Práctica Evaluable Tema 6.

Programación orientada a objetos

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 este tema 6.

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 6
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: **Apellidos_Nombre_PracT6.zip**

  Por ejemplo, si te llamas Andrés Pérez Gómez el archivo debe llamarse *Perez_Gomez_Andres_PracT6.zip*

Desarrollo.

Ejercicio.

Nombre del proyecto: “JuegosDeGuerra”

Puntuación máxima: 10 puntos

Diseña un programa para jugar al conocido juego de los barquitos o hundir la flota.

- El programa permitirá jugar a un jugador contra el ordenador
- Ambos dispondrán de un tablero de 20 x 20 casillas, donde deberán desplegar sus 10 barcos: 1 portaaviones, 1 acorazado, 3 destructores, 4 fragatas y 1 submarino. 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.

Se deberán cumplir los siguientes requerimientos:

1. (0,5 puntos) La clase **Casilla** para gestionar las casillas del juego. Tendrá los siguientes atributos:

- Fila y columna que ocupa la casilla en el tablero.
- Estado de la casilla, que podrá ser “agua”, “barco” o “barco tocado”, y se representarán como punto “·”, “B” o “X”, respectivamente.

Define un **constructor** que reciba como parámetros la fila y columna y asigne sus valores, y establezca el estado por defecto a “agua”.

2. (1,5 puntos) La clase abstracta **Barco** para almacenar las características comunes a todos los barcos. Cada contrincante dispondrá de sus barcos. Tendrá como atributos**:**

- Un array unidimensional de objetos de tipo *Casilla*, con las casillas que ocupará el barco.
- Un booleano para indicar si se trata de un barco de superficie o no.

Dispondrá, al menos, de los siguientes métodos:

- 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.
- Se puede añadir cualquier otro método o atributo que se considere necesario para todos los barcos. 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 estas subclases:

- Portaaviones, con un tamaño de 5 casillas.

- Acorazado, con 4 casillas..
- Destructor, con 3 casillas.
- Fragata, con 3 casillas.
- Submarino, con 2 casillas.

Estas clases únicamente llaman desde su constructor al constructor del padre para indicar su tamaño.

3. (4 puntos) La clase **Tablero** almacenará la información y la funcionalidad de cada uno de los tableros del juego. Tendrá los siguientes atributos:

- Un array bidimensional de casillas (objetos de tipo *Casilla*), de 20 filas y 20 columnas
- Un array con los objetos de tipo *Barco* a colocar en el tablero

El **constructor** inicializará el array de casillas, poniendo todas en el estado "agua", el array de barcos, añadiendo los de cada tipo. Inicialmente estos barcos no estarán colocados en el tablero.

Dispondrá, al menos, de los siguientes métodos:

- **PonerBarco** devolverá un booleano y recibirá como parámetros:
  - El tipo de barco a añadir, a elegir entre portaaviones, destructor,... Se pueden 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.
- **Rellenar** no devolverá nada. 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).
- **Generar** 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.

4. (3 puntos) El juego.

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

- Inicializar los tableros:
  - Crear un tablero para el jugador y otro para el ordenador.
  - Pedirle al usuario que escoja entre colocar sus barcos, llamando al método *Rellenar* de su tablero, o colocarlos con el método *Generar*.
  - Colocar los barcos del ordenador en su tablero, llamando al método *Generar* de dicho tablero.

- Definir un bucle del juego, donde en cada iteración, alternativamente, tendrá el turno el jugador y el ordenador, empezando siempre por el jugador. Finalizará cuando uno de los dos haya ganado.
- Si es el turno del jugador, comenzará limpiando la consola y mostrará el tablero del ordenador, pero sin revelar las posiciones no tocadas de los barcos (se mostrarán como agua). Después, pedirá que elija una fila y columna del tablero de su oponente, hasta que dichos datos sean correctos. A continuación, comprobará si hay algún barco en esa posición del tablero enemigo, y mostrará uno de estos tres mensajes:
  - AGUA (si no había nada).
  - TOCADO (si se ha alcanzado un barco, pero aún no está hundido).
  - TOCADO Y HUNDIDO (si se han alcanzado todas las posiciones del barco).

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

  Tras mostrar esta información, se mostrará 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.
- Si es el turno del ordenador, también comenzará limpiando la pantalla, se generará aleatoriamente una fila y columna, y se comprobará si hay algún barco en la posición indicada 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, con sus barcos y se esperará a que éste pulse "*Intro*" para pasar al siguiente turno.

