

Observer

Patrón de
comportamiento

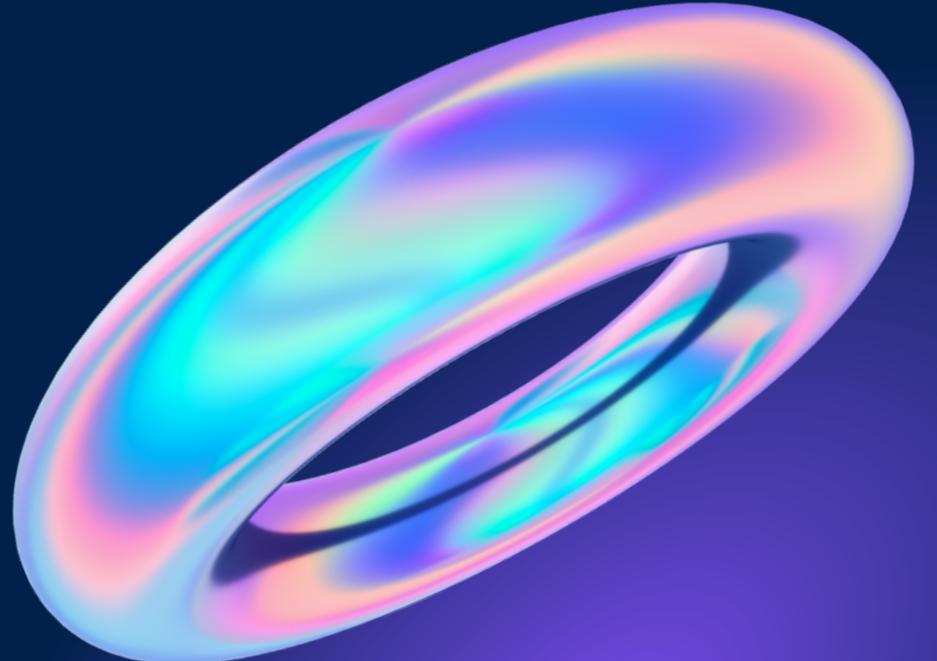
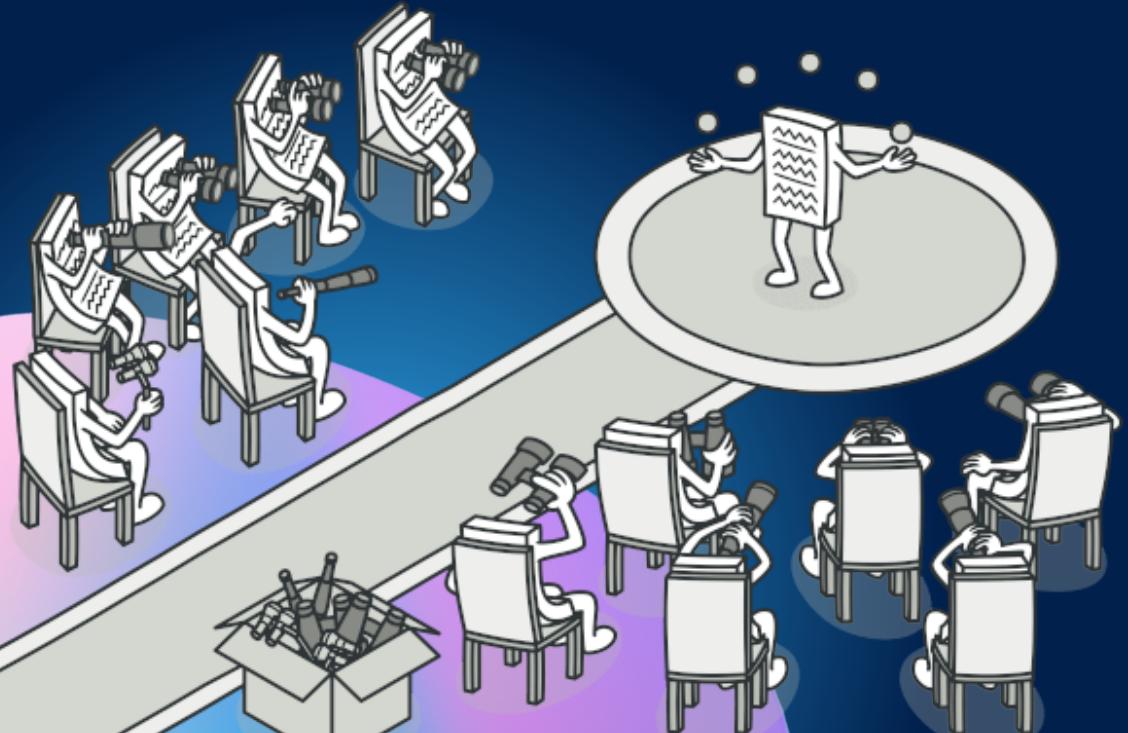
Integrantes

