DESARROLLO DE SOFTWARE

Paradigmas de Diseño (Implementación)

COHESIÓN / ACOPLAMIENTO

Cohesión

La cohesión se refiere a la fuerza de la relación entre los elementos dentro de un módulo. Un módulo con alta cohesión tiene elementos que se relacionan estrechamente y contribuyen a una única función o responsabilidad. Se distinguen diferentes niveles de cohesión

Acoplamiento

El acoplamiento se refiere al grado de interdependencia entre módulos. Un módulo con bajo acoplamiento depende mínimamente de otros módulos, lo que facilita su reutilización y modificación independiente. Se distinguen diferentes niveles de acoplamiento:

COHESIÓN / ACOPLAMIENTO

Ejemplos:

1. **autenticarUsuario(String usuario, String contraseña);**
2. **listarProductos(List productos, Integer pagina, Integer tamaño);**
3. **agregarItemCarrito(Integer idItem, String idCarrito);**

Una excelente forma de ver si tu código cumple con alta cohesión y bajo acoplamiento es la sencillez para testearlo!!!

SOLID

Clases de 1 sola tarea (Cohesión)

Orientado a frameworks (extender funcionalidad)

Herencia

Interfaces desacopladas y específicas.

Inyección de Dependencias. Las dependencias deberían ser a nivel implementacion y no a nivel interfaces.

PATRONES DE DISEÑO

Plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error.

Estos patrones proporcionan un enfoque estructurado y reutilizable para resolver situaciones recurrentes en el desarrollo de software. Los patrones de diseño ayudan a los desarrolladores a comunicarse y compartir soluciones eficientes y efectivas que han demostrado ser exitosas en el pasado.

Cada patrón de diseño tiene un propósito específico y ofrece una solución a un problema de diseño común. Al utilizar un patrón de diseño, los desarrolladores pueden aprovechar la experiencia acumulada y evitar reinventar la rueda, lo que resulta en un software más eficiente, mantenible y escalable.

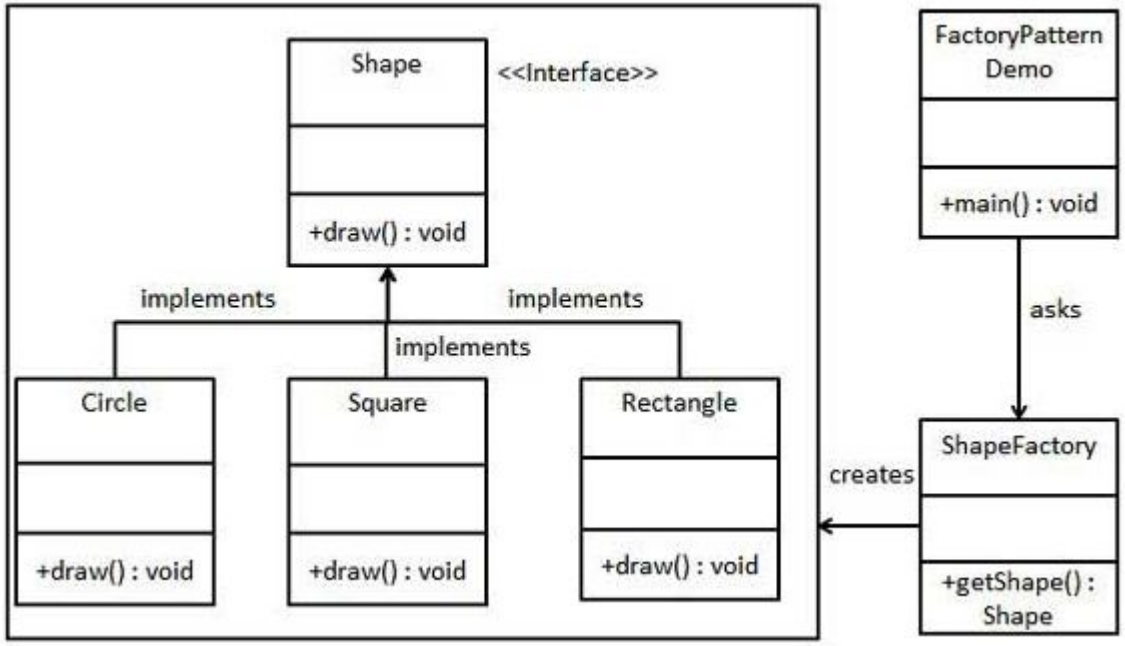
Es importante tener en cuenta que los patrones de diseño no son algoritmos o código listo para usar, sino más bien pautas y descripciones de soluciones de diseño. Los desarrolladores deben adaptar e implementar los patrones de diseño de acuerdo con los requisitos y características específicas de su proyecto de software.

