Tecnológico de Monterrey

“PROYECTO INTEGRADOR DE PROGRAMACIÓN ORIENTADA A OBJETOS”

Sergio Lopez Urzaiz
A00827462

29 de mayo del 2020

David Alonso Cantú Delgado

Sergio Lopez Urzaiz A00827462 29/5/2020

Abstract:

Para el entregable final de esta clase, se nos pidió escribir un código en c++. Dicho código debe de ser capaz de leer 3 archivos csv, de los cuales obtendrá datos para luego permitir que el usuario vea los datos de diferentes maneras. Este proceso debe de ser obtenido mediante el uso de herencia, polimorfismo, y sobrecarga. Por lo tanto, tendremos que escribir un código main que haga uso de diferentes clases para así poder realizar lo que se nos pide.

Código:

```
// Creamos el menu donde llamaremos a todas las opciones que puede llamar el usuario

bool archivos = false;
int opcion = 0;
double min, max;
string gen, serie, video;
cout << "Introduzca una opcion" << endl;
cout << "0. Salir" << endl;
cout << "1. Cargar archivos" << endl;
cout << "2. Mostrar los videos en general con un cierto rango de calificacion de un cierto genero" << endl;
cout << "3. Mostrar los videos en general de un cierto genero" << endl;
cout << "4. Mostrar los episodios de una determinada serie con un rango de calificacion determinada" << endl;
cout << "5. Mostrar las peliculas con cierto rango de calificacion" << endl;
cout << "6. Calificar un video" << endl;
cout << "7. Tiempo para ver una serie" << endl;
cin >> opcion;
```

El código le presentará al usuario con un menú de 8 opciones, las cuales determinarán qué hará el código.

OPCIÓN 1:

```
ifstream datosPelicula("Files/Peliculas.csv");
ifstream datosSerie("Files/Series.csv");
ifstream datosEpisodios("Files/Episodios.csv");

// Validamos que los archivos existan
try {
    valida("Files/Peliculas.csv");
} catch(runtime_error& e) {
    cout << e.what() << " de peliculas" << endl;
    archivos = false;
}
try {
    valida("Files/Series.csv");
} catch(runtime_error& e) {
    cout << e.what() << " de series" << endl;
    archivos = false;
}
try {
    valida("Files/Episodios.csv");
} catch(runtime_error& e) {
    cout << e.what() << " de episodios" << endl;
    archivos = false;
}
```

Esta opción permitirá al usuario cargar los datos de los archivos que con el directorio que él ha introducido. El código, entonces, hará uso de la función “válida” para asegurarse de que los tres archivos existan. En caso de no existir un archivo, el código no cargará ningun tipo de datos y mandará un mensaje de error.

```
void valida(string nombre) {
    ifstream archivo(nombre);
    if(!archivo.is_open()) {
        throw runtime_error("El archivo no existe");
    }
}
```

En caso de que los tres archivos existan, el código mantendrá el valor “true” en la variable booleana “archivo” y seguirá corriendo.

```
vector<Serie*> series;

int temporadas;

getline(datosSerie, line);
while(getline(datosSerie, line)){
    stringstream ss(line);
    col = 0;
    while(getline(ss, valor, delim)){
        switch(col){
            // ID
            case 0:
                id =atoi(valor.c_str());
                break;
            // Nombre
            case 1:
                nombre = valor;
                for(int i = 0; i < nombre.size(); i++){
                    if(nombre[i] == ' '){
                        nombre[i] = '_';
                    }
                }
                break;
            // Genero
            case 2:
                genero = valor;
                break;
            // Temporadas
            case 3:
                temporadas = atoi(valor.c_str());
                break;
        }
        col++;
    }
    series.push_back(new Serie(id, genero, nombre, temporadas));
}
```

El código ahora pasará por 3 instancias, siendo las similares al código presentado antes. El código leerá el archivo línea por línea e irá asignando los valores a objetos del tipo perteneciente al archivo siendo leído. Después, el código meterá dichos objetos en uno de dos vectores de apuntadores: películas y episodios irán en el vector “vídeos”, mientras que series irán en el vector “series”. Las clases Película, Serie y Episodio heredan de la clase Video. Por esto mismo, sería posible inclusive poner todos los datos en un mismo vector. Pero, para facilitar nuestro proceso, los dividiremos en dos vectores. Al terminar de leer los valores y cargar los vectores, el usuario podrá hacer uso de las otras opciones del menú.

