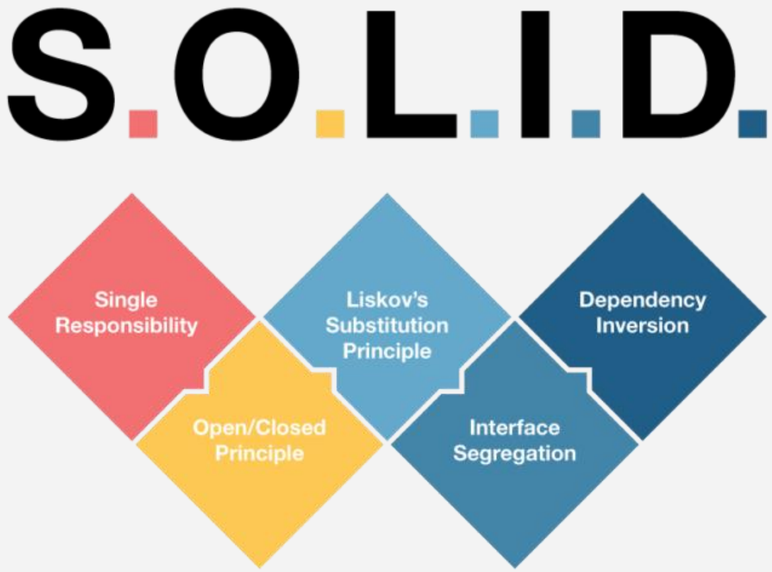
Sistema de Ventas

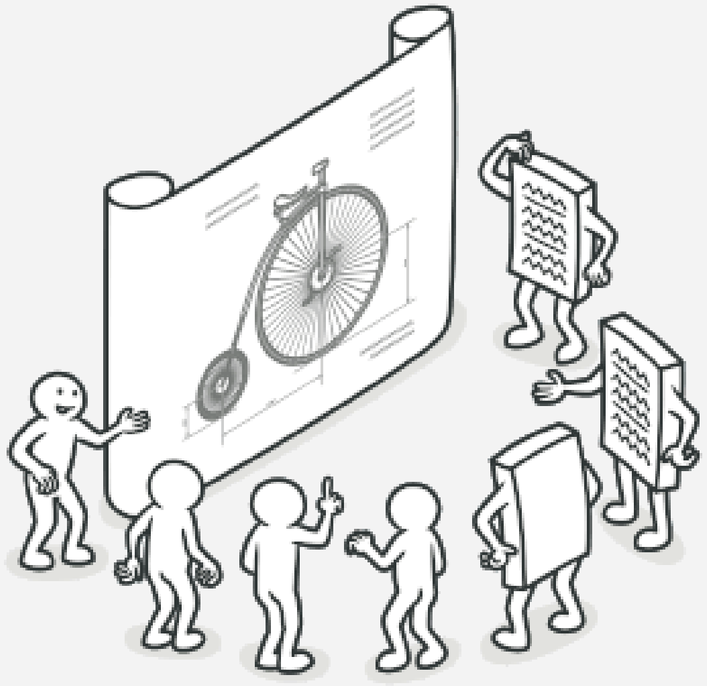
Integrantes:

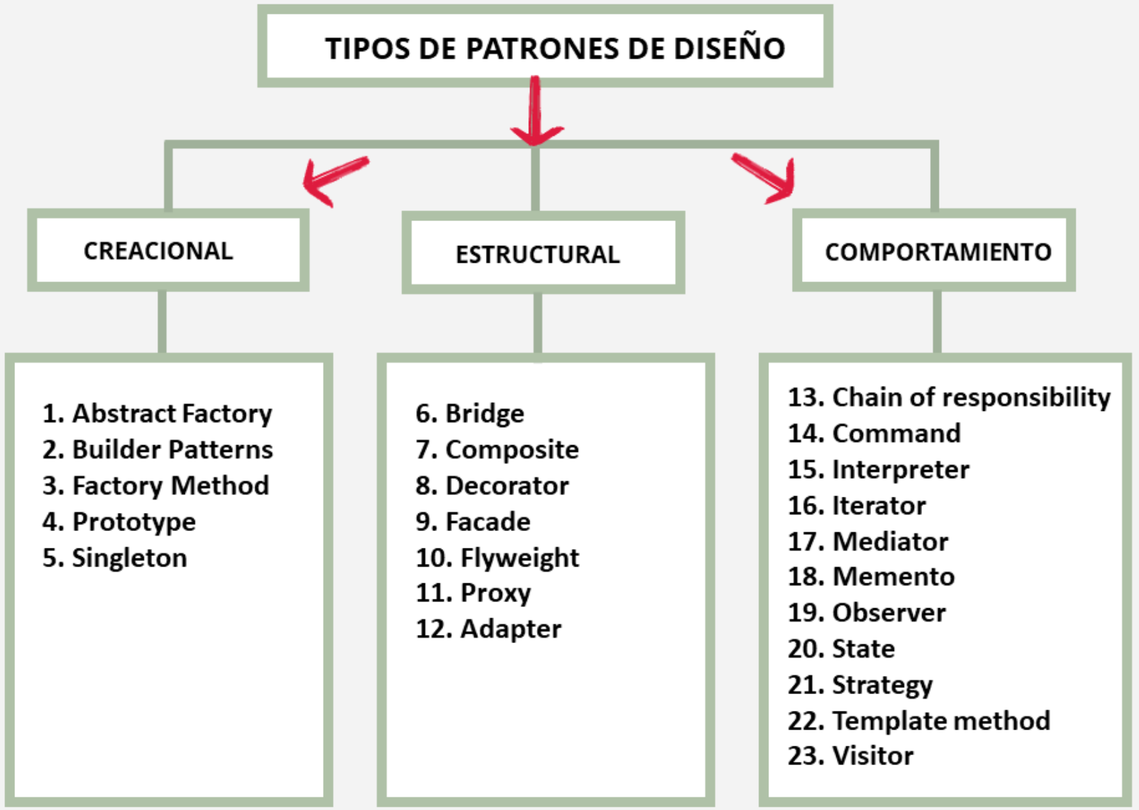
- Luis Angel Oros Sicha
- Sandro Piero Salazar Camacho

