Perdón por la letra :D

Fecha: / /

POO:

ASOCIACIÓN

COMPOSICIÓN: Creo antes todos los objetos necesarios para que se cumpla la composición

AGREGACIÓN

Flecha punteada ----> Creación de un objeto NEW

UNIDAD 2:

ARQUITECTURA EN CAPAS: *

ARQUITECTURA DE SOFTWARE

↳ Es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución

ESTILOS ARQUITECTÓNICOS

- Definen los patrones posibles de las aplicaciones
- Permiten evaluar alternativos con ventajas y desventajas conocidas ante diferentes conjuntos de req. no funcionales
- Sirven para sintetizar estructuras de soluciones

* SISTEMA EN CAPAS PUROS → ¡IMPORTANTE!! final

CADA CAPA SOLO PUEDE COMUNICARSE CON LA VECINA

AUNQUE ESTA SOLUCIÓN PUEDE SER MENOS EFICIENTE EN ALGUNOS CASOS FACILITA LA PORTABILIDAD DE LOS DISEÑOS.

Fecha: / /

(2)

3 CAPAS:

CAPA DE PRESENTACIÓN (capa gráfica lo que ve el usuario)

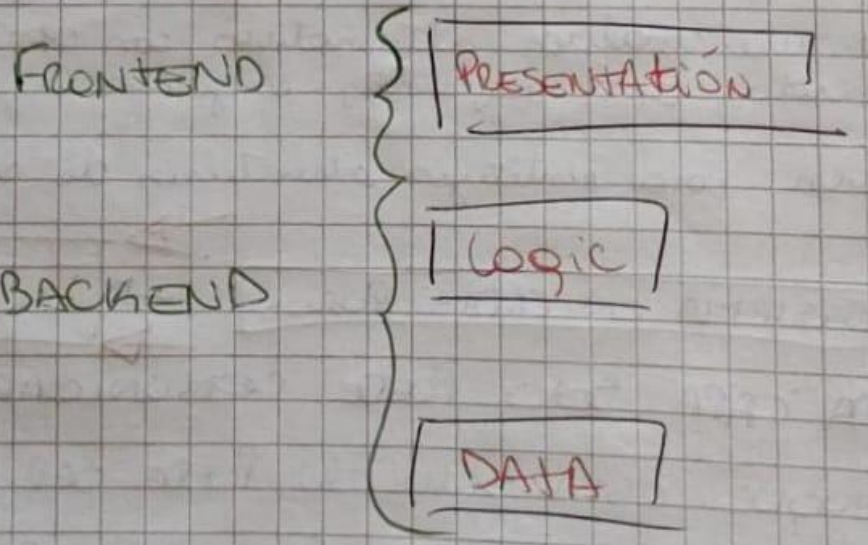
- Presentación de interfaz de usuario
- Captura de datos del usuario
- Validación de datos del usuario

CAPA DE NEGOCIO (lógica reglas de negocio)

- Modulo de negocio
- Recepción de solicitudes de la capa de Presentación
- Llamado a la capa de Datos

CAPA DE DATOS (almacenamiento de datos)

- Almacenamiento de datos
- Recuperación de datos (consulta)
- Interacción con capa de negocio



