Battleship sprint 1

A lo largo de este semestre, deberás implementar cinco versiones (*sprints*) diferentes de dificultad incremental del juego "Hundir la Flota" (Battleship).

Deberás **enviar** una solución intermedia, **antes del 18 de Marzo a las 13:00 que contenga los tres primeros sprints**, y una solución final alrededor del 10 de Mayo (la fecha de esta última se confirmará más adelante). Ambas entregas deben ser enviadas, y ambas deben tener una calidad aceptable, para tener oportunidad de aprobar el curso por evaluación continua ordinaria.

1 El juego básico

Battleship es un juego muy popular de **dos jugadores** que cuentan con una flota de barcos cada uno y deben disparar a la flota del oponente (adivinando las posiciones en las que se encuentran) hasta hundir completamente la flota. Cada jugador dispone de **dos tableros** que usa para marcar la posición de sus barcos en uno y los disparos realizados a la flota del oponente en otro.

Los jugadores alternan turnos, realizando un disparo a una casilla del tablero de los barcos del oponente indicando para ello las coordenadas de dicha casilla.

El objetivo del juego es destruir la flota del jugador oponente hundiendo todos sus barcos.

La localización de la flota permanece oculta al otro jugador, por lo que los jugadores deben intentar adivinar dónde se encuentra dicha flota.

2 Implementación básica del juego

Los jugadores serán un usuario humano (*user*) que juega contra un usuario artificial, la máquina (*computer*).

El juego se lleva a cabo a través de la terminal. Es decir, cualquier interacción con el usuario humano (imprimir los tableros, disparar y mensajes de error, así como entradas del usuario) serán hechas a través de la consola.

Los barcos serán colocados inicialmente en coordenadas **fijas** tanto para la flota del usuario como para la flota de la máquina.

2.1 Configuración del juego

La aplicación guardará el estado del juego usando dos tableros, cada uno almacena inicialmente los barcos de un jugador y posteriormente se añadirán los disparos del oponente. Al comienzo serán colocados diferentes tipos de **barcos**. El resto de las celdas contendrán **agua**.

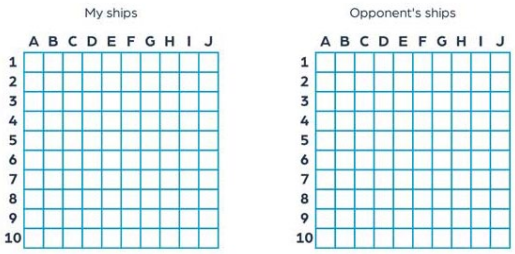


Figura 1: Tableros para jugar a batalla de barcos

El tamaño del tablero así como el número y longitud de los barcos será fija, y la posición será, en esta la misma para ambos jugadores en esta primera versión (sprint 1). Los tableros son cuadrados, de 10×10, y cada casilla individual en el tablero está identificada por su columna y su fila (por ejemplo, C4 corresponde a la columna 2 y la fila 3 de la matriz que la contenga; A1 corresponde a la columna 0 fila 0 de la matriz que la contenga).

Cada flota consistirá en **varios barcos de diferentes longitudes**; esto es, cada barco ocupa diferente número de casillas en el tablero. En este primer sprint, las flotas contienen **un Barco de Batalla** (longitud cuatro), **dos Cruceros** (longitud tres), **tres Destructores** (longitud dos) y **cuatro Submarinos** (longitud uno). Nótese que cada uno tiene longitud diferente y hay varios de cada tipo.

Al comienzo del juego, la aplicación creará dos tableros y **colocará una flota de barcos en cada uno en coordenadas fijas**.

Cada jugador mantendrá, referencias a ambos tableros. Un tablero estará referido como ***myShips*,** y contendrá los **barcos del propio jugador, así como disparos del oponente**. El otro tablero, ***opponentShips*,** contendrá la **flota del oponente, así como los disparos hechos por el propio jugador sobre el oponente**

El otro jugador contendrá referencias a los mismos tableros, pero en el orden opuesto. Esto es, el tablero *myShips* de un jugador será el tablero *opponentShips* para el otro jugador.