NOTA: En caso de no cargar los valores e intentar usar una de las otras opciones, el código le mencionara al usuario que no ha cargado los datos y no realizará nada. Tambien, el codigo no permitirá que el usuario cargue los valores más de una vez.

OPCIÓN 2:

```
// Mostrar los videos en general con un cierto rango de calificación de un cierto género
if (opcion == 2){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Favor de introducir la calificacion minima: ";
        cin >> min;
        cout << "Favor de introducir la calificacion maxima: ";
        cin >> max;
        cout << "Favor de introducir el genero que desea ver (Drama, Misterio, Accion): ";
        cin >> gen;
        mostrarRangoGenero(videos, min, max, gen);
    }
}
```

En este caso, el código le pedirá al usuario que introduzca una calificación mínima y una calificación máxima. Después le pedirá que introduzca el género que desea ver, y mandara a llamar la función “mostrarRangoGenero” con los parámetros que introdujo el usuario.

```
void mostrarRangoGenero(vector<Video*> videos, double min, double max, string gen) {
    for(int i = 0; i < videos.size(); i++) {
        if((videos[i]->getGenero() == gen) and (*videos[i] >= min) and (*videos[i] <= max)){
            videos[i]->imprimir();
        }
    }
}
```

La función hará uso de operadores y sobrecarga de operadores para solo imprimir aquellos videos que sean parte del género y del rango de calificación deseados por el usuario.

OPCION 3:

```
// Mostrar los videos en general de un cierto género
if (opcion == 3){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Favor de introducir el genero que desea ver (Drama, Misterio, Accion): ";
        cin >> gen;
        mostrarGenero(videos, gen);
    }
}
```

En este caso, el código le pedirá al usuario que simplemente introduzca un género. Luego, el código llamara a la función “mostrarGenero” con el parámetro que el usuario introduzca.

```
void mostrarGenero(vector<Video*> videos, string gen) {
    for(int i = 0; i < videos.size(); i ++) {
        if(videos[i]->getGenero() == gen){
            videos[i]->imprimir();
        }
    }
}
```

La función mostrará los videos que sean parte del género que el usuario desee ver.

OPCIÓN 4:

```
if (opcion == 4){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Favor de introducir la serie de la cual desea ver episodios" << endl;
        for(int i = 0; i < series.size(); i++){
            cout << series[i]->getNombre() << endl;
        }
        cin >> serie;
        cout << "Favor de introducir la calificacion minima: ";
        cin >> min;
        cout << "Favor de introducir la calificacion maxima: ";
        cin >> max;
        mostrarSerieRango(videos, series, min, max, serie);
    }

}
```

En este caso, el código imprimirá todas las series que haya cargado. Después le pedirá al usuario que introduzca la serie que desea ver, asi como la calificacion minima y maxima de los episodios de la serie que el código mostrará. Luego, mandará a llamar la función “mostrarSerieRango” con los parámetros que el usuario introduzca.

```
void mostrarSerieRango(vector<Video*> videos, vector<Serie*> series, double min, double max, string serie) {
    int id;
    for(int i = 0; i < series.size(); i++){
        if(series[i]->getNombre() == serie){
            id = series[i]->getId();
        }
    }
    for(int i = 0; i < videos.size(); i++){
        if((videos[i]->getId() == id) and (*videos[i] >= min) and (*videos[i] <= max)){
            videos[i]->imprimir();
        }
    }
}
```

La función primero obtendrá el ID de la serie y luego imprimirá los episodios que tengan el mismo ID de serie y que estén dentro del rango de calificación deseado por el usuario.

OPCION 5:

```
if (opcion == 5){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Favor de introducir la calificacion minima: ";
        cin >> min;
        cout << "Favor de introducir la calificacion maxima: ";
        cin >> max;
        mostrarPeliculaRango(videos, series, min, max);
    }
}
```

En esta ocasión, el codigo solo le pedirá al usuario que introduzca una calificación mínima y una máxima y luego llamará la función “mostrarPeliculaRango” con dichos parámetros.

```
void mostrarPeliculaRango(vector<Video*> videos, vector<Serie*> series, double min, double max) {
    bool print;
    for(int i = 0; i < videos.size(); i++){
        print = true;
        for(int j = 0; j < series.size(); j++){
            if(videos[i]->getId() == series[j]->getId()){
                print = false;
            }
        }
        if ((print == true) and (*videos[i] >= min) and (*videos[i] <= max)){
            videos[i]->imprimir();
        }
    }
}
```

Las películas tienen ID diferente al de las series. Por lo tanto, el código imprimirá todos aquellos videos que difieran de ID con todas las series y que estén en el rango de calificación deseado por el usuario.

OPCION 6:

```
if (opcion == 6){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Introduzca el video a calificar. Use '_' para los espacios: ";
        for(int i = 0; i < videos.size(); i++){
            videos[i]->imprimir();
        }
        cin >> video;
        cout << "Inroduzca la calificacion que desea asignar: ";
        cin >> calificacion;
        califica(videos, video, calificacion);
    }
}
```

En esta ocasión, el codigo le pedirá al usuario que introduzca el nombre de un video que desea calificar, así como un valor de calificación que le desea asignar. El código luego llamará la función “califica” con los parámetros introducidos por el usuario.

```
void califica(vector<Video*> videos, string video, double calificacion) {
    for(int i = 0; i < videos.size(); i++){
        if(videos[i]->getNombre() == video){
            videos[i]->setCalificacion(calificacion);
            cout << "Le asigno la calificacion: " << calificacion << " a " << videos[i]->getNombre() << endl;
        }
    }
}
```

La función buscará el video que tenga el nombre que introdujo el usuario y le asignará la calificación. Luego, el código desplegará el video y la nueva calificación.

OPCION 7:

```
if (opcion == 7){
    if(archivos == false) {
        cout << "No hay archivos cargados" << endl;
    }
    else {
        cout << "Introduzca la serie que desea saber cuanto tardaria en verla: " << endl;
        for(int i = 0; i < series.size(); i++) {
            cout << series[i]->getNombre() << endl;
        }
        cin >> serie;
        tiempo(videos, series, serie);
    }
}
```

En esta ocasión, el codigo simplemente le pedirá al usuario que introduzca el nombre de una serie y llamara a la función tiempo con la serie que el usuario introdujo.

```
void tiempo(vector<Video*> videos, vector<Serie*> series, string serie) {
    int id, horas = 0, mins = 0;
    string temp;
    for(int i = 0; i < series.size(); i++) {
        if(series[i]->getNombre() == serie) {
            id = series[i]->getId();
        }
    }
    for(int i = 0; i < videos.size(); i++) {
        if (videos[i]->getId() == id){
            temp = videos[i]->getDuracion();
            horas += int(temp[0]) - 48;
            mins += ((int(temp[2]) - 48) * 10 + (int(temp[3]) - 48));
        }

    }
    horas += mins / 60;
    mins = mins % 60;
    cout << "Tardaria " << horas << " horas con " << mins << " minutos" << endl;
}
```

La función entonces sumará la duración de todos los episodios de la serie que el usuario introdujo e imprimirá cuantas horas y minutos tarda el usuario en terminarla.