Práctica Evaluable Tema 7.

Utilización avanzada de clases

Objetivos.

- Repasar los conceptos estudiados hasta ahora.

Consideraciones iniciales.

- La práctica consiste en un único proyecto a realizar en Visual Studio o similar, que se evaluará sobre 10 puntos
- Cada clase debe estar en un fichero fuente propio, con el nombre de la clase y extensión ".cs", como en los ejercicios que habéis hecho durante estos temas 6 y 7

Código implementado

Para cada archivo fuente entregado se deberá incluir como comentario en las primeras líneas del archivo el nombre del autor, y una breve descripción de en qué consiste el archivo o clase.

Además, en la clase principal (la que tenga el método Main) se incluirá un listado de todos los apartados, indicando si han sido implementados totalmente, parcialmente o no ha sido realizado.

Por ejemplo:

```
/*
Perez Gomez, Andres
Practica Evaluable Tema 7
Apartado 1 si / no / parcialmente
Apartado 2 si / no / parcialmente
Apartado 3 si / no / parcialmente
...
*/
```

Entrega.

Se debe entregar un archivo comprimido ZIP con el proyecto Visual Studio completo.

- Nombre del archivo: **Grupo_Apellidos_Nombre_PracT7.zip**

  Por ejemplo, si te llamas Andrés Pérez Gómez y eres de 1º de DAM B, el archivo debe llamarse *B_Perez_Gomez_Andres_PracT7.zip*.

Desarrollo.

Ejercicio 1.

Nombre del proyecto: “HundirLaFlota”

Puntuación máxima: 10 puntos

Diseña un programa para jugar al conocido juego de mesa “Hundir la flota”.

- El programa permitirá jugar a un jugador contra el ordenador
- Ambos dispondrán de un tablero de 8 x 8 casillas, donde deberán desplegar sus 4 barcos: un portaaviones, un buque, una fragata y una lancha. Cada barco tendrá un número de casillas a ocupar
- Por turnos, jugador y ordenador irán eligiendo casillas del tablero oponente, y recibiendo como respuesta si en esa casilla había barco o no.
- Ganará la partida el primero que consiga destruir todos los barcos del oponente

Para cumplir este propósito, deberás seguir estas pautas que se indican a continuación.

1. Las casillas (0,5 puntos)

Para gestionar las casillas del juego, crearemos una clase llamada **Casilla** (recuerda, deberás crearla en un archivo llamado *Casilla.cs*). Esta clase tendrá los siguientes **atributos** privados:

- Fila y columna que ocupa la casilla en el tablero (enteros ambos)
- Estado de la casilla, que será un carácter (*char*) que podrá ser agua (la representaremos con un punto '·'), barco (lo representaremos con 'B') o barco tocado (lo representaremos con 'X').

Define un **constructor** que reciba como parámetros la fila y columna y asigne sus valores, y establezca el estado por defecto a “agua”. Define además **propiedades** públicas para acceder a cada elemento privado (consulta el apartado 6.9 de los apuntes del Tema 6).

2. Los tipos de barcos (2 puntos)

Se tendrán cuatro tipos de barcos en el tablero, de forma que cada contrincante dispondrá de un barco de cada tipo.

Para almacenar las características comunes a todos los barcos, crearemos una clase padre abstracta llamada **Barco.** Su único **atributo** (protegido) será un array unidimensional de casillas (objetos de tipo *Casilla* definido antes), con las casillas que ocupará el barco. Además, la clase dispondrá, al menos, de:

- Un **constructor** que recibirá como parámetro el tamaño del array de casillas, e internamente inicializará el array con ese tamaño.
- Un **método** llamado “EstaHundido” que devolverá un booleano indicando si el barco ha sido ya hundido o no. Se deberán recorrer las casillas del barco y ver su estado, para comprobar si todas están tocadas, o si hay alguna que no.
- Además, puedes añadir cualquier otro método o atributo que consideres necesario para todos los barcos en esta clase. Por ejemplo, puede resultar útil un método para asignar las casillas al barco, una vez lo coloquemos en el tablero.

