Tecnológico de Monterrey

Situación Problema

A01644423

Luis Ignacio Gómez López

A01645384

Juan Pablo De La Peña González

A01645653

Marissa Edith Luna Landa

Programación Orientada a Objetos

Grupo 319

Tania del Carmen Rodríguez Flores

Fecha de entrega: 14 de junio de 2024.

Índice

Introducción

En los últimos años, la popularidad de los servicios de streaming de video bajo demanda ha experimentado un notable aumento. Estos servicios se destacan ya sea por la gran cantidad de videos que ofrecen o por el contenido exclusivo que proporcionan. En este contexto, se presenta el presente documento que introduce una versión preliminar de un proyecto diseñado para apoyar a un futuro proveedor de este tipo de servicios.

El proyecto se enfoca en la gestión de dos tipos de videos: películas y series. Cada video será identificado por un ID único, un nombre, una duración y un género, siendo los géneros principales drama, acción, aventura y comedia. En el caso específico de las series, estas estarán compuestas por episodios, y cada episodio contará con un título y la temporada a la que pertenece.

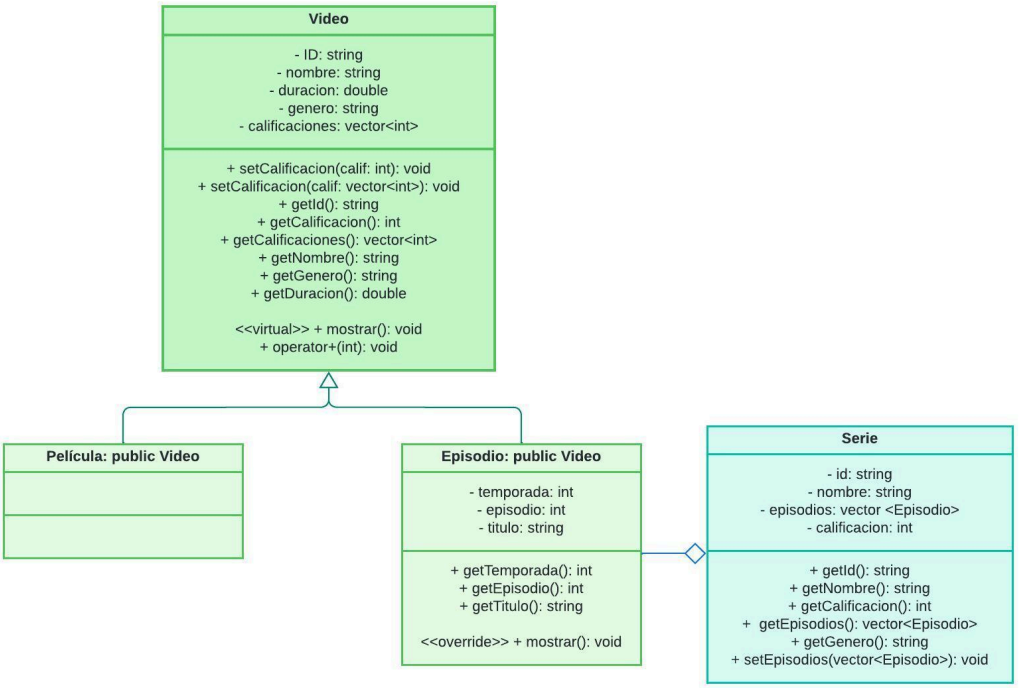
Uno de los objetivos primordiales de este proyecto es la evaluación de los videos mediante una calificación promedio en una escala del 1 al 5, donde 5 representa la calificación más alta. Esta evaluación permitirá a los usuarios y al proveedor del servicio tener una referencia clara sobre la calidad y popularidad del contenido ofrecido.

Aunado a esto, el usuario será capaz de acceder a un menú, en el que podrá manipular y acceder al catálogo de videos. El desarrollo del programa se basará principalmente en los principios fundamentales de la Programación Orientada a Objetos, tales como la herencia, el polimorfismo y la encapsulación.

Roles

Integrante	Actividades
Luis Ignacio	Realización de acciones número 1 para importar los datos del archivo al programa, y número 0, para sobreescribir el archivo de texto utilizando la información del programa.
Juan Pablo	Realización de acciones número 5 y 6: “calificar videos” y “mostrar registros de calificaciones”
Marissa Edith	Realización de acciones 2, 3 y 4: Búsqueda y filtrado de contenido a partir de iteraciones y condicionales. Validación de entradas de usuarios en estos mismos índices (calificaciones, géneros y opciones).

Diagrama de clases UML



1. Clase Base Video

La clase Video representa la abstracción fundamental de cualquier contenido audiovisual en el sistema. Agrupa atributos comunes como ID, nombre, duración, género, y calificaciones, lo cual es esencial para cualquier tipo de video.

Al tener una clase base con atributos y métodos comunes, se evita la redundancia y facilita la extensión del sistema. Esto es crucial para mantener el código limpio y fácil de mantener.

2. Clases Derivadas Película y Episodio

Ambas clases (Película y Episodio) heredan de Video, pero introducen atributos específicos (temporada y episodio para Episodio, y posiblemente otros para Película en un desarrollo más avanzado). Esto permite representar diferentes tipos de contenido de manera específica y clara.

El método mostrar es marcado como <<override>> en la clase Episodio, lo que indica que tiene su propia implementación de cómo mostrar el contenido. Esto es esencial para adaptarse a las diferencias en la presentación de películas y episodios dentro de la plataforma de streaming.

3. Clase Serie

La clase Serie se enfoca en la gestión y organización de episodios específicos bajo una misma entidad a través de un vector. Esto facilita la gestión y organización de múltiples episodios, lo cual es fundamental para la navegación y estructura de las series en la plataforma. Aunque la clase Serie no es heredada, sí depende de la clase Episodio, pues uno de sus atributos es representado por un vector de tipo Episodio.

Ejecución

Al ejecutarse se despliega un menú con las distintas acciones posibles.

```
PS C:\Users\juanp\Desktop\TEC\2do Semestre\Programación orientada a objetos\Projects\proyectoC\avance4> ./main
----------MENU----------
Ingrese la opcion que desea ejecutar:
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificacion o de un cierto genero
3. Mostrar los episodios de una determinada serie con una calificaion determinada
4. Mostrar las peliculas con cierta calificacion
5. Calificar un video
6. Mostrar registros calificaciones
0. Salir
```

El primer caso se encarga de cargar el archivo de datos (.txt) dentro del programa y de verificar que esto se haya hecho correctamente, así como permitir que las demás acciones puedan ejecutarse. Posteriormente se vuelve a desplegar el menú principal.

```
6. Mostrar registros calificaciones
0. Salir
1
El archivo fue cargado correctamente

----------MENU----------
Ingrese la opcion que desea ejecutar:
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificacion o de un cierto genero
```

El segundo caso, mostrar los videos en general con una cierta calificación o de un cierto género, muestra dos opciones, ya sea filtrar por una calificación o por un género en específico; desplegando ambas, películas y series, junto con un formato que incluye información relevante.

```
2
¬¦Deseas buscar por calificacion(1) o por genero(2)?
1
Ingrese la calificacion deseada
5
PELICULAS
--------------------
ID: P006
Nombre: The Shawshank Redemption
Genero: Drama
Duracion: 142
Calificacion: 5
--------------------
```

```
2
¬¦Deseas buscar por calificacion(1) o por genero(2)?
2
Ingrese el genero deseado (Accion, Aventura, Comedia o Drama)
Drama
PELICULAS
--------------------
ID: P006
Nombre: The Shawshank Redemption
Genero: Drama
Duracion: 142
Calificacion: 5
--------------------
```

Dentro del tercer caso, se da la opción de mostrar los episodios de una determinada serie con una calificación determinada. Empieza por enlistar las series a elegir, para después solicitar una calificación a filtrar, finalmente desglosa información de los episodios que coincidan con esta información.

```
3
Seleccione la serie a buscar:
1. Breaking Bad
2. Game of Thrones
3. Stranger Things
4. Friends
5. The Office
1
Ingrese la calificacion deseada a buscar:
5
--------------------
No. Episodio: 3
Temporada: 7
Titulo: Hazard Pay
Calificacion: 5
```

Para la cuarta acción, realiza esto mismo pero ahora en el caso de las películas.

```
4
Ingrese la calificacion deseada a buscar:
5
--------------------
ID: P006
Nombre: The Shawshank Redemption
Genero: Drama
Duracion: 142
Calificacion: 5
--------------------
```