Fecha: / /

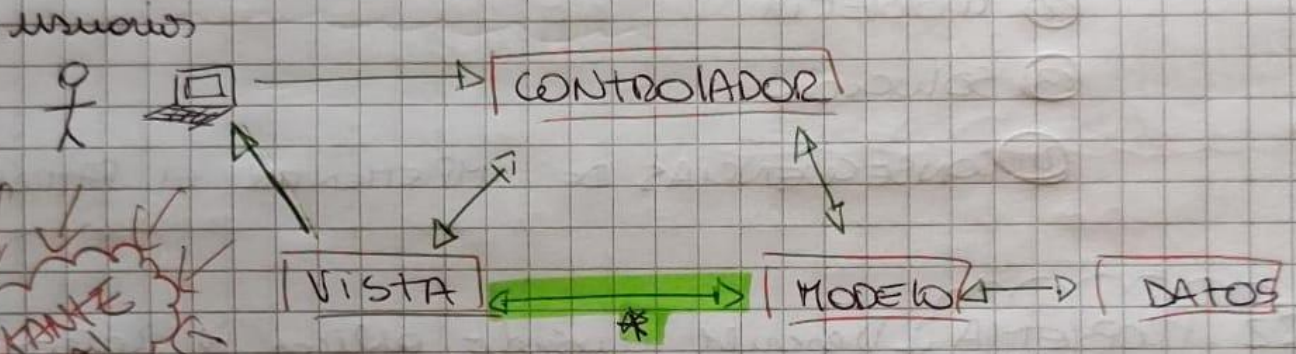
ARQUITECTURA MVC

↳ AÑOS 70 / LENGUAJE SMALLTALK

- **M**ODELO → OBJETO DE LA APLICACIÓN / DATOS APP lógica
- **V**ISTA → REPRESENTACIÓN EN PANTALLA
- **C**ONTROLADOR → MODO EN QUE LA INTERFAZ REACCIONA A LA ENTRADA DEL USUARIO

INCREMENTAR FLEXIBILIDAD y REUTILIZACIÓN

MVC → Desacopla las vistas de los modelos estableciendo entre ellos un protocolo de SUSCRIPCIÓN/NOTIFICACIÓN



IMPORTANTE FINAL

MVC PASIVO: NO NOTIFICA LOS CAMBIOS EN EL MODELO (no iría la flecha entre vista y modelo) *

MVC ACTIVO: EL MODELO NOTIFICA A LA VISTA SOBRE CAMBIOS (sí iría la flecha) *

Fecha: / /

(4)

Erich GAMMA LIBRO

PATRONES DE DISEÑO

PATRÓN DE DISEÑO: Describe un problema que ocurre repetidas veces en nuestro entorno, así como la solución a ese problema donde lo mismo se expresa en términos de objetos. → SOLUCIÓN

SOLUCIÓN GENÉRICA A UN PROBLEMA NO TRIVIAL, NO FACIL

PATRÓN → (4) ELEMENTOS ESCENCIALES

1. NOMBRE
2. PROBLEMA
3. SOLUCIÓN
4. CONSECUENCIAS DE IMPLEMENTAR EL PATRÓN

2. PROBLEMA: - Describe cuándo aplicar el patrón
- Explica el problema y su contexto
- A veces el problema incluye una serie de condiciones que deben darse para que tenga sentido aplicar el patrón

3. SOLUCIÓN: - Elementos que constituyen el diseño, sus relaciones, responsabilidades y colaboraciones

- NO DESCRIBE UN DISEÑO EN CONCRETO, sino que UN PATRÓN ES COMO UNA PLANTILLA QUE PUEDE APLICARSE EN MUCHAS SITUACIONES DIFERENTES.

ME DA EL DIAGRAMA DE CLASES INCOMPLETO

Fecha: / /

4. CONSECUENCIAS: - Son los resultados, las ventajas e inconvenientes de aplicar el patrón

- SON FUNDAMENTALES PARA EVALUAR LAS ALTERNATIVAS DE DISEÑO Y COMPRENDER LOS COSTOS Y BENEFICIOS DE APLICAR EL PATRÓN
- INCLUYEN SU IMPACTO SOBRE LA FLEXIBILIDAD, EXTENSIBILIDAD, PORTABILIDAD, SEGURIDAD, USABILIDAD DE UN SISTEMA

CLASIFICACIÓN DE LOS PATRONES

2 CRITERIOS → 1. PROPÓSITO: QUÉ HACE UN PATRÓN

→ ÁMBITO: ESPECIFICA SI EL PATRÓN SE APLICA PRINCIPALMENTE A CLASES U OBJETOS

- CLASES → ESTÁTICO
- OBJETOS → DINÁMICO

PROPÓSITO:

- DE CREACIÓN: Tienen que ver con el proceso de creación de los objetos.
- ESTRUCTURAL: tratan con la composición de clases u objetos
- DE COMPORTAMIENTO: Caracterizan el modo en que las clases y objetos interactúan y se reparten la responsabilidad.

