Задание:

Вариант 3. Требования к программе

Определить новые версии классов **Team** и **ResearchTeam** из лабораторной работы 2.

В новой версии класса **ResearchTeam** использовать типы

* System.Collections.Generic.List<Person>для списка участников проекта;
* System.Collections.Generic.List<Paper> для списка публикаций;

В новых версиях классов **Team** и **ResearchTeam** сохранить все остальные поля, свойства и методы из предыдущих версий, внести необходимые исправления в код свойств и методов из-за изменения типа полей для списков.

В новую версию класса **Team** добавить реализацию интерфейсa System.IComparable для сравнения объектов Team по полю с номером регистрации.

В новую версию класса **ResearchTeam**  добавить реализацию интерфейсa System.Collections.Generic.IComparer<ResearchTeam> для сравнения объектов ResearchTeam по названию темы исследований.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<ResearchTeam>, который можно использовать для сравнения объектов типа ResearchTeam по числу публикаций.

Определить класс **ResearchTeamCollection**,который содержит

закрытое поле типа System.Collections.Generic.List<ResearchTeam>;

метод void AddDefaults (), c помощью которого в список List<ResearchTeam> можно добавить некоторое число элементов типа ResearchTeam для инициализации коллекции по умолчанию;

метод void AddResearchTeams ( params ResearchTeam []) для добавления элементов в список List<ResearchTeam>;

перегруженную версию виртуального метода string ToString() для формирования строки c информацией обо всех элементах списка List<ResearchTeam>, которая содержит значения всех полей, список участников проекта и список публикаций для каждого элемента ResearchTeam;

виртуальный метод string ToShortString(), который формирует строку с информацией обо всех элементах списка List<ResearchTeam>, включающую значения всех полей, число участников проекта и число публикаций для каждого элемента ResearchTeam, но без списков участников и публикаций.

В классе **ResearchTeamCollection** определить методы, выполняющие сортировку списка List<ResearchTeam>

по номеру регистрации с использованием интерфейса IComparable, реализованного в классе Team;

по названию темы исследований с использованием интерфейса IComparer<ResearchTeam>, реализованного в классе ResearchTeam;

по числу публикаций с использованием интерфейса IComparer<ResearchTeam>, реализованного во вспомогательном классе.

В классе **ResearchTeamCollection** определить свойства и методы, выполняющие операции со списком List<ResearchTeam> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций со списком:

свойство типа int (только