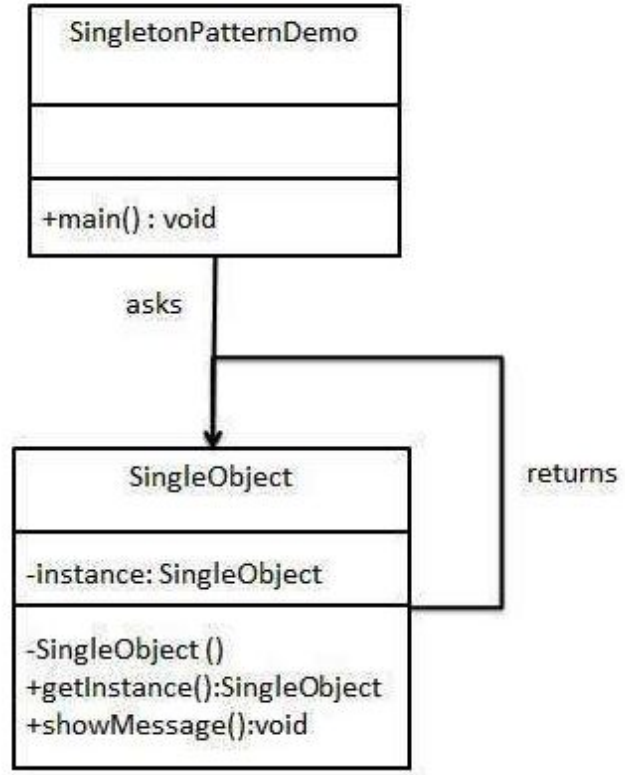
Los patrones de diseño de software se basan en principios y prácticas de diseño que han evolucionado a lo largo del tiempo. A menudo, se describen utilizando terminología y notaciones específicas para facilitar su comprensión y aplicación.

GOF (Gang of Four)

