Instituto Tecnológico de Estudios Superiores de Monterrey
Campus Santa Fe

Programación orientada a objetos

Ariel Lucien García Gamboa

Proyecto Integrador

(Propuesta de solución a Situación Problema)

Javier Corona Del Río
A01023063

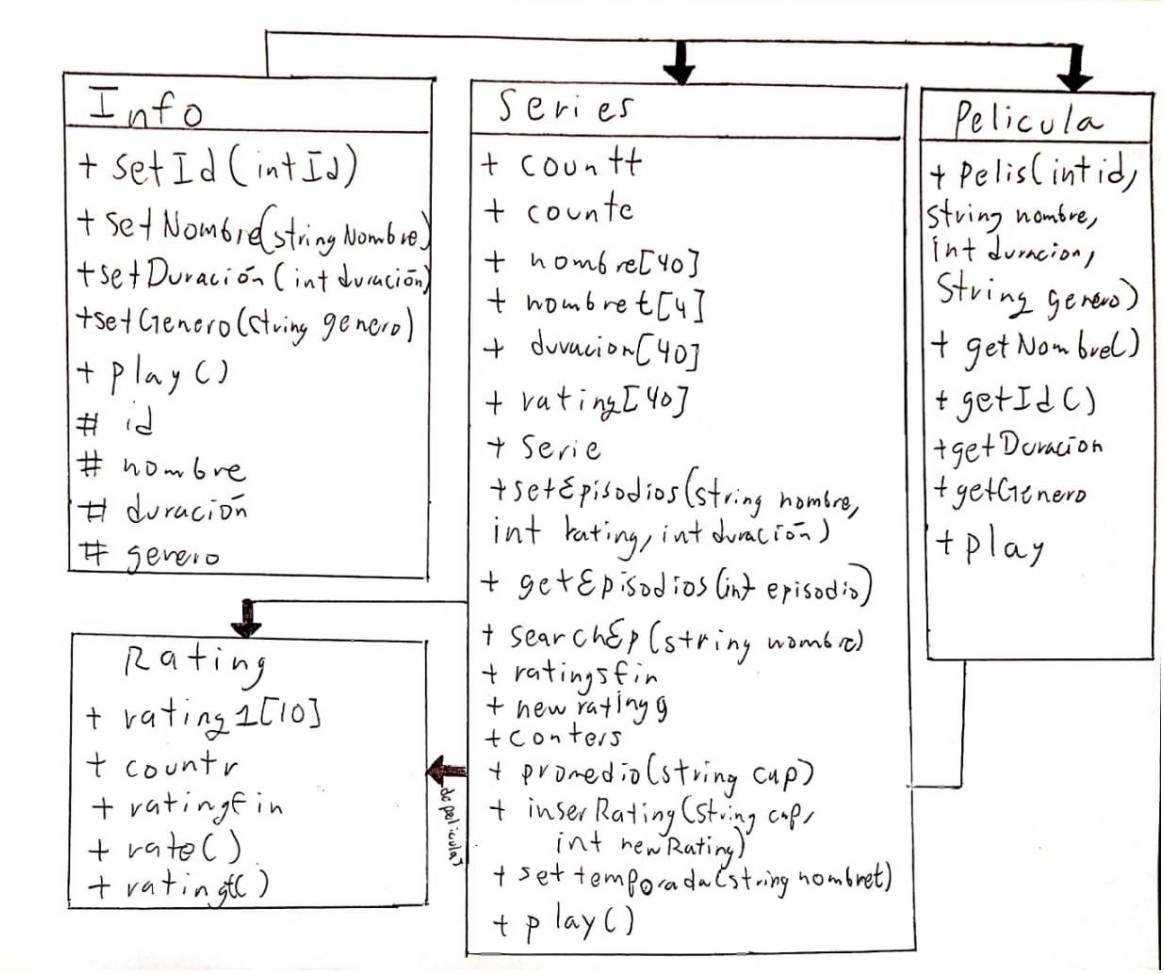
12-Junio-2020

Índice de contenido

Introducción

Para la resolución del problema se hizo un modelado de un servicio de streaming en dónde se debe ocupar conceptos de herencia, modificadores de acceso, sobrecarga, sobrescritura de métodos, polimorfismo, clases abstractas, conjunto de clases, el uso de excepciones predefinidas y definidas. El modelado tiene que contener videos, episodios de una serie y películas con sus determinadas calificaciones. Con el fin de desarrollar de buena manera lo anteriormente mencionado se debía hacer primero un análisis del planteamiento del problema, más adelante identificar objetos, definir características y comportamientos, posteriormente crear un diseño de clases y para más organización elaborar un diagrama UML, para finalmente programar la lógica, agregar las clases dichas en el UML y crear la solución con las condiciones determinadas. Todo esto para tener un prototipo de un tipo Netflix!

Diagrama de clases UML



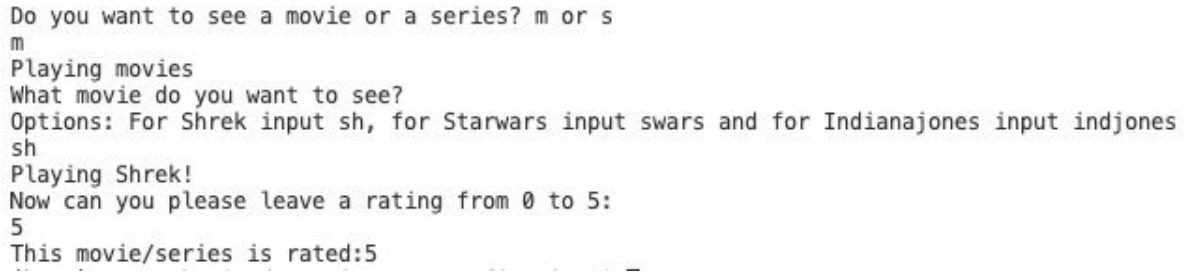
Se me ocurrió este diseño porque se me facilitaba mandar Info.h a series para heredar clases que las series necesitan para su conformación y definir diversas clases para que se pueda tener en esta clase los capítulos. También se derivó la clase abstracta play( ).

En el otro lado para la clase película mande de las misma manera la información primordial de las películas para posteriormente declararlas en el main, de igual manera recibe play( ).

Las clases película y serie necesitan de la clase rating para crear una conexión en el main y definir su promedio final con los datos preestablecidos y la información dada por el usuario.

Ejecución del programa

- **Para película:**



Cuando llega la toma de decisión entre ver una película y una serie el usuario debe poner m para cumplir la condición y poder elegir una película. Cuando esto pasa la clase abstracta se imprime diciendo que el usuario está viendo una película con “Playing movies”. Después el usuario se encuentra entre ver Star Wars, Indiana Jones y la mejor opción obviamente, ver Shrek. Cuando hay una elección de camino por parte del usuario, este se encuentra supuestamente viendo la película. Cuando este termina supuestamente de verla, le sale la opcion de darle una calificación dependiendo su gusto. Cuando esto pasa sale la calificación total de la película y finaliza el código.

- **Para series:**

```
Do you want to see a movie or a series? m or s
s
Playing series
What series do you want to see?
Options: For Suits input suts and for Friends input frends
suts
Which season of Suits you want to see?
Options t1, t2, t3, t4
t1
What episode you want to see?: one to teen
three
Playing episode, three
Leave a rating for the episode, please
4
Thanks
Now the rating is:
3
```

Cuando llega la toma de decisión entre ver una película y una serie el usuario debe poner s para cumplir la condición y poder elegir entre la variedad de series que hay predeterminadas. Pasando la primera decisión se le deja al usuario escoger entre ver Suits y Friends. El usuario debe escoger, cuando lo hace debe de escoger de nuevo entre qué temporada ver de la temporada uno a la cuatro. Cuando se escoge la temporada que se quiere ver, de igual manera se debe elegir, ahora entre qué episodio quiere ver el usuario. Se puede escoger del episodio uno al diez. Ya que se haya escogido, supuestamente ahora el episodio está siendo visualizado por la audiencia y sale el nombre del episodio. Para evitar poner cualquier tontería inventada, hice una numeración con string de los episodios para ambos casos y todas las temporadas del uno al diez cómo fue antes mencionado. Cuando el usuario acaba de ver el episodio, este mismo debe calificar el episodio. Cuando lo hace, en la clase rating hay un contador para que se tiene una suma de las calificaciones dadas y la predeterminada, después se divide con las veces que se ha calificado este episodio. Con esto se hace la operación para determinar el nuevo rating del episodio y lo imprime. Aquí se acaba todo el proceso de ver un episodio en la temporada escogida y calificarlo.

Argumentación

a) Se identifican de manera correcta las clases a utilizar:

Considero que si se logró esto. Se puede uno quitar dudas viendo el diagrama UML. En esta imagen se ve en donde se usaron las clases y cómo se comprueba qué sirve y cumple con todas las condiciones pedidas, entonces se puede decir que si se implementan de buena manera las clases.

b) Se emplea de manera correcta el concepto de Herencia:

En la captura a un costado se ve cómo se implementa el concepto de herencia desde el h “Info”, hacía los archivos películas y series. Se hizo de esa manera para heredar las clases que escogí para no tener qué trancibirlas de nuevo porque en ambos casos se necesitan para la resolución del programa. Con poner un dos puntos y definir las clases que se quería arrastrar para correr en este archivo bastó para usar este concepto. Ejemplo : class Pelicula:public Info

```
#include <iostream>
using namespace std;
#include "info.h"

#pragma once
class Pelicula:public Info
{
public:
    void Pelis(int id, string nombre, int duracion, string genero)
    {
        setId(id);
        setNombre(nombre);
        setDuracion(duracion);
        setGenero(genero);
    }
```

```
#include <iostream>
#include <string>
using namespace std;
#include "info.h"


#pragma once
class Series:public Info
{
public:

    int countt = 0;
```

c) Se emplea de manera correcta los modificadores de acceso:

Considero que logré implementar esto a la perfección porque cómo se puede ver en el UML implemente modificadores de acceso públicos y protegidos. Desafortunadamente en este caso no encontré la utilidad de los privados, esto fue porque no fueron necesarios para la resolución del programa.

d) Se emplea de manera correcta la sobrecarga y sobreescritura de métodos:

Se implementó de manera correcta la sobrecarga y sobrescritura de métodos cuando se requería hacer el metodo para conseguir y determinar el promedio dependiendo lo dado por el usuario. El dato dado por el usuario se suma al promedio predeterminado y lo divide con el dato del contador, osea las veces qué se calificó, para printear el promedio y/o calificación del capítulo ó película.
Se adjunta prueba a un costado. En el lado derecho para las películas y abajo para los capítulos de las series.

```
void promedio(string cap)
{
    int id = searchEp(cap);
    cout<<ratingS[id]<<endl;
}

void insertRating(string cap, int newRating)
{
    int id = searchEp(cap);
    ratingS[id] = (ratingS[id]+newRating)/2;
}
```

```
void rate()
{
    cout << "Now can you please leave a rating from 0 to 5:" << endl;
    cin >> rating1[countr];
    while (rating1[countr] < 0 || rating1[countr] > 5)
    {
        cout << "Please a rating betwen 0-5, please:" << endl;
        cin >> rating1[countr];
    }
    countr++;

}

int ratingt()
{
    for (int i = 0; i < countr; i++)
    {
        ratingfin = ratingfin +rating1[i];
    }
    ratingfin = ratingfin/countr;
    cout << "This movie is rated:" << ratingfin << endl;
    return ratingfin;
}
```

e) (12 pts) Se emplea de manera correcta el concepto de Polimorfismo

Para conseguir usar el concepto de polimorfismo se usaron las clases con virtual. Haciendo que derive una clase base para posteriormente acceder a la función oculta de la base por su nombre completo. Se demostrará la implementación con una captura alado.

```
#pragma once
class Info
{
public:

    virtual void setId(int id)
    {
        id = id;
    }

    virtual void setNombre(string nombre)
    {
        nombre = nombre;
    }

    virtual void setDuracion(int duracion)
    {
        duracion = duracion;
    }

    virtual void setGenero(string genero)
    {
        genero = genero;
    }
```

f) (12 pts) Se emplea de manera correcta el concepto de Clases Abstractas:
Se utilizó una clase abstracta para definir cuando se hace una elección de camino, si se va a ver una película o una serie, ahi se imprimirá qué se está viendo dependiendo el caso. Todo esto pasa en el main. Este concepto se puso en la clase info para posteriormente definir su funcionalidad en serie y película. Se adjunta cómo se implementó en Info.h y su función en película.h.

```
void play()
{
    cout << "Playing movies" << endl;

}
```

```
void play()
{
    cout << "Playing movies" << endl;

}
```

Identificación de casos que harían que el proyecto deje de funcionar

- Sí en un cin se pone un string en vez de un int.
- Que el usuario quiera ver un episodio que no exista
- Cuando el usuario quiera ver una película que no esté.
- Cuando el usuario quiera ver una serie que no esté.
- Cuando el usuario ponga un espacio después del cin.
- Que el usuario use un valor double en vez de int para definir el promedio.
- Si el usuario se equivoca con la redacción en la decisión de ver qué película o serie ver.

Conclusión

Considero este proyecto fundamental para nuestra evolución como programadores es por eso me emocione mucho al empezar la preparación, diseñar mi UML y finalmente con todo esto programar la lógica y funciones para terminar. Estoy feliz porque logré entender los temas vistos en clase de herencia, poliformismo y apuntadores, los cuales me sirvieron para el desarrollo del código. Y también pude

aplicarlos de buena manera. Espero que siga desarrollando de buena manera como programador en los siguientes semestres y qué lo aprendido en este curso me sirva para esto. Me gusto mucho este proyecto, espero en un futuro pueda elaborar más iguales.

Referencias consultadas

Correa, C. (2020). Concepto de Herencia - Herencia en C++ (Práctica 3). Recuperado de, https://www.codingame.com/playgrounds/50747/herencia-en-c-practica-3/concepto-de-herencia

Stack Exchange Inc. (2017, 13 abril). Polimorfismo en C++. Recuperado de, https://es.stackoverflow.com/questions/62746/polimorfismo-en-c

w3schools. (2020). C++ Encapsulation and Getters and Setters. Recuperado de, https://www.w3schools.com/cpp/cpp_encapsulation.asp