Universidad Tecnológica Nacional Facultad Regional Avellaneda										
Técnico Superior en Programación - Técnico Superior en Sistemas Informáticos										
Materia: Laboratorio de Programación II										
Apellido:				Fecha:		27/07/2017				
Nombre:				Docente(2):						
División:				Nota(2):						
Legajo:				Firma(2):						
Instancia(1):	PP		RPP		SP		RSP		FIN	X

(1) Las instancias validas son: 1er Parcial (**PP**), Recuperatorio 1er Parcial (**RPP**), 2do Parcial (**SP**), Recuperatorio 2do Parcial (**RSP**), Final (**FIN**). Marque con una cruz.
(2) Campos a ser completados por el docente.

Colocar sus datos personales en el nombre del proyecto principal, colocando: Apellido.Nombre.AñoCursada. Ej: Pérez.Juan.2016. **No sé corregirán proyectos que no sea identificable su autor.**

TODAS las clases e interfaces deberán ir en una Biblioteca de Clases llamada *Entidades*.
No se corregirán exámenes que no compilen.

Combate

Criterios de evaluación

- Se deberá entregar un código limpio y acorde a las reglas de estilo de la cátedra.
- Colocar sus datos personales en el nombre de la carpeta principal y la solución: *Apellido.Nombre.Div. Ej: Pérez.Juan.2D*. No sé corregirán proyectos que no sea identificable su autor.
- No se corregirán exámenes que no compilen.
- **Reutilizar** tanto código como crean necesario.
- Aplicar los principios de la programación orientada a objetos.

Consigna

Parte I

Crear una base de datos llamada `COMBATE_DB` y ejecutar el siguiente script:

```
USE [COMBATE_DB]
GO

CREATE TABLE [dbo].[PERSONAJES](
    [id] [int] NOT NULL,
    [nombre] [varchar](150) NOT NULL,
    [nivel] [smallint] NOT NULL,
    [clase] [smallint] NOT NULL,
    [titulo] [varchar](500) NULL,
 CONSTRAINT [PK_PERSONAJES] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY =
OFF) ON [PRIMARY]
) ON [PRIMARY]

GO

INSERT INTO [dbo].[PERSONAJES]
           ([id]
           ,[nombre]
           ,[nivel]
           ,[clase]
           ,[titulo])
     VALUES
           (1
           ,'Falcorn'
           ,1
           ,1
           ,'Defender of the Alliance');

INSERT INTO [dbo].[PERSONAJES]
           ([id]
           ,[nombre]
           ,[nivel]
           ,[clase]
           ,[titulo])
     VALUES
           (2
           ,'NWBZPWNR'
           ,1
           ,2
           ,null);

GO
```

Parte II

Crear un proyecto del tipo **Biblioteca de Clases** y agregarle los siguientes elementos:

Excepción `BusinessException`

- Crear una excepción personalizada `BusinessException` con dos constructores, uno que reciba sólo el mensaje y otro que reciba además la `InnerException`.

Enumerado `LadosMoneda`

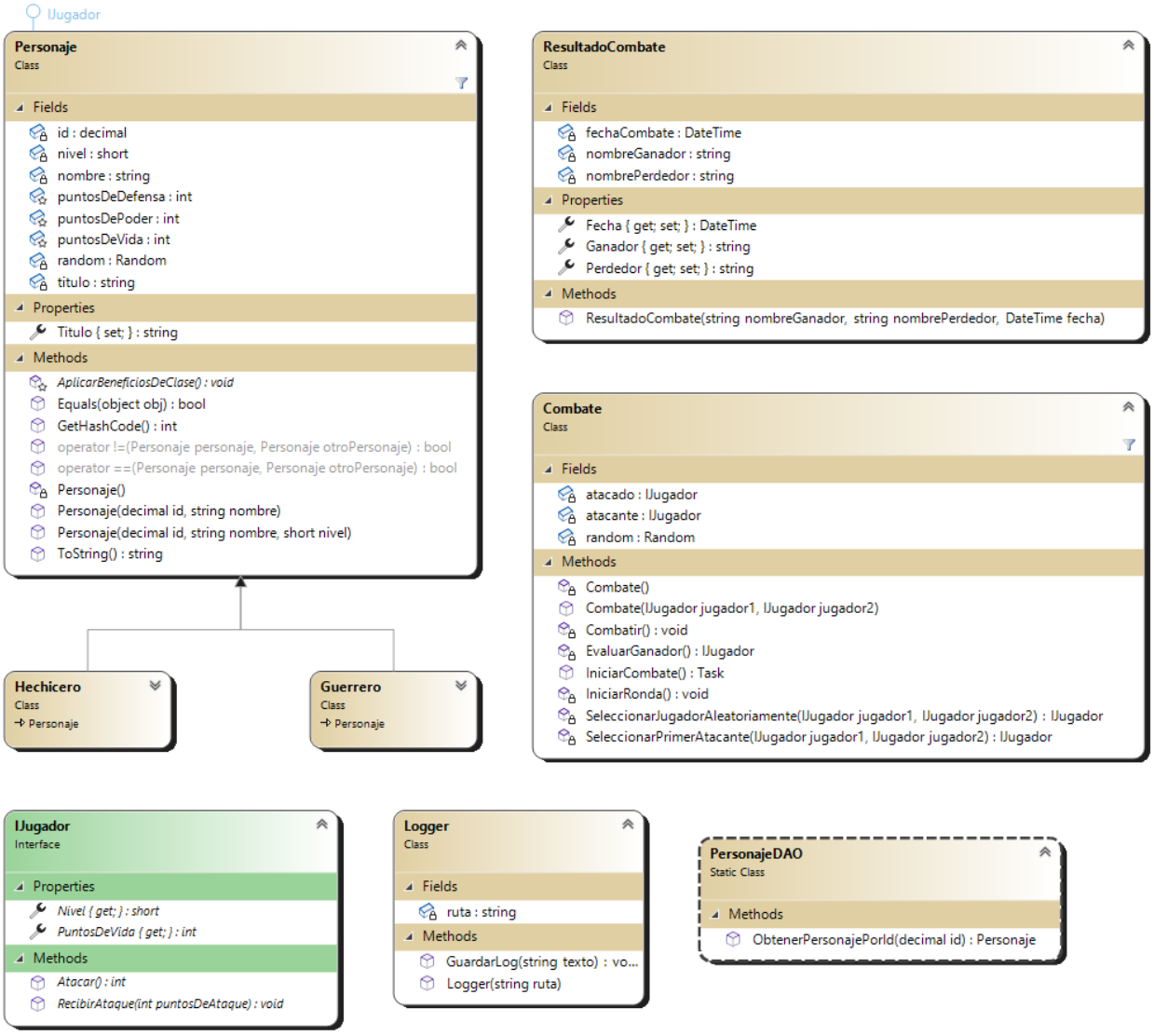
- Tiene dos posible valores. `Cara` con valor `1` y `Ceca` con valor `2`.