2.2 Colocación de la flota de barcos en el tablero de los jugadores

Al comienzo del juego, **la aplicación colocará automáticamente los barcos de cada jugador en sus tableros en coordenadas fijas. Ambas flotas en las mismas coordenadas**.

Los tableros serán implementados a través de una rejilla (*grid*) como un array de enteros:

- Cada tipo de barco se representa por un número del 1 al 4 (1 para **S**ubmarinos, 2 para **D**estructores, 3 para **C**ruceros y 4 para barcos de **B**atalla) **mientras que no hayan sido disparados**.
- **Cuando se dispara a una casilla con barco**, la aplicación reemplaza el contenido por **un número negativo con el mismo valor absoluto**.
- **Las Casillas con agua no disparadas** contienen un **0**, mientras que las **casillas con agua ya disparadas** contienen un **-10**.

1	0	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0
2	2	0	2	2	0	2	2	0	0
0	0	0	0	0	0	0	0	0	0
3	3	3	0	3	3	3	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	4	4	4	4	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

***Array 1**: Contenido inicial del array*

2.3 Realización de una jugada

Los jugadores **realizan turnos de disparos eligiendo una posición (coordenadas) y disparando en el tablero del oponente**. La aplicación responde con **"*HIT*"** o con "***MISS***", según sea el resultado (tocado o agua) y cambia el contenido de la cuadrícula objetivo de manera apropiada, como se indica en el apartado anterior.

Por ejemplo, si un jugador selecciona D6 y el oponente no tienen ningún barco localizado en esta coordenada del array (fila 5, columna 3), la aplicación responderá con *MISS* y debe guardar -10 en esta coordenada de la del array del **tablero de barcos del oponente**.

Una vez que un jugador ha usado su turno y se ha procesado, **le toca al otro jugador**, aún en el caso de que el primero haya logrado disparar a un barco.

Tan pronto como todos **los barcos de un jugador hayan sido hundidos**, el **juego finaliza** y **el oponente gana** el juego.

2.4 Procesamiento del turno

El turno de un jugador consiste en las **acciones** siguientes, **sin importar que el jugador en turno sea el usuario o la máquina**. La única diferencia está en el paso 2. El jugador usuario siempre tiene el primer turno.

1. Mostrar ambos tableros desde la perspectiva del usuario. En el propio muestra sus barcos y los disparos de la máquina, en el del oponente muestra los disparos que él hizo sobre el tablero de la máquina. (ver apartado 3.2 y 3.3).
2. El jugador al que le toca jugar **elige la casilla objetivo** (coordenada) en el tablero del oponente para disparar. Cómo elegir este disparo es la única acción diferente para usuario (que introduce la posición por teclado) y máquina (que se genera aleatoriamente).

3. El jugador **realiza el disparo** sobre el tablero de barcos del oponente (recuérdese que el tablero *opponentShips* para el usuario es el mismo tablero que *myShips* para el oponente). Nótese que no te advierte cuando un disparo se refiere a una coordenada ya disparada. Es procesada justo como cualquier otra.
4. Si un barco (o un trozo de él) **ocupa** (**u ocupó**) la casilla que dispara el usuario, la aplicación debe anunciar un disparo mostrando ***"HIT"*** por consola. En otro caso, si **no ocupa (ni ocupó)** la casilla ningún barco, sino que hay agua, **se muestra** "***MISS***" por consola.
5. **Se comprueba si hay ganador** (si todos los barcos del oponente han sido hundidos) en cuyo caso imprime un mensaje de felicitación o bien se cambia el turno de juego si no hay ganador todavía.

3 Interfaz con el Usuario

Esta sección describe la forma en se debe mostrar en pantalla los dos tableros, cómo el usuario introduce coordenadas y cualquier salida producida por la aplicación. Esta versión usará una **interfaz de usuario basada en texto**.