PATRONES CREACIONALES

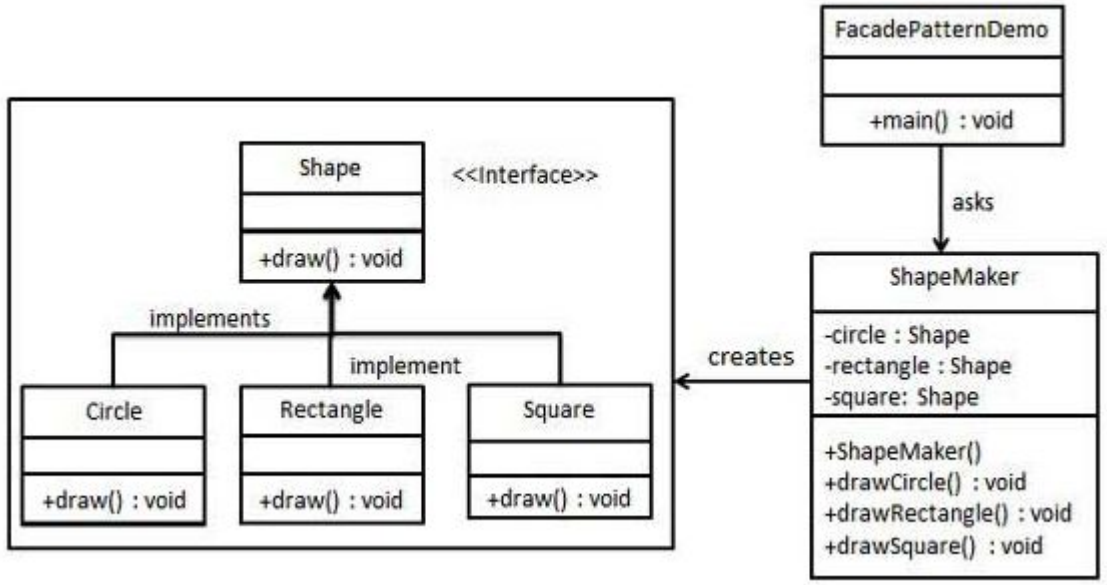
FACTORY AND **SINGLETON** PATTERN

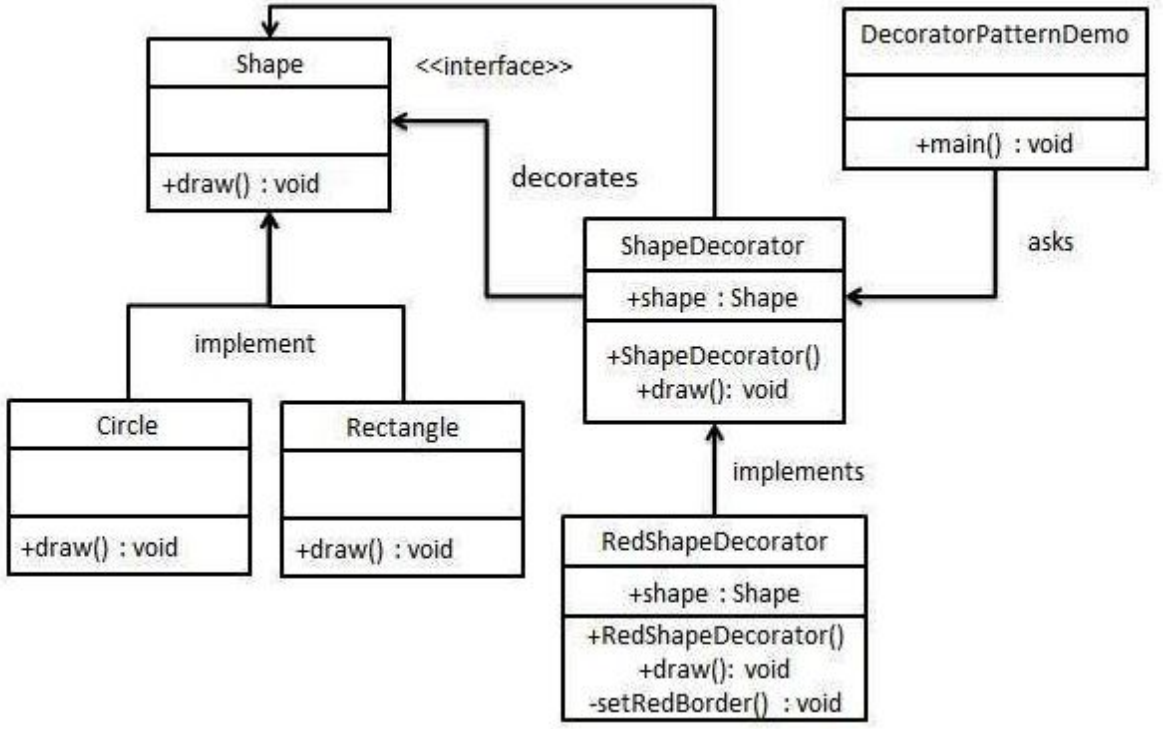




PATRONES ESTRUCTURALES

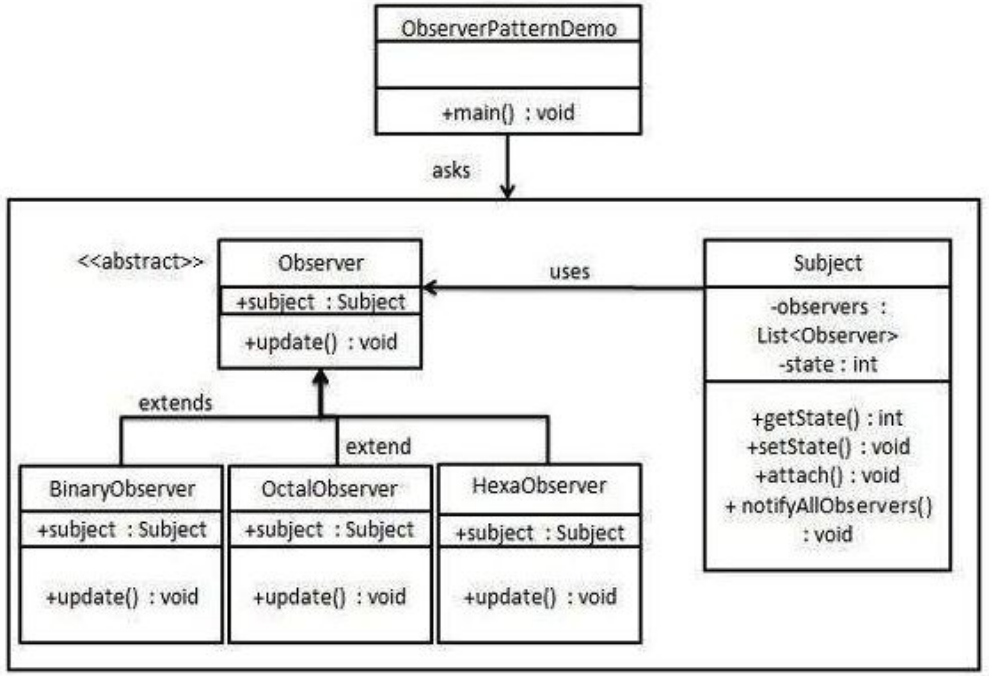
FACADE AND **DECORATOR** PATTERN

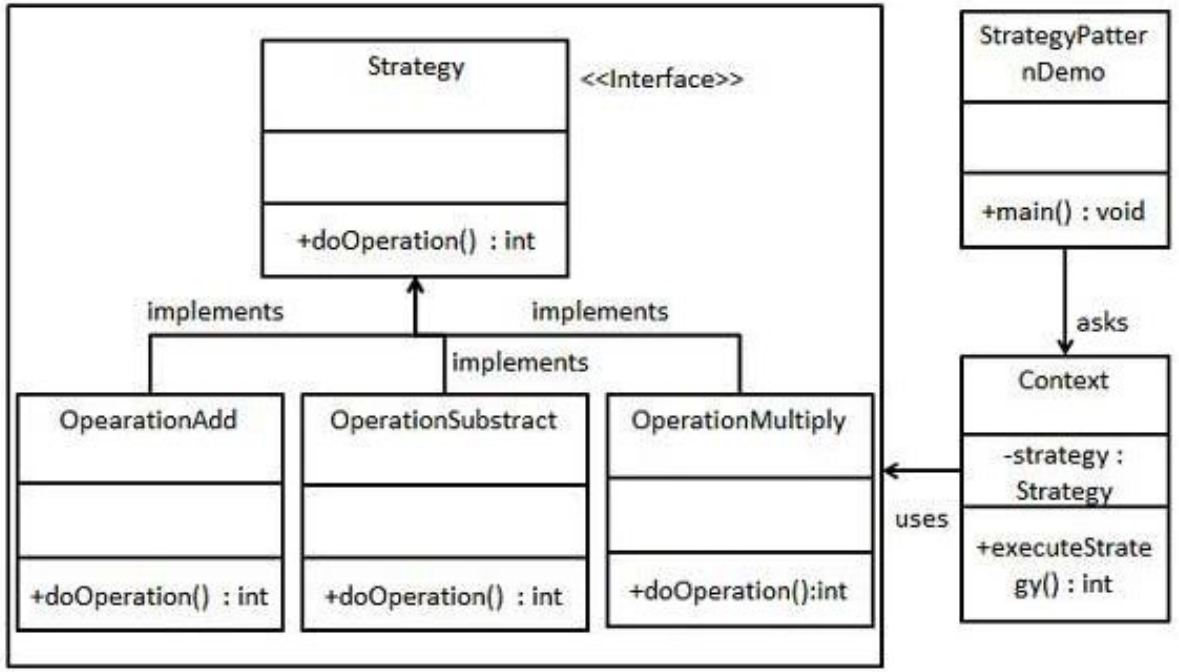




