TEC | Tecnológico de Costa Rica

Tecnológico de Costa Rica

Documentación

Villanueva Quirós Francisco Javier

Jimenez Calvo Wander José

Gutiérrez Badilla José Julián

Ortiz Jimenez Aaron

Sistema de Gestión de Equipos de Trabajos Guía

IC-6821 Diseño de Software

Tercera Fase Proyecto IS 2024

Introducción

En el marco del curso IC-6821 Diseño de Software, se presenta la tercera fase del proyecto "Sistema de Gestión de Equipos de Trabajo Guía" donde en esta fase se tiene como objetivo principal la incorporación de un patrón estructural y dos patrones de comportamiento en la arquitectura de la aplicación. Esto permitirá ampliar la funcionalidad del sistema para integrar nuevos servicios, mejorando la extensibilidad y mantenibilidad del código.

El proyecto, llevado a cabo por los estudiantes Villanueva Quirós Francisco Javier, Jimenez Calvo Wander José, Gutiérrez Badilla José Julián, y Ortiz Jimenez Aaron, se enfoca en tres áreas clave:

1. **Expansión de la Entidad Estudiante:** Transformar la entidad Estudiante en un perfil de usuario completo que pueda interactuar con el sistema pero que este mismo no altere las entidades existentes, además se incluye la implementación de un buzón de notificaciones exclusivo para los estudiantes.

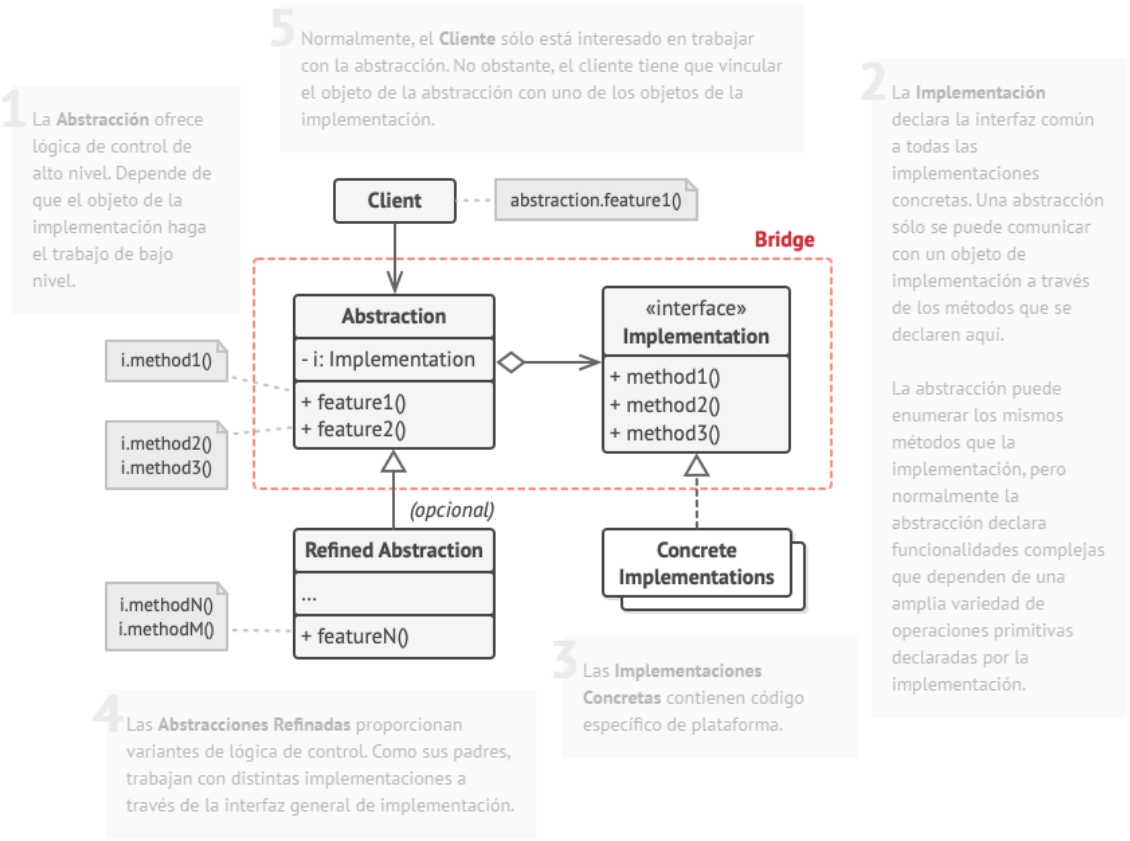
2. **Manejo de Activaciones y Recordatorios de Actividades:** Desarrollar un sistema que maneje automáticamente el cambio de estado de las actividades y envíe recordatorios basados en fechas configuradas, asegurando que las actividades sean notificadas y recordadas correctamente.

3. **Implementación de los Patrones de Diseño:** Integrar los patrones Bridge, Visitor y Observer en la arquitectura del sistema para separar la abstracción de la implementación, permitir la adición de nuevas operaciones sin modificar las clases existentes, y gestionar la mensajería y notificaciones de manera eficiente.

Diagramas Generales

Bridge

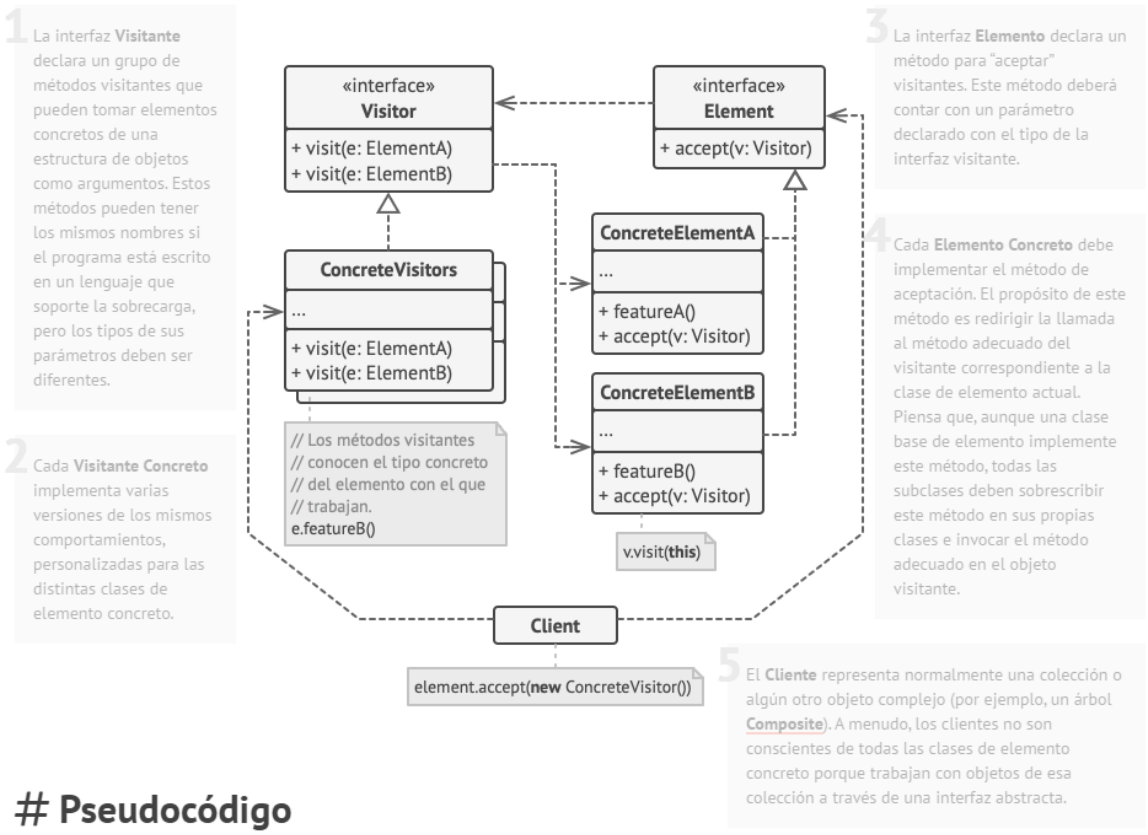
El patrón estructural Bridge es un patrón que se encarga de dividir la abstracción de la clase con respecto a la implementación, esto permite que ambas partes logren evolucionar de forma independiente, por este motivo este patrón es útil cuando tenemos una abstracción que posee múltiples implementaciones y se busca evitar la combinatoria de clases derivadas, en el mundo actual la implementación del patrón Bridge se da en sistemas gráficos donde se desea separar la interfaz con la implementación que realiza, a continuación, se referencia la imagen general del patrón estructural Bridge:



1.0 Generalización del patrón Bridge

Visitor

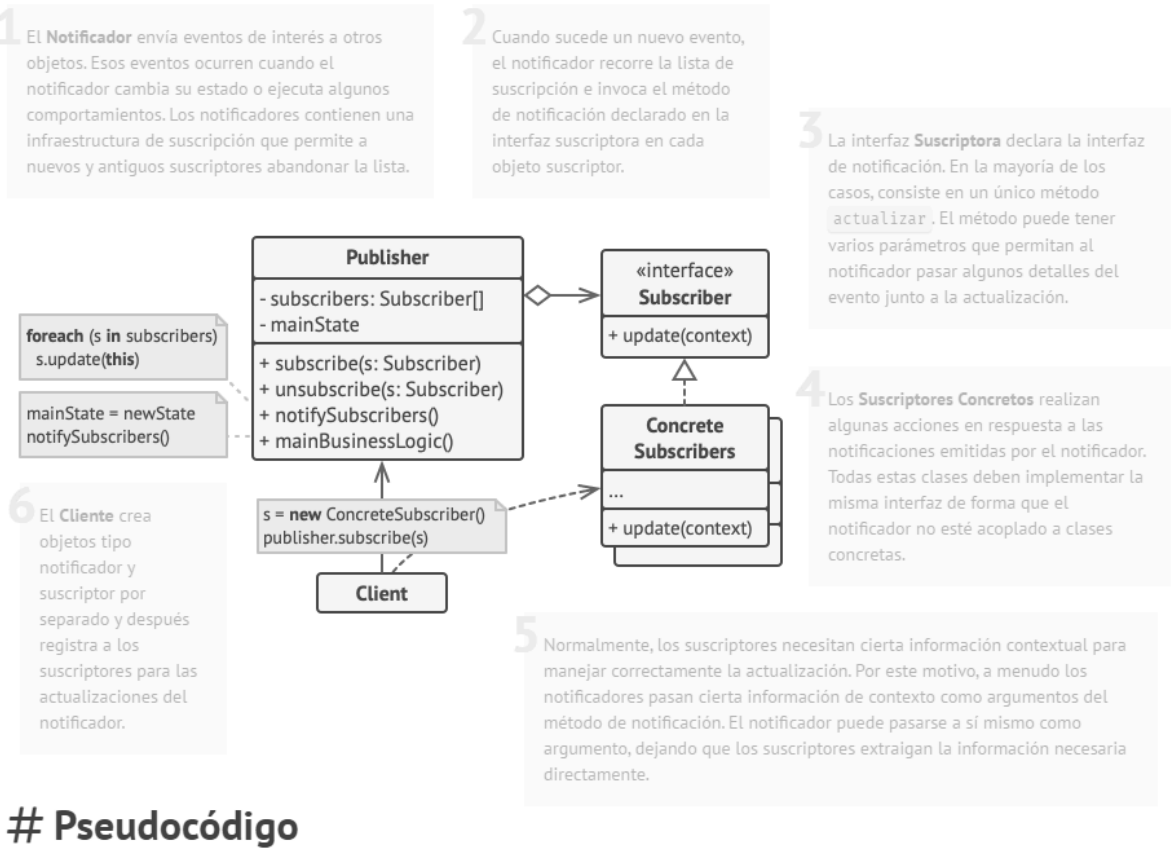
El patrón de comportamiento Visitor posee la característica de permitir agregar operaciones nuevas a estructuras de objetos, donde no se modifica las clases de dichos objetos en los cuales opera, este tiene dos componentes claves para su uso, los cuales son el visitado, que como su nombre indica son los elementos a los cuales se puede acceder, mientras que su otra parte serían los visitantes, que son aquellos que implementan las operaciones, se utiliza mucho para la optimización y generación de código más que todo sobre estructuras basadas en árboles, a continuación, se presenta la imagen general del patrón Visitor:



2.0 Generalización del patrón Visitor

Observer

El patrón de comportamiento Observer es uno que define una dependencia de uno a muchos entre los objetos en los cuales se trabaja para que de esta manera cuando un objeto busque un cambio de estado todas aquellas dependencias de este mismo sean notificadas y a su vez actualizadas de manera automática, este sistema es ideal para implementar en sistemas de eventos y el uso de notificaciones, a continuación, se referencia la imagen general del patrón Observer:



3.0 Generalización del patrón Observer

Patrón Bridge en la Arquitectura

Justificación del Uso del Patrón Bridge

- Abstracción e Implementación Separadas

En nuestra implementación, la clase Usuario actúa como la abstracción que contiene una referencia a TipoUsuario, la cual es una interfaz mientras que las clases concretas (Estudiante, AsistenteAdministrativo, Profesor) implementan la interfaz TipoUsuario, de esta manera se aplica el término general del patrón Bridge, donde nos buscamos enfocar en que la clase de abstracción que posee múltiples implementaciones evite la combinatoria de sus propias derivadas.

- Extensibilidad

En nuestro caso, al utilizar el patrón Bridge permitimos agregar nuevas clases de usuarios sin modificar como tal la clase Usuario, por lo tanto, permitimos facilitar tanto la extensibilidad como el mantenimiento general del código.

- Independencia de la Implementación

La abstracción (Usuario) y su implementación (TipoUsuario y sus clases concretas) pueden evolucionar de manera independiente, lo cual como se comentaba en la sección de patrones generales, es un objetivo central del patrón Bridge.

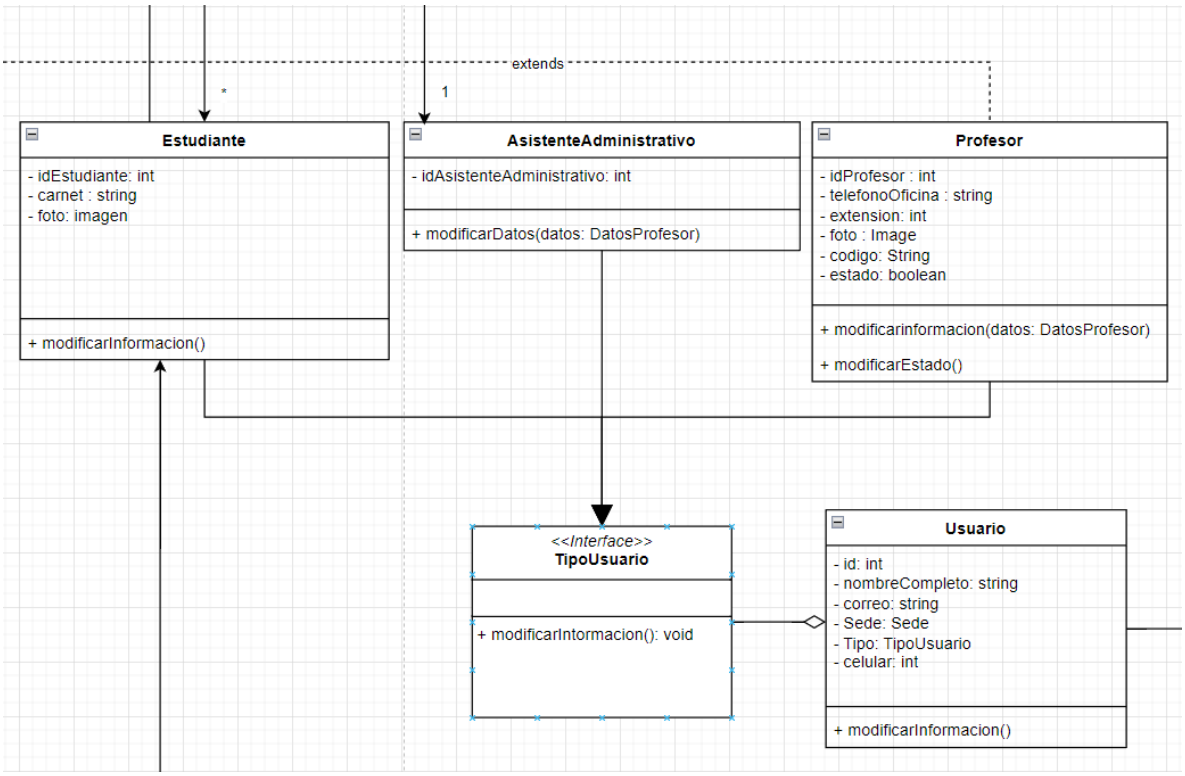
Partes de la Arquitectura que Generan el Uso del Bridge

- **Abstracción (Usuario):** Es el que contiene tanto la lógica de control de alto nivel como una referencia directa a la interfaz TipoUsuario.

- **Interfaz (TipoUsuario):** Aquí se definen los métodos que deben implementarse por las clases concretas implementadas en nuestro caso.

- **Implementaciones Concretas:** Son aquellas que proporcionan la implementación específica de los métodos definidos en la interfaz TipoUsuario.

Imagen de la implementación del Patrón Bridge



4.0 implementación del patrón Bridge

El uso del patrón Bridge en este contexto permite una clara separación de la abstracción (Usuario) y su implementación (TipoUsuario y sus clases concretas), promoviendo la flexibilidad y mantenibilidad del sistema.

Incorporación de los Patrones VISITOR y OBSERVER

Patrón Observer

Contextualización: El patrón Observer permite que un objeto notifique a otros objetos sobre cambios en su estado, este es ideal para implementar en sistemas de eventos y notificaciones.

Implementación del Patrón Observer

```
ALTER PROCEDURE [dbo].[Recordatorios](
    @inFechaActual DATE
)
AS
BEGIN
    DECLARE @OutResult INT
BEGIN TRY
    DECLARE @FechasCumple TABLE(
        idActividad INT,
        fechaRecordatorio1 DATE,
        fechaRealizacion DATE,
        Estado VARCHAR(32),
        siguienteRecorda DATE
    );
    INSERT INTO @FechasCumple(idActividad, fechaRecordatorio1, fechaRealizacion,
    Estado, siguienteRecorda)
    SELECT
        A.id,
        A.fechaRecordatorio,
        A.fechaRealizacion,
        E.nombre,
        DATEADD(DD, A.frecuenciaRecor, A.fechaRecordatorio)
    FROM dbo.Actividades A
    INNER JOIN dbo.Estado E ON E.id = A.idEstado
    WHERE (A.fechaRecordatorio = @inFechaActual) AND (E.nombre = 'NOTIFICADA' OR E.nombre = 'PLANEADA' )
```

5.0 Trigger para Notificaciones de Cancelación

Se utiliza un trigger en la tabla Actividades para notificar a los usuarios cuando una actividad cambia de estado a "CANCELADA", a continuación, se describen los elementos del patrón Observer:

Sujeto (Publisher): La tabla Actividades.

Observadores (Subscribers): Los usuarios que deben ser notificados sobre el cambio de estado.

Evento (Event): La actualización del estado de una actividad.

Notificación (Notify): Inserción en la tabla Notificaciones para alertar a los usuarios.

```
BEGIN TRANSACTION InsertNotificaciones

INSERT INTO dbo.Notificaciones(idUsuario, idActividad, fecha,ifLeido)
SELECT
    U.id,
    C.idActividad,
    GETDATE(),
    0
FROM @FechasCumple C, dbo.Usuario U
WHERE U.idTipo = 2

UPDATE dbo.Actividades
SET fechaRecordatorio = CASE
                            WHEN S.siguienteRecorda < S.fechaRealizacion THEN S.siguienteRecorda
                            ELSE fechaRecordatorio
                        END,
    idEstado = CASE
                    WHEN S.Estado = 'PLANEADA' THEN (SELECT id FROM dbo.Estado WHERE nombre = 'NOTIFICADA')
                    ELSE idEstado
                END
FROM @FechasCumple S
WHERE id = S.idActividad
```

6.0 Notificaciones de Recordatorios

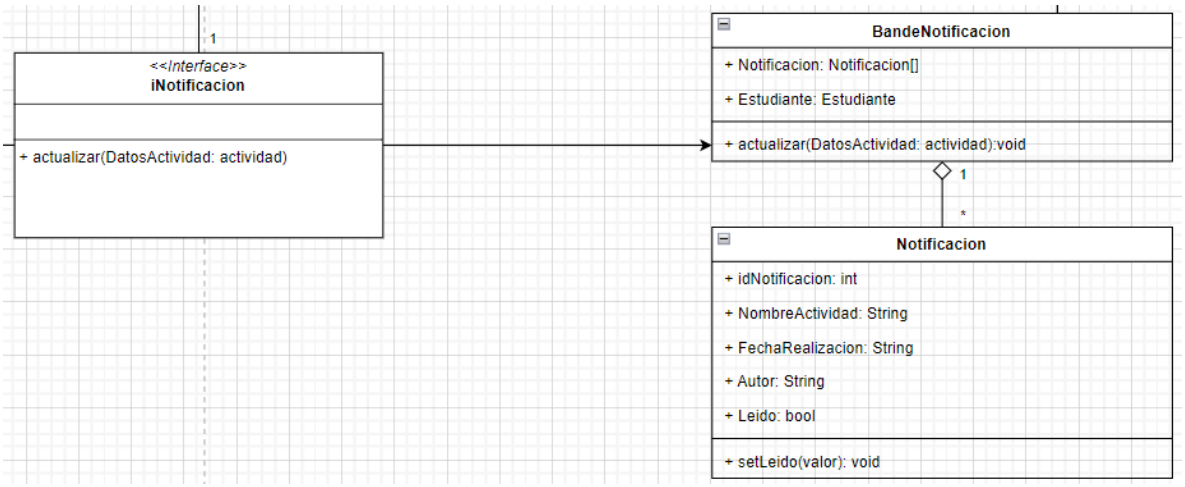
Se utiliza un procedimiento almacenado que envía recordatorios y publica notificaciones, se presentan los siguientes elementos del patrón Observer:

Sujeto (Publisher): La tabla Actividades y el procedimiento almacenado.

Observadores (Subscribers): Los usuarios que deben ser notificados de los recordatorios.

Evento (Event): La fecha del recordatorio.

Notificación (Notify): Inserción en la tabla Notificaciones.



7.0 Diagrama de Observer

En este contexto podemos ver el uso extensivo del patrón Observer, este mismo permite gestionar entre las notificaciones de las actividades y asegurar que los estudiantes sean informados sobre cambios relevantes.

Patrón Visitor

Contextualización: El patrón Visitor permite definir una operación nueva sin cambiar las clases de los elementos sobre los que opera, ideal para la para la optimización y generación de código.

Implementación del Patrón Visitor

```
USE [GETG]
GO
/****** Object:  Trigger [dbo].[NuevoEstado]    Script Date: 6/13/2024 2:41:07 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[NuevoEstado]
ON [dbo].[Actividades]
AFTER UPDATE
AS
BEGIN
    DECLARE @Estado VARCHAR(32);
    SET @Estado = 'PLANEADA'
    IF EXISTS(SELECT 1 FROM inserted I INNER JOIN dbo.Estado E ON I.idEstado = E.id
    WHERE E.nombre = 'CANCELADA')
    BEGIN
        INSERT INTO dbo.Notificaciones(idActividad, idUsuario, ifLeido, fecha)
        SELECT
            i.id,
            U.id,
            0,
            GETDATE()
        FROM inserted i, dbo.Usuario U
        WHERE U.idTipo = 2
    END
END
```

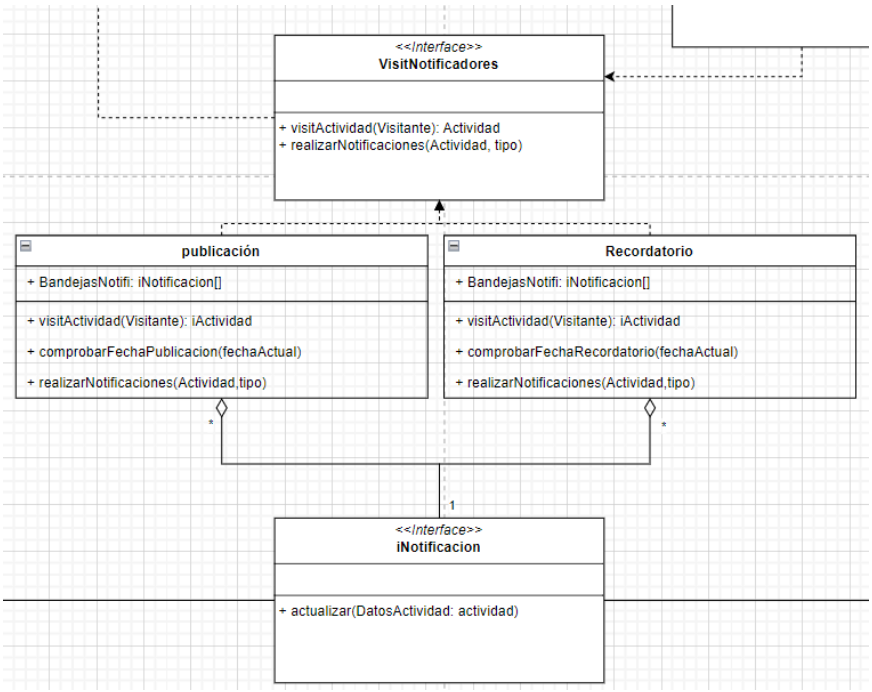
8.0 Procedimiento para Visitar Datos

Se utiliza un procedimiento almacenado que visita los datos de actividades basándose en el estado de la actividad presente, a continuación, se presentan los siguientes elementos del patrón Visitor:

Visitante (Visitor): El código que visita las actividades para procesarlas.

