Universidad Tecnológica Nacional Facultad Regional Avellaneda	UTN AVELLANEDA Tecnicatura Universitaria en Programación

Materia:	**Laboratorio de computación II**											
Apellido[1]:					Fecha:							
Nombre/s[1]:					Docente/s[2]:				**Scarafilo**			
División[1]:					Nota[2]:							
DNI[1]:					Firma[2]:							
Instancia[2][3]:	**P1**		**RP1**		**P2**		**RP2**		**F**	**X**	**RF**	

Parte 1

Crear una base de datos llamada COMBATE_DB con la siguiente tabla:

Personajes
[id - int - not NULL - PK - Auto Increment]
[nombre - varchar - 150 - not NULL]
[nivel - smallint - not NULL]
[clase - smallint - not NULL]
[titulo - varchar - 500 - NULL]

Ejemplo de personaje:

id	nombre	nivel	clase	titulo
1	Falcorn	1	1	Defender of the Alliance

Agregar tantos datos a la base de datos como desee (puede usar chatgpt o mockaroo para hacerlo)

Parte 2

Crear un proyecto del tipo ***Biblioteca de clases*** y agrega los siguientes componentes:

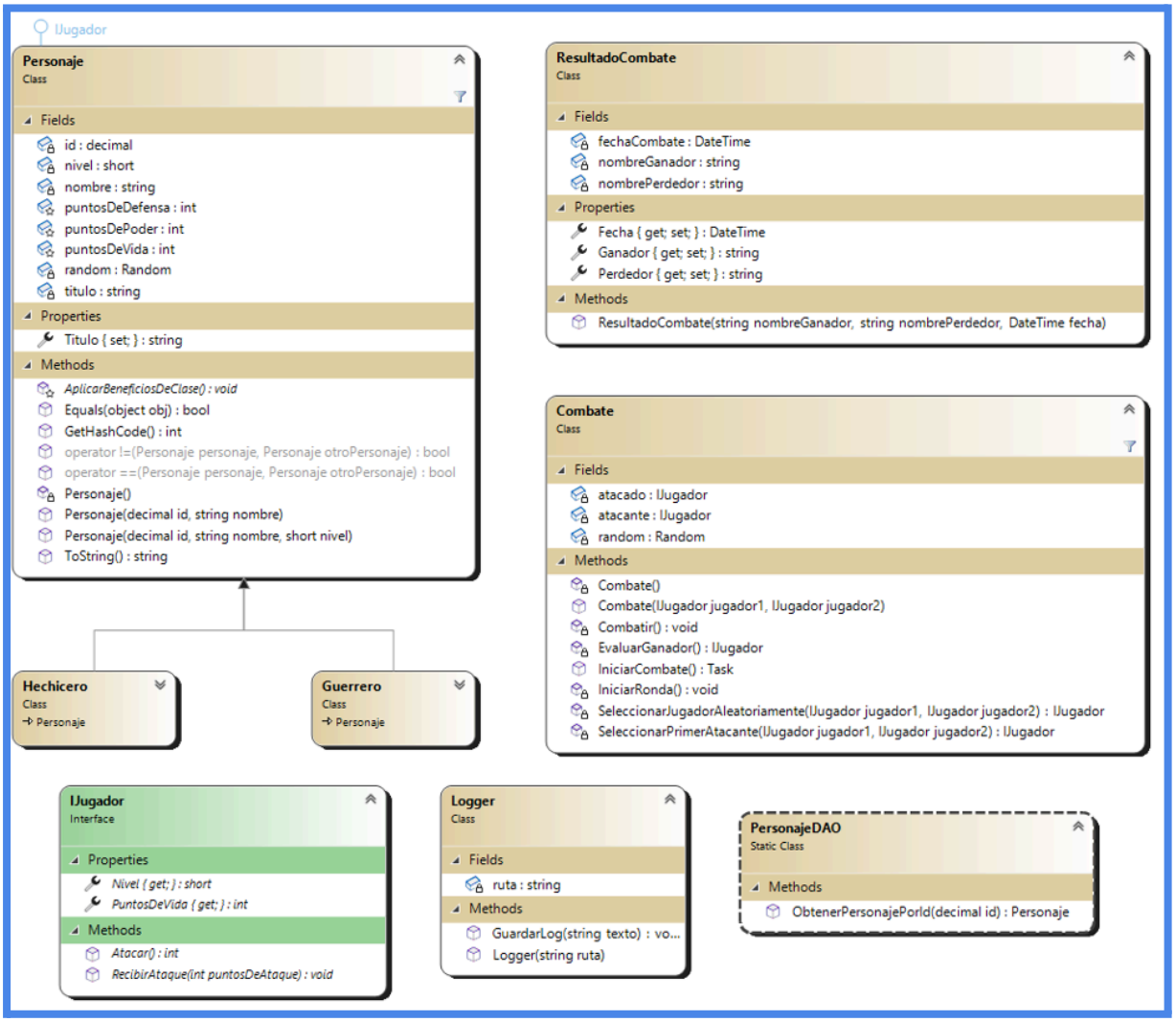
Clase BusinessException:
Crear una excepción personalizada con dos constructores: uno que reciba solo el mensaje y otro que reciba además la InnerException.

