```
from abc import ABC, abstractmethod
import random


class Pokemon(ABC):
    def __init__(self, nombre: str):
        self._nombre = nombre
        self._vida: int = 100
        self._ataque = self.genero_atributo()
        self._defensa = self.genero_atributo()
        self._velocidad = self.genero_atributo()
        self._debilidad = self.genero_atributo()
        self._salvajismo = self.genero_atributo()

    @property
    def vida(self):
        return self._vida

    @property
    def ataque(self):
        return self._ataque

    @property
    def defensa(self):
        return self._defensa

    @property
    def velocidad(self):
        return self._velocidad

    @property
    def debilidad(self):
        return self._debilidad

    @property
    def salvajismo(self):
        return self._salvajismo

    @vida.setter
    def vida(self, valor):
        self._vida = valor

    @ataque.setter
    def ataque(self, valor):
```

Comentado [1]: El programa ejecuta aunque tira un par de None

Comentado [2]: Bien la clase abstracta

```
        self._ataque = valor

    @defensa.setter
    def defensa(self, valor):
        self._defensa = valor

    @velocidad.setter
    def velocidad(self, valor):
        self._velocidad = valor

    @debilidad.setter
    def debilidad(self, valor):
        self._debilidad = valor

    @salvajismo.setter
    def salvajismo(self, valor):
        self._salvajismo = valor

    def genero_atributo(self):
        return random.randint(0, 100)

    def imprimir_pokemon(self):
        print(
            f"Nombre: {self._nombre} \n Ataque: {self._ataque} \n Defensa: {self._defensa} \n
Velocidad: {self._velocidad} \n Salvajismo: {self._salvajismo} \n"
        )

    @abstractmethod
    def atacar(self, objetivo):
        pass

    @abstractmethod
    def defender(self, atacante):
        pass



from Pokemon import Pokemon
import random


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

Comentado [3]: Si va a ser un metodo que solo va utilizar esta clase o sus hijas la visibilidad de la clase esta mal como publica.

Comentado [4]: Este metodo no necesariamente era abstracto, porque dos de sus hijos lo hacian igual asi que podria tener un cuerpo aca que compartir

Comentado [5]: Y el atributo debilidad?

```
    def atacar(self, objetivo):
        probabilidad_critico = random.randint(0.0, 1.0)
        if (objetivo.__class__.__name__ == "Agua") and (probabilidad_critico >= 0.7):
            objetivo.vida -= self.ataque * 0.5
            print(f"{self.nombre} ataco a {objetivo.nombre} con un daño critico de {self.ataque *
0.5}")
            print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")
        else:
            if objetivo.defensa > self.ataque:
                print(f"{objetivo.nombre} se defendio del ataque de {self.ataque} puntos de daño")
            else:
                objetivo.vida -= self.ataque
                print(f"{self.nombre} ataco a {objetivo.nombre} con un daño de {self.ataque}")
                print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")

        if objetivo.vida <= 0:
            print(f"{objetivo.nombre} murio")

    def defender(self, atacante):
        probabilidad_evadir_ataque = random.randint(0.0,1.0)
        if (self.velocidad > 50) and (probabilidad_evadir_ataque >= 0.5):
            print(f"{self.nombre} evadio el ataque.")
        elif self.defensa > atacante.ataque:
            print(f"{self.nombre} se defendio del ataque de {atacante.ataque} puntos de daño")
        else:
            self.vida -= atacante.ataque
            print(f"{self.nombre} recibio el ataque de {atacante.nombre} con {atacante.ataque}
daño")
            print(f"Vida restante: {self.vida}")

        if self.vida <= 0:
            print(f"{self.nombre} murio.")




