|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
| Практическое задание № 1 | | | |
| по дисциплине «Разработка объектно-ориентированных программ с использованием C#/C++» | | | |
| **Интерфейсы** | | | |
|  | | | |
|  | Бригада №10 | ПМ-92 Богданов антон |
|  | ПМ-92 старцев максим |
| Преподаватель | Ступаков Илья Михайлович |
| Дата | 15.12.2020 |
|  |  |
|  |  |
|  | | | |
| Новосибирск | | | |

**Вариант №10**

В качестве обрабатываемых данных на вход программе подаются файлы students.txt и teachers.txt

### students.txt

Геннадий Александрович Горин 4 ютуб20 5 11/12/1972

Дио Иванович Брандо 2 ДжоДжо1 4,2 12/09/1867

Гарри Джеймс Поттер 1 Гриффиндор 2,9 31/07/1980

### teachers.txt

Андрей Петрович Петров ютуб20 Декан 10/05/2015 01/09/1996

Стас Станиславович Стасов нет Ректор 01/01/2020 01/01/2019

**Код программы**

using System;

using System.IO;

using System.Linq;

using System.Collections.Generic;

using System.Globalization;

namespace pz2

{

class Program

{

const string CONSOLE\_INPUT = "> ";

const string CONSOLE\_OUTPUT = ":: ";

static void Main(string[] args)

{

University university = new University();

foreach (var teacher in File.ReadAllLines("teachers.txt")

.Select(teacherString => Teacher.Parse(teacherString)))

university.Add(teacher);

foreach (var student in File.ReadAllLines("students.txt")

.Select(studentString => Student.Parse(studentString)))

university.Add(student);

Console.WriteLine("Введите help, чтобы увидеть команды");

bool isQuit = false;

string[] input;

while (!isQuit)

{

input = ConsoleInput().Split();

if (input.Length == 1 && input[0] == String.Empty)

continue;

switch (input[0])

{

case "help":

ConsoleOutput("add {student|teacher} [params] - добавить кого-нибудь");

ConsoleOutput("remove {student|teacher} [params] - удалить кого-нибудь");

ConsoleOutput("get {students|teachers|persons} [params] - вывести всех студентов/преподавателей/персон");

ConsoleOutput("find lastname [фамилия] - найти персону по фамилии");

ConsoleOutput("find department [название\_факультета] - найти преподавателя по факультету");

ConsoleOutput("quit - выйти");

break;

case "add":

if (input.Length > 1)

switch (input[1])

{

case "student":

try

{

university.Add(Student.Parse(String.Join(" ", input.Skip(2))));

ConsoleOutput("Студент успешно добавлен в университет");

}

catch

{

ConsoleOutput("Неверный формат ввода студента. Требуемый формат:");

ConsoleOutput("Имя Отчество Фамилия Курс Группа Средний\_Балл Дата\_Рождения");

}

break;

case "teacher":

try

{

university.Add(Teacher.Parse(String.Join(" ", input.Skip(2))));

ConsoleOutput("Преподаватель успешно добавлен в университет");

}

catch

{

ConsoleOutput("Неверный формат ввода преподавателя. Требуемый формат:");

ConsoleOutput("Имя Отчество Фамилия Факультет Должность Дата\_Начала\_Деятельности Дата\_Рождения");

}

break;

default:

ConsoleOutput("add {student|teacher|person} [params] - добавить кого-нибудь");

break;

}

else ConsoleOutput("add {student|teacher} [params] - добавить кого-нибудь");

break;

case "remove":

if (input.Length > 1)

switch (input[1])

{

case "student":

try

{

bool isRemoved = false;

var studentToRemove = Student.Parse(String.Join(" ", input.Skip(2)));

foreach (var student in university.Students)

if (studentToRemove.Equals(student))

{

isRemoved = true;

university.Remove(student);

break;

}

if (isRemoved)

ConsoleOutput("Студент успешно отчислен из университета");

else

ConsoleOutput("Такого студента в университете не существует");

}

catch

{

ConsoleOutput("Неверный формат ввода студента. Требуемый формат:");

ConsoleOutput("Имя Отчество Фамилия Курс Группа Средний\_Балл Дата\_Рождения");

}

break;

case "teacher":

try

{

bool isRemoved = false;

var teacherToRemove = Teacher.Parse(String.Join(" ", input.Skip(2)));

foreach (var teacher in university.Teachers)

if (teacherToRemove.Equals(teacher))

{

isRemoved = true;

university.Remove(teacher);

break;

}

if (isRemoved)

ConsoleOutput("Преподаватель успешно уволен из университета");

else

ConsoleOutput("Такого преподавателя в университете не существует");

}

catch

{

ConsoleOutput("Неверный формат ввода преподавателя. Требуемый формат:");

ConsoleOutput("Имя Отчество Фамилия Факультет Должность Дата\_Начала\_Деятельности Дата\_Рождения");

}

break;

default:

ConsoleOutput("remove {student|teacher} [params] - удалить кого-нибудь");

break;

}

else ConsoleOutput("remove {student|teacher} [params] - удалить кого-нибудь");

break;

case "get":

if (input.Length > 1)

switch (input[1])

{

case "persons":

ConsoleOutput("Все персоны:");

foreach (var person in university.Persons)

ConsoleOutput(person.ToString());

break;

case "students":

ConsoleOutput("Все студенты:");

foreach (var student in university.Students)

ConsoleOutput(student.ToString());

break;

case "teachers":

ConsoleOutput("Все преподаватели:");

foreach (var teacher in university.Teachers)

ConsoleOutput(teacher.ToString());

break;

default:

ConsoleOutput("get {students|teachers|persons} [params] - вывести всех студентов/преподавателей/персон");

break;

}

else ConsoleOutput("get {students|teachers|persons} [params] - вывести всех студентов/преподавателей/персон");

break;

case "find":

if (input.Length > 1)

switch (input[1])

{

case "lastname":

if (input.Length > 2)

foreach (var person in university.FindByLastName(input[2]))

ConsoleOutput(person.ToString());

else

ConsoleOutput("find lastname [фамилия] - найти персону по фамилии");

break;

case "department":

if (input.Length > 2)

foreach (var person in university.FindByDepartment(input[2]))

ConsoleOutput(person.ToString());

else

ConsoleOutput("find lastname [фамилия] - найти персону по фамилии");

break;

default:

ConsoleOutput("find lastname [фамилия] - найти персону по фамилии");

ConsoleOutput("find department [название\_факультета] - найти преподавателя по факультету");

break;

}

else

{

ConsoleOutput("find lastname [фамилия] - найти персону по фамилии");

ConsoleOutput("find department [название\_факультета] - найти преподавателя по факультету");

}

break;

case "quit":

ConsoleOutput("Выходим...");

isQuit = true;

break;

default:

ConsoleOutput("Не понимаю, введите <help>");

break;

}

}

}

static private void ConsoleOutput(string outputText)

{

Console.WriteLine(CONSOLE\_OUTPUT + outputText);

}

static private string ConsoleInput()

{

Console.ForegroundColor = ConsoleColor.Blue;

Console.Write(CONSOLE\_INPUT);

Console.ForegroundColor = ConsoleColor.White;

return Console.ReadLine();

}

}

static class Helper

{

public static int YearsFromSometimesToToday(DateTime someDate)

{

DateTime today = DateTime.Now;

return today.Year - someDate.Year +

((someDate.Month >= today.Month && someDate.Day >= today.Day) ? -1 : 0);

}

}

interface IPerson

{

string Name { get; }

string Patronomic { get; }

string LastName { get; }

DateTime Date { get; }

int Age { get; }

}

class Student : IPerson

{

public Student(string name, string patronomic, string lastName,

ushort course, string group, float averangeGrade, DateTime date)

{

Name = name;

Patronomic = patronomic;

LastName = lastName;

Course = course;

Group = group;

AvarangeGrade = averangeGrade;

Date = date;

}

public static Student Parse(string studentString)

{

var data = studentString.Split();

Student student = new Student(

data[0], data[1], data[2], ushort.Parse(data[3]),

data[4], float.Parse(data[5]),

DateTime.ParseExact(data[6], @"dd/MM/yyyy", CultureInfo.InvariantCulture)

);

return student;

}

public bool Equals(Student student)

{

if (Name == student.Name && Patronomic == student.Patronomic &&

LastName == student.LastName && Date == student.Date &&

Course == student.Course && Group == student.Group &&

AvarangeGrade == student.AvarangeGrade

)

return true;

return false;

}

public override string ToString()

{

string result = $"{Name} {Patronomic} {LastName}, Курс: {Course}, Группа: {Group}, Средний Балл: {AvarangeGrade}, Дата Рождения: {Date.ToString(@"dd MMM yyyy")}";

return result;

}

public string Name { get; }

public string Patronomic { get; }

public string LastName { get; }

public DateTime Date { get; }

public ushort Course { get; }

public string Group { get; }

public float AvarangeGrade { get; }

public int Age

{

get => Helper.YearsFromSometimesToToday(Date);

}

}

class Teacher : IPerson

{

public enum Positions : ushort

{

Ректор,

Декан,

ЗамДекана,

Преподаватель,

}

public Teacher(string name, string patronomic, string lastName,

string department, Positions position,

DateTime jobPlacement, DateTime date)

{

Name = name;

LastName = lastName;

Patronomic = patronomic;

Department = department;

Position = position;

JobPlacement = jobPlacement;

Date = date;

}

public static Teacher Parse(string teacherString)

{

var data = teacherString.Split();

Teacher teacher = new Teacher(

data[0], data[1], data[2], data[3],

(Positions)Enum.Parse(typeof(Positions), data[4]),

DateTime.ParseExact(data[5], @"dd/MM/yyyy", CultureInfo.InvariantCulture),

DateTime.ParseExact(data[6], @"dd/MM/yyyy", CultureInfo.InvariantCulture)

);

return teacher;

}

public bool Equals(Teacher teacher)

{

if (Name == teacher.Name && Patronomic == teacher.Patronomic &&

LastName == teacher.LastName && Date == teacher.Date &&

Department == teacher.Department && Position == teacher.Position &&

JobPlacement == teacher.JobPlacement

)

return true;

return false;

}

public override string ToString()

{

string result = $"{Name} {Patronomic} {LastName}, {Position}{(Position == Positions.Ректор ? "" : " факультета " + Department)}, Стаж: {Experience} лет, Дата Рождения: {Date.ToString(@"dd MMM yyyy")}";

return result;

}

public string Name { get; }

public string Patronomic { get; }

public string LastName { get; }

public DateTime Date { get; }

public string Department { get; }

public Positions Position { get; }

public DateTime JobPlacement { get; }

public int Experience

{

get => Helper.YearsFromSometimesToToday(JobPlacement);

}

public int Age

{

get => Helper.YearsFromSometimesToToday(Date);

}

}

interface IUniversity

{

void Add(IPerson person);

bool Remove(IPerson person);

IEnumerable<IPerson> FindByLastName(string lastName);

IEnumerable<Teacher> FindByDepartment(string text);

IEnumerable<IPerson> Persons { get; }

IEnumerable<Student> Students { get; }

IEnumerable<Teacher> Teachers { get; }

}

class University : IUniversity

{

private List<IPerson> persons = new List<IPerson>();

public void Add(IPerson person)

{

persons.Add(person);

}

public bool Remove(IPerson person)

{

return persons.Remove(person);

}

public IEnumerable<IPerson> FindByLastName(string lastName)

{

return persons.Where(person => person.LastName == lastName)

.OrderBy(person => person.LastName);

}

public IEnumerable<Teacher> FindByDepartment(string text)

{

return persons.OfType<Teacher>()

.Where(teacher => teacher.Department == text)

.OrderBy(teacher => teacher.Position);

}

public IEnumerable<IPerson> Persons

{

get => persons.OrderBy(person => person.Date);

}

public IEnumerable<Student> Students

{

get => persons.OfType<Student>().OrderBy(student => student.Date);

}

public IEnumerable<Teacher> Teachers

{

get => persons.OfType<Teacher>().OrderBy(teacher => teacher.Position);

}

}

}