Elemento (Element): Las actividades con diferentes estados (PLANEADA, NOTIFICADA).

Método accept: Lógica de selección y procesamiento basada en el estado.



9.0 Diagrama de Recordatorio y Publicación

La interfaz VisitNotificadores se encarga de definir los métodos que deben implementar los visitantes, por otro lado, las clases concretas (publicación y Recordatorio representan las operaciones específicas que se realizan cuando se visita una actividad donde a su vez los métodos de cada una de estas implementan tanto métodos de comportamiento, como métodos específicos de las mismas.

Diagrama de clases Actualizado

A continuación, se presenta en un hipervínculo el diagrama de clases con las actualizaciones tanto de las implementaciones del patrón de comportamiento Visitor descritas anteriormente como las del patrón de comportamiento Observer además de la implementación solicitado acerca del patrón estructural, que, en nuestro caso, fue implementado con el método Bridge

Diagrama de Clases - Grupo 4

A parte de las actualizaciones de dichos patrones comentados anteriormente, no hubo cambios significativos en el estado del diagrama, por este motivo, solamente se simplificó la lectura de los componentes ya existentes y se buscó adherir los patrones de comportamiento y estructurales nuevos.

Cuadro de Análisis de Resultados

A continuación, se presenta un cuadro de análisis de resultados donde el equipo valora en un rango de 0-100 todas y cada una de las funcionalidades solicitadas en esta fase, además se le suma una justificación en caso de que la completitud no haya sido total:

Funcionalidad	Valoración (0-100)	Justificación
Expansión de la entidad Estudiante	100%	
Servicio de notificaciones asociado al estudiante	95%	No se borran las notificaciones ya leídas.
Manejo de activaciones y recordatorios	100%	
Implementación del patrón Visitor	100%	
Implementación del patrón Observer	100%	
implementación del patrón estructural	100%	

Lecciones Aprendidas

A continuación, se presenta un cuadro de lecciones aprendidas individual en el proceso de construcción del proyecto tanto en el plano del manejo de las tecnologías utilizadas, el contenido del curso puesto en práctica, la experiencia del trabajo en equipo:

Estudiante	Retroalimentación
Jimenez Calvo Wander	Con respecto al desarrollo de esta fase, hemos tenido la oportunidad de trabajar y profundizar en diferentes patrones de diseño fundamentales, tanto el visitor como el observer que eran requisito directo como la implementación a nuestro parecer del bridge ha jugado un papel muy importante para mejorar varios aspectos del código que ya teníamos, estos aspectos como la flexibilidad y la mantenibilidad de nuestra aplicación me ha permitido apreciar la necesidad de estructurar bien el código antes de programar y como esto será beneficioso en futuros proyectos.
Gutiérrez Badilla José Julián	A lo largo del desarrollo del proyecto, se hizo uso de tecnologías cloud como AWS para el manejo de APIs y bases de datos. Esto fue una curva de aprendizaje sumamente elevada y provechosa. Asimismo, el uso de patrones de diseño simplifican el código, y a su vez, lo hacen más legible para el trabajo en equipo.
Ortiz Jimenez Aaron	Gracias a los patrones mostrados en clases y las diferentes prácticas realizadas, nos facilitó su uso en el diseño de la evolución del proyecto,

	incluso, permitió realizar pocos cambios en ciertos elementos ya implementados gracias a estos. Por lo que puedo concluir la importancia de los patrones de diseño en reducir la complejidad significativamente en la evolución de la aplicación si los empleamos correctamente según el problema a resolver, e incluso favorece su reúso en otros proyectos con elementos similares.
Villanueva Quirós Francisco	Para esta fase, se implementaron muchos conocimientos de la fase anterior. Sin embargo, se añadió el uso de los patrones de observer y visitor que nos ayudan a simplificar las tareas. Por lo que, para esta fase se utilizaron estos dos patrones con profundidad para finalizar la entrega y se reforzaron los conocimientos anteriores.