Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas I (P175B118)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Studentas

Lekt. Kęstutis Simonavičius

Dėstytojas

Kaunas 2021

TURINYS

1. Duomenų klasė

1.1. Darbo užduotis

U1-9. IMDB.

Turite iš IMDB „ištrauktą“ filmų sąrašą. Duomenų faile pateikta informacija apie filmus: filmo pavadinimas, leidimo metai, žanras, kino studija, režisierius, 2 aktoriai, pajamos. • Raskite pelningiausią 2019 m. filmą, ekrane atspausdinkite šio filmo pavadinimą, režisierių, bei kiek filmas uždirbo. Jei yra keli, spausdinkite visus. • Raskite daugiausiai filmų pastačiusį režisierių, ekrane atspausdinkite jo pavardę. Jei yra keli, spausdinkite visus. • Sudarykite filmų, kuriuose vaidino N. Cage, sąrašą, į failą „Cage.csv“ įrašykite filmų pavadinimus, leidimo metus bei kino studijos pavadinimus.

1.2. Programos tekstas

AllMovieInfo.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// Saves Important Information For ALL IMDB Class Objects
    /// </summary>
    static class AllMovieInfo
    {
        private static Dictionary<string, int> DirectorPopularity = new Dictionary<string,
int>();

        /// <summary>
        /// Finds Directors With The Most Movies Directed. Returns List String Object.
        /// </summary>
        /// <returns></returns>
        public static List<string> FindBestDirectors()
        {
            List<string> directors = new List<string>();
            int filmsDirected = 0;

            foreach (string key in DirectorPopularity.Keys)
            {

                if (filmsDirected < DirectorPopularity[key])
                {
                    filmsDirected = DirectorPopularity[key];
                    directors.Clear();
                    directors.Add(key);
                }
                else if (filmsDirected == DirectorPopularity[key])
                {
                    directors.Add(key);
                }
            }
```

```
            return directors;
        }

        /// <summary>
        /// Adds a Movie Tally To The Director
        /// </summary>
        /// <param name="director"></param>
        public static void AddDirector(string director)
        {
            /// <summary>
            /// Records how many movies a director has directed.
            /// </summary>

            if (DirectorPopularity.ContainsKey(director) == false)
                DirectorPopularity.Add(director, 0);

            DirectorPopularity[director]++;
        }
    }
}
```

IMDB.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Lab01
{
    /// <summary>
    /// IMDB Movie Object
    /// </summary>
    class IMDB
    {
        public string       Name     { get; set; }
        public int          Date     { get; set; }
        public string       Genre    { get; set; }
        public string       Studio   { get; set; }
        public string       Director { get; set; }
        public List<string> Actors   { get; set; }
        public int          Revenue  { get; set; }

        public IMDB(string name,
                    int    date,
                    string genre,
                    string studio,
                    string director,
                    string actor1,
                    string actor2,
                    int    revenue)
        {
            Name     = name;
            Date     = date;
            Genre    = genre;
            Director = director;
            Revenue  = revenue;
            Studio   = studio;

            Actors = new List<string>();
            Actors.Add(actor1);
            Actors.Add(actor2);
```

```
            AllMovieInfo.AddDirector(Director);
        }
    }
}
```

InOutHelpers.cs:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// File Input Output Helper
    /// </summary>
    static class InOutHelpers
    {
        // Text formatting const
        private const int tSize = -20;

        /// <summary>
        /// REwrites Initial Data. Takes List<IMDB>, string outputPath. Returns void
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="outputPath"> output path to where to write the data</param>
        public static void WriteInitialData(List<IMDB> movies, string outputPath)
        {



            using (StreamWriter sw = new StreamWriter(outputPath))
            {
                sw.WriteLine($"{"Name",tSize}|" +
                             $"{"Date",tSize}|" +
                             $"{"Genre",tSize}|" +
                             $"{"Studio",tSize}|" +
                             $"{"Director",tSize}|" +
                             $"{"Actors",(tSize * 2) - 1}|" +
                             $"{"Revenue",-10}|");

                foreach (IMDB movie in movies)
                    sw.WriteLine($"{movie.Name,tSize}|" +
                                 $"{movie.Date,-tSize}|" +
                                 $"{movie.Genre,tSize}|" +
                                 $"{movie.Studio,tSize}|" +
                                 $"{movie.Director,tSize}|" +
                                 $"{movie.Actors[0],tSize}|" +
                                 $"{movie.Actors[1],tSize}|" +
                                 $"{movie.Revenue,10}|");
            }
        }

        /// <summary>
        /// Writes Data to Output File
        /// </summary>
        /// <param name="movies">List IMDB Object</param>
        /// <param name="ouputPath">Output File Path</param>
        public static void PrintMoviesToCSV(this List<IMDB> movies, string ouputPath)
        {
            using (StreamWriter sw = new StreamWriter(ouputPath))
            {
```

```
                sw.WriteLine($"{"Name",tSize};{"Date",tSize};{"Studio",tSize}");
                foreach (IMDB movie in movies)
                    sw.WriteLine($"{movie.Name};{movie.Date};{movie.Studio}");
            }

        }


        /// <summary>
        /// Reads Data, returns List IMDB Object
        /// </summary>
        /// <param name="filePath">Input File Object</param>
        /// <returns></returns>
        public static List<IMDB> ReadData(string filePath)
        {

            List<IMDB> output = new List<IMDB>();

            using (StreamReader sr = new StreamReader(filePath))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    string[] data = line.Split(';');
                    IMDB imdb = new IMDB(data[0],
                                         int.Parse(data[1]),
                                         data[2],
                                         data[3],
                                         data[4],
                                         data[5],
                                         data[6],
                                         int.Parse(data[7]));

                    output.Add(imdb);
                }
            }

            return output;
        }
    }
}
```

TaskUtils.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab01
{
    /// <summary>
    /// Task Utilities For Console And Extension Methods For Filtering
    /// </summary>
    static class TaskUtils
    {
        /// <summary>
        /// Prints Most Profitable Movies
        /// </summary>
        /// <param name="movies">List IMDB Object</param>
        public static void PrintMostProfitable(this List<IMDB> movies)
        {
            if (movies.Count == 0)
                Console.WriteLine("No Movies Found");
            else
```

```
            {
                Console.WriteLine($"{"Name",-20}|{"Director",-20}|{"Revenue",-20}");
                foreach (IMDB movie in movies)
                {
                    Console.WriteLine($"{movie.Name,-20}|{movie.Director,-
20}|{movie.Revenue,6}");
                }
            }
        }

        /// <summary>
        /// Prints Directors in "PrintBestDirectors" format
        /// </summary>
        /// <param name="directors">List IMDB Object</param>
        public static void PrintBestDirectors(this List<string> directors)
        {
            Console.WriteLine("Best Directors: ");
            if (directors.Count == 0)
                Console.WriteLine("No Directors Found");
            else
                foreach (string director in directors)
                    Console.WriteLine(director);
        }

        /// <summary>
        /// Finds Movies With Given Actor (string)
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="actor">Actor Name to Be Searched</param>
        /// <returns></returns>
        public static List<IMDB> FindMoviesWith(this List<IMDB> movies, string actor)
        {
            List<IMDB> output = new List<IMDB>();

            foreach (IMDB movie in movies)
                if (movie.Actors.Contains(actor))
                    output.Add(movie);

            return output;
        }

        /// <summary>
        /// Finds Most Profitable Movies in a given year (int)
        /// </summary>
        /// <param name="movies">List IMDB object</param>
        /// <param name="year">int year when the movie was released</param>
        /// <returns></returns>
        public static List<IMDB> FindMostProfitable(this List<IMDB> movies, int year)
        {

            List<IMDB> output = new List<IMDB>();
            int profitability = 0;

            Console.WriteLine($"Most Profitable Movies in Year: {year}");
            foreach (IMDB movie in movies)
            {
                if (movie.Date == year)
                {
                    if (profitability < movie.Revenue)
                    {
                        profitability = movie.Revenue;
                        output.Clear();
                        output.Add(movie);
                    }
                    else if (profitability == movie.Revenue)
                    {
```

```
                        output.Add(movie);
                    }

                }
            }

            return output;
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;

namespace Lab01
{
    class Program
    {
        const string CDd = @"imdb2.txt";
        const string CDinitial = @"imdbInitial.txt";
        const string CDcsv = @"MoviesWith.csv";
        static void Main(string[] args)
        {
            List<IMDB> imdb = InOutHelpers.ReadData(CDd);
            InOutHelpers.WriteInitialData(imdb, CDinitial);
            imdb.FindMostProfitable(2019).PrintMostProfitable();
            Console.WriteLine(new string('-', 74));
            AllMovieInfo.FindBestDirectors().PrintBestDirectors();
            imdb.FindMoviesWith("N. Cage").PrintMoviesToCSV(CDcsv);
            Console.ReadLine();
        }
    }
}
```

