# ¿Qué es un Patrón de Diseño?

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Hay que tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Existes varios patrones de diseño popularmente conocidos, los cuales se clasifican como se muestra a continuación:

* **Patrones Creacionales:** Inicialización y configuración de objetos.
* **Patrones Estructurales**: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
* **Patrones de Comportamiento**: Más que describir objetos o clases, describen la comunicación entre ellos.

Veremos un poco en qué consisten los distintos tipos de patrones, cuáles son sus fines y qué beneficios nos aportan.

## Patrones Creacionales

* **Fábrica Abstracta (** Abstract Factory **)**

El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como por ejemplo la creación de interfaces gráficas de distintos tipos (ventana, menú, botón, etc.).

* **Método de Fabricación (** Factory Method **)**

Parte del principio de que las subclases determinan la clase a implementar.

* **Prototipado ( Prototype )**

Se basa en la clonación de ejemplares copiándolos de un prototipo.

* **Singleton**

Restringe la instanciación de una clase o valor de un tipo a un solo objeto.

* **MVC ( Model View Controler )**

Este patrón se eligió para ser posteriormente detallado, además de entregar ejemplo de la vida cotidiana.

## Patrones Estructurales

* **Adaptador (Adapter):** Convierte una interfaz en otra.
* **Puente (Bridge):** Desacopla una abstracción de su implementación permitiendo modificarlas independientemente.
* **Objeto Compuesto (Composite):** Utilizado para construir objetos complejos a partir de otros más simples, utilizando para ello la composición recursiva y una estructura de árbol.
* **Envoltorio (Decorator):** Permite añadir dinámicamente funcionalidad a una clase existente, evitando heredar sucesivas clases para incorporar la nueva funcionalidad.
* **Fachada (Facade):** Permite simplificar la interfaz para un subsistema.
* **Peso Ligero (Flyweight):** Elimina la redundancia o la reduce cuando tenemos gran cantidad de objetos con información idéntica.
* **Apoderado (Proxy):** Un objeto se aproxima a otro.

## Patrones de Comportamiento

* **Cadena de responsabilidad (Chain of responsibility):** La base es permitir que más de un objeto tenga la posibilidad de atender una petición.
* **Orden (Command):** Encapsula una petición como un objeto dando la posibilidad de “deshacer” la petición.
* **Intérprete (Interpreter):** Intérprete de lenguaje para una gramática simple y sencilla.
* **Iterador (Iterator):** Define una interfaz que declara los métodos necesarios para acceder secuencialmente a una colección de objetos sin exponer su estructura interna.
* **Mediador (Mediator)**: Coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.
* **Recuerdo (Memento):** Almacena el estado de un objeto y lo restaura posteriormente.
* **Observador (Observer):** Notificaciones de cambios de estado de un objeto.
* **Estado (Server):** Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.
* **Estrategia (Strategy):** Utilizado para manejar la selección de un algoritmo.
* **Método plantilla (Template Method**): Algoritmo con varios pasos suministrados por una clase derivada.
* **Visitante (Visitor):** Operaciones aplicadas a elementos de una estructura de objetos heterogénea.

# MVC

El **Modelo Vista Controlador (MVC)** es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se podrían definir como sigue:

El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'

La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida

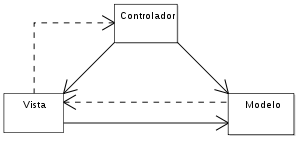


Fig 1.- Indica relaciones si es una línea solida indica asociación directa, si es punteada indica asociación indirecta

## Ejemplo práctico N°1

En el caso de preparar una comida: la receta con las instrucciones es el controlador, el producto en este caso el plato con la comida es la vista y los ingredientes son el modelo.

## Ejemplo práctico N°2

Otro caso puede ser las notas de un alumno: el reporte final o concentración de notas seria la vista, el libro de clases es el modelo y los cálculos de las notas acumulativas y cálculo de promedio son el controlador

# Conclusión

En este artículo se logró ilustrar sobre las distintas categorías y tipos de patrones de diseño, no debemos “reinventar la rueda” en varias de nuestras aplicaciones. Hay mucho trabajo ya realizado, testeado y aceptado que en un entorno similar a mi problema ya aporta una solución satisfactoria. ¿Para qué voy a inventar un ladrillo si ya otro lo hizo y el mismo ya fue usado en la edificación de millones de estructuras con éxito? Este modelo de MVC es fácil y flexible a estructuración del código, con datos, implementación e interfaz rígidas, y si el framework está bien hecho la seguridad también lo estará y la reutilización y mantención del código será amigable siempre y cuando el desarrollador tenga buenas prácticas en el desarrollo del software.