Instituto Tecnológico de Estudios Superiores de Monterrey

Tecnológico de Monterrey

Reporte: Entrega final del reto

Programación Orientada a Objetos (POO)

TC 1030

Group 301

Autores:

David Elias Vargas Ayala A01753587

Juan Manuel Sánchez Velázquez A01802469

Professor: Antonio Víctor Mejía Olvera

Fecha de entrega: Miércoles 11 de Junio del 2025

Índice:

Introducción:

Los servicios de streaming son un negocio que ha proliferado con el paso de los años debido a su costo relativamente competitivo ante otras opciones de renta de productos, siendo una opción que permite a los usuarios de estas plataformas poder ver películas y series en esta misma, lo que ha provocado que múltiples empresas como Netflix, Amazon o Disney, comienzan a emerger en este negocio.

Esto dio paso a la creación de sistemas informáticos que permiten la gestión de contenido para estas plataformas, estas plataformas permiten a los usuarios interactuar con el programa de manera sencilla.

Contando con lo anterior, este proyecto tiene como objetivo modelar un servicio de streaming, que permita a los usuarios interactuar con un sistema para calificar, filtrar y mostrar tanto películas como series, de acuerdo con criterios como la calificación y el género. Además, el sistema debe ser capaz de gestionar episodios de series y ofrecer un método para la calificación para videos.

Diagramas de Clases UML:

Los diagramas de clases UML con los que se trabajaron fueron desarrollados conforme las necesidades de las entregas. Se crearon diversas versiones que se presentan a continuación:

Versión 1:

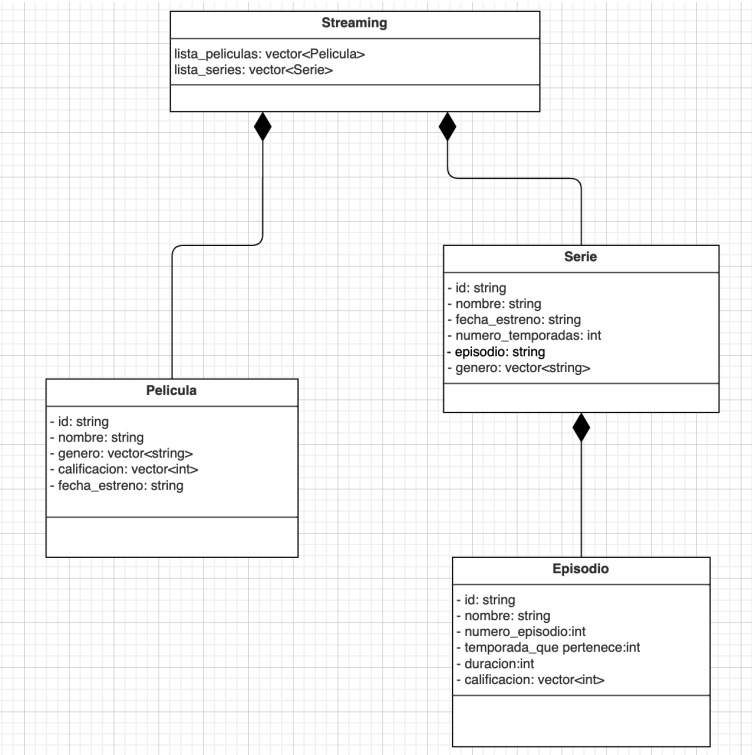


Figura 1: Versión 1 del diagrama UML

En este diseño inicial, el objetivo principal fue representar una plataforma de streaming con los elementos básicos: películas y series. Para ello, se definieron cuatro clases: Streaming, Pelicula, Serie y Episodio. La clase Streaming contiene listas de películas y series, lo cual da la estructura de una plataforma. Se utilizó la composición para mostrar que Streaming se compone de objetos tipo Pelicula y tipo Serie, y que a su vez, Serie se compone de objetos tipo Episodio. Esta primera versión se enfoca solamente en establecer la relación de pertenencia entre las clases.

Este modelo permite organizar el contenido de manera jerárquica y se observa cómo una plataforma de streaming almacena y accede a películas y series. También permite modelar la relación entre series y sus episodios. La elección de vectores para contener objetos múltiples (películas, series o géneros) facilita el acceso y la escalabilidad del catálogo.

Decisiones de diseño:

- Se decidió crear clases separadas para Pelicula, Serie y Episodio, lo cual mejora la claridad.
- Se establecieron relaciones de composición entre Streaming y su contenido, así como entre Serie y Episodio, reflejando una estructura jerárquica.
- Diseño sin métodos: En esta fase, se evitó incluir comportamientos (métodos), centrándose únicamente en la estructura de datos.

Versión 2:

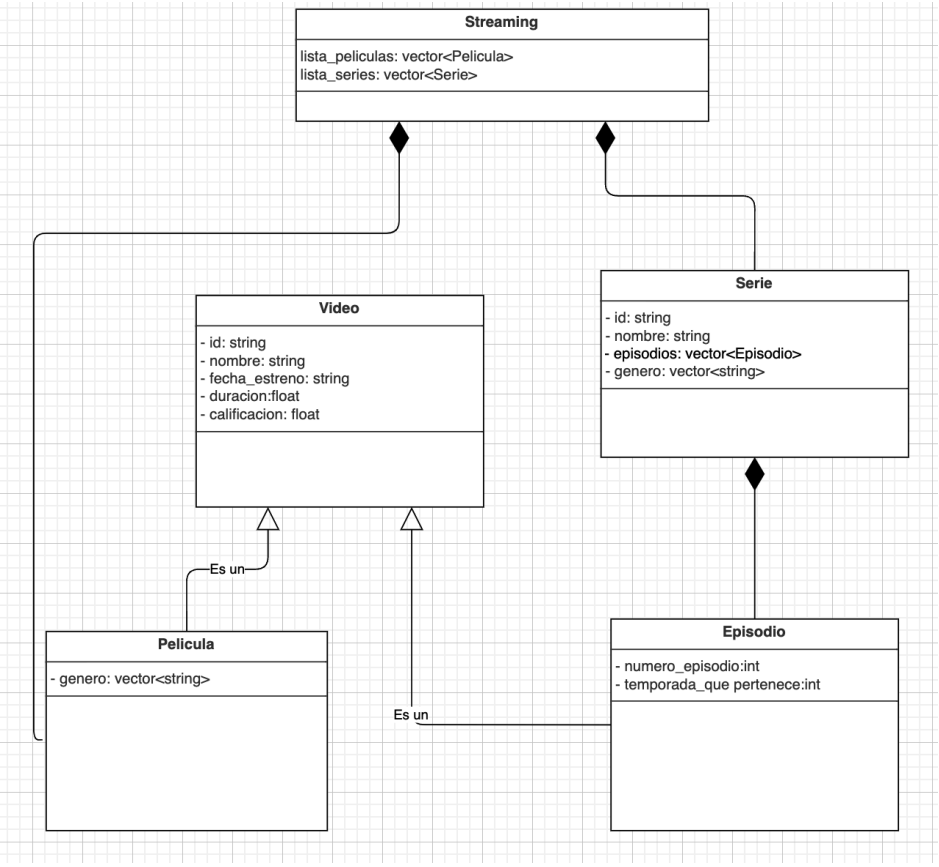


Figura 2: Versión 2 del diagrama UML

Este segundo diseño introduce una mejora importante al agrupar atributos comunes de películas y episodios en una nueva clase llamada Video. La clase Serie se modificó para tener una lista de objetos Episodio, lo cual corrige el error del diseño anterior donde solo existía un atributo episodio como string. Aquí, tanto Película como Episodio heredan de Vídeo, permitiendo agrupar sus atributos de forma centralizada.

La clase Video encapsula las propiedades compartidas entre distintos tipos de contenido audiovisual. Además, la corrección en la clase Serie mediante el uso de vector<Episodio> mejoró el modelo.

Decisiones de diseño:

- Se identificaron atributos repetidos en Película y Episodio, los cuales fueron extraídos a una clase padre común para una mejor distribución.
- Se usó herencia simple para especializar Video en Pelicula y Episodio, aprovechando las propiedades comunes.
- Se ajustó el atributo episodio para que sea una colección (vector<Episodio>).

Diagrama de secuencia y versión 3 UML:

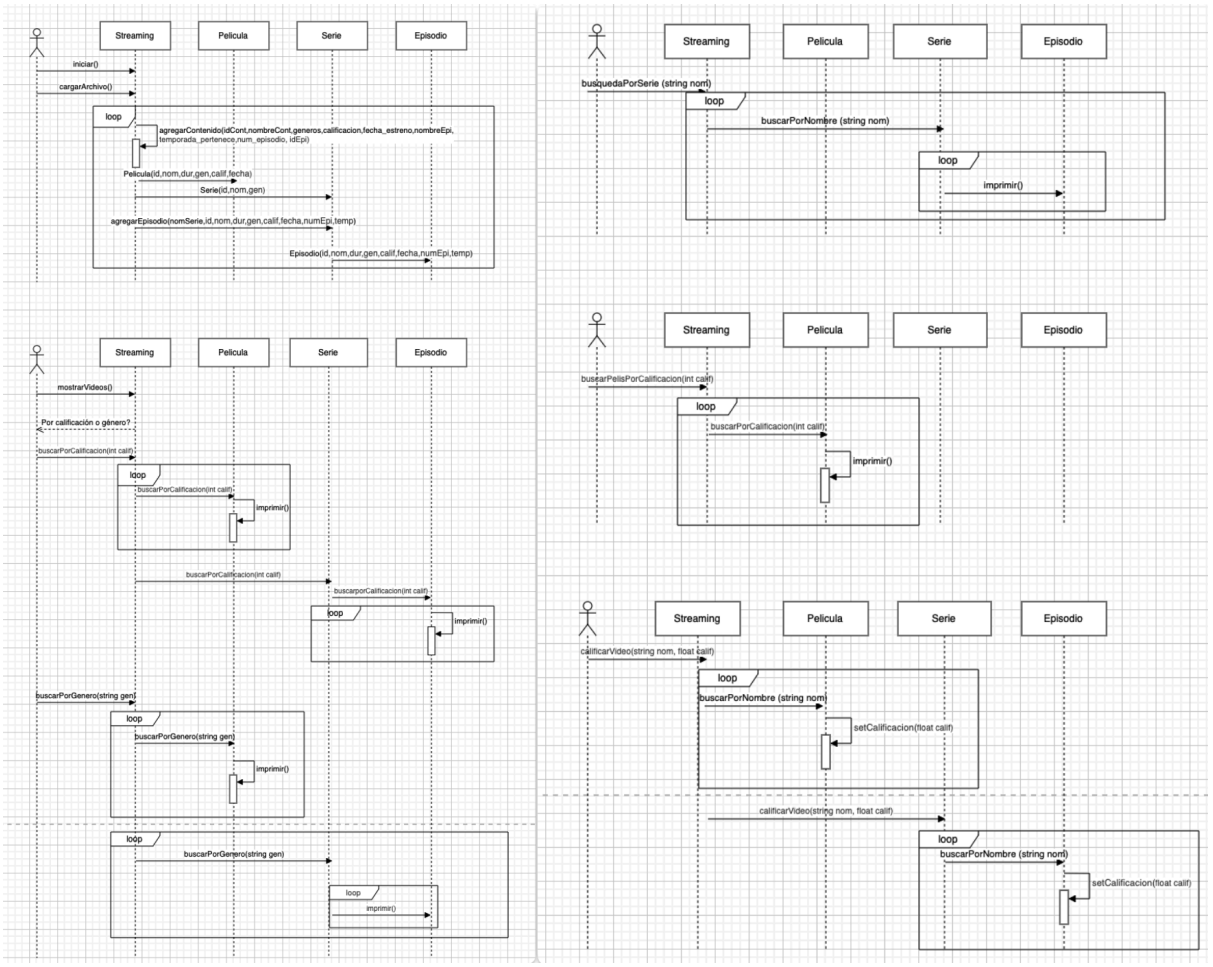


Figura 4: Diagramas de secuencia de los métodos

Para el trabajo de esta versión se requirió la creación de un diagrama de secuencia, con la intención de encontrar los métodos de cada clase.

1. cargarArchivo()

Este método permite leer un archivo de datos (formato CSV o Excel) que contiene la información de los videos. Al iniciar el proceso, el sistema recorre cada línea del archivo, identifica el tipo de contenido (película, serie o episodio) y crea los objetos correspondientes con sus atributos (como ID, nombre, duración, calificación, fecha, género, número de episodio, temporada, etc.). Finalmente, agrega cada objeto a las listas correspondientes dentro del sistema para su posterior uso.

2. mostrarVideos()

El método mostrarVideos() ofrece al usuario dos opciones de filtrado: por calificación o por género. Si el usuario elige filtrar por calificación, el sistema recorre la lista de videos y muestra aquellos cuya calificación supera el parámetro especificado. Si elige filtrar por género, el sistema recorre la lista de videos mostrando solo aquellos que coincidan con el género seleccionado. Ambos procesos implican recorrer los objetos de tipo Pelicula y Serie y utilizar sus respectivos métodos de búsqueda e impresión.

3. busquedaPorSerie(string nom)

Este método permite buscar una serie específica mediante su nombre. El sistema recorre el vector de contenido y verifica si son series, para posteriormente localizar aquella que coincide con el nombre proporcionado. Una vez encontrada, el sistema recorre los episodios de dicha serie y muestra en pantalla la información de cada uno, incluyendo nombre, número de episodio, temporada, duración, fecha de estreno y calificación.

4. buscarPelisPorCalificacion(int calif)

Este método filtra exclusivamente las películas cuya calificación es superior al parámetro proporcionado. El sistema recorre la lista de películas almacenadas, evalúa si cada una cumple con el criterio y, en caso afirmativo, muestra su información en pantalla.

5. calificarVideo(string nom, float calif)

El método permite al usuario asignar una nueva calificación a un video existente. El sistema solicita el nombre del video (ya sea película, serie o episodio), lo busca en la lista correspondiente y, si lo encuentra, actualiza su calificación mediante el método setCalificacion(). Este proceso puede aplicarse tanto a videos individuales como a episodios específicos dentro de una serie.

6. salir()

Este método finaliza la ejecución del programa. No realiza operaciones sobre los objetos almacenados, simplemente termina el ciclo principal del sistema y cierra la aplicación.

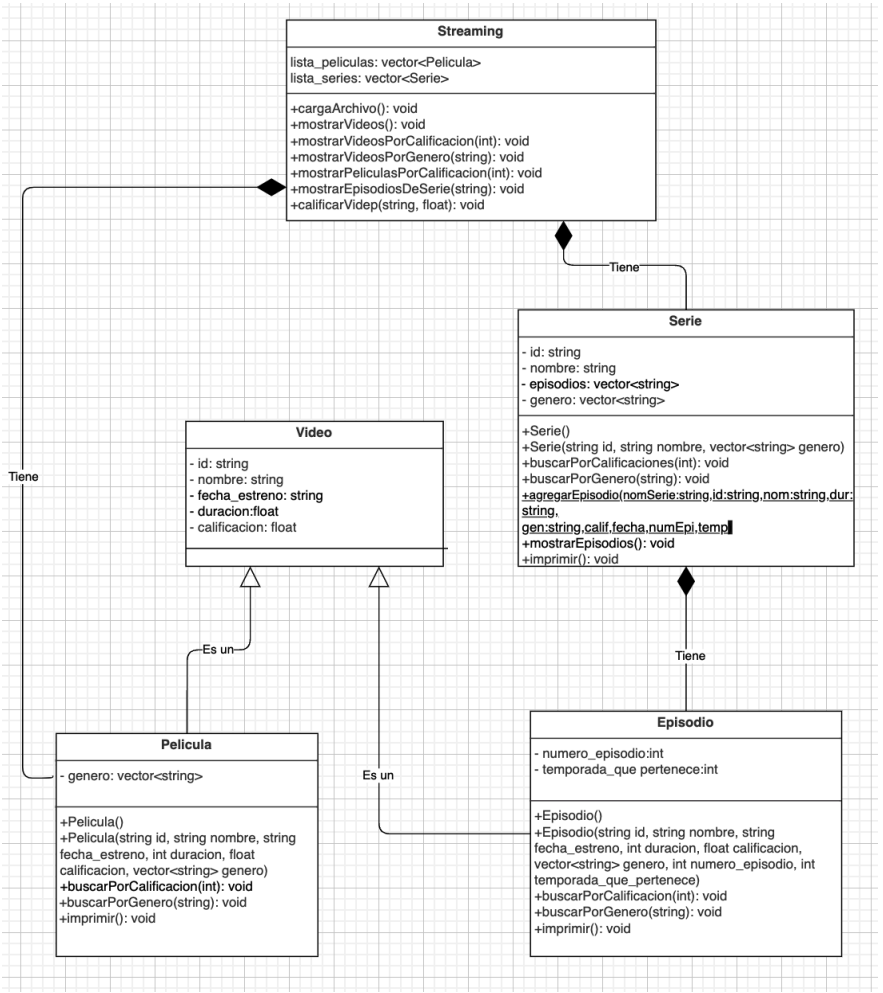


Figura 4: Versión 3 del diagrama UML

En esta versión, el diseño tuvo cambios importantes debido a la inclusión de métodos en las clases y también se tomó un enfoque más completo de programación orientada a objetos. La clase Streaming ahora contiene múltiples métodos para cargar archivos, mostrar contenido, buscar por calificación y calificar videos. Además, se refuerza el uso de la clase base Video (que heredan Pelicula y Episodio), y se optimiza el uso de vectores en Serie para agrupar episodios. Las clases Pelicula y Episodio también definen sus propios métodos, como calcularCalificacionPromedio() o mostrarDatos(), lo que permite encapsular funcionalidad específica de cada tipo de contenido.

La inclusión de métodos permite que cada clase gestione su propio comportamiento. Se justifica la separación de responsabilidades: Streaming se encarga de administrar el catálogo, mientras que Video, Pelicula, Serie y Episodio manejan su propia lógica interna. Esto permite escalar el sistema con mayor facilidad.

Decisiones de diseño

1. Se agregaron métodos a las clases principales, lo cual permite que cada objeto sea responsable de sus propias acciones.

2. Se delimitaron claramente las funciones del sistema, asignando a Streaming las tareas de gestión general del contenido y a las demás clases la gestión interna de sus datos.

3. Se mantuvo la estructura jerárquica con Video como clase base, fortaleciendo la reutilización.

4. Se comenzó a aplicar una programación más realista, en donde cada clase tiene tanto atributos como métodos que representan comportamientos esperados.

Versión 4.0:

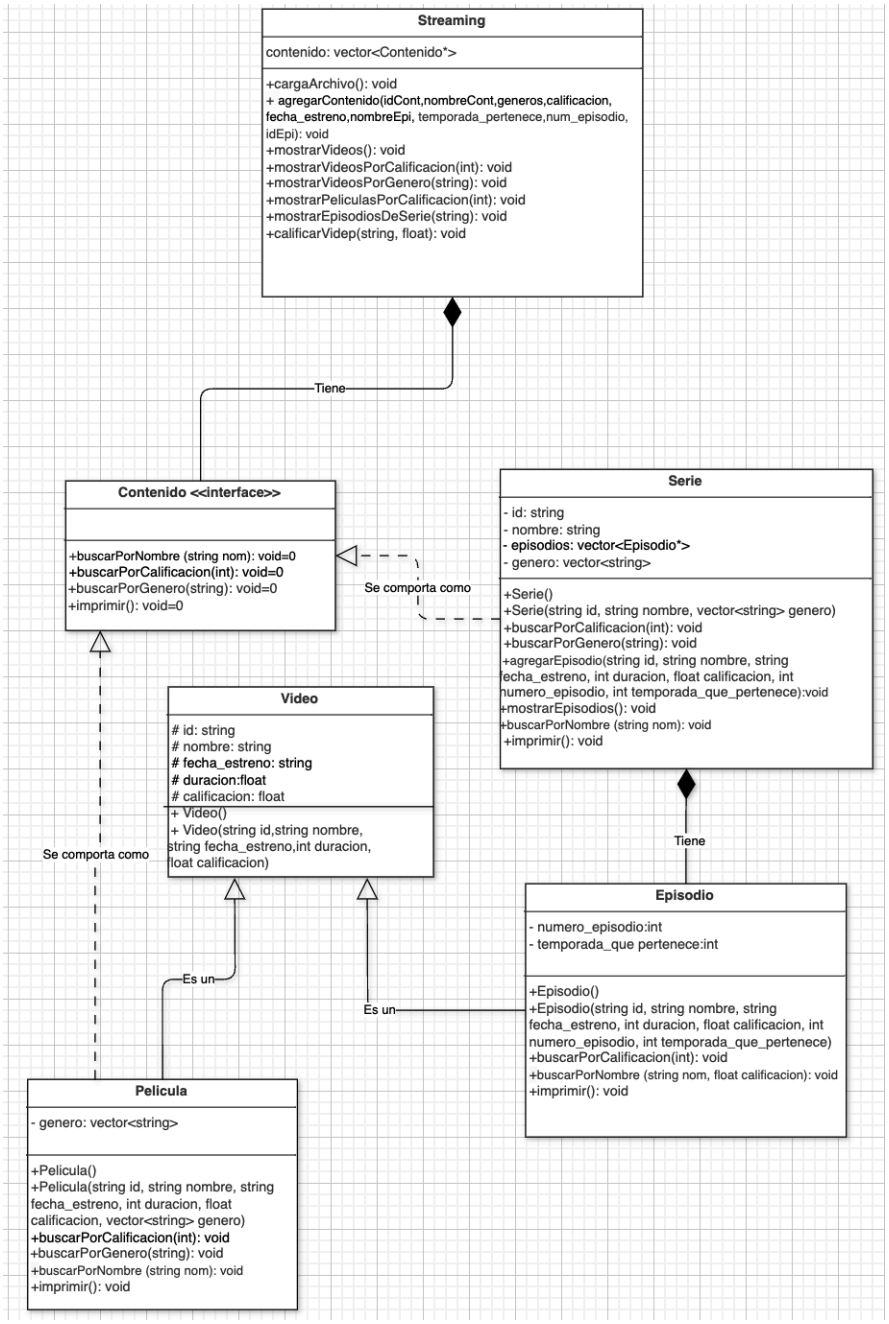


Figura 5: Versión 4 del diagrama UML

Esta versión del diseño introduce una interfaz llamada Contenido, de la cual Video y Serie adquieren su comportamiento, es decir, sus métodos. La clase Video mantiene su rol como contenedor común de atributos compartidos entre Pelicula y Episodio, debido a que ambos son videos. Streaming se conecta directamente con Contenido, mediante una relación de composición, permitiendo agregar polimorfismo a las tres clases, Serie, Episodio y Pelicula, por lo que finalmente, el diseño aplica principios de herencia, polimorfismo y encapsulamiento.
Este modelo permite una gestión unificada del contenido a través del polimorfismo, facilitando el manejo de películas y series como objetos de tipo Contenido. Esto es útil para la escalabilidad del sistema (por ejemplo, si se desea agregar nuevos tipos de contenido como documentales o cortos). La abstracción de responsabilidades y la jerarquía clara mejoran la mantenibilidad, extensibilidad y comprensión del sistema.
Decisiones de diseño:

1. La creación de clase abstracta Contenido permite generalizar todo el contenido bajo un mismo tipo, lo que facilita el uso de polimorfismo.

2. Gracias a los métodos virtuales, se permite que cada tipo de contenido implemente su propia versión de métodos comunes como imprimir().

3. Streaming ya no gestiona listas específicas de películas y series, sino un vector general de Contenido, lo que simplifica la lógica y permite añadir nuevos tipos de contenido sin modificar la clase base.

Versión 5:

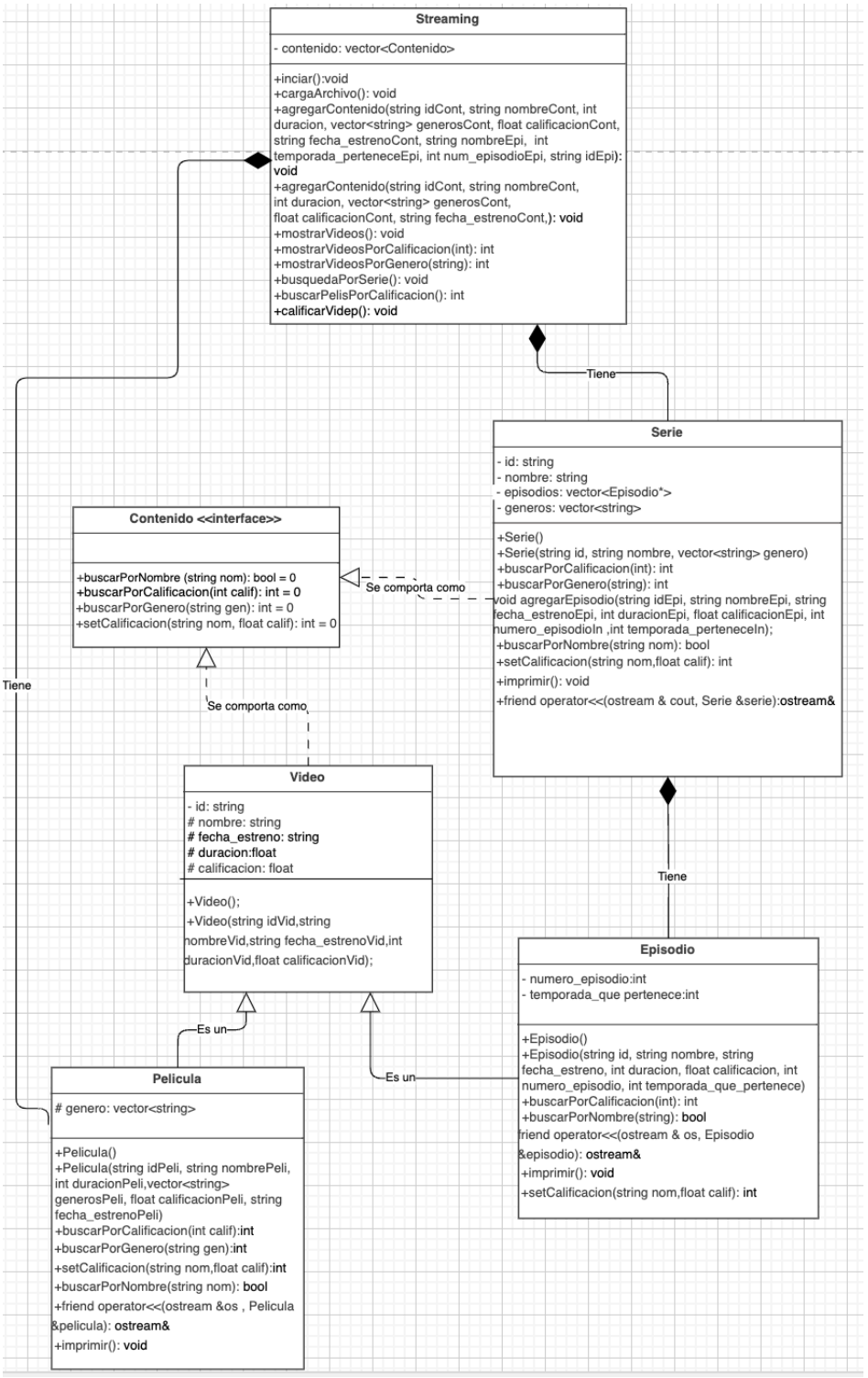


Figura 6: Versión final del diagrama UML

Esta es la versión final del el diagrama UML, el cual describe todos los métodos empleados durante el programa, sus tipos de retorno y lso atributos de cada clase, se hicieron ajustes en algunas clases en cuanto al modificador de acceso de atributos de privado a protegido para que clases derivadas puedan acceder y cambiar los mismos, además se modificó el tipo de retorno de varias funciones para la implementación de manejo de excepciones, de

forma que esos retornos se guardan en una variable y dependiendo del valor de la misma se pueden lanzar excepciones dependiendo del contexto.
Adicionalmente en la última entrega se añadieron métodos dentro de streaming para mejorar su funcionamiento, un ejemplo de ello es el método iniciar, el cual se encarga de administrar todas las funcionalidades del sistema de Streaming.
Además se agregó sobrecarga de operadores dentro del código, el cual se decidió que se le aplicase al método imprimir indirectamente, de forma que ahora el método imprimir llama a una salida de flujo de datos que imprime toda la información correspondiente de cada contenido mediante el operador <<.
Finalmente dentro de los métodos de la interfaz contenidos, se agregó uno nuevo para que se cambie la calificación de cada contenido.

Argumentación para el diseño del código:

Elecciones de diseño:

En esta parte del reporte, explicamos cómo y por qué construimos nuestro sistema de streaming de la manera en que lo hicimos. Se tomó en cuenta cada punto como una característica que el sistema debe cumplir, y nosotros consideramos que nuestra solución es es adecuada para satisfacer las necesidades de este proyecto.

1. Cargar la lista de películas y series (desde un archivo CSV)

Lo que hace: El sistema puede leer la información de todas las películas y series desde un archivo. Esto evita la necesidad de escribir manualmente los archivos.
Cómo lo hicimos: Se creó una parte del programa encargada de "leer" ese archivo (que debe ser un CSV). Esta parte toma cada línea de información y la convierte en objetos de "Película" o "Serie" dentro del sistema.