Fecha: / /

6

AMBITO:

- CLASE: Se ocupan de las relaciones entre las clases y sus subclases. Estas relaciones se establecen a través de herencia, de modo que SON RELACIONES ESTÁTICAS (fijadas en tiempo de compilación)

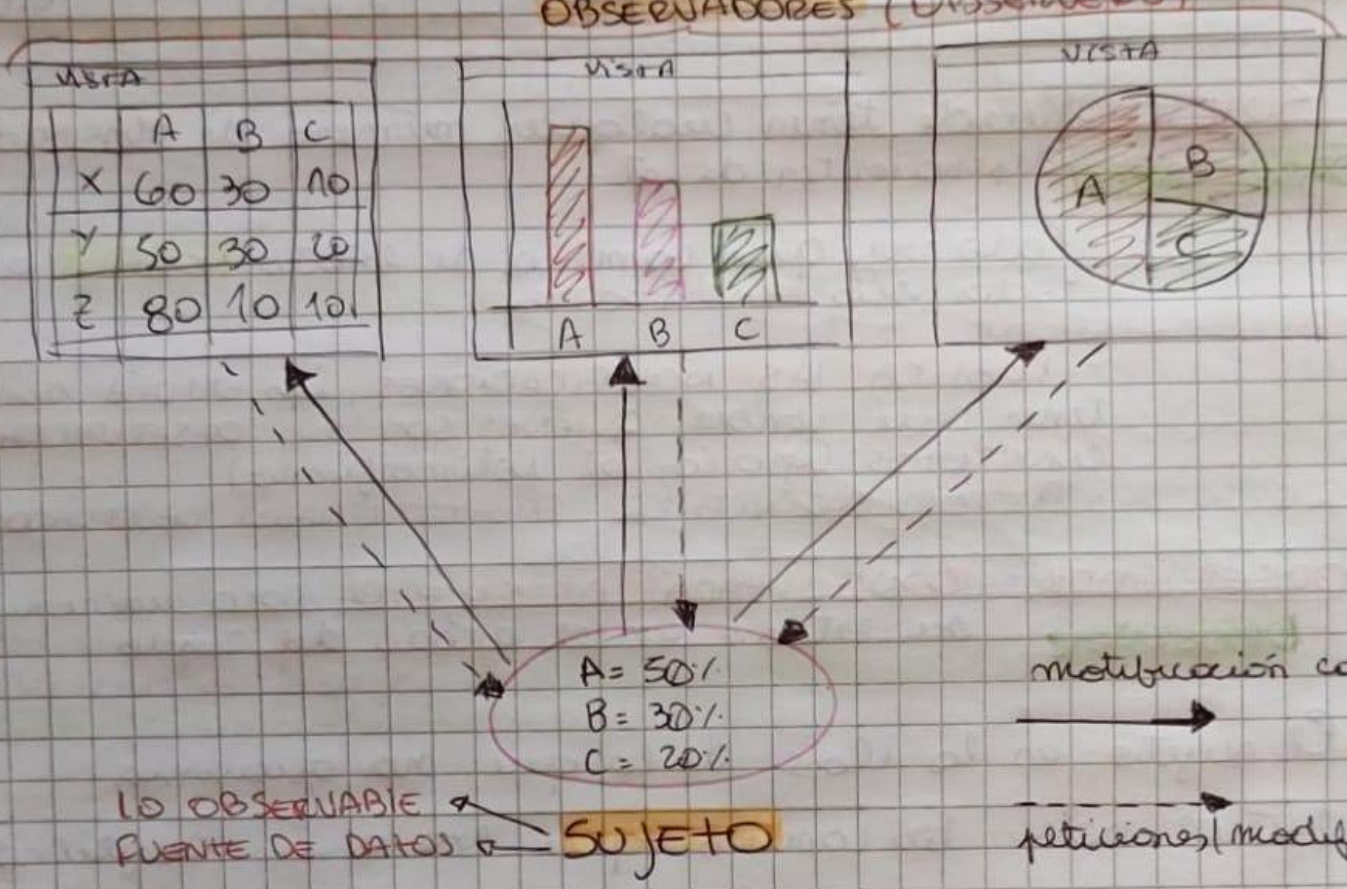
- OBJETO: tratan las relaciones entre objetos, que SON DINÁMICAS ya que pueden cambiare en tiempo de ejecución.

