МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ

ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)

Кафедра

«Автоматизированные системы управления»

Курсовая работа по дисциплине

«Объектно-ориентированное программирование»

Вариант № 21

Тема:

«Разработка информационной системы «Хлебозавод»

Выполнил: студент группы \_\_\_\_\_\_

Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Москва 2019

# Задание

Требуется разработать программную систему, предназначенную для организаторов соревнований по футболу в рамках первенства страны. Такая система должна обеспечивать хранение сведений о командах, участвующих в первенстве, об игроках команд, о расписании встреч и их результатах, о цене билетов на игры.

Сведения о команде: название команды, город, где она базируется, имя главного тренера, место в таблице прошлого сезона, расписание встреч. В один день команда может участвовать только в одной встрече.

Сведения об игроке включают в себя фамилию и имя игрока, его возраст, номер и амплуа в команде. Сведения о стадионе, на котором происходит встреча, содержат город, в котором он находится, название стадиона, и его вместимость.

Должны быть созданы обобщенные списки:

* Сведения об игроках.
* Сведения о команде.
* Сведения о стадионе.
* Сведения о матчах.

Должны быть реализованы следующие запросы:

* Даты встреч указанной команды, ее противники и счет.
* Номера и фамилии игроков команд, участвовавших во встрече, которая проходила в указанный день на стадионе.
* Цена билета на матч между указанными командами.
* Команды, имеющие наилучшую и наихудшую разницу забитых и пропущенных мячей.
* Команды, занявшие призовые места.
* Расписание игр по стадионам.

# Введение

Цель курсовой работы: освоение методов обработки информации в приложении с помощью LINQ запросов.

Задача: написать программу для работы с обобщенными локальными списками, позволяющую организовать взаимодействие с структурированными данными, находящимися в текстовых файлах, выделять из них необходимую информацию, группировать и сортировать коллекции, а также объединять несколько коллекций для получения наиболее полной информации.

Программа предназначена для контроля информационной системы «Соревнования по футболу»: получения сведений о командах, их составе и месте в рейтинге, а также для решения круга специализированных задач:

– сортировка и фильтрация данных по различным критериям;

– получение информации о матчах, которые проводятся в данный момент;

– отследить цены на матч между двумя любыми командами на любом из доступных стадионов.

# Структура программы

## Классы программы

### Класс Team

Содержит данные о футбольных командах.

|  |  |
| --- | --- |
| Объявление поля класса | Описание |
| public uint Key{ get; set; } | Идентификатор команды (первичный ключ) |
| public string Name{ get; set; } | Название команды |
| public string City{ get; set; } | Город |
| public string TrName{ get; set; } | Фамилия тренера |
| public uint Rat{ get; set; } | Место в рейтинге |

### Класс Player

Содержит данные об игроках футбольных команд.

|  |  |
| --- | --- |
| Объявление поля класса | Описание |
| public uint Key { get; set; } | Идентификатор игрока (первичный ключ) |
| public uint TeamKey { get; set; } | Идентификатор команды |
| public string Name { get; set; } | Фамилия |
| public uint Age { get; set; } | Возраст |
| public string Role { get; set; } | Амплуа |
| public uint Number { get; set; } | Номер |

### Класс Stadium

Содержит данные о стадионах, на которых проводятся матчи.

|  |  |
| --- | --- |
| Объявление поля класса | Описание |
| public uint Key { get; set; } | Идентификатор стадиона (первичный ключ) |
| public string Name { get; set; } | Название стадиона |
| public string City { get; set; } | Город |
| public uint Capacity { get; set; } | Вместимость |

### Класс Match

Содержит данные о проводимых матчах.

|  |  |
| --- | --- |
| Объявление поля класса | Описание |
| public uint Key { get; set; } | Идентификатор матча (первичный ключ) |
| public uint StadKey { get; set; } | Идентификатор стадиона |
| public uint FirstTeamKey { get; set; } | Идентификатор первой команды |
| public uint SecTeamKey{ get; set; } | Идентификатор второй команды |
| public DateTime Date { get; set; } | Дата проведения матча |
| public uint FirstScore { get; set; } | Счет первой команды |
| public uint SecScore { get; set; } | Счет второй команды |
| public double Price{ get; set; } | Цена билета на матч |

## Коллекции программы

* + 1. static List<Team> lstTeams: содержит список экземпляров класса Team;
    2. static List<Player> lstPlayers: содержит список экземпляров класса Player;
    3. static List<Stadium> lstStadiums: содержит список экземпляров класса Stadium;
    4. static List<Match> lstMatches: содержит список экземпляров класса Match.