Principios SOLID



Introducción





Patrones usados

01
Creacional
Builder

02
Creacional
Factory Method

03
Estructural
Facade

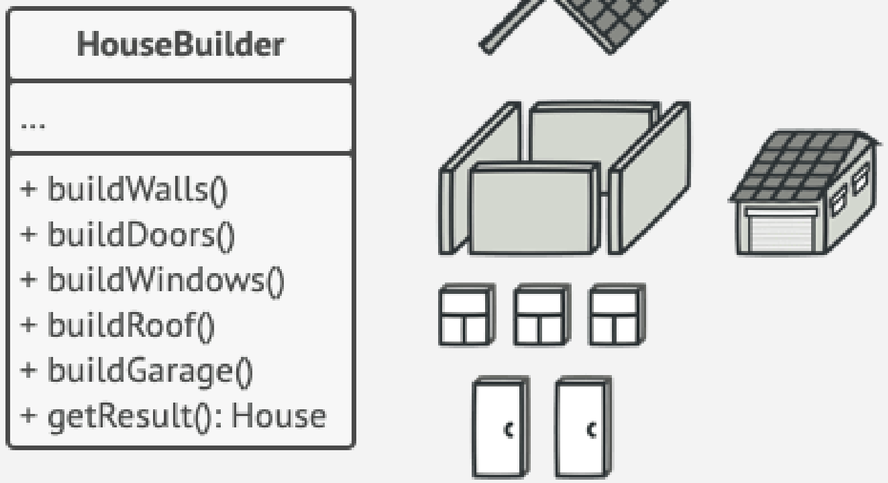
04
Comportamiento
Observer

05
Comportamiento
Command

06
Comportamiento
Strategy

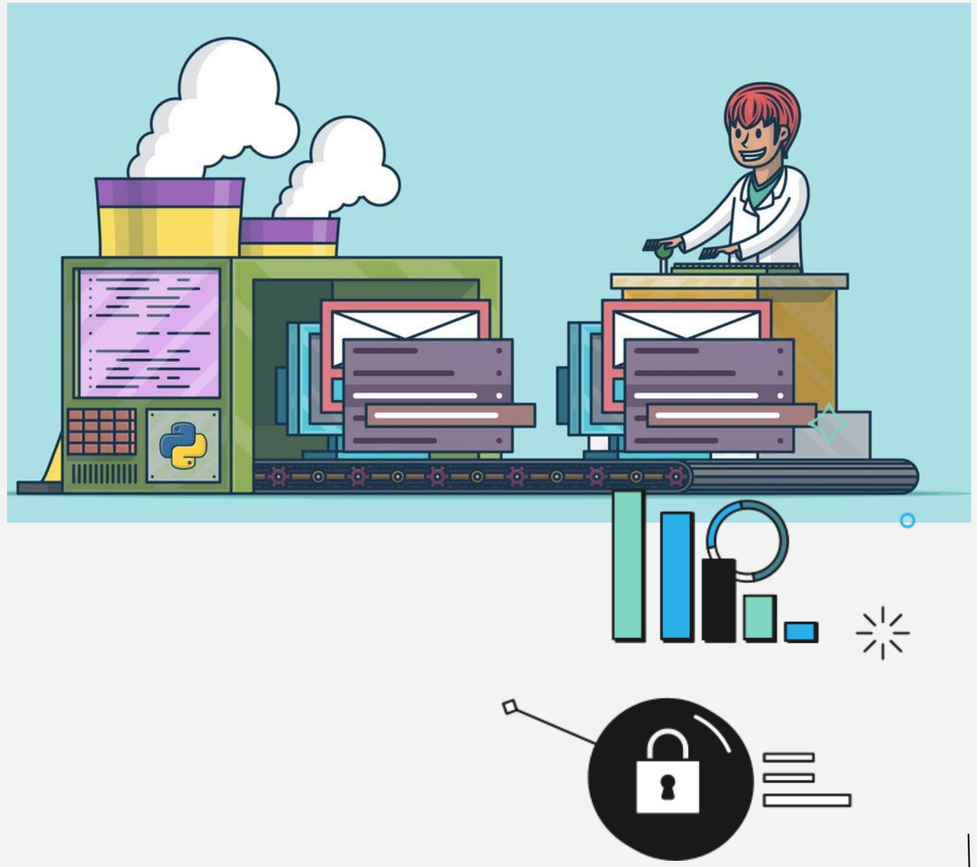
Patrón Builder

El patrón Builder separa la construcción de un objeto complejo paso a paso, permitiendo diferentes configuraciones sin contaminar su clase. Usa un Builder para definir la lógica de construcción y un Director (opcional) para orquestarla. Ideal para evitar constructores largos y crear objetos con múltiples variantes.



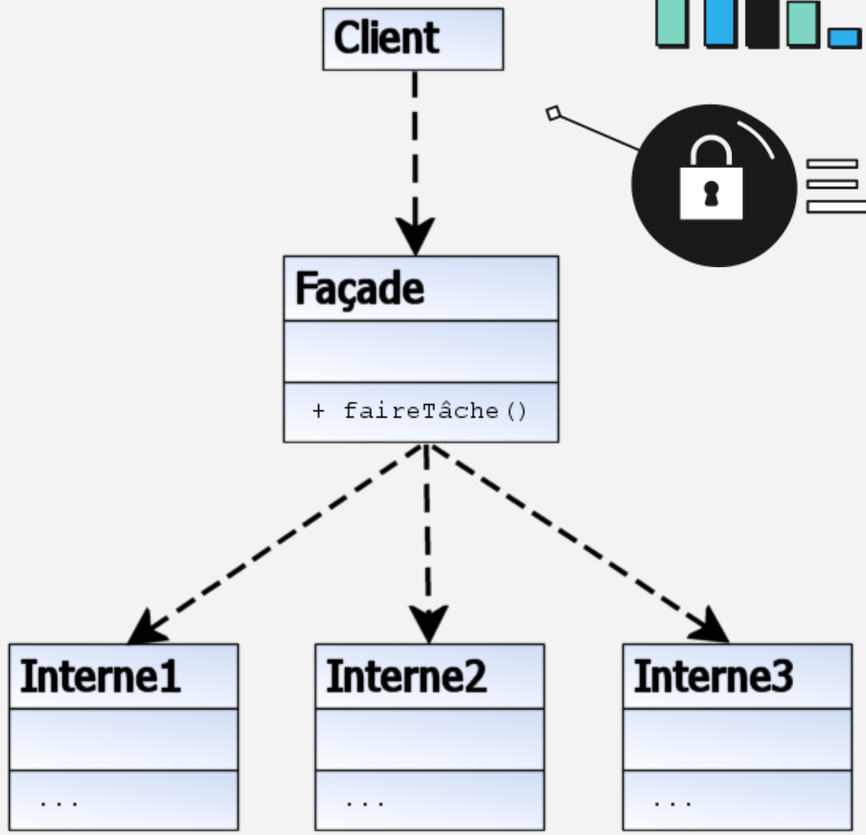
Patrón Factory Method

El Factory Method es un patrón creacional que delega la creación de objetos en subclases, permitiendo que una clase padre defina una interfaz común pero deje la implementación concreta a sus hijas.
✓ Ventaja: Flexibilidad para extender tipos de objetos sin modificar código existente.
✓ Uso típico: Cuando una clase no puede anticipar el tipo exacto de objetos que debe crear.
Ejemplo: Una fábrica de Productos (electrónicos/ropa) donde cada subclase decide qué instancia retornar.