- RELACIÓN ESTÁTICA (en tiempo de compilación)
  Una vez que compilé no puedo hacer cambios en las estructuras de las clases
- RELACIÓN DINÁMICA (en tiempo de ejecución)
  Puedo interactuar con los objetos, cambiar su estado interno

PATRÓN OBSERVER:

PROPÓSITO: Define una dependencia de 1 a muchos entre distintos objetos, de forma que cuando un OBJETO cambia de ESTADO (se modifique alguno de sus atributos) SE NOTIFIQUE Y SE ACTUALICEN AUTOMÁTICAMENTE TODOS LOS OBJETOS QUE DEPENDEN DE ÉL

OBSERVADORES (OBSERVERS)



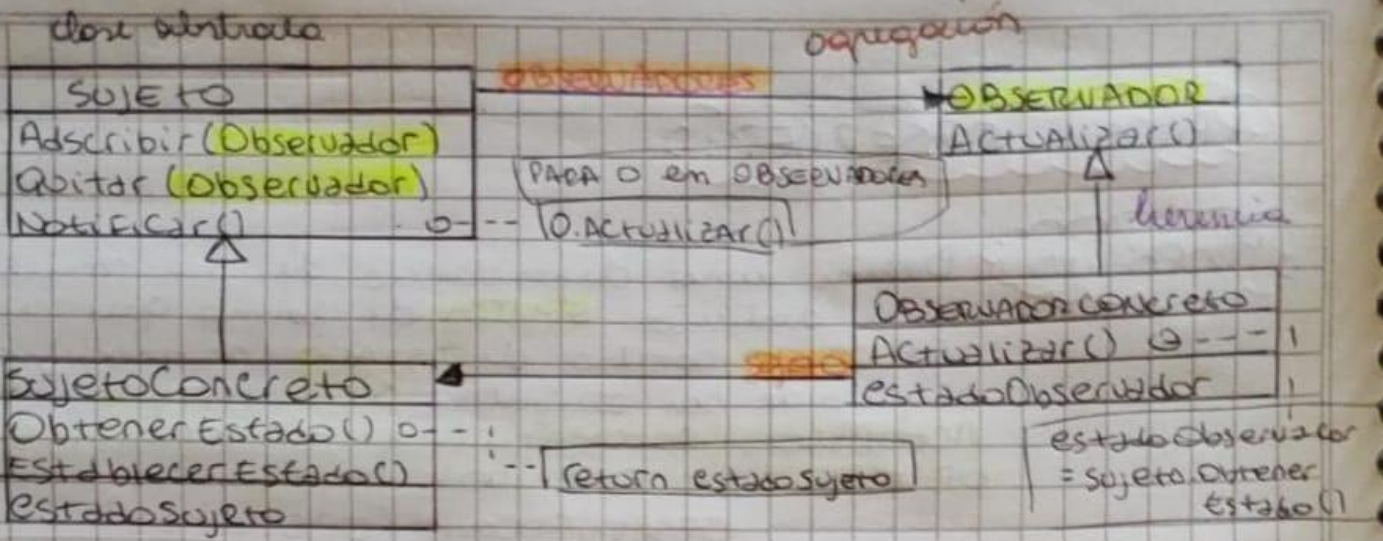
LAS VISTAS NO CONOCEN AL SUJETO, PERO SE COMPORTAN COMO SI LO CONOCIERAN

CUANDO SE CAMBIE LA INFORMACIÓN EN LA HOJA DE CÁLCULO (sujeto) LAS VISTAS VAN A REFLEJAR EL/LOS CAMBIOS

Fecha: 31/1

8

clase abstracta



SUJETO: - Puede tener cualquier número de observadores dependientes de él

OBSERVABLE

- Cada vez que cambia de estado NOTIFICA a todos sus observadores

- PUBLICA LAS NOTIFICACIONES, los ENVÍA sin tener que conocer QUIÉNES SON SUS OBSERVADORES CONCRETOS (gracias al polimorfismo)
- PUEDEN SUSCRIBIRSE INDETERMINADOS OBSERVADORES