Fabio Sanabria Valerín C07194
Esteban Iglesias Vargas C03913



Problema

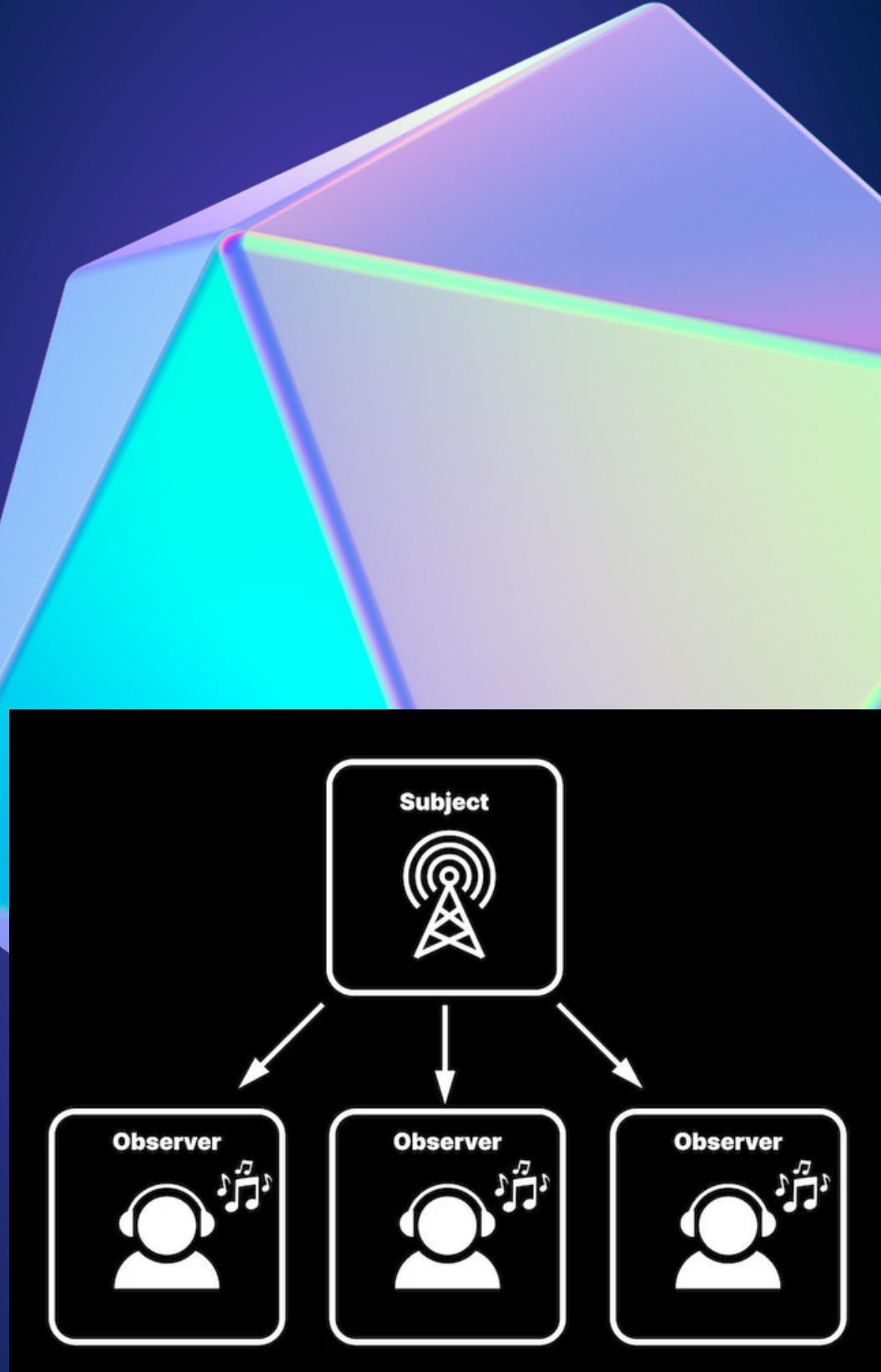
¿Cómo resolverían el problema de las notificaciones?



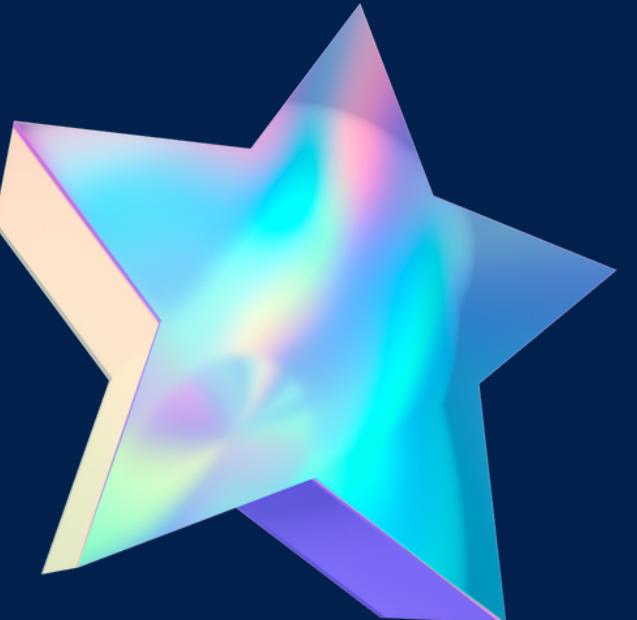
SUBSCRIBE

Concepto

El patrón Observer es un patrón de diseño de comportamiento que define una relación de uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, este notifica y actualiza automáticamente a todos los objetos que dependen de él.



Partes del patrón Observer



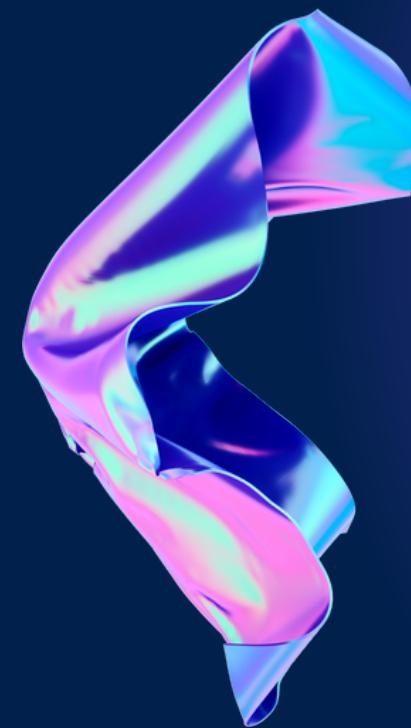
Observable

Sujeto que es
observado



Observador

Mira al Sujeto



Obs. Concreto

Implementa al
observador

Partes del patrón Observer



Sujeto u Observable

- Es un objeto o sujeto que será observado
- Mantiene una lista de los observadores
- Notifica cambios de estado a los observadores
- Hay una o varias clases concretas que lo implementan



Observador

- Interfaz o clase que define a los observadores
- Mantiene el método update para la notificación

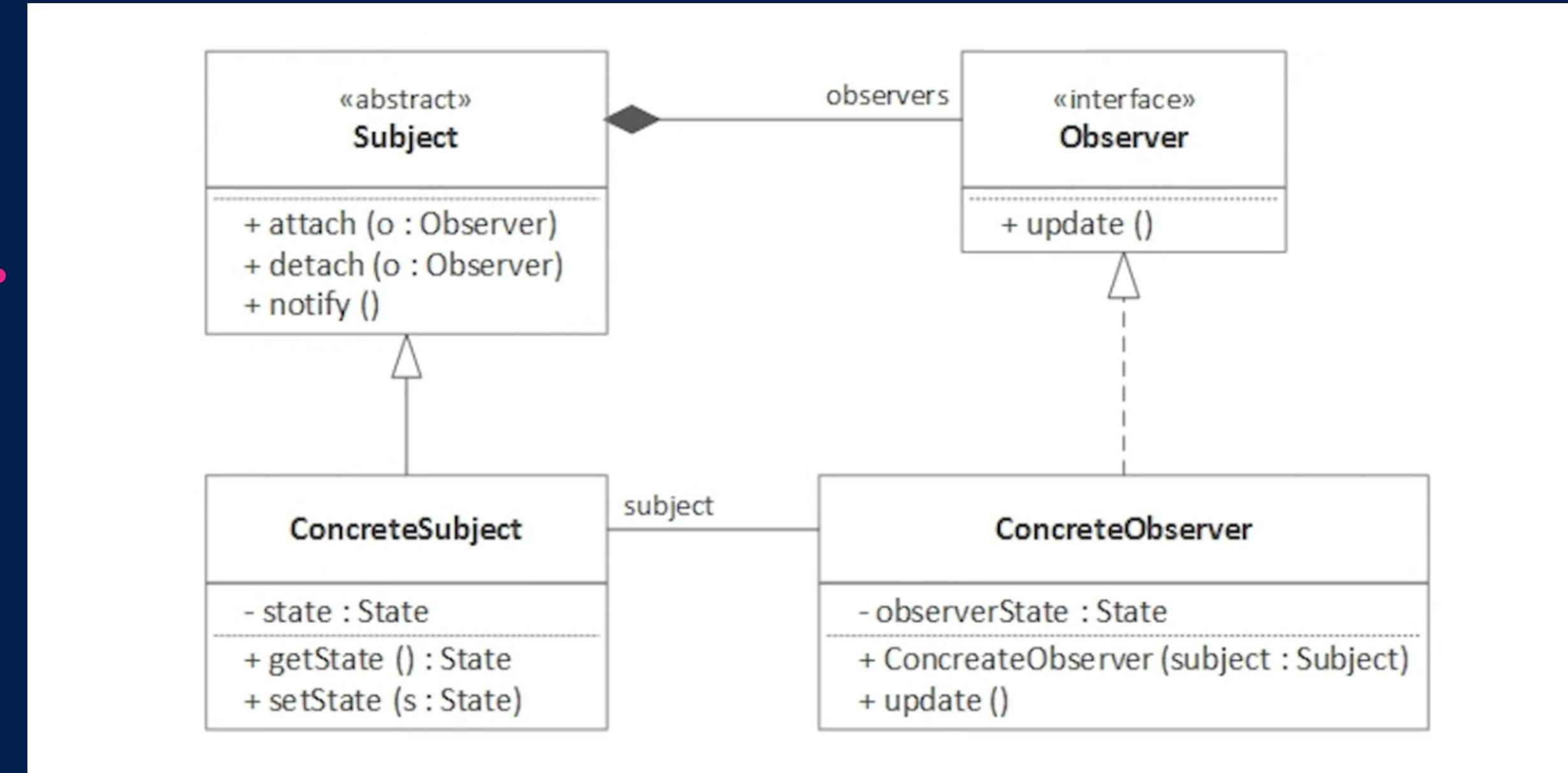
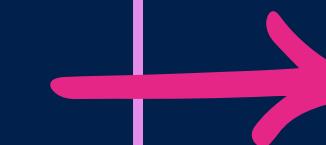


Observador Concreto

- Implementa la interfaz del observer
- Mantiene referencia del sujeto y puede acceder al cambio
- Realiza cambios de acuerdo a la notificación

DIAGRAMA DEL PATRÓN OBSERVER

Diagrama que especifica los diferentes componentes del patrón Observer y como estos interactúan



Ejemplo donde se puede aplicar el patrón Observer

Se desea implementar una funcionalidad en Mediación Virtual donde cada vez que un profesor añada una tarea en un curso donde el estudiante esté inscrito, el sistema le notifique a este que el profesor subió una nueva asignación. Además, si el estudiante retira el curso o es expulsado, no podrá recibir las notificaciones del curso

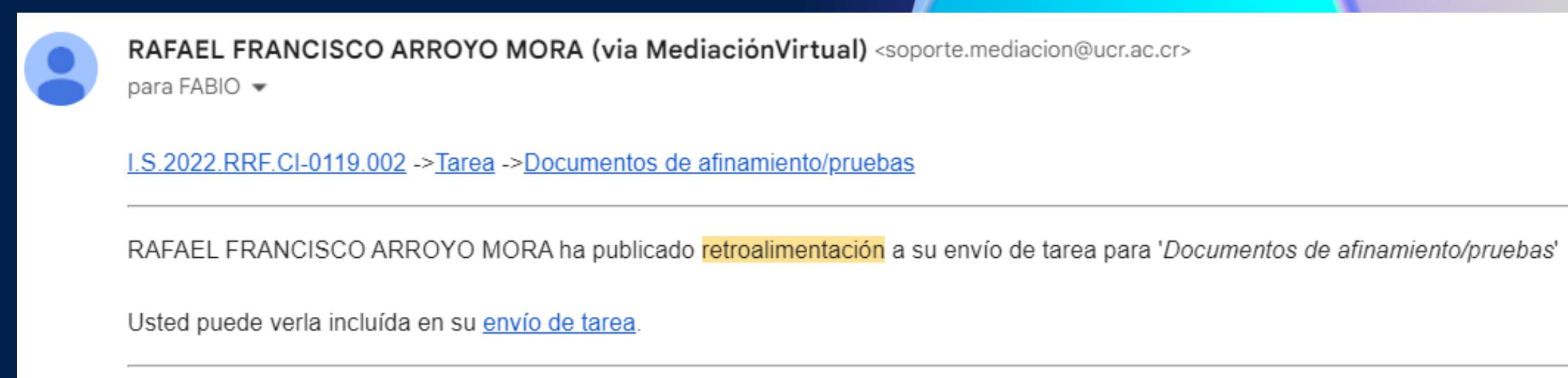
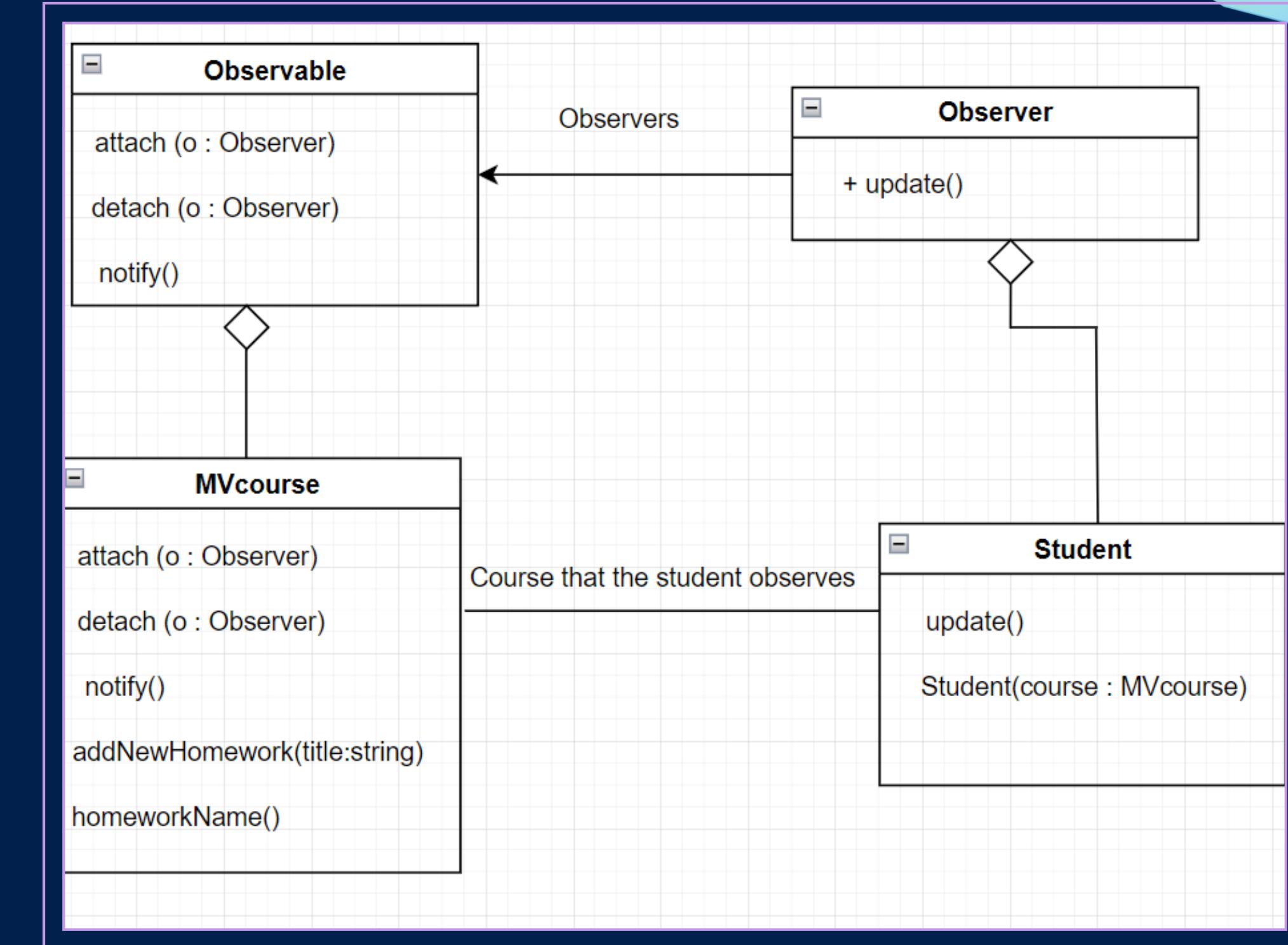