3.1 Imprimir tableros de usuario

Imprime ambos tableros en consola desde la perspectiva del usuario: *myShips* y *oppoentShip* del usuario. Se deben imprimir **uno al lado del otro**, **con una línea de título identificando cada tablero**, como se indica en la figura 2, rellenando cada casilla con el símbolo correspondiente según se indica en la tabla 1, y separando las casillas por una barra horizontal | (ver figura 3).

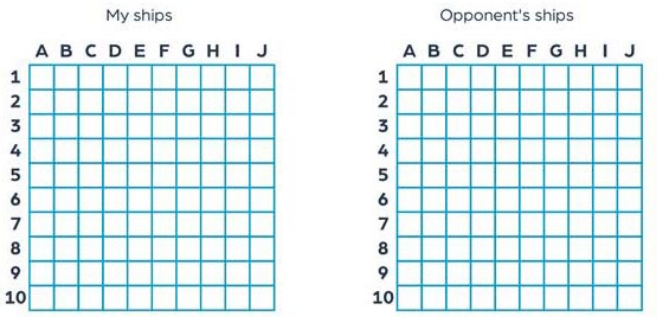


Figura 2: No olvides etiquetar las fila y las columnas en ambos tableros e identificar tableros

La siguiente tabla muestra los **caracteres para mostrar en el tablero**. Nótese que ha diferentes caracteres para cada tipo de barco

Displayed square	Description
BBBB	Battleship (each square a B)
CCC	Cruiser (each square a C)
DD	Destroyer (each square a D)
S	Submarine
*****	Hit
·	Missed shot (\u00B7, middle dot)
Blank space	In the left board, water squares; in the right, those not hit yet

*Tabla 1: Caracteres **para visualizar el contenido** de las cuadrículas*

3.2 Mostrar tablero myShips

Usa B, C, D y S para mostrar barcos y blancos para mostrar casillas que son de agua.

Cuando el jugador *computer* dispara uno de los barcos del usuario, se mostrará el carácter '*' en lugar de la letra correspondiente). Los disparos perdidos hechos por el jugador *computer*, se mostrarán con el carácter

'.'. Imprime '|' para separar columnas. Imprime una cabecera con las letras para reconocer cada columna. Recuerda también imprimir el número de fila al comienzo de la impresión de cada fila.

3.3 Mostrar tablero opponentShips

La salida del tablero del oponente depende de un parámetro del juego (modo de juego).

Por defecto, jugando en **modo NORMAL**, **solo se muestran casillas que el usuario ha disparado**: si contuvieron un barco, usando el carácter '*' mientras que disparos perdidos serán mostrados usando símbolo '.'. El resto de las casillas, es decir **aquellas que aún no se han disparado, serán mostradas como blancos**.

Si el **modo** de juego es **DEPURACIÓN**, entonces **se mostrarán también los barcos del oponente** del mismo modo que los barcos del propio usuario.

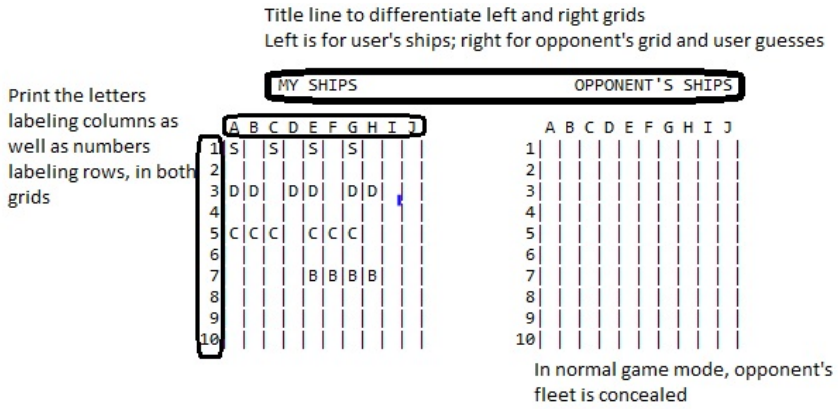


*Figura 3: Pantalla inicial del juego en **modo Normal***

3.4 Entrada del usuario

Cuando se pide al usuario que introduzca las coordenadas, se referirá a la fila y columna objetivo, **utilizando la letra y el número que se visualiza en pantalla** (figura 3).