2. Mostrar los videos en general, ya sea por:

1) Que tengan una calificación mayor a un número solicitado por el usuario

Lo que hace: Si se solicita ver solo las películas o series que tienen, por ejemplo, 7 de calificación o más, el programa encuentra las películas y las imprime.
Cómo lo hicimos: Se consideró que tanto las películas como los episodios de las series son, al final, "videos". Así que creamos una base común para todos los "videos" que identifica su calificación. Luego, cada tipo de video (película o episodio) revisa su propia calificación. Cuando se le solicita al programa que busque, este le pregunta a cada video individualmente si cumple con la calificación deseada.

Por qué esta forma y no otra:

- Funciona para todos los "videos" porque al tener una forma estándar de preguntar por la calificación en un "video" general, no necesitamos escribir el mismo código una y otra vez para películas y para episodios. Si en el futuro se desea añadir una nueva clase de video como "documentales", sabrán cómo manejar su calificación de la misma forma. Esto hace que el código sea más ordenado y fácil de trabajar.

- Cada película o episodio es responsable de determinar si su calificación es la adecuada, sin que el programa principal tenga que intervenir en sus procesos internos.

2) Que sean de un cierto tipo o "género"

Lo que hace: Se le solicita al programa que te muestre solo videos de un genero especifico.
Cómo lo hicimos: Tanto las películas como las series tienen su propia lista de "géneros" a los que pertenecen (una película puede ser de “Action” y “Drama”). Cuando pides un género, el programa le pregunta a cada película o serie si lo tiene en su lista.

Por qué esta forma y no otra:
- Cada película o serie es la encargada de manejar su propia información de géneros.
- Diseñamos el sistema para que una película o serie pueda tener más de un género según los requerimientos de los datos de la base de datos.

Otras ideas que tuvimos (y por qué no las usamos):
- Si hubiéramos determinado que un video solo puede tener un género, sería muy limitado y apto para la base de datos con la que se trabajo. Además esto provocaría que no hubiera más de un género.
- De nuevo, una única función que buscara géneros para todos los tipos de video sería complicada de mantener y crecer.

3. Mostrar los episodios de una serie específica.

Lo que hace: Si el usuario solicita una serie en particular, el programa puede mostrarte todos los episodios con los que esta cuenta.
Cómo lo hicimos: Cuando creamos una "Serie" en el programa, esta "contiene" todos sus "Episodios". De esta forma, cada serie sabe cuáles son sus episodios. Además, hicimos que, al "imprimir" una serie, automáticamente se impriman todos los detalles de sus episodios.

Por qué esta forma y no otra:
- Así es como funciona en la vida real, una serie está hecha de episodios. Nuestro código refleja esa relación de forma muy clara.
- Al hacer que la serie tenga un método para mostrar sus episodios, es muy sencillo pedirle al programa que imprima una serie y que automáticamente muestre todo su contenido.
- Puedes ir directamente a una serie y ver todo lo que tiene, sin tener que buscar cada episodio por separado.

Otras ideas que tuvimos (y por qué no las usamos):
- Se consideró tener una lista de todos los episodios por separado. Sería muy difícil determinar qué episodio pertenece a qué serie. Con nuestra forma, la serie "guarda" sus episodios, lo que simplifica mucho las cosas.
- En lugar de imprimir la serie y que ella sola se encargue de sus episodios, podríamos haber tenido que llamar a muchas funciones distintas. La manera en la que fue realizada es más directa y ordenada.

4. Mostrar solo las películas que tienen una calificación alta.

Lo que hace: Similar a la función anteriormente vista, pero esta vez se toma enfoque unicamente en las películas que superan una calificación que es solicitada.
Cómo lo hicimos: El programa tiene una función específica para esta tarea. El programa accede directo a su lista de "Películas" y le pregunta a cada una si su calificación es mayor o igual a la que se solicita. Si lo es, la muestra.

Por qué esta forma y no otra:

- Aunque se pudo introducir en una opción más general, tener una opción directa para "películas con buena calificación" hace que el menú sea más claro y fácil de usar para el usuario.

Otras ideas que tuvimos (y por qué no las usamos):

- Se pudo haber introducido a la búsqueda general de videos , pero creemos que tener una opción separada es más intuitivo para el usuario que quiere buscar específicamente películas.

5. Calificar un video (ponerle una nueva puntuación)

Lo que hace: Permite al usuario cambiar la calificación que un video ya tiene.
Cómo lo hicimos: Cada "video" tiene una forma de "recibir" una nueva calificación. El programa solicita al usuario ingresar el nombre del video que quieres calificar y la nueva puntuación. Después, busca ese video, ya sea entre las películas o dentro de las series (buscando el episodio correcto). Una vez que lo encuentra, le solicita al video que actualice su calificación.