OBSERVADOR: - CONSULTARÁ AL SUJETO para sincronizar su estado con el estado del sujeto

OBSERVER

- El sujeto es la clase de la que nos queremos interesar sobre los cambios que va a ir sufriendo

- LA SUSCRIPCIÓN DE OBSERVADORES AL SUJETO ES DINÁMICA PORQUE ES AGREGACIÓN POR TIEMPO DE EJECUCIÓN

- UN OBSERVADOR SÓLO SE SUSCRIBE A 1 SUJETO
  1 SUJETO TIENE MUCHOS OBSERVADORES

Fecha: / /

CUÁNDO APLICAR EL PATRÓN OBSERVER

- Cuando una abstracción tiene dos aspectos y uno depende del otro. Encapsular estos aspectos en objetos separados permite modificarlos y reutilizarlos en forma independiente

- Cuando un cambio en un objeto requiere cambiar otros, y no sabemos cuántos objetos necesitan cambiarse

- Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quiénes son dichos objetos. (Para evitar acoplamiento)

VENTAJAS:

1- **BAJO ACOPIAMIENTO ENTRE SUJETO Y OBSERVADOR**
El sujeto sabe que tiene una lista de observadores, que se ajustan a la interfaz Observador entonces el sujeto **NO CONOCE LA CLASE CONCRETA DE NINGÚN OBSERVADOR.**

2- **CAPACIDAD DE COMUNICACIÓN MEDIANTE DIFUSIÓN:**
La notificación enviada por el sujeto **no necesita especificar su receptor** porque se envía a todos los objetos suscritos. Al sujeto no le importa cuántos objetos suscritos haya, **su responsabilidad es notificar a sus observadores.**

DESVENTAJAS:

1- **SIN HISTORIAL DE CAMBIOS:**
No tengo manera de ver qué cambios se hicieron

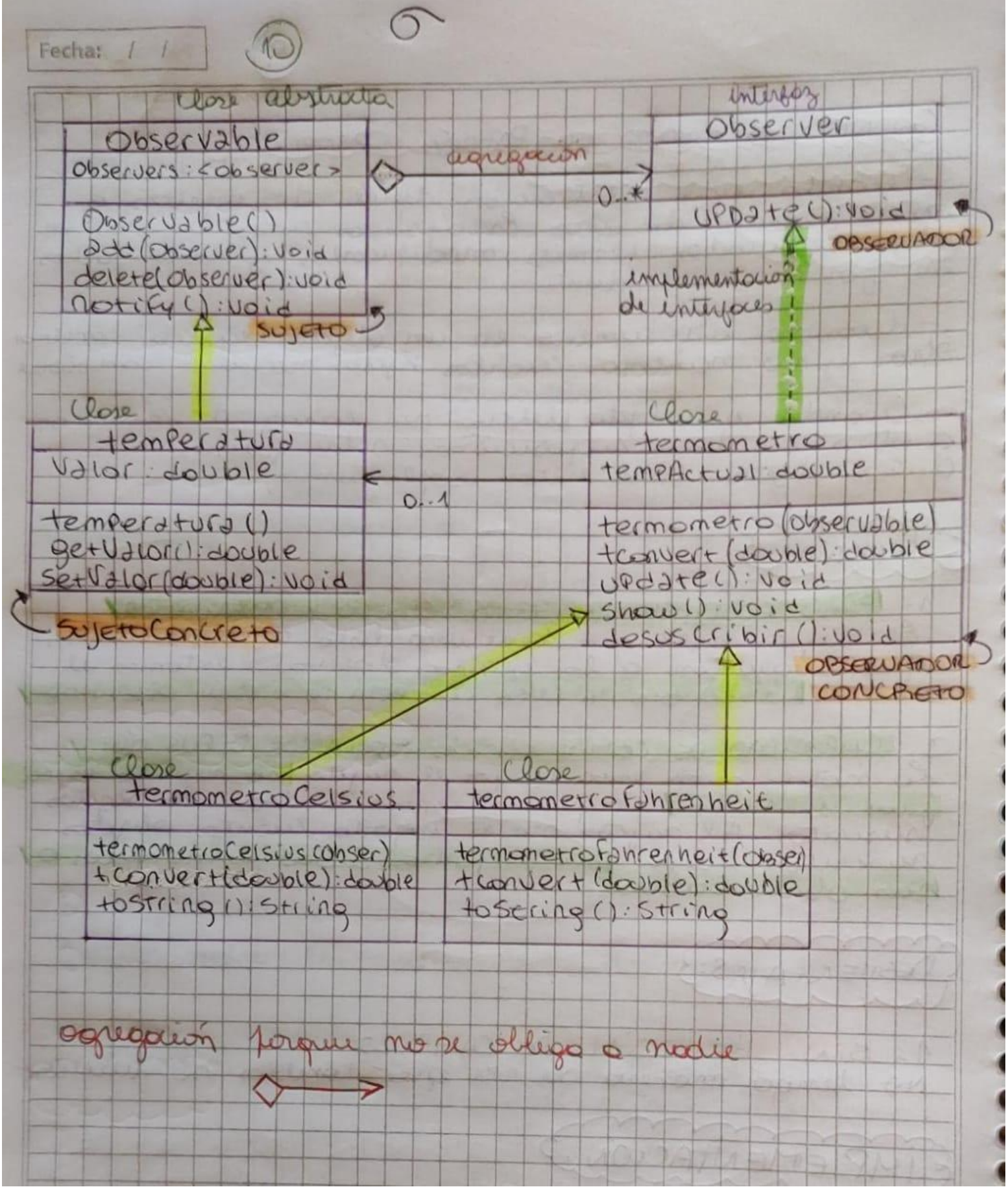
IMPLEMENTACIÓN:

MODELO PUSH: El sujeto envía a los observadores toda información acerca del cambio si se la solicitan o no

MODELO PULL: el sujeto sólo envía la **notificación mínima** y después los observadores piden los detalles
Es solo avisa que hubo un cambio despues el observador pide el nuevo estado.

Fecha: / /

10



agregación porque no se obliga a nadie

Fecha: / /

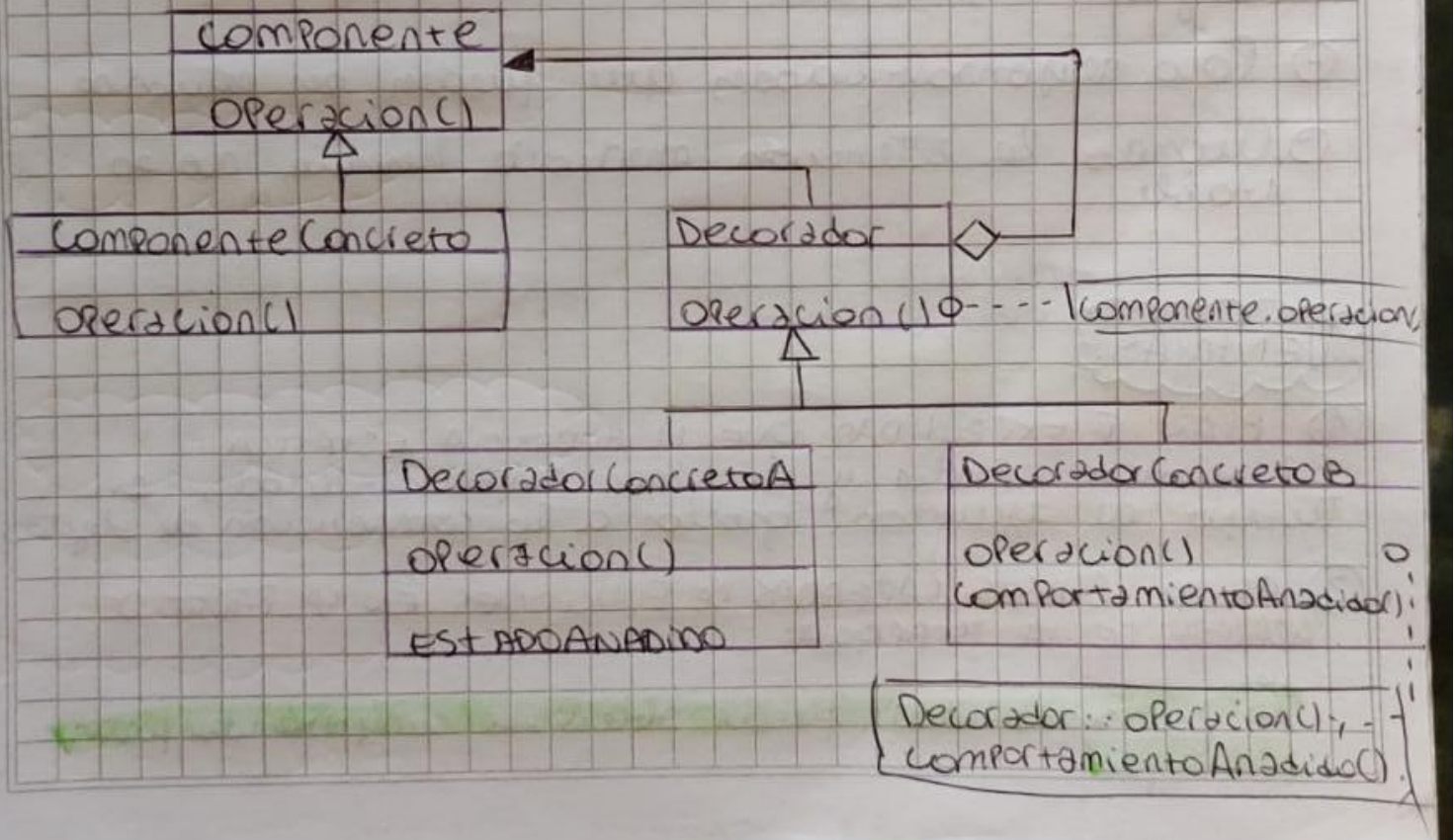
PATRÓN DECORATOR

PROPÓSITO: Permite asignar responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender funcionalidad

MOTIVACIÓN:

- Añadir responsabilidades a objetos individuales en vez de a todo una clase
- La herencia es ligada en tiempo de compilación por lo tanto es estática y no puede cambiar una variable que declare previamente.
- Flexibilidad al encerrar el componente en otro objeto que añade la responsabilidad

  DECORADOR
- El DECORADOR se ajusta a la interfaz del componente que decora, su presencia es transparente para sus clientes



Fecha: / /

12

PARTICIPANTES:

COMPONENTE: Define la interfaz para objetos a los que se puede añadir responsabilidades dinámicamente.

COMPONENTECONCRETO: Define un objeto al que se pueden añadir responsabilidades adicionales.

DECORADOR: Mantiene una referencia a un objeto componente y define una interfaz que se ajusta a la interfaz del componente.

DECORADORCONCRETO: Añade responsabilidades al componente.

CUANDO APLICARLO:

- Cuando necesite añadir objetos individuales de forma dinámica y transparente (sin afectar a otros objetos)
- Para responsabilidades que pueden ser retiradas
- Cuando la extensión mediante herencia no es viable.

VENTAJAS:

1. MAS FLEXIBILIDAD QUE LA HERENCIA ESTÁTICA
Se pueden añadir y eliminar responsabilidades en tiempo de ejecución gracias a la composición de objetos

