SESIÓN 12: Patrones de Comportamiento y Ejemplos en Apps Modernas
Cómo los objetos cooperan para construir software inteligente

1. ¿QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Los patrones de comportamiento definen **cómo colaboran e interactúan los objetos en un sistema**.
Mientras que:

- Los patrones creacionales responden a *cómo crear objetos*
- Los patrones estructurales responden a *cómo organizar objetos*

Los patrones de comportamiento responden a:
¿Cómo se comunican los objetos entre sí para trabajar como un equipo flexible, escalable y desacoplado?

2. ¿POR QUÉ EXISTEN ESTOS PATRONES?

Problemas comunes que resuelven:

- **Evitar que los objetos dependan demasiado unos de otros**
  - Si cambio uno, no debería romperse media aplicación.
- **Manejar cambios en tiempo real**
  - Notificaciones, eventos, actualizaciones dinámicas.
- **Reemplazar algoritmos sin reescribir el programa**
  - Cambiar la forma de calcular rutas, recomendaciones o filtros.
- **Encapsular acciones para ejecutarlas, deshacerlas o programarlas**
  - Perfecto para comandos, automatizaciones y sistemas de edición.
- **Adaptar el comportamiento según el estado interno**
  - Ejemplo: un pedido no se puede cancelar cuando ya está enviado.

3. LOS 4 PATRONES FUNDAMENTALES

1.OBSERVER (Observador)

"Un objeto emite un evento y muchos reaccionan sin depender directamente de él."

Idea principal

Un **sujeto** mantiene una lista de **observadores**.
Cuando ocurre un cambio, el sujeto los notifica automáticamente.

Ejemplos reales

App	Ejemplo	Explicación
Instagram	Notificaciones de likes y comentarios	Cada seguidor es un "observador".
YouTube	Aviso "nuevo vídeo del canal X"	El canal es el sujeto, los suscriptores los observadores.
TikTok	"Alguien ha respondido a tu comentario"	Event-driven.
Fortnite	Eventos globales del juego	Todos los jugadores reciben el evento a la vez.

Ejemplo práctico

Simula el patrón en clase:

- Alumno A (sujeto) levanta la mano.
- Tres alumnos voluntarios (observadores) reaccionan diciendo "AVISO RECIBIDO".

Así de simple funciona en software.

2. STRATEGY (Estrategia)

“Permite cambiar un algoritmo sin modificar el código que lo usa.”

Idea principal

El algoritmo cambia, pero el objeto que lo usa permanece igual.

Ejemplos reales

App	Estrategias posibles
Google Maps / Uber	Ruta más rápida, más corta, evitando peajes, ecológica…
Spotify	Diferentes algoritmos de recomendación por usuario.
Amazon	Ordenar productos: precio, relevancia, más vendidos, entrega más rápida.
Netflix	Elegir códec de vídeo según ancho de banda.

Ejemplo práctico en clase

Pídele a un alumno que ordene una lista de números.

- Estrategia 1: ordenar de menor a mayor
- Estrategia 2: ordenar de mayor a menor
- Estrategia 3: ordenar números pares primero

El “contexto” (la lista) es el mismo.
La estrategia cambia, el algoritmo cambia.

COMMAND (Comando)

“Cada acción es un objeto que se puede ejecutar, deshacer o rehacer.”

Idea principal

Perfecto para **acciones reversibles**, **historiales**, **automatizaciones** y **tareas programadas**.

Ejemplos reales

App / Sistema	Acción como comando
Photoshop / CapCut	Deshacer, rehacer filtros y ediciones.
Whatsapp	Eliminar mensaje «para mí / para todos».
Videojuegos	Movimiento reversible, combo ejecutado, macro de acciones.
Domótica (Alexa)	"Encender luces", "poner música", "programar alarma".

Ejercicio
Define los comandos para una app de dibujo:

- Dibujar línea
- Borrar
- Deshacer
- Rehacer
- Cambiar color

Cada acción es un comando distinto.

3. STATE (Estado)

"Un objeto cambia su comportamiento según su estado interno."

Idea principal
Evita escribir código tipo:
if (estado == "PREPARANDO") {…}
else if (estado == "ENVIADO") {…}
else if (estado == "ENTREGADO") {…}

En su lugar, cada estado es un objeto con su propio comportamiento.

Ejemplos reales

Caso	Estados posibles
Amazon	Pedido: recibido → preparando → enviado → entregado → devuelto
WhatsApp	Usuario: online → escribiendo → offline
Netflix	Reproductor: play → pause → buffering → stop
Juegos	Personaje: atacando → defendiendo → cargando habilidad

Ejemplo práctico

Simula un pedido en clase:

1. "Pedido Recibido" → permite cancelar.
2. "Enviado" → no permite cancelar.
3. "Entregado" → permite valorar.

Cada estado tiene reglas diferentes.

4. TABLA COMPARATIVA DE LOS 4 PATRONES

Patrón	Problema que resuelve	Ejemplo claro	Ventaja principal
Observer	Notificar cambios a muchos objetos	Notificaciones de Instagram	Escalabilidad
Strategy	Intercambiar algoritmos	Rutas de Google Maps	Flexibilidad
Command	Encapsular acciones reversibles	Deshacer en Photoshop	Control total
State	Cambiar comportamiento con el estado	Estados del pedido de Amazon	Código limpio

5. ACTIVIDADES COMPLETAS PARA CLASE

ACTIVIDAD 1: Identificar patrones en apps que usan a diario

Analizar una app (WhatsApp, TikTok, Discord, Spotify) y responder:

1. ¿Dónde aparece Observer?
2. ¿Dónde aparece Strategy?
3. ¿Dónde aparece Command?
4. ¿Dónde aparece State?

Solución orientativa para TikTok

- **Observer**: notificaciones de nuevos seguidores.
- **Strategy**: elegir calidad de vídeo según velocidad de conexión.
- **Command**: deshacer cambios en edición.
- **State**: vídeo pausado / reproduciéndose / cargando.

ACTIVIDAD 2: Diseñar sistema de notificaciones

Aplicar **Observer**.

Se debe identificar:

- Sujeto
- Observadores
- Evento
- Acción resultante

Ejemplo: “nuevo mensaje en un grupo de WhatsApp”.

ACTIVIDAD 3: Cambiar de estrategia

Aplicar **Strategy**.

Dales una lista de productos y varias estrategias:

- Ordenar por precio
- Ordenar por valoración
- Ordenar por antigüedad

Que diseñen cómo se cambiaría la estrategia sin reescribir la aplicación.

ACTIVIDAD 4: Sistema de comandos de un videojuego

Aplicar **Command**.
Diseñar:

- Mover personaje
- Lanzar ataque
- Deshacer movimiento
- Repetir ataque (macro)

ACTIVIDAD 5: Estados de un pedido

Aplicar **State**.
Diseñar estados:

- Recibido
- Preparando
- Enviado
- Entregado

Y definir:

- Acciones permitidas
- Transiciones válidas

6. RESUMEN

Los patrones de comportamiento son la base de apps modernas.

- **Observer**: notificaciones y eventos.
- **Strategy**: cambiar algoritmos fácilmente.
- **Command**: ejecutar y deshacer acciones.
- **State**: comportamiento dinámico según el estado.

Los cuatro permiten escribir aplicaciones:

- Escalables
- fáciles de mantener
- flexibles
- que no se rompen ante cambios