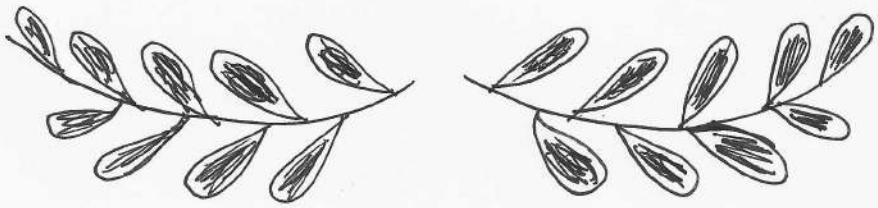


CAPITULO 4:

PATRONES DE DISEÑO

Los patrones de diseño facilitan la solución de problemas comunes existentes en el desarrollo de software con la interacción entre interfaz de usuario, lógica de negocio y los datos.

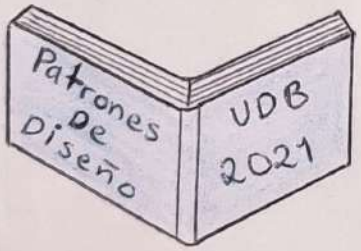
Tipos de patrones de diseño

Se clasifican en:

1. Creacionales: su objetivo es resolver los problemas de creación de instancias.

2. Estructurales: se ocupa de resolver problemas sobre la estructura de las clases.

3. Comportamiento: Ayudan a resolver problemas relacionados con el comportamiento de la aplicación. (interacción y responsabilidad entre objetos y clases de la aplicación).



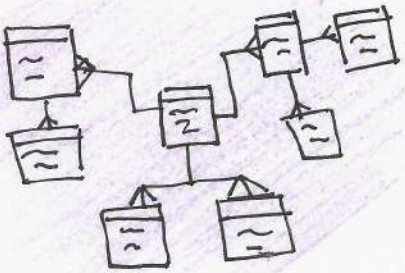
Dos patrones más utilizados
- MVC (Modelo Vista Controlador)
- MVP (Modelo Vista Presentador)

Ambos tratan de separar la presentación de la lógica de negocio y de los datos. Esto facilita el desarrollo de una aplicación y favorece su mantenimiento.

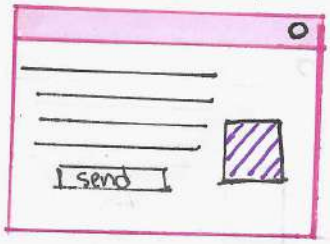
La idea principal es que cada una de las capas tenga su propia responsabilidad.

El patrón de diseño MVC es uno de los más conocidos por la comunidad de desarrolladores de software. Plantea el uso de 3 capas para separar la interfaz de usuario de los datos y la lógica del negocio. Estas capas son:

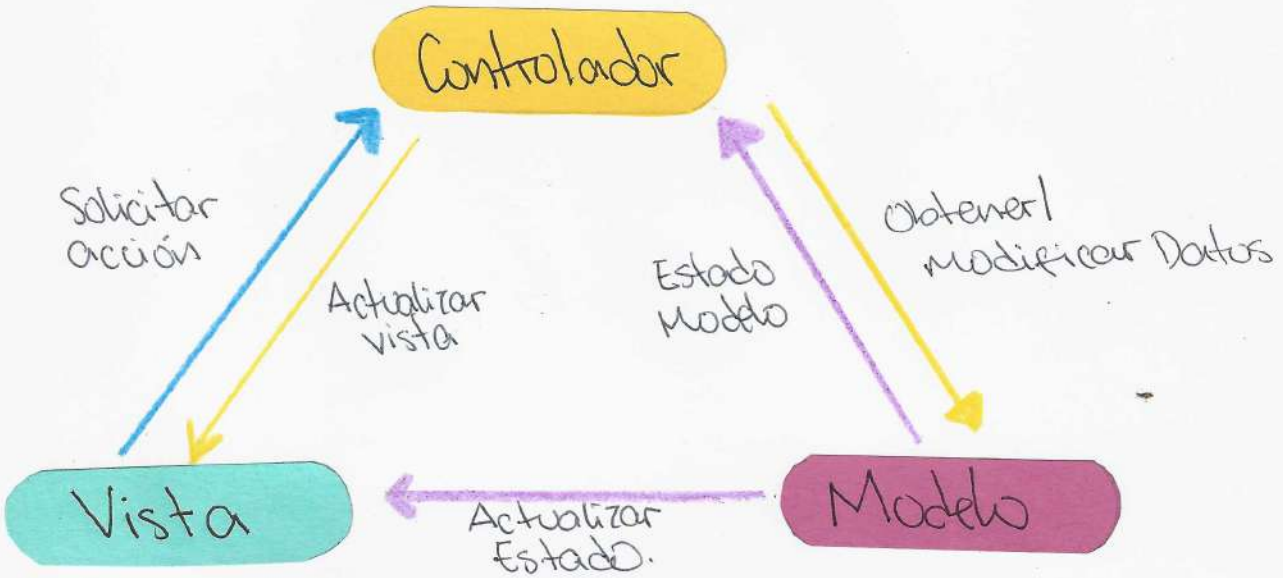
Modelo: Contiene el conjunto de clases que definen la estructura de datos con los que vamos a trabajar en el sistema. Su principal responsabilidad es el almacenamiento y persistencia de los datos de nuestra aplicación. El modelo es independiente de la representación de los datos en la vista.



