SCRIPTING
INGENIERÍA EN DISEÑO DE ENTRETENIMIENTO DIGITAL
UNIVERSIDAD PONTIFICIA BOLIVARIANA

TALLER PRÁCTICO 2

FECHA: Septiembre 24 de 2025 **PROFESOR:** Andrés Pérez Campillo

NOTAS PREVIAS:

- Por favor **LEA TODO EL ENUNCIADO** antes de responder cualquier pregunta. Si existe alguna imprecisión con la redacción que pueda sugerir ambigüedad en la pregunta o si el enunciado contiene preguntas sobre temas no tratados en el curso o sus prerrequistos, hacer la observación pública y el profesor la atenderá.
- **La fecha límite para hacer subidas al repositorio es el 24 de septiembre de 2025 a las 07:59 hrs. Cualquier subida al repositorio después de esta hora será ignorada**
- **El código de su implementación debe entregarse en un repositorio de GitHub.**
- **La entrega se realizará vía Aula Digital, y bajo ningún concepto se aceptará entrega por otro medio, salvo problemas técnicos generalizados. Si presenta la entrega por fuera del Aula Digital, se asumirá como NO ENTREGADA.**
- Incluya en su solución un archivo *readme*. Detalle los miembros que participaron de la entrega y cualquier consideración que crea pertinente para su revisión. Si usted trabaja en grupo y no hace referencia a sus compañeros, solo se le calificará a quien figure en la entrega.
- **Si su código entregado presenta fallos de compilación la nota será CERO PUNTO CERO (0.0).**
- Puede consultar las notas de clase, material del curso y el manual de referencia de *C#*, o sitios como *dotnetpealrs*, *codeproject*, foros técnicos, etc. **Si utiliza LLMs para solucionar esta evaluación, se asumirá que el código que entregue fue revisado, depurado y corregido por usted; y, en consecuencia, lo domina lo suficientemente bien para una sustentación en vivo si su examen requiere una revisión detallada. Si usted falla en sustentar el código que presenta en una potencial revisión, se considerará caso de fraude.**
- **ESTÁ PROHIBIDO** el uso de cualquier soporte no autorizado por el profesor.
- **La actividad se entregará de manera INDIVIDUAL o GRUPAL HASTA TRES MIEMBROS.**. **Colaborar con algún compañero de clase fuera de su grupo de trabajo, comunicarse por cualquier medio con algún tercero ajeno al curso, solicitar ayuda en foros técnicos durante el tiempo de evaluación está incluido en los soportes NO AUTORIZADOS por el profesor y se constituye en caso de fraude.**
- En caso de fraude, se anulará el examen asignando una nota de **CERO PUNTO CERO (0.0)** y se procederá de acuerdo al Reglamento Estudiantil de Pregrado de la Universidad.

COMPONENTE PRÁCTICO (5.0)

El objetivo de este ejercicio es realizar una implementación aplicando el proceso de Desarrollo Dirigido por Pruebas (*TDD*) a partir de un escenario hipotético. Como tal, procure seguir el proceso, priorizando el diseño de las pruebas unitarias y el correcto funcionamiento del entorno de pruebas del proyecto.

Su entrega consiste en un proyecto de pruebas de NUnit. Entregar código probado vía consola tendrá una reducción del 50% de la nota obtenida en el ejercicio.

Para la implementación, tenga en cuenta los siguientes elementos:

1. Considere los tipos de datos a usar y sus limitaciones según la descripción del problema.
2. Utilice `[TestCase]` y `[Test]` donde lo vea pertinente.
3. Asegúrese de probar que su proyecto de pruebas corra correctamente antes de gastar demasiado tiempo en la implementación. Si tiene inconvenientes técnicos con el *framework* de pruebas, comuníquelo oportunamente.
4. El primer *commit* en su repositorio es el proyecto con las pruebas unitarias diseñadas, pero no hay código que haga aprobar las pruebas (excepto si sus pruebas son llenadas con llamados a *Assert.Pass()* o son definiciones de clases vacías para que su código compile correctamente).
5. Los *commits* subsiguientes en su repositorio son correcciones a la implementación de su diseño, que permitan satisfacer las pruebas solicitadas.
6. Las pruebas deben poder correrse de forma individual o colectiva, sin que ello altere el resultado de las pruebas.