A partir de esta clase *Barco*, heredarán las cuatro subclases concretas, que definiremos en sus respectivos ficheros fuente:

- **Portaaviones**, que llamará al constructor del padre para inicializar el array con 4 casillas
- **Buque**, que llamará al constructor del padre para inicializar el array con 3 casillas
- **Fragata**, que llamará al constructor del padre para inicializar el array con 2 casillas
- **Lancha**, que llamará al constructor del padre para inicializar el array con 1 casilla

Como ves, estas clases únicamente llaman desde su constructor al constructor del padre para indicar su tamaño, no tienen más código.

3. El tablero (4 puntos)

Definiremos también una clase **Tablero** que almacenará la información y la funcionalidad de cada uno de los tableros del juego. Como **atributos**, tendrá los siguientes:

- Un array bidimensional de casillas (objetos de tipo *Casilla*), de 8 filas y 8 columnas
- Un array de 4 barcos (objetos de tipo *Barco*) a colocar en el tablero

(1 *puntos*) En el **constructor**, inicializaremos el array de casillas, poniendo todas en el estado “agua”. Se inicializará también el array de barcos, añadiendo uno de cada tipo (portaaviones, buque, fragata y lancha). Inicialmente estos barcos estarán sin colocar aún en el tablero.

Además, tendremos los siguientes métodos en la clase:

- (*1 punto*) **ponerBarco**, será privado y devolverá un booleano. Recibirá como parámetro el tipo de barco a añadir (a elegir entre portaaviones, buque, fragata o crucero... los puedes identificar con un código numérico, por ejemplo), la fila y columna donde colocar un extremo (enteros) y la orientación (si va a ir en horizontal o en vertical). El método comprobará que las coordenadas y orientación indicadas son correctas (es decir, que el barco no se va a salir del tablero poniéndolo ahí), y que no hay ningún otro barco en el camino que va a ocupar el nuevo. Devolverá *true* si ha sido posible poner el barco en esa posición (y lo dejará puesto ahí) y *false* si no.
- (*1 punto*) **Rellenar**, será publico y no devolverá nada. Internamente, tendrá el código para pedirle al usuario los datos de ubicación de sus barcos. Para cada barco, le indicará que elija el tipo de barco, fila y columna donde colocar un extremo, y orientación, y luego usar el método *ponerBarco* para ubicarlo allí si es posible (si no, se repetirá el proceso para ese barco).
- (1 *punto*) **Generar**, será público y no devolverá nada. Contendrá código para generar aleatoriamente las posiciones y orientaciones de los barcos, y colocarlos en el tablero. Se puede generar aleatoriamente el número de fila y columna (entre los límites permitidos), y la orientación, y luego llamar a *ponerBarco* para ver si es posible, y repetir el proceso para cada barco hasta que se pueda.
- Se pueden añadir, además, los métodos y elementos que se consideren apropiados

4. El juego (3,5 puntos)

El programa principal se almacenará en una clase llamada **Juego** (se deberá renombrar la clase *Program* que se crea típicamente como principal en los proyectos). Dicho programa principal se encargará de:

a) (*0,5 puntos*) Inicializar los tableros

- Crear un tablero para el jugador y otro para el ordenador

- Pedirle al usuario que coloque sus barcos, llamando al método *Rellenar* de su tablero.
- Colocar aleatoriamente los barcos del ordenador en su tablero, llamando al método *Generar* de dicho tablero

b) (*0,5 puntos*) Definir un bucle del juego, donde, en cada iteración, alternativamente, tendrá el turno el jugador y el ordenador (se empezará siempre por el jugador). El bucle finalizará cuando uno de los dos haya ganado.

c) (*1,5 punto*) Si es el turno del jugador, comenaremos por limpiar la consola con `Console.Clear()`. Después, le pediremos que elija una fila y columna del tablero de su oponente, hasta que dichos datos sean correctos. Después, se comprobará si hay algún barco en esa posición del tablero enemigo, y se mostrará uno de estos tres mensajes:

- AGUA (si no había nada)
- TOCADO (si hemos alcanzado un barco, pero aún no lo hemos hundido del todo)
- TOCADO Y HUNDIDO (si hemos alcanzado todas las posiciones de un barco)

Se deberá disponer de algún mecanismo para “marcar” las casillas tocadas del tablero oponente.

