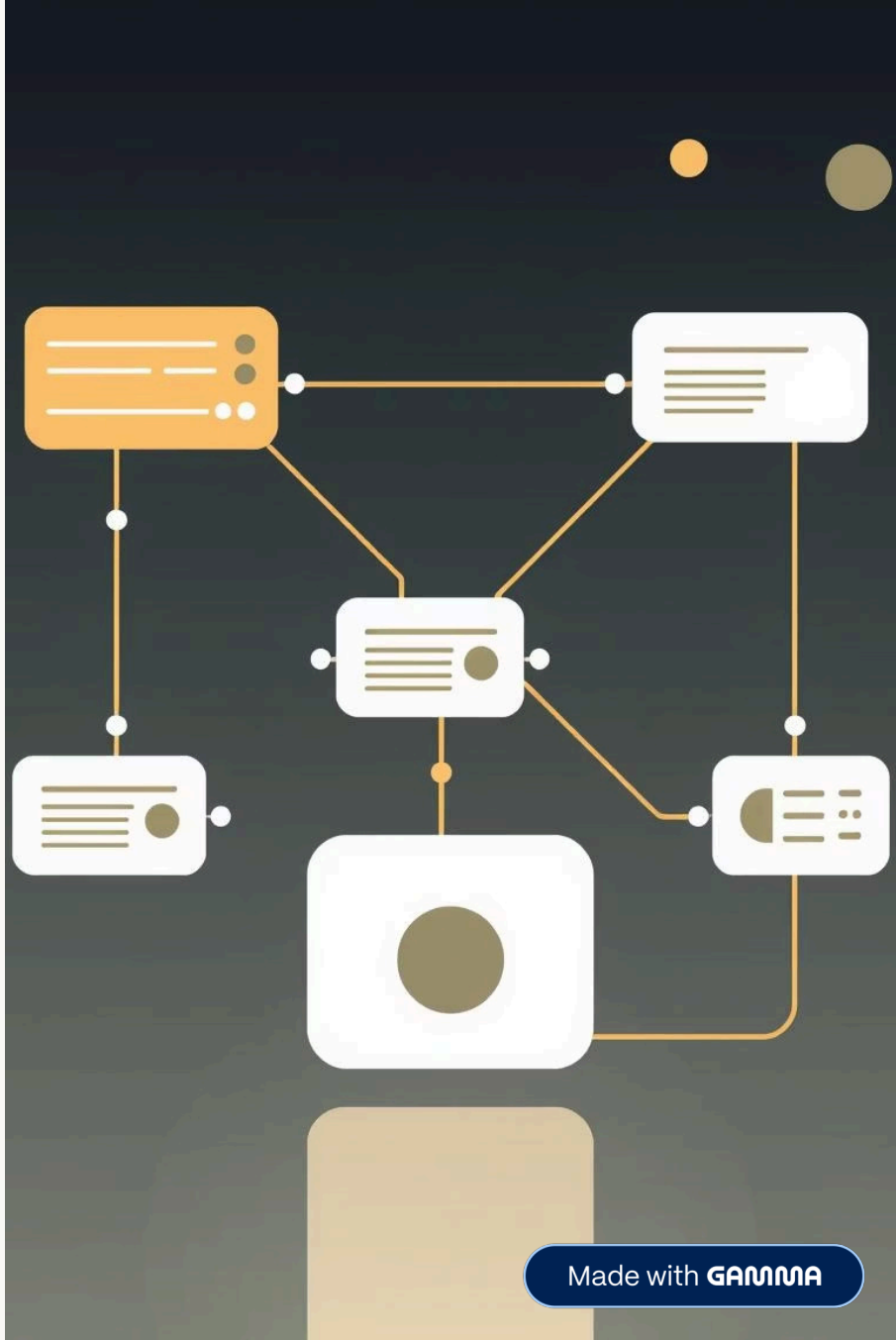
Patrón Chain of Responsibility: Gestión Flexible de Solicitudes

El patrón Chain of Responsibility es un patrón de diseño comportamental que permite pasar una solicitud a lo largo de una cadena de objetos hasta que uno de ellos la maneje. Este patrón desacopla el emisor de la solicitud de los receptores concretos, facilitando la extensión y el mantenimiento del código. Es ideal para sistemas con múltiples etapas de procesamiento o filtros, donde no se sabe de antemano quién debe manejar la petición.



¿Cómo funciona? Estructura y Flujo

Cada objeto, conocido como "handler", en la cadena decide si procesa la solicitud o la pasa al siguiente handler en la secuencia. Esta cadena se puede construir y modificar dinámicamente en tiempo de ejecución, lo que otorga una gran adaptabilidad.

Ventajas clave:

- Flexibilidad: Permite añadir o reordenar handlers sin modificar el código del cliente que emite la solicitud.
- Bajo acoplamiento: Promueve el principio de responsabilidad única, reduciendo las dependencias entre el emisor y los receptores.
- Reutilización y Extensión: Facilita la reutilización de componentes y la extensión del sistema con nuevas lógicas de procesamiento.

Ejemplo: Animales que deciden qué comer

Imaginemos un sistema donde diferentes tipos de animales (handlers) deciden si pueden comer un determinado alimento (solicitud). Cada animal verifica si el alimento es apto para su dieta, y si no lo es, lo pasa al siguiente.

```
from abc import ABC, abstractmethod

class AnimalHandler(ABC):
    def __init__(self, next_handler=None):
        self.next_handler = next_handler

    @abstractmethod
    def handle_food(self, food):
        pass

class CarnivoreHandler(AnimalHandler):
    def handle_food(self, food):
        if food == "carne":
            return f"El carnívoro está comiendo {food}."
        elif self.next_handler:
            return self.next_handler.handle_food(food)
        return f"Nadie pudo comer {food}."

class HerbivoreHandler(AnimalHandler):
    def handle_food(self, food):
        if food == "hierba":
            return f"El herbívoro está comiendo {food}."
        elif self.next_handler:
            return self.next_handler.handle_food(food)
        return f"Nadie pudo comer {food}."

class OmnivoreHandler(AnimalHandler):
    def handle_food(self, food):
        if food in ["carne", "hierba", "fruta"]:
            return f"El omnívoro está comiendo {food}."
        elif self.next_handler:
            return self.next_handler.handle_food(food)
        return f"Nadie pudo comer {food}."

# Construir la cadena
omnivore = OmnivoreHandler()
carnivore = CarnivoreHandler(next_handler=omnivore)
herbivore = HerbivoreHandler(next_handler=carnivore)

# Probar con diferentes alimentos
print(herbivore.handle_food("hierba"))
print(herbivore.handle_food("carne"))
print(herbivore.handle_food("fruta"))
print(herbivore.handle_food("piedra"))
```

En este ejemplo, la solicitud del alimento se pasa desde el herbívoro. Si este no puede procesarlo, lo pasa al carnívoro, y si este tampoco, llega al omnívoro. Si ninguno puede, se indica que nadie lo comió.