Enumerado LadosMoneda:
Tiene dos posibles valores: ***Cara*** con valor 1 y ***Ceca*** con valor 2.

Clase estática Aleatorio:
Definirá un método estático **TirarUnaMoneda** que retornará de forma aleatoria alguno de los valores del enumerado **LadosMoneda**.

Parte 3

Implementar el siguiente diagrama de clases:



El diagrama de clases no está completo, se espera que se agreguen a la solución los elementos necesarios para cumplir con las consignas.

Especificaciones:

Clase Personaje:

- Implementa la interfaz ***IJugador***.
- No se puede instanciar.
- Tendrá una propiedad *Título* de solo escritura que cambia el valor del atributo *título*.
- El atributo random es estático y debe inicializarse en un constructor estático.

- Constructores de instancia:
    - Todos los personajes arrancan con una base de:
        - 100 puntos de defensa.
        - 100 puntos de poder por cada nivel que tenga el personaje.
        - 500 puntos de vida por cada nivel que tenga el personaje.
    - Si se usa la sobrecarga de constructores que no recibe un nivel, por defecto será 1.
    - El constructor debe recibir un nombre que no sea null ni solamente espacios en blanco, de lo contrario lanzar la excepción ArgumentNullException (definida en .NET).
    - Asegurarse que el nombre proporcionado no tenga espacios en blanco al inicio o al final. Si los tuviera, eliminarlos del string.
    - Inicializar el id con el argumento proporcionado.
    - Valida que el nivel se encuentre entre el máximo y el mínimo permitido (incluidos).
        - El máximo de nivel deberá estar definido en una constante de la clase y será 100.
        - El mínimo de nivel deberá estar definido en una constante de la clase y será 1.
        - Si el nivel no es válido, lanzará la excepción personalizada BusinessException con un mensaje descriptivo.
- Dos personajes serán iguales sólo si tienen el mismo id. Cambiar el comportamiento por defecto de las operaciones de comparación: operador ==, método Equals y método GetHashCode. **(Si solo das segundo parcial, crea un método de instancia para tal efecto)**

Si se llama al método GetHashCode de dos valores numéricos (por ejemplo el id del personaje) que sean iguales (tengan el mismo valor), retorna el mismo código hash.