class Fuego(Pokemon):
    def __init__(self, nombre):
        super().__init__(nombre)

    def atacar(self, objetivo):
        probabilidad_critico = random.randint(0.0, 1.0)
        if (objetivo.__class__.__name__ == "Hierba" and probabilidad_critico >= 0.7):
```

Comentado [6]: Si la probilidad de un critico se da el otro pokemon no se defiende? en el else preguntas primero si la defensa es mayor al atque y esto deberia ser lo primero que es como general. Aunque segun el enunciado donde se veia esta posibilidad es a la hora de defender. ya que si el ataque es menor a los puntos de defensa este ataque no se realiza

Despues esta mal preguntar de que clase es el objetivo cuando hay un atributo debilidad que te indica si el pokemon al que voy a atacar es debil al tipo de pokemon del que pertenece el ataque.

seria algo como si objetivo.debilidad == "Hierba"

Comentado [7]: Idem al error del pokemon tipo hierba

```
            objetivo.vida -= self.ataque * 0.5
            print(f"{self.nombre} ataco a {objetivo.nombre} con un daño critico de {self.ataque *
0.5}")
            print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")
        else:
            if objetivo.defensa > self.ataque:
                print(f"{objetivo.nombre} se defendio del ataque de {self.ataque} puntos de daño")
            else:
                objetivo.vida -= self.ataque
                print(f"{self.nombre} ataco a {objetivo.nombre} con un daño de {self.ataque}")
                print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")

        if objetivo.vida <= 0:
            print(f"{objetivo.nombre} murio")


    def defender(self, atacante):
        if self.defensa > atacante.ataque:
            print(f"{self.nombre} se defendio del ataque de {atacante.ataque} puntos de daño")
        else:
            self.vida -= atacante.ataque
            print(f"{self.nombre} recibio el ataque de {atacante.nombre} con {atacante.ataque}
daño")
            print(f"Vida restante: {self.vida}")

        if self.vida <= 0:
            print(f"{self.nombre} murio")

from Pokemon import Pokemon
import random


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

    def atacar(self, objetivo):
        if (objetivo.__class__.__name__ == "Fuego"):
            objetivo.vida -= self.ataque * 0.7
            print(f"{self.nombre} ataco a {objetivo.nombre} con un daño critico de {self.ataque *
0.5}")
            print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")
        elif objetivo.defensa > self.ataque:
            print(f"{objetivo.nombre} se defendio del ataque de {self.ataque} puntos daño")
```

Comentado [8]: idem a lo del pokemon de tipo hierba

Comentado [9]: idem a lo anterior

Comentado [10]: mismo error que en las anteriores clases con respecto a preguntar de clase es.

```
        else:
            objetivo.vida -= self.ataque
            print(f"{self.nombre} ataco a {objetivo.nombre} con un daño de {self.ataque}")
            print(f"{objetivo.nomnbre} le queda {objetivo.vida} de vida restante")

        if objetivo.vida <= 0:
            print(f"{objetivo.nombre} murio")


    def defender(self, atacante):
        reducir_dano = random.randint(0.0,1.0)
        if reducir_dano >= 3.0:
            if self.defensa > atacante.ataque:
                print(f"{self.nombre} se defendio del ataque de {atacante.ataque} puntos de daño")
            else:
                self.vida -= atacante.ataque - 1.05
                print(f"{self.nombre} redujo el ataque de {atacante.nombre} a {atacante.ataque - 1.05}
daño")
                print(f"Vida restante: {self.vida}")
        else:
            if self.defensa > atacante.ataque:
                print(f"{self.nombre} se defendio del ataque de {atacante.ataque} puntos de daño")
            else:
                self.vida -= atacante.ataque
                print(f"{self.nombre} recibio el ataque de {atacante.nombre} con {atacante.ataque}
daño")
                print(f"Vida restante: {self.vida}")

        if self.vida <= 0:
            print(f"{self.nombre} murio")
import random


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

    @property
    def nombre(self):
        return self.__nombre
```

Comentado [11]: no verifica si la defensa es mayo o menor al ataque

```
    @property
    def nivel_entrenador(self):
        return self.__nivel_entrenador

    @property
    def pokemon_principal(self):
        return self.__pokemon_principal

