Tecnológico de Monterrey

E2. Integrative Project

Danna Valeria Guzmán Hdz-A00837310

Luis Gerardo Juarez Garcia - A00836928

15 of June 2023

Objects Oriented In Programing

Luis Andres Castillo Hernan.

Introduction

In the rapidly evolving world of streaming services, the demand for low-cost, on-demand access to a vast array of videos has soared. Platforms like Netflix, Disney, and DC have revolutionized the entertainment industry by offering extensive libraries of movies and series to their subscribers. As aspiring content providers in this thriving domain, we are tasked with developing a computational system that can efficiently manage videos, encompassing both movies and series. To accomplish this, we will harness the power of Object-Oriented Programming concepts, specifically in the context of the versatile C++ programming language.

In this project, we will embark on a journey to design a sophisticated application that seamlessly handles video information and generates insightful reports based on various criteria. Leveraging the fundamental principles of OOP, such as inheritance, polymorphism, and operator overloading, we will create an elegant and extensible solution that caters to the diverse needs of our users.

To lay the foundation of our project, we will construct a UML (Unified Modeling Language) Class Diagram that accurately represents the problem situation. This visual representation will encapsulate the essential entities, attributes, and relationships within our system, providing a blueprint for the design and implementation stages.

By combining the rich features of the C++ language with the principles of OOP, we aim to construct an application that seamlessly reads and processes information about different types of videos. Through the generation of insightful reports, users will be empowered to explore movies of specific genres, discover series within their preferred genres, or make informed choices based on video ratings.

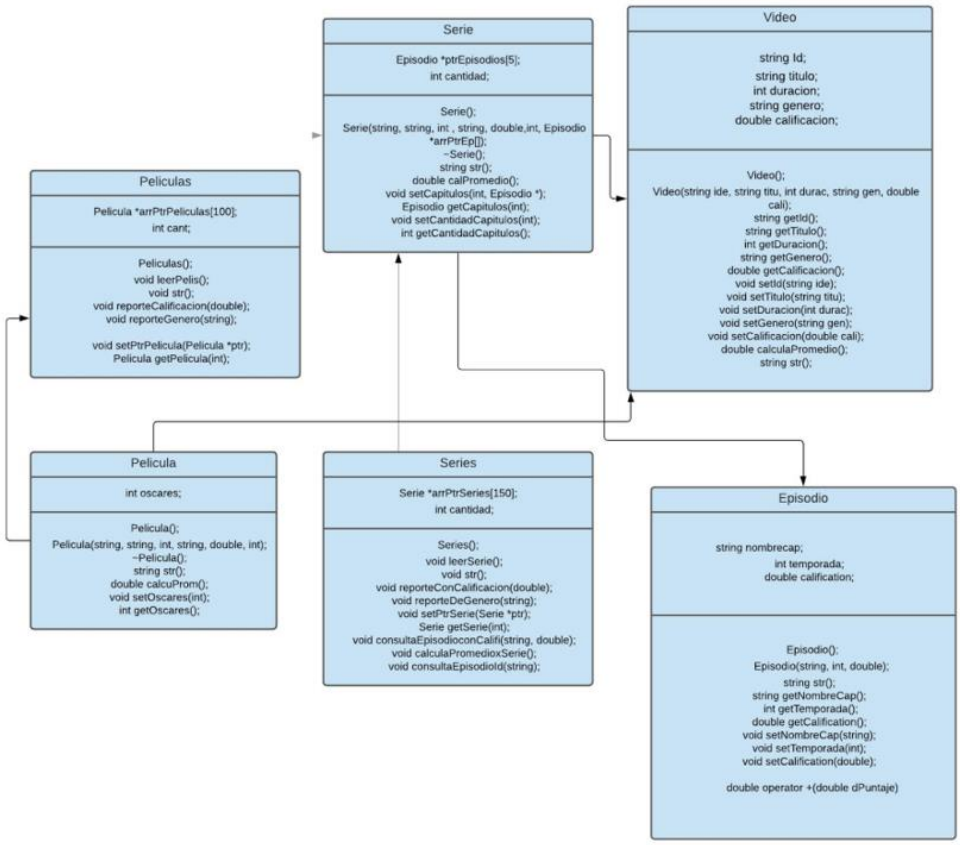
As we embark on this journey, we will not only acquire technical skills but also cultivate the attitudes and values necessary for problem-solving. We will develop a keen understanding of systemic analysis, responsible decision-making, and the utilization of research methodologies and engineering tools. By embracing these principles, we will enhance our ability to tackle complex problem situations and deliver impactful solutions.

DIAGRAM UML

A UML (Unified Modeling Language) diagram is a visual representation of the structure, relationships, and behavior of a system or software application. It helps in understanding the different components of a system and how they interact with each other.

In the context of the provided code, a UML diagram can be used to illustrate the classes and their relationships.

To better understand the operation of the program and all the classes used in this project, here is a UML diagram of the problem situation.



CODE

Main

```
#include <iostream>
#include <string>
#include "Movies.h"
#include "Series.h"
#include <fstream>
using namespace std;

//Function that rates a movie
void rateMovie(Movie &movieData){
    double rating;

    cout << "Enter the rating for the movie: ";
    cin >> rating;

    // Set the rating for the movie
    movieData.setCalification(rating);

    cout << "Movie rated successfully." << endl;.
}


//Function that registers an episode
void readEpisodeData(Episode &dataEpisode){
    string episode_title;
    int season;
    int calific;
    int canti;

    cout << "How many episodes you want to add: "<<endl;
    cin >> canti;
```

```
    cout << "How many episodes you want to add: "<<endl;
    cin >> canti;

    for(int cont = 0; cont < canti; cont++){
        cout << "Name of the episode: ";
        cin >> episode_title;
        cin.ignore();
        dataEpisode.setChapName(episode_title);

        cout <<"Season: ";
        cin >> season;
        cin.ignore();
        dataEpisode.setSeason(season);

        cout << "Calification: ";
        cin >> calific;
        cin.ignore();
        dataEpisode.setCalification(calific);
        cout<<endl;
    }

}

//void replaceNumberInFile(const string& Movies, int oldValue, int
newValue) {
//    ifstream inputFile(Movies);
//    if (!inputFile) {
//        cerr << "Error: Failed to open file '" << Movies << "'" <<
endl;
//        return;
```

```
//    ostringstream fileContents;
//    string line;
//
//    while (getline(inputFile, line)) {
//        size_t pos = line.find(',');
//        if (pos != string::npos) {
//            int value = stoi(line.substr(pos + 1));
//            if (value == oldValue) {
//                line = line.substr(0, pos + 1) +
to_string(newValue);
//            }
//        }
//
//        fileContents << line << endl;
//    }
//
//    inputFile.close();
//
//    ofstream outputFile(Movies);
//    if (!outputFile) {
//        cerr << "Error: Failed to open file '" << Movies << "'" <<
endl;
//        return;
//    }
//
//    outputFile << fileContents.str();
//    outputFile.close();
//}
```