- Definir el método **AplicarBeneficiosDeClase** que debe ser implementado de manera obligatoria por las clases derivadas.
- Tendrá dos eventos llamados ***AtaqueLanzado*** y ***AtaqueRecibido*** respectivamente, cuyos manejadores recibirán un Personaje y un int y no retornan nada.
- Al atacar:
    - Se detendrá el hilo de ejecución por un tiempo aleatorio entre 1 y 5 segundos. **(Solo final)**
    - Retornará los puntos de ataque que tendrán un valor entre un 10% y un 100% de los puntos de poder. El porcentaje a aplicar se debe definir de manera aleatoria.
    - Por último, lanzar el evento ***AtanqueLanzado*** pasándole como argumento la distancia del personaje que está atacando y los puntos de ataque calculados. Solo lanzar el evento si el mismo tiene suscriptores.

- Al recibir el ataque:
    - El personaje se defenderá restando a los puntos de ataque recibidos entre un 10% y un 100% de los puntos de defensa. El porcentaje a aplicar se debe definir de manera aleatoria.
    - Una vez que se ejecutó la defensa se restarán los puntos de ataque resultantes a los puntos de vida del personaje.
    - Los puntos de vida no pueden quedar en negativo, el valor mínimo es 0.
    - Por último, lanzar el evento ***AtaqueRecibido*** pasándole como argumentos la instancia del personaje que está recibiendo el ataque y los puntos de ataque que impactaron efectivamente (luego de aplicar la defensa). Solo lanza el evento si el mismo tiene suscriptores.
- Cambiar el comportamiento del método ***ToString*** para que retorne el nombre del personaje. Si además, el personaje tiene un título, retornará ***"nombre, título".***

Clase Guerrero:

- Deriva de personaje.
- Implementa el método ***AplicarBeneficiosDeClase*** aplicando una bonificación para el personaje de un 10% de puntos de defensa adicionales. Descartar los decimales.

Clase Hechicero:

- Deriva de Personaje.
- Implementa el método ***AplicarBeneficiosDeClase*** aplicando una bonificación para el personaje de un 10% de puntos de poder adicionales. Descartar los decimales.

Clase Combate:

- No puede tener clases derivadas.
- Tiene un evento llamado ***RondaIniciada***. Sus manejadores reciben dos objetos de tipo IJugador y no retornan nada.
- Tiene un evento llamado ***CombateFinalizado***. Sus manejadores reciben un objeto de tipo ***IJugador*** y no retornan nada.
- El atributo random es estático y debe inicializarse en un constructor estático.
- El método de clase ***SeleccionarJugadorAleatoriamente*** lanza una moneda para elegir de forma aleatoria un jugador. Reutilizar código.
    - Si sale cara, retorna al jugador 1.
    - Si sale ceca, retorna al jugador 2.
- El método de clase ***SeleccionaPrimerAtacante*** elige al jugador que ejecutará el primer ataque a partir del siguiente criterio:
    - Si el nivel de los jugadores es diferente, empieza a atacar el jugador con menos nivel.
    - Si el nivel de los jugadores es igual, selecciona uno de forma aleatoria. Reutilizar código.

- El método ***IniciarRonda***:
    - Lanza el evento ***RondaIniciada*** pasándole como primer argumento al jugador atacante y como segundo al jugador atacado. Solo lanza el evento si el mismo tiene suscriptores.
    - Genera un ataque del jugador atacante y lo impacta en el jugador atacado.
- El método ***EvaluarGanador*** retorna al jugador atacante si el jugador atacado tiene cero puntos de vida. De lo contrario si el atacado todavía tiene vida, intercambia roles (el jugador atacante pasará a ser atacado, y el atacado pasará a ser el atacante) y retornará null.
- El método ***Combatir***:
    - Llama a ***InicarRonda*** y luego a ***EvaluarGanador***, repite este proceso hasta que se encuentre un ganador, es decir que ***EvaluarGanador*** no retorne null.
    - Una vez que haya un ganador lanza el evento ***CombateFinalizado*** pasándole como argumento al jugador ganador siempre y cuando el evento tenga suscriptores.
    - Cuando el combate finaliza genera una instancia de ***ResultadoCombate*** y la serializa a formato XML o JSON (a elección del alumno). Se debe instanciar ***ResultadoCombate*** con los siguientes datos:
        - Nombre del ganador (ToString)
        - Nombre del perdedor (ToString)
        - Fecha y hora actual
- El método ***IniciarCombate*** ejecuta ***Combatir*** en un hilo secundario y retorna el objeto ***Task***. **(Solo final) En caso del segundo parcial solo iniciar el combate.**
- El constructor de instancia usa el método ***SeleccionarPrimerAtacante*** para definir cuál de los dos jugadores será el atacante, inicializando dicho atributo con ese jugador. El atacado será, por descarte, el jugador que no haya sido elegido como atacante.