    @property
    def pokedex(self):
        return self.__pokedex

    @nivel_entrenador.setter
    def nivel_entrenador(self, nivel):
        self.__nivel_entrenador = nivel

    @pokemon_principal.setter
    def pokemon_principal(self, pokemon):
        self.__pokemon_principal = pokemon

    @pokedex.setter
    def pokedex(self, pokemon):
        self.__pokedex.append(pokemon)

    def atrapar_pokemon(self, pokemon):
        lista = [True, False]
        captura = random.choice(lista)
        if self.nivel_entrenador > pokemon.salvajismo:
            for i in range(random.randint(1,3)):
                pokemon.vida -= pokemon.vida * 0.1
                if pokemon.vida > 0 and captura:
                    print(
                        f"{pokemon._nombre} tipo {pokemon.__class__.__name__} capturado!\n"
                    )
                    self.pokedex = pokemon
                elif pokemon.vida <= 0:
                    print(f"{pokemon.vida} murio")

            print(f"{pokemon._nombre} se escapo y no logro ser capturado\n")

    def imprimir_pokedex(self):
        contador = 1
        for pokemon in self.pokedex:
            print(f"Pokemon {contador}: {pokemon.imprimir_pokemon()}")
```

Comentado [12]: Metodos incesario si no se van a usar

Comentado [13]: no era necesario un random para saber si lo atrapa o no. la condicion para atraparlo es que el nivel del entrenador sea mayor al nivel de salvajismo del pokemon y si esto ocurria lo podia atrapar realizandole un ataque y viendo que la vida no sea 0 o menor o directamente atrapandolo sin realizar el ataque.

```
            contador += 1

    def imprimir_entrenador(self):
        print("================================================================")
        print(f"Entrenador: {self.nombre} \nNivel: {self.nivel_entrenador} \n")
        print("==================POKEDEX==================")
        print(f"Pokedex: {self.imprimir_pokedex()} \n")



from Hierba import Hierba
from Fuego import Fuego
from Agua import Agua
from Entrenador import Entrenador
import random


nombres = [
    "Mango",
    "Palta",
    "Azul",
    "Rojito",
    "Enano",
    "Remolacha",
    "Aceituna",
    "Burro",
    "Papiro",
    "Muzzarella",
    "Jamon",
    "Lentejuela",
]
tipos_pokemones = [Hierba, Fuego, Agua]

entrenador = Entrenador(
    "Daniel", random.randint(1, 100), random.choice(tipos_pokemones)
)

# Se van a repetir nombres y tipos en la pokedex y las capturas porque hay pocos nombres y
tipos

for i in range(1, 10):
    pokemon = random.choice(tipos_pokemones)(random.choice(nombres))

    entrenador.atrapar_pokemon(pokemon)
```

Comentado [14]: Mal la visibiliad es un metodo que no va salir de esta clase por lo tanto debe ser privado

Comentado [15]: Bien

Comentado [16]: No corresponde al enunciado

entrenador.imprimir_entrenador()

Devolución:

HERENCIA:

En el tema herencia tiene las clases correspondientes pero en el único lugar que usa una posible reutilización de código es en el constructor pero no se da cuenta que tiene código repetido en los métodos atacar, dejando en dudas si entiende la utilización del super y el polimorfismo

OCULTAMIENTO:

Las visibilidad de los atributos están bien pero se olvida de que los métodos también tiene visibilidades, crea métodos que no van a ser utilizados fuera de la clase de manera pública dejando en duda si entiende el concepto de ocultamiento de la información.

POLIMORFISMO:

Crea la clase padre de manera abstracta y el método atacar de manera abstracta cuando en el enunciado indica que hay dos tipos de pokémon que atacan de la misma manera pudiendo hacer este desarrollo en el padre y utilizarlo en dos de sus tres hijos

ABSTRACIÓN:
No realiza una correcta abstracción del problema planteado en el enunciado. Como consecuencia arrastra errores en los otros temas.

DESAPROBADO