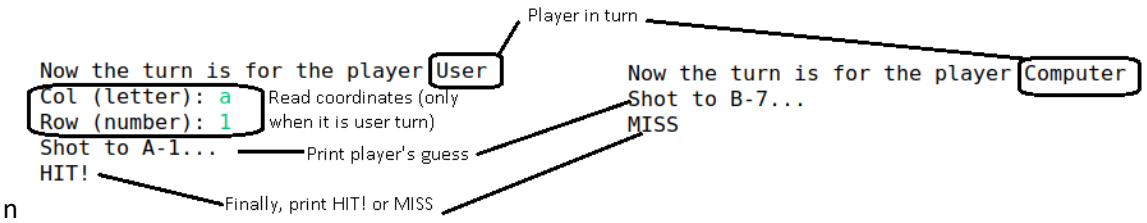
Nótese que hay correspondencia entre las letras mostradas y las columnas indexadas del array, así como los números escritos por el usuario y las filas indexadas del array. Así cuando el usuario escribe columna A fila 4, se traduce en la columna 0 del array (letra A) la fila 3 del array (fila 4 en pantalla).

3.5 Gestión de la interacción con el usuario

Aparte de mostrar los tableros del usuario y preguntar por las coordenadas de disparo, habrá mensajes para gestionar la interacción con el usuario.

- Imprime un mensaje mostrando el **nombre del jugador que tiene el turno**.
- Cuando es el turno del Usuario, **pregúntale por las coordenadas del siguiente disparo**. Pregunta por la columna (letra) primero; después la fila (número).
- Cuando es el turno de la máquina, únicamente **imprime las coordenadas generadas aleatoriamente**, en lugar de leer la entrada.
- Imprime **mensajes con las coordenadas** y finalmente **HIT** o bien **MISS** (dependiendo del resultado del disparo).

Por favor, intenta ceñirte a las siguientes figuras.



n

Interacción con el Usuario utilizando entrada/salida estándar

La siguiente figura muestra que tableros del usuario podrían aparecer después de algunos turnos

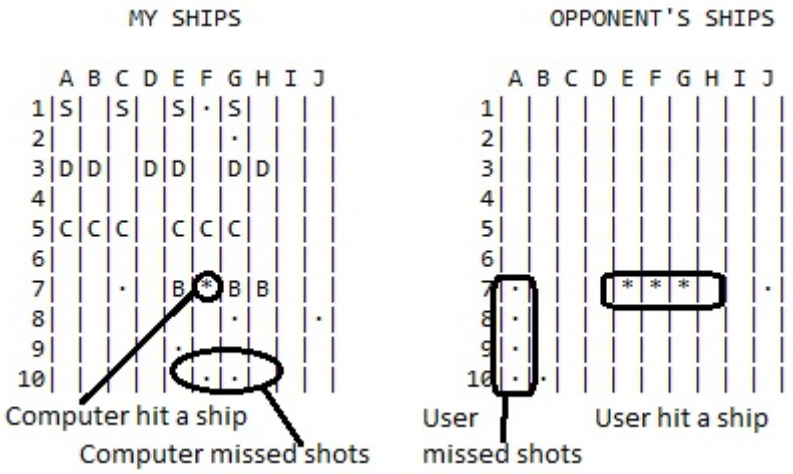


*Figura 1: Ejemplo de salida después de algunos disparos en modo **normal***

La aplicación se comporta igual cuando es el turno de la máquina, excepto que genera una coordenada de disparo (**no se generará nunca una coordenada de disparo ya realizado en otra jugada**) en lugar de leer la entrada desde teclado.

