



Nombre

Instrucciones para la realización de la prueba:

- Duración de la prueba **2 h 45 m** desde el comienzo de la misma.
- Lee atentamente el enunciado y no formules tus preguntas en voz alta, levanta la mano y espera a que el profesor te diga que tienes que hacer.
- Si te atascas en un apartado déjalo y pasa al siguiente. Cada apartado tiene su puntuación, independientemente del funcionamiento del programa final y el orden propuesto en el enunciado.
- Penalizará con **0,1 décimas** en cada uno de los siguientes casos:
 - Llaves mal puestas e instrucciones mal tabuladas dentro de un bloque.
 - No seguir las reglas de nomenclatura de C#.
 - Identificador de variable o método que a juicio del profesor no esté bien puesto porque no sigue las reglas descritas en clase.
- **No se puede usar ningún tipo de dispositivo USB, ni código de ejercicios o ejemplos realizados en clase. Solo los pdf de apuntes que se te proporcionarán para la ocasión.**
- En caso de pillar a algún alumno copiando o usando material no autorizado se le retirará el examen y su calificación en la evaluación será un suspenso.

Pregunta 1 (8,5 puntos)

Vamos a un programa utilizando POO para representar el valor de una serie de piezas de ajedrez en un tablero.

Nota: Si no se especifica nada supondremos que la accesibilidad de los campos y métodos es **privada**.

1. Definiremos el tipo **enumerado** `ColorAjedrez` que definirá los valores `Blanco` y `Negro`.
2. **(0,5 puntos)** Definiremos la clase **abstracta** `Pieza` que tendrá:
 - Un campo `color` de '*solo lectura*' y del tipo enumerado `ColorAjedrez`.
 - Definirá un **accesor** público que me devolverá el color de la pieza.
 - Definirá un **método abstracto** `double Valor()` público que, cuando definamos concreciones de `Pieza` en las subclases, me devolverá el valor de la pieza.
 - Definirá un **método abstracto** `string Nombre()` público que, cuando definamos concreciones de `Pieza` en las subclases, me devolverá nombre de la misma "Peon" "Caballo", etc.
 - Invalidación del método `ToString()` para que muestre el nombre y color de una pieza. Ej. "*Peon Blanco*"
3. **(0,5 puntos)** Definiremos las siguientes subclases o concreciones de la abstracción `Pieza`:
 - `Peon` con valor de **1**
 - `Caballo` con valor de **3**
 - `Alfil` con valor de **3,5**
 - `Torre` con valor de **5,5**
4. **(0,25 puntos)** Definiremos la clase `Casilla` que tendrá:
 - El campo `pieza` de la clase `Pieza` (Agregación).
 - El campo `color` de '*solo lectura*' del tipo enumerado `ColorAjedrez`.
 - Un **constructor por defecto** público donde la pieza que contiene la casilla la estableceremos a null a **null**.

Importante: Cuando la pieza de una casilla sea **null** significará que la casilla **no contiene ninguna pieza**.

- (0,25 puntos) 'Accesores' y 'Mutadores' público a los campos de las clase, y que **usaremos en el resto de la clase**.

5. (0,25 puntos) Definiremos la excepción personalizada **AjedrezException**.

6. Definiremos la clase **Tablero** que tendrá:

- Definirá la constante entera privada **DIMENSION_TABLERO** que asignaremos el literal entero **8** y que usaremos cuando sea necesario;
- El campo **casillas** que será una **matriz** de la clase **Casilla** de 'solo lectura'. (Es una composición 1 a *) (0,25 puntos) la definición correcta ambos campos y clase.
- (1,5 puntos) Un **constructor por defecto** público que inicializará la matriz con **8 x 8** objetos casilla (sin piezas) y a su respectivo color.

📌 **Nota:** Para la inicialización tendremos en cuenta que la casilla inferior izquierda es **negra**.

• Métodos **públicos**:

- (0,25 puntos) **HayPieza** : Que me devolverá un **bool** que me indicará si hay una pieza o no en una determinada casilla del tablero.
- (0,25 puntos) **VerPieza** : Que me devolverá la **Pieza** situada en una determinada casilla del tablero y en caso de no haber pieza generará una excepción del tipo **AjedrezException** con el mensaje: *"No hay ninguna pieza en la casilla <TextoCoordenada>"*.
- (0,5 puntos) **QuitaPieza** : Que quitará la **Pieza** situada en una determinada casilla del tablero **retornándola** al finalizar la llamada y en caso de no haber pieza generará una excepción del tipo **AjedrezException** con el mensaje: *"No hay ninguna pieza en la casilla <TextoCoordenada>"*.
- (0,5 puntos) **PonPieza** : Que pondrá una **Pieza** en una determinada casilla del tablero y en caso de que la pieza recibida por parámetro sea **null** generará una excepción del tipo del tipo **ArgumentNullException** con el mensaje: *"La pieza a poner en <TextoCoordenada> no puede ser null"*.