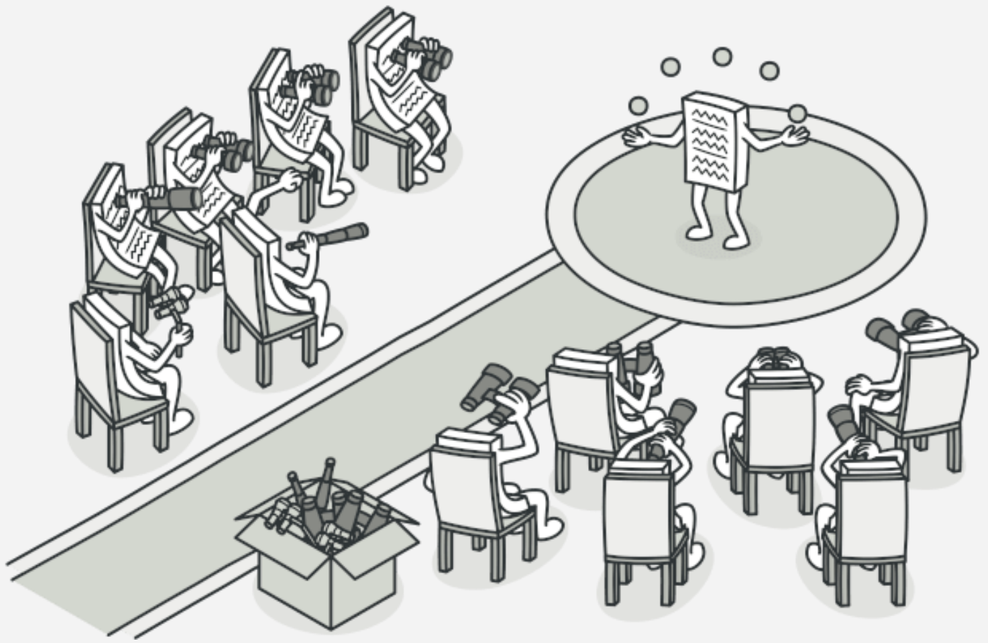
Patrón Facade

El patrón Facade proporciona una interfaz simplificada para un sistema complejo, ocultando sus detalles internos y facilitando su uso.
✓ Ventaja: Reduce la complejidad para el cliente, actuando como un "punto de acceso único" a subsistemas.
✓ Uso típico: Cuando hay múltiples componentes interdependientes o APIs complicadas que deben unificarse.
Ejemplo: Un sistema de ventas con múltiples módulos (inventario, pagos, envíos) donde Facade ofrece un único método realizarVenta() que coordina todo internamente.