La quinta acción, calificar un video, permite agregar a las calificaciones de estos un nuevo registro insertado por el usuario. Ofrece tres opciones, película, serie o salir. Por el lado de las películas simplemente requiere ingresar su respectivo ID, más una calificación del 1 al 5, por su puesto verificando que coincidan, para después mostrar una nota de que se ha realizado con éxito.

```
5
Calificar... opciones:
  1. Pelicula
  2. Serie
  3. Salir
1
-CALIFICAR VIDEO-
Lista de videos:
  ID  -  Nombre
  P001  -  Inception
  P002  -  The Godfather
```

```
  P020  -  Goodfellas
Selecciona ID del video a calificar: P001

Okay... Inception
Insertar calificacion del 1 al 5: 4

Calificacion actualizada
```

Ahora, por el lado de las series, se lleva a cabo un proceso similar con la diferencia en que en este es necesario realizar una búsqueda primero por series, para después seleccionar un episodio especificado de esta a calificar.

```
2
-CALIFICAR VIDEO-
Lista de videos:
  ID  -  Nombre
  S1  -  Breaking Bad
  S2  -  Game of Thrones
  S3  -  Stranger Things
  S4  -  Friends
  S5  -  The Office
Selecciona ID del video a calificar: S1

Okay... Breaking Bad
Episodios:
  S101  -  Say My Name
  S106  -  Ozymandias
  S111  -  Hazard Pay
Selecciona ID del episodio a calificar: S101
Okay... Say My Name
Insertar calificacion del 1 al 5: 5

Calificacion actualizada
```

El caso seis se trata de una acción extra la cual muestra tal cual los registros de calificaciones de cada uno de los videos de la base de datos, ya sea por películas o series. Consideramos esto importante para verificar durante la ejecución que se estén realizando correctamente los cambios.

```
6
Mostrar calificaciones... opciones:
  1. Pelicula
  2. Serie
  3. Salir
1
Lista de peliculas:
  Nombre  -  Calificaiones
  Inception  -  (4,2,5,5,4)
  The Godfather  -  (5,4,3)
```

Por último, el caso 0 realiza la tarea de sobreescribir el archivo con las actualizaciones y de finalizar la ejecución del programa.

```
  0. Salir
  0
  Saliendo...
PS C:\Users\juanp\Desktop\TEC\2do Semestre\Programación orientada a objetos\Projects\proyectoC\avance4>
```

Dentro de cada caso se verifica y notifica de posibles errores, como lo son opciones no existentes o la realización de acciones sin antes haber cargado los datos del archivo de texto.

Argumentación

a) Clases

Para la realización del código se utilizaron 4 clases: **Video**, **Pelicula**, **Episodio** y **Serie**. De las cuales, **Video** es la clase base padre abstracta, mientras que **Episodio** y **Pelicula** son sus clases derivadas.

La clase **Video** se define como abstracta porque proporciona una base común para todos los tipos de videos sin necesidad de ser instanciada directamente. Esta clase incluye atributos esenciales como **ID**, **nombre**, **duracion** y **genero**, los cuales son esenciales tanto como para episodios como para películas.

Adicionalmente, también se tiene a la clase **Serie**, que se encarga de agrupar todos los episodios a través de vectores para cada objeto, además de tener otros atributos y métodos propios.

```cpp
class Video {
private:
  string nombre, genero;
  double duracion;
  vector<int> calificaciones;
  string ID;
```

```cpp
class Episodio : public Video {
  private:
  int temporada, episodio;
  string titulo;
```

```cpp
class Pelicula : public Video {
public:
  Pelicula(string, string, string, double);
  virtual ~Pelicula();

private:
};
```

```cpp
class Serie {
  private:
  string id;
  string nombre;
  vector<Episodio> episodios;
  int calificacion;
```

b) Herencia

Como se mencionó anteriormente, tanto los episodios como las películas son tipos de video, y por lo tanto comparten ciertas características. Bajo este razonamiento, se decidió que las clases encargadas de gestionar a éstos mismos, tendrían que ser heredadas de **Video**, para así poder conservar sus atributos, y agregar más de ser necesario.

La herencia también permite heredar los métodos de **Video**; sin embargo, al declarar como virtual a algunos como **mostrar()**, también facilitó la sobreescritura de estos mismos.

A diferencia de los métodos y atributos, los constructores no se heredan, por lo que fue necesario definir constructores específicos en cada clase derivada para inicializar adecuadamente los atributos heredados de la clase base. Esto implica que, aunque las subclases **Pelicula** y **Episodio** heredan los atributos como **ID**, **nombre**, **duracion** y **genero** de la clase Video, cada una debe tener su propio constructor para asignar valores a estos atributos al momento de crear instancias de las clases derivadas.

c) Modificadores de acceso

En todas las clases utilizadas por este programa, las variables se declaran como privadas, mientras que los métodos e inicializadores se declaran como públicos.

d) Sobrecarga y sobreescritura de métodos

En la clase base **Video** se tiene una función virtual **mostrar()** que está inicializada para imprimir el Id, nombre, género, duración y calificación de un objeto de la clase **Video**; mientras que la clase **Episodio** sobreescribe esta misma clase para imprimir el número de episodio, la temporada, el título y la calificación de este.

e) Polimorfismo

A través del programa se utiliza el polimorfismo en distintos momentos.
Primeramente, se inicializan dos **apuntadores** a vectores de las clases **Serie** y **Pelicula**.

En los métodos **importarDatos(), videos_calif_genre(), sobreescribirDatos(), calificarVideo()** y **mostrarCalifs()** del **main**, se reciben **apuntadores** de vectores de la clase **Serie** y **Pelicula**, mientras que el método **series_episodios_calif()** se recibe un apuntador de un vector de la clase **Pelicula**, y en el método **peliculas_calif()**, se recibe un apuntador de un vector de la clase **Series**.

También es importante mencionar que en la mayoría de los bucles 'for', se utilizan bucles **'for' basados en intervalos**, que utilizan **apuntadores** o direcciones de memoria para iterar por los objetos.

Esto se utiliza para poder iterar de forma más fácil por cada elemento de los apuntadores de vectores se las clases Series y Película.

Más adelante también se establece la sobrecarga de operadores como polimorfismo estático, además del polimorfismo dinámico establecido en este rubro.

f) Clases Abstractas

La clase base **Video** es una clase abstracta, ya que involucra la **función virtual mostrar()**, que más tarde se sobreescribe por la clase heredada Episodio, como se estableció anteriormente.

g) Se sobrecarga al menos un operador en el conjunto de clases propuestas

Como sobrecarga de operadores (que también entra dentro de polimorfismo estático), se sobrecarga el **operador '+'** en la clase **Video**, y se utiliza para agregar una calificación al vector de calificaciones de la clase.

Identificación de casos que harían que el proyecto deje de funcionar

Al comenzar el programa es necesario que el usuario cargue un archivo de texto, pues sin este, no es posible utilizarlo. Sería necesario realizar una validación, en la que si no hay un archivo importado, entonces el programa se aborte. También sería factible incluir un archivo de texto por default.

Conclusiones

Marissa: el proyecto presentado no solo ofrece una estructura sólida para la gestión de videos en un servicio de streaming, sino que también destaca la importancia de aplicar principios fundamentales de la programación orientada a objetos como herencia, polimorfismo, abstracción y encapsulación. Estos principios permiten crear un sistema flexible y escalable que facilita la incorporación de nuevas funcionalidades y la mejora continua del servicio. Personalmente, considero que la integración de estos conceptos es crucial para desarrollar soluciones tecnológicas eficientes, capaces de adaptarse a las cambiantes demandas del mercado y proporcionar una experiencia de usuario superior.

Luis Ignacio: A través de este proyecto pude aprender a manejarme más eficazmente a través de usar memoria y polimorfismo dinámico para poder traer información que se guardan en archivos y modificar esta misma información a través de todo el programa.
Me pareció también muy interesante aprender a utilizar regex para poder separar las líneas de información del archivo, así como aprender a utilizar bucles ‘for’ basados en intervalos para poder iterar por los vectores de Serie y Pelicula de forma más dinámica.

Juan Pablo: Esta situación problema fue de gran apoyo para enfatizar la mayor parte de los conocimientos vistos a lo largo del periodo, juntos en un mismo programa enfocado a una problemática real; problemática que dio paso a realizar una abstracción de los elementos propios de la naturaleza del ejercicio, relacionándolos y utilizándolos dentro del código. El planteamiento representó un parteaguas para la estructura y camino del proyecto, base sobre la que se construyó todo los demás funcionamientos. Para el funcionamiento del programa hizo falta un gran número de funciones, tanto de acceso a atributos como de interconexión entre clases. La situación problema representó además un reto extra, por la necesidad de implementar métodos nuevos para la mayoría de nosotros, como lo fue la carga de archivos.

Referencias

TylerMSFT. (2023a, April 3). *Clases abstractas (C++)*. Microsoft Learn. https://learn.microsoft.com/es-es/cpp/cpp/abstract-classes-cpp?view=msvc-170

Guzman, H. C. (n.d.). *Herencia de clases | Apuntes lenguaje C++ | Hektor Profe*. https://docs.hektorprofe.net/cpp/11-clases/05-herencia-clases/