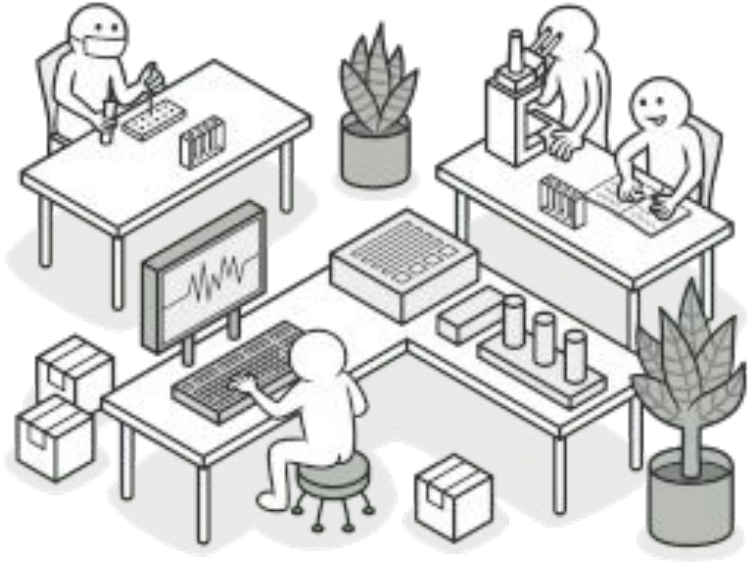
Patrones de diseño



Condiciones

- Debe resolver un problema en un contexto en particular.
- Debe ser recurrente: la solución debe ser relevante en otras situaciones.
- Debe ser educativo: la solución debe permitir entender cómo adaptarlo a la variante particular del problema donde se quiera aplicar.
- Debe tener un nombre identificativo para poder referirse al patrón.

¿Qué son los patrones de diseño?

Son un conjunto de soluciones a problemas que ocurren de forma constante. Estos patrones describen la esencia de la solución a cada uno de esos problemas de tal modo que pueda utilizarse en cualquier lenguaje o situación específica.

Para que una solución programática pueda considerarse un patrón, debe cumplir ciertas condiciones

Clasificación de patrones

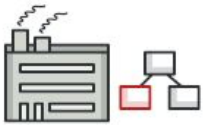
De Creación

Los patrones creacionales proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización del código existente.

El objetivo de estos patrones es abstraer el proceso de instanciación. Ayudan a que el sistema no necesite saber cómo se deben crear, componer y representar un conjunto de objetos.

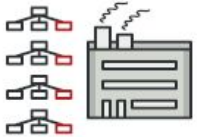
Clasificación de patrones

de Creación



Factory Method

Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

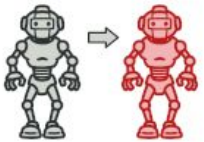


Abstract Factory

Permite producir familias de objetos relacionados sin especificar sus clases concretas.

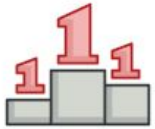
Builder

Permite construir objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.



Prototype

Permite copiar objetos existentes sin que el código dependa de sus clases.



Singleton

Permite asegurarnos de que una clase t
única instancia a la vez que proporcion
de acceso global a dicha instancia.

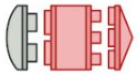
de Estructura

Clasificación de patrones

Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes, a la vez que se mantiene la flexibilidad y eficiencia de estas estructuras.

Estos patrones describen cómo las clases y los objetos se componen para formar estructuras más complejas. Se dividen en patrones orientados a clases (centrados en la herencia) y orientado a objetos (centrados en composición).

Clasificación de patrones **de Estructura**



Adapter

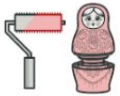
Permite la colaboración entre objetos con interfaces incompatibles.

Bridge

Permite dividir una clase grande o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.

Composite

Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

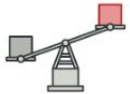


Decorator

Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

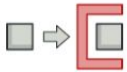
Facade

Proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.



Flyweight

Permite mantener más objetos dentro de la cantidad disponible de memoria RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.



Proxy

Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

Clasificación de patrones

de Comportamiento

Los patrones de comportamiento tratan con algoritmos y la asignación de responsabilidades entre objetos.

Estos patrones están relacionados con algoritmos y la asignación de responsabilidades entre objetos. Describen tanto los patrones entre clases u objetos como también los patrones de comunicación entre ellos. Estos patrones describen complejos flujos de control que son difíciles de seguir en tiempo de ejecución. El objetivo de estos patrones es que podamos desentendernos del flujo de control para concentrarnos en el modo en que se comunican o relacionan los objetos.

Clasificación de patrones de Comportamiento

Chain of Responsibility

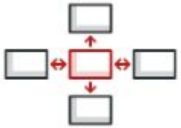
Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

Command

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

Iterator

Permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).



Mediator

Permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.