Se le ha asignado la tarea de programar parte del sistema de combates para un clon de Pokémon con fines académicos (por aquello de las patentes). Lo que debe implementar es el módulo de cálculo de daños según el tipo. En general, se siguen las siguientes reglas de diseño:

- Los **[Pokémon]** tienen los siguientes atributos:
    - **Name:** El nombre de la especie de Pokémon
    - **Level (LV):** El nivel del Pokémon (1-99). El valor por defecto es 1.
    - **Attack (ATK):** El factor de ataque del Pokémon (1-255). Se usa en ataques físicos. El valor por defecto es 10.
    - **Defense (DEF):** El factor de defensa del Pokémon (1-255). Se usa en ataques físicos. El valor por defecto es 10.
    - **Special Attack (SpATK):** El factor de ataque del Pokémon (1-255). Se usa en ataques especiales. El valor por defecto es 10.
    - **Special Defense (SpDEF):** El factor de defensa del Pokémon (1-255). Se usa en ataques especiales. El valor por defecto es 10.
    - **Type:** Los tipos de la especie de Pokémon. Puede ser uno solo o una combinación de dos tipos diferentes.
    - **Moves:** Los movimientos que puede usar cada Pokémon. Mínimo debe tener un movimiento, y máximo puede tener cuatro.
- Cada **[Move]** tiene los siguientes atributos:

- **Name:** El nombre del movimiento
- **Base Power (PWR)**: El poder base del ataque (1-255). El valor por defecto es 100.
- **Speed:** El factor que determinará el orden en que se ejecutará el movimiento (1-5). Para este ejercicio, se va a descartar la velocidad del Pokémon en la fórmula. El valor por defecto es 1
- **Type**: El tipo del movimiento. A diferencia de los Pokémon, cada movimiento siempre es de un único tipo. No hay valor por defecto.
- **MoveType:** Si es físico (*Physical*) o especial (*Special*). No hay valor por defecto.

- Los movimientos de los Pokémon siguen los multiplicadores de daño de acuerdo a los siguientes factores:

Attacking \ Defending	Rock	Ground	Water	Electric	Fire	Grass	Ghost	Poison	Psychic	Bug
Rock	1x	0.5x	1x	1x	2x	0.5x	1x	1x	1x	2x
Ground	2x	1x	1x	2x	1x	0.5x	1x	2x	1x	0.5x
Water	2x	2x	0.5x	1x	2x	0.5x	1x	1x	1x	1x
Electric	1x	0x	2x	0.5x	1x	0.5x	1x	1x	1x	1x
Fire	0.5x	1x	0.5x	1x	0.5x	2x	1x	1x	1x	2x
Grass	2x	2x	2x	1x	0.5x	0.5x	1x	0.5x	1x	0.5x
Ghost	1x	1x	1x	1x	1x	1x	2x	1x	2x	1x
Poison	0.5x	0.5x	1x	1x	1x	2x	0.5x	0.5x	1x	1x
Psychic	1x	1x	1x	1x	1x	1x	1x	2x	0.5x	1x
Bug	1x	1x	1x	1x	0.5x	2x	0.5x	0.5x	2x	1x

- Si un Pokémon tiene un tipo, se computará el daño total siguiendo la tabla de tipos de arriba.
  - Ej. Si un Pokémon tipo Fire recibe un ataque de tipo Water, este recibirá el doble del daño del ataque - 2x por el factor [Water vs Fire].
- Si un Pokémon tiene dos tipos, se computará el daño total por cada uno de los tipos.
  - Ej. Si un Pokémon tipo Fire/Ground recibe un ataque tipo Water, este recibirá el *cuádruple* del daño del ataque - 2x por el factor [Water vs Fire] y 2x por el factor [Water vs Ground].
  - Ej. Si un Pokémon tipo Fire/Ground recibe un movimiento tipo Electric, este *no* recibirá daño - 1x por el factor Electric vs Fire, y 0x por el factor Electric vs Ground.
- El cálculo del daño producido por un ataque se calcula por la siguiente fórmula:

$$DMG = \left(\frac{(2 \times \frac{LV}{5} + 2) \times (PWR \times \frac{ATK}{DEF} + 2)}{50} \right) \times MOD$$

$$DMG = \left(\frac{(2 \times \frac{LV}{5} + 2) \times (PWR \times \frac{SpATK}{SpDEF} + 2)}{50} \right) \times MOD$$

Donde ATK y DEF se usan para movimientos *Physical*, mientras que SpATK y

SpDEF se usan para movimientos *Special*; y MOD es el factor de daño de la tabla de tipos.

