Tecnológico de Monterrey

Proyecto Integrador

(Propuesta de solución a Situación Problema)

Programación Orientada a Objetos (F1030)

Grupo:

301

Profesora:

Rosa Guadalupe Paredes Juárez

Alumno y Matrícula:

Juan Pedro Medina Silva -A01662202

Fecha:

15 de junio de 2022

TABLA DE CONTENIDOS:

1. Introducción (planteamiento del problema)

Actualmente, muchas compañías grandes de televisión o de cine han buscado crear nuevos servicios de streaming de video bajo demanda, generando popularidad alrededor del mundo. A estos servicios de streaming se le puede realizar un modelado para poder prepararnos por si en un futuro cercano una compañía proveedora de este tipo de servicios llegara a necesitar nuestra ayuda.

Para el modelado se va a buscar mantener dos presentaciones del formato de Video, que serán Movie y Serie. Ambas van a tener las siguientes características: ID, nombre, duración, género, y calificaciones (del 1 al 5). Van a existir dentro de Series un grupo de episodios que van a tener, las características de título y temporada en la que está. Dentro de nuestra plataforma va a haber una interfaz que va a ser capaz de mostrar los videos en general con sus calificaciones, mostrar los episodios de una determinada serie con sus calificaciones, y mostrar las películas con sus calificaciones.

Para poder satisfacer a nuestro futuro cliente, vamos a tener que utilizar los conocimientos actuales que disponemos sobre la programación orientada a objetos. Primero debemos de realizar un diagrama de clases UML para poder comenzar a establecer que va a ser parte de nuestro modelo y nos ayudará a realizar con mayor facilidad el código.

2. Diagrama de clases UML con una argumentación del por qué del diseño

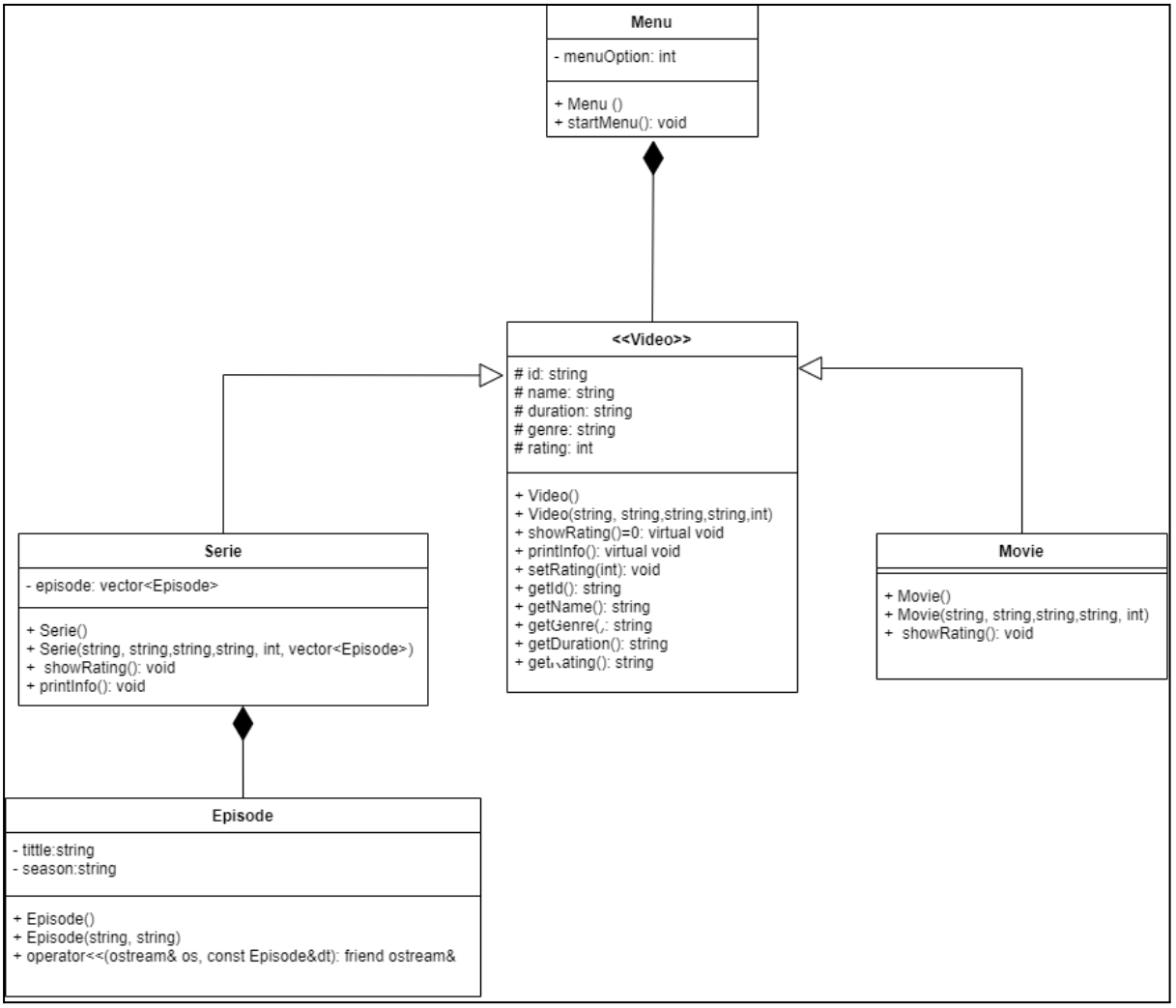

FIGURA.1 Diagrama de Clases UML del Reto.

Se utilizó este diseño debido a que evidentemente en la situación problema podemos evidenciar de que la clase Serie y Movie heredan tanto atributos como métodos de la clase Video. La clase Video además es una clase abstracta debido a que presenta una función virtual pura y sirve como base para las clases que le heredan.

La herencia entre Serie y Movie con Video facilita que no se tengan que repetir líneas de código por la cantidad de métodos y atributos que comparten. Existen dos relaciones de composición en el diagrama, el primero es Episode con Serie y esta se aplica debido a que sin la clase Serie no puede existir Episode porque básicamente no puede existir un episodio que

no sea parte de una Serie. La segunda relación de composición se presenta entre Menu y Video, esta se da igual de que sin Menú no hay Video. Este es el diseño más simple y que mejor se ajustó a nuestra situación problema.

3. Liga de Replit:

   https://replit.com/join/ofbmbvoycx-juanpepe12

4. Ejemplo de ejecución (capturas de pantalla)

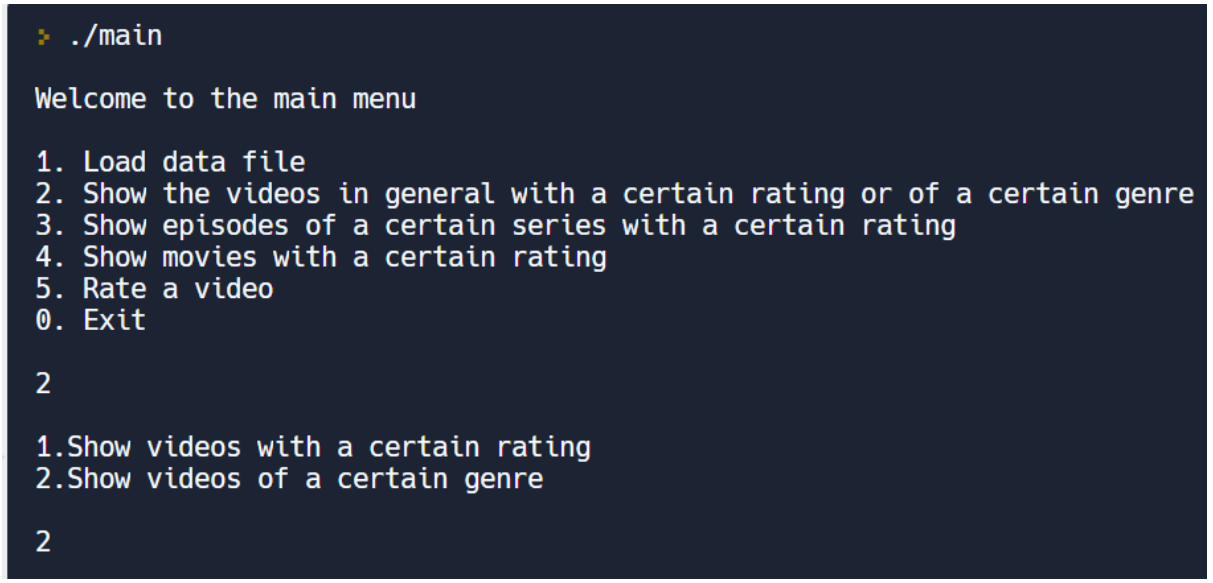

FIGURA. 2 Ejecución de código secuencia 1 de 8.