PATRONES DE COMPORTAMIENTO

OBSERVER AND **STRATEGY** PATTERN

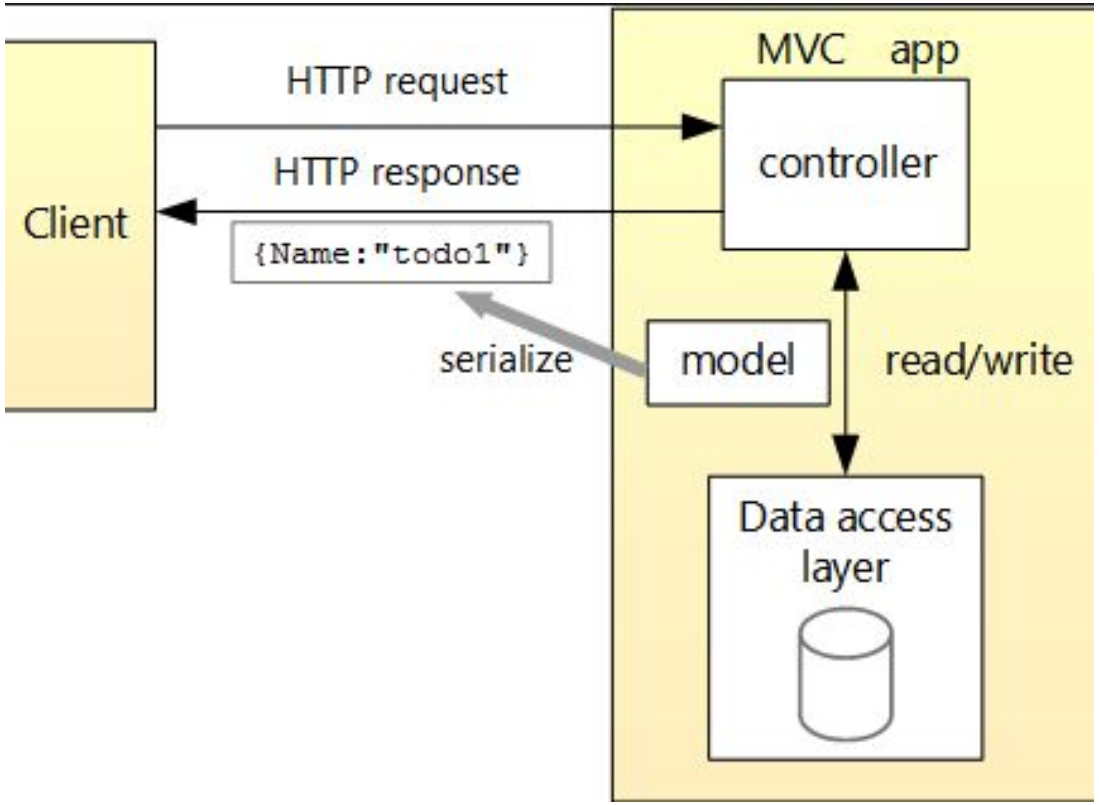




MVC - MODEL VIEW CONTROLLER

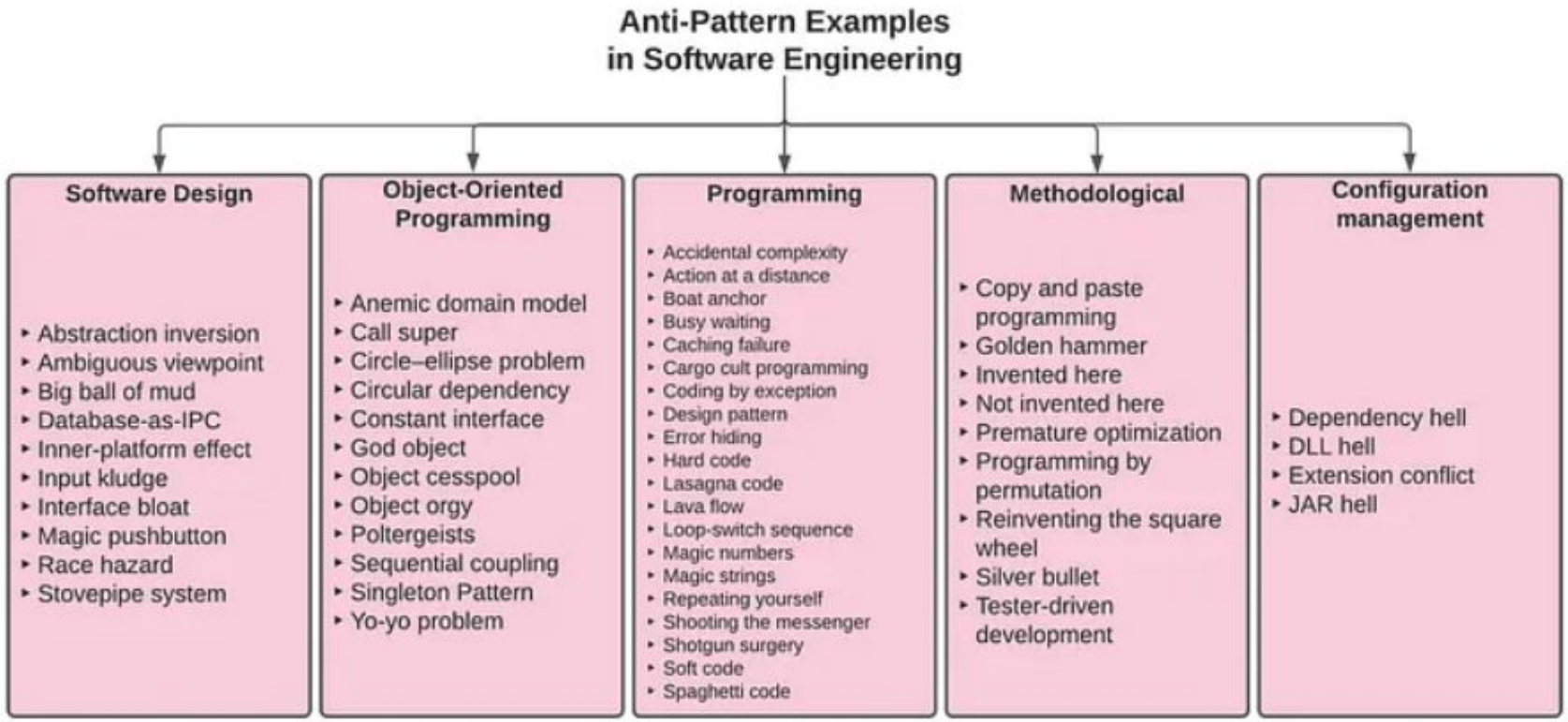
MODELO: *Un objeto llevando información*
VISTA: *Como se va a mostrar ese objeto*
CONTROLADOR: *Controla la información que va al modelo y actualiza la vista*

Ejemplo con el tp: caso ***Método de Pago API***



ANTI PATTERN

Malas prácticas, patrones oscuros… parece un patrón de diseño pero no es una solución completa.



ANTI PATTERN - EJEMPLOS

MAGIC PUSH BUTTON: Error común en desarrollo de interfaces gráficas donde el botón OK (Submit o similar) HACE TODO sin abstracción directamente en la interfaz.

ERROR HIDING: Esconder los errores, capturarlos y no mostrarlos.

GOD OBJECT: Objetos que hacen muchas cosas y interactúan con todos los servicios y por lo tanto saben TODO

PROYECTOS

BUSCAR PARADIGMAS Y DONDE IMPLEMENTARLOS