Clase `ExtensionRandom`

- Extenderá el tipo `Random` y le agregará el método de extensión `TirarUnaMoneda` que retornará de forma aleatoria alguno de los valores del enumerado `LadosMoneda`.

Parte III

Desarrollar el siguiente esquema de clases:



ALERTA

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

Clase `Personaje`

- Implementa la interfaz `IJugador`.
- No se puede instanciar.
- Tendrá una propiedad `Titulo` de sólo escritura que cambia el valor del atributo `titulo`.
- 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 por cada nivel que tenga el personaje.
        - 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 espacio en blanco, de lo contrario lanzar la excepción `ArgumentNullException` (ya definida en .NET).
    - Asegurarse de 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 permitidos (incluídos).
        - 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`.

TIP

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.

- Define 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 retornarán nada.
- Al atacar:
    - Se detendrá el hilo de ejecución por un tiempo aleatorio de entre 1 y 5 segundos.
    - Retornará los puntos de ataque que tendrán un valor de entre un 10% y un 100% de los puntos de poder. El porcentaje a aplicar se debe definir de manera aleatoria.
    - Por último, lanza el evento `AtaqueLanzado` pasándole como argumentos a la instancia del personaje que está atacando y los puntos de ataque calculados. Sólo lanza el evento si el mismo tiene subscriptores.

- Al recibir un ataque:
    - El personaje se defenderá restando a los puntos de ataque recibidos entre un 10% y un 100% de los puntos de defenza. El porcentaje a aplicar se debe definir de manera aleatoria.
    - Una vez que se ejecutó la defenza, 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 cero.
    - Por último, lanza el evento `AtaqueRecibido` pasándole como argumentos a la instancia del personaje que está recibiendo el ataque y los puntos de ataque que impactaron efectivamente (luego de aplicar la defenza). Sólo lanza el evento si el mismo tiene subscriptores.
- Cambiar el comportamiento por defecto del método `ToString` para que retorne el nombre del personaje. Si además el personaje tiene un título, retonará "*`nombre`*, *`título`*".

Clase `Guerrero`

- Deriva de `Personaje`.
- Implementa el método `AplicarBeneficiosDeClase` aplicando una bonificación para el personaje de un 10% de puntos de defenza 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 debe poder heredarse de ella (no puede ser clase base de otras clases).
- 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 `SeleccionarPrimerAtacante` 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` pasandole como primer argumento al jugador atacante y como segundo argumento al jugador atacado. Sólo lanza el evento si el mismo tiene subscriptores.
    - 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 los roles (el jugador atacante pasará a ser el atacado, y el atacado pasará a ser el atacante) y retorna `null`.
- El método `Combatir`:
    - Llama a `IniciarRonda` 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`.

¿CÓMO QUE TASK? ¿QUÉ ES ESO? ¡YO VI HILOS CON LA CLASE THREAD!

Si no viste `Task` en tu cursada y preferís usar `Thread` no hay problema, retornar el objeto de tipo `Thread` en su lugar y avisar al profesor para que te indique qué modificar en el método main al final del examen.

- 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 sobrescribir el contenido del archivo, anexar.

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 IV

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 inválido.
- Que cuando un `Personaje` recibe un ataque no quede con puntos de vida negativos, sino en cero.
- Que se inicien correctamente los puntos de defenza para cada tipo de personaje.

Parte V

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 VI

- 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 `IniciarRonda`.
- Manejar el evento `CombateFinalizado` del combate usando el manejador `FinalizarCombate`.

Parte VII

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

Resultado esperado

Al completar todos los puntos, compilar y ejecutar la solución se deberá ver una salida similar a la siguiente: