Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación orientada a objetos

Nombre del profesor:
Gabriela Zamora Leal

Proyecto integrador

Cyrce Danae Salinas Rojas ITC | A01666121

Indice de contenido

Introducción

En el contexto actual, la industria del entretenimiento ha experimentado un crecimiento significativo en la oferta de servicios de streaming de video bajo demanda. Plataformas como Netflix, Disney+, y DC Universe han revolucionado la forma en que los usuarios consumen contenido multimedia, ofreciendo una vasta biblioteca de videos que abarca desde películas hasta series exclusivas. Cada uno de estos servicios se distingue por su enfoque y catálogo: algunos priorizan la cantidad y variedad de videos disponibles, mientras que otros se enfocan en contenidos propios de alta calidad.

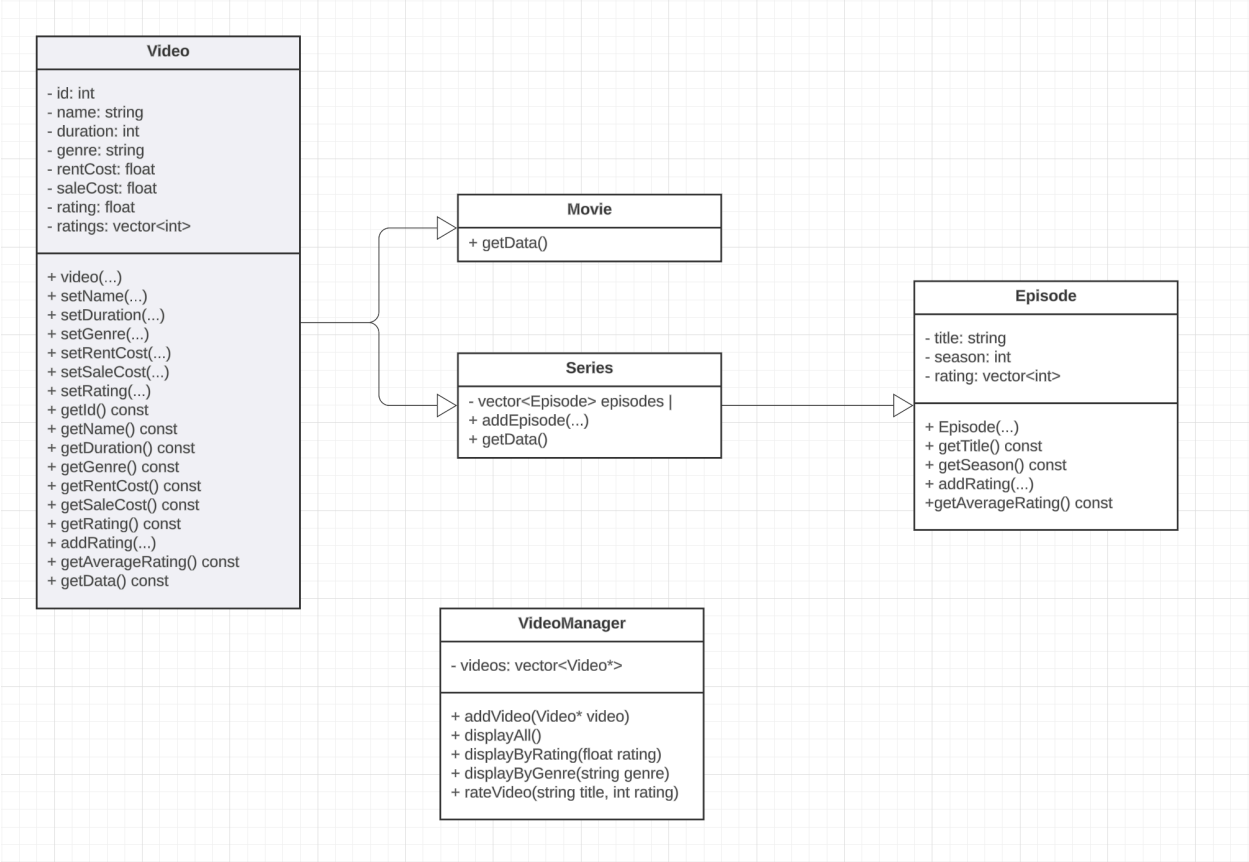
Ante esta tendencia, surge la necesidad de modelar un sistema que permita a futuros proveedores de servicios de streaming gestionar y presentar sus contenidos de manera eficiente y atractiva para los usuarios. Este sistema debe ser capaz de manejar dos tipos principales de videos: películas y series. Cada video posee atributos esenciales como un ID único, un nombre, una duración y un género (drama, acción, misterio). Además, las series están compuestas por episodios, los cuales tienen su propio título y están asociados a una temporada específica.

Una característica crucial del sistema es la capacidad de registrar y mostrar las calificaciones promedio que los usuarios asignan a cada video, utilizando una escala de 1 a 5, donde 5 representa la máxima calificación. Para lograr esto, el sistema debe cumplir con las siguientes funcionalidades:

- Mostrar una lista general de videos con sus respectivas calificaciones.
- Mostrar los episodios de una determinada serie junto con sus calificaciones.
- Mostrar las películas con sus calificaciones.

La implementación de este sistema requiere la aplicación de conceptos de Programación Orientada a Objetos (POO) como herencia, polimorfismo y sobrecarga de operadores. Estos conceptos serán fundamentales para diseñar un diagrama de clases que represente la estructura del sistema de manera clara y eficiente.

Diagrama de clases



Este diagrama UML muestra las relaciones entre las clases, los atributos y los métodos de cada clase. Puedes utilizar una herramienta de diagramación como Lucidchart, Draw.io, o incluso un software especializado en UML como StarUML para crear una versión gráfica de este diagrama textual.

Ejemplo de ejecucion

Link con video corriendo

https://youtu.be/Rds9FGgXlBE

Argumentación de las partes del proyecto

Este proyecto consiste en la creación de un sistema para gestionar información sobre videos, incluyendo películas y series. A continuación, se argumenta cada parte del proyecto, describiendo su propósito y su implementación.

1. Clase Video

Propósito: La clase Video actúa como una clase base para representar videos genéricos, proporcionando los atributos y métodos comunes a todos los tipos de videos, como el ID, el nombre, la duración, el género, el costo de alquiler y venta, y la calificación.

Implementación:

> **Atributos**: Almacena información básica sobre el video.
>
> **Métodos**: Proporciona getters y setters para cada atributo, un método para agregar calificaciones (addRating), y un método para calcular la calificación promedio (getAverageRating). Además, getData es un método virtual para permitir que las clases derivadas lo sobreescriban y presenten información específica.

2. Clase Movie

Propósito: Movie es una clase derivada de Video que representa específicamente películas. Aprovecha los atributos y métodos heredados de Video y puede agregar funcionalidades adicionales en el futuro si es necesario.

Implementación:

> **Atributos y Métodos**: Hereda todos los atributos y métodos de Video. Sobrescribe el método getData para mostrar la información específica de una película.

3. Clase Series

Propósito: Series es otra clase derivada de Video, diseñada para manejar series de televisión. Además de los atributos heredados, Series agrega una colección de episodios (episodes).

Implementación:

> **Atributos**: Incluye una lista de objetos Episode que representan los episodios de la serie.
>
> **Métodos**: Proporciona un método para agregar episodios (addEpisode) y sobrescribe getData para mostrar información tanto de la serie como de sus episodios.

4. Clase Episode

Propósito:Episode representa un episodio individual dentro de una serie. No hereda de Video, ya que su contexto es específico a series y no se maneja de forma independiente en este sistema.

Implementación:

> **Atributos**: Almacena información sobre el título y la temporada del episodio, así como sus calificaciones.
>
> **Métodos**: Similar a Video, proporciona métodos para agregar calificaciones y calcular la calificación promedio.

5. Clase VideoManager

Propósito: VideoManager maneja la colección de videos (películas y series) y proporciona la lógica para realizar diversas operaciones sobre esta colección.

Implementación:

> **Atributos**: Mantiene un vector de punteros a Video (videos), permitiendo el almacenamiento polimórfico de Movie y Series.
>
> **Métodos**: addVideo para agregar videos a la colección. displayAll para mostrar la información de todos los videos.

displayByRating para mostrar videos con una calificación mínima especificada.

displayByGenre para mostrar videos de un género específico.

rateVideo para permitir la calificación de un video por su título.

