**Ejercicio 4 (3 puntos) Tiempo estimado: 60 minutos**.

Existen multitud de juegos de videoconsola, móvil o navegador de internet que se encuentran basados en una competición entre dos
equipos liderados por un entrenador. Cada equipo se encuentra formado no solo por el entrenador, sino también por un conjunto de
combatientes. Sin embargo, solamente los combatientes son los que compiten.

En este ejercicio, vamos a hacer una versión programada en Python para que dos jugadores puedan jugar a este juego. Los jugadores se encontrarán representados por los entrenadores de cada uno de los equipos.

La aplicación programada en Python va a comenzar permitiendo que cada jugador introduzca las características de los combatientes que van a formar su equipo. Cada jugador va a tener como máximo 3 guerreros. Una vez formados ambos equipos se podrá empezar la partida.

No es necesario implementar el docString correspondiente a las funciones y métodos desarrollados, aunque se recomienda hacer el diagrama de flujo de los métodos en papel de forma previa a su resolución.

En base a estas especificaciones se solicita que:

**a) Programe la clase Warrior.**

- Cada guerrero va a estar caracterizado por:
  - Tipo de guerrero. En esta primera versión del juego, solamente 4 tipos de guerrero van a existir (Boxeador, Gladiador, UFC - Ultimate Fighting Championship, y MMA - Mixed martial arts).
  - Tipo de arma que lleva a cabo el guerrero. En esta primera versión del juego, solamente 4 tipos de arma van a existir (Puñetazo, Patada, Codazo, Espada).
  - Puntos de salud que tiene el guerrero. Deben de estar entre 1 y 100.
  - Índice de ataque del guerrero. Deben de estar entre 1 y 10. En base al tipo de arma los valores son: Puñetazo:4, Patada:6, Codazo: 8, Espada: 10.
  - Índice de defensa del guerrero. Deben de estar entre 1 y 10.
  - Nota: Es obligatorio el uso de enumerados para los atributos del tipo de guerrero y tipo de arma. El índice de ataque del guerrero se recomienda que contenga el valor del tipo de arma. Se recuerda que el enumerado en Python tiene el formato "Clave / Valor".
- Incluya los atributos de esta clase y establezca la visibilidad adecuada (público, privado, protegido). Añada cualquier atributo y/o método que considere necesario.
- Programe un **constructor** que reciba los datos necesarios para crear un guerrero. El método **debe verificar** el tipo y valor de cada uno de los parámetros y **lanzar la excepción** correspondiente cuando no se cumplan los requisitos.
- Programe los métodos **setters y getters** para la clase en función de lo que necesite. Si no necesita algún o ningún getter y/o setter, **argumente por qué** en un comentario del módulo.
- Programe el método **is_alive(self)** de la clase Warrior. Este método sirve para saber si el guerrero está vivo.
- Programe el método fight_**attack(self, Warrior warrior_to_attack)**. Método que implementa el ataque del guerrero usando un golpe sobre otro guerrero. Este método se basa en el método fight_defense(self, int points_of_damage) del guerrero atacado. Se aplicará el índice de ataque del guerrero atacante como representación del golpe dado.

- Programe el método fight_defense(self, int points_of_damage).Este método implementa la defensa del guerrero de un golpe de otro guerrero. Esta método actualiza el atributo de puntos de salud que tiene el guerrero en base a los puntos de daño recibidos menos el índice de defensa del propio guerrero. Si el índide de defensa es mayor que los puntos de ataque recibidos, el método devolverá false y el guerrero no sufrirá ningún daño.
- Pruebe los objetos de la clase Warrior con los test cases que se le han pasado. Modifíquelos si lo ve necesario, pero añada una justificación de cada uno de los cambios. Se penalizarán los cambios realizados en los tests proporcionados.

Para ayudar en el desarrollo de este ejercicio, **se le hace entrega de un UML parcialmente completo** de la posible implementación de este juego. Se puede modificar o realizar un UML completamente distinto, el cuál es necesario explicar brevemente en un documento de texto o en los comentarios de un fichero .py. En caso de realizar un UML diferente, no se aplicarán los criterios en base a los casos de prueba facilitados, que es probable que no funcionen sin una adaptación previa.

**Se facilitan también los archivos vacíos** dónde **deberían estar implementadas las clases** que se piden y que tienen que ser completadas por el alumno. En dichas clases, están ya añadidos los casos de tests de cada una de ellas.

**b) Programe una clase Coach.**

- Cada entrenador va a estar caracterizado por:
  - Un nombre que lo identifique (por ejemplo, Raúl, Luis, María,..).
  - La lista de guerreros que forman su equipo. Al menos debería existir un guerrero.
- Incluya los atributos de esta clase y establezca la visibilidad adecuada (público, privado, protegido). Añada cualquier atributo y/o método que considere necesario.
- Programe un **constructor** que reciba los datos necesarios para crear un entrenador. El método debe **verificar el tipo y valor** de cada uno de los parámetros y **lanzar la excepción** correspondiente cuando no se cumplan los requisitos.
- **No es un requisito necesario** la programación de métodos para añadir/eliminar guerreros de un entrenador ya existente.
- Programe los métodos setters y getters para la clase en función de lo que necesite. Si no necesita algún o ningún getter y/o setter, argumente por qué en un comentario del módulo.
- Programe el método is_**undefeated(self)**. Este método sirve para saber si el entrenador está aún sin ser derrotado. Para ello evaluará que al menos uno de sus guerreros esté vivo.
- Programe el método **surrender(self)**. Este método sirve para que el propio Coach se rinda. Para ello tiene que poner los puntos de vida de cada uno de sus guerreros a 0.
- Programe los métodos str y repr para que cada uno muestre la información tal y como se especifica a continuación. El método str debe mostrar la información:
  - [WARRIOR_TYPE_NAME] with a [WEAPON_TYPE_NAME] and health: [HEALTH_POINTS]

  El método **repr** debe mostrar la información:
  - [ID]\t[WARRIOR_TYPE_NAME]\t[WEAPON_TYPE_NAME]
- Pruebe los objetos de la clase Coach con los test cases que se le han pasado. Modifíquelos si lo ve necesario, pero añada una justificación de cada uno de los cambios. Este punto será evaluado inversamente proporcional al número de cambios realizados en los tests cases.

Para ayudar en el desarrollo de este ejercicio, **se le hace entrega de un UML parcialmente completo** de la posible implementación de este juego. Se puede modificar o realizar un UML completamente distinto, el cuál es necesario explicar brevemente en un documento de texto o en los comentarios de un fichero .py. En caso de realizar un UML diferente, no se aplicarán los criterios en base a los casos de prueba facilitados, que es probable que no funcionen sin una adaptación previa.

**Se facilitan también los archivos vacíos** dónde **deberían estar implementadas las clases** que se piden y que tienen que ser completadas por el alumno. En dichas clases, están ya añadidos los casos de tests de cada una de ellas.

**Ejercicio 5 (2 puntos) Tiempo estimado: 20 minutos**.

Se propone una extensión del juego propuesto en el ejercicio 4 en la que, en vez de jugar con un solo tipo de guerrero, se pueda jugar con tres tipos de guerrero diferentes.

Los tres tipos de guerrero que se van a implementar son Guerrero defensivo, Guerrero atacante y Super Guerrero.

En base a estas especificaciones en este ejercicio se solicita que:

a) Programe una clase que implemente un **guerrero defensivo** (Defensive Warrior).
   - Este guerrero va a poseer una característica nueva: Índice de defensa especial del guerrero. Debe de estar entre **11 y 20.**
   - La defensa de este guerrero se implementa de tal forma que **cuando sea este guerrero atacado exista un 50% de probabilidad de poder defenderse mediante una defensa normal o con una defensa especial**. Este 50% se puede calcular utilizando el módulo **random**.
   - En el caso de que el índice de defensa aplicado sea mayor que el índice de ataque recibido, se entenderá como que no hay sufrido un ataque.

b) Programe una clase que implemente un **guerrero ofensivo** (Offensive Warrior).
   - Este guerrero va a poseer una característica nueva: Índice de ataque especial del guerrero. Debe de estar entre 11 y 20.
   - El ataque de este guerrero se implementa de tal forma que cuando **este guerrero esté atacando exista un 50% de probabilidad de poder atacar mediante un ataque normal o con un ataque especial**.
   - Tenga en cuenta que si **se cambia el arma**, se **recomienda también cambiar tanto el índice de ataque normal como el índice de ataque especial del guerrero**. Se puede utilizar un valor aleatorio entre 11 y 20 al cambiar de arma para modificar el índice especial del guerrero.

c) Programe una clase que implemente un **super guerrero** (Super Warrior).
   - Este guerrero es la **combinación del guerrero defensivo y guerrero ofensivo** en un mismo guerrero. Este guerrero implementará el método de ataque del guerrero ofensivo y el método de defensa del guerrero defensivo (ver UML).
   - La utilización de la **herencia múltiple** para la implementación del super guerrero es **totalmente opcional**. **No se penalizará la no utilización de ésta**.

Reutilice tanto código como sea posible del ejercicio 4 para hacer esta extensión del juego.

No es necesario implementar el docString correspondiente a las funciones y métodos desarrollados, aunque se recomienda hacerlo para facilitar la comprensión por parte del estudiante.

**Ejercicio 6 (2 puntos) Tiempo estimado: 30 minutos.**

En este último ejercicio se pide que se implemente el módulo principal del juego. Este módulo principal va a implementar la partida que van a jugar los jugadores, los cuales se encuentran identificados como los entrenadores de los combatientes.
En este módulo main, lo primero que se hace es obtener la configuración deseada de los guerreros por parte de cada entrenador. Hay que tener en cuenta, que cada entrenador solamente va a tener como máximo tres combatientes al iniciar la partida.

Las **características de cada uno de los grupos de combatientes** que va a poseer cada entrenador van a ser introducidas **a través de archivos CSV**. Los dos archivos CSV a utilizar por en este ejercicio han sido proporcionados:

- **coach_1_warriors.csv**
- **coach_2_warriors.csv**

Se deben crear los guerreros leyendo los ficheros csv para cada uno de los entrenadores (ver Flowchart).

Seguidamente, una vez que se tienen las configuraciones de cada uno de los equipos, el combate puede comenzar entre los entrenadores.

Los combates entre guerreros se van a realizar secuencialmente. Además, un combate se entiende como aquel llevado a cabo por un guerrero de cada uno de los equipos. Por lo tanto, en cada turno un guerrero de uno de los equipos va a atacar a otro guerrero del otro equipo. Una vez que dos guerreros entran en combate, no se para este combate hasta que uno de los dos guerreros es derrotado, momento en el que el siguiente guerrero del entrenador debe entrar al combate, solicitando al entrenador que seleccione un guerrero de aquellos que estén vivos.

Los entrenadores van a elegir el orden en que los guerreros son puestos en combate. Para ello, se debe listar al jugador la lista de guerreros vivos y no seleccionados aún y éste debe introducir el id de cada guerrero. Se recomienda crear una estructura de datos independiente que contenga únicamente los guerreros del entrenador vivos y que se vayan sacando de dicha estructura una vez hayan sido batidos/muertos. Una vez que el guerrero es derrotado, no puede volver a entrar en combate.

Cuando uno de los dos entrenadores tenga ya todos sus guerreros derrotados al acabar el turno, se acaba el juego, y se indica no solo quién es el ganador, sino también las estadísticas de cada equipo en base a los puntos de vida de los guerreros. En el caso de que los dos acaben sin guerreros en el mismo turno, el juego indicará un empate. En cada turno los dos guerreros atacan con independencia de si en ese mismo turno es derrotado.

Recuerde utilizar las clases implementadas tanto en el ejercicio 4 como en el ejercicio 5 para implementar todos los componentes de este juego.

El diagrama de flujo de este módulo principal le ha sido proporcionado a modo de guía.