5. (1 punto) Además de la implementación del programa siguiendo los pasos indicados, se tendrá en cuenta:

- Las "**Observaciones generales de la práctica**".
- La usabilidad, es decir, que el usuario que lo emplee sepa en todo momento lo que tiene que introducir, y se le muestre la información adecuada.
- La utilización adecuada de métodos y funciones auxiliares.

Observaciones específicas de la práctica.

- Para mostrar datos por pantalla o pedir datos al usuario solo estará permitido utilizar las siguientes funciones de Console: ReadLine, Read, WriteLine, Write, Clear(), SetCursorPosition(x, y).

  SetCursorPosition(x, y): cambia la posición del cursor ("x" se empieza a contar desde el margen izquierdo, empezando en 0, e "y" desde la parte superior de la pantalla, también comenzando desde 0).
- Para la generación de "números aleatorios" será necesario utilizar los conceptos que aparecen en los apuntes del "Tema 9 - Parte 3: Generación de números aleatorios".

Ejemplo de funcionamiento

Iniciando el juego y estableciendo los barcos del jugador:

```
Rellenando barco: PORTAAVIONES
Introduce fila (1 – 20): 2
Introduce columna (1 – 20): 2
Orientación horizontal? (S/N): S

Rellenando barco: ACORAZADO
Introduce fila (1 – 20): 5
Introduce columna (1 – 20): 4
Orientación horizontal? (S/N): N

Rellenando barco: DESTRUCTOR
Introduce fila (1 – 20):  20
Introduce columna (1 – 20):  20
Orientación horizontal? (S/N): S
No cabe horizontal
Introduce fila (1 – 20): 7
Introduce columna (1 – 20): 7
Orientación horizontal? (S/N): S


[…]
Repetir hasta colocar todos los barcos
[…]



Generando barcos del ordenador...
No cabe vertical
Proceso finalizado. Pulsa Intro para comenzar!
```

Una opción interesante sería dibujar el tablero del usuario antes de la introducción de cada barco.

El turno del jugador:

```
Turno del jugador.
Introduce fila (1 - 20): 3
Introduce columna (1 - 20): 5
TOCADO Y HUNDIDO

· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · X X · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
· · · · · · · · · · · · · · · · · · · ·
Pulsa Intro para continuar...
```

El turno del ordenador:

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

Para mayor facilidad de juego se podría dibujar un borde para el tablero de juego y numerar las filas y columnas.

Observaciones generales de la práctica

La **comprobación de los datos** introducidos por la consola:

- Cualquier dato de tipo texto será válido siempre que no esté vacío. Si está vacío, se volverá a pedir de nuevo.
- Cualquier dato de tipo numérico será válido con TryParse o mediante un bloque try-catch-finally y, deberá estar entre los límites acordados según el dato en cuestión. Si no cumple las condiciones, se volverá a pedir de nuevo.

En las **clases y subclases**, salvo en el programa principal, se deberá:

- Definir, al menos, un **constructor** que reciba tantos parámetros como atributos tenga la clase,y, asigne cada parámetro a su atributo correspondiente.
- Definir explícitamente el tipo de **visibilidad** (privado, protegido o público) de los métodos y atributos de cada clase.
- Definir las correspondientes **propiedades** (descriptores de acceso) **get/set** para acceder a cada elemento privado, según las convenciones del lenguaje c#.
- Redefinir (*override*) el método ***ToString*** en las clases en las que se desee mostrar información por pantalla, para indicar el formato de salida.
- A criterio del alumno, añadir otros atributos, constructores o métodos que se consideren oportunos. El profesor valorará si esos elementos añadidos son de utilidad o no.
- Utilizar **un único fichero para cada clase o subclase**, con el nombre de la clase y extensión ".cs".

Se valorará negativamente:

- Los errores en tiempo de ejecución por la introducción de datos incorrectos.
- La repetición innecesaria de código en cualquier parte del programa. Por ejemplo, volver a asignar manualmente en el constructor de una clase hija atributos que ya asigna la clase padre, o repetir el código para pedir al usuario los datos de varios tipos.
- Las **funciones o métodos** excesivamente largos o complejos, como por ejemplo, el programa principal**, que deberían tener dividida su funcionalidad adecuadamente en funciones auxiliares.**
- La utilización de varios puntos de salida ("return") en las funciones que devuelvan algún tipo de dato, con excepción de las funciones recursivas.
- La suciedad de código, en lo referente a lo estudiado en el módulo de Entornos de Desarrollo. Fundamentalmente, se evaluaran las siguientes malas prácticas:
  - Nombres poco apropiados de variables, funciones o clases.
  - 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).
- 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.