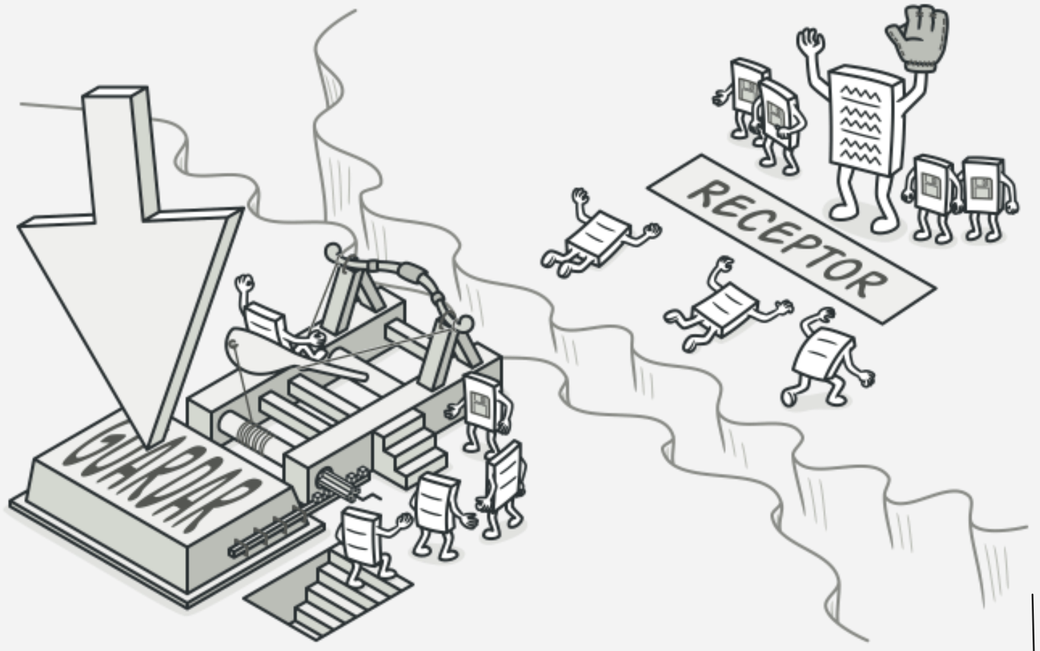
Patrón Observer

El patrón Observer establece una relación de dependencia uno-a-muchos entre objetos, donde cuando un objeto (sujeto) cambia su estado, notifica automáticamente a todos sus observadores.
✓ Ventaja: Desacopla el sujeto de sus observadores, permitiendo reacciones en tiempo real sin modificar el núcleo.
✓ Uso típico: En sistemas de eventos, notificaciones o actualizaciones en cascada (ej: alertas de stock, actualización de UI).



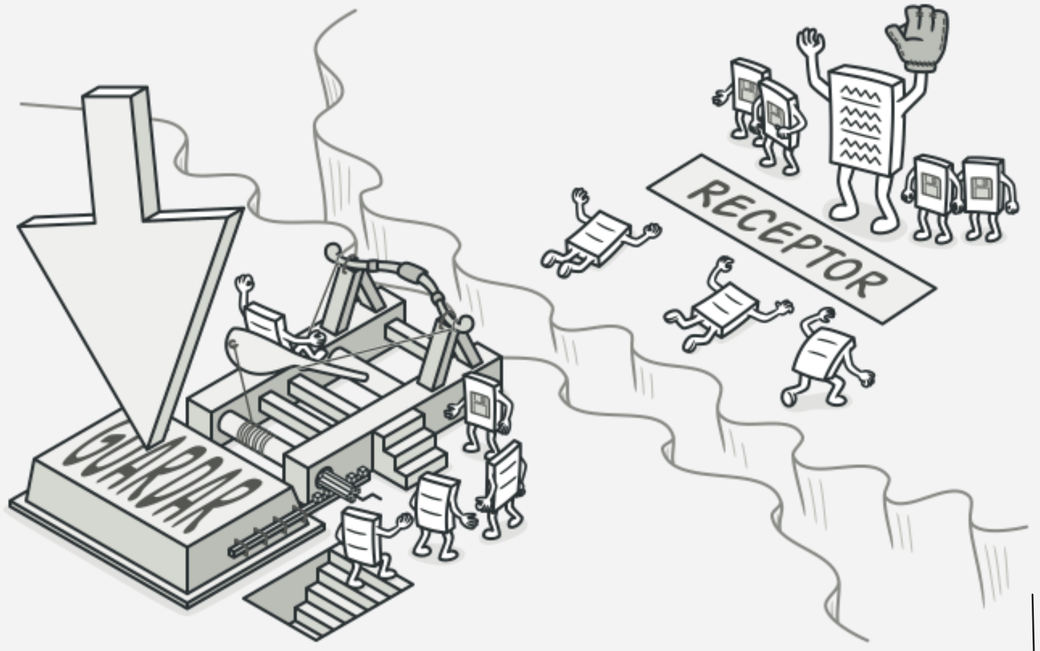
Patrón Command

El patrón Command convierte solicitudes en objetos independientes, permitiendo operaciones como ejecutar, deshacer o encolar acciones. Ideal para manejar transacciones reversibles (ej: pagos con rollback) y desacoplar al invocador del receptor. Ejemplo: Un comando ProcesarPago que ejecuta/revierte transacciones bajo demanda.



Patrón Strategy

El patrón Strategy define una familia de algoritmos intercambiables, encapsulando cada uno en una clase independiente y permitiendo que varíen sin modificar el código que los usa.
✓ Ventaja: Flexibilidad para cambiar comportamientos en tiempo de ejecución (ej: políticas de descuentos, métodos de ordenamiento).
✓ Uso clave: Cuando un sistema requiere múltiples variantes de una misma operación (ej: calcular impuestos, aplicar filtros).
Ejemplo: Un sistema de ventas con estrategias de descuento (DescuentoFijo, DescuentoPorVolumen), seleccionables dinámicamente.



Ahora el SOFTWARE

GRACIAS