Ingeniería Informática

Programación Orientada a Objetos

TP: Entrega 3

Autores:
Manuel Lopez Cosmitz
Hugo Elio Marchesin

5 de junio de 2025

Índice

Enunciado

1. Consigna

1.1. Idea general

El trabajo consiste en realizar un sistema en el cual los usuarios puedan:

- Registrarse/Iniciar sesión.
- Tener una lista de contenidos (libros, películas, etc.) propia.
- Agregar/quitar contenidos de dicha lista.
- Puntuar contenidos concretos.
- Agregar/quitar amigos.
- Obtener recomendaciones de amigos/contenidos.

1.2. Cosas a tener en cuenta

Según lo planteado en el enunciado, más allá de la parte algorítmica o las distintas interacciones, lo más relevante es la interacción, comportamiento y responsabilidades de los distintos objetos planteados.

Clases del sistema

2. Entidades

2.1. Contenidos

Dado que se va a trabajar con distintos tipos de contenidos, y que muchos de ellos tienen comportamiento y características similares, optamos por partir de una clase abstracta `Contenido`.

```
public abstract class Contenido {
    private String nombre;
    private String tipo;
    private List<Float> calificaciones = new ArrayList<>();
    // ...
}
```

Listing 1: Vistazo de la clase abstracta.

Para posteriormente, extender la misma para suplir la funcionalidad necesaria, por ejemplo en el caso de las películas:

```
public class Pelicula extends Contenido {
    private float duracion;

    public Pelicula(String nombre, String tipo, String plataforma, LocalDateTime
    fechaCreacion, float duracion) {
        super(nombre, tipo, plataforma, fechaCreacion);
        this.duracion = duracion;
    }
    // ...
}
```

Listing 2: Vistazo de una implementación de la clase abstracta.

2.2. Clientes

Para el caso de los clientes, optamos por encapsular toda la lógica en una única clase, ya que solamente contamos con un único tipo.

```
public class Cliente {
    private int id;
    private String user;
    private String password;
    private List<Cliente> friends;
    // ...
}
```

Listing 3: Vistazo de la implementación del cliente.

2.3. Entidades de sistema

Dado que en la ejecución real de la aplicación se va a trabajar con un conjunto de clientes/contenidos, optamos por definir una serie de clases encargadas de gestionar distintos comportamientos pertinentes a la lógica de ejecución.

- `GestorClientes`: esta clase puntualmente se encarga del manejo de la lista de clientes, tareas como el registro, la dada de baja, así como la búsqueda de un cliente concreto y otras tareas similares.
- `PlataformaCentral`: esta clase sería el equivalente de la clase previa, pero referida a los contenidos, se encarga de todas las tareas generales pertinentes a la búsqueda o listado de distintos tipos de contenidos.

- `AdministradorDeContenido`: otra clase encargada del manejo de contenidos, esta puntualmente maneja toda tarea referida a la manipulación de los mismos, ya sea en el caso de darles una puntuación como también agregar/quitar nuevos.

2.4. Entidades de ejecución

Teniendo en cuenta que la simulación de la ejecución de la aplicación la hacemos en consola, creamos una clase que se encargue puntualmente de esta tarea, facilitando tareas como el uso de las distintas entidades de sistema.

Lo pensamos de manera tal que, en caso de que eventualmente se quiera pasar a una implementación con interfaz gráfica, bastaría con crear otra de estas clases, sin necesidad de afectar a los otros componentes del sistema.

```
public final class App {
    public static String user = "";
    public static Scanner scanner = new Scanner(System.in);
    public final static AdministradorDeContenido adminContenido = new
    AdministradorDeContenido(new PlataformaCentral());
    public final static GestorClientes gestorClientes = new GestorClientes();

    public static Contenido crearContenido() {}
    public static Contenido eliminarContenido() {}
    public static boolean iniciarSesion() {}
    public static boolean registrarse() {}
    public static void listarContenidos(char opt) {}
    public static void agregar(char opt) {}
    public static void nuevaWishlist() {}
    public static void calificar() {}
    public static void listarAmigos() {}
    public static void agregarAmigo() {}
    public static void eliminarAmigo() {}

    private static Cliente currentUser() {
        return App.gestorClientes.getCliente(user);
    }
}
```

Listing 4: Vistazo de la clase principal del sistema.

Como se puede ver en el extracto previo, la clase contiene una gran cantidad de métodos referidos a tareas varias, tareas realizadas por las entidades de sistema planteadas anteriormente. Pensamos a esta clase como un intermediario entre ellas, principalmente porque acá manejamos la lógica del ingreso de datos por consola. Esta idea queda más clara si vemos uno de todos esos métodos.

```
public final class App {
    //...
    public static void agregarAmigo() {
        var amigos = currentUser().getFriends();
        System.out.print("Nombre del amigo a agregar: ");
        var nombre = scanner.nextLine();
        var existe = gestorClientes.existe(nombre);
        if (existe.isEmpty()) {
            System.out.println("No existe un usuario con ese nombre.");
        } else if (amigos.stream().anyMatch(c -> c.getUser().equals(nombre))) {
            System.out.println("Ya tenes a esta persona en tu lista de amigos.");
        } else {
        currentUser().agregarAmigo(existe.get());
        }
    }
}
```

Listing 5: Vistazo de uno de los métodos referidos al ingreso de datos.

Otra clase de este tipo sería `Api.java`, la cual creamos para simular al sistema de IA externo que menciona el enunciado. Lo implementamos de manera tal que recomiende usuarios y contenidos de

manera aleatoria, basándose en los valores existentes. Para más detalles consultar la implementación.

2.5. Entidades auxiliares

Estas clases fueron creadas únicamente para facilitar el trabajo, no tienen mapeo alguno con objetos referentes a la aplicación. Un ejemplo de una de estas sería una de las clases creadas para agilizar el manejo de datos que vienen del usuario. La clase en cuestión se llama `ContenidoInput`, se trata de una clase abstracta que generaliza el parseo de datos para los distintos tipos de contenido. Dado que son clases de poca importancia, nos abstenemos de mostrar extractos en este informe.

Ciclo de ejecución

3. Funcionamiento del sistema

En el siguiente apartado vamos a explicar cómo se comportaría nuestro programa en una posible ejecución real. Dado que planteamos una ejecución por consola, vamos a basar el siguiente apartado en la funcionalidad prevista en el main (`Main.java`), la cual se basa en el uso de la clase `App` como entidad principal.

El recorrido por las distintas funcionalidades se va a realizar mediante opciones, un comportamiento esperado dado que estamos trabajando en consola.

3.1. Inicio

Al inicio el programa le brinda al usuario (en esta etapa refiriéndose a usuario/consumidor de la aplicación, no a un usuario registrado en el sistema) una serie de opciones, siendo las más importantes:

- Inicio de sesión.
- Registro de un nuevo usuario.

Algo **fundamental** a tener en cuenta con respecto al resto de opciones:

- Agregar contenido a la plataforma.
- Quitar contenido de la plataforma.

Es el hecho de que nosotros planteamos que los administradores del sistema sean quienes agreguen los contenidos y que los usuarios simplemente hagan uso de ellos, ya sea puntuándolos o agregándolos a su perfil. Tal y como funciona una plataforma como Netflix por ejemplo, los administradores agregan las películas y los usuarios las consumen. La existencia de estas opciones en nuestra ejecución se da básicamente por temas de testing/conveniencia, ya que, como se dijo anteriormente, el usuario no debería crear o quitar contenidos, pero nos es imposible simular dicho comportamiento.