1.3. Pradiniai duomenys ir rezultatai

Pirmas testinis variantas

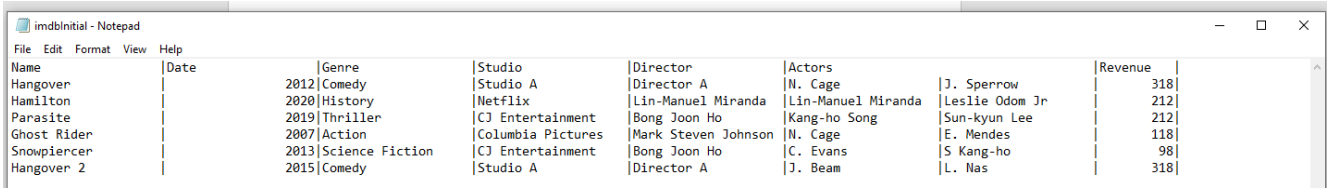
Pradiniai duomenys:

imdb.txt

```
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom Jr;212
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;118
Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho;98
Hangover 2;2015;Comedy;Studio A;Director A;J. Beam;L. Nas;318
```

Rezultatai

imdbinitial.txt:



Console:

```
C:\Users\norsta\Desktop\KTU-OOP-Semester1-main\Lab01\Lab01\bin\Debug\net461\Lab01.exe
Most Profitable Movies in Year: 2019
Name                |Director            |Revenue
Parasite            |Bong Joon Ho        |    212
--------------------------------------------------------------------------------
Best Directors:
Director A
Bong Joon Ho
```

MoviesWith.csv:

	A	B	C
1	Name	Date	Studio
2	Hangover	2012	Studio A
3	Ghost Rider	2007	Columbia Pictures
4			

Antras testinis variantas

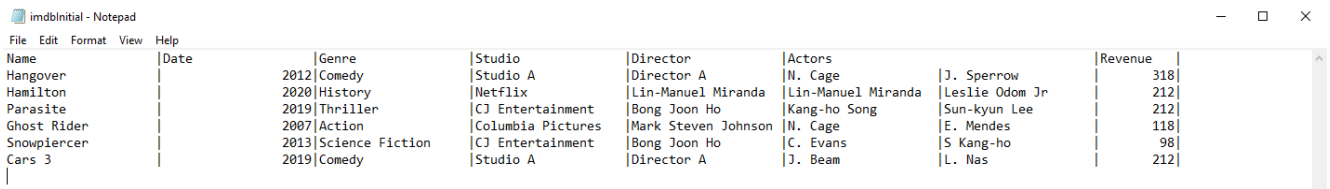
Pradiniai duomenys:

Imdb2.txt

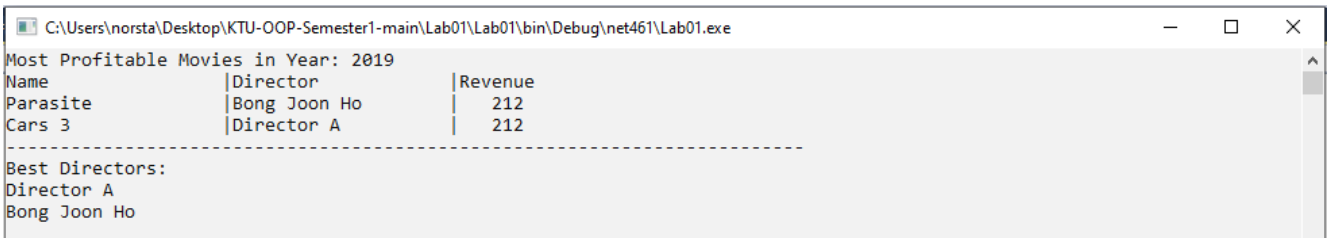
```
Hangover;2012;Comedy;Studio A;Director A;N. Cage;J. Sperrow;318
Hamilton;2020;History;Netflix;Lin-Manuel Miranda;Lin-Manuel Miranda;Leslie Odom Jr;212
Parasite;2019;Thriller;CJ Entertainment;Bong Joon Ho;Kang-ho Song;Sun-kyun Lee;212
Ghost Rider;2007;Action;Columbia Pictures;Mark Steven Johnson;N. Cage;E. Mendes;118
Snowpiercer;2013;Science Fiction;CJ Entertainment;Bong Joon Ho;C. Evans;S Kang-ho;98
Cars 3;2019;Comedy;Studio A;Director A;J. Beam;L. Nas;212
```

Rezultatai:

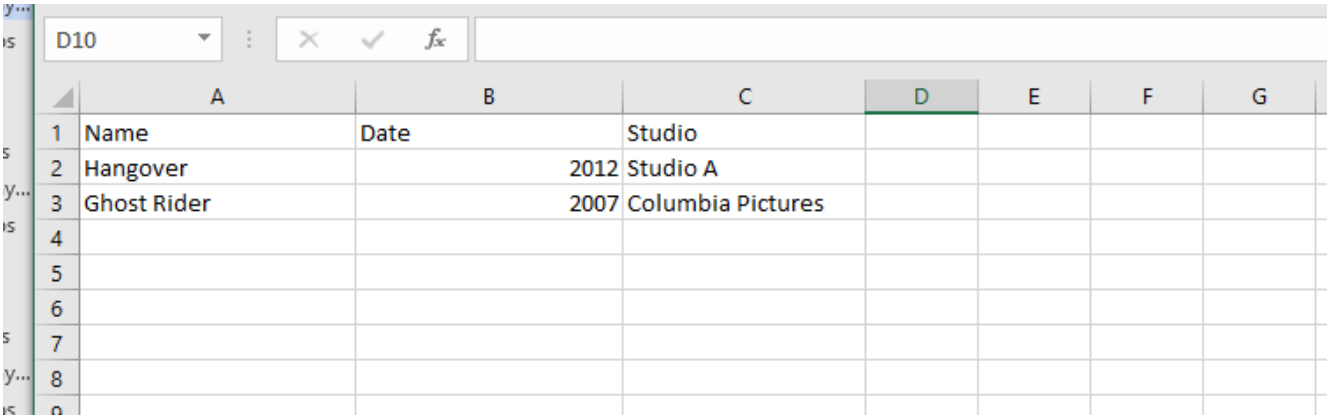
ImdbInitial.txt:



Console:



MoviesWith.csv:



1.4. Dėstytojo pastabos

2. Skaičiavimų klasė

2.1. Darbo užduotis

2.2. Programos tekstas

2.3. Pradiniai duomenys ir rezultatai

2.4. Dėstytojo pastabos

3. Konteineris

3.1. Darbo užduotis

3.2. Programos tekstas

3.3. Pradiniai duomenys ir rezultatai

3.4. Dėstytojo pastabos

4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

4.2. Programos tekstas

4.3. Pradiniai duomenys ir rezultatai

4.4. Dėstytojo pastabos

5. Paveldėjimas

5.1. Darbo užduotis

5.2. Programos tekstas

5.3. Pradiniai duomenys ir rezultatai

5.4. Dėstytojo pastabos