Trabajo Práctico N°2 – Paradigma de Programación Orientada a Objetos

Justificación del Diagrama de Clases – Entrega Intermedia: 27/05/25

1. Introducción

Este documento justifica el diseño del diagrama de clases propuesto para el sistema de crafteo de objetos, en el marco del Trabajo Práctico N°2. El objetivo es aplicar los conceptos del paradigma de la Programación Orientada a Objetos (POO) para modelar un sistema inspirado en la mecánica de crafteo presente en videojuegos.

2. Objetivo del Modelo

El modelo busca representar la lógica de crafteo mediante clases que reflejen objetos simples e intermedios, recetas, inventarios y operaciones de fabricación. Se priorizó el diseño modular, con bajo acoplamiento y alta cohesión, permitiendo la incorporación de nuevas reglas o elementos sin modificar el código existente, cumpliendo el principio de abierto/cerrado.

3. Descripción General

El sistema modela los siguientes aspectos:

- **Elementos**: Representan ítems del juego, pueden ser simples, complejos o catalizadores.
- **Recetas**: Definen los ingredientes necesarios para fabricar un nuevo elemento.
- **Inventario**: Almacena los elementos que posee el jugador.
- **Crafteador**: Gestiona la lógica de crafteo, verificación, historial y tiempos.
- **RegistroCrafteo**: Registra cada operación realizada, sus ingredientes y tiempo.
- **Archivos JSON**: Se usan para cargar recetas e inventarios externos.

4. Justificación de Clases y Relaciones

- **Elemento (abstracto)**: Clase base con atributos comunes (nombre, tiempo) y una estructura de subelementos. Permite la extensión por herencia.
- **ElementoSimple / ElementoComplejo**: Heredan de Elemento. El simple no requiere subelementos, el complejo sí. Ambos implementan tiempo() y subelementos().
- **Catalizador**: Subclase especializada que optimiza el crafteo, con un tipo asociado y lógica de aplicación separada.
- **IngredienteDeReceta**: Asocia un Elemento con una cantidad. Utilizado dentro de Receta.
- **Receta**: Contiene una lista de ingredientes. Tiene métodos para carga desde JSON y validación de recetas.

- **Inventario**: Gestiona los elementos del jugador. Permite cargar y guardar su estado, cumpliendo con el bonus del inventario final.
- **Crafteador**: Núcleo del sistema. Puede verificar crafteos posibles, mostrar árboles de crafteo, calcular tiempos y registrar el historial.
- **RegistroCrafteo**: Representa un evento de crafteo, registrando objeto creado, ingredientes usados, fecha y duración.
- **Relaciones**:
    - **Composición** entre Receta e IngredienteDeReceta: un ingrediente no existe fuera del contexto de una receta. A eliminar una receta, también se eliminan sus ingredientes.
    - Asociación entre Receta y Elemento: cada receta produce un único elemento como resultado. Esta asociación refleja la relación directa entre una receta y el objeto que genera.
    - Agregación entre Crafteador y RegistroCrafteo (historial): el Crafteador mantiene un historial de crafteos realizados, pero los registros pueden existir independientemente. Por eso se usa agregación.
    - Uso del patrón polimórfico entre Elemento y sus subclases: elemento es una clase abstracta, y ElementoSimple, ElementoComplejo y Catalizador implementan sus métodos de forma distinta, permitiendo tratarlos de forma uniforme en el sistema.

5. Consideraciones de Diseño

- Se aplica **herencia** para reutilizar comportamientos entre tipos de elementos.
- Se respeta la **responsabilidad única**: cada clase tiene un propósito claro y limitado.
- La carga desde archivos externos permite separar datos de lógica.
- El diseño permite extender el sistema con nuevas recetas, catalizadores o mesas sin alterar las clases existentes (principio O/C).
- Las colecciones (List, Map) se usan para gestionar ingredientes, subelementos e inventario.

6. Conclusión

El diagrama propuesto cubre los requerimientos de la consigna, modelando de forma clara la lógica del crafteo. Se garantiza modularidad, extensibilidad y una base sólida para implementar todas las funcionalidades requeridas. Para la entrega final, se espera refinar este modelo e incorporar más funcionalidades opcionales como catalizadores, mesas de trabajo e integración con Prolog.