3.6 Juego en modo Depuración

Mientras, por defecto, la flota de la máquina estará oculta para el jugador usuario, debe ser posible, solo por motivo de conveniencia, mostrar no solo los disparos del usuario sino también la localización de los barcos de la máquina, y así el usuario puede jugar conociendo la posición de los barcos enemigos y hundir la flota de la máquina rápidamente.

Por tanto, el juego puede ser jugado en modo normal o depuración y se usará un parámetro para lanzar el juego de un modo u otro.

El modo de juego afecta únicamente a la manera de mostrar, no al procedimiento del juego en sí mismo.

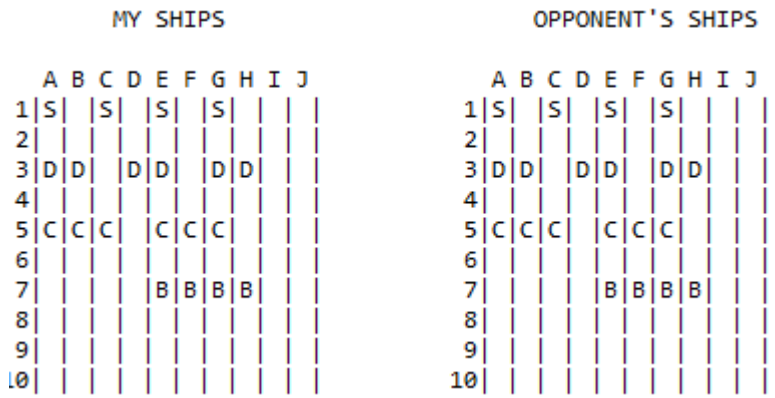


*Figure 2: Pantalla inicial en modo **depuración***

En cada paso del juego, serán siempre mostrados los barcos del oponente (flota de la máquina). Así que, después de algunos turnos, este podría ser un ejemplo de visualización:

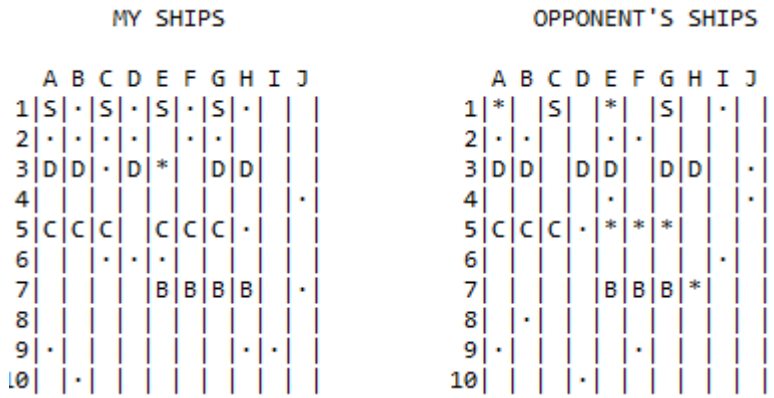


Figure 3: Ejemplo de salida después de algunos turnos en modo depuración

4 Qué hacer

Junto con este documento, se proporciona un Proyecto java mínimo (esqueleto). Éste contiene la clase *Main* con el método *main()* para lanzar el juego.. Renombra el proyecto como:

apellido1_apellido_2_nombre_battleship_sprint1

e **implementa las siguientes clases como se describe más abajo**. Aquí se describen los **métodos públicos** de cada clase, pero puedes (y debes) **usar tantos métodos privados y campos (atributos) como necesites**.

4.1 Clase Game

Ejecuta el bucle principal del juego (turnos) y es la única responsable de la interacción con el usuario.

Métodos públicos:

`Constructor Game(HumanPlayer human, ComputerPlayer computer)`

Recibe un objeto de tipo *HumanPlayer* como primer parámetro y un *ComputerPlayer*, como segundo. Otros atributos del juego como el tamaño de los tableros y el modo de juego toman valores por defecto. Esto es, tableros para jugar de 10x10. Modo de juego Normal.
Además, se deben crear los tableros con los barcos de ambos jugadores y asignarles la referencia a los jugadores. Recuerda que cada jugador guarda el tablero con sus barcos y el tablero del oponente (y viceversa).

