U6P03-Programas_03. Orientación a objetos

1. Descripción

Desarrolla y prueba en Python los programas que permitan resolver los requisitos planteados.

Dentro de la carpeta **PEPXX** crea las subcarpetas **U6/03_orientación_a_objetos .** Para cada programa crea un archivo que contenga: al inicio comentando el enunciado y a continuación la solución.

Puedes crear un repositorio en GitHub con todo el código.

2. Formato de entrega

Será propuesto en clase por el profesor.

3. Trabajo a realizar

El objetivo es crear un programa para gestionar una “batalla de personales fantásticos”. Tiene que existir un módulo con todas las clases y un programa con una función `main`.

Vas a desarrolla el programa en varias fases.

3.1.1. Clase base Personaje (Herencia básica)

Clase abstracta llamada `Personaje` que represente a un personaje genérico del juego.

- Atributos
  - `nombre`
  - `vida`
- Métodos `@property` para
  - `nombre (solo lectura)`
  - `vida (lectura y escritura, nunca debe quedar negativa).`
  - `vivo (solo lectura, devuelve True/False)`
- Un método abstracto:
  - `atacar(self, objetivo)` que será redefinido por las subclases.

3.1.2. Subclase Guerrero y composición con Arma

Clase `Arma`

- Atributos
  - `nombre`
  - `danio (solo lectura)`

- Métodos `@property` para
  - `nombre (solo lectura)`
  - `danio (solo lectura)`

Clase `Guerrero`, heredando de `Personaje`.

- Atributos:
  - Añade un atributo `arma` (objeto `Arma`)
- Métodos
  - Implementa el método `atacar()` para
    - Calcular el daño como daño base del arma + pequeño bonus aleatorio.
  - Restar ese valor a la vida del objetivo.

3.1.3. Subclase Mago con un diccionario de hechizos

Crea la clase `Mago`, heredando de `Personaje`.

- Atributos:
  - Un `diccionario de hechizos` con formato:

```
{
   "Bola de fuego": 18,
  "Rayo": 22
}
```

- Métodos
  - Implementa el método `atacar()` seleccionando un hechizo al azar y aplicando su daño al objetivo.

3.1.4. Uso correcto de properties en todas las clases

Asegúrate de que:

- `Personaje` controla la vida mediante un setter.
- `Arma, Guerrero y Mago` exponen sus atributos mediante `@property`.

3.1.5. Polimorfismo

Demuestra que:

- Tanto `Guerrero` como `Mago` implementan su propia versión de atacar.
- El programa debe poder ejecutar: `p1.atacar(p2)` sin saber si `p1` es un `Mago` o un `Guerrero`.

3.1.6. Sistema de combate por turnos

En el programa principal implementa una función llamada `combate(personaje1 , personaje2)` que:

- Muestre el inicio del combate.
- Mientras ambos personajes estén vivos:
  - `personaje1` ataca a `personaje2`.
  - Si `personaje2` sigue vivo, `personaje2` ataca a `personaje1`.
- Declare un ganador mostrando su vida restante.

3.1.7. Simulación y demostración final

En el programa principal crea:

- Un `Guerrero` con un `arma`.
- Un `Mago` con un diccionario de hechizos.
- Invoca a `combate(guerrero, mago)`.

3.1.8. Opciones de ampliación

Si quieres puedes completar la práctica con opciones extra:

- Añade mana al mago y coste de hechizos.
- Permite cambiar el arma del Guerrero.
- Añade clase `Arquero` con su forma de atacar.
- Guardar y cargar personajes en ficheros JSON.
- Guardar y cargar personajes en una base de datos.
- Modo torneo: todos contra todos.