Memento

Permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

Observer

Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

Arquitectónico

Clasificación de patrones

Estos son patrones más complejos que intentan ofrecer soluciones a problemas recurrentes relacionados a la arquitectura de un sistema, es decir, cómo están organizadas y definidas las distintas partes internas de un programa en su conjunto.

Patrón MVC: Modelo/Vista/Controlador

Este patrón divide una aplicación en 3 capas.

- La capa del Modelo contiene la funcionalidad central y la operación con los datos.
- La capa de Vista se encarga de mostrar la información al usuario de una manera intuitiva.
- La capa del Controlador maneja la interacción entre el servidor y los usuarios como así también de invocar las operaciones de los modelos.

MVC consiste en tres tipos de objetos.

El Modelo es el objeto de aplicación, la Vista es su representación en pantalla y el Controlador define el modo en que la interfaz reacciona a la entrada del usuario. Antes de MVC, los diseños de interfaces de usuario tendían a agrupar estos objetos en uno solo.

MVC los separa para incrementar la flexibilidad y reutilización. MVC desacopla las vistas de los modelos estableciendo entre ellos un protocolo de suscripción/notificación.

Una vista debe asegurarse de que su apariencia refleja el estado del modelo. Cada vez que cambian los datos del modelo, éste se encarga de avisar a las vistas que dependen de él. Como respuesta a dicha notificación, cada vista tiene la oportunidad de actualizarse a sí misma.

PATRÓN MODELO VISTA CONTROLADOR

Las principales relaciones en MVC se dan entre los patrones de diseño

- Composite -> Patrón de Estructura
  Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.
- Strategy -> Patrón de Comportamiento
  Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.

Capa de Modelos

Esta capa es la encargada de ejecutar las operaciones propias del sistema. Son las menos dependientes de cualquier herramienta o framework que usemos para trabajar, y por ende las que más código nuestros requieren.

Un **ejemplo de clases** que podemos considerar que van dentro de esta capa son: **Persona, Socio, Préstamo, Usuario, etc.**

Capa de Modelos

Por otra parte, esta también es la capa más cercana a la BBDD, ya que necesita procesar datos. Estos datos debemos leerlos de una BBDD, entregárselos a la capa del Modelo, y que ésta luego nos devuelva el nuevo conjunto de datos modificados para volverlos a persistir en la BBDD. De todas maneras, por más que sea la capa más cercana a la BBDD, tampoco es responsabilidad suya comunicarse con la BBDD.

Capa intermedia entre BBDD y Modelos
Patrón DAO (optativa)

El patrón DAO (Data Access Object, Objeto de Acceso a Datos), es el patrón de diseño que veremos en próximas clases y es el que se encarga de leer y persistir los datos que tienen los modelos. De esta forma, la capa del Modelo se desliga completamente de la responsabilidad de saber cómo y de qué manera leer, crear, actualizar o eliminar datos de la BBDD.

Capa de Controladores

Esta capa la trabajaremos más adelante y es la encargada de orquestar las operaciones de nuestro sistema que son invocadas por el usuario (a través de la capa de Vistas).

Siguiendo el ejemplo de la biblioteca, un posible listado de operaciones serían las siguientes:

- Iniciar/cerrar sesión.
- Operación con usuarios (admins o socios). Altas, bajas, modificaciones ABM
- Operación con libros. Altas, bajas, modificaciones
- Realización de préstamos.

Capa de Controladores

Cada una de estas tareas y el subconjunto de tareas de cada una de ellas estarán agrupadas por una clase Controlador, donde cada uno de sus métodos representa cada una de las subtareas. Así, tendríamos las siguientes clases Controlador con sus adecuados métodos:

- **SesionControlador**.
  - iniciar()
  - cerrar()
- **UsuarioControlador**.
  - alta()
  - baja()
  - modificar()
  - listar()
- **LibroControlador**.
  - buscar()
  - listar()
  - alta()
  - baja()
  - modificar()
- **PrestamoControlaldor**.
  - alta()
  - baja()
  - modificar()

Capa de CONTROLADORES

Cada una de estas operaciones involucran a una o más clases de nuestra capa de Modelos. Es el Controlador el que sabe qué clases y cómo deben actuar en cada operación. Por eso lo de controlador: ***controla*** u orquesta la ejecución de cada operación de nuestro sistema.

Capa de VISTAS

Es la capa responsable de la **UI/UX (Interfaz de Usuario/Experiencia del Usuario**). En el caso de nuestra carrera está compuesto por los lenguajes de la web: **HTML, CSS y JavaScript**.

Las vistas son responsables de presentar adecuadamente la información como también las operaciones que podemos realizar. La lógica de esta capa está dada por diversos factores según el sistema. Por ejemplo, qué vistas pueden accederse de forma pública y cuáles no; qué pueden realizar los usuarios normales o los administradores.

Bibliografia/página Recomendada

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

https://1drv.ms/b/s!AmfQF8P5GBbl6Ql4IVo5LcAOSzay?e=tYo9US

https://1drv.ms/b/s!AmfQF8P5GBbl6Qo_E0aZO8e2nmc-?e=OIYudg.