`Constructor Game(HumanPlayer human, ComputerPlayer computer, int size)`

Los primeros dos parámetros son los mismos que en el constructor anterior. El tercero se refiere al tamaño de los tableros para jugar en lugar del valor por defecto. Debe estar dentro del rango [5, 15]. Cualquier otro valor será ignorado, y en su lugar será usado el tamaño por defecto. El modo de juego se coloca al modo por defecto.

`void setDebugMode(boolean mode)`

Asigna el modo de juego. Si el argumento es false, se coloca a modo normal (la flota de la máquina estará oculta). Si es true, el modo de juego será depuración que permitirá en su momento mostrar las coordenadas de la flota del oponente.

`void play()`

Es el único responsable de la interacción con el usuario, así como la gestión del bucle principal. En este bucle, los jugadores alternan turnos de disparo mientras no haya ganador.
No importa el turno del jugador, la aplicación se comporta de la misma forma:
Se generan nuevas coordenadas (automáticamente cuando es el turno de la máquina o se recoge desde la entrada estándar cuando es el turno del usuario)
Se guarda un disparo en esas coordenadas en el tablero del jugador oponente.
El resultado del disparo se imprime. Se imprime un mensaje con la información apropiada ("Hit" si hay un barco en esas coordenadas del oponente, o "Miss" cuando no hay barco en ellas. (Véase apartado 2.4).

4.2 Class Board

Esta clase representa un tablero cuadrado donde la aplicación colocará los barcos. Para ello, almacena la localización de los barcos conteniendo una referencia a un array de dos dimensiones de enteros (la cuadrícula (grid)). El contenido de cada celda o elemento de la matriz da información sobre el estado del juego en esa coordenada.

0: casilla agua no disparada; no hay barcos colocados en ella y no ha sido disparada.

Desde 1 a 4: hay un barco o pieza de barco en esta casilla. Si es 1, el barco es un Submarino. En otro caso es parte de un barco mayor. (se describen todos en el apartado 2.2).

De -1 a -4: esta pieza de barco fue disparada.

-10: un disparo perdido (un disparo a casilla con agua).

Métodos públicos:

`Constructor Board(int size)`

En este constructor el tamaño del array será el que reciba como parámetro, size x size.

`int getSize()`

Devuelve un número de filas y columnas de la cuadrícula (la dimensión).

`boolean shootAt(Coordinate coordinates)`

Guarda un disparo en esas coordenadas. Devuelve true si hay un barco en ellas. Esto supone cambiar el contenido de la casilla por el nuevo estado (un valor negativo como se explica en 2.2). Nota que, si el Usuario dispara la misma posición dos veces, la aplicación siempre devolverá true y el contenido de la casilla no cambará del primer al segundo disparo.

`boolean isFleetSunk()`

Comprueba si todos los barcos de la flota están hundidos, en cuyo caso devuelve true; en caso contrario devuelve false.

`List<Coordinate> getNotFiredCoordinates()`

Devuelve la lista de coordenadas del tablero que no han sido disparadas todavía.

`char[][] getFullStatus()`

Devuelve un array de 2D de caracteres representando el estado del tablero. Un carácter en las coordenadas (x, y) representa el estado de esa casilla, como se describe en la tabla 1.

`char[][] getMinimalStatus()`

Como el anterior, devuelve un array 2D de caracteres. Sin embargo, devuelve el valor actual de aquellas casillas en la cuadrícula que han sido ya disparadas. Aquellas que aún no han sido disparadas, devuelven un carácter blanco, independientemente de que contengan un barco o no.

4.3 Class HumanPlayer

Se trata de la clase que almacena el estado del juego del jugador humano. Contiene referencias a dos tableros, objetos de tipo *Board* (*myShips* y *myOpponentShips*).

Métodos públicos:

`Constructor HumanPlayer(String name)`
Crea un nuevo objeto para un jugador humano y guarda el nombre del jugador. Este nombre debe siempre ser diferente de *null*, y de cadena vacía; **en otro caso, lo denominaremos "*user*"**.