3.2. Usuario iniciado

Cuando el sistema detecta que el usuario ha ingresado, las opciones brindadas pasan a ser otras. Puntualmente:

- Gestionar contenidos: agregar contenidos vistos o a la wishlist, puntuar, etc.
- Gestionar amigos: agregar o quitar amigos.

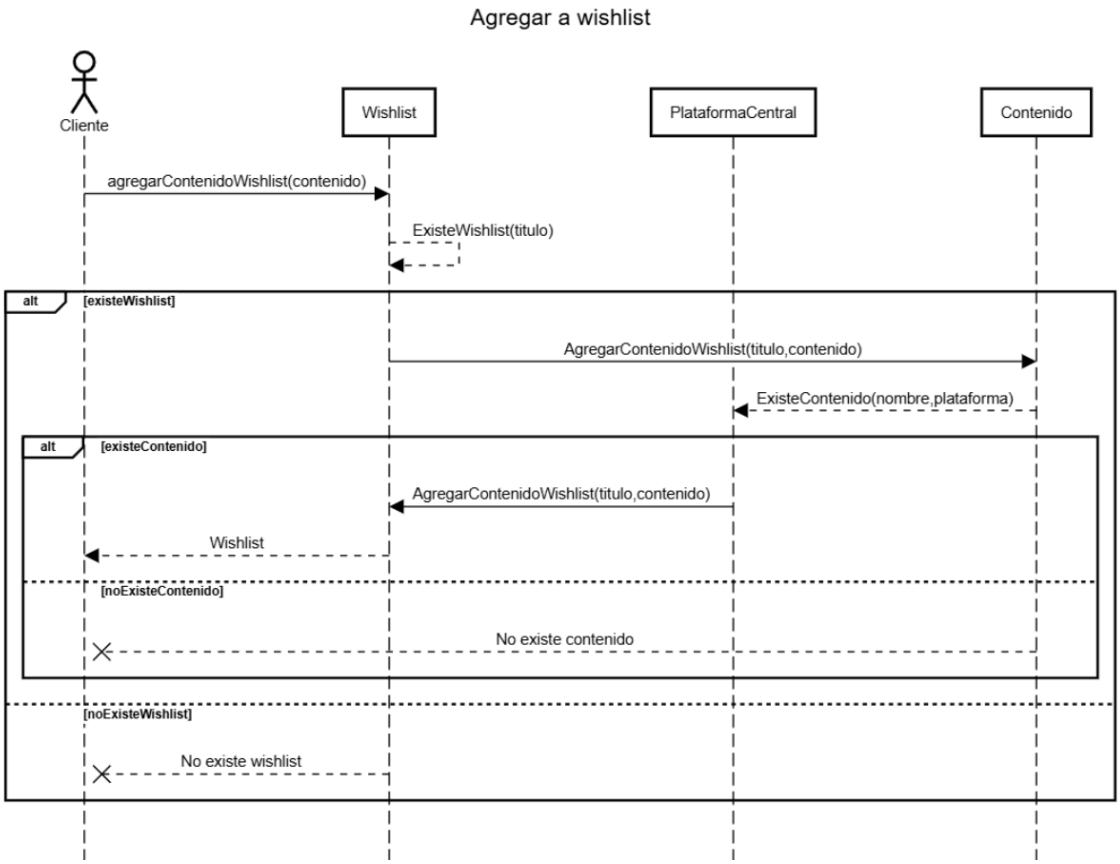
Creemos que la lógica en el main está lo suficientemente clara como para abstenernos de hacer una explicación paso por paso en este informe. Dado que la ejecución sigue fluyendo en distintos prompts de diversas opciones, recomendamos consultar `Main.java` para una vista clara de todo. Concluimos detallando que este flujo de ejecución continua hasta que el usuario cierre sesión/cierre el programa.