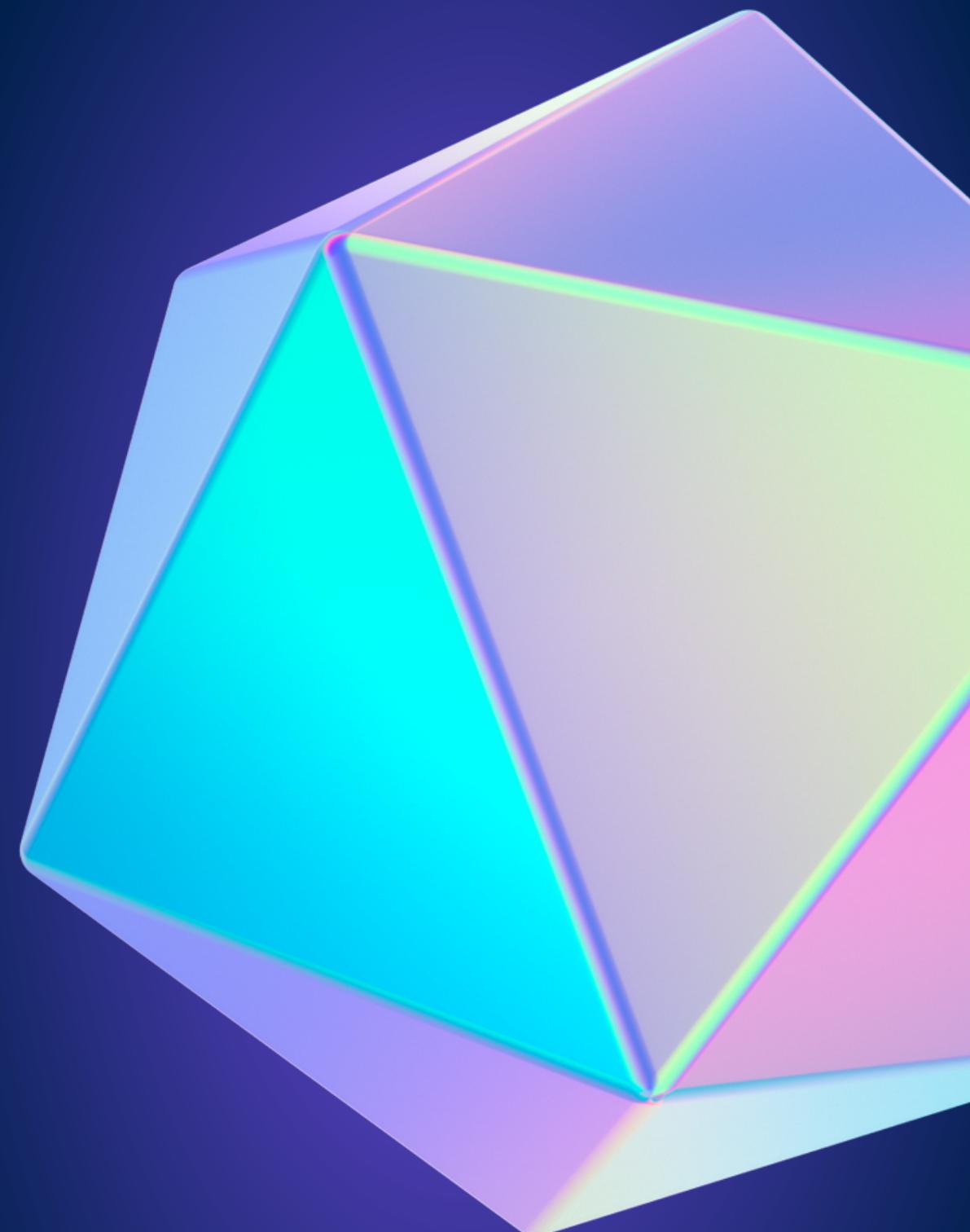
DIAGRAMA DEL EJEMPLO DE MEDIACION VIRTUAL

Diagrama que especifica
como interactuan los
diversos elementos





¿Cómo implementarlo en código?