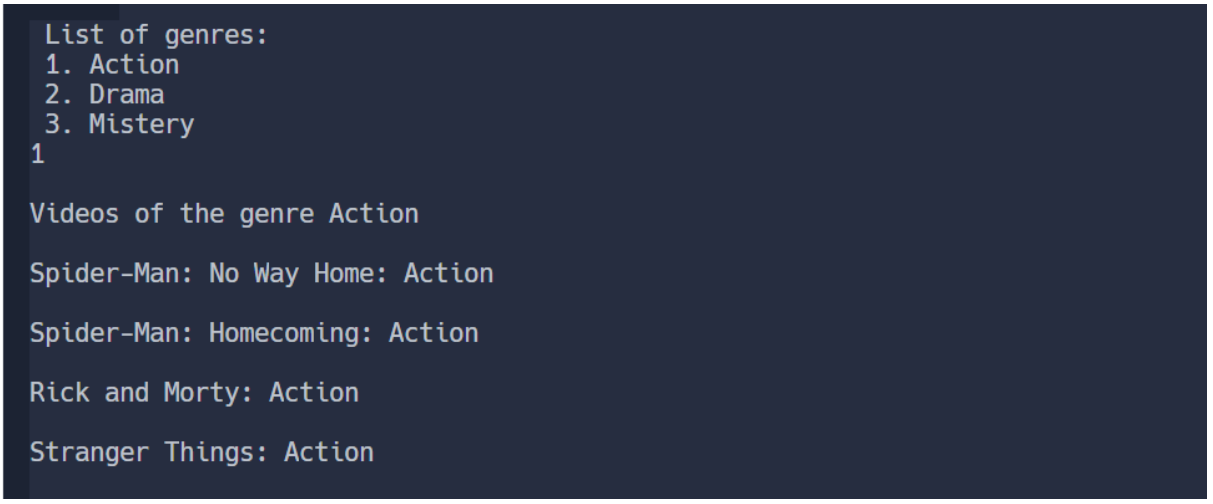

FIGURA. 3 Ejecución de código secuencia 2 de 8.

```
Welcome to the main menu

1. Load data file
2. Show the videos in general with a certain rating or of a certain genre
3. Show episodes of a certain series with a certain rating
4. Show movies with a certain rating
5. Rate a video
0. Exit

2

1.Show videos with a certain rating
2.Show videos of a certain genre

1
```

FIGURA. 4 Ejecución de código secuencia 3 de 8.

```
Show videos rated (0-5) by: 5

Videos with the rating of 5

The movie The Godfather has a rating of 5/5

The serie Breaking Bad has a rating of 5/5

The serie Grey's Anatomy has a rating of 5/5
```

FIGURA. 5 Ejecución de código secuencia 4 de 8.

```
Welcome to the main menu

1. Load data file
2. Show the videos in general with a certain rating or of a certain genre
3. Show episodes of a certain series with a certain rating
4. Show movies with a certain rating
5. Rate a video
0. Exit

4
Show movies rated (0-5) by:
4

Movies with the rating of 4

Spider-Man: No Way Home: The movie Spider-Man: No Way Home has a rating of 4/5

Scarface: The movie Scarface has a rating of 4/5
```

FIGURA. 6 Ejecución de código secuencia 5 de 8.

```
Welcome to the main menu

1. Load data file
2. Show the videos in general with a certain rating or of a certai
3. Show episodes of a certain series with a certain rating
4. Show movies with a certain rating
5. Rate a video
0. Exit

5
```

FIGURA. 7 Ejecución de código secuencia 6 de 8.

```
Decide which one to change depending on the titles

0) The movie Spider-Man: No Way Home has a rating of 4/5
1) The movie Spider-Man: Homecoming has a rating of 3/5
2) The movie The Godfather has a rating of 5/5
3) The movie Scarface has a rating of 4/5
4) The movie Sherlock Holmes has a rating of 3/5
5) The movie Arrival has a rating of 1/5
6) The serie Breaking Bad has a rating of 5/5
7) The serie Game of Thrones has a rating of 2/5
8) The serie Rick and Morty has a rating of 3/5
9) The serie Prision Break has a rating of 4/5
10) The serie Stranger Things has a rating of 0/5
11) The serie Grey's Anatomy has a rating of 5/5
10
Enter the new rating of the title #10
2
The serie Stranger Things has a rating of 2/5
```

FIGURA. 8 Ejecución de código secuencia 7 de 8.

```
Welcome to the main menu

1. Load data file
2. Show the videos in general with a certain rating or of a certain genre
3. Show episodes of a certain series with a certain rating
4. Show movies with a certain rating
5. Rate a video
0. Exit

0

Closing Menu.
Have a great day
```

FIGURA. 9 Ejecución de código secuencia 8 de 8.

En la ejecución del código representado en las FIGURAS 2-9, se puede observar una secuencia guiada donde se viaja por todas las “ventanas” del menú donde se observa que no hay fallas e imprime lo deseado.

5. Argumentación de los criterios evaluados en la evidencia.

Punto a. Se identificó correctamente las clases a utilizar.

En las primeras semanas de clases se realizó un esqueleto para nuestro diagrama de Clases UML, donde presentamos las clases a utilizar y el tipo de relaciones que presentaban con las demás. Siendo las clases establecidas, Video, Serie, Movie, y Episode; en el que las clases Serie, Movie heredan de la clase Video y Episodio, tiene una relación de composición con la clase Serie. A estas clases se les agregó una llamada Menú, donde va a estar la interfaz del menú con sus distintas opciones a escoger.

Punto b. Se emplea de manera correcta el concepto de herencia.

La herencia es la capacidad de una clase de derivar propiedades y características de otra clase. En el código se presenta este concepto con las clases de Serie y Movie, que heredan los métodos y atributos de la clase padre Video.

```
10
11 ▼ class Video{
12   protected:
13   /* data */
14       string duration;
15       string id;
16       string name;
17       int rating;
18       string genre;
19   public:
20   /* methods */
21     Video();
22     Video(string,string,string,string,int);
23     virtual void showRating()=0;
24     virtual void printInfo();
25     void setRating(int);
26     string getId();
27     string getName();
28     string getGenre();
29     string getDuration();
30     int getRating();
31   };
```

FIGURA. 9 Clase Padre (Video) en la situación Problema.

```
class Serie:public Video {
private:
/* data */
    vector<Episode> episodes;
public:
/* methods */
    Serie();
    Serie(string,string,string,string,int, vector<Episode>);
    void showRating();
    void printInfo();


};
```

FIGURA. 10 Clase Hija (Serie) en la situación Problema.

```
class Movie:public Video {
 private:
 /* data */
 public:
 /* methods */
     Movie();
     Movie(string,string,string,string,int);
     void showRating();

 };


 #endif /* Movie_h */
```

FIGURA. 11 Clase Hija (Movie) en la situación Problema.

En toda la ejecución del código se puede ver como las clases Serie y Movie heredan métodos y atributos de serie. Se puede evidenciar principalmente en que la mayoría de los atributos que utilizan ambas clases son heredados de serie (name, id, genre, duration, rating).

Punto c. Se emplean de manera correcta los modificadores de acceso.

En todas las clases que no son clases padres (Video) en la relación de herencia utilizan modificadores de acceso protegidos en todos sus atributos para que sus hijas puedan tener acceso, mientras que el resto de las clases usan en los atributos el modificador de acceso private (solo acceso para ellos). En los dos casos mencionados previamente se utilizaron en los métodos el modificador de acceso public para poder llamar a estos métodos desde

cualquier clase o en el main. Se puede observar mejor en la FIGURA.1, el uso de los modificadores de acceso.

Punto d. Se emplea de manera correcta la sobrecarga y sobrescritura de métodos.

En el código se emplea la sobrecarga de métodos debido a que en la clase Padre Video presentamos con un constructor llamado Video que contiene todos los atributos de la clase a declarar, pero también existe un método setter que le cambia el valor a la calificación de cualquier Video. Se utiliza la sobreescritura de métodos debido a que la clase Serie hereda de la clase Video el método de printInfo(), pero cada uno de las clases realiza este método a su manera.

```
void Video::printInfo(){
  cout<<name<<"Episode";
}
```

FIGURA. 12 Clase Padre (Video) manera de usar printInfo().

```
void Serie::printInfo(){
  cout<<"Episodes of "<< name<<": \n";
  for (int i=0; i<episodes.size();i++){
    int z=i+1;
    cout<<z<< ")" <<episodes[i];
  }
```

FIGURA. 13 Clase Hija (Serie) con nueva manera de usar printInfo().

Se puede observar en la Figura 11-12, que en la sobreescritura de métodos se utiliza el mismo nombre del método pero con otra manera de realizarlo. Se utiliza de manera correcta la sobrecarga y sobreescritura de métodos.

Punto e. Se emplea de manera correcta el concepto de Polimorfismo.

El concepto de Polimorfismo como la herencia nos permite heredar atributos y métodos de otra clase, el polimorfismo usa estos métodos para realizar diferentes tareas. Por lo que te permite realizar una misma acción de diferentes maneras. En el reto se aplicó utilizando el método mencionado anteriormente void printInfo(), sobreescrito en la clase Serie, pero el método showRating() presente en la clase Video realiza distintas acciones en la clase Serie y Movie. Demostración a continuación:

```
void Serie::showRating(){
    cout<< "The serie "<<name<<" has a rating of " <<rating<<"/5";
}
```

FIGURA. 14 Polimorfismo en la clase Serie.

```
void Movie::showRating(){
    cout<< "The movie "<<name<<" has a rating of " <<rating<<"/5";
}
```

FIGURA. 15 Polimorfismo en la clase Movie.

Punto f. Se emplea de manera correcta el concepto de Clases Abstractas.

En el diagrama de clases UML FIGURA.1, se representó la clase Video como clase abstracta. Las clases abstractas no se pueden instanciar, se usan generalmente como una base para clases hijas, y necesitan una función virtual pura para poder ser abstracta.

```
virtual void showRating()=0;
```

FIGURA. 16 Función virtual pura en la clase Video.

En la FIGURA.16 se puede observar la clase virtual pura presente en la clase Video y si vemos a detalle el código jamas instanciamos a la clase Video, porque la usamos como molde de nuestras clases Serie y Movie.

Punto g. Se sobrecarga al menos un operador en el conjunto de clases propuestas.
En la clase Episode se aplica la sobrecarga de operadores para poder mostrar en la pantalla los episodios presentes en cada Serie. Se sobrecargó el operador " << " y podemos observar a continuación como se hizo

```
ostream& operator<< (ostream& os, const Episode&dt){
  os << dt.season <<' '<< dt.title<<endl;
  return os;
}
```

FIGURA. 17 Sobrecarga de operador << en la clase Episode.

6. ¿Por qué se eligió esta solución y no otras?

Esta solución la escogí porque me pareció efectiva, cumple con todos los aspectos exigidos en el modelado, y se acopla a mis conocimientos en el tema. Además lo establecí desde la semana 2 (realizando el diagrama de clases UML del reto), por lo que la idea a resolver mediante está solución no es nueva, sino que lleva semanas en pie y no cambio mucho. Comparado con otras ideas, capaz no es la mejor aplicación de los conocimientos o es la más eficiente pero así fue como yo lo plantee y considero que lo resolví de manera muy veloz. La mejor manera de solucionar este problema es con la Programación Orientada a Objetos, porque te ahorras líneas de código y las cualidades que esta metodología presenta te ayuda a hacer el trabajo más fácil.

7. Identificación de casos que harían que el proyecto deje de funcionar

No es que el código dejará de funcionar, sino que no va a enviar nada de regreso si pide un valor de calificación y ningún Video tiene aquel valor en sus datos. Además el código está diseñado para que el usuario otorgue una calificación del 1 al 5 a los Videos que quiera calificar, porque es como la metodología de las estrellas.

8. Conclusión

Para describir cómo funciona algunas funciones presentes en un servicio de streaming utilizando Programación Orientado a Objetos, el diagrama de clases UML como el código realiza lo que se le exige para este modelado. Me parece que este reto fue una gran manera de trabajar, reforzar, y aplicar los conocimientos aprendidos en clase. Considero que la entrega de manera individual de esta situación problema me sorprendió un poco al principio, pero ya culminado el proyecto estoy seguro que me ayudó a obtener las subcompetencias que me pudieron haber faltado desarrollar en la materia de Pensamiento computacional orientado a objetos. Descubrí que los conocimientos obtenidos en la materia F1030, programación orientada a objetos, pueden ser aplicados a cualquier situación de la vida cotidiana y que en el futuro me serán muy útiles, por lo que hay que seguir trabajando y aprendiendo por cuenta propia.

9. Referencias consultadas

- Parewa Labs Pvt (s.f). C++ Vectors. https://www.programiz.com/cpp-programming/vectors
- GeekforGeeks (12 junio,2022). Inheritance in C++. https://www.geeksforgeeks.org/inheritance-in-c/
- GeekforGeeks (28 enero,2022).Polymorphism in C++.https://www.geeksforgeeks.org/inheritance-in-c/
- w3schools (s.f.).Java Polymorphism. https://www.w3schools.com/java/java_polymorphism.asp