Diagramas

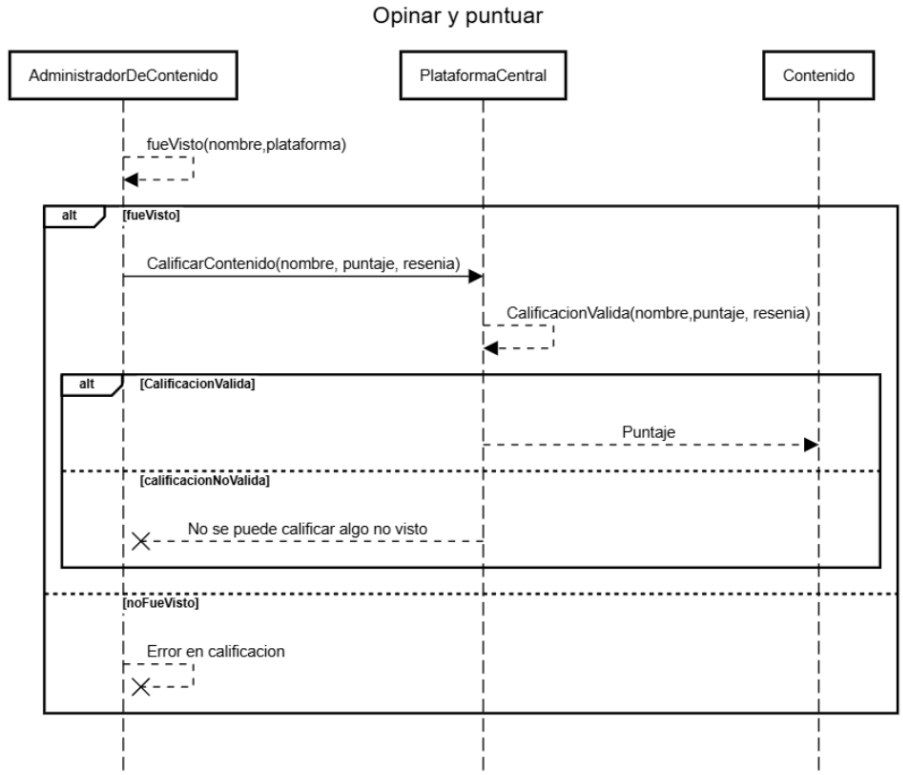
4. Diagramas de secuencia

Importante: en los siguientes diagramas optamos por aislar la clase `App`, dado que se trata de una clase puramente hecha para mostrar una posible ejecución.

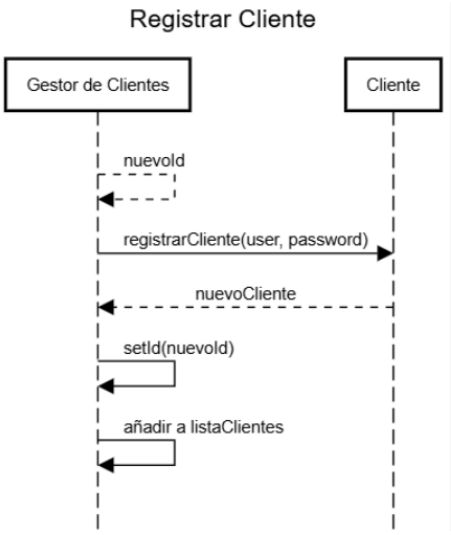
4.1. Agregar a wishlist



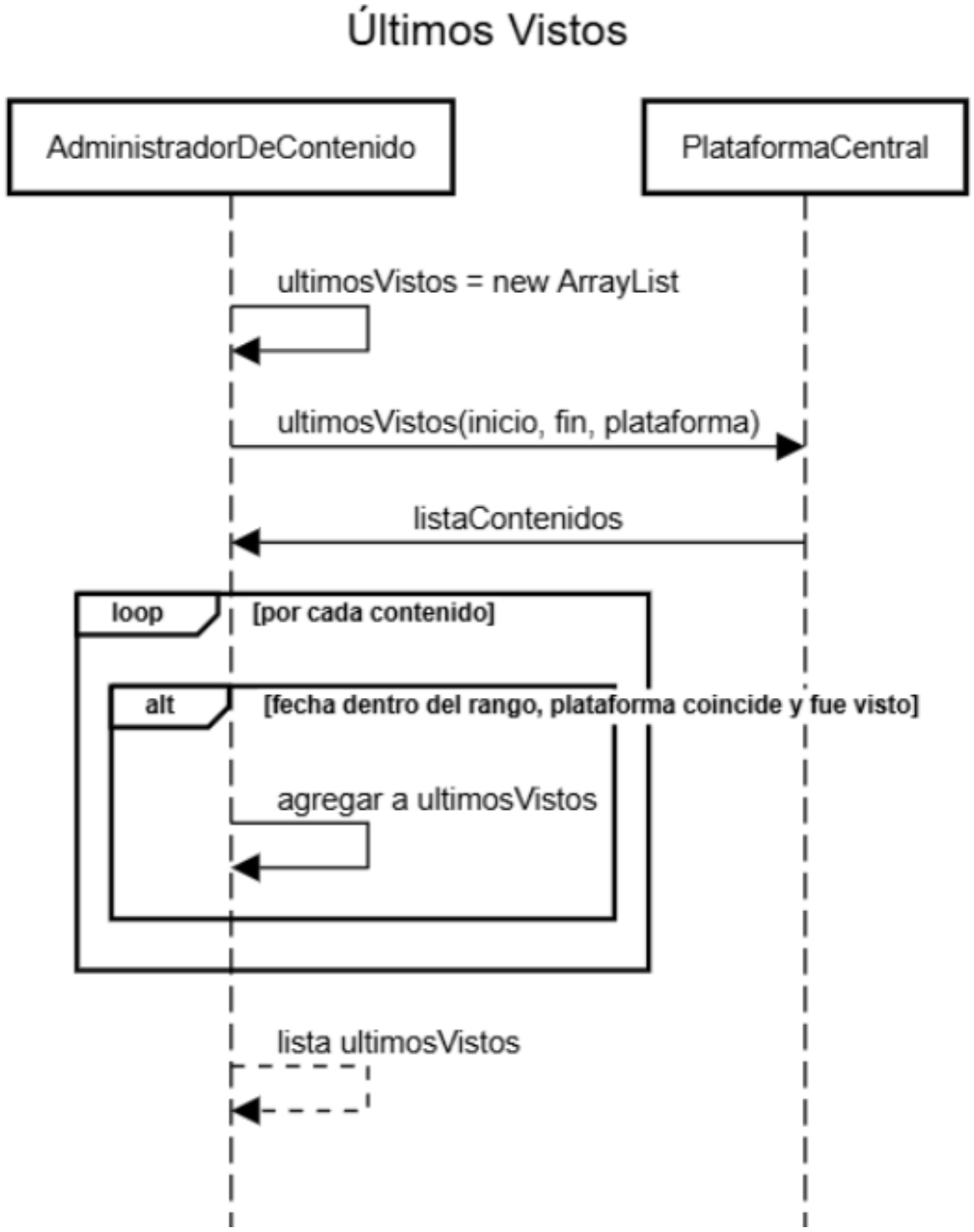
4.2. Opinar y puntuar



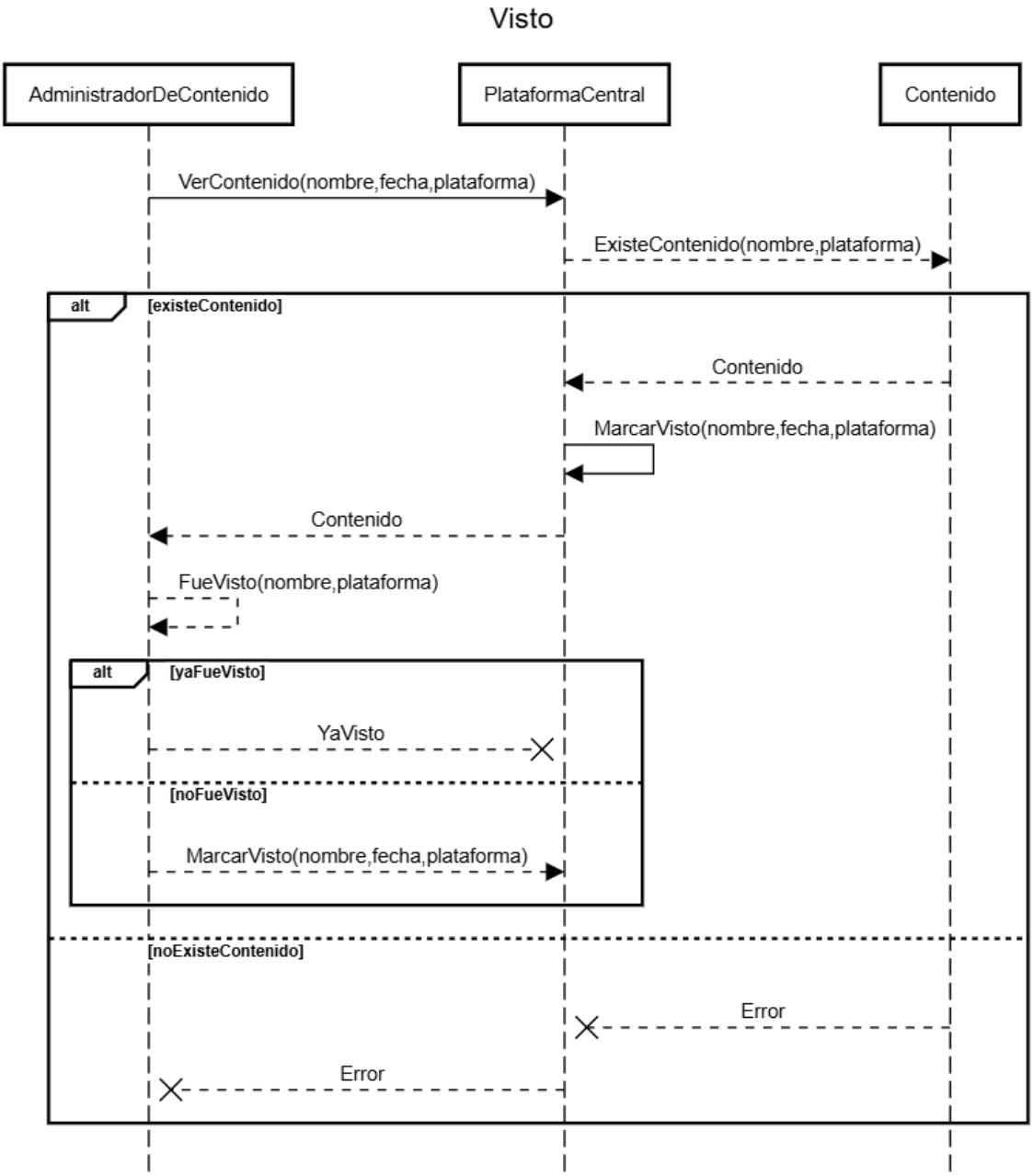
4.3. Registrar cliente



4.4. Últimos vistos



4.5. Vistos


Visto
AdministradorDeContenido
PlataformaCentral
Contenido
VerContenido(nombre,fecha,plataforma)
ExisteContenido(nombre,plataforma)
alt
[existeContenido]
Contenido
MarcarVisto(nombre,fecha,plataforma)
Contenido
FueVisto(nombre,plataforma)
alt
[yaFueVisto]
YaVisto
[noFueVisto]
MarcarVisto(nombre,fecha,plataforma)
[noExisteContenido]
Error
Error

5. Diagrama de clases