Clase Logger:

- Tiene un constructor de instancia que recibe como argumento la ruta donde se almacenará el log.
- Tiene un método ***GuardarLog*** que guarda el texto recibido como argumento en el archivo del log. No sobreescribir el contenido del archivo.

Clase PersonajeDAO:

- Será estática.
- Su método ***ObtenerPersonajePorId*** consulta la base de datos buscando en la tabla Personajes por el id recibido como argumento, y retorna una instancia de Personaje con los datos recuperados.
    - Si el registro tiene el valor 1 en su columna clase, se deberá instanciar y retornar un ***Guerrero***.

- Si el registro tiene el valor 2 en su columna clase, se deberá instanciar y retornar un ***Hechicero***.
- Si no encuentra nada, retorna null.

Parte 4 (Solo final)

Crear un proyecto de ***pruebas unitarias*** y probar las siguientes funcionalidades:

- Que se lance la excepción ***BusinessException*** cuando se trata de instanciar un Personaje con un nivel no válido.
- Que cuando el Personaje recibe un ataque no quede con puntos de vida negativos, sino en cero.
- Que se inicien correctamente los puntos de defensa para cada tipo de personaje.

Parte 5

Crear un proyecto de consola y reemplazar el contenido de la clase Program con el siguiente código:

```
static void Main(string[] args)
{
    Personaje personaje1 = PersonajeDAO.ObtenerPersonajePorId(1);

    Personaje personaje2 = PersonajeDAO.ObtenerPersonajePorId(2);

    Combate combate = new Combate(personaje1, personaje2);

    Console.WriteLine("¡FIGHT!");

    combate.IniciarCombate().Wait();
}

static void IniciarRonda(IJugador atacante, IJugador atacado)
{
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine();
    Console.WriteLine("--------------------------------");
    Console.WriteLine($"¡{atacante} ataca a {atacado}!");
}



static void FinalizarCombate(IJugador ganador)
{
    Console.ForegroundColor = ConsoleColor.Gray;
    Console.WriteLine();
    Console.WriteLine("--------------------------------");
```

```
    Console.WriteLine($"Combate finalizado. El ganador es {ganador}.");
}

static void MostrarAtaqueLanzado(Personaje personaje, int puntosDeAtaque)
{
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine($"{personaje} lanzó un ataque de {puntosDeAtaque}
puntos.");
}

static void MostrarAtaqueRecibido(Personaje personaje, int puntosDeAtaque)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine($"{personaje} recibió un ataque por {puntosDeAtaque}
puntos. Le quedan {personaje.PuntosDeVida} puntos de vida.");
}
```

Parte 6

- Manejar los eventos ***AtaqueLanzado*** de los personajes usando el manejador ***MostrarAtaqueLanzado***.
- Manejar los eventos ***AtaqueRecibido*** de los personajes usando el manejador ***MostrarAtaqueRecibido***.
- Manejar el evento ***RondaIniciada*** del combate usando el manejador InicarRonda.
- Manejar el evento ***CombateFinalizado*** del combate usando el manejador ***FinalizarCombate***.

Parte 7

- Manejar las posibles excepciones de tipo ***BusinessException*** mostrando su mensaje por salida de la consola.
- Manejar las posibles excepciones de cualquier otro tipo mostrando su mensaje y su ***StackTrace*** por la salida de consola. Almacenar dichos datos en un archivo de texto log.txt, usando la funcionalidad de Logger.