```
from abc import ABC, abstractmethod
from random import *

class Pokemon(ABC):
    def __init__(self,nombre):
        self.__nombre = nombre
        self._tipo = None
        self.__vida = 100
        self.__ataque = self.generar_atributos()
        self.__defensa = self.generar_atributos()
        self.__velocidad = self.generar_atributos()
        self._debilidad = None
        self.__salvajismo = self.generar_atributos()

    # @property
    # def getDebilidad(self):
    #     return self.__debilidad

    # @getDebilidad.setter
    # def setDebilidad(self,debilidad):
    #     self.__debilidad = debilidad

    @property
    def getSalvajismo(self):
        return self.__salvajismo

    @getSalvajismo.setter
    def setSalvajismo(self,valor):
        self.__salvajismo = valor

    @property
    def getVida(self):
        return self.__vida

    def generar_atributos(self):
        return randint(0,100)

    @abstractmethod
    def imprimir(self):
        print('Nombre:',self.__nombre)
        print('Defensa:',self.__ataque)
        print('Ataque:',self.__defensa)
        print('Ataque:',self.__velocidad)
        print('Ataque:',self.__salvajismo)
```

**Comentado [1]:** Los atributos privados no son heredados por los hijos

**Comentado [2]:** Si es un metodo que solo se usa en esta clase y sus hijas esta mal la visibiliadad

```
    @abstractmethod
    def ataque(self,pokemon):
        probabilidad = randint(1,10)
        if self._tipo == pokemon._debilidad and probabilidad < 8: #probabilidad del %70
            return self.__ataque+self.__ataque*0.5 #incrementa el ataque en %50
        else:
            return self.__ataque

    @abstractmethod
    def defensa(self,ataque):
        if ataque > self.__defensa:
            self.__vida -= ataque - self.__defensa
```

**Comentado [3]:** Si son metodos abstractos no deberian tener cuerpo

```
from Pokemon import Pokemon
from random import *

class Hierba(Pokemon):
    def __init__(self, nombre):
        super().__init__(nombre)
        self._tipo = 'Hierba'
        self._debilidad = 'Fuego'

    def imprimir(self):
        return super().imprimir()

    def ataque(self, pokemon):
        return super().ataque(pokemon)

    def defensa(self,ataque):
        probabilidad = randint(1,2) #probabilidad del %50
        if not(self.__velocidad > 50 and probabilidad == 1): #uso un not para pensarlo de forma
inversa
            super().defensa(ataque)
        #si no entra en la condición significa que esquivó el ataque

from Pokemon import Pokemon

class Fuego(Pokemon):
    def __init__(self, nombre):
        super().__init__(nombre)
        self._tipo = 'Fuego'
        self._debilidad = 'Agua'
```

**Comentado [4]:** mal, si no se extiende el cuerpo o se redefine no tiene sentido volver a declarar un metodo que va hacer lo mismo que hace el del padre

**Comentado [5]:** idem

**Comentado [6]:** Mismos errores

```
    def imprimir(self):
        return super().imprimir()

    def ataque(self, pokemon):
        return super().ataque(pokemon)

    def defensa(self, ataque):
        return super().defensa(ataque)

from Pokemon import Pokemon
from random import *

class Agua(Pokemon):
    def __init__(self, nombre):
        super().__init__(nombre)
        self._tipo = 'Agua'
        self._debilidad = 'Hierba'

    def imprimir(self):
        return super().imprimir()

    def ataque(self, pokemon):
        if self._tipo == pokemon._debilidad:
            return self.__ataque+self.__ataque*0.7 #incrementa el ataque en %70
        else:
            return self.__ataque

    def defensa(self, ataque):
        probabilidad = randint(1,10)
        if probabilidad < 4: #probabilidad del %30
            super().defensa(ataque*0.5) #reduce el daño %50
        else:
            super().defensa(ataque)

def __init__(self,nombre,pokemonMain):
        self.__nombre = nombre
        self.__nivel = randint(1,100)
        self.__pokemonMain = pokemonMain
        self.__pokedex = []

    def atraparPokemon(self,pokemonDisputado):
        salvajismoActual = pokemonDisputado.getSalvajismo()
        capturado = False #variable para evitar capturarlo + de una vez en el for
        if self.__nivel > salvajismoActual:
```

**Comentado [7]:** Mismos errores

```
            self.__pokedex.append(pokemonDisputado) #si tiene el nivel requerido lo captura
directamente
            capturado = True
            print("Pokemón atrapado con éxito!")
        else:
            for turno in range(1,3): #el Pokemón del Entrenador realiza ataques y el contrario
defiende durante 3 turnos
                pokemonDisputado.defensa(self.__pokemonMain.ataque(pokemonDisputado))
                salvajismoActual -= salvajismoActual*0.1 #en cada ataque disminuyo el Salvajismo
en %10
                pokemonDisputado.setSalvajismo(salvajismoActual) #actualizo el valor en el
Pokemón
                if pokemonDisputado.getVida() > 0 and self.__nivel > salvajismoActual and not
capturado: #mientras esté vivo, si el salvajismo disminuye lo suficiente lo captura
                    self.__pokedex.append(pokemonDisputado)
                    capturado = True
                    print("Pokemón atrapado con éxito!")
        if not capturado:
            print("Falló la captura :(")



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

nombres =
['pokemon1','pokemon2','pokemon3','pokemon4','pokemon5','pokemon6','pokemon7','pokemon8
','pokemon9','pokemon10','pokemon11','pokemon12']
pokemones = [] #lista de TODOS los pokemones

for i in range(1,10):
    tipo_pokemon = random.randint(1,3)
    match tipo_pokemon:
        case 1: nuevoPokemon = Hierba(random.choice(nombres))
        case 2: nuevoPokemon = Fuego(random.choice(nombres))
        case 3: nuevoPokemon = Agua(random.choice(nombres))
    pokemones.append(nuevoPokemon) #lleno la lista con 10 pokemones

entrenador1 = Entrenador('Ash',random.choice(pokemones)) #el pokemón principal del
entrenador se asigna aleatoriamente
```

entrenador1.atraparPokemon(random.choice(pokemones))

Herencia: Está conformadas las clases perteneciente a la herencia. Pero cómo utiliza el super en los hijos da entender que no entiende el concepto de herencia y sus beneficios.

Encapsulamiento: Tiene algunas visibilidades mal declaradas. Como atributos privados que luego no se van a heredar a los hijos.

Polimorfismo: Los errores cometidos en herencia afectan a este apartado en el mal uso de la reutilización de código. No realizó la impresión de la pokédex que era una de las partes donde se podía ver el uso del polimorfismo.

Abstracción: Los métodos abstractos no deben tener un cuerpo definido. Accede a atributos de un objeto sin usar un método de acceso.

DESAPROBADO