DISEÑO DE SISTEMAS

Juan Carlos Molina Lozano

Docente

CONTENIDO

- Introducción
- Objetivos de la Clase
- Propósito de los Patrones de Comportamiento
- Características de los Patrones de Comportamiento
- Ventajas y Desventajas

OBJETIVOS DE LA CLASE

- Comprender qué son los patrones de comportamiento en el diseño de software.
- Reconocer su importancia para mejorar la comunicación entre objetos y responsabilidades.
- Identificar los principales patrones de comportamiento y analizar su aplicación en ejemplos prácticos.

INTRODUCCIÓN

En el diseño de sistemas complejos, no solo importa cómo se crean los objetos, sino también cómo interactúan entre ellos para cumplir con tareas específicas. Los patrones de comportamiento resuelven problemas relacionados con la asignación de responsabilidades, la comunicación entre clases y la flexibilidad en la ejecución de algoritmos. Este tipo de patrones permiten construir software más modular, mantenible y escalable, favoreciendo el principio de responsabilidad única y el principio de abierto/cerrado.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Los patrones de comportamiento son soluciones reutilizables a problemas comunes que surgen al intercambiar mensajes entre objetos. En otras palabras, definen cómo los objetos se comunican y cooperan entre sí para lograr una tarea. En lugar de centrarse en la estructura o creación, se centran en la lógica del flujo y en cómo se delegan responsabilidades.

Principales Patrones de Comportamiento

Observer (Observador)

Command (Comando)

Mediator (Mediador)

Chain of Responsibility (Cadena de Responsabilidad)

Template Method (Método Plantilla)

Strategy (Estrategia)

Iterator (Iterador)

Memento (Recuerdo)

State (Estado)

Visitor (Visitante)

PATRONES DE COMPORTAMIENTO

Observer (Observador)

- Propósito:

Permite que un objeto (sujeto) notifique automáticamente a múltiples observadores cuando cambia su estado.

Aplicación:

Ideal para sistemas donde muchas partes deben reaccionar a cambios de otra parte, como interfaces gráficas, sensores o eventos.

Ejemplo:

Un botón notifica a varios listeners (son objetos o funciones que escuchan eventos, en este caso, el evento de que el botón fue presionado) cuando se hace clic.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Strategy (Estrategia)

- Propósito:

Permite definir una familia de algoritmos, encapsularlos y hacerlos intercambiables.

- Aplicación:

Útil cuando un algoritmo puede variar según el contexto, por ejemplo, métodos de pago (tarjeta, efectivo, PayPal).

- Ejemplo:

Una app de navegación usa distintos algoritmos para calcular rutas: más corta, sin peajes, por autopista.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Command (Comando)

- Propósito:

Encapsula una solicitud como un objeto, permitiendo parametrizar clientes con distintas solicitudes, colas o historial de operaciones.

- Aplicación:

Ideal para menús, operaciones “deshacer/rehacer”, tareas programadas.

- Ejemplo:

Un control remoto ejecuta comandos como encender, apagar, subir volumen. Cada acción es un objeto "comando".

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Iterator (Iterador)

- Propósito:

Permite acceder secuencialmente a los elementos de una colección sin exponer su representación interna.

- Aplicación:

Recorrer listas, árboles, conjuntos de objetos.

- Ejemplo:

Un iterador permite recorrer una lista de productos sin importar cómo están almacenados internamente.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Mediator (Mediador)

- Propósito:

Define un objeto que encapsula la interacción entre otros objetos, reduciendo el acoplamiento entre ellos.

- Aplicación:

Interfaces donde muchos objetos deben interactuar (como componentes GUI) sin conocerse directamente.

- Ejemplo:

Una torre de control aéreo coordina varios aviones sin que se comuniquen entre sí.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Memento (Recuerdo)

- Propósito:

Permite capturar y restaurar el estado interno de un objeto sin violar su encapsulamiento.

- Aplicación:

Sistemas con funcionalidad “deshacer” o recuperación de estado.

- Ejemplo:

Un editor de texto guarda estados para permitir al usuario deshacer cambios.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Chain of Responsibility (Cadena de Responsabilidad)

- Propósito:

Permite pasar una solicitud entre una cadena de objetos hasta que uno la maneje.

- Aplicación:

Filtros, validadores o manejo de eventos.

- Ejemplo:

Un sistema de soporte pasa un ticket desde el primer nivel de atención hasta el experto si es necesario.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

State (Estado)

- Propósito:

Permite que un objeto cambie su comportamiento cuando cambia su estado interno, como si cambiara de clase.

- Aplicación:

Sistemas que cambian de comportamiento en tiempo de ejecución (como reproductores multimedia, cajeros automáticos).

- Ejemplo:

Un cajero cambia de comportamiento si hay fondos, si está sin red o si está en mantenimiento.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Template Method (Método Plantilla)

- Propósito:

Define el esqueleto de un algoritmo en una clase base y permite que las subclases redefinan ciertos pasos sin cambiar la estructura general.

- Aplicación:

Procesos estandarizados con pasos comunes pero algunos personalizables.

- Ejemplo:

Un método para generar informes que siempre tiene encabezado, cuerpo y pie, pero cada uno puede ser personalizado.

QUÉ SON LOS PATRONES DE COMPORTAMIENTO?

Visitor (Visitante)

- Propósito:

Permite agregar nuevas operaciones a objetos sin modificar sus clases, separando los algoritmos de la estructura de datos.

- Aplicación:

Sistemas donde se necesita aplicar múltiples operaciones sobre estructuras de objetos complejas.

- Ejemplo:

Una aplicación fiscal aplica distintos cálculos (IVA, retención, auditoría) sobre distintas facturas sin modificar las clases de factura.

CARACTERÍSTICAS DE LOS PATRONES DE COMPORTAMIENTO

- Se enfocan en la interacción entre objetos.
- Distribuyen de manera flexible el comportamiento y las responsabilidades.
- Permiten que los objetos cooperen sin acoplarse fuertemente.
- Facilitan la reutilización de algoritmos y procesos.

VENTAJAS Y DESVENTAJAS

Ventajas

- ☑ Promueven bajo acoplamiento entre clases.
- ☑ Aumentan la flexibilidad del sistema.
- ☑ Fomentan la reutilización de comportamientos.
- ☑ Facilitan el cambio y mantenimiento del código.

Desventajas

- ✗ Pueden generar más clases y estructuras.
- ✗ Aumentan la complejidad inicial del diseño.
- ✗ Requieren una comprensión profunda de la colaboración entre objetos.

REFERENCIAS

Refactoring Guru

Patrones de Comportamiento

https://refactoring.guru/es/design-patterns/behavioral-patterns

PREGUNTAS