Williams, Walter

```
from agua import *
from fuego import *
from hierba import *
from entrenador import *

pokemon1 = Agua("Squirtle")
pokemon1.mostar_pokemon()
entrenador1 = Entrenador("daira",random.randint(0,100), Agua("Squirtle"))

lista_pokemon = []

for i in range(10):
  tipo = random.randint(1,3)
  if tipo == 1:
    pokemon_agua = Agua("Pokemon {}".format(i))
    lista_pokemon.append(pokemon_agua)
  elif tipo == 2:
    pokemon_fuego = Fuego("Pokemon {}".format(i))
    lista_pokemon.append(pokemon_fuego)
  else:
    pokemon_hierba = Hierba("Pokemon {}".format(i))
    lista_pokemon.append(pokemon_hierba)

for pokemon in lista_pokemon:
  entrenador1.atrapar_pokemon(pokemon)
entrenador1.mostrar_pokedex()

class Entrenador:
  def __init__(self, nombre, nivel, pokemon):
    self.__nombre = nombre
    self.__nivel = nivel
    self.__pokemon = pokemon
    self.__pokedex = []

  def atrapar_pokemon(self, pokemon):
    if self.__nivel > pokemon.get_salvajismo():
      for i in range(3):
        pokemon.ataque(self.__pokemon)
        if pokemon.get_vida() <= 0:
          print("No se pudo atrapar el pokemon")
          break
        if self.__nivel > pokemon.get_salvajismo():
          print("Pokemon atrapado")
          self.__pokedex.append(pokemon)
          break
        pokemon.set_salvajismo(pokemon.get_salvajismo() - pokemon.get_salvajismo() * 0.1)
    else:
      print("No se pudo atrapar el pokemon")

  def mostrar_pokedex(self):
    print("")
    print(f"Entrenador: {self.__nombre} Nivel: {self.__nivel}")
```

Comentado [1]: No se usa

Comentado [2]: Bien

Comentado [3]: Tambien se podia hacer f"Pokemon {i}"

Comentado [4]: Bien la visibilidad de los atributos

Comentado [5]: Esta condicion no hacia falta se debia pasar directamente a la logica de intento de ataque

Comentado [6]: Bien la logica general

Comentado [7]: Esta logica se puede trasladar al metodo de defensa, dado que siempre que el pokemon reciba un ataque se reducira su salvajismo

Comentado [8]: Dado que esto no solo muestra el listado de pokemon, lo renombraria a imprimir

Williams, Walter

```
    print("Pokedex:")
    for pokemon in self.__pokedex:
        pokemon.mostar_pokemon()

import random
from abc import ABC
class Pokemon(ABC):
  def __init__(self, nombre,tipo,debilidad,vida=100):
      self._nombre = nombre
      self._vida = vida
      self._tipo = tipo
      self._debilidad = debilidad
      self._ataque = random.randint(0,100)
      self._defensa = random.randint(0,100)
      self._velocidad = random.randint(0,100)
      self._salvajismo = random.randint(0,100)

  def get_vida(self):
      return self._vida

  def set_vida(self,vida):
      self._vida = self._vida - vida

  def get_tipo(self):
      return self._tipo

  def get_ataque(self):
      return self._ataque

  def get_salvajismo(self):
      return self._salvajismo

  def set_salvajsmo(self,salvajismo):
      self._salvajismo = self._salvajismo - salvajismo

  def recibir_danio(self, danio, oponente):
      oponente.set_vida = oponente.get_vida() - danio

  def ataque(self, oponente):

      if oponente.get_tipo == self._debilidad:
          probabilidad = random.randint(1,100)

          if probabilidad <= 70:
              self.recibir_danio(self._ataque + self._ataque * 0.5, oponente)
      else:
          self.recibir_danio(self._ataque, oponente)

  def mostar_pokemon(self):
      print(f"Nombre: {self._nombre} Tipo: {self._tipo} Ataque: {self._ataque} Defensa: {self._defensa}
Velocidad: {self._velocidad} Salvajismo: {self._salvajismo}")
```

Comentado [9]: No es necesario pasar la vida segun el enunciado, se podia asignar 100 directamente al atributo dado que siempre tendra ese valor

Comentado [10]: - Bien la visibilidad de los atributos y la asignacion de sus valores
- Cuidado con la indentacion en python, aca no afecta, pero una mala indentacion puede derivar en errores de ejecucion

Comentado [11]: No definir getters y setters si no van a tener uso

Comentado [12]: Esto no es la logica de un setter. Los setters se usan para asignar el valor tal cual se recibe al atributo. Si tiene otro tipo de logica no debe llamarse set_atributo

Comentado [13]: Idem a set_vida

Comentado [14]: La logica que se intenta colocar aca deberia estar resuelta en el metodo de defensa, donde oponente no llegaria por parametro sino que seria la referencia "self".
El metodo de ataque debe calcular el daño a aplicar sobre el objeto atacado e invocar al metodo defensa de ese objeto.