Por qué esta forma y no otra:

- Al igual que con la búsqueda de calificaciones, cualquier "video" sabe cómo recibir una nueva puntuación. Esto es muy útil ya que no se necesita redactar código diferente para películas y episodios.
- La forma en que se actualiza la calificación está dentro de cada video. Esto permite asegurar que la calificación se encuentra en un intervalo determinado.
- Las series, al tener muchos episodios, se encargan de que la calificación se aplique al episodio correcto.
-

Otras ideas que tuvimos (y por qué no las usamos):

- Nuestro proyecto se enfoca en calificar episodios individuales. Si se hubiera requerido calificar series enteras, la lógica sería diferente, pero para este caso, calificar por episodio es más preciso.

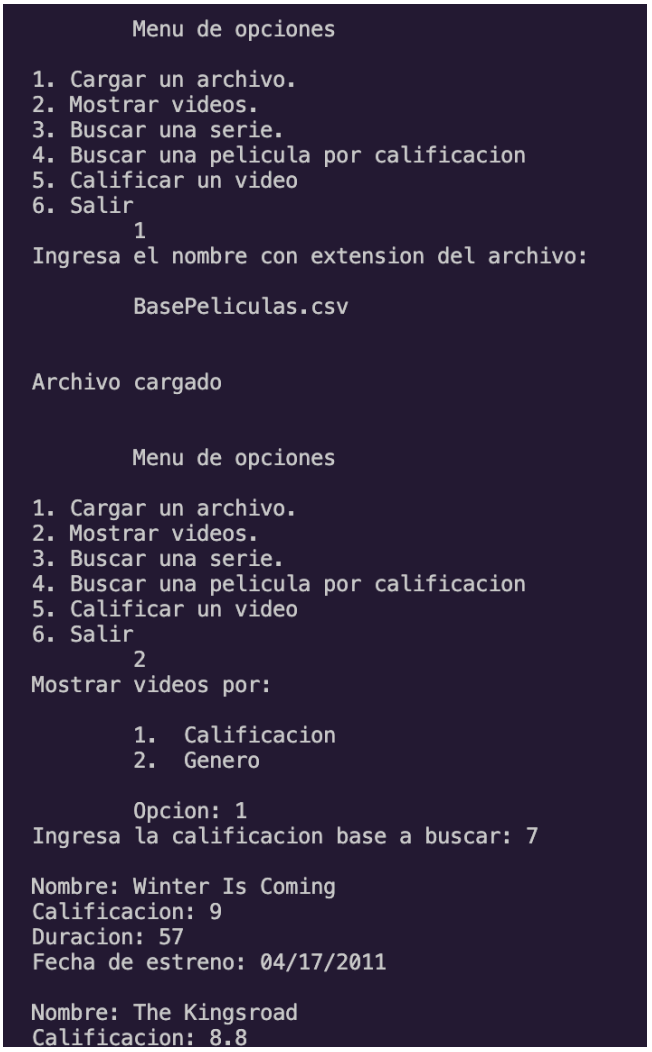
6. Salir del programa:

Lo que hace: Simplemente detiene la ejecución del programa cuando el usuario lo desea.
Cómo lo hicimos: El programa muestra un menú que se repite una y otra vez. Cuando eliges la opción "Salir", el programa deja de repetirse y finaliza de forma ordenada.

Por qué esta forma y no otra:

- Le da al usuario la libertad de decidir cuándo quiere terminar.
- Para un programa de este tipo, es la forma más sencilla y segura de cerrarlo, asegurando que todo se limpie correctamente antes de finalizar.

Ejemplo de Ejecución:



Conclusiones:

Este documento ha presentado la presentacion de un sistema de modelado de servicio de streaming en el que se pone en práctica la Programación Orientada a Objetos (POO). El uso de la herencia y el polimorfismo han sido pilares fundamentales en este proyecto y se han demostrado en la estructura de clases las clases, permitió tener una buena gestión de las clases de películas y series, facilitando la implementación de funcionalidades clave como la carga de datos, búsquedas y calificación de contenido.

Las decisiones de diseño fueron adaptándose conforme las necesidades del proyecto. Aunque existen áreas para futuras mejoras, el proyecto ha logrado simular las operaciones básicas de un servicio de streaming de manera efectiva, permitiendo tener una buena comprensión de cómo los principios de la POO se traducen en soluciones de software prácticas y eficientes.

Referencias Consultadas

GeeksforGeeks. (s.f.). *OOPs Object Oriented Design*. Recuperado de https://www.geeksforgeeks.org/oops-object-oriented-design/

Deitel, P. J., & Deitel, H. M. (2018). *C++ How to Program* (10a ed.). Pearson. Recuperado de https://learning.oreilly.com/library/view/c-how-to/9780134448930/xhtml/fileP700101197300000000000000015B2.xhtml#P700101197300000000000000015CF

Larman, C. (2020). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (3a ed.). Pearson. Recuperado de https://learning.oreilly.com/library/view/the-object-oriented-thought/9780135182130/

Ceballos Sierra, F. J. (s.f.). *Enciclopedia del Lenguaje C++*. Recuperado de https://elhacker.info/manuales/Lenguajes%20de%20Programacion/C/kupdf.net_enciclopedia-del-lenguaje-c-ceballos-sierra-fco-javierau.pdf