# Входные данные

Для инициализации списков используются текстовые файлы со следующей структурой:

– файл, содержащий сведения о футбольных командах (Таблица 1).

Таблица 1 – Структура файла со сведениями о футбольных командах

|  |  |  |  |
| --- | --- | --- | --- |
| № параметра при чтении | Название параметра | Обозначение | Тип |
| 1 | Название команды | Name | string |
| 2 | Город | City | string |
| 3 | Фамилия тренера | TrName | string |
| 4 | Место в рейтинге | Rat | uint |

– файл, содержащий сведения об игроках (Таблица 2).

Таблица 2 – Структура файла со сведениями об игроках

|  |  |  |  |
| --- | --- | --- | --- |
| № параметра при чтении | Название параметра | Обозначение | Тип |
| 1 | Идентификатор команды | TeamName | uint |
| 2 | Фамилия игрока | Name | string |
| 3 | Возраст | Age | uint |
| 4 | Амплуа | Role | string |
| 5 | Номер | Number | uint |

– файл, содержащий сведения о стадионах (Таблица 3).

Таблица 3 – Структура файла со сведениями о стадионах

|  |  |  |  |
| --- | --- | --- | --- |
| № параметра при чтении | Название параметра | Обозначение | Тип |
| 1 | Название стадиона | Name | string |
| 2 | Город | City | string |
| 3 | Вместимость | Capacity | uint |

– файл, содержащий сведения о футбольных матчах (Таблица 4).

Таблица 1 – Структура файла со сведениями о футбольных матчах

|  |  |  |  |
| --- | --- | --- | --- |
| № параметра при чтении | Название параметра | Обозначение | Тип |
| 1 | Идентификатор стадиона | StadKey | uint |
| 2 | Идентификатор первой команды | FirstTeamKey | uint |
| 3 | Идентификатор второй команды | SecTeamKey | uint |
| 4 | Дата проведения | Date | DateTime |
| 5 | Счет | Score | uint |
| 6 | Цена билета | Price | double |

# Код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace AppFootballCompetitions

{

class Program

{

const string EXIT\_TIP = "\nДля возврата в меню нажмите любую клавишу...";

const string TEAM\_PATH = "Teams.txt";

const string PLAYER\_PATH = "Players.txt";

const string STADIUM\_PATH = "Stadiums.txt";

const string MATCH\_PATH = "Matches.txt";

static List<Team> lstTeams = new List<Team>();

static List<Player> lstPlayers = new List<Player>();

static List<Stadium> lstStadiums = new List<Stadium>();

static List<Match> lstMatches = new List<Match>();

static void Main(string[] args)

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Title = "Справочная система футбольных соревнований";

FileReader(TEAM\_PATH, 1, 4);

FileReader(PLAYER\_PATH, 2, 5);

FileReader(STADIUM\_PATH, 3, 3);

FileReader(MATCH\_PATH, 4, 6);

Console.Clear();

//Console.ReadKey();

char ch;

do

{

ch = MainMenu();

switch (ch)

{

case '1':

do

{

Console.Clear();

Console.WriteLine("Cписок встреч указанной команды, ее противников, счет:");

Console.Write("\nВведите название команды: ");

if (MatchesListQry(Console.ReadLine().ToLower()))

break;

Console.Write("\nВведенное название некорректно или не найдено! Повторите ввод...");

System.Threading.Thread.Sleep(1500);

} while (true);

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '2':

do

{

Console.Clear();

Console.WriteLine("Cписок игроков команд, участвовавших в встрече в указанный день на указанном стадионе:");

Console.WriteLine("\nВведите дату и название стадиона через запятую: ");

if (PlayersListQry(Console.ReadLine().ToLower()))

break;

Console.Write("\nВведенные данные некорректны или не найдены! Повторите ввод...");

System.Threading.Thread.Sleep(1500);

} while (true);

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '3':

do

{

Console.Clear();

Console.WriteLine("Цена на билет на матч между указанными командами:");

Console.WriteLine("\nВведите две команды через запятую: ");

if (PriceQry(Console.ReadLine().ToLower()))

break;

Console.Write("\nВведенные данные некорректны или не найдены! Повторите ввод...");

System.Threading.Thread.Sleep(1500);

} while (true);

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '4':

Console.Clear();

Console.WriteLine("Команды, имеющие наилучшую и наихудшую разницу забитых и пропущенных голов:");

BestWorstMatches();

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '5':

Console.Clear();

Console.WriteLine("Cписок команд, занявших призовые места:");

ChampList();

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '6':

Console.Clear();

Console.WriteLine("Расписание игр по стадионам:");

StadiumMatchesList();

Console.Write(EXIT\_TIP);

Console.ReadKey();

break;

case '7':

//do

//{

// Console.Clear();

// Console.WriteLine("Список фильмов, снятых указанным режиссером:");

// Console.WriteLine("\nВведите имя и фамилию режиссера: ");

// if (DirFilmlistQry(Console.ReadLine().ToLower()))

// break;

// Console.Write("\nВведенные имя и фамилия некорректны или не найдены! Повторите ввод...");

//} while (true);

//Console.Write(EXIT\_TIP);

//Console.ReadKey();

//break;

case '9':

break;

default:

Console.WriteLine("Ошибка ввода! Введите цифру от 1 до 9");

System.Threading.Thread.Sleep(1500);

break;

}

} while (ch != '9');

}

static char MainMenu()

{

Console.Clear();

Console.WriteLine("1 - Вывести список встреч указанной команды, ее противников, счет");

Console.WriteLine("2 - Вывести список игроков команд, участвовавших в встрече в указанный день на указанном стадионе");

Console.WriteLine("3 - Вывести цену на билет на матч между указанными командами");

Console.WriteLine("4 - Вывести команды, имеющие наилучшую и наихудшую разницу забитых и пропущенных голов");

Console.WriteLine("5 - Вывести список команд, занявших призовые места");

Console.WriteLine("6 - Вывести расписание игр по стадионам");

Console.WriteLine("7 - Редактировать список игроков");

Console.WriteLine("8 - Редактировать список матчей");

Console.WriteLine("9 - Выход");

return Console.ReadKey(true).KeyChar;

}

static bool MatchesListQry(string team\_name)

{

uint t\_key = lstTeams.Find(f => f.Name.ToLower() == team\_name).Key;

//uint i =( (lstMatches[1].FirstTeamKey == t\_key) ? lstMatches[1].FirstTeamKey : (lstMatches[1].SecTeamKey == t\_key) ? lstMatches[1].SecTeamKey : 0);

var qr1 = lstMatches.Where(w => w.FirstTeamKey == t\_key || w.SecTeamKey == t\_key);

if (qr1.Count() == 0)

{

return false;

}

Console.Write("\n{0}:", lstTeams.Find(f => f.Key == t\_key).Name);

foreach (var v in qr1)

{

Console.Write("\n\tПротивник: {0, -15} {1} {2}:{3}",

lstTeams.Find(f => f.Key == v.SecTeamKey).Name, v.Date.ToString("d"), v.FirstScore, v.SecScore);

}

return true;

}

static bool PlayersListQry(string data)

{

string[] str = data.Split(',');

DateTime dt;

if (!DateTime.TryParse(str[0], out dt))

return false;

var qr = lstMatches.Where(w => w.Date == dt).Where(w1 => (w1.StadKey ==

(lstStadiums.Find(f => f.Name.ToLower() == str[1].Trim()).Key))).Join(lstTeams, m => m.FirstTeamKey,

t => t.Key, (m, t) => new { t.Name, t.Key }).GroupJoin(lstPlayers, q=>q.Key, p=>p.TeamKey,

(l, k)=> new { name = l.Name, lst = k});

if (qr.Count() == 0)

{

return false;

}

foreach (var v in qr)

{

Console.Write("\nКоманда: {0, -15}", v.name);

foreach (var w in v.lst)

{

Console.Write("\n\t {0, -15}", w.Name);

}

Console.Write("\n");

}

return true;

}

static bool PriceQry(string data)

{

string[] str = data.Split(',');

uint t\_key1 = lstTeams.Find(f => f.Name.ToLower() == str[0].Trim()).Key;

uint t\_key2 = lstTeams.Find(f => f.Name.ToLower() == str[1].Trim()).Key;

var qr = lstMatches.Where(w => w.FirstTeamKey == t\_key1 ).Where(w=> w.SecTeamKey == t\_key2);//.Join(lstTeams, m => m.FirstTeamKey,

//t => t.Key, (m, t) => new { t.Name, t.Key }).GroupJoin(lstPlayers, q => q.Key, p => p.TeamKey,

// (l, k) => new { name = l.Name, lst = k });

if (qr.Count() == 0)

{

return false;

}

foreach (var v in qr)

{

Console.Write("\nДата: {0} Цена: {1}", v.Date, v.Price);

}

return true;

}

static void BestWorstMatches()

{

int worst = 0;

int best = 0;

Match bestMatch = null;

Match worstMatch = null;

foreach (Match m in lstMatches)

{

int range = Convert.ToInt32((int)m.FirstScore -(int) m.SecScore);

if (range > 0 && range > best)

{

best = range;

bestMatch = m;

}

else if (range <= 0 && range < worst)

{

worst = range;

worstMatch = m;

}

}

Console.Write("\nКоманда с наилучшей разницей голов: {0}",

lstTeams.Find(f=>f.Key== bestMatch.FirstTeamKey).Name);

Console.Write("\nКоманда с наихудшей разницей голов: {0}",

lstTeams.Find(f => f.Key == worstMatch.FirstTeamKey).Name);

Console.WriteLine();

}

static void ChampList()

{

lstTeams.OrderBy(o => o.Rat);

foreach (Team t in lstTeams)

{

if(t.Rat<=3)

Console.Write("\n{0}-e место: {1}", t.Rat, t.Name);

}

}

static void StadiumMatchesList()

{

var qr = lstMatches.Join(lstStadiums, m => m.StadKey, s => s.Key, (m, s) => new

{

m.FirstTeamKey,

m.SecTeamKey,

m.FirstScore,

m.SecScore,

m.Date,

m.Price,

m.StadKey,

s.Name

}).OrderBy(o => o.Name).GroupBy(g=>g.StadKey);

foreach (var v in qr)

{

Console.Write("\n{0}:", lstStadiums.Find(f=>f.Key== v.Key).Name);

foreach (var w in v)

{

Console.Write("\n\t{0}", lstTeams.Find(f => f.Key == w.FirstTeamKey).Name);

Console.Write(" {0}", lstTeams.Find(f => f.Key == w.SecTeamKey).Name);

Console.Write(" {0} {1}:{2} {3}", w.Date.ToString("d"), w.FirstScore, w.SecScore, w.Price);

}

}

}

static void FileReader(string path, int lst\_ident, int file\_field\_count)

{

StreamReader fIn = new StreamReader(path, Encoding.Default);

if (lst\_ident == 1)

{

do

{

string[] str = new string[file\_field\_count];

string[] strline = fIn.ReadLine().Split(';');

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

{

if (strline[i].Trim() == "")

continue;

else

str[i] = strline[i].Trim();

}

lstTeams.Add(new Team(Convert.ToUInt32(lstTeams.Count + 1), Convert.ToString(str[0]),

Convert.ToString(str[1]), Convert.ToString(str[2]), Convert.ToUInt32(str[3])));

} while (!fIn.EndOfStream);

}

else if (lst\_ident == 2)

{

do

{

string[] str = new string[file\_field\_count];

string[] strline = fIn.ReadLine().Split(';');

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

{

if (strline[i].Trim() == "")

continue;

else

str[i] = strline[i].Trim();

}

/\*if (str[4].Trim() == "")

str[4] = "-";

if (str[5].Trim() == "")

str[5] = "0";\*/

lstPlayers.Add(new Player(Convert.ToUInt32(lstPlayers.Count + 1),

Convert.ToUInt32(str[0]), Convert.ToString(str[1]), Convert.ToUInt32(str[2]),

Convert.ToString(str[3]), Convert.ToUInt32(str[4])));

} while (!fIn.EndOfStream);

}

else if (lst\_ident == 3)

{

do

{

string[] str = new string[file\_field\_count];

string[] strline = fIn.ReadLine().Split(';');

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

{

if (strline[i].Trim() == "")

continue;

else

str[i] = strline[i].Trim();

}

lstStadiums.Add(new Stadium(Convert.ToUInt32(lstStadiums.Count + 1), Convert.ToString(str[0]),

Convert.ToString(str[1]), Convert.ToUInt32(str[2])));

} while (!fIn.EndOfStream);

}

else

{

do

{

uint f\_team = 0;

uint s\_team = 0;

uint f\_score = 0;

uint s\_score = 0;

string[] str = new string[file\_field\_count];

string[] strline = fIn.ReadLine().Split(';');

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

{

if (strline[i].Trim() == "")

continue;

else

{

str[i] = strline[i].Trim();

if(i == 1)

f\_team = lstTeams.Where(w => w.Name == str[i]).FirstOrDefault().Key;

else if (i == 2)

s\_team = lstTeams.Where(w => w.Name == str[i]).FirstOrDefault().Key;

else if (i == 4)

{

string[] strscore = str[i].Split(':');

f\_score = Convert.ToUInt32(strscore[0]);

s\_score = Convert.ToUInt32(strscore[1]);

}

}

}

lstMatches.Add(new Match(Convert.ToUInt32(lstMatches.Count + 1), Convert.ToUInt32(str[0]),

f\_team, s\_team, Convert.ToDateTime(str[3]), f\_score, s\_score,

Convert.ToDouble(str[5])));

} while (!fIn.EndOfStream);

}

}

}

}