Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2018-2

9 de septiembre de 2018
No Evaluada

Actividad 04

Programación Orientada a Objetos 101

Introducción

Luego de muchos años siendo ayudante del ramo de Programación Avanzada, el malvado mafioso Tini Tamburini fue otorgado con el poder de la omnipotencia, por lo que ahora puede controlar directamente todo el universo. Sin embargo, el malvado Tini se encuentra muy ocupado manejando a la mafia, por lo que no tiene el tiempo para realizar el trabajo de modelar su nuevo universo, por esto le pide a los alumnos de su ramo favorito que modelen un mundo de ficción como a él le gustaría.

Para lograr esto, él te pide que modeles a los **aventureros**, **monstruos** y **clanes** del nuevo mundo.

Clases

Para lograr satisfacer al todopoderoso jefe de la mafia, deberás modelar las clases que él te ha pedido, las cuales son

- **Aventurero:** Estas personas son aventureros que no están especializados en algún área específica de pelea. Cada uno tiene un `nombre`, una cantidad de `vida` y atributos básicos como `ataque` y `velocidad`. El `poder` de un aventurero es simplemente la suma de su vida, su ataque y su velocidad. La vida de todos los aventureros **siempre** será un número entero que estará entre 0 y 100. Es importante que si la vida baja a negativo, entonces se queda en 0 y si sube a un número mayor a 100, entonces quedará en 100.
- **Guerrero:** Estos aventureros están especializados para pelear físicamente. Su largo entrenamiento les permite usar escudos, por lo que poseen el atributo de `defensa`. Su poder se calcula

$$0{,}8 \times vida + 2{,}2 \times ataque + 1{,}5 \times defensa + 0{,}5 \times velocidad$$

- **Mago:** Estos aventureros están especializados para combatir con magia. Su extenso conocimiento arcano les otorga el atributo de `magia`. Su poder se calcula

$$vida + 0{,}1 \times ataque + 2{,}5 \times magia + 1{,}4 \times velocidad$$

- **Monstruo:** Los monstruos son manifestaciones de maldad pura, por lo que tienen cuatro atributos, `nombre`, `vida`, `poder`, `jefe`. Jefe es un booleano que representa si el monstruo es uno de los jefes de una mazmorra o no, si es `True`, entonces el poder del monstruo se multiplica por 3, la `vida` del monstruo posee un mínimo de 0 al igual que los aventureros, sin embargo no posee un límite superior

- **Clan:** Los clanes son asociaciones de aventureros. Cada clan posee un nombre, una lista de los aventureros y un rango, el cual puede ser:
  - `Bronce`: Si el clan tiene de 0 a 2 aventureros
  - `Plata`: Si el clan tiene de 3 a 5 aventureros
  - `Oro`: Si el clan tiene 6 o más aventureros

  Finalmente, cada clan tiene un `poder` correspondiente, el cual se calcula

  $$\sum_{i=1}^{n} p_i \times a$$

  Con p_i el poder individual de cada aventurero del clan y a un ponderador, que es
  - 0.5 para clanes con rango `Bronce`
  - 0.75 para clanes con rango `Plata`
  - 1.2 para clanes con rango `Oro`
- **Mazmorra:** Las mazmorras son congregaciones de monstruos, por lo que son casi exactamente igual a los clanes, excepto por que tienen monstruos en vez de aventureros. Poseen `nombres` y una `lista de monstruos`, la cual se utiliza para calcular el `poder` de la mazmorra como la suma del `poder` de todos los monstruos dentro de ella.

Métodos

Para hacer el código un poco más sencillo de manejar, deberás agregar los siguientes métodos a las clases pedidas.

- `grito_de_guerra(self)`: Todos los aventureros deben poseer este método, cuando se ejecute se deberá imprimir en pantalla el nombre del aventurero con : y luego *"¡Gloria al gran Tini!"*
- `agregar(self, entidad)`: Clanes y mazmorras deben poseer este método, cuando se ejecute se deberá revisar que el argumento sea una persona/monstruo caso de los clanes/mazmorras respectivamente y luego agregarlo a la lista. Si el argumento no es del tipo correcto se debe imprimir un mensaje que diga: "No es posible agregar monstruos a los clanes", o "No es posible agregar personas a las mazmorras".
- `remover(self, entidad)`: Se debe agregar a los clanes y las mazmorras, al ejecutar este método se deberá encontrar la entidad en la lista de personas o monstruos y removerlo. En caso de no estar la lista se debe imprimir el mensaje: "Entidad no encontrada".

Métodos mágicos (*dunder methods*)

El malvado Tini te pidió que le agregaras algo de magia a tu código, por lo que le deberás agregar los siguientes métodos mágicos a las clases pedidas para satisfacer su petición.

- `__add__(self, other)`: tanto los clanes como mazmorras deben tener este método, al sumar dos clanes o mazmorras se creará un tercero con todos los `miembros` de ambas entidades y con `nombre` igual a la concatenación de los dos `nombres` anteriores. Tengan en cuenta que no se pueden sumar clanes con mazmorras.
- `__str__(self)`: todas las clases deberán poseer este método, cuando se ejecute deberá devolver:

- `Mazmorras:` El nombre y poder total
- `Aventureros:` El nombre y poder del aventurero
- `Clanes:` El nombre, poder total, rango y cantidad de personas
- `Monstruo:` El nombre del monstruo y su poder

No hay un modo específico de mostrar la información pero debe quedar clara a la hora de imprimirla.

Todas las clases deberán poseer los siguientes métodos. Un objeto se considera menor que otro si el poder de este es menor que el poder del segundo (Ver la descripción de cada clase para saber como se calcula el poder). Esto le deberá permitir al doctor utilizar sort() en una lista con clanes, mazmorras, heroes y monstruos sueltos para que se ordenen de más debil a más fuerte. a

- `__lt__(self, other):` retorna `True` si el poder del primer objeto es menor que el del segundo. De no ser así, retorna `False`.
- `__eq__(self, other):` retorna `True` si el poder de ambos objetos es igual. De no ser así, retorna `False`.
- `__gt__(self, other):` retorna `True` si el poder del primer objeto es mayor que el del segundo. De no ser así, retorna `False`.

Notas

- Para revisar si un determinado `objeto` es una instancia de la clase `Clase`, puedes usar `isinstance(objeto, Clase)`, esto les puede ayudar ya que si una clase es herencia de una clase base, entonces todas sus instancias también son instancias de la clase base.
- En esta actividad es importante analizar qué cualidades de cada clase deben ser atributos y cuáles corresponden a *properties*.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC04/`
- **Hora del último *push*:** 16:00