Ln 2, Col 20 History

```cpp
//Streaming Menu
int streaMenu(){

    int option;
    cout<<"---------------------------------------"<<endl;
    cout << "\n========Movies=========            "<<
    "\n1. Read Movies File                            "<<
    "\n2. Rate a Movie                             "<<
    "\n3. Movie Report                           "<<
    "\n4. Calification Report                    "<<
    "\n5. Gender Report                      "<<
    "\n========Series==============                    "<<
    "\n6. Read Series and Episodes Files     "<<
    "\n7. Rate a Serie                       "<<
    "\n8. Calification Report                   "<<
    "\n9. Gender Report                 "<<
    "\n======== Episode ==============                      "<<
    "\n10. Check all episodes of a series by ID "<<
    "\n11. Calculate average per series                     "<<
    "\n0. Exit                                      ";

    cout<<"\nTell me the option you want to do: " <<endl;

    cin >> option;
    return option;
}

int main() {
    //Variables that start making variable designations work
```

```
int main() {
    //Variables that start making variable designations work
    Movies movies;
    Movie *ptrMovie;
    Series ser;
    Episode epis;
    Episode *ptrEpisode;

    int option;
    double cal;
    string gender;
    string sId;

    cout <<"Welcome to our streaming proyect\n"<<
    "We have a lot of Movies an Series.\n"<<
    "What do you want to do?: \n";
    //option Menu
    option = streaMenu();
    while (option != 0){
        switch (option){
            case 1:
                movies.readMovs(); //Read document from movies.csv
and print them
                break;
            case 2:
                //string Movies = "Movies.csv";
                //int oldValue = 3;
                //int newValue = 7;
                //replaceNumberInFile(Movies, oldValue, newValue);
                //cout << "Reemplazo realizado con éxito." << endl;
```

```
            case 3:
                movies.str(); //Give the report of the movies
including the new one that has just been released
                break;
            case 4:
                cout <<"Tell me the calification of the movie: ";
                cin >> cal;
                movies.calificationReport(cal); //After entering the
rating, it shows you the movies that have that rating
                break;
            case 5:
                cout <<"Tell me the gender of the movie: ";
                cin >> gender;
                movies.genderReport(gender); //After entering the
genre of the movie, it shows you the movies that have that genre
                break;
                //From this case, start the series menu
            case 6:
                ser.readSerie(); //Read the series that are in the
ArchivoSerie.csv document and all the episodes that are in the
Episodes.csv document and classify the episodes depending on their
series, which in this case is their ID
                break;
            case 7:
                ser.str(); //Give the report of the series in the
FileSeries.csv
                break;
            case 8:
                cout <<"Tell me the calification of the series";
                cin >> cal;
```

```
    rating designated by the user himself, search in the Archive.Series
    for series with that rating and show their episodes at the same time
    (without considering their rating)
72                  break;
73              case 9:
74                  cout <<"Tell me the gender of the series: ";
75                  cin >> gender;
76                  ser.reportOfGender(gender); //After giving the genre
    designated by the same user, search in File.Series for series with
    that rating and display their episodes at the same time
77                  break;
78              case 10:
79                  cout <<"Tell me the ID of the series: ";
80                  cin >> sId;
81                  ser.consultEpisodeID(sId);
82                  break;
83                  //After entering the iD of the series, it is possible
    to display all the episodes of the series with respect to the ID, and
    thanks to the queryEpisode function
84              case 11:
85
86                  break;
87
88              default:
89                  cout << "Error, Invalid Option";
90                  //If a different number from 1 to 13 is typed, the
    menu is repeated until you place the correct option depending on the
    menu
91                  break;
92          }
```

The provided code is a streaming program that allows managing movies and series. Here is a description of the classes used in the code:

Movies: This class handles movies. It contains functions to read movie data from a file (readMovs()), display a report of the movies (str()), and generate reports based on ratings (calificationReport()) and genre (genderReport()).

Movie: This class represents an individual movie. It has methods to set and get the movie's title, rating, and genre.

Series: This class handles series and episodes. It has functions to read series and episode data from files (readSerie()), display a report of the series (str()), generate reports based on ratings (reportWithCalification()) and genre (reportOfGender()), and query episodes of a series by ID (consultEpisodeID()).

Episode: This class represents an individual episode of a series. It has methods to set and get the episode's title, season, and rating.

In the main() function, objects of the mentioned classes are used to perform various operations based on the user's selected option:

- The movie menu options are related to the Movies class and call their respective methods.
- The series menu options are related to the Series class and call their respective methods.
- The "Rate a Movie" option allows rating a movie and calls the rateMovie() function, which takes a Movie object as a parameter.
- The "Read Series and Episodes Files" option calls the readSerie() function of the Series class to read series and episode data from files.
- The "Check all episodes of a series by ID" option allows querying episodes of a series by their ID and calls the consultEpisodeID() function of the Series class.
- Additionally, there is commented code that attempts to replace a number in a file, but it appears to be incomplete and not used in the main program.

EPISODE.CPP

```cpp
#include "Episode.h"
//default Constructor
Episode::Episode(){
    chapname = "-";
    season = 0;
    calification = 0;

}
//Constructor with parameters
Episode::Episode(string numcap, int seas, double grade){
    chapname = numcap ;
    season = seas;
    calification = grade;
}
//Function that prints the data
string Episode::str(){
    return "\nName of the chapter: "+ chapname + " \nSeason: "
+to_string(season)+ " \nCalification: "+ to_string(calification)+"\n";
}
//CapName get method
string Episode::getChapName(){
    return chapname;
}
//Season get method
int Episode::getSeason(){
    return season;
//    return 0;
}
//Calification get method
double Episode::getCalification(){
```

```cpp
//Constructor with parameters
Episode::Episode(string numcap, int seas, double grade){
    chapname = numcap ;
    season = seas;
    calification = grade;
}
//Function that prints the data
string Episode::str(){
    return "\nName of the chapter: "+ chapname + " \nSeason: "
+to_string(season)+ " \nCalification: "+ to_string(calification)+"\n";
}
//CapName get method
string Episode::getChapName(){
    return chapname;
}
//Season get method
int Episode::getSeason(){
    return season;
//    return 0;
}
//Calification get method
double Episode::getCalification(){
    return calification;
//    return 1.0;
}
//set NombreCap Method
void Episode::setChapName(string numcap){
    chapname = numcap ;
}
```

```cpp
    return calification;
//    return 1.0;
}
//set NombreCap Method
void Episode::setChapName(string numcap){
    chapname = numcap ;
}
//Season set Method
void Episode::setSeason(int seas){
    season = seas;
}
//set Calification method
void Episode::setCalification(double grade){
    calification = grade;
}

```

EPISODE.H

```cpp
#ifndef Episode_h
#define Episode_h
#include <stdio.h>
#include <string>
using namespace std;

class Episode{
protected: //Protected members
    string chapname;
    int season;
    double calification;
public: // Members and public functions
    Episode(); //Default constructor
    Episode(string, int, double); //Constructor with parameters
    string str(); //string that prints the corresponding data in order
    string getChapName(); //Method that returns the protected variable chapname
    int getSeason(); //Method that returns the protected variable season
    double getCalification(); //Method that returns the protected variable qualification
    void setChapName(string); //Method that modifies the chapname variable
    void setSeason(int); //Method that modifies the variable season
    void setCalification(double); //Method that modifies the variable qualification

    // Overload the + operator in the score and grade
    double operator +(double dScore){
        return dScore + calification;
    string getChapName(); //Method that returns the protected variable chapname
    int getSeason(); //Method that returns the protected variable season
    double getCalification(); //Method that returns the protected variable qualification
    void setChapName(string); //Method that modifies the chapname variable
    void setSeason(int); //Method that modifies the variable season
    void setCalification(double); //Method that modifies the variable qualification

    // Overload the + operator in the score and grade
    double operator +(double dScore){
        return dScore + calification;
    }
};


#endif /* Episodio_hpp */
```

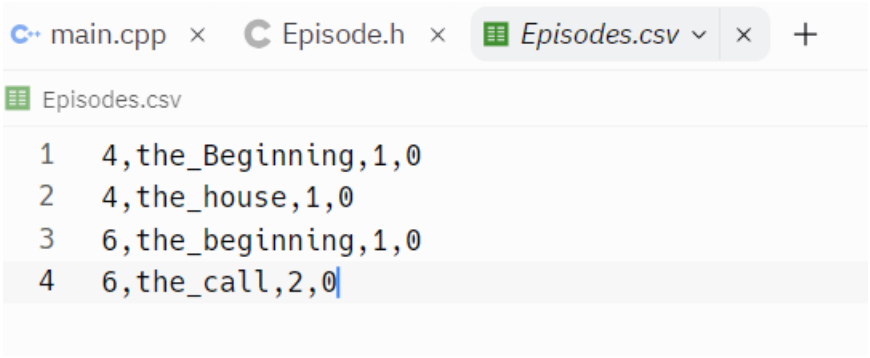
In Episode.cpp:

- The default constructor Episode::Episode() initializes the class members with default values.
- The constructor with parameters Episode::Episode(string, int, double) assigns the provided values to the corresponding members.
- The Episode::str() function returns a string that represents the episode's data.
- The getChapName(), getSeason(), and getCalification() methods are accessor methods that return the values of the corresponding members.
- The setChapName(), setSeason(), and setCalification() methods are modification methods that update the values of the corresponding members.

In Episode.h:

The header file declares the Episode class, including the declaration of the members and methods mentioned in Episode.cpp.

Episode.csv:



Movie.cpp

```cpp
#include "Movie.h"
Movie::Movie():Video(){ //Constructor default child and parent
    oscars = 0;
}
Movie::Movie(string ide, string titl, int durat, string gen, double cali, int osca): Video(ide,titl,durat,gen,cali){ //Constructor with parameters (Daughter and Father)
    oscars = osca;
}
//Destructor
Movie::~Movie(){

}
//str method that prints the function video::str including the number of oscars
string Movie::str(){
    return Video::str() + " \nOscars: " + to_string(oscars);
}
//Function that returns the getDuration method of the Video(Parent) class
double Movie::calcuAverage(){
    return getDuration();
}
//Oscars set method
void Movie::setOscars(int osca){
    oscars = osca;
}
//Oscar get method
int Movie::getOscars(){
    return oscars;
```

Ln 1, Col 1 History

Movie.h

```
#ifndef Movie_h
#define Movie_h

#include <stdio.h>
#include <string>
#include "Video.h"
//Polymorphism of the Video class
class Movie: public Video{
protected:
    int oscars;
public:
    Movie();//default constructor
    Movie(string, string, int, string, double, int); //constructor
with parameters
    ~Movie(); //Destructor
    string str(); //str method
    double calcuAverage(); //average function
    void setOscars(int); //Set oscars method
    int getOscars(); //get oscars method
};



#endif /* Movie_h */
```

Movies.cpp

```cpp
#include "Movies.h"
#include "Movie.h"

#include <sstream>
#include <fstream>
//Default constructor method of the Movies class
Movies::Movies(){
    cant = 0;
    for(int r = 0; r < 100; r++){
        arrPtrMovies[r]=nullptr;

    }
}
//function read movies
void Movies::readMovs(){
    fstream movs;
movs.open("Movies.csv"); //Opens the Movies.txt file
    string row[6]; //6 columns of data are counted, these are
separated by a comma
    string line, word;

    //While reading the Movies.csv document line by line
    while(getline(movs, line)){
        cout << cant << line << endl;
        stringstream s(line);
        // Print the line of the document
        int r = 0;
        //Reads the doc line by line and stops until the commas it sees
        while(getline(s, word, ',')){
            row[r++]= word;
```

```
29              row[r++]= word;
30          }
31          // An instance of the Movie class is created to be registered
32          arrPtrMovies[cant] = new Movie();
33          cout << "ID: " << row[0] << "\n";
34          cout << " Title: " << row[1] << "\n";
35          cout << " Duration: " << row[2] << "\n";
36          cout << " Gender: " << row[3] << "\n";
37          cout << " Calification: " << row[4] << "\n";
38          cout << " Oscars: " << row[5] << "\n";
39          //The object was created, now the values of each attribute are
    assigned to it using the corresponding set methods
40          arrPtrMovies[cant]->setId(row[0]);
41          arrPtrMovies[cant]->setTitle(row[1]);
42          arrPtrMovies[cant]->setDuration(stoi(row[2]));
43          arrPtrMovies[cant]->setGender(row[3]);
44          arrPtrMovies[cant]->setCalification(stoi(row[4]));
45          arrPtrMovies[cant]->setOscars(stoi(row[5]));
46          cout << arrPtrMovies[cant++]->str() << endl;
47      }
48      movs.close(); // close the file
49  } // The function ends
50  void Movies::str(){
51      double prom;
52      prom = 0;
53      cout << "ID      TITLE       DURATION     GENDER    CALIFICATION
     OSCARS   \n";
54      cout << "-----------------------------------------------------------
    ----------\n";
55      //The average of the movies is done using the get rating
```

Movies.cpp > str

```
53      cout << "ID      TITLE       DURATION     GENDER    CALIFICATION
     OSCARS   \n";
54      cout << "-----------------------------------------------------------
    ----------\n";
55      //The average of the movies is done using the get rating
56      for (int r = 0; r < cant; r++){
57          cout << arrPtrMovies[r]->str() << endl;
58          prom = prom + arrPtrMovies[r]->getCalification();
59      }
60      cout << "Prom of the movies: " << prom / cant <<endl;
61  }
62  //Function in which a movie is searched by a rating designated by the
    user
63  void Movies::calificationReport(double grade){
64      cout << "ID      TITLE       DURATION     GENDER    CALIFICATION
     OSCARS   \n";
65      cout << "-----------------------------------------------------------
    ----------\n";
66
67      for(int r = 0; r < cant; r++){
68          if(arrPtrMovies[r]->getCalification() == grade)
69              cout << arrPtrMovies[r]->str() << endl;
70      }
71  }
72  //Function in which a movie is searched by the genre designated by the
    user
73  void Movies::genderReport(string gender){
74      cout << "ID      TITLE       DURATION     GENDER    CALIFICATION
     OSCARS   \n";
75      cout << "-----------------------------------------------------------
```

```
74      cout << "ID      TITLE       DURATION     GENDER    CALIFICATION
     OSCARS   \n";
75      cout << "-----------------------------------------------------------
    ----------\n";
76      for(int r = 0; r < cant; r++){
77          if(arrPtrMovies[r]->getGender() == gender)
78              cout << arrPtrMovies[r]->str() << endl;
79      }
80  }
81  // the pointer is realized
82  void Movies::setPtrMovie(Movie *ptr){
83  //    cout << ptr << endl;
84      arrPtrMovies[cant] = ptr;
85      cant = cant + 1;
86  }
87  // getMovies function linked with Movie class
88  Movie Movies::getMovie(int movi){
89      Movie movie;
90
91      if(movi >=0 && movi <= cant)
92          return (*arrPtrMovies[movi]);
93      else
94          return movie;
95  }
96
97
```

Movies.csv

Movies.csv

```
1   1,Batman,120,Accion,0,0
2   3,The_Hobbit,124,drama,0,0
3   5,Roma,90,drama,0,0
4   17,Inception,125,thriller,0,0
5
```

Movies.h

```
#ifndef Movies_h
#define Movies_h

#include <stdio.h>
#include "Movie.h"
#include <iostream>
using namespace std;

class Movies{
private:
     //array of part of movie class with value from 0 to 100
    Movie *arrPtrMovies[100];
    int cant;
public:
    Movies(); //Default Constructor
    void readMovs(); //Funcion to read movies
    void str(); //str Function
    void calificationReport(double); //Function to report by rating
    void genderReport(string);
    void setPtrMovie(Movie *ptr); //Movie class pointer
    Movie getMovie(int); // getMovie method that is part of the movie
class
};


#endif /* Peliculas_hpp */
```

File "Movie.cpp":

- Defines the implementation of the `Movie` class, which is a subclass of the `Video` class.
- Includes a default constructor and a parameterized constructor.
- Provides methods to get and set the number of Oscars for a movie.
- Implements the `str()` method, which returns a string representing the movie information, including the inherited information from the parent class `Video`.
- Implements the `calcuAverage()` method, which calculates the average duration of the movie using the `getDuration()` method from the parent class.
- Provides methods to set and get the number of Oscars for the movie.

File "Movie.h":

- Defines the declaration of the `Movie` class, which inherits from the `Video` class.

- Includes necessary libraries and defines the protected and public variables and methods of the class.

File "Movies.cpp":

- Defines the implementation of the `Movies` class.
- Includes the header files "Movies.h" and "Movie.h".
- Provides a default constructor for the `Movies` class, which initializes variables and an array of pointers to `Movie` objects.
- Implements the `readMovs()` method, which reads movie information from a CSV file named "Movies.csv" and creates corresponding `Movie` objects using the read data.
- Implements the `str()` method, which prints the information of the movies stored in the array of pointers.
- Implements the `calificationReport()` and `genderReport()` methods, which search for movies by rating and genre respectively and print them.
- Provides methods to set and get pointers to `Movie` objects.
- Provides the `getMovie()` method, which returns a specific movie from the array of pointers.

The "Movies.csv" file is a CSV file that contains information about movies. Each line of the file represents a movie and contains the following fields separated by commas:

- ID: Unique identifier of the movie.
- Title: Title of the movie.
- Duration: Duration of the movie in minutes.
- Genre: Genre to which the movie belongs.
- Rating: Rating of the movie.
- Oscars: Number of Oscars won by the movie.

Serie.cpp

```cpp
#include "Serie.h"
//Default constructor of the serial class linked to the one of the Video class
Serie::Serie():Video(){
    quantity = 0;
    for(int iR = 0; iR < 5; iR++)
        ptrEpisodes[iR] = nullptr;
}
// Constructor with parameters of the serial class linked to that of the Video class
Serie::Serie(string ide , string titu, int durac, string gen, double cali,int cant, Episode *arrPtrEp[]):Video(ide, titu, durac, gen, cali){

    quantity = 0;
    for(int iR = 0; iR < cant; iR++)
        ptrEpisodes[iR] = arrPtrEp[iR];
    cout << "-----Se cumple-----"<<endl;

}
//Destructor
Serie::~Serie(){

}
//Str function of class series
string Serie::str(){
    string sEpisodes = "";
    int iR = 0;
    while( iR < quantity){
        sEpisodes = sEpisodes + ptrEpisodes[iR]->str()+"\n";
        iR++;
```

```
21  //Str function of class series
22  string Serie::str(){
23      string sEpisodes = "";
24      int iR = 0;
25      while( iR < quantity){
26          sEpisodes = sEpisodes + ptrEpisodes[iR]->str()+"\n";
27          iR++;
28      }
29      return Video::str() + "\nEpisodes = " + to_string(quantity) +
    "\nList of Episodes: " + sEpisodes + "\n";
30  }
31
32  //Function with operator overload
33  double Serie::calAverage(){
34      double acum = 0;
35      for(int iB = 0; iB < quantity; iB++){
36          acum = acum + ptrEpisodes[iB]->getCalification();
37  // OVERLOADS THE SUM OPERATOR IN THE EPISODE CLASS
38          acum = *ptrEpisodes[iB] + acum ;
39      }
40      if(quantity > 0){
41          return (acum / quantity)/2;
42      }
43      return -1;
44  }
45
46
47  //validate that iNum >=0 && iNum <5-delete - object -
48  //set method for Chapters
49  void Serie::setChapters(int iNum, Episode *episodio){
```

```
48  //set method for Chapters
49  void Serie::setChapters(int iNum, Episode *episodio){
50     ptrEpisodes[iNum] = episodio;
51      quantity++;
52
53  }
54  // get method for Chapters
55  Episode Serie::getChapters(int iNum){
56      return *ptrEpisodes[iNum];
57
58  }
59  //Method set for QuantityChapters
60  void Serie::setQuantityChapters(int cant){
61      quantity = cant;
62  }
63  //Method get for QuantityChapters
64  int Serie::getQuantityChapters(){
65      return quantity;
66  }
67
```

Serie.h

```h
#ifndef Serie_h
#define Serie_h
#include <stdio.h>
#include <string>
#include "Video.h"
#include "Episode.h"
//Polymorphism of the Video class
class Serie: public Video{
protected:
//Array of episodes with value up to 5
 Episode *ptrEpisodes[5];
 int quantity;

public:
    Serie(); //Default constructor of the String class
    Serie(string, string, int , string, double,int, Episode *arrPtrEp[]); //Constructor with parameters of the Series class
    ~Serie(); //Destructor of the String class
     string str(); //Str method that prints the data of Video::Str and prints the number of episodes and the list of episodes for each series
     double calAverage(); //Function that calculates the average of the series
     void setChapters(int, Episode *); //Chapter set method
     Episode getChapters(int); //Chapter get method
     void setQuantityChapters(int); //Method set of QuantityChapters
     int getQuantityChapters();//Method get of QuantityChapters


};
#endif /* Serie_hpp */
```

SerieFile.csv

main.cpp × Episode.h × Movies.cpp × SerieFi

SerieFile.csv

```csv
4,Stranger_Things,45,thriller,0,1
6,The_Chosen,51,drama,0,1

```

Series.cpp

```cpp
#include "Series.h"
#include "Episode.h"
#include <sstream>
#include <fstream>
//constructor default method for class series
Series::Series(){
    quantity = 0;
    for(int iR = 0; iR<150; iR++ ){
        arrPtrSeries[iR] = nullptr;
    }

}
//Function read series and episodes
void Series::readSerie(){
    fstream seri;

    int iEpisode = 0;

    int arrIdSerie[150];

    Episode *arrPtrEpisode[150];

    Episode *arrPtrEpis[5];

    seri.open("Episodes.csv", ios::in);
    string row[6];
    string line, word;

    while(getline(seri, line)){

```

```
24
25      seri.open("Episodes.csv", ios::in);
26      string row[6];
27      string line, word;
28
29      while(getline(seri, line)){
30
31          cout << iEpisode << " = " << line << endl;
32          stringstream s(line);
33
34          int iR = 0;
35
36          while(getline(s,word, ',')){
37              row[iR++] = word;
38          }
39
40          arrPtrEpisode[iEpisode] = new Episode();
41
42          cout << "\nID Serie: " << row[0] << "\n";
43          cout << "\nTitle: " << row[1] << "\n";
44          cout << "\nSeason: " << row[2] << "\n";
45          cout << "\nCalification: " << row[3] << "\n";
46
47          arrIdSerie[iEpisode] = stoi(row[0]);
48
49          arrPtrEpisode[iEpisode]->setChapName(row[1]);
50          arrPtrEpisode[iEpisode]->setSeason(stoi(row[2]));
51          arrPtrEpisode[iEpisode]->setCalification(stod(row[3]));
52
53          cout << "---"<< arrPtrEpisode[iEpisode]->str() << endl;
54          iEpisode++;
```

```
55      }
56
57      seri.close();
58
59      for(int iR = 0; iR < iEpisode; iR++)
60      cout <<"$"<< arrPtrEpisode[iR]->str() <<endl;
61
62
63      cout <<"--------------------------Series Report-----------------
    ----------------\n";
64
65      seri.open("SerieFile.csv", ios::in);
66      quantity = 0;
67      while(getline(seri, line)){
68          cout << quantity << " : "<< line << endl;
69
70          stringstream s(line);
71
72          int iR = 0;
73
74          while(getline(s,word,',')){
75
76
77              row[iR++]= word;
78          }
79          cout << "ID: " << row[0] << "\n";
80          cout << "Title: " << row[1] << "\n";
81          cout << "Duration: " << row[2] << "\n";
82          cout << "Gender: " << row[3] << "\n";
83          cout << "Calification: " << row[4] << "\n";
```

```
83          cout << "Calification: " << row[4] << "\n";
84          cout << "Episodes: " << row[5] << "\n";
85
86      for(int iR = 0; iR < 5; iR++)
87          arrPtrEpis[iR] = nullptr;
88
89    arrPtrSeries[quantity] = new
    Serie(row[0],row[1],stoi(row[2]),row[3],stod(row[4]),stoi(row[5]),arrP
    trEpis);
90
91          int iE = 0;
92
93          for(int iEpisod = 0; iEpisod < iEpisode & iE < 5; iEpisod++){
94              if (arrIdSerie[iEpisod] == stoi(row[0])){
95                  arrPtrSeries[quantity]->setChapters(iE,
    arrPtrEpisode[iEpisod]);
96                  iE++;
97              }
98
99          }
100         cout << "----inside object: "<< arrPtrSeries[quantity] <<
    endl << arrPtrSeries[quantity]->str() << endl;
101         cout << "End of reading the other series "<< endl;
102         quantity = quantity + 1;
103     }
104     cout << "It's over " << endl;
105     seri.close();
106
107 }
108 //Function str of the Series class that calculates the report of the
```

```
    series (Its average)
109 void Series::str(){
110     double prom;
111     prom = 0;
112 
113     cout << "ID         Series        Duration     Gender
    Calification   Episodes  List of Episodes  \n";
114     cout << "-----------------------------------------------------
    ------------\n";
115 
116     for(int iR = 0; iR < quantity; iR++){
117         cout << arrPtrSeries[iR]->str() << endl;
118         prom = prom + arrPtrSeries[iR]->getCalification();
119 
120     }
121     cout<< endl;
122     cout << "Average Series: "<< prom / quantity << endl;
123 }
124 
125 //Function in which a series is searched for by a qualification
    designated by the user
126 void Series::reportWithCalification(double calific){
127         cout << "ID         Series        Duration     Gender
    Calification   Episodes  List of Episodes  \n";
128     cout << "-----------------------------------------------------
    -------------\n";
129     for(int iR = 0; iR < quantity; iR++){
130         if(arrPtrSeries[iR]->getCalification()== calific)
131             cout << arrPtrSeries[iR]->str() << endl;
132     }
```

```
            if(arrPtrSeries[iR]->getCalification()== calific)
131             cout << arrPtrSeries[iR]->str() << endl;
132         }
133         cout << "------------------------------------------------------------
    -------------\n";
134     }
135
136     //Function in which a series is searched by the gender designated by
    the user
137     void Series::reportOfGender(string gener){
138
139           cout << "ID         Series        Duration     Gender
    Calification   Episodes  List of Episodes  \n";
140         cout << "------------------------------------------------------------
    -------------------------\n";
141         for(int iR = 0; iR < quantity; iR++){
142             if(arrPtrSeries[iR]->getGender() == gener){
143                 cout << arrPtrSeries[iR]->str() << endl;
144             }
145         }
146         cout << "------------------------------------------------------------
    -------------\n";
147     }
148
149     //The pointer of the series class is made using the series class
150     void Series::setPtrSerie(Serie *ptr){
151         arrPtrSeries[quantity] = ptr;
152         quantity = quantity+1;
153     }
154     // get method of the String class
155     Serie Series::getSerie(int iSerie){
```

```
152     quantity = quantity+1;
153 }
154 // get method of the String class
155 Serie Series::getSerie(int iSerie){
156     Serie serie;
157 
158     if (iSerie >= 0 && iSerie <= quantity)
159         return (*arrPtrSeries[iSerie]);
160     else
161         return serie;
162 }
163 
164 //Function that consults all the episodes of the series that have the
    rating requested by the user
165 void Series::EpisodeConsultWithCal(string iD, double dCal){
166     Serie consult;
167     for(int iR = 0; iR < quantity; iR++)
168         if(arrPtrSeries[iR]->getId() == iD){
169 
170             for(int iEpisode = 0; iEpisode < arrPtrSeries[iR]-
    >getQuantityChapters(); iEpisode++){
171                 if(arrPtrSeries[iR]-
    >getChapters(iEpisode).getCalification() == dCal)
172                     cout << arrPtrSeries[iR]-
    >getChapters(iEpisode).str()<<endl;
173         }
174     }
175 }
176 
177 //Function that calculates the average of each series through its
```

```
177  //Function that calculates the average of each series through its
     episodes and makes the average
178  void Series::calculateAveragePerSerie(){
179      double dCalProm;
180
181      for(int iR = 0; iR < quantity; iR++){
182          dCalProm = arrPtrSeries[iR]->calAverage();
183          arrPtrSeries[iR]->setCalification(dCalProm);
184      }
185
186  }
187
188  //Function that prints all the episodes of the series depending on
     their ID
189  void Series::consultEpisodeID(string iD){
190      Serie consult;
191      for(int iR = 0; iR < quantity; iR++)
192          if(arrPtrSeries[iR]->getId() == iD){
193              for(int iEpisode = 0; iEpisode < arrPtrSeries[iR]-
     >getQuantityChapters(); iEpisode++)
194              cout << arrPtrSeries[iR]->getChapters(iEpisode).str()
     <<endl;
195          }
196  }
197
```

Series.h

```
#ifndef Series_h
#define Series_h
#include <stdio.h>
#include <iostream>
using namespace std;
#include "Serie.h"

class Series{
private:
//Array is the series class with value up to 150
    Serie *arrPtrSeries[150];
    int quantity;
public:
//Public methods and functions of the series class
    Series(); //Default Constructor
    void readSerie(); //Constructor with parametros
    void str(); //print all
    void reportWithCalification(double); //Function that makes the
report of the series through the qualification
    void reportOfGender(string); //Function that makes the report of
the series through the gender
    void setPtrSerie(Serie *ptr); //Pointer of series class
    Serie getSerie(int); //getSerie method of the Series class
    void EpisodeConsultWithCal(string, double); //Function to consult
episodes of a series by rating
    void calculateAveragePerSerie(); //Function to calculate the
average of the series by rating the episodes of each series
    void consultEpisodeID(string); //Function to consult all the
episodes of a series by means of its ID
};
```

In "Serie.h":

- The header file includes necessary dependencies, such as "Video.h" and "Episode.h".
- The class "Serie" is declared, which is a subclass of the "Video" class.
- It contains protected member variables: an array of pointers to Episode objects (**ptrEpisodes**) and an integer **quantity**.
- Public member functions are declared, including constructors, a destructor, and various getter and setter methods.

In "Serie.cpp":

- The implementation file defines the member functions declared in "Serie.h".
- The parameterized constructor initializes the Serie object with provided values and assigns the episode pointers from the given array.
- The destructor is empty, implying no explicit cleanup is required.
- The **str()** method returns a string representation of the Serie object, including the video details and a list of episodes.
- The **calAverage()** method calculates the average rating of the series based on the ratings of its episodes.
- Various setter and getter methods are implemented to manage episodes and their quantities.

In "Series.h":

- The header file includes necessary dependencies and declares the class "Series".
- The class contains private member variables: an array of pointers to Serie objects (**arrPtrSeries**) and an integer **quantity**.
- Public member functions are declared, including constructors, methods for reading series and episodes from files, generating reports, and performing operations on the series and episodes.

In "Series.cpp":

- The implementation file defines the member functions declared in "Series.h".
- The default constructor initializes the Series object with default values and sets the array of Serie pointers to nullptr.
- The **readSerie()** method reads series and episode information from files and creates corresponding objects.
- The **str()** method generates a report of all the series, including their details and episodes.
- Other methods, such as **reportWithCalification()**, **reportOfGender()**, **EpisodeConsultWithCal()**, etc., perform specific operations on the series and episodes.

Video.cpp

```cpp
#include "Video.h"
//Default Constructor
Video::Video(){
   Id = "_";
   title = "_";
   duration = 0;
   gender = "_";
   calification = 0;
}
//Constructor with parameters
Video::Video(string ide, string titl, int durat, string gen, double cali){
    Id = ide;
    title = titl;
    duration = durat;
    gender = gen;
    calification = cali;
}
//get Method ID
string Video::getId(){
    return Id;
}
//get Method TITLE
string Video::getTitle(){
    return title;
}
//get Method DURATION
int Video::getDuration(){
    return duration;
}
```

```
30  //get Method GENRE
31  string Video::getGender(){
32      return gender;
33  }
34  //get Method CALIFICATION
35  double Video::getCalification(){
36      return calification;
37  }
38  //set Method ID
39  void Video::setId(string ide){
40      Id = ide;
41  }
42  //set Method TITLE
43  void Video::setTitle(string titl){
44      title = titl;
45  }
46  //set Method DURATION
47  void Video::setDuration(int durat){
48      duration = durat;
49  }
50  //set Method GENRE
51  void Video::setGender(string gen){
52      gender = gen;
53  }
54  //get Method CALIFICATION
55  void Video::setCalification(double cali){
56      calification = cali;
57  }
58  //double method to calculate average
59  double Video::calculateAverage(){
```

```cpp
//set Method GENRE
void Video::setGender(string gen){
    gender = gen;
}
//get Method CALIFICATION
void Video::setCalification(double cali){
    calification = cali;
}
//double method to calculate average
double Video::calculateAverage(){
    return calification;
}
//print method
string Video::str(){
  return "\nID: " + Id + " \nTitle: " + title + " \nDuration: " +
to_string(duration) + " \nGender:" +  gender + " \nCalification: " +
to_string(calification);
}

```

Video.h

```cpp
#ifndef Video_h
#define Video_h
// VIDEO CLASS
#include <stdio.h>
#include <iostream>
#include <string>
using namespace std;

class Video{
private:
        string Id;
        string title;
        int duration;
        string gender;
        double calification;
  public:
        Video(); //Default Constructor
        Video(string ide, string titl, int durat, string gen, double
cali); //Constructor with parametros
        string getId(); //get Method ID
        string getTitle(); //get Method Title
        int getDuration(); //get Method Duration
        string getGender(); //get Method Genre
        double getCalification(); //get Method Calification
        void setId(string ide); //set Method ID
        void setTitle(string titl); //set Method Ttile
        void setDuration(int durat); //set Method Duration
        void setGender(string gen); //set Method Genre
        void setCalification(double cali); //set Method Calification
       double calculateAverage(); //Function to calculate average
```

```
21          int getDuration(); //get Method Duration
22          string getGender(); //get Method Genre
23          double getCalification(); //get Method Calification
24          void setId(string ide); //set Method ID
25          void setTitle(string titl); //set Method Ttile
26          void setDuration(int durat); //set Method Duration
27          void setGender(string gen); //set Method Genre
28          void setCalification(double cali); //set Method Calification
29         double calculateAverage(); //Function to calculate average
30          string str(); //print all
31      };
32
33
34  #endif /* Video_hpp */
```

The "video.h" file contains the declaration of the class and its methods, while the "video.cpp" file contains the implementation of the class methods.

Private attributes of the "Video" class:

- Id: Represents the ID of the video (string).
- title: Represents the title of the video (string).
- duration: Represents the duration of the video (integer).
- gender: Represents the genre of the video (string).
- calification: Represents the rating of the video (floating-point number).

Public methods of the "Video" class:

- Video(): Default constructor that initializes the video attributes with default values.
- Video(string ide, string titl, int durat, string gen, double cali): Constructor that allows setting custom values for the video attributes.
- getId(): Method that returns the ID of the video.
- getTitle(): Method that returns the title of the video.
- getDuration(): Method that returns the duration of the video.
- getGender(): Method that returns the genre of the video.
- getCalification(): Method that returns the rating of the video.
- setId(string ide): Method that sets the ID of the video.
- setTitle(string titl): Method that sets the title of the video.
- setDuration(int durat): Method that sets the duration of the video.
- setGender(string gen): Method that sets the genre of the video.
- setCalification(double cali): Method that sets the rating of the video.

- calculateAverage(): Method that calculates the average rating of the video (currently just returns the rating without performing any calculation).
- str(): Method that returns a string representation of all the attributes of the video.

Execution example video https://youtu.be/qEgANGasb4A

```
========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
1
01,Batman,120,Accion,0,0
ID: 1
 Title: Batman
 Duration: 120
 Gender: Accion
 Calification: 0
 Oscars: 0

ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0
13,The_Hobbit,124,drama,0,0
ID: 3
 Title: The_Hobbit
 Duration: 124
 Gender: drama
 Calification: 0
 Oscars: 0
```

```
ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0
25,Roma,90,drama,0,0
ID: 5
 Title: Roma
 Duration: 90
 Gender: drama
 Calification: 0
 Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0
317,Inception,125,thriller,0,0
ID: 17
 Title: Inception
 Duration: 125
 Gender: thriller
 Calification: 0
 Oscars: 0

ID: 17
Title: Inception
Duration: 125
Gender:thriller
Calification: 0.000000
Oscars: 0
--------------------------------------
```

```
========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
2
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

-------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0
```

```
========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
2
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

-------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
```

```
ID: 17
Title: Inception
Duration: 125
Gender:thriller
Calification: 0.000000
Oscars: 0
Prom of the movies: 0
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
3
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
```

```
ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 17
Title: Inception
Duration: 125
Gender:thriller
Calification: 0.000000
Oscars: 0
Prom of the movies: 0
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
4
Tell me the calification of the movie: 0
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
```

```
ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 17
Title: Inception
> sh -c make -s
> ./main
Welcome to our streaming proyect
We have a lot of Movies an Series.
What do you want to do?:
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
```

```
ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0
13,The_Hobbit,124,drama,0,0
ID: 3
 Title: The_Hobbit
 Duration: 124
 Gender: drama
 Calification: 0
 Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0
25,Roma,90,drama,0,0
ID: 5
 Title: Roma
 Duration: 90
 Gender: drama
 Calification: 0
 Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0
317,Inception,125,thriller,0,0
ID: 17
 Title: Inception
 Duration: 125
 Gender: thriller
 Calification: 0
 Oscars: 0
```

```
ID: 17
Title: Inception
Duration: 125
Gender:thriller
Calification: 0.000000
Oscars: 0
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
5
Tell me the gender of the movie: drama
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

-------------------------------------------------------------------
--

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
```

```
ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
Calification: 0.000000
Oscars: 0
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
2
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
Gender:Accion
```

```
========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
4
Tell me the calification of the movie: 0
ID      TITLE       DURATION     GENDER    CALIFICATION    OSCARS

-------------------------------------------------------------------
--

ID: 1
Title: Batman
Duration: 120
Gender:Accion
Calification: 0.000000
Oscars: 0

ID: 3
Title: The_Hobbit
Duration: 124
Gender:drama
Calification: 0.000000
Oscars: 0

ID: 5
Title: Roma
Duration: 90
Gender:drama
```

```
========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series=============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
6
0 = 4,the_Beginning,1,0

ID Serie: 4

Title: the_Beginning

Season: 1

Calification: 0
---
Name of the chapter: the_Beginning
Season: 1
Calification: 0.000000

1 = 4,the_house,1,0

ID Serie: 4

Title: the_house

Season: 1

Calification: 0
---
Name of the chapter: the_house
```

```
--------------------------Series Report-------------------------
---------
0 : 4,Stranger_Things,45,thriller,0,1
ID: 4
Title: Stranger_Things
Duration: 45
Gender: thriller
Calification: 0
Episodes: 1
-----Se cumple-----
----inside object: 0x104bdf0

ID: 4
Title: Stranger_Things
Duration: 45
Gender:thriller
Calification: 0.000000
Episodes = 2
List of Episodes:
Name of the chapter: the_Beginning
Season: 1
Calification: 0.000000


Name of the chapter: the_house
Season: 1
Calification: 0.000000



End of reading the other series
1 : 6,The_Chosen,51,drama,0,1
ID: 6
Title: The_Chosen
Duration: 51
Gender: drama
Calification: 0
Episodes: 1
-----Se cumple-----
----inside object: 0x104c320

ID: 6
```

```
----inside object: 0x104c320

ID: 6
Title: The_Chosen
Duration: 51
Gender:drama
Calification: 0.000000
Episodes = 2
List of Episodes:
Name of the chapter: the_beginning
Season: 1
Calification: 0.000000


Name of the chapter: the_call
Season: 2
Calification: 0.000000



End of reading the other series
It's over
--------------------------------------

========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
9
Tell me the gender of the series: drama
```

```
------------------------------------------------------------------------
----
--------------------------------------


========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
7. Rate a Serie
8. Calification Report
9. Gender Report
======== Episode ==============
10. Check all episodes of a series by ID
11. Calculate average per series
0. Exit
Tell me the option you want to do:
10
Tell me the ID of the series: 4

Name of the chapter: the_Beginning
Season: 1
Calification: 0.000000


Name of the chapter: the_house
Season: 1
Calification: 0.000000

--------------------------------------


========Movies=========
1. Read Movies File
2. Rate a Movie
3. Movie Report
4. Calification Report
5. Gender Report
========Series==============
6. Read Series and Episodes Files
```

```
ame of the chapter: the_house
eason: 1
alification: 0.000000

--------------------------------------

=======Movies=========
. Read Movies File
. Rate a Movie
. Movie Report
. Calification Report
. Gender Report
=======Series=============
. Read Series and Episodes Files
. Rate a Serie
. Calification Report
. Gender Report
======= Episode ==============
0. Check all episodes of a series by ID
1. Calculate average per series
. Exit
ell me the option you want to do:
```

Identification of cases that would prevent the project from functioning properly,

```
77  //Function that calculates the average of each series through its
    episodes and makes the average
78  void Series::calculateAveragePerSerie(){
79      double dCalProm;
80
81      for(int iR = 0; iR < quantity; iR++){
82          dCalProm = arrPtrSeries[iR]->calAverage();
83          arrPtrSeries[iR]->setCalification(dCalProm);
84      }
85
86  }
87
```

In the problem scenario, many tests were conducted in order to arrive at the correct function or project structure. An example of this is when the initial function was not providing the required average, so it was modified until reaching the current function of calculating the average using operator overloading.

```
21: note: jump bypasses variable initialization
                int newValue = 7;
                    ^
./main.cpp:142:21: note: jump bypasses variable initialization
                int oldValue = 3;
                    ^
./main.cpp:141:24: note: jump bypasses variable initialization
                string Movies = "Movies.csv";
                       ^
./main.cpp:156:13: error: cannot jump from switch statement to thi
s case label
            case 5:
            ^
./main.cpp:143:21: note: jump bypasses variable initialization
                int newValue = 7;
                    ^
./main.cpp:142:21: note: jump bypasses variable initialization
                int oldValue = 3;
                    ^
./main.cpp:141:24: note: jump bypasses variable initialization
                string Movies = "Movies.csv";
                       ^
./main.cpp:151:13: error: cannot jump from switch statement to thi
s case label
            case 4:
            ^
./main.cpp:143:21: note: jump bypasses variable initialization
> sh -c make -s
> ./main
```

The error occurs because there are jumps in the program's execution flow that bypass variable initialization at certain points.

Conclusion

In conclusion, this project on modeling a streaming service has provided us with a valuable opportunity to apply Object-Oriented Programming concepts in a practical context. By incorporating inheritance, polymorphism, and operator overloading, we have designed a system that can effectively manage videos, including movies and series, and provide valuable insights into their ratings. Through the use of appropriate UML class diagrams, we have created a structure that captures the relationships and attributes of the video classes accurately.

This project has also emphasized the importance of abstraction and problem analysis in designing software solutions. By identifying the main elements of the problem, abstracting them into suitable class structures, and selecting appropriate algorithms and data structures, we have built an application that meets the requirements of the problem situation.

Furthermore, this project has fostered important attitudes and values such as responsibility, research, and the understanding of engineering methodologies and tools. We have recognized the need for systematic analysis and accurate identification of factors that contribute to observed deviations in order to provide effective problem-solving solutions.

Overall, this project has equipped us with the necessary knowledge and skills to tackle real-world scenarios in the field of Object-Oriented Programming, reinforcing our understanding of inheritance, polymorphism, operator overloading, and their application in software design. By embracing these concepts and methodologies, we are better prepared to develop robust, scalable, and maintainable software solutions in the future.

References Consulted:

1. Anonymous, Programming in C++/Operator Overloading. Retrieved on June 5, 2023. Website: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Sobrecarga_de_Operadores

2. CamiloCorreaUdeA, Polymorphism. Retrieved on June 15, 2023. Website: Polimorfismo - Manejo dinámico de memoria y Polimorfismo (Práctica 4) (codingame.com)
3. CamiloCorreaUdeA. Class Members in C++. Retrieved on June 13,2023 Website: https://www.codingame.com/playgrounds/50557/clases-y-objetos-en-c-practica-1/miembros-de-clase-en-c-variables-y-metodos#:~:text=Los%20m%C3%A9todos%20Get%20y%20Set,actual%20de%20la%20variable%20privada.