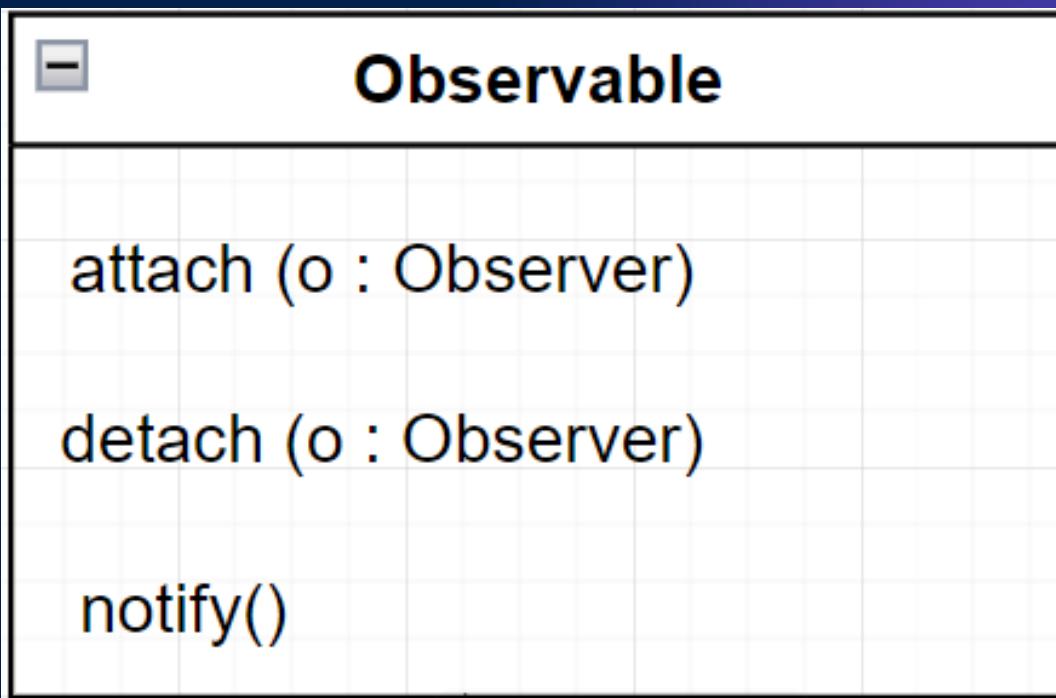
Clase abstracta: Sujeto u Observable

```
# Observable es una clase abstracta
class Observable:
    # Constructor de la clase
    def __init__(self):
        pass
        # self.studentsInCourse = []

    # Metodo para añadir observador
    def attach(self, o):
        pass

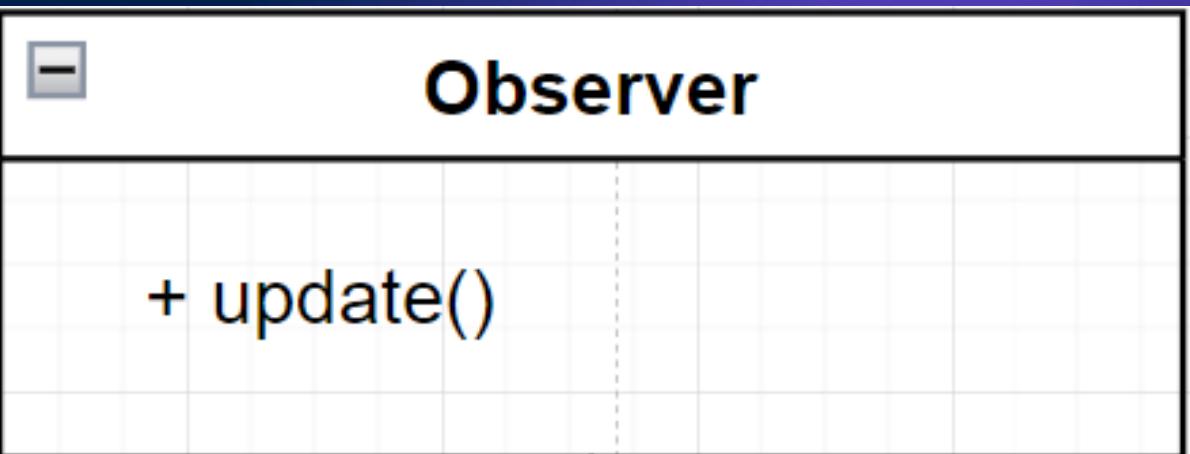
    # Metodo para eliminar observador
    def detach(self, o):
        pass

    # Metodo para notificar a todos los observadores
    def notify(self):
        pass
```



Clase abstracta: Observadores

```
# Observer es una clase abstracta
class Observer:
    def update(self):
        pass
```



