Clase 14: Ejercicio - Tamagotchi

Propósito: practicar los patrones *State* y *Observer* modelando un Tamagotchi con estados de ánimo/ necesidad y un sistema de notificaciones de eventos.

1. Contexto

Construirás un **Tamagotchi** con tres estados de ánimo/necesidad y un ciclo simple de vida. El **patrón *State*** representará los estados (p. ej., `Hungry`, `Sleepy`, `Happy`) y definirá cómo responde a acciones del usuario (`feed()`, `sleep()`, `play()`). El **patrón *Observer*** permitirá que observadores externos se suscriban a eventos del Tamagotchi (cambios de estado, acciones válidas/ inválidas, etc.).

2. Objetivos de aprendizaje

- Aplicar *State* para eliminar condicionales y encapsular comportamientos por estado.
- Aplicar *Observer* para desacoplar la emisión de eventos de sus receptores.
- Diseñar interfaces claras para *contexto*, *estado*, *sujeto* y *observador*.
- Escribir reglas de transición y validación como **comportamiento de los estados**.
- Practicar pruebas unitarias de transiciones y notificaciones.

3. Requisitos

3.1. Parte 1: Modelar estados con el patrón State

En esta primera parte implementarás la **lógica interna del Tamagotchi** usando el patrón de diseño **State** para representar su comportamiento dependiente del estado.

Tu objetivo es modelar los estados `Happy`, `Hungry` y `Sleepy` y sus transiciones tal como indica la figura, sin usar condicionales en el contexto.

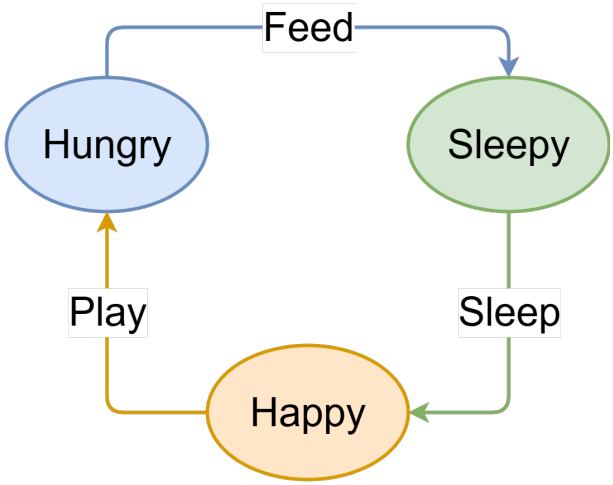

Figura 1: Diagrama de estados del Tamagotchi. Cualquier acción no listada para un estado debe **fallar**.

El Tamagotchi **siempre inicia** en `Happy`.

3.2. Parte 2: Notificaciones con *Observer*

En esta parte integrarás un **controller** y dos **observers** para escuchar eventos del Tamagotchi. El objetivo es **mostrar la estructura del patrón *Observer***, manteniendo la lógica simple y sin inspección de tipos.

3.2.1. Estructura objetivo

- `Tamagotchi` actúa como **emisor** de dos flujos de eventos:
  - `ValidTransition` (transición válida).
  - `InvalidTransition` (transición inválida).
- `Controller` posee **dos observadores**:
  - `Observer[ValidTransition]` ⟶ maneja transiciones válidas.
  - `Observer[InvalidTransition]` ⟶ maneja transiciones inválidas.
- Cada acción (`feed`, `play`, `sleep`) se ejecuta en el estado actual. Si la acción es válida, se notifica `ValidTransition`; si el estado la rechaza lanzando `InvalidTransitionException`, se notifica `InvalidTransition`.