**EXAMEN: Programación Orientada a Objetos en Java**

**📌 Evolución Pokémon, Herencia, Polimorfismo y Equipamiento de Armas**

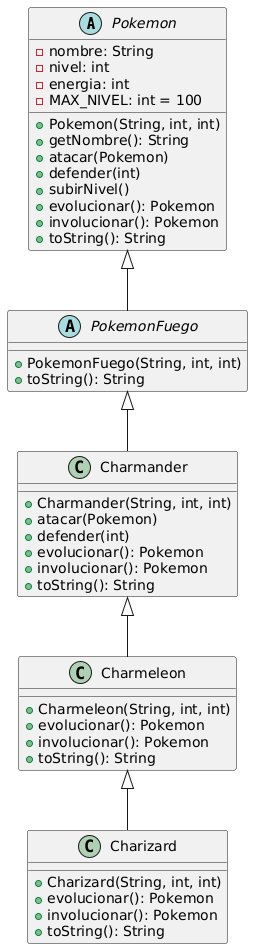
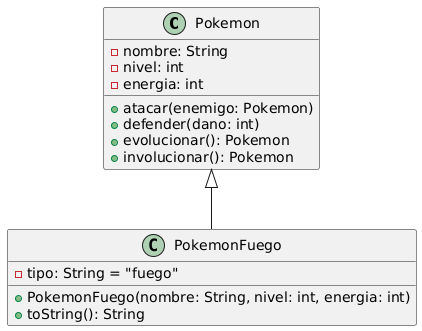
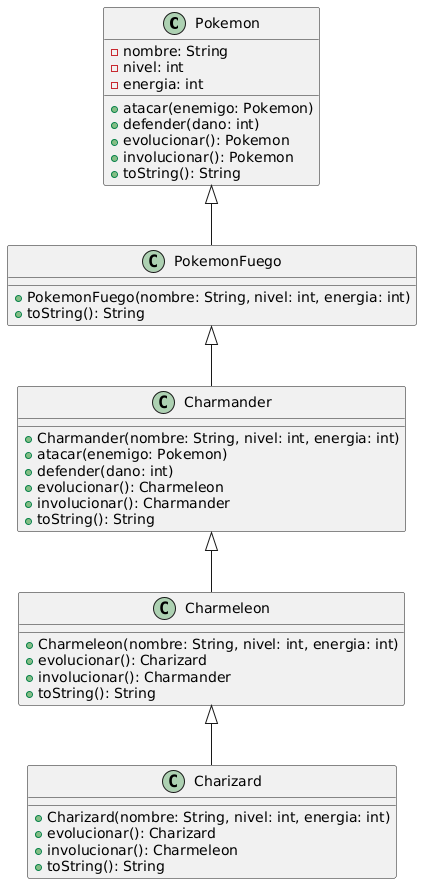
**💡 Instrucciones:**

* Implementa las clases según lo indicado en cada ejercicio.
* Comenta tu código explicando cada método.
* Asegúrate de que tu código funcione correctamente en un entorno Java.

**📍 Primera Parte: Evolución e Involución de Pokémon (2,5 Puntos)**

En esta primera parte, trabajarás con la jerarquía de clases de Pokémon y la mecánica de **evolución** e **involución** utilizando **herencia, polimorfismo y casting dinámico**.

**📌 Ejercicio 1: Creación de la Jerarquía de Clases Pokémon**



📌 **Implementa la siguiente jerarquía de clases según el diagrama UML:   
✅ Clase base Pokemon**

* **Atributos:**
  + **nombre: Nombre del Pokémon.** (Private hasta que se demuestre lo contrario)
  + **nivel: Nivel del Pokémon.** (Private hasta que se demuestre lo contrario)
  + **energia: Energía actual del Pokémon.** (Private hasta que se demuestre lo contrario)
  + **MAX\_NIVEL: Constante final con el valor 100. (public static)**
* **Métodos:**
  + **public String getNombre(): Devuelve el nombre del Pokémon.**
  + **public abstract void atacar(Pokemon enemigo):** Deben ser implementados por cada Pokémon concreto.
  + **public abstract void defender(int dano):** Deben ser implementados por cada Pokémon concreto.
  + **public abstract Pokemon evolucionar():** Deben ser implementados por cada Pokémon concreto.
  + **public abstract Pokemon involucionar():** Deben ser implementados por cada Pokémon concreto.
  + **public final void subirNivel():**
    - Aumenta el nivel en 1 si aún no ha alcanzado MAX\_NIVEL.
    - Imprime un mensaje indicando que el Pokémon ha subido de nivel.
    - Si ya está en MAX\_NIVEL, indica que no puede subir más.
  + **public String toString():**
    - Devuelve un mensaje con el nombre, nivel y energía del Pokémon.

**📌 Crea también las clases intermedias:**

**✅ Clase PokemonFuego**

* Extiende Pokemon y añade un atributo String tipo;
* Modifica toString() para añadir “FUEGO” al imprimir un Pokémon de fuego.

**✅ Clases concretas con tres niveles de evolución:**

* 🔥 Charmander → Charmeleon → Charizard
  + **public abstract void atacar ():**
    - Debe ser implementado por Charmander.
    - Debe imprimir qué ataque usa el Pokémon y a qué enemigo golpea.
    - Debe llamar al método defender(int dano) del enemigo para restarle energía.
  + **public abstract void defender(int dano):**
    - Reduce la energía del Pokémon según el daño recibido.
    - Muestra la energía restante tras recibir el golpe.
  + **public abstract Pokemon evolucionar(): (amarillo revisa el UML)**
    - Devuelve una nueva instancia del Pokémon en su siguiente forma evolutiva.
    - Muestra un mensaje indicando que el Pokémon ha evolucionado.
    - Devuelve la siguiente evolución del Pokémon.
  + **public abstract Pokemon involucionar(): (amarillo revisa el UML)**
    - Usa casting dinámico para convertir el Pokémon a su forma anterior.
    - Si ya es su forma base, imprime un mensaje indicando que no puede involucionar más.
    - Usa casting dinámico para volver a la fase anterior.