Clase Concreta: MVcourse (Sujeto)

```
class MVcourse(Observable):
    # Constructor de la clase MVcourse
    def __init__(self):
        self.studentsInCourse = []
        self.homeworkPosted = ""

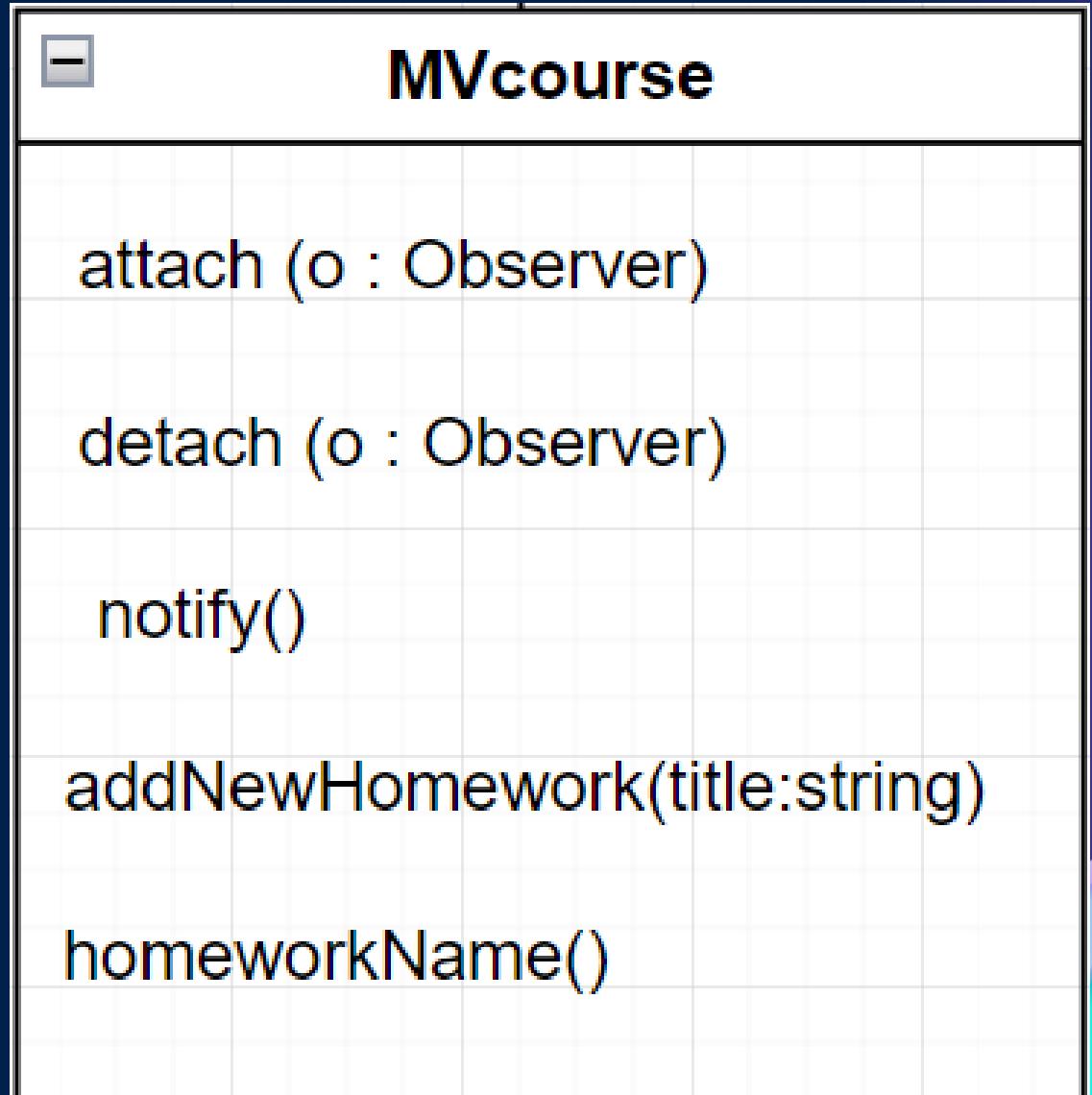
    # Metodo para agregar a un observador
    def attach(self, o):
        self.studentsInCourse.append(o)

    # Metodo para quitar a un observador
    def detach(self, o):
        if o in self.studentsInCourse:
            self.studentsInCourse.remove(o)

    # Metodo que añade un titulo a la tarea y llama a notify
    # cada vez que se
    def addNewHomework(self, title):
        self.homeworkPosted = "Nombre: " + title
        print("El profesor ha añadido la tarea a mediación virtual🌐\n\n")
        self.notify()

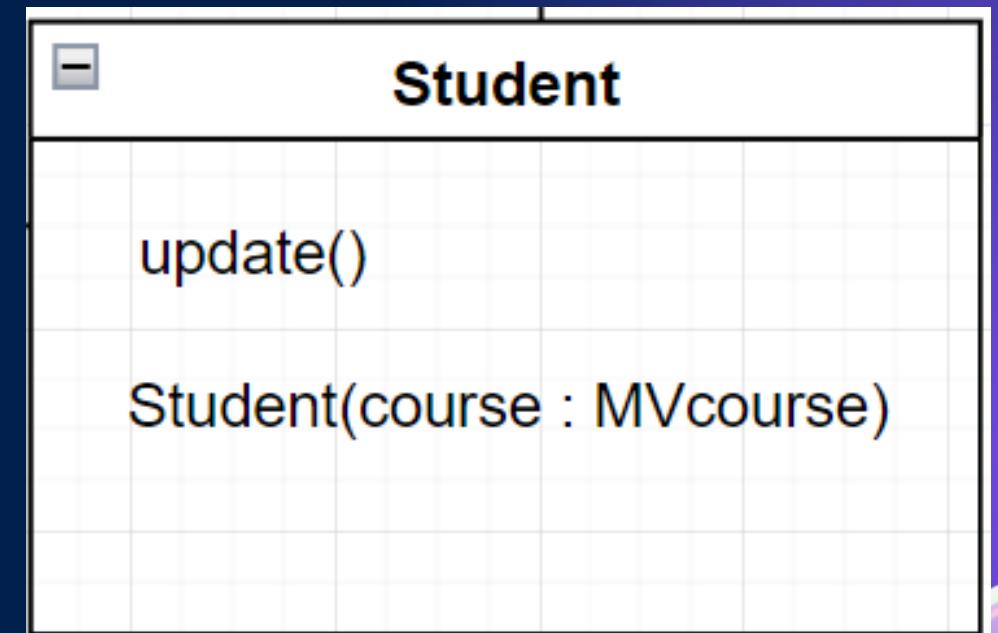
    # Retorna el nombre de la tarea subida
    def homeworkName(self):
        return self.homeworkPosted

    # Este metodo es el encargado de llamar al metodo update de cada estudiante de
    # la cola para notificarles sobre la actualizacion de la estructura del observable
    def notify(self):
        for Student in self.studentsInCourse:
            Student.update()
```



