Capítulo 3 ~ Principios SOLID

¿Qué es SOLID? -> Principios para facilitar a los desarrolladores la labor de crear programas legibles y mantenibles

S - Principio de responsabilidad única: Una clase debe encapsular una única funcionalidad, es decir, hacer aquello que debe hacer y nada más para que sea más sostenible y fácil de mantener

O - Principio de ser abierto y cerrado: Debemos preparar el código para que esté abierto a extensiones y cerrado a modificaciones

Código que permita extender clases usando abstracción

Que todo lo que funcione ya no se toque

L - Principio de sustitución de Liskov: Toda clase que extienda la funcionalidad de una clase padre base debe implementar la funcionalidad de la clase base sin alterar su comportamiento

I - Principio de segmentación de interfaz: Se debe evitar en la medida de lo posible que nuestras clases contengan métodos de interfaces, que no implementamos al no ser necesaria su implementación para la clase en cuestión. Se considera como buena práctica dividir la interfaz en interfaces más pequeñas

D - Principio de inversión de dependencias: Este principio establece que, en el dominio de nuestro sistema, debemos utilizar interfaces o abstracciones, elementos que cambian con poca frecuencia, de tal forma que sean las concreciones de menor nivel las que dependan de abstracciones o interfaces y no a la inversa

Capítulo 4 - Patrones de diseño

Patrones de diseño → Resuelven los problemas relacionados con la interacción entre interfaz de usuario, lógica de negocios y los datos

Modelo vista controlador: Plantea el uso de 3 capas para separar la interfaz de usuario de los datos y la lógica del negocio

Modelo: Conjunto de clases que definen la estructura de datos con la que se va a trabajar, su principal utilidad es el almacenamiento y persistencia de los datos.

Vista: Interfaz de usuario de nuestra aplicación. Maneja la interacción del usuario con la interfaz para mandar peticiones al controlador

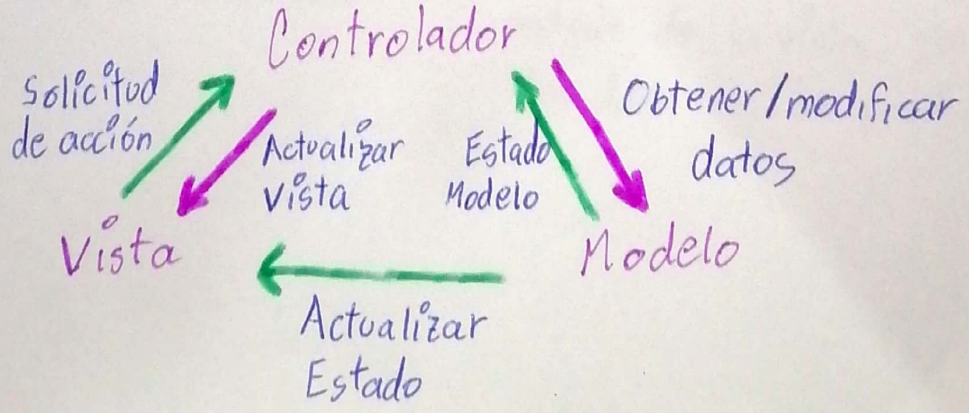
Controlador: Capa intermedia. Responde a eventos, capturándolos en la interfaz y procesando la información en el modelo, realizando las modificaciones en los datos

→ Secuencia de interacción

✓ Usuario interactúa con la vista → ✓ El controlador atiende la acción → ✓ El control solicita al modelo realizar las acciones

✓ El modelo se ve modificado y la información es mostrada nuevamente por el controlador en la vista

---



Modelo Vista presentador: separa aún más la vista de la lógica de negocios y los datos

Modelo: Capa encargada de gestionar los datos. En esta capa se encuentra la lógica de negocio de nuestra aplicación

Vista: Interfaz de comportamiento de lo que podemos hacer con la vista. Sin embargo son las activity o fragments los encargados de mantener la interacción del usuario por pantalla para comunicarse con el Presentador

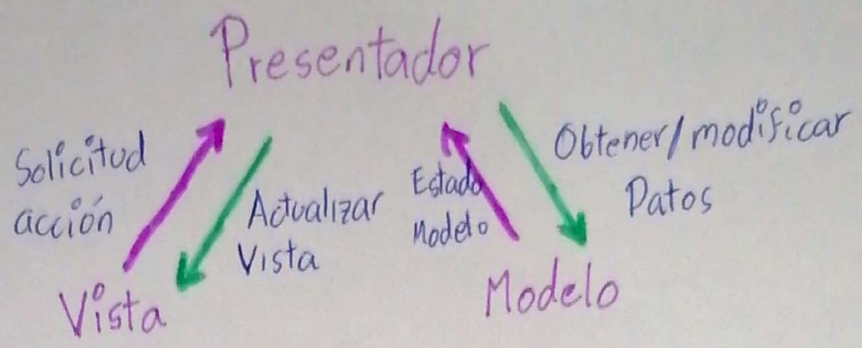
Presentador: Intermediario entre el modelo y la vista. Envía las acciones de la vista hacia el modelo de tar Forma que, cuando el modelo procese la petición y devuelva los datos, el presentador los devolverá asímismo a la vista.

⇝ Secuencia de interacción

✓El usuario interactúa con la interfaz de usuario → ✓ La activity recibe el evento y la vista, al conocer su Presentador, llama al método encargado de esa acción ↓

✓El presentador recibe la tarea que debe realizar ← ✓ El presentador llama a un Interactor del modelo Para que realice la tarea ↓

✓El Interactor analiza la petición y envía los resultados al presentador → ✓ Al recibir los resultados de la Petición, el presentador Invoca un método de la vista para representar los datos en la pantalla



⇝ Patrón Observer

Dos objetos con una responsabilidad bien definida

Observables: Objetos con un estado concreto, capaces de informar a su subscriptores que desean ser notificados sobre sus cambios de estado

Observadores: Objetos que se suscriben a los objetos observables y que solicitan ser notificados cuando el estado de estos observables cambie