Comentado [15]: Bien por definir un metodo general compartido de ataque
Pero la implementacion esta mal porque get_tipo es un metodo, no una variable o atributo, por lo tanto debe llamarse/invocarse asi: get_tipo() no get_tipo, para poder acceder al valor devuelto
Tambien falta atacar cuando no hay aumento del daño por la probabilidad del 70%

Comentado [16]: Tambien se podia multiplicar (ataque * 1.5)

Comentado [17]: Este metodo estaria mejor nombrado como imprimir

Williams, Walter

```
from pokemon import *

class Agua(Pokemon):
  def __init__(self, nombre):
    super().__init__(nombre, "Agua", "Hierba")

  def ataque(self, oponente):
    if oponente.get_tipo == self._debilidad:
        self.recibir_danio(self._ataque * 0.7,oponente)
    else:
      self.recibir_danio(self._ataque,oponente)

  def defender(self,oponente):
    if self._defensa < oponente.get_ataque:
      if random.randint(1,100) <= 30:
        self.vida_restante(oponente.get_ataque * 0.5)
      else:
        self.vida_restante(oponente.get_ataque)

from pokemon import *
class Fuego(Pokemon):
  def __init__(self, nombre):
    super().__init__(nombre, "Fuego", "Agua")

  def defender(self,oponente):
    if self._defensa < oponente.get_ataque:
      self.vida_restante(oponente.get_ataque)

from pokemon import *
class Hierba(Pokemon):
  def __init__(self, nombre):
    super().__init__(nombre, "Hierba", "Fuego")

  def defender(self, oponente):
    if self._defensa < oponente.get_ataque:
      if self._velocidad > 50:
        if random.randint(1,100) >= 50:
          self.vida_restante(oponente.get_ataque)
      else:
        self.vida_restante(oponente.get_ataque)
```

Devolución

- Herencia: Más o menos, delfine bien clases, usa bien el super constructor, define comportamiento heredado, pero no define el método defensa en la clase base de Pokemon
  - Bien definida la estructura de clases
  - Bien por el uso del super constructor incluyendo tipo y debilidad en la instanciacion
  - Bien el comportamiento heredado de ataque para Fuego y Hierba
  - Mal por no definir el método de defensa en la clase base para luego ser heredado o implementado en cada clase
- Ocultamiento: Bien con detalles
  - Bien definidas las visibilidades de los atributos y métodos

Comentado [18]: Bien el uso del constructor

Comentado [19]: La dinamica de ataque esta bien, pero arrastra el mismo error de la clase Pokemon con el metodo get_tipo

Comentado [20]: Por un lado oponente.get_ataque esta mal invocado como paso con get_tipo en otros metodos.
Pero ademas la logica de este metodo no esta bien planteada. El enunciado menciona que el metodo defender recibe el valor de daño o ataque recibido, no una instancia de Pokemon. El parametro de entrada deberia ser danio_recibido en vez de oponente y las referencias a oponente.get_ataque en realidad deberian ser danio_recibido. De esa manera este metodo defender, se invoca desde el metodo ataque a traves de la referencia oponente de ese metodo, con el ataque calculado en ese metodo

Comentado [21]: vida_restante no esta declarado, quizas se quiso usar recibir_danio, pero ese metodo tiene el parametro oponente que no seria correcto

Comentado [22]: Bien

Comentado [23]: Mismos errores que en casos anteriores
- oponente.get_ataque no esta bien invocado
- Oponente no es el parametro a recibir, deberia ser el daño
- Ese daño es el que debe compararse con la defensa

Comentado [24]: Bien

Comentado [25]: Bien el intento de logica pero sufre los mismos errores descritos antes

Williams, Walter

- La unica observacion en este sentido es el método recibir_danio que más allá del error en la logica implementada deberia ser protected para el uso que se le da

- Polimorfismo: Mal, hay comportamiento polimorfico para el ataque, pero el método defensa no se encuentra declarado en la clase base
  - Bien por la definición de logica compartida para el ataque de Fuego y Hierba
  - Por otro lado, el método defender deberia haberse declarado al menos cómo abstracto en la clase base para poder referenciarlo sin importar de que tipo de Pokemon se tratase
- Abstraccion: Mal, parece entender la logica del problema general pero comete errores, no comprende la dinamica de ataque-defensa, no logra implementar la defensa, referencia métodos que no existen, definición de métodos set que no actuan cómo tales
  - Las mayores falencias están en el proceso de abstraccion
  - No comprende la dinamica de ataque y defensa de los personajes
  - Define un método recibir_danio, que en realidad está suplantando de forma equivocada el método defender, que por esto termina quedando sin uso.
  - Referencia un método vida_restante que no está declarado. Por los errores en las condiciones, este método no se intenta ejecutar nunca, asi que no arroja errores de ejecucion.
  - Las logicas de ataque y defensa parecen comprendidas pero por los errores mencionados, no funcionan o no se invocan, en el caso de la defensa.
  - La simulacion y la logica de atrapado están bien planteadas
  - Existen métodos set_vida set_salvajismo que no actuan cómo sets, sino que actuan cómo otro tipo de método
  - Usa un método vida_restante que no está declarado
- Otros
  - Se tiene en consideracion la mal explicacion de la dinamica entre nivel del entrenador y salvajismo
  - Es un error grave el error en la invocacion de los métodos get_ataque y get_tipo, se los trata cómo atributos cuando son métodos, asi los valores se comparan con el objeto que representa al método al momento de la ejecucion y no con el valor que deberia devolver, por lo cual las condiciones siempre devolveran un resultado erroneo
  - No definir getters o setters que no se usen

DESAPROBADO