Національний університет “Львівська політехніка”

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Об’єктно-орієнтоване програмування»**

**На тему:  
«***Додаток менеджменту списку фільмів***»**

Студента групи   *ПЗ-25*

спеціальності 6.121

“Інженерія програмного забезпечення”

*Сьомко П.Я.*

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів\_\_\_Оцінка ECTS\_\_\_

Члени комісії          \_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів 2021

**Зміст**

**1. Технічне завдання**

1. Розробити програму засобами ООП на мові С++/С#/Java згідно вказаного варіанту (Варіант №16).
2. Передбачити віконний режим роботи програми та інтерфейс користувача.
3. Передбачити ввід даних у двох режимах:

* З клавіатури;
* З файлу;

1. Передбачити у програмі виняткові ситуації.
2. Продемонструвати викладачу роботу розробленої програми.
3. Сформувати звіт курсової роботи обсягом не менше 20 сторінок.

**Завдання варіанту №16**

Створити таблицю

**№ |  Прізвище  |   Ім‘я   |  Рік народж.  |  Стать  |   t   |  Гемоглобін**

Назва фільму | Режисер | Рік випуску | Актори | Бюджет | Країна виробництва | Тривалість

1) Алгоритмом простої вибірки відсортувати записи за Країною виробництва.

2) Визначити Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

3) За заданою країною виробництва знайти всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

4) Для кожного Режисера визначити фільм з найбільшою тривалістю.

5) Знайти найдорожчий та найстарший фільм одночасно.

6) За заданим актором визначити всі фільми, в яких він (вона) знімались.

7) Встановити найбільш популярного актора.

Для [клас](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54317&displayformat=dictionary)у створити: 1) [Конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) за замовчуванням; 2) [Конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) з параметрами; 3) [конструктор](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54330&displayformat=dictionary) копій; 4) перевизначити операції >>, << для зчитування та запису у файл. Для демонстрації роботи програми використати засоби візуального середовища програмування.

# 2. Алгоритм розв’язку задачі

2.1. Задача зчитування даних з файлу.

Алгоритм RF.

RF1. Виклик функції зчитування даних.

RF2. Користувач обирає шлях до файлу в файловому провіднику.

RF3. Перевірка чи шлях до файлу існує і файл не порожній. Якщо результат false, виклик відповідного повідомлення, переходимо до пункту RF7.

RF4. Відкриваємо файл.

RF5. Порядкове зчитування даних.

RF6. Закриття файлу.

RF7. Кінець.

2.2. Задача додавання елемента.

Алгоритм AN.

AN1. Виклик функції для додавання нового елемента.

AN2. Ввести дані.

AN3. Перевірка даних. Якщо дані не коректні, перейти до пункту АN5.

AN4. Додавання елемента в кінець списку.

AN5. Кінець.

2.3. Задача сортування списку.

Алгоритм SE.

SE1. Виклик функції сортування.

SE2. Вибір критерія сортування.

SE3. Відсортувати список відомим алгоритмом.

SE4. Кінець.

2.4. Задача знаходження акторів за певними критеріями.

Алгоритм FA.

FA1. Виклик функції для знаходження акторів.

FA2. Задати критерії пошуку.

FA3. Виділити список акторів зі списку фільмів.

FA4. Знаходження елементів, які відповідають заданим критеріям.

FA5. Повернути знайдені дані.

FA6. Кінець.

2.5. Задача запису у файл.

Алгоритм WF.

WF1. Виклик функції запису і вибір користувачем файлу.

WF2. Перевірка шляху до файлу. Якщо не існує, виклик відповідного повідомлення, перехід до пункту WF5.

WF3. Перевірка чи файл відкритий. Якщо ні, виклик відповідного повідомлення, перехід до пункту WF5.

WF4. Запис даних.

WF5. Закриття файлу.

WF6. Кінець.

**4. Код програми.**

4.1 FillmData.cs

using FilmApp.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FilmApp

{

public class FilmData

{

public static int counter;

protected string \_name;

protected string \_director;

protected int \_year;

protected ActorsList \_actors;

protected double \_price;

protected string \_country;

protected double \_time;

public string Name { get { return \_name; } set { \_name = value; } }

public string Director { get { return \_director; } set { \_director = value; } }

public int Year { get { return \_year; } set { \_year = value; } }

public ActorsList Actors { get { return \_actors; } set { \_actors = value; } }

public double Price { get { return \_price; } set { \_price = value; } }

public string Country { get { return \_country; } set { \_country = value; } }

public double Time { get { return \_time; } set { \_time = value; } }

static FilmData() { counter = 0; }

public FilmData() { counter++; }

public FilmData(string name,string director, int year, ActorsList actors,

double price, string country, double time)

{

\_name = name;

\_director = director;

\_year = year;

\_actors = actors;

\_price = price;

\_country = country;

\_time = time;

counter++;

}

public FilmData(FilmData val)

{

\_name = val.\_name;

\_director = val.\_director;

\_year = val.\_year;

\_actors = val.\_actors;

\_price = val.\_price;

\_country = val.\_country;

\_time = val.\_time;

counter++;

}

}

}

4.2 FilmList.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace FilmApp.Model

{

public class FilmList

{

List<FilmData> list = null;

public FilmList()

{

list = new List<FilmData>();

}

public FilmList(List<FilmData> list)

{

this.list = list;

}

public List<FilmData> List { get { return list; } set { list = value; } }

public void SortByCountry()

{

//Алгоритмом простої вибірки відсортувати записи за Країною виробництва

if (list == null || list.Count == 0)

throw new NullReferenceException("Your list is empty");

FilmData fix, min;

int min\_index;

for (int j = 0; j < list.Count - 1; j++)

{

fix = list[j];

min = list[j + 1];

min\_index = j + 1;

for (int i = j + 1; i < list.Count; i++)

{

if (String.Compare(min.Country, list[i].Country, true) > 0)

{

min = list[i];

min\_index = i;

}

}

//міняємо місцями фіксований елемент і мінімальний

if (String.Compare(fix.Country, min.Country, true) > 0)

{

list[j] = min;

list[min\_index] = fix;

}

}

}

public string FindPopularActor()

{

// Встановити найбільш популярного актора

Dictionary<string, int> actors = new Dictionary<string, int>();

//формуємо словник акторів і кількістю їхніх зйомок у фільмах

for (int i = 0; i < list.Count; i++)

{

for (int j = 0; j < list[j].Actors.Actors.Count; j++)

{

if(!actors.TryAdd(list[i].Actors.Actors[j], 1))

{

actors[list[i].Actors.Actors[j]]++;

}

}

}

int max = 0;

//знаходимо максимальну кількість зйомок

foreach (var item in actors)

{

if(max < item.Value)

{

max = item.Value;

}

}

//повертаємо актора з максмальною кількістю зйомок

foreach (var item in actors)

{

if (max == item.Value)

return item.Key;

}

return string.Empty;

}

public List<FilmData> FindAllFilmsWithActor(string actor\_name)

{

// За заданим актором визначити всі фільми, в яких він (вона) знімались

if (list == null || list.Count == 0)

throw new NullReferenceException("Your list is empty!");

List<FilmData> films = new List<FilmData>();

//знаходимо фільми з потрібним актором

foreach (var film in list)

{

foreach (var actor in film.Actors.Actors)

{

if(actor == actor\_name)

{

films.Add(film);

}

}

}

return films;

}

public List<FilmData> FindForDirectorsLongestFilm()

{

//Для кожного Режисера визначити фільм з найбільшою тривалістю.

if (list == null || list.Count == 0)

throw new NullReferenceException("Your list is empty");

List<FilmData> films = new List<FilmData>();

Dictionary<string, FilmData> longest\_films = new Dictionary<string, FilmData>();

double max\_time = 0;

foreach (var film in list)

{

// find max time of director films

foreach (var max\_time\_film in list)

{

if(max\_time\_film.Director == film.Director)

{

if(max\_time < max\_time\_film.Time)

{

max\_time = max\_time\_film.Time;

}

}

}

// add film with max time to new list

foreach (var max\_time\_film in list)

{

if (max\_time\_film.Director == film.Director)

{

if (max\_time == max\_time\_film.Time)

{

longest\_films.TryAdd(max\_time\_film.Director, max\_time\_film);

break;

}

}

}

max\_time = 0;

}

films = longest\_films.Select(val => val.Value).ToList();

return films;

}

public FilmList FindTheMostExpensiveAndOldest()

{

///Знайти найдорожчий та найстарший фільм одночасно

if (list.Count == 0)

{

throw new NullReferenceException("Your list is empty");

}

else if (list.Count == 1)

{

return this;

}

List<FilmData> films = list.ToList() ;

//шукаємо найдорожчі фільми

List<FilmData> max\_price = list.Where(val => val.Price == list.Max(film => film.Price)).ToList<FilmData>();

//шукаємо найстарші фільми

films = max\_price.Where(val => val.Time == max\_price.Min(film => film.Time)).ToList<FilmData>();

return new FilmList(films);

}

public List<FilmData> FindSameDirectorsAndLowesBudget()

{

///Визначити Назви фільмів, в яких однакові

///режисери та найменші бюджети одночасно.

if (list == null || list.Count == 0)

throw new NullReferenceException("Your list is empty!");

//групуємо фільми по режисерах

var group\_films = from filmdata in list group filmdata by filmdata.Director;

List<double> film\_price = new List<double>(group\_films.Count());

int i = 0;

//рахуємо суми їхніх бюджетів

foreach (IGrouping<string, FilmData> g in group\_films)

{

film\_price.Add(new double());

foreach (var film in g)

film\_price[i] += film.Price;

i++;

}

//знаходимо індекс мінімального

int index = film\_price.IndexOf(film\_price.Min());

List<FilmData> films = group\_films.ElementAt(index).Select(g => g).ToList();

return films;

}

public List<FilmData> FindByCountry(string country)

{

///За заданою країною виробництва знайти всі фільми,

///в яких найбільші бюджети і найменша тривалість одночасно.

if (list == null || list.Count == 0)

throw new NullReferenceException("Your list is empty!");

//вибираємо фільми із заданої країни

List<FilmData> films = list.Where(val => val.Country == country).ToList<FilmData>();

if (films.Count == 0)

throw new ArgumentException("There is no films from this country");

if (films.Count == 1)

return films;

// вибираємо фільми з максимальною ціною

List<FilmData> max\_price = films.Where(

film => film.Price ==

films.Max(val => val.Price)).ToList<FilmData>();

// вибираємо фільми з найменшою тривалістю

List<FilmData> min\_time = max\_price.Where(

val => val.Time ==

max\_price.Min(film => film.Time)).ToList<FilmData>();

return min\_time;

}

}

}

4.3 FilmsData

using FilmApp.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace FilmApp.AppInteraction

{

public enum ToChange { Yes, No}

public class FilmsData

{

public delegate void FilmListEvent(FilmList list, ToChange value, string text);

public static event FilmListEvent FilmsFill;

public void UpDateFilms(FilmList filmList, ToChange value, string text)

{

FilmsFill?.Invoke(filmList, value, text);

}

}

}

4.4 ActorList.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FilmApp.Model

{

public class ActorsList

{

List<string> actors = null;

public List<string> Actors { get { return actors; } set { actors = value; } }

public ActorsList() { actors = new List<string>(); }

public ActorsList(string name)

{

if(actors==null)

actors = new List<string>();

actors.Add(name);

}

public ActorsList(List<string> actors\_list)

{

actors = actors\_list;

}

public ActorsList(ActorsList \_list)

{

actors = \_list.actors;

}

public override string ToString()

{

string str = string.Empty;

foreach (var actor in actors)

{

str += actor + " ; ";

}

return str;

}

}

}

4.5 MainWindow.xaml.cs

using FilmApp.Model;

using System;

using System.Windows;

using System.IO;

using Newtonsoft.Json;

using FilmApp.AppInteraction;

using System.Diagnostics;

using Microsoft.Win32;

namespace FilmApp

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

FilmList list =null;

FilmsData filmsData;

public MainWindow()

{

InitializeComponent();

list = new FilmList();

filmsData = new FilmsData();

}

private void BSort\_Click(object sender, RoutedEventArgs e)

{

try

{

list.SortByCountry();

filmsData.UpDateFilms(list,ToChange.Yes, "Your films sorted by country:");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BWriteFilmsToFile\_Click(object sender, RoutedEventArgs e)

{

try

{

string films = System.Text.Json.JsonSerializer.Serialize(list);

File.WriteAllText($"{Directory.GetCurrentDirectory()}/films.json", films);

MessageBox.Show("Your data saved to file.", "Successe", MessageBoxButton.OK, MessageBoxImage.Asterisk);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BReadFilmsFromFile\_Click(object sender, RoutedEventArgs e)

{

try

{

if(list!=null && list.List.Count > 0)

{

MessageBoxResult result = MessageBox.Show("Do you want to read new data? Your old data will be clear.", "Question",