6. main.cpp

Propósito: El archivo main.cpp contiene la lógica principal del programa, gestionando la interacción del usuario con el sistema.

Implementación: Funciones de Entrada: inputMovieDetails y inputSeriesDetails permiten al usuario ingresar datos específicos para películas y series.

Interacción del Usuario: Pregunta al usuario el número de películas y series que desea ingresar, luego solicita detalles para cada uno, incluidas las calificaciones.

Operaciones del Sistema: Usa VideoManager para agregar videos, mostrar todos los videos, mostrar videos por calificación o género, y calificar un video específico.

Limpieza: Elimina la memoria asignada dinámicamente para evitar fugas de memoria.

Identificación de casos de error

Es nuestro caso, al dar un valor no correspondiente al requerido, es decir si el valor que te piden es un numerico y tu das un caracter, el programa comienza a trabarse y deberas parar la ejecucion con control + c. Se anexa un ejemplo de lo que sucede si ocurre lo antes mencionado.

https://youtu.be/HhdMkPSHOtE

Conclusión personal

Trabajar en este proyecto de gestión de videos me dio una valiosa experiencia en varios aspectos de la programación y diseño de software.

Herencia: La estructura de clases con Video como clase base y Movie y Series como derivadas demuestra cómo reutilizar código y extender funcionalidades.

Polimorfismo: El uso de métodos virtuales como getData permite llamadas polimórficas, donde el tipo específico del objeto se determina en tiempo de ejecución.

Encapsulación: El uso de getters y setters para acceder a atributos privados ayuda a mantener la integridad de los datos.

Diseño Modular y Extensible: La separación de responsabilidades en diferentes clases y archivos facilita el mantenimiento y escalabilidad del código.
Añadir nuevos tipos de videos o funcionalidades adicionales sería relativamente sencillo gracias a este diseño modular.

Gestión de Memoria: La correcta asignación y liberación de memoria dinámica usando punteros y el vector de punteros a Video asegura que no haya fugas de memoria, lo cual es crucial en proyectos de C++.

Interacción con el Usuario: La implementación de un sistema interactivo que solicita y maneja la entrada del usuario mejora la comprensión de cómo diseñar interfaces de usuario en aplicaciones de consola.
Desafíos que puedo enfrentar

Manejo de Memoria: Asegurar que toda la memoria asignada dinámicamente se libere adecuadamente es crítico. Un error en este aspecto podría causar fugas de memoria y problemas de rendimiento.

Gestión de Complejidad: A medida que el sistema crezca, mantener el código limpio y comprensible puede volverse desafiante. Es importante seguir buenas prácticas de programación y refactorizar cuando sea necesario.

Errores en la Entrada del Usuario: Manejar entradas incorrectas o inesperadas del usuario requiere una validación exhaustiva de entradas y una robusta gestión de errores para evitar fallos en el programa.

Extensibilidad: Aunque el diseño actual es modular, añadir nuevas funcionalidades o tipos de videos puede requerir cambios significativos en el código existente, lo que debe hacerse con cuidado para no introducir errores.

Este proyecto no solo es un ejercicio técnico, sino también una práctica en el diseño de software bien estructurado. La implementación exitosa de este sistema de gestión de videos proporciona una sólida comprensión de los conceptos fundamentales de OOP, habilidades en el manejo de memoria y la capacidad de diseñar sistemas extensibles y mantenibles. Enfrentar y superar los desafíos que presenta también ayuda a desarrollar un enfoque meticuloso y reflexivo hacia la programación, esencial para cualquier desarrollo de software profesional.

Referencias consultadas

BillWagner. (2023, June 4). *Programación orientada a objetos: herencia - C#*. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/object-oriented/inheritance

BillWagner. (2023, June 4). Polimorfismo - C#. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/object-oriented/polymorphism

Simplilearn. (2021, September 12). *C++ Inheritance Tutorial | Introduction to inheritance in C++ programming with example | Simplilearn* [Video]. YouTube. https://www.youtube.com/watch?v=qYY9eR7Ldek

freeCodeCamp.org. (2021, February 2). *Object Oriented Programming (OOP) in C++ course* [Video]. YouTube. https://www.youtube.com/watch?v=wN0x9eZLix4