`String getName()`
Devuelve el nombre del jugador

`void setMyShips(Board board)`
Asigna el parámetro recibido al tablero *myShips* que contendrá sus barcos y los disparos del enemigo, con el parámetro que recibe (de momento supondremos que nunca será *null*)

`void setOpponentShips(Board board)`
Asigna el parámetro recibido al tablero *opponentShip* que almacena los disparos propios y los barcos del oponente (por el momento, suponemos que nunca será *null*).

`Board getMyShips()`
Devuelve el tablero que guarda *myShips* (con mis barcos y los disparos del oponente).

`Board getOpponentShips()`
Devuelve el tablero *opponentShips* (con los barcos del oponente y mis disparos).

`boolean hasWon()`
Devuelve true si la flota del oponente ha sido hundida; false en otro caso.

`boolean shootAt (Coordinate Coordinate)`
Dispara sobre el tablero del oponente. Devuelve true si el disparo toca un barco y false en otro caso.

`Coordinate makeChoice()`
Lee las coordenadas por teclado y devuelve un objeto de tipo *Coordinate*. El paquete útil contiene algunos métodos útiles para entrada/salida. Debes suponer que el usuario nunca introducirá valores incorrectos.

4.4 Class ComputerPlayer

Es casi lo mismo que la clase anterior (*HumanPlayer*). Las únicas diferencias son en el constructor y el método *makeChoice*.

`Constructor ComputerPlayer(String name)`
Crea un Nuevo objeto y guarda el nombre del jugador. Este nombre debe siempre ser diferente de *null* y cadena vacía; en otro caso, lo denominaremos "***computer***".

`Coordinate makeChoice()`
Genera aleatoriamente coordenadas **que no hayan sido disparadas anteriormente**.

4.5 Class TurnSelector

Maneja los turnos de juego. La primera vez siempre será turno del Usuario. A partir de ahí, los turnos se alternarán, aun cuando un jugador toca o hunde a un barco del oponente.

Métodos públicos:

`Constructor TurnSelector()`
Inicializa el turno de usuario. La primera vez, el turno es siempre para el Usuario.

`int next()`
Devuelve alternado 1 o 0. Cada número representa un turno de jugador diferente: 1 es para el usuario, 0 es para la máquina.

4.6 Class Coordinate

Esta clase representa una posición en el tablero en un array de dos dimensiones.

Métodos públicos:

`Constructor Coordinate (int x, int y)`
Crea un objeto de tipo *Coordinate*, a partir de los parámetros **x para la columna, e y para la fila**.

`Constructor Coordinate (char x, int y)`
Crea un objeto Coordenada a partir de la etiquete que representa la columna (x), que es una letra, y la etiqueta que representa la fila, que es un número (y).

`int getCol()`
Devuelve el valor de la columna.

`int getRow()`
Devuelve el valor de la fila.

`String toString()`
Sobreescribe el método *toString* y devuelve el valor de las coordenadas con el formato: Coordenada [x = 0, y = 1] (por ejemplo)

`String toUserString()`
Devuelve el valor de las coordenadas en un formato más adecuado para el usario. En lugar de Coordenada [x = 1, y = 1] este método devuelve B-2

`boolean equals(Object o1)`
Sobreescribe el método equals. Devuelve true si todos los campos de ambos objetos son iguales. Puede generarse automáticamente con (source/generate equals)

4.7 Class BoardBuilder

Esta clase crea un array cuadrado de dos dimensiones de enteros, y lo rellena colocando números del 1 al 4 en, en posiciones fijas El tamaño de este array viene determinado por el tamaño del objeto *Board*.

Métodos públicos:

`BoardBuilder of(int size)`
Almacena en el atributo size de la clase, el tamaño del array que se va a crear con el valor que recibe como parámetro. Devuelve el propio objeto *BoardBuilder*. Recuerda que *size* debe estar en el rango [5,15].