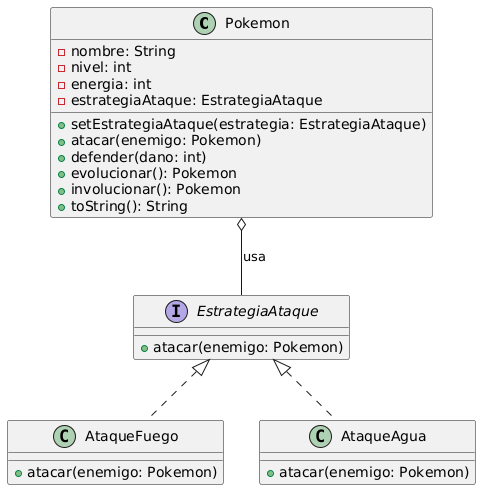
📌 **Ejemplo de uso en main():**

|  |  |
| --- | --- |
| public static void main(String[] args) {  Pokemon charmander = new Charmander("Charmander", 5, 100);  System.out.println(charmander.toString());  System.out.println("\n🔥 ¡ Pokémon! 🔥\n");  // Evolucionar, fíjate en la herencia  Charmeleon charmeleon = (Charmeleon) charmander.evolucionar(); // Charmander -> Charmeleon  System.out.println("Después de evolucionar: " + charmeleon.toString());  Charizard charizard = (Charizard) charmeleon.evolucionar(); // Charmeleon -> Charizard  System.out.println("Después de evolucionar: " + charizard.toString());  // Involucionar polimorfismo  Charmeleon charmeleon1 = charizard.involucionar(); // Charizard -> Charmeleon  System.out.println("Después de involucionar: " + charmeleon1.toString());  } | Charmander{nombre='Charmander', nivel=5, energia=100}  🔥 ¡ Pokémon! 🔥  Charmander está evolucionando a Charmeleon!  Después de evolucionar: Charmeleon { Nombre: Ahora soy un Charmaleon., Nivel: 5, Energía: 100 }  Ahora soy un Charmaleon. está evolucionando a Charizard!  Después de evolucionar: Charizard { Nombre: Ahora soy un Charizard. , Nivel: 5, Energía: 100 (Fuerza de Dragón) }  Ahora voy hacia un Charmeleon. involuciona a Charmeleon...  Después de involucionar: Charizard { Nombre: Superpoderes, pero en memoria sigo siendo Charizard, Nivel: 5, Energía: 100 } |
| */\*¿ Porque sucede que aunque involuciones de Charizard a Charmeleon, cuando llamas a toString siempre llama al toString de Charizard? \*/* | |
| *Respuesta:* | |

**📌 Aplicando OCP (1 Punto) - Uso de EstrategiaAtaque**

📌 **Problema Actual:**  
Cada Pokémon **define su propio ataque dentro de su clase**, lo que hace difícil agregar nuevos ataques sin modificar el código existente. ¿Qué principio es el que incumplimos?

📌 **Solución:**  
✅ **Crear una interfaz EstrategiaAtaque** que permita añadir nuevos ataques sin modificar las clases existentes.



|  |  |
| --- | --- |
| PlantUML Diagram | PlantUML Diagram |

|  |  |
| --- | --- |
| CÓDIGO DE AYUDA | |
| public interface EstrategiaAtaque {  void atacar(Pokemon enemigo);  } | |
| public abstract class Pokemon {  // …… todo lo que ya tengamos  //Pista:  protected EstrategiaAtaque estrategiaAtaque;  public void setEstrategiaAtaque(EstrategiaAtaque estrategia) {  this.estrategiaAtaque = estrategia; } *// SOLID -> Inyección de estrategia*    // … todo lo que ya tengamos  } | |
| public class AtaqueFuego implements EstrategiaAtaque {  @Override  public void atacar(Pokemon enemigo) {  System.out.println("🔥 Ataque de Fuego impacta a " + enemigo.getNombre() + "!");  enemigo.defender(25);  }  }  public class AtaqueAgua implements EstrategiaAtaque {  @Override  public void atacar(Pokemon enemigo) {  System.out.println("💧 Ataque de Agua golpea a " + enemigo.getNombre() + "!");  enemigo.defender(30);  }  } | |
| public static void main(String[] args) {  System.*out*.println("\n⚔️ POKÉMON SOLID ⚔️");  *// Crear Pokémon* Pokemon charmander = new Charmander("CHARMANDER", 50, 500);  Pokemon squirtle = new Squirtle("SQUIRTLE", 20, 400);   *// Asignar estrategias de ataque* charmander.setEstrategiaAtaque(new AtaqueFuego());  squirtle.setEstrategiaAtaque(new AtaqueAgua());   *// Ataque entre Pokémon* charmander.atacar(squirtle);  squirtle.atacar(charmander); } | ⚔️ POKÉMON SOLID ⚔️  CHARMANDER usa su estrategia de ataque contra SQUIRTLE  🔥 Ataque de Fuego impacta a SQUIRTLE!  SQUIRTLE recibe 25 resiste el ataque! Energía restante: 375  SQUIRTLE usa Pistola Agua contra CHARMANDER!  💧 Ataque de Agua golpea a CHARMANDER!  CHARMANDER recibe 30 de daño! Energía restante: 470 |