Clase Concreta: Students(Observadores)

```
# Clase concreta del observer, este es el estudiante
class Student(Observer):
    # Constructor, recibe un observable o
    # sujeto para observarlo
    def __init__(self, observable):
        self.observable = observable
    # Metodo que es llamado por observable en
    # alguna parte para notificar al estudiante
    def update(self):
        print("Entimado estudiante, el profesor a agregado una tarea")
        print(self.observable.homeworkName())
```



Main

```
def main():
    # Curso en mediacion virtual
    course = MVcourse()

    # Creado observadores, estos estaran pendientes a las tareas
    # que se suban a mediacion virtual
    student1 = Student(course)
    student2 = Student(course)
    student3 = Student(course)

    # El curso añade a los estudiantes a su cola para notificarlos
    course.attach(student1)
    course.attach(student2)
    course.attach(student3)

    # Se elimina a un estudiante de la cola del curso
    course.detach(student2)

    # Se añaden nuevas tareas al curso de mediacion virtual
    course.addNewHomework('Trabajo de investigación: Patrones de software II📝\n')
    course.addNewHomework('Laboratorio: Pátrones estructurales📝\n')
    course.addNewHomework('Proyecto: Etapa 3📝\n')

if __name__ == '__main__':
    main()
```

Salida esperada

El profesor ha añadido la tarea a mediación virtual 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Trabajo de investigación: Patrones de software II 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Trabajo de investigación: Patrones de software II 

El profesor ha añadido la tarea a mediación virtual 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Laboratorio: Pátrones estructurales 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Laboratorio: Pátrones estructurales 

El profesor ha añadido la tarea a mediación virtual 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Proyecto: Etapa 3 

Entimado estudiante, el profesor ha agregado una tarea 

Nombre: Proyecto: Etapa 3 

Ventajas

- Desacoplamiento
- Fomenta el cumplimiento del principio SOLID
- Fomenta el cumplimiento del principio KISS
- Actualización en tiempo real

Trade-off

- Complejidad adicional
- Sobrecarga de rendimiento
- Posible falta de control de secuencia

Patrones relacionados



Evitar

- Eliminar las referencias de los sujetos en la lista antes de eliminarlo del todo, esto para evitar problemas de memoria y notificaciones a sujetos no existentes.
- Gestionar bien la lista de observadores para no tener múltiples referencias al mismo sujeto y notificar múltiples veces (indeseadas).

Sugerencias

- Añade que se pueda hacer “Attach” y “Detach” de manera dinámica en tiempo de ejecución, esto permite flexibilidad y extensibilidad.
- Añadir mecanismos de notificaciones específicas, esto para que solo se suscriban a aquellos sujetos que les interesan.
- Realizar agrupación de notificaciones en casos de múltiples cambios en cortos períodos de tiempo.
- Gestionar bien las notificaciones para que solo sean notificaciones relevantes y evitar actualizaciones innecesarias.

Bibliografía



[1] Helm, R., Gamma, E., Vlissides, J., & Johnson, R. (2005). Design Patterns. Addison Wesley

[2] BettaTech. OBSERVER (El PATRÓN que lo ve TODO) | PATRONES de DISEÑO. (26 de septiembre de 2020). Accedido el 31 de mayo de 2023. [Video en línea]. Disponible: hello@reallygreatsite.com

[3] "Observer". Refactoring and Design Patterns. <https://refactoring.guru/design-patterns/observer> (accedido el 31 de mayo de 2023).