Tras mostrar esta información, se imprimirá por pantalla el tablero del ordenador, pero sin revelar las posiciones no tocadas de los barcos (se mostrarán como agua). Se esperará a que el jugador pulse Intro para pasar al siguiente turno.

d) (*1 punto*) Si es el turno del ordenador, también comenzaremos limpiando la consola con `Console.Clear()`, se generará aleatoriamente una fila y columna, y se comprobará si hay algo ahí en el tablero del jugador, mostrando uno de los tres mensajes vistos antes (AGUA, TOCADO o TOCADO Y HUNDIDO).

Tras el mensaje, se mostrará el tablero actualizado del jugador (mostrando los barcos también en este caso) y se esperará a que éste pulse Intro para pasar al siguiente turno.

Ejemplo de funcionamiento

Aquí veis un ejemplo de lo que podría mostrar el juego por pantalla:

1) Iniciando el juego y estableciendo los barcos del jugador:

```
Rellenando barco: PORTAAVIONES
Introduce fila (1 - 8):
2
Introduce columna (1 - 8):
2
Orientación horizontal? (S/N):
S
Rellenando barco: BUQUE
Introduce fila (1 - 8):
5
Introduce columna (1 - 8):
4
Orientación horizontal? (S/N):
N
Rellenando barco: FRAGATA
Introduce fila (1 - 8):
8
Introduce columna (1 - 8):
8
Orientación horizontal? (S/N):
S
No cabe horizontal
Introduce fila (1 - 8):
7
Introduce columna (1 - 8):
7
Orientación horizontal? (S/N):
S
Rellenando barco: LANCHA
Introduce fila (1 - 8):
7
Introduce columna (1 - 8):
2
Orientación horizontal? (S/N):
N
Generando barcos del ordenador...
No cabe vertical
Proceso finalizado. Pulsa Intro para comenzar!
```

2) Partida: turno del jugador:

```
Turno del jugador.
Introduce fila (1 - 8):
3
Introduce columna (1 - 8):
5
TOCADO Y HUNDIDO
· · · · · · · ·
· · · · · · · ·
· · · · X · · ·
· · · · · · · ·
· · · · · · · ·
· · · · · · · ·
· · · · · · · ·
· · · · · · · ·
Pulsa Intro para continuar...
```

3) Partida: turno del ordenador:

```
Turno del ordenador.
Fila 7, Columna 4
TOCADO
· · · · · · · ·
· B B B B · · ·
· · · · · · · ·
· · · · · · · ·
· · · B · · · ·
· · · B · · · ·
· B · X · · B B
· · · · · · · ·
Pulsa Intro para continuar...
```

Observaciones generales de la práctica

- Además de los elementos a añadir indicados en cada clase, puedes añadir otros atributos, constructores o métodos si lo consideras oportuno. Se valorará después si esos elementos añadidos son de utilidad o no.

- Recuerda que se valorará negativamente la suciedad de código, en lo referente a lo visto en el Tema 9 del bloque 1 del módulo de Entornos de Desarrollo. Fundamentalmente, evaluaremos negativamente las siguientes malas prácticas:
  - Nombres poco apropiados de variables, funciones o clases (ya se venía penalizando con anterioridad).
  - Espaciado y alineación vertical (separación entre funciones y entre bloques de código con propósito diferente).
  - Espaciado y alineación horizontal (incluyendo que las líneas de código no excedan del ancho recomendado).
- Recuerda también que, en las funciones que devuelvan algún tipo de dato (*return*), se valorará negativamente que haya más de un punto de salida. Las buenas prácticas de programación indican que las funciones deben tener un único punto de salida (una única instrucción *return*).
- También se valorará negativamente la utilización incorrecta de las estructuras de control, estructuras repetitivas, las instrucciones,... Algunos casos típicos que :
  - Utilización incorrecta de la estructura de control “switch-case”.
  - La instrucción “goto” no debe utilizarse salvo en “switch-case”.
  - Los bucles “for” sólo deberían utilizarse cuando se conoce el principio y fin del buble.