Vista: Contiene la interfaz de usuario de nuestra aplicación. Maneja la interacción del usuario con la interfaz para enviar peticiones al controlador. Podemos tener múltiples vistas para representar un mismo modelo de datos.

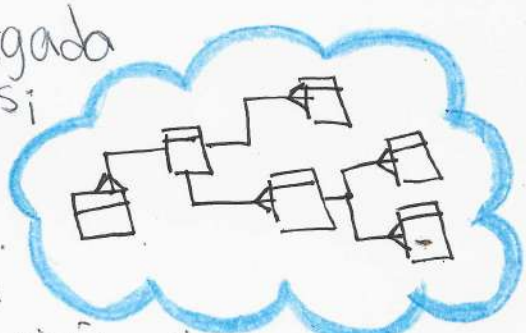


Controlador: Capa intermediaria entre la vista y el modelo. Es capaz de responder a eventos capturandolos por la interacción de usuarios en la interfaz, para posteriormente procesar la petición y solicitar datos o modificarlos en el modelo, retornando a la vista el modelo para representarlo en la interfaz.

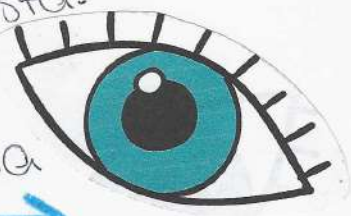


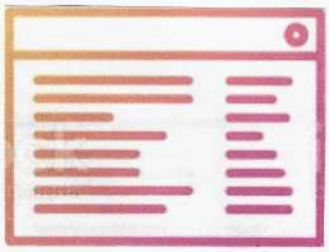
El patrón MVP deriva del MVC y nos permite separar aún más la vista de la lógica del negocio y de los datos. En este patrón, toda la lógica de la presentación de la interfaz reside en el Presentador, de forma que este da el formato necesario a los datos y los entrega a la vista para que esta simplemente pueda mostrarlos sin realizar ninguna lógica adicional.
Capas que componen este patrón:

Modelo: Es la capa encargada de gestionar los datos; su principal responsabilidad es la persistencia y almacenamiento de datos. En esta capa se encuentra la lógica del negocio de la aplicación, utilizando los interactores para realizar peticiones al servidor con el fin de obtener o actualizar los datos y devolvérselos al presentador.



Vista: La vista NO es una Activity o Fragment, simplemente es una interfaz de comportamiento de lo que podemos realizar con la vista. Sin embargo, son las Activity o fragments los encargados de atender a la interacción del usuario por pantalla





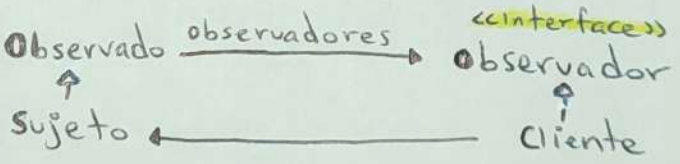
Para comunicarse con el presentador. Únicamente deben implementar la interfaz con la vista, que servirá de puente de comunicación entre el presentador y las Activities de forma que a través de los métodos implementados representen por pantalla los datos.

Presentador: Es la capa que actúa como intermediaria entre el modelo y la vista. Se encarga de enviar las acciones de la vista hacia el modelo de tal forma que, cuando el modelo procese la petición y devuelva los datos, el presentador los devolverá asimismo a la vista. El presentador NO necesita conocer la vista, ya que se comunica con ella usando una interfaz.

File Edit View Run Terminal Help X

Patrones de diseño •

```
<h2> Patrón Observer </h2>
<p> El patrón Observer se basa en dos objetos
    con una responsabilidad bien definida: </p>
<ol>
    <li> Observables </li>
    <p> Son objetos con un estado concreto,
        capaces de informar a los suscriptores
        suscritos al observable y que desean ser
        notificados sobre cambios de estado
        de estos objetos.
    </p> </br>
    <li> Observadores </li>
    <p> son objetos que se suscriben a los objetos
        observables y que solicitan ser notificados
        cuando el estado de los observables cambie.
    </p>
</ol>
</br>
```




```
<h2> Patrón Singleton </h2>
<p> Es un patrón de diseño creacional que
    nos permite asegurarnos de que una
    clase tenga una única instancia, a
    la vez que proporciona un punto de
    acceso global a dicha instancia
</p>
```