`int[][] build()`

Devuelve un nuevo array cuadrado de enteros con el contenido fijo que se muestra en **Array 1** (de la sección 2.2). Debería estar invocado antes el método `of` para dar un valor correcto al atributo size. Si no se ha realizado esto, se deberá lanzar una *IllegalStateException* (revisa *util.StateChecks*).

4.8 Class ConsoleInteraction

Esta clase contiene los métodos para visualizar el estado del juego.

Métodos públicos:

`public static void showGameStatus(Board left, Board right, boolean mode)`

Muestra los tableros (la del usuario a la izquierda, la de la máquina a la derecha) de acuerdo con el modo de juego. Intenta ceñirte a la descripción que aparece en la sección 3.3 tanto como te sea posible

5 Tests

A continuación, se describen los test JUnit que debes incluir. Los casos de uso serán comprobados como se describe más abajo. Implementa métodos de test preparando el contexto, ejecutando los métodos correspondientes y comprobando los resultados esperados, en cualquier caso.

5.1 Clase Board

`Método test de shootAt`

Casos de uso:

- Después de disparar a una casilla que aún no ha sido disparada y que no contienen barco, el método devolverá falso y la casilla en cuestión se marcará como disparada. (-10).
- Después de disparar una casilla ya disparada que originalmente no contenía un barco, el método debe devolver false y la casilla permanecerá como disparada (-10).
- Después de disparar una casilla aún no disparada que contiene un barco, el método devolverá true y la casilla cambiará a disparada (su valor en negativo).
- Finalmente, después de disparar una casilla disparada que originalmente contenía un barco, el contenido permanecerá igual y el método devolverá true.

`Método test de isFleetSunk`

Casos de uso:

- Cuando hay varios barcos a flote.
    - Después de un disparo que falla, devuelve false
    - Después de un disparo que toca pero que no hunde uno de los barcos, devuelve false
    - Después de un disparo que hunde uno de los barcos, pero aún quedan otros a flote, devuelve false.
- Cuando solo hay un barco a flote, después de un disparo que lo hunde, devuelve true.

`Test method getNotFiredCoordinates`

Casos de uso:

- Un tablero sin haber sido aún usado, devuelve todas las casillas.
- Un tablero sin haber sido aún usado y después de disparar una casilla que no ha sido disparada aún, devuelve todas las casillas salvo la casilla disparada.
- Retorna lo mismo que en el punto anterior, después de disparar una casilla que ya había sido disparada.

Se pueden utiliza los métodos *equals* y *containsAll* para realizar las pruebas.

6 Requerimientos mínimos para calificar tu trabajo

Antes de entregar, asegúrate de que **tu proyecto se ejecuta sin errores.** Es decir, debes desarrollar una versión simple pero que funcione.

Aparte de estos requisitos mínimos, ten en cuenta lo siguiente:

- Debes escribir Junit para todos los métodos descritos en la sección de test y **deben todos deben pasar la ejecución correctamente**.
- Tu programa **no debe producir ningún tipo de mensaje** o texto a través de la salida estándar o de error estándar **más que los indicados en el enunciado**.
- **Todas las clases deben estrictamente encajar con las interfaces públicas escritas en el esqueleto**: no debes cambiar la signatura de ningún método (nombre del método, número, tipo y orden de los parámetros, y tipo de datos devuelto) o su comportamiento.
- Debes respetar los nombres de los paquetes, jerarquía de paquetes y las clases que pertenecen a cada paquete.
- A menos que se indique lo contrario, la clase principal no debe cambiarse.
- A menos que se indique explícitamente lo contrario, la **validación** de argumentos debe implementarse **para métodos públicos**. En esta etapa, simplemente lanza *IllegalArgumentException* junto con un mensaje adecuado como

`<pon aquí el valor inválido que causó la excepción> es un valor inválido para el parámetro <pon aquí nombre del parámetro>`

7 Envío del trabajo

No tienes que entregar este primer sprint para calificación, pero será la base para el segundo y subsecuentes sprints.