Trabajo Práctico en Pares: Sistema de Gestión de Criaturas Elementales.

Objetivo General.

El objetivo de este trabajo práctico es que el equipo desarrolle un sistema orientado a objetos que modele un **ecosistema de criaturas elementales**, sus **transformaciones**, **interacciones**, **poderes especiales** y los **maestros que las entrenan**, aplicando programación orientada a objetos avanzada, pruebas automatizadas y trabajo colaborativo profesional.

Modalidad de Trabajo.

- El trabajo se realizará en **equipos de a dos personas**. Se espera una participación **colaborativa y equitativa**, tanto en la implementación del sistema como en la toma de decisiones, diseño, pruebas y documentación.
- Cada equipo deberá:
    - Crear un **repositorio compartido en GitHub** para alojar el proyecto.
    - **Agregar a ambos integrantes como colaboradores** del repositorio.
    - Comunicar al docente **por mensajería** (u otro medio institucional indicado) los **nombres de los integrantes y el enlace al repositorio**, lo cual permitirá realizar el seguimiento del trabajo por parte del equipo docente.
- El seguimiento incluirá la revisión del historial (mínimo 40% cada integrante) de commits, issues, pull requests y demás actividades dentro del repositorio, con foco en la **colaboración real y continua** de ambos integrantes.

Actividades esperadas.

- **Diseño e implementación del sistema**, cumpliendo los requisitos funcionales provistos en el enunciado (capturas, intimidación, evolución de prófugos y reportes).
- Aplicar **principios de diseño orientado a objetos**: herencia, polimorfismo, composición, delegación, cohesión y bajo acoplamiento.
- Utilizar **TDD (Test-Driven Development)** para validar la lógica del sistema con **JUnit-4**.
- Mantener un flujo de trabajo con **commits frecuentes y significativos** por parte de ambos integrantes.
- Usar GitHub para organizar tareas y dividir el trabajo.
- Se espera un proyecto Java tradicional, correctamente estructurado en carpetas, con uso adecuado de *source folders*, organización en paquetes, y aplicación correcta de las convenciones de nomenclatura: **UpperCamelCase** para clases e interfaces, y **lowerCamelCase** para variables y métodos.

ENUNCIADO.

Contexto General.

En el mundo de **Elandria**, existen criaturas elementales que conviven en distintos territorios: bosques, montañas, lagos y llanuras. Cada criatura posee una afinidad elemental (agua, fuego, aire o tierra), un conjunto de habilidades y un nivel de energía que determina su poder.

Los **Maestros Elementales** son quienes entrenan, transforman y controlan a estas criaturas para mantener el equilibrio del reino.

El Consejo de Elandria se encuentra desbordado y necesita un sistema para:

- Gestionar criaturas elementales.
- Registrar entrenamientos y transformaciones.
- Controlar las interacciones entre criaturas.
- Obtener reportes estratégicos.
- Asegurar que los maestros realicen prácticas éticas.

Parte I – Criaturas Elementales y Maestros.

Criaturas.

Cada criatura tiene:

- **Nombre**
- **Nivel de energía** (0–200)
- **Afinidad elemental** (AGUA, FUEGO, AIRE, TIERRA)
- **Comportamiento emocional**: tranquila o inestable

Existen 3 tipos de criaturas:

1. **Criaturas Salvajes**
    - Más difíciles de controlar.
    - Cuando se intenta entrenarlas, pueden aumentar su energía de manera impredecible (deben lanzar una *unchecked exception* cuando superan 200).
2. **Criaturas Domesticadas**
    - Aumentan su energía de forma estable.
    - Nunca se vuelven inestables.
3. **Criaturas Ancestrales**
    - Su energía nunca puede bajar de 100.
    - Son extremadamente poderosas, pero sensibles a entrenamientos extremos.

Maestros Elementales.

Los maestros tienen:

- **Nombre**
- **Nivel de maestría** (1 a 50)
- **Afinidad elemental** principal
- **Colección (HashMap)** de criaturas a su cargo, identificadas por nombre.

Un maestro puede:

- **Entrenar una criatura**: aumenta la energía según reglas del tipo de criatura.
- **Pacificar una criatura inestable**: mecanismo polimórfico.
- **Transformar una criatura** mediante rituales especiales (ver Parte II).

Si el maestro intenta entrenar una criatura sin suficiente maestría, debe lanzarse una **excepción checked**, por ejemplo:

.

Parte II – Transformaciones Elementales.

Una criatura puede adquirir transformaciones (patrón decorador) que alteran sus características.

Transformaciones disponibles:

1. **Bendición del Río.**
   - Duplica la energía, pero nunca supera 180.
2. **Llama Interna.**
   - Si la afinidad es fuego → +30 energía.
   - Si no → la criatura se vuelve inestable.
3. **Vínculo Terrestre.**
   - Garantiza que nunca quede con energía menor a 50.
4. **Ascenso del Viento.**
   - Convierte temporalmente al tipo **AIRE**, sin perder afinidades previas.

Cada transformación suma una capa (como en el TP original) sin perder las anteriores.

La aplicación de transformaciones debe ser polimórfica y estar cubierta por tests.

Parte III – Interacciones entre Criaturas.

Cuando dos criaturas interactúan:

- Si comparten afinidad → ambas ganan 10 de energía.
- Si son opuestas (agua–fuego / aire–tierra) → se desestabilizan (se vuelven inestables).
- Si una es ancestral → siempre domina la interacción:
  - La ancestral gana 20 energía.
  - La otra pierde 15 energía (mínimo 0).

Las reglas deben implementarse con una **jerarquía de clases bien diseñada** y respetando **SRP y OCP**.

Parte IV – Reportes para el Consejo.

Se requiere implementar un módulo que permita:

1. **Listar todas las criaturas registradas por todos los maestros.**
2. **Obtener la criatura con mayor energía total.**
3. **Determinar qué maestro tiene más criaturas transformadas.**
4. **Obtener un mapa (HashMap) con la cantidad de criaturas por afinidad elemental.**

Los reportes deben estar cubiertos por tests TDD desde el inicio.

Requisitos Mínimos de Aprobación.

Criterio	**Descripción**
Diseño completo	Todas las partes funcionales implementadas.
Aplicación de OOP	Uso correcto de herencia, polimorfismo, interfaces, clases abstractas.
Trabajo en equipo	Commits equitativos (mín. 40% por integrante).
GitHub	Uso de ramas, issues, PRs. Historias de cambios legibles.
TDD + JUnit4	Tests suficientes, mínimo 70% cobertura.
Excepciones	Al menos una checked y una unchecked correctamente justificadas.
Colecciones avanzadas	Uso central de HashMap y estructuras auxiliares.
Código limpio	SOLID, cohesión, bajo acoplamiento. Nombres claros.

Plazo de entrega.

- Fecha límite de entrega: **sábado 22 de noviembre, a las 23:59 hs.**