Proyecto Integrador

Grupo 317

Daniel Sánchez Cataño [A01663268]

Marco Antonio García Mendoza [A01026487]

Ricardo André Mancera Ortega [A01664892]

Junio, 2023

ITESM - CCM

Programación Orientada a Objetos

Índice

- **Introducción**
- **Diagrama UML**
- **Ejemplo de Ejecución**
- **Argumentación del Código**
- **Identificación de casos que harían que no funcionase**
- **Conclusiones personales**

Introducción

En este proyecto es un sistema que gestiona películas, series y episodios, permitiéndonos explorar y calificar estos videos. A través de una interfaz de usuario interactiva, podemos realizar diversas acciones, desde cargar archivos de datos hasta buscar videos por calificación o género.

Se muestra un Menú con las siguientes opciones:

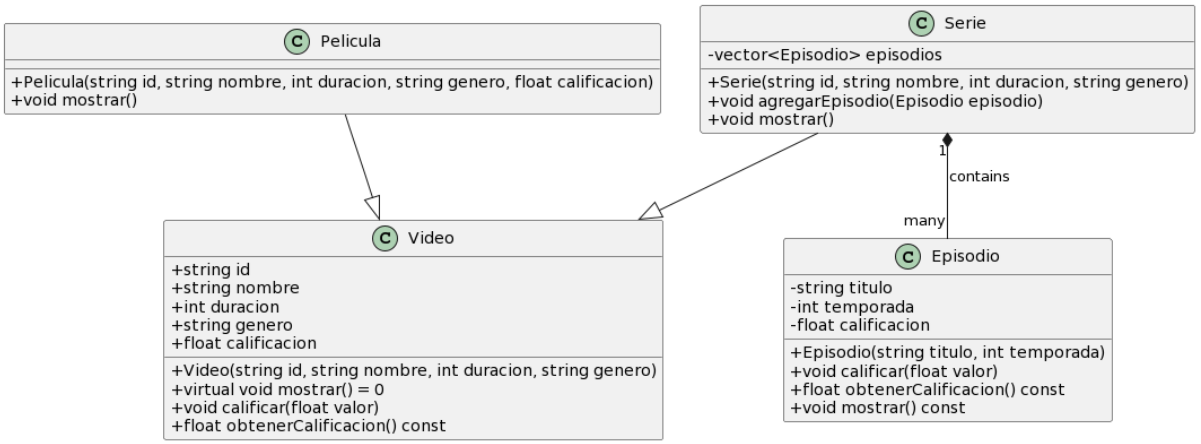
----- Menú -----

1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

El usuario digita el número deseado, ya sea para cargar archivos, visualizar los videos existentes, calificar un video, o salir.

Para este código utilizamos los conceptos aprendidos en clase de herencia, polimorfismo para construir el sistema de clases (“Video”; “Serie”, “Episodio” y “Película”).

Diagrama UML



- **Clase Video:** Esta es la clase base de la que básicamente todos las demás clases heredan atributos, tiene los atributos id, nombre, duración, género y calificación. También tiene los métodos mostrar(), calificar(float valor) y obtenerCalificacion(). El método mostrar() es un método virtual puro.

- **Clase Pelicula:** Esta clase hereda de la clase Video. Tiene un constructor y el método mostrar(), que es una implementación del método virtual puro de la clase padre.
- **Clase Serie:** Al igual que Pelicula, esta clase también hereda de Video. Tiene un atributo adicional: episodios, que es un vector de objetos Episodio. También tiene los métodos agregarEpisodio(Episodio episodio) y mostrar(). El método agregarEpisodio() se utiliza para agregar un episodio a la serie, y el método mostrar() es una implementación del método virtual puro de la clase padre.
- **Clase Episodio:** Esta clase no hereda de ninguna otra clase. Tiene los atributos titulo, temporada y calificacion, y los métodos calificar(float valor), obtenerCalificacion() y mostrar().
- **Relación entre Serie y Episodio:** La relación entre Serie y Episodio se representa con una línea que conecta las dos clases. El "1" cerca de Serie y "many" cerca de Episodio indica que una serie puede tener muchos episodios. Esta es una relación de composición, lo que significa que los episodios pertenecen a una serie y si la serie se elimina, también se eliminarán los episodios.

Ejemplo de Ejecución

```
----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o
 de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 2
Ingrese la calificación mínima: 2
Ingrese el género: Drama

Videos encontrados:
Película - ID: 2, Nombre: The Shawshank Redemption, Duración:
142 min, Género: Drama, Calificación: 2
```

- **Mostrar los videos en general con una cierta calificación o de un cierto género**
  - Esta opción funciona como un filtro, en este caso le pedimos que Buscara un video dentro de nuestra "base de datos" que cumpla con las características dadas: una calificación de 2 y que sea de Drama

```
----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o
 de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 3
Ingrese el ID de la serie: 4

Episodios encontrados:
Episodio - Título: Pilot, Temporada: 1, Calificación: 0
Episodio - Título: Crazy Handful of Nothin', Temporada: 1, Cal
ificación: 0
Episodio - Título: Full Measure, Temporada: 2, Calificación: 0
```

- **Mostrar los episodios de una determinada serie**
  - Como se puede apreciar aquí en vez de Mostrar los episodios con cierta calificación, mostramos los episodios de la serie que se solicita, esto se debe a que ya estaba bastante avanzado el codigo y esta era la única forma de solucionarlo, ya que al intentar añadir calificaciones individuales a cada episodio, el tejido de la realidad y el tiempo como lo conocemos sería destrozado (no compila el código), aun así creemos que mostrar los episodios de una serie es una opción muy útil si eres un usuario de esta plataforma.
  - En este caso se nos pide que proporcionemos el ID de la serie de la cual queremos ver los episodios, siendo en este caso el ID:4 que pertenece a Breaking Bad, al hacer este input se nos desplegará la lista de episodios que tenemos registrada bajo esa serie

```
----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o
 de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 4
Ingrese la calificación mínima:
4

Películas encontradas:
Película - ID: 1, Nombre: Inception, Duración: 148 min, Género
: Ciencia ficción, Calificación: 5
Película - ID: 3, Nombre: The Dark Knight, Duración: 152 min,
Género: Acción, Calificación: 4
```

- **Mostrar las películas con cierta calificación**
  - Al contrario de la opción 3 esta función funciona de manera exquisita, 10/10, verdaderamente una obra maestra, En este caso en nuestra “base de datos” tenemos 2 películas con calificaciones mayores a 4: “The Dark Knight” e “Inception”, que en efecto son mostradas al pedir que se muestren las que tengan una calificación mínima de 4.

```
Selecciona una opción: 5
Ingrese el título del video: The Shawshank Redemption
Ingrese la calificación (1-5): 5

----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o de un
cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 4
Ingrese la calificación mínima: 1

Películas encontradas:
Película - ID: 1, Nombre: Inception, Duración: 148 min, Género: Cienc
ia ficción, Calificación: 0
Película - ID: 2, Nombre: The Shawshank Redemption, Duración: 142 min
, Género: Drama, Calificación: 3
Película - ID: 3, Nombre: The Dark Knight, Duración: 152 min, Género:
 Acción, Calificación: 0
```

- **Calificar un video**
  - Aquí podemos ver que después de calificar un video (en este caso, The Shawshank Redemption le damos una calificación de 5) se puede sacar el promedio de todas las calificaciones, y ya después que queremos mostrar las calificaciones de algu

➢

```
----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o
 de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 2
Ingrese la calificación mínima: 5
Ingrese el género: Drama

Videos encontrados:
Película - ID: 2, Nombre: The Shawshank Redemption, Duración:
142 min, Género: Drama, Calificación: 5
```

➢

➢ De igual manera funciona al usar la opción 2, como mostrado anteriormente normalmente "The Shawshank Redemption" hubiera aparecido al buscar Calificación: 2 Género: drama, pero como su calificación se actualizo, ahora aparece al buscarlo con una calif. de 5.

```
----- Menú -----
1. Cargar archivo de datos
2. Mostrar los videos en general con una cierta calificación o
 de un cierto género
3. Mostrar los episodios de una determinada serie
4. Mostrar las películas con cierta calificación
5. Calificar un video
0. Salir

Selecciona una opción: 0
```

❖ **Salir**

➢ No hay mucho que decir, finaliza la ejecución del programa.

Argumentación del Código

Se asume que por argumentación de los puntos a) a h) se refiere a esto

- La aplicación debe mostrar el siguiente menu ciclado:
- Cargar archivo de datos
- Mostrar los videos en general con una cierta calificación o de un cierto género
- Mostrar los episodios de una determinada serie con una calificación determinada
- Mostrar las películas con cierta calificación
- Calificar un video
  - pedir título a calificar
  - pedir valor otorgado
- Salir

a) (10 pts) Se identifican de manera correcta las clases a utilizar

En los archivos Episodio.h, Pelicula.h, Serie.h y Video.h, se pueden identificar las clases que se van a utilizar a lo largo del código.

```
class Episodio {
private:
    string titulo;
    int temporada;
    float calificacion;
```

```
class Pelicula : public Video {
public:
    Pelicula(string id, string nombre, int duracion, string genero, float
calificacion);
    // Constructor de la clase Pelicula

    void mostrar();
    // Método para mostrar la información de la película
};
```

```
class Serie : public Video {
public:
    vector<Episodio> episodios; // Vector para almacenar los episodios de
la serie

public:
```

b) (12 pts) Se emplea de manera correcta el concepto de Herencia

En Pelicula.h y Serie.h, la clase Pelicula y la clase Serie heredan de la clase Video.

```
class Pelicula : public Video {
public:
    Pelicula(string id, string nombre, int duracion, string genero, float
calificacion);
```

```
class Serie : public Video {
public:
    vector<Episodio> episodios; // Vector para almacenar los episodios de
la serie
```

c) (10 pts) Se emplea de manera correcta los modificadores de acceso

En todos los archivos de clase (.h), se utilizan correctamente los modificadores de acceso 'private' y 'public'

```
class Episodio {
private:
    string titulo;
    int temporada;
    float calificacion;

public:
    // Constructor de la clase Episodio
    // Se encuentra en el archivo de declaración (.h)
    Episodio(string titulo, int temporada);
```

d) (12 pts) Se emplea de manera correcta la sobrecarga y sobreescritura de métodos

La sobrescritura de métodos se ve en las clases derivadas de Video (Pelicula y Serie), donde el método mostrar() se redefine.

```
void Pelicula::mostrar() {
    cout << "Película - ID: " << id << ", Nombre: " << nombre << ",
Duración: " << duracion << " min, Género: " << genero << ", Calificación:
" << obtenerCalificacion() << endl;
}
    // Muestra la información de la película en la consola
```

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

En main.cpp, se utiliza el polimorfismo al tratar a los objetos de las clases derivadas (Pelicula y Serie) como objetos de la clase base (Video).

```
int main() {
    vector<Video*> videos;

    // Películas
    videos.push_back(new Pelicula("1", "Inception", 148, "Ciencia
ficción", 5));
    videos.push_back(new Pelicula("2", "The Shawshank Redemption", 142,
"Drama", 2));
    videos.push_back(new Pelicula("3", "The Dark Knight", 152, "Acción",
4));
```

f) (12 pts) Se emplea de manera correcta el concepto de Clases Abstractas

En el archivo Video.h, la clase Video se define como una clase abstracta al declarar el método mostrar() como puramente virtual.

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

g) (12 pts) Se sobrecarga al menos un operador en el conjunto de clases propuestas

No hay sobrecarga dentro del código.

h) (opcional) Se utiliza de manera correcta el uso de excepciones tanto predefinidas como definidas por el programador

- **Mostrar los videos en general con cierta calificación/género**

```
51              case 2: {
52                  // Mostrar los videos en general con una cierta
    calificación o de un cierto género
53                  float calificacionMinima;
54                  string genero;
55
56                  cout << "Ingrese la calificación mínima: ";
57                  cin >> calificacionMinima;
58                  cout << "Ingrese el género: ";
59                  cin.ignore();
60                  getline(cin, genero);
61
62                  cout << "\nVideos encontrados:" << endl;
63                  for (const auto& video : videos) {
64                      if (video->obtenerCalificacion( ) >=
    calificacionMinima && video->genero == genero) {
65                          video->mostrar( );
66                      }
67                  }
68
69                  break;
70              }
```

Aquí se traen declaraciones de variables que ya habíamos establecido en sus correspondientes clases, aquí son calificación mínima y género y esencialmente lo que hace este caso 2 es encontrar un video que cumpla con las especificaciones que se le dieron en un input y desplegar el/los videos que encuentre con esas especificaciones

- **Mostrar los episodios de una determinada serie con una determinada calificación**

```
71              case 3: {
72                  // Mostrar los episodios de una determinada serie
73                  string idSerie;
74
75                  cout << "Ingrese el ID de la serie: ";
76                  cin >> idSerie;
77
78                  cout << "\nEpisodios encontrados:" << endl;
79                  for (const auto& video : videos) {
80                      Serie* serie = dynamic_cast<Serie*>(video);
81                      if (serie && serie->id == idSerie) {
82                          for (const auto& episodio : serie-
>episodios) {
83                              episodio.mostrar();
84                          }
85                          break;
86                      }
87                  }
88
89                  break;
90              }
```

Como mencionado anteriormente, intentar agregar una calificación individual a cada episodio ocasionaría la muerte espontánea del universo (no funcionaría el código) y como queremos evitar eso se optó por simplemente llamar a los episodios de una dada serie.

El caso 3 busca el ID de una serie ingresado por el usuario, busca la serie correspondiente en el vector videos y muestra los episodios asociados a esa serie. Esto permite visualizar los episodios de una determinada serie de manera individual. El dynamic_cast comprueba si video es una instancia de Serie utilizando dynamic_cast<Serie*>(video). Si se cumple la condición, significa que el video es una serie. En ese caso, se accede a los episodios asociados a la serie y se muestra su información.

- **Mostrar las películas con cierta calificación**

```
91              case 4: {
92                  // Mostrar las películas con cierta calificación
93                  float calificacionMinima;
94
95                  cout << "Ingrese la calificación mínima: ";
96                  cin >> calificacionMinima;
97
98                  cout << "\nPelículas encontradas:" << endl;
99                  bool encontradas = false;
100                 for (const auto& video : videos) {
101                     Pelicula* pelicula = dynamic_cast<Pelicula*>(video);
102                     if (pelicula && (pelicula->obtenerCalificacion() >=
    calificacionMinima || pelicula->obtenerCalificacion() == 0)) {
103                         pelicula->mostrar();
104                         encontradas = true;
105                     }
106                 }
107
108                 if (!encontradas) {
109                     cout << "No se encontraron películas con la calificación
    mínima especificada." << endl;
110                 }
111
```

Aquí, lo primero que se hace es preguntarle al usuario la calificación mínima, luego se itera cada elemento el el vector de videos, al igual que en el caso anterior se usa dynamic_cast para determinar si es lo que estamos buscando y en caso de que si, se utiliza mostrar() se utiliza una variable bool llamada encontradas para llevar un seguimiento de si se encontraron películas que cumplan con los criterios o no. Inicialmente, se establece como false.
Si se encuentra una película que cumple con los criterios, se establece encontradas como true.

- **Calificar un video**

```
114             case 5: {
115                 // Calificar un video
116                 string titulo;
117                 float valor;
118
119                 cout << "Ingrese el título del video: ";
120                 cin.ignore();
121                 getline(cin, titulo);
122                 cout << "Ingrese la calificación (1-5): ";
123                 cin >> valor;
124
125                 for (const auto& video : videos) {
126                     if (video->nombre == titulo) {
127                         video->calificar(valor);
128                         break;
129                     }
130                 }
131
132                 break;
133             }
134             default:
135                 cout << "Opción inválida. Por favor, selecciona una opción
    válida." << endl;
```

De manera similar a todo lo anterior, se itera sobre el vector video para buscar el video que fue pedido por el usuario, tras eso, se usa el método calificar() para actualizar la calificación del video encontrado sacando el promedio, aquí se usa el polimorfismo para tratar a todos los videos(películas y episodios) como objetos de la clase video, independientemente del tipo real del video.

Al inicio hubo problemas con este caso, sin embargo ahí entra la función ignore() ya que esta descarta el carácter de la nueva línea, de esta manera permite que se haga una lectura correcta sin obtener caracteres que no queramos.

- **Salir**

```
if (opcion == 0) {
    break;
}
```

Si escoges 0 se finaliza el codigo, asi de simple

Identificación de casos que harían que no funcionase

- **Carga de datos incorrecta:** Si se modifican las instrucciones del analisis de datos en el main y se introducen errores en la creación de objetos Película y Serie, como proporcionar argumentos incorrectos o no proporcionar los suficientes, podría generarse un comportamiento inesperado o errores durante la ejecución del programa.

- **Entrada de datos inválida:** Si el usuario ingresa datos inválidos o inesperados al interactuar con las opciones del menú, puede provocar errores o comportamientos inesperados.

- **Falta de gestión de memoria:** El programa utiliza new para crear instancias de objetos Pelicula y Serie. Si no se realiza una adecuada gestión de memoria y no se liberan los recursos correctamente utilizando delete al finalizar el programa, podría provocar problemas de rendimiento.

- **Dependencia de las rutas de archivo:** Si el programa depende de archivos externos que deben estar presentes en ubicaciones específicas, (justo lo que planteamos de la base de datos) y estos archivos se mueven o eliminan, el programa podría no funcionar correctamente debido a la incapacidad de cargar los datos esperados.

- **Intentar agregar calificaciones individuales a los episodios**

Conclusiones personales

- **Daniel**

Wow, pues por donde empezar, este proyecto llevo demasiado esfuerzo, no tuvimos que haberlo dejado para la última semana, por algo el archivo final se llama “Reto Intento 2.1 (1)” porque madre mía, el intento uno fue un fracaso total, se intentaron hacer demasiadas cosas, que un archivo de datos.txt, que borrar el Episodio.cpp porque lo necesario ya estaba y solo estorbaba, no estaba optimizado, en ningún momento compiló y hasta me salió pantallazo azul, ya para el segundo intento tuvimos que empezar desde 0 pero supimos optimizar mejor las cosas y decidimos tomar un enfoque más simple, sin embargo aunque compilaba, había mucho que cambiar porque solo dos opciones funcionaban bien bien, y asi estuvimos creando distintos Replits para no cambiar lo que funcionaba e intentar arreglar lo que no, le digo que intentar agregar calificaciones individuales a los episodios y que estas fueran analizadas resulto imposible sin tener que modificar todo nuevamente, pero si se fija, hay “indicios” de que si intentamos agregar la calificación, nada mas no se uso, pero si, puedo concluir que este fue un proyecto muy interesante y desafiante, realmente puso a prueba lo que aprendimos en el curso y la gloria que sentí cuando finalmente funcionó es indescriptible.

- **Marco**

Definitivamente fue un reto este código, primero que compilara y después quitar los bucles infinitos. En aquellos momentos de desesperación recordé en mi mente la voz del profe diciendo “Cuidado con los ciclos for, hace más ineficiente el código” y no sólo aprendí que los puede hacer más ineficiente, si no que además si no se cierra bien se puede arruinar el código. También rompimos varias veces (especialmente yo) una de las principales reglas de un programador: “Si no está roto, no le mueva”. Varias veces estaba ligeramente roto y pues buscaba soluciones, pero desafortunadamente yo le encontraba más problemas aún. Aquí si

agradecí mucho el trabajo de Daniel porque el fue el que encontró la mayoría de las soluciones para que este código funcionara correctamente. Fue un proyecto bastante interesante donde aprendí mucho, pero más que aprender por primera vez, me retó (o obligó) a repasar y entender mejor los conceptos de la clase, ya que gracias a la correcta comprensión de lo que representaba la herencia, el polimorfismo y la sobrecarga fue que llegamos a la solución (además de muchas ganas). Me gustó mucho el proyecto y disfruté mucho la clase.

- **André**

Este proyecto fue uno de los códigos más difíciles en los que haya trabajado dentro de mi carrera, pero gracias a este trabajo tuve un mejor conocimiento acerca de C++ el cuál se me complicaba mucho. Fue un desafío muy complicado, en especial tener que arreglar todos los errores que salían mientras agregamos más cosas al código, ya que en algunos momentos tuvimos que corregir todo desde el inicio. Agradezco este curso por ayudarnos a desarrollar nuestras habilidades colaborativas y en especial por enseñarnos más conceptos de programación.

Referencias:

- Concepto de Herencia - Herencia en C++ (Práctica 3). (n.d.). CodinGame. Recuperado 15 de junio 2023, de: https://www.codingame.com/playgrounds/50747/herencia-en-c-practica-3/concepto-de-herencia
- Polimorfismo - Manejo dinámico de memoria y Polimorfismo (Práctica 4). (n.d.). CodinGame. Recuperado 15 de junio 2023, de: https://www.codingame.com/playgrounds/51214/manejo-dinamico-de-memoria-y-polimorfismo-practica-4/polimorfismo
- CSV file management using C++. (2020). GeeksforGeeks. Recuperado 15 de junio 2023, de: https://www.geeksforgeeks.org/csv-file-management-using-c/

- C++ Polymorphism. (2023). GeeksforGeeks. Recuperado 15 de junio 2023, de: https://www.geeksforgeeks.org/cpp-polymorphism/
- Cai, S. X. (2023). Operator Overloading in C++. GeeksforGeeks. Recuperado 15 de junio, 2023, de https://www.geeksforgeeks.org/operator-overloading-cpp/