CMP-5015Y Coursework 3 - Offline Movie Database in C++

100251167 (afz18mcu)

Saturday 25th April, 2020 18:11

PDF prepared using LaTeX template v1.00 .

☑ I agree that by submitting a PDF generated from this template I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

Movie.h

```
/**
 * Created by Rohan on 25/03/2020.
 * Last Edited on 25/4/2020. Added comments.
**/

#ifndef OFFLINEMOVIEDATABASE_CPP_MOVIE_H
#define OFFLINEMOVIEDATABASE_CPP_MOVIE_H

#include <string>
#include <fstream>
#include <sstream>
#include <iostream>
#include <ostream>

using namespace std;
namespace movie
{
    class Movie
    {

    public:
        //nts* unsigned is used to make the variable only represent natural
           numbers
        string filmTitle;
        int releaseYear;
        string ageRating;
        string genre;
        int length;
        int viewerRating;


    public:
        //setter methods for title, release year, age rating, genre, length and
           viewer rating
        //nts*-inline function is a function that is expanded in line when it is
           called, they are faster cos no push and pop on and off the stack
        inline void setFilmTitle(string &newFilmTitle)
        {
            this->filmTitle = newFilmTitle;
        }

        inline void setReleaseYear(int &newReleaseYear)
        {
            this->releaseYear = newReleaseYear;
        }

        inline void setAgeRating(string &newAgeRating)
        {
            this->ageRating = newAgeRating;
        }

        inline void setGenre(string &newGenre)
        {
            this->genre = newGenre;
        }

        inline void setLength(int &newLength)
        {
            this->length = newLength;
        }
        inline void setViewerRating(int &newViewerRating)
        {
            this->viewerRating = newViewerRating;
        }

        // getter methods for title, release year, age rating, genre, length and
           viewer rating
        inline string getFilmTitle()
        {
            return this->filmTitle;
        }

        inline const int getReleaseYear()
        {
            return this->releaseYear;
        }

        inline string getAgeRating()
        {
            return this->ageRating;
        }

        inline string getGenre()
        {
            return this->genre;
        }

        inline const int getLength()
        {
            return this->length;
        }

        inline const int getViewerRating()
        {
            return this->viewerRating;
        }

        //constructor for Movie class
        Movie(string newFilmTitle, int newReleaseYear, string newAgeRating,
           string newGenre,
              int newLength, int newViewerRating);

        //nts*-the equivalent of a toString from Java in Cpp is the write()
           method

        //operator in ostream is like the override for Java's toString method. It
            is an overloading operator
        //change so that it prints like the .txt file
        friend inline ostream& operator<<(ostream &stream, movie::Movie &movie1)
        {
            stream  << "Film Title: " << movie1.filmTitle << " | "
                    << "Year of Release: " << movie1.releaseYear << " | "
                    << "Age Rating: " << movie1.ageRating << " | "
                    << "Genre: " << movie1.genre << " | "
                    << "Length of Film: " << movie1.length << " | "
                    << "Viewer Rating: " << movie1.viewerRating << " | " << endl;
            return stream;
        }

        //overloading the input operator
        friend inline istream& operator>> (istream& stream, Movie movie1);
    };
}
```

```
void testMovie();

#endif //OFFLINEMOVIEDATABASE_CPP_MOVIE_H
```

Movie.cpp

```cpp
/**
 * Created by Rohan on 25/03/2020.
 * Last Edited on 25/4/2020. Added comments.
**/

#include "Movie.h"
/**
 * Movie.h and Movie.cpp - A Movie object describes the information stored about
     a particular film,
 * such that there will be a separate Movie object for each film held in the
 * database. The class should have a suitable collection of constructors,
     accessor methods
 * etc. and the stream I/O and relational operators should be implemented.
 **/

// create a movie object for each film
/* movie object should contain :
  1. Title
  2. Year of Release
  3. Age rating
  4. Genre
  5. Length of film (in minutes)
  6. User Rating
*/

using namespace std;
    // initialising Movie constructor method from .h file
    movie::Movie::Movie(string newFilmTitle, int newReleaseYear, string
       newAgeRating, string newGenre, int newLength,
                        int newViewerRating)
    {
        this->filmTitle = newFilmTitle;
        this->releaseYear = newReleaseYear;
        this->ageRating = newAgeRating;
        this->genre = newGenre;
        this->length = newLength;
        this->viewerRating = newViewerRating;
    }

// test harness for Movie.h and Movie.cpp
// creating a new movie object
void testMovie()
{
    movie::Movie newMovie = movie::Movie("Hello", 1990, "UG", "Adventure", 120,
       0);
    cout << newMovie << endl;
}
```

MovieDatabase.h

```cpp
/**
 * Created by Rohan on 25/03/2020.
 * Last Edited on 25/4/2020. Added comments.
**/

#ifndef OFFLINEMOVIEDATABASE_CPP_MOVIEDATABASE_H
#define OFFLINEMOVIEDATABASE_CPP_MOVIEDATABASE_H

#include <vector>
#include <algorithm>
#include "Movie.h"

using namespace std;

// nts*- allows to create new names for types
namespace movie{
    class MovieDatabase
    {
    private:
        // vectors are the c++ equivalent of an arraylist in java
        // creating a vector called moviesVector to store all the different
           movies from films.txt
        vector<movie::Movie> moviesVector;

    public:
        // empty MovieDatabase constructor
        MovieDatabase()
        {
        }

        // read in the file and use getline to split the the line with the
           delimiter specified and the push it into the vector.
        MovieDatabase(string fileName)
        {
            // reads in the file
            ifstream file(fileName);

            string titleToken;
            string yearToken;
            string ageToken;
            string genreToken;
            string lengthToken;
            string viewToken;

            // while loop to keep going till we reach the reach the end of the
               file
            while (!file.eof())
            {
                // getline take a string variable and a delimiter. ot goes from
                   where you are in the file to what you have specified
                // it will store everything between that in the variable you give
                   it
                // skips  first "
                getline(file, titleToken, '"');
                //gets the name of the film/everything between "
                getline(file, titleToken, '"');

                // in this case we skip the first comma and get everything after
                   it until the next comma.
                getline(file, yearToken, ',');
                getline(file, yearToken, ',');
                getline(file, ageToken, '"');
                getline(file, ageToken, '"');

                getline(file, genreToken, '"');
                getline(file, genreToken, '"');

                getline(file, lengthToken, ',');
                getline(file, lengthToken, ',');

                // we do not need a delimiter here as this is the last item on
                    the line.
                getline(file, viewToken);

                // push_back adds the movie to the back of the vector
                // stoi is use to convert a string to an int
                moviesVector.push_back(movie::Movie(titleToken, stoi(yearToken),
                    ageToken,
                                                    genreToken, stoi(lengthToken)
                                                        , stoi(viewToken)));

                // if there is an empty line anywhere in the file it skips said
                    line.
                if(file.peek() == '\n')
                    file.get();
            }
        }

        // addMovie method adds movie to the vector
        inline void addMovie(Movie movie)
        {
            moviesVector.push_back(movie);
        }

        // overloading the output operator by going through each movie in the
            vector and send it to the stream
        friend inline ostream& operator<<(ostream &stream, movie::MovieDatabase &
            movieDB)
        {
            for(movie::Movie movie1 : movieDB.moviesVector)
                stream << movie1;
            return stream;
        }


        // Sorting Functions
        // using lambdas to sort the movies
        // lambdas are anonymous functions that can capture the local variables
            of the scope in which they are enclosed

        // sort in chronological order (ascending and descending but will only be
             using ascending in this case)
        inline void sortChronologyAsc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){
                return a.getReleaseYear() < b.getReleaseYear();});
        }
        inline void sortChronologyDesc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){
                return a.getReleaseYear() > b.getReleaseYear();});
        }
```

```cpp
        // sort by title length (ascending and  descending)
        inline void sortTitleLengthAsc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getFilmTitle().length() < b.getFilmTitle().length();});
        }
        inline void sortTitleLengthDesc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getFilmTitle().length() > b.getFilmTitle().length();});
        }

        // sort by film length in minutes (ascending and descending)
        inline void sortLengthOfFilmAsc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getLength() < b.getLength();});
        }
        inline void sortLengthOfFilmDesc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getLength() > b.getLength();});
        }

        //sort by viewer rating (ascending and descending)
        inline void sortViewerRatingAsc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getViewerRating() < b.getViewerRating();});
        }
        inline void sortViewerRatingDesc()
        {
            sort(moviesVector.begin(), moviesVector.end(), [](Movie a, Movie b){ return a.getViewerRating() > b.getViewerRating();});
        }

        // defining the getBy methods that will be used to get different items in the film
        movie::MovieDatabase getByReleaseYear(int releaseYear);
        movie::MovieDatabase getByGenre(string genre);
        movie::MovieDatabase getByViewerRating(int viewerRating);
        movie::MovieDatabase getByAgeRating(string ageRating);

        // indexOfVector is used to get the movie at the index that is specified in main.cpp
        movie::Movie indexOfVector(int vectorIndex)
        {
            return moviesVector.at(vectorIndex);
        }

        //overloading the input operator but not used here.
        friend  inline ifstream& operator>>(ifstream& file, MovieDatabase movieDatabase1);
    };
}
void testMovieDatabase();
```

```
#endif //OFFLINEMOVIEDATABASE_CPP_MOVIEDATABASE_H
```

MovieDatabase.cpp

```cpp
/**
 * Created by Rohan on 25/03/2020.
 * Last Edited on 25/4/2020. Added comments.
**/

#include "MovieDatabase.h"
/**
 * MovieDatabase.h and MovieDatabase.cpp - A collection of Movie objects, one for
 * each film described in the data file. The class should provide overloaded I/O
    operators
 * for reading the data from file and displaying the database on the terminal and
 * for answering the database queries. Rather than writing a program that only
    implements the
 * current specifications, we should write maintainable programs that are easily
    extended
 * to answer a variety of database queries, without writing a lot of extra code (
    i.e. methods
 * that answer generic queries are better than methods that answer very specific
    queries).
**/

using namespace std;

// gets the movie that was released in the specified year
movie::MovieDatabase movie::MovieDatabase::getByReleaseYear(int releaseYear)
{
    // getting all the movies release in the specified year
    // create a temporary MovieDatabase to collect the movies that fulfil the
        condition
    MovieDatabase subMovieDatabase = MovieDatabase();
    // goes through the vector and if the the parameter, releaseYear is the same
        as the movie's release year
    // it returns all the movies where that condition is fulfilled
    for(Movie movie : moviesVector)
        if(movie.getReleaseYear() == releaseYear)
            subMovieDatabase.addMovie(movie);
    return subMovieDatabase;
}

//get genre using getline to split the slashes and then search for the genre we
    are specified
movie::MovieDatabase movie::MovieDatabase::getByGenre(string genreSearch)
{
    MovieDatabase subMovieDatabase = MovieDatabase();
    // goes through the vector and if the parameter, genreSearch is the same as
        the movie's genre
    // it returns all the movies where that condition is fulfilled
    for(Movie movie : moviesVector)
    {
        // because each movie has mutiple genres, we split the genre by the slash
            and then search
        string genre = movie.getGenre();
        istringstream iStringStream(genre);
        string genreToken;
        while(!iStringStream.eof())
        {
            getline(iStringStream, genreToken, '/');
            if (genreToken == genreSearch)
                subMovieDatabase.addMovie(movie);
        }
    }
    return subMovieDatabase;
}

// get the movie with the specified viewer rating
movie::MovieDatabase movie::MovieDatabase::getByViewerRating(int viewerRating)
{
    MovieDatabase subMovieDatabase = MovieDatabase();
    // goes through the vector and if the parameter, viewerRating is the same as
       the movie's viewer rating
    // it returns all the movies where that condition is fulfilled
    for(Movie movie : moviesVector)
        if(movie.getViewerRating() == viewerRating)
            subMovieDatabase.addMovie(movie);
    return subMovieDatabase;
}

// gets the movie with the specified age rating or certificate
movie::MovieDatabase movie::MovieDatabase::getByAgeRating(string ageRating)
{
    MovieDatabase subMovieDatabase = MovieDatabase();
    // goes through the vector and if the parameter, ageRating is the same as the
        movie's age rating/certificate
    // it returns all the movies where that condition fulfilled
    for(Movie movie : moviesVector)
        if(movie.getAgeRating() == ageRating)
            subMovieDatabase.addMovie(movie);
    return subMovieDatabase;
}

//test harness for MovieDatabase.h and MovieDatabase.cpp
void testMovieDatabase()
{
    movie::MovieDatabase movieDatabase = movie::MovieDatabase("films.txt");
    movieDatabase.sortChronologyAsc();
    movieDatabase.sortTitleLengthDesc();
    cout << movieDatabase;
}
```

main.cpp

```
/**
 * Created by Rohan on 25/03/2020.
 * Last Edited on 25/4/2020. Added comments.
**/

#include <iostream>
#include <fstream>
#include <string>
#include "Movie.h"
#include "MovieDatabase.h"

using namespace std;

int main()
{
    // creates a movie database with all the movies in films.txt
    movie::MovieDatabase movieDatabase = movie::MovieDatabase("films.txt");

    // test harnesses for MovieDatabase and Movie
    //testMovieDatabase();
    //testMovie();

    // Chronologically Sort all the movies
    movieDatabase.sortChronologyAsc();
    cout << "Chronologically Ordered" << endl << movieDatabase << endl;

    // Displays the Third longest Film-Noir
    movie::MovieDatabase filmNoir = movieDatabase.getByGenre("Film-Noir");
    filmNoir.sortLengthOfFilmDesc();
    movie::Movie thirdLongest = filmNoir.indexOfVector(2);
    cout << "Third Longest Film-Noir" << endl << thirdLongest << endl;

    // Displays the Eight most recent UNRATED Film
    movie::MovieDatabase unrated = movieDatabase.getByAgeRating("UNRATED");
    unrated.sortChronologyDesc();
    movie::Movie eightMostRecent = unrated.indexOfVector(7);
    cout << "Eight Most Recent UNRATED Film" << endl << eightMostRecent << endl;

    // Displays the film with the Longest Title
    movieDatabase.sortTitleLengthDesc();
    movie::Movie longestTitle = movieDatabase.indexOfVector(0);
    cout << "Film with The Longest Title" << endl << longestTitle << endl;

}
```