2. EVITA CLASES CARGADAS DE FUNCIONES EN LA PARTE DE ARRIBA DE LA JERARQUÍA:

La funcionalidad puede obtenerse componiendo partes simples

Fecha: / /

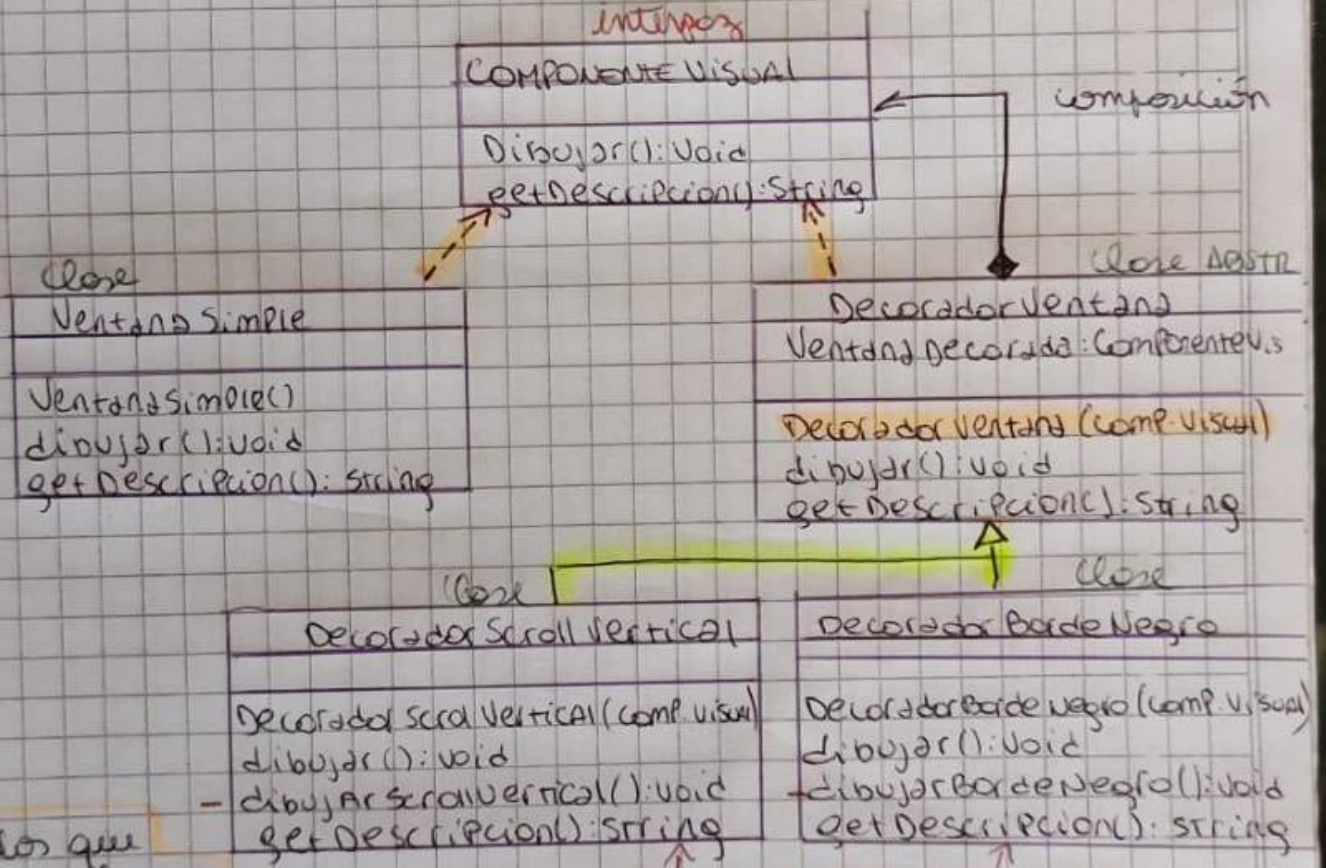
DESVENTAJAS:

1. MUCHOS OBJETOS PEQUEÑOS

Un diseño que use el patrón Decorator suele dar como resultado un sistema formado por muchos objetos pequeños muy parecidos. Son fáciles de adaptar pero pueden ser difíciles de aprender y de depurar

IMPLEMENTACIÓN

- Los clases componentes son interfaces
- La definición de cómo se representan los datos debería delegarse en las subclases
- El Decorator es como un patrón que cambia la piel del objeto, solo la parte exterior del mismo.



clase ABSTRACTA PEOPLE = porque 00S941 Clase 4