Además, si en la casilla donde queremos colocar una pieza ya hubiese una pieza. Lanzaremos una excepción del tipo **AjedrezException** con el mensaje:

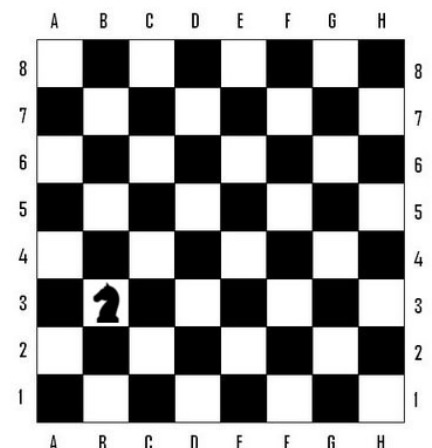
"En la casilla <TextoCoordenada> ya hay una pieza <ToString de pieza>"

Para indicar la '*coordenada*' de la casilla en el tablero en todos estos métodos, usaremos la notación estándar del ajedrez. Esto es, una cadena con la coordenada de la casilla formada por una letra mayúscula de la **A a la H** que indicará la columna y un número del **1 al 8** (de abajo a arriba) que indicará la fila. Ej. **"B3"**

```
Tablero tablero = new Tablero();
Pieza caballo = new Caballo(ColorAjedrez.Negro);
tablero.PonPieza("B3", caballo);
```

Además, estos métodos deberán controlar **mediante una expresión regular** que la cadena que se le pasa para situar la pieza tenga un valor válido dentro del rango del producto cartesiano del tablero y con la letra de la columna en mayúscula. En caso contrario se lanzará una excepción del tipo **AjedrezException** con el mensaje:

"<TextoCoordenada> no es una codificación válida para una casilla del tablero."



👉 **Importante:** Si te fijas bien en el dibujo ten en cuenta a la hora de codificar y decodificar la coordenada de una casilla en ajedrez a la fila y columna de la matriz, que la **letra indica la columna** y el **número la fila** de tal manera que la fila 8 del tablero, es la fila de índice 0 en la matriz. En el ejemplo la casilla **"B3"** es la **fila 5 y columna 2** en la matriz.

💡 **Pista:** Recuerda que `(char)('A' + índice columna matriz)` me devolverá la letra de la columna y una operación similar hará el proceso inverso.

- Además, de la puntuación por definir e implementar correctamente cada uno de los métodos anteriormente descritos, también puntuará:

(1 punto) Definir correcta y eficientemente la expresión regular que controle que es una coordenada válida del tablero.

(1 punto) No **repetir** código de comprobación de la coordenada.

- (1 punto) **Invalidación** del método `ToString()` de la clase `Object` de tal manera que me muestre para ese tablero:

1. Pieza (Nombre Color) seguida de la casilla donde se encuentra (Coordenada Color) por Ej.

`"Caballo Negro en B3 Blanco"` ;

2. Seguida de la información total la suma del valor de las piezas blancas y negras en ese tablero.

Un ejemplo de `ToString()` de un objeto tablero según las especificaciones podría ser:

```
Alfil Blanco en C8 Negro
Peon Negro en B6 Blanco
Caballo Negro en B3 Blanco
Torre Blanco en A1 Blanco
Valor total blancas = 9
Valor total negras = 4
```

7. Usa el programa principal que se te proporciona junto con este enunciado, ajustando los parámetros en tus interfaces si fuera necesario, para al ejecutarlo debes obtener la siguiente salida:

```
Alfil Blanco en C8 Negro
Peon Negro en B6 Blanco
Caballo Negro en B3 Blanco
Torre Blanco en A1 Blanco
Valor total blancas = 9
Valor total negras = 4


Peon Negro en B7 Blanco
Alfil Blanco en H3 Blanco
Torre Blanco en A1 Blanco
Valor total blancas = 9
Valor total negras = 1
```

- (0,5 puntos) Correcta ejecución del programa principal sin errores de compilación.

Pregunta 2 (1,5 puntos)

Realiza un programa simple de consola que al ejecutarlo en el terminal reciba por parámetro la ruta de un fichero HTML a abrir.

1. **(0,5 puntos)** Define un método `static string LeePagina(string pagina)` que reciba el fichero HTML a leer y devuelva en una cadena todo su contenido leído con un `StreamReader`.

 **Nota:** Debes asegurarte que los Streams queden cerrados aunque se produzca un error en la lectura.

2. Define un método `void MuestraCabecerasH2(string html)` que reciba el string con el HTML a leído en el método anterior y muestre el **contenido** de aquellas cabeceras con la etiqueta **h2**.

Ten en cuenta que la sintaxis de la etiqueta **h2** puede ser la siguiente: `<h2 atributos>contenido</h2>` o

`<h2>contenido</h2>` siendo **atributos** y **contenido** cualquier caracter menos el `'\n'`.

(0,25 puntos) Definición del patrón de la ER de la que se extraiga con un grupo etiquetado el contenido.

(0,25 puntos) Recorrido de las coincidencias en la cadena de entrada mostrando el contenido de la cabecera seguido de un salto de línea.

3. **(0,5 puntos)** Realiza un programa principal que capture cualquier `IOException` producida por estos métodos además del número de parámetros pasados a el Main y que muestre las cabeceras del fichero HTML indicado en el parámetro de entrada.

Para poder probarlo junto con este enunciado se te proporciona el archivo `Tema8_2.html` y el resultado de las diferentes ejecuciones **debe ser**:

```
C:\Examen\Ej2\dotnet run Tema8_2.html
Indice
Flujos de datos en serie o streams
Manejo de excepciones con ficheros
Caso de estudio de StreamWriter, StreamReader y excepciones

C:\Examen\Ej2\dotnet run
Debes pasar como argumento el nombre del html a leer.

C:\Examen\Ej2\dotnet run inexistente.html
Could not find file 'C:\Examen\Ej2\inexistente.html'.
```