Como ejercicio de orientación a objetos, usted va a definir 5 especies de Pokémon a su discreción. Todas estas especies deben ser clases separadas, que definen el nombre de la especie y su tipo. Los demás atributos deben ser inicializados usando un constructor por defecto (Ej. `Pikachu pikachu = new Pikachu();`) (1.0)

Para las pruebas, se le pide validar:

1. Al crear instancias de **Pokémon** y **Move**, los valores de los atributos se respetan según las condiciones descritas antes (0.5).
2. Elabore casos de prueba en que calcule el modificador (MOD) esperado para todas las combinaciones de tipos en un ataque. Considere los casos de Pokémon con tipos dobles (1.0)
    a. Utilice instancias de sus especies de Pokémon para probar esto (0.5)

3. Deben probar los siguientes casos de resultados de la fórmula de combate (2.0):

Test Case	Attacking LV	Move PWR	Attacking ATK	Defending DEF	MOD	Expected DMG
1	1	1	1	1	0	0
2	1	1	1	1	1	1
3	5	50	100	50	2	16
4	5	50	100	50	1	5
5	10	20	30	15	1	5
6	12	40	60	80	2	9
7	25	80	120	60	1	40
8	30	100	50	100	4	58
9	40	150	200	150	1	37
10	50	128	200	100	1	58
11	50	128	200	100	4	455
12	60	200	250	200	1	132
13	70	180	200	100	2	435
14	80	90	45	90	1	33
15	90	255	200	50	2	1,554
16	99	255	255	1	2	108,206
17	99	255	255	255	4	856
18	99	255	255	255	0	0
19	99	255	1	255	1	2
20	45	60	10	200	1	2
21	20	30	5	250	1	1
22	2	10	1	255	1	1
23	3	5	2	3	1	1
24	15	200	255	255	1	33
25	16	200	255	254	1	34
26	17	200	255	128	1	36
27	33	77	77	77	1	25
28	48	33	99	11	4	508
29	55	44	88	22	1	44
30	66	11	11	11	1	8
31	77	123	200	100	2	326
32	88	200	100	50	4	1,197
33	10	200	200	200	0	0
34	50	255	100	50	0	0
35	75	180	255	180	0	0
36	99	255	255	1	0	0
37	25	60	40	20	0	0
38	60	100	255	128	1	40
39	80	90	45	90	1	17
40	99	200	150	150	1	84

Sin embargo, para probarlos deben utilizar instancias de Pokémon que satisfagan los parámetros del caso. Ej. Si tiene los siguientes parámetros:

LV = 50 **PWR** = 128
ATK = 128 **DEF** = 128
MOD = 1

Debe utilizar una instancia de Pokémon *attacking* que tenga LV 50 y ATK 128, y usar una instancia de movimiento con PWR de 128 contra un Pokémon *defending* que tenga DEF 128; asegurándose que el tipo del movimiento genere un MOD de 1 contra los tipos del Pokémon *defending*. Utilice intercaladamente movimientos *Physical* y *Special* (Pares e impares, respectivamente).

PISTAS

1. Para los tipos de Pokémon y de movimiento, considere usar enumeraciones (enums).
2. Para los casos de prueba usando instancias de Pokémon, es posible pasar los objetos usando `TestCaseSource`, pero no está obligado a usarlo. Esto facilitaría escribir el código de pruebas usando las instancias de los Pokémon en casos de prueba parametrizados.
3. Las clases a desarrollar, salvo las de pruebas, son mucho más pequeñas de lo que cree.
4. Para los casos que no usen alguna de sus 5 especies definidas, puede simplemente usar la clase base de Pokémon para crear instancias sin restricciones de tipos o parámetros.