
```

A continuación, se debe introducir un carácter para indicar el movimiento de los números: [r R] Right, [l L] Left, [u U] Up, [d D] Down. Si pulsamos la tecla R el tablero quedaría:

```
2 0 0
0 0 2
0 0 0

Mueve los números en una dirección [r R]/[l L]/[u U]/[d D]:
```

Si pulsamos de nuevo la tecla R el tablero quedaría:

```
0 0 2
0 0 2
0 0 2

Mueve los números en una dirección [r R]/[l L]/[u U]/[d D]:
```

Si pulsamos de nuevo la tecla └ el tablero quedaría:

```
2 0 2
2 0 0
2 0 0

Mueve los números en una dirección [r R]/[l L]/[u U]/[d D]:
```

Si pulsamos de nuevo la tecla └ el tablero quedaría:

```
2 2 0
2 0 0
2 0 2

Mueve los números en una dirección [r R]/[l L]/[u U]/[d D]:
```

Hay que tener en cuenta que cada vez que se realiza un nuevo movimiento, se tiene que añadir un 2 en una posición aleatoria del tablero. Los resultados serán diferentes en cada partida ya que se utilizan posiciones aleatorias.

En la sesión 2 además de compactar, se irán sumando de dos en dos los números consecutivos iguales.

El proyecto deberá tener:

1. Clase **Game2048** que incluya las constantes necesarias (la dimensión mínima permitida del tablero será 3X3 y la máxima 10x10) y la propiedad board como matriz de números enteros, con los siguientes métodos públicos:

2. Un **constructor sin parámetros** ***public Game2048()*** para crear un tablero de dimensión por defecto 3x3

3. Un **constructor con un parámetro** ***public Game2048 (int dimension)*** que creará un tablero con filas y columnas igual a la dimensión recibida. En caso de recibir una dimensión incorrecta se creará un tablero de dimensión por defecto 3x3

4. Un **constructor con un parámetro** ***public Game (int[][] board)*** que recibe un tablero. Si el parámetro board es null o el tablero tiene una dimensión no válida o las filas del tablero tienen distinto tamaño se deberá lanzar una excepción.

5. Método ***public int[][] getBoard().*** Devuelve **una copia** de la matriz para que, desde la interfaz, pueda ser pintada como se desee. También será utilizada en las pruebas.

6. Método ***public void restart().*** Permite restablecer el tablero asignando a todas las casillas un cero (se vacía el tablero) salvo una casilla que de manera aleatoria se le asigna un 2 para añadir el primer valor al tablero. Reutiliza el código siempre que puedas.

7. Método ***public void next()***. Añade un nuevo número 2 en una posición aleatoria del tablero que esté vacía (con valor 0).

8. Método ***public boolean isBoardFull().*** Comprueba si el tablero está lleno. Esto ocurre cuando todas las celdas o posiciones del tablero tienen un número distinto de cero.

9. Método ***public void compactRight().*** Desplaza todos los números a la derecha en cada fila, dejando los huecos a la izquierda.

10. Método ***public void compactLeft().*** Desplaza todos los números a la izquierda en cada fila, dejando los huecos a la derecha.

11. Método ***public void compactUp().*** Desplaza todos los números hacia arriba en cada fila, dejando los huecos abajo.

12. Método ***public void compactDown().*** Desplaza todos los números abajo en cada fila, dejando los huecos arriba.

13. **Genere la documentación con JavaDoc**. El código del proyecto debe incluir comentarios de documentación y comentarios de implementación.

14. **Realice las pruebas de los constructores y métodos con JUnit**. Puede usar el aserto assertArrayEquals. Recuerde que es posible inicializar una matriz con valores constantes.

```
int[][] matriz = {{2,0,0},{0,0,0},{2,0,2}};
```

Nota. Se incluye un método estático getSum que puede ser usado para las pruebas de los métodos next y restart. Se recuerda que para invocar a un método estático de una clase la sintaxis es: `NombreClase.nombreMetodoEstático(parámetros)`