# TC 1030.1 – Programación Orientada a Objetos

Tecnológico de Monterrey

Evidencia Situación Problema: Modelado de servicio de *streaming*

Nombre del alumno: Luis Eduardo Gutiérrez Saenz Pardo

Matrícula: A01283825

Profesor: David Alonso Cantú Delgado

12/06/20

**ÍNDICE**

## INTRODUCCIÓN

Las plataformas de streaming cada vez son más populares a nivel mundial, plataformas como Netflix, Amazon Prime Video, Disney +, entre otros han tomado el liderato del mercado del video bajo demanda, contando con enormes catálogos de series, películas y documentales de manera digital. El uso de sistemas computacionales para la organización, despliegue e interacción con dichas cantidades de información sobre los videos bajo demanda es esencial para el éxito de estas compañías, las cuales han buscado innovar en sistemas cada vez más avanzados para llevar a cabo tareas como la recomendación de títulos, asignar calificaciones y sortear categorías.

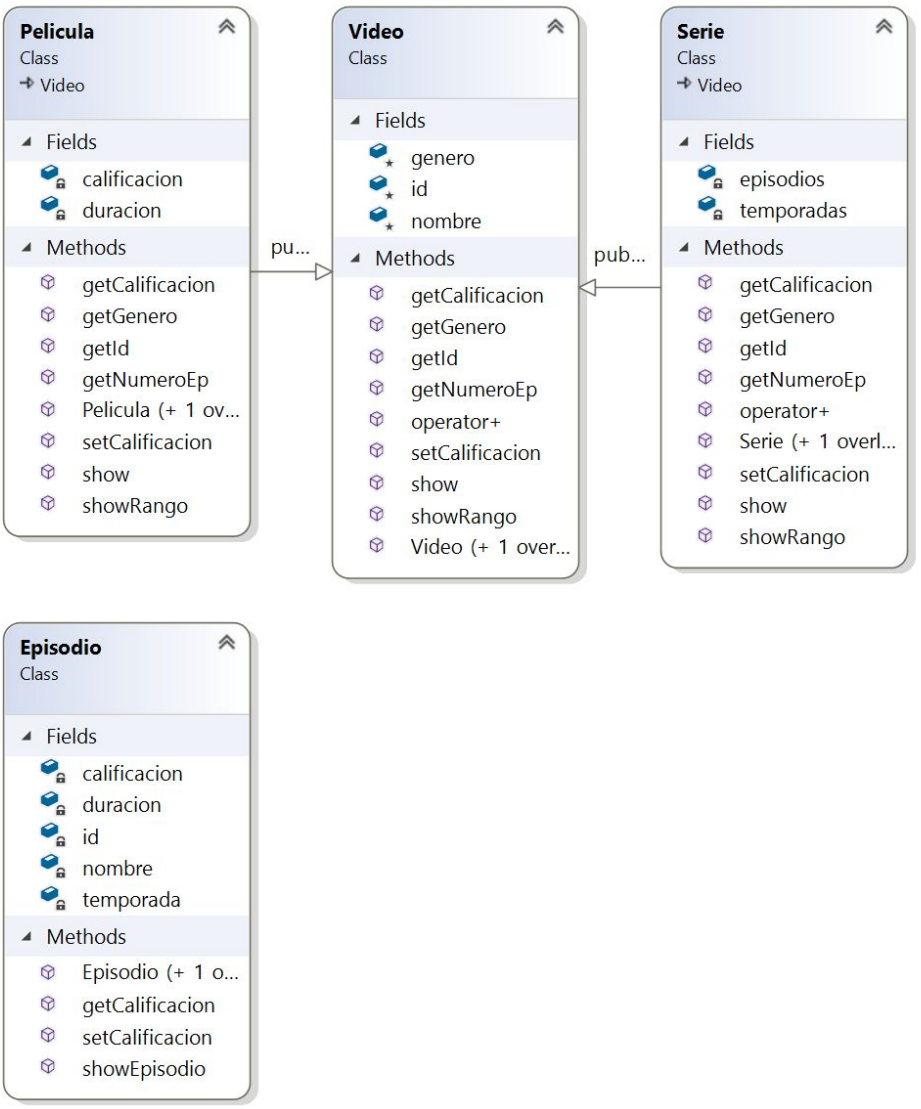
El reto que se presenta para esta evidencia es el de llevar a cabo una simulación limitada a las funcionalidades más básicas de un servicio de streaming mediante un sistema computacional programado en C++, que permita buscar por categorías, por calificaciones y asignar calificaciones a los diferentes tipos de series y películas de una base de datos.

Las instrucciones básicas del programa requerido son:

1. Mostrar los videos en general con sus calificaciones
2. Mostrar los episodios de una determinada serie con sus calificaciones
3. Mostrar las películas con sus calificaciones
4. Establecer filtros por género y rango de calificación
5. Cambiar y asignar calificaciones

Para este proyecto, es preciso hacer uso de los conceptos de la programación orientada a objetos, haciendo uso de sus propiedades principales de herencia, abstracción, encapsulación y polimorfismo.

## DIAGRAMAS DE CLASES




Decidí hacer uso del concepto de herencia creando una clase Video, la cual es una clase abstracta, es decir que su función no es instanciar objetos, sino que establecer métodos virtuales los cuales mediante polimorfismo fueran usados por la clases hijas Serie y Película. Los atributos de la clase Video tienen un modificador de acceso de tipo protected, esto permite que solo las clases que heredan a esta puedan acceder de manera directa a estos atributos. De igual forma se estableció una clase Episodio, la cual nos serviría para instanciar objetos de tipo Episodio para después ser almacenados en un vector de episodios en la clase Serie.

Se escogió este diseño de clases, para así tener un mayor control sobre los métodos de los objetos y poder acceder de manera sencilla, ahorrar código y repetir procesos.

## EJEMPLO DE EJECUCIÓN

```
------------MENU-------------
1) Cargar archivos de datos
2) Mostrar videos con rango de calificacion y genero
3) Mostrar videos de genero especifico
4) Mostrar episodios de serie con rango de calificacion
5) Mostrar peliculas con rango de calificacion
6) Calificar un video
0) Salir
Opcion:
```

```
Seleccione un genero
1)DRAMA 2)ACCION 3)MISTERIO: 1

Nombre de Serie: Game of Thrones  Genero: Drama
    Episodios:
      [1]
       Nombre de Episodio: Winter is coming  Temporada: 1
       Duracion: 1:02 Calificacion: 9.1
      [5]
       Nombre de Episodio: The Wolf and the Lion  Temporada: 1
       Duracion: 0:55 Calificacion: 9.1
      [6]
       Nombre de Episodio: A Golden Crown  Temporada: 1
       Duracion: 0:53 Calificacion: 9.2
      [7]
       Nombre de Episodio: You Win or You Die  Temporada: 1
       Duracion: 0:58 Calificacion: 9.2
      [8]
       Nombre de Episodio: The Pointy End  Temporada: 1
       Duracion: 0:59 Calificacion: 9
      [9]
       Nombre de Episodio: Baelor  Temporada: 1
       Duracion: 0:57 Calificacion: 9.6
      [10]
       Nombre de Episodio: Fire and Blood  Temporada: 1
       Duracion: 0:53 Calificacion: 9.5
```

```
Nombre de Pelicula: Ad Astra  Genero: Drama
Duracion: 2:03 Calificacion: 6.6

Nombre de Pelicula: The Platform  Genero: Drama
Duracion: 1:34 Calificacion: 7

Nombre de Pelicula: Star Wars: The Rise of Skywalker  Genero: Acci├│n
Duracion: 2:22 Calificacion: 6.7

Nombre de Pelicula: The Lion King  Genero: Drama
Duracion: 1:58 Calificacion: 6.9

Nombre de Pelicula: Onward  Genero: Accion
Duracion: 1:42 Calificacion: 7.5

Nombre de Pelicula: Extraction  Genero: Drama
Duracion: 1:56 Calificacion: 6.8

Nombre de Pelicula: Beauty and the Beast  Genero: Accion
Duracion: 2:09 Calificacion: 7.1
```

## ARGUMENTACIÓN DE LA SOLUCIÓN

Primero, referente a las clases utilizadas, las instrucciones especifican que teníamos que implementar una solución para videos, los cuales podrían ser series con un conjunto de episodios o películas. Es por esto que decidí utilizar una clase abstracta Video, de la cual heredan sus atributos las clases Película y Series. Esto permitiría reciclar los métodos mediante el uso de polimorfismo y la sobrecarga de métodos, ya que muchos eran muy parecidos para ambas clases. De igual forma se identificó una clase Episodio, la cual nos permitiría instanciar objetos individuales con los datos de cada episodio para cada serie específica, y así poder acceder a sus atributos y métodos de manera sencilla. Esta división permitió una solución que permitiera crear un conjunto de episodios como atributo de la clase Serie mediante el establecimiento de un vector de episodios de la clase Episodio.

Al hacer uso de herencia en estas clases, se logró reciclar código mediante el uso de polimorfismo, esto resulta una buena solución ya que nos ahorra tiempo en la implementación de la solución. De igual forma resultó una buena solución ya que esto permitió juntar ambos tipos de video en un solo vector, lo cual facilitó en gran medida hacer referencia a sus métodos y atributos. A comparación con otros diseño de solución, esta solución permite mantener un mayor control sobre las variables y los objetos que hacen referencia a atributos y métodos similares entre las clases Película y Series. Junto con esto, los modificadores de acceso adecuados en el uso de herencia, lograron mantener una protección de los atributos de la clase padre, estableciendo un modificador de accesos de tipo protected.

La sobrecarga de métodos y el polimorfismo fueron esenciales para acceder a los métodos requeridos para cada tipo de clase, ya que ambas estaban en el mismo vector. Esto se logró mediante los métodos virtual establecidos en la clase abstracta Video, en donde los métodos eran heredados a las clases hijas, de esta forma según el tipo de objeto cambiaba el tipo de método que se realizaba a pesar de estar definido de la misma forma a través de las clases.

Referente a la sobrecarga de operadores, esta fue esencial para simplificar la escritura del código de algunos métodos. Por ejemplo para agregar episodios al vector episodios de una serie se hizo uso de la sobrecarga del operador +. Esto ayudó a mantener el código más organizado y facilitar su entendimiento.

Finalmente se usaron excepciones para validar los métodos de lectura de archivos, esto ayuda en gran medida a que si existe un problema en la lectura del archivo, el programa no se detenga en un error o *crash*.

## IDENTIFICACIÓN DE CASOS QUE HACEN QUE EL PROGRAMA DEJE DE FUNCIONAR

Para esta solución se buscó validar todos los tipos de datos de entrada del usuario, para así mantener coherencia entre ellos. Se hizo uso de la validación de integridad de los datos, por ejemplo al pedir un valor numérico, si se pone una letra el usuario es notificado de su error.

De igual manera se validó que los índices introducidos por el usuario estuvieran en los límites de los tamaños de los vectores empleados. Así como también se hizo uso de excepciones para evitar *crashes* en el código al leer los archivos. De igual forma se hizo una validación que no se pudiera cargar los archivos dos veces, ya que esto duplicaría los datos, y que no se pudiera acceder a otros métodos hasta cargar los archivos.

Si bien estos datos están validados y hacen que el programa en casi el cien por ciento de los casos no deje de funcionar. Sin embargo, existen datos que al ser introducidos por el usuario pueden afectar a la lógica del programa, por ejemplo calificaciones negativas o mayores a 10 o  rangos inferiores mayores a los rangos superiores

## CONCLUSIONES

En conclusión personal, creo que este proyecto fue una gran implementación de los conocimientos sobre la programación orientada a objetos. Creo que fue un ejemplo práctico muy adecuado el cual permitió implementar una solución computacional apegada al rigor adecuado para este curso.

Me siento satisfecho con mi trabajo, ya que pude aprender diferentes conceptos y aplicarlos en una solución integradora. Creo que lo que más me llevo de este proyecto y este curso es el hecho de saber cómo diseñar una solución con base a la programación orientada a objetos.