MessageBoxButton.YesNo, MessageBoxImage.Question);

if (result == MessageBoxResult.No)

{

return;

}

}

OpenFileDialog fileDialog = new OpenFileDialog();

fileDialog.Multiselect = false;

fileDialog.Filter = "Text files|\*.json\*.\*";

fileDialog.DefaultExt = ".txt";

Nullable<bool> dialogOk = fileDialog.ShowDialog();

string filePath=string.Empty;

if (dialogOk == true)

{

filePath = fileDialog.FileNames[0];

}

if (!filePath.Contains(".json"))

{

throw new Exception("We don't work with this type of files!");

}

FilmData.counter = 0;

//$"{Directory.GetCurrentDirectory()}/films.json")

using (StreamReader read = new StreamReader(filePath))

{

string json = read.ReadToEnd();

if (json == string.Empty)

throw new Exception("Your file is empty! Try to open another file.");

list = JsonConvert.DeserializeObject<FilmList>(json);

}

if (list.List.Count == 0)

throw new Exception("We can't convert your file data to film. Fix your data to json format and try again.");

filmsData.UpDateFilms(list, ToChange.Yes,"Your films:");

MessageBox.Show("You successfully readed films from file", "Success", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BFindPopularActor\_Click(object sender, RoutedEventArgs e)

{

try

{

MessageBox.Show(list.FindPopularActor(), "The most popular actor:",MessageBoxButton.OK ,MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BShowAllList\_Click(object sender, RoutedEventArgs e)

{

try

{

if (list.List.Count == 0)

throw new Exception("Your list is empty!");

filmsData.UpDateFilms(list, ToChange.No, "Your films:");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BFindOldestExpesive\_Click(object sender, RoutedEventArgs e)

{

try

{

filmsData.UpDateFilms(list.FindTheMostExpensiveAndOldest(), ToChange.No,"Your the most expensive and the oldest films:");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BOpenMenu\_Click(object sender, RoutedEventArgs e)

{

BOpenMenu.Visibility = Visibility.Collapsed;

BCloseMenu.Visibility = Visibility.Visible;

MainGrid.Margin = new Thickness(200, 0, 0, 0);

}

private void BCloseMenu\_Click(object sender, RoutedEventArgs e)

{

BOpenMenu.Visibility = Visibility.Visible;

BCloseMenu.Visibility = Visibility.Collapsed;

MainGrid.Margin = new Thickness(70, 0, 0, 0);

}

private void BInfo\_Click(object sender, RoutedEventArgs e)

{

MessageBox.Show("This app created by Somko Pavlo.\nStudent of NULP, PZ-25.", "Info", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

4.6 Dashboard.xaml.cs

using FilmApp.AppInteraction;

using FilmApp.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Controls.Primitives;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace FilmApp.MVVM.ViewModel

{

/// <summary>

/// Interaction logic for DashboardViewModel.xaml

/// </summary>

public partial class DashboardViewModel : UserControl

{

private readonly FilmsData \_filmdata;

private FilmList list;

public DashboardViewModel()

{

InitializeComponent();

FilmTable.ItemsSource = new List<FilmData> { };

\_filmdata = new FilmsData();

FilmsData.FilmsFill += OnInteract;

}

private void OnInteract(FilmList obj, ToChange value, string text)

{

try

{

if (value == ToChange.Yes)

{

amount\_fo\_films.Text = "Amount of films in app: " + FilmData.counter.ToString();

list = obj;

}

FilmTable.ItemsSource = obj.List;

FilmTable.Items.Refresh();

info\_text.Text = text;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BFindByCountry\_Click(object sender, RoutedEventArgs e)

{

try

{

if (parametr\_to\_find.Text == string.Empty || parametr\_to\_find.Text.Any(c => !char.IsLetter(c)))

throw new Exception("Your parameter is not valid.");

if (list.FindByCountry(parametr\_to\_find.Text).Count == 0)

throw new Exception("There is not films from input country!");

FilmTable.ItemsSource = list.FindByCountry(parametr\_to\_find.Text);

FilmTable.Items.Refresh();

info\_text.Text = "Your films from " + parametr\_to\_find.Text + " with the bigest budget and lowest time:";

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void FindFilmsWithActor\_Click(object sender, RoutedEventArgs e)

{

try

{

if (parametr\_to\_find.Text == string.Empty || parametr\_to\_find.Text.Any(c => char.IsDigit(c)))

throw new Exception("Your parameter is not valid.");

if (list.FindAllFilmsWithActor(parametr\_to\_find.Text).Count == 0)

throw new Exception("There is not films with input actor!");

FilmTable.ItemsSource = list.FindAllFilmsWithActor(parametr\_to\_find.Text);

FilmTable.Items.Refresh();

info\_text.Text = "Your films with " + parametr\_to\_find.Text ;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BFindWithIdentDirAndLowesPRice\_Click(object sender, RoutedEventArgs e)

{

try

{

FilmTable.ItemsSource = list.FindSameDirectorsAndLowesBudget();

FilmTable.Items.Refresh();

info\_text.Text = "Your films with identical directors and lowest price:";

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void FindLongesstForEachDirector\_Click(object sender, RoutedEventArgs e)

{

try

{

FilmTable.ItemsSource = list.FindForDirectorsLongestFilm();

FilmTable.Items.Refresh();

info\_text.Text = "Your the longest films for each director :";

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void BAddNewFilm\_Click(object sender, RoutedEventArgs e)

{

try

{

if (film\_name.Text.Any(c => !char.IsLetter(c)) ||

film\_director.Text.Any(c => !char.IsLetter(c)) ||

film\_country.Text.Any(c => !char.IsLetter(c)) ||

film\_actors.Text.Any(c => char.IsDigit(c)) ||

film\_budget.Text.Any(c => !char.IsDigit(c)) ||

film\_time.Text.Any(c => !char.IsDigit(c)) ||

film\_year.Text.Any(c => !char.IsDigit(c)))

{

throw new Exception("There is an error in your data, so we can't add this to films list. Fix your data and try again.");

}

else

{

var film = new FilmData

{

Name = film\_name.Text,

Director = film\_director.Text,

Actors = new ActorsList(film\_actors.Text.Split(',').ToList<string>()),

Country = film\_country.Text,

Price = Convert.ToDouble(film\_budget.Text),

Time = Convert.ToDouble(film\_time.Text),

Year = Convert.ToInt32(film\_year.Text),

};

list.List.Add(film);

FilmTable.ItemsSource = list.List;

FilmTable.Items.Refresh();

}

info\_text.Text = "Your films:";

MessageBox.Show("You successfully added film", "Success", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

# 5. Протокол роботи

Програма виконує такі завдання:

* Алгоритмом простої вибірки сортує записи за Країною виробництва

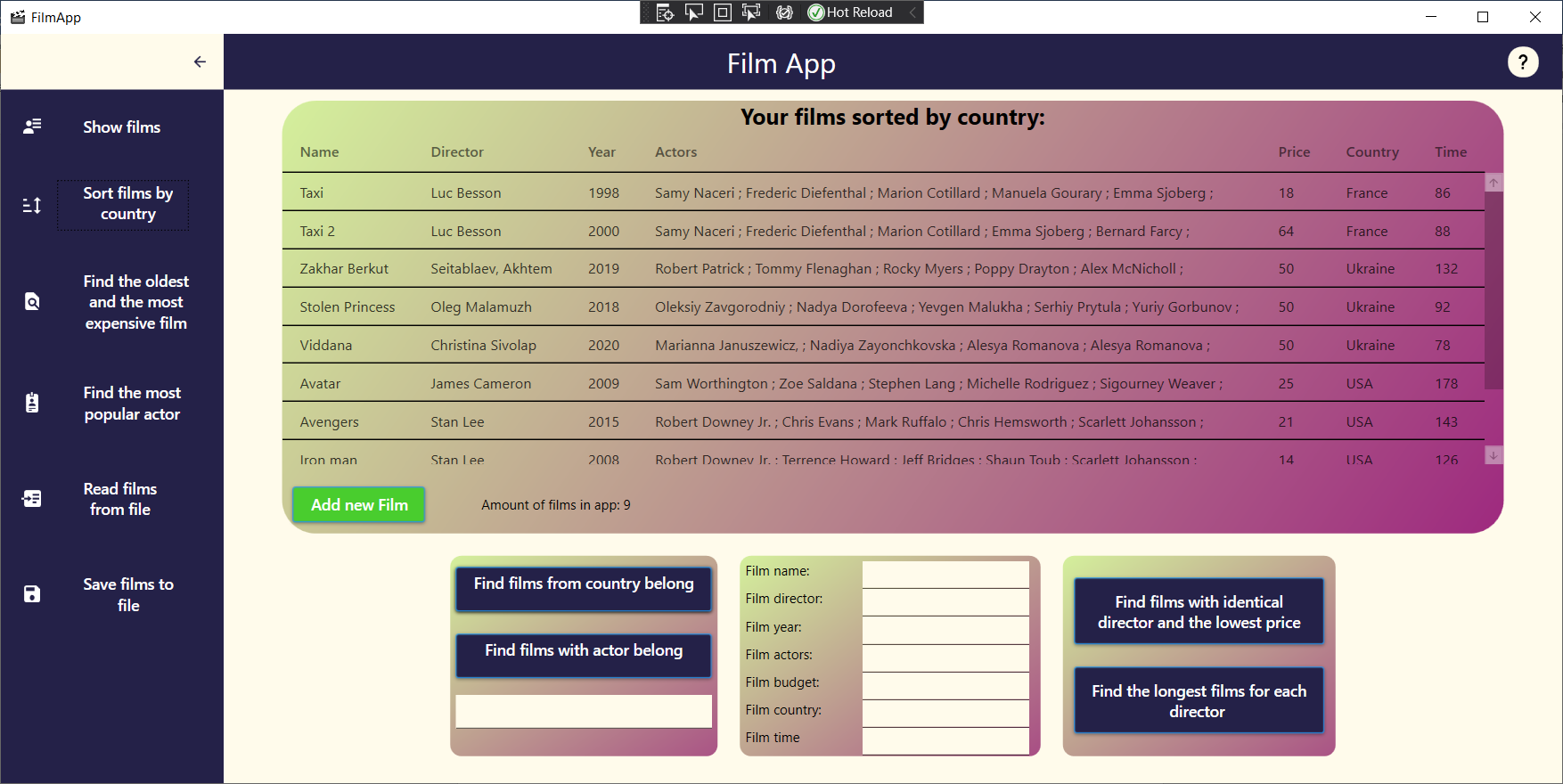


Рисунок 5.1 посортовані фільми за країною

* Визначає Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

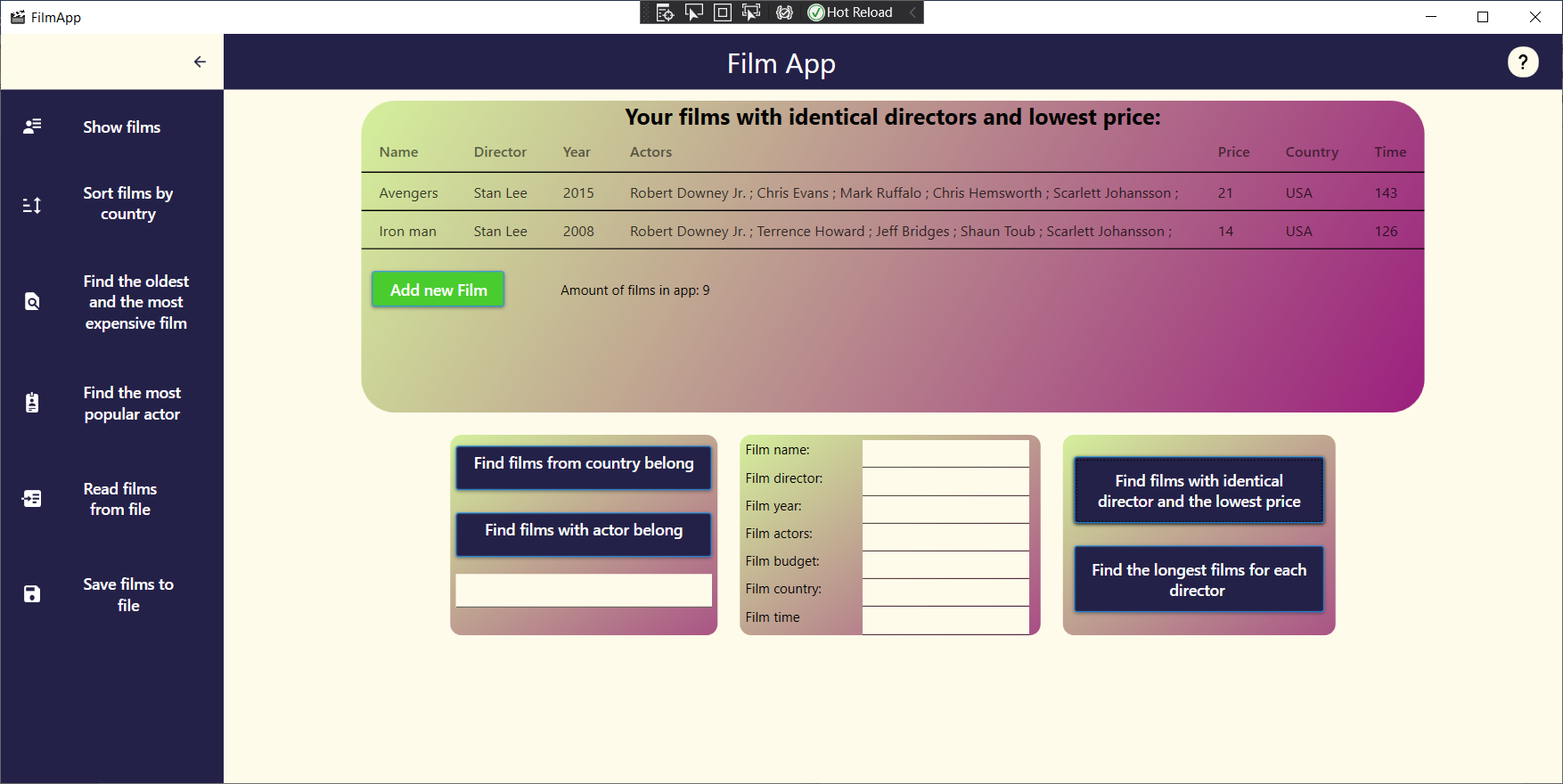


Рисунок 5.2 Назви фільмів, в яких однакові режисери та найменші бюджети одночасно.

* За заданою країною виробництва знаходить всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

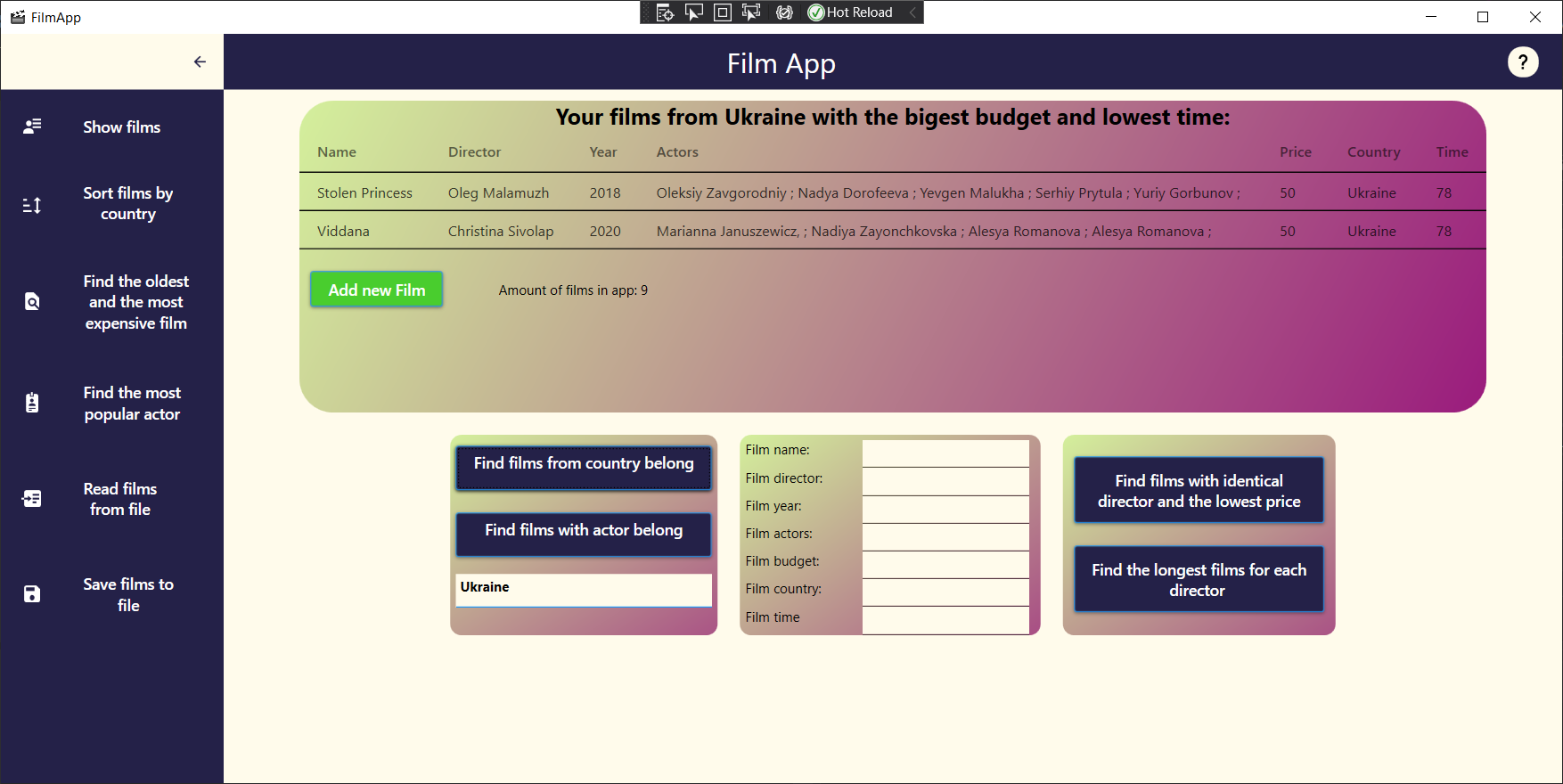


Рисунок 5.3 За заданою країною виробництва всі фільми, в яких найбільші бюджети і найменша тривалість одночасно.

* Для кожного Режисера визначає фільм з найбільшою тривалістю

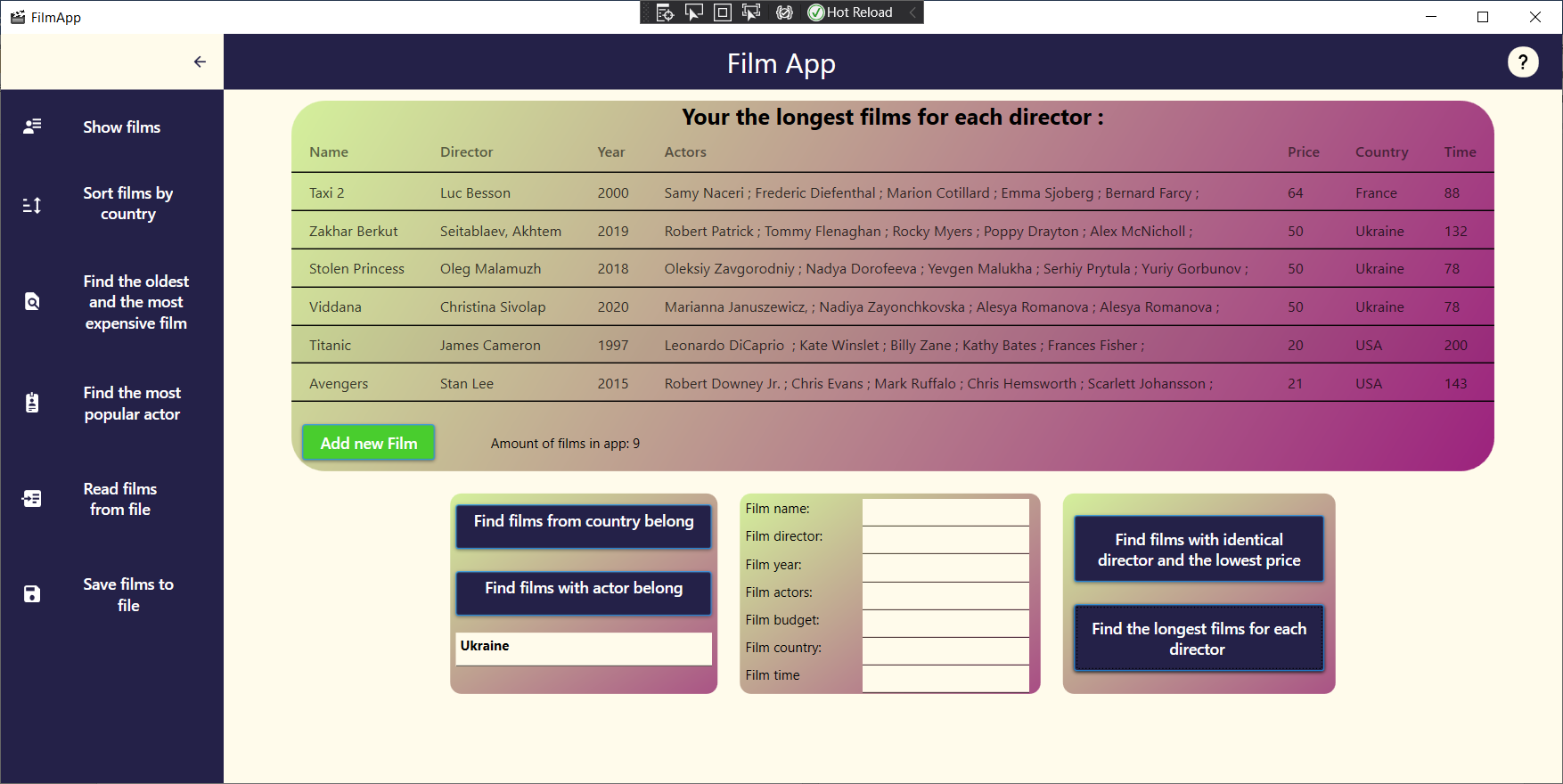


Рисунок 5.4 Для кожного Режисера фільм з найбільшою тривалістю.

* Знаходить найдорожчий та найстарший фільм одночасно.

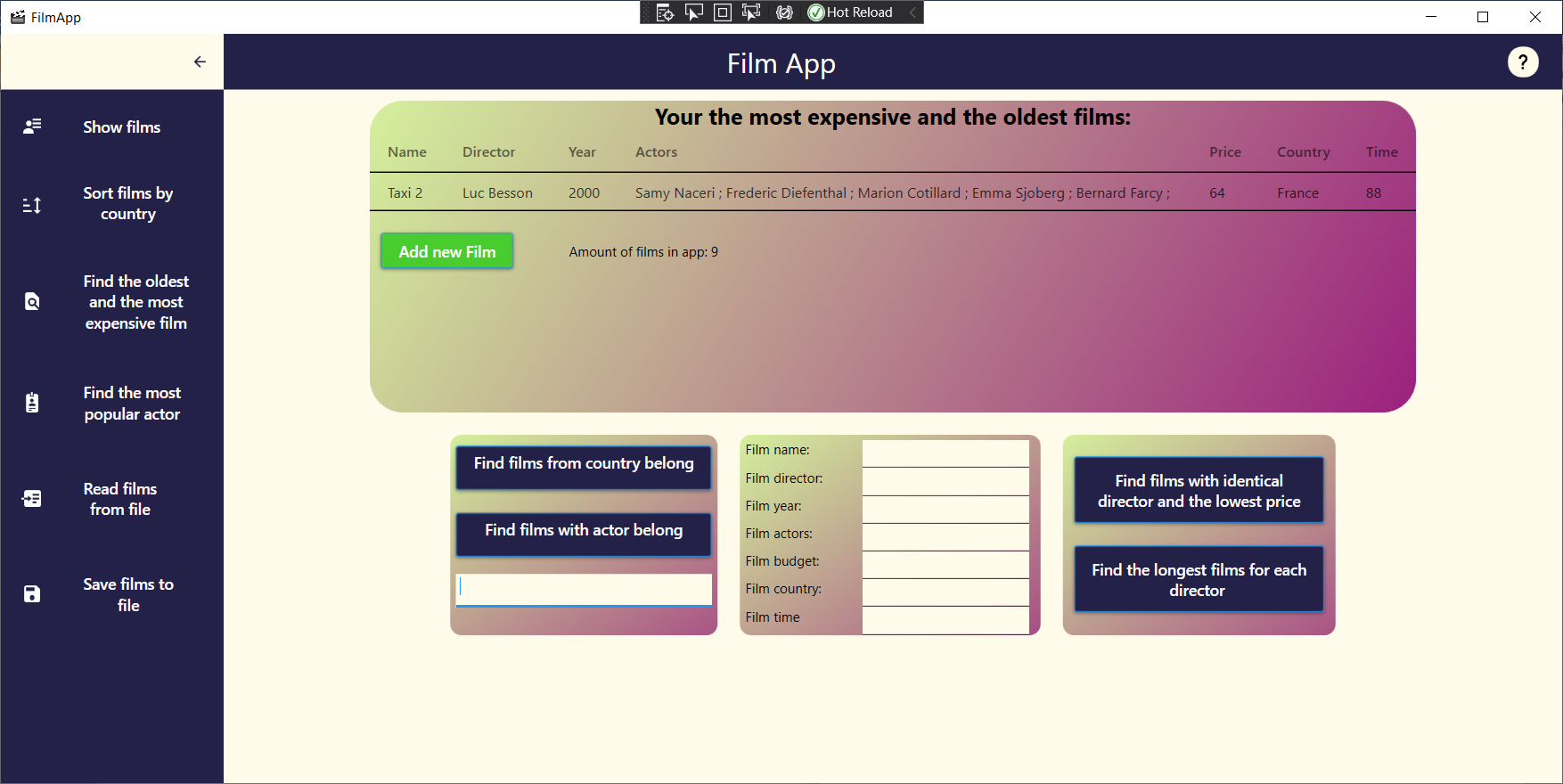


Рисунок 5.5 найдорожчий та найстарший фільм одночасно.

* За заданим актором визначає всі фільми, в яких він (вона) знімались.

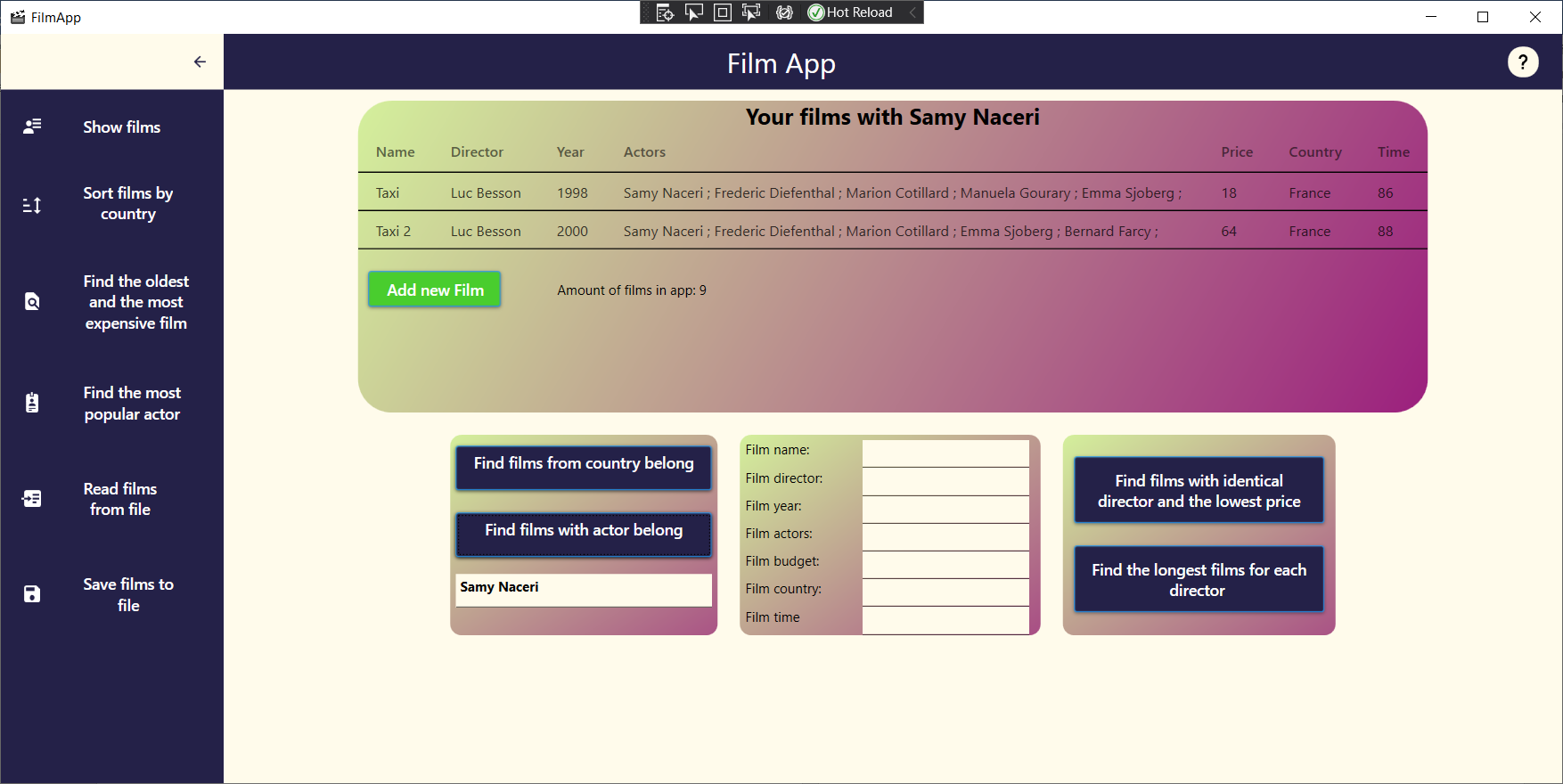


Рисунок 5.6 За заданим актором визначити всі фільми, в яких він (вона) знімались.

* Встановить найбільш популярного актора.

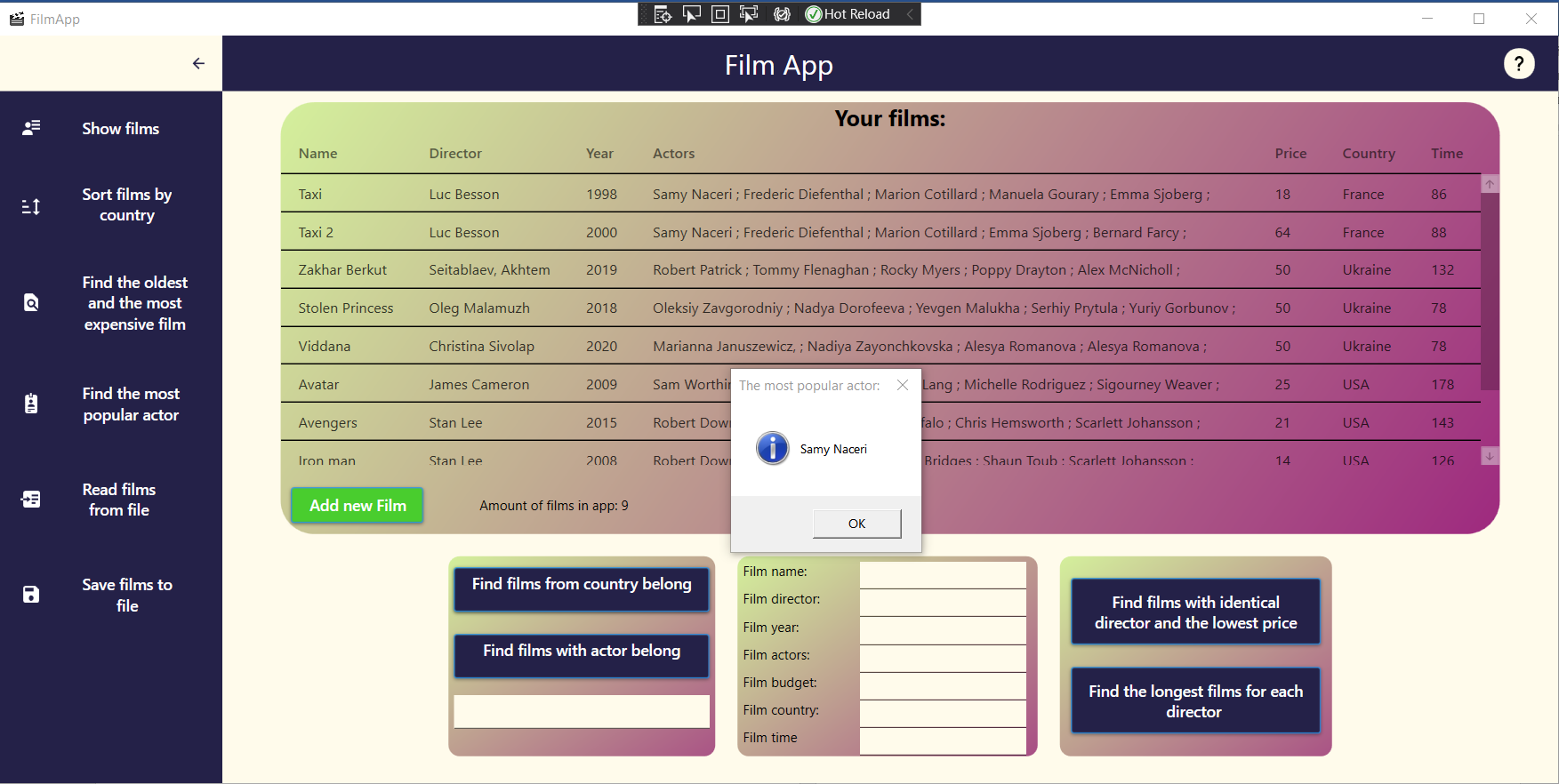


Рисунок 5.7 Встановити найбільш популярного актора.

# 7.  Виняткові ситуації.

У програмі передбачено такі виняткові ситуації:

* Читаємо дані з пустого файлу

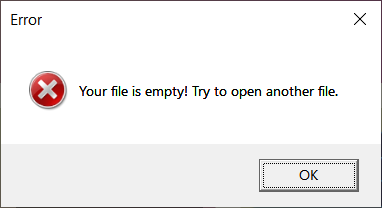


Рисунок 11 помилка, пустий файл

* Дані у файлі некоректного формату

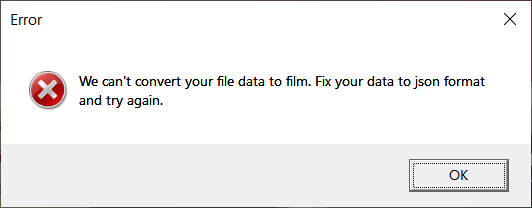


Рисунок 12 помилка, некоректні дані у файлі

* Помилка заповнення певного поля у файлі

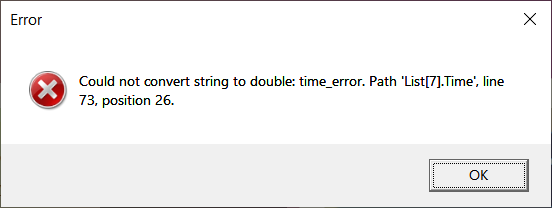


Рисунок 13 некоректно заповнений певний рядок у файлі

* Помилка при вводі даних з клавіатури

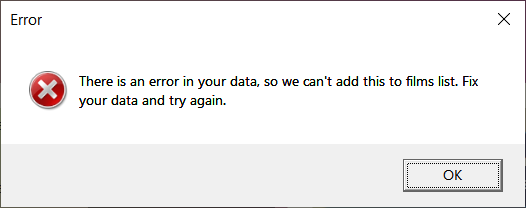


Рисунок 14 помилка при некоректних вхідних даних з клавіатури

* Помилка при виклику функцій над пустим списком

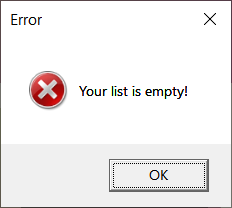


Рисунок 15 помилка при спробі виконати дії над пустим списком

* Помилка при пошуку фільмів з некоректно введеної країни

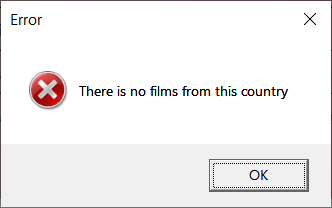


Рисунок 16 помилка при пошуку фільмів з некоректної країни

* Помилка при пошуку фільмів з некоректно введеного

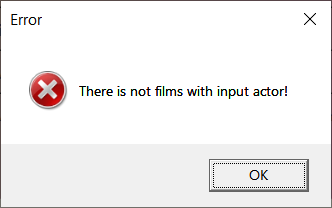


Рисунок 17 помилка при пошуку некоректного актора

# 8. Висновки

Під час виконання даної роботи, я розробив програму FilmApp, у візуальному середовищі розробити – Visual Studio 2019 з використанням технології WPF. Програма створена, щоб працювати з базою даних фільмів та їх параметрами( назва, режисер, рік випуску, список акторів, бюджет, країна розробки, час). Програма написано мовою C#.

При написанні коду, я дотримався стилю об’єктно орієнтованого програмування: для представлення моделей використав різні класи, та для взаємодії між ними використав їхні методи, дотримався принципу інкапсуляції. До написаних класів входять наступні класи: FilmData, ActorsList, FilmList, FilmsData.

# 9. Список використаної літератури

1. <http://vns.lp.edu.ua/> - віртуальне середовище навчання НУ «ЛП».
2. <https://docs.microsoft.com/ru-ru/dotnet/csharp/> - документація мови програмування C# від Microsoft.
3. <https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-5.0> – документація технології WPF від Microsoft.
4. <https://metanit.com/sharp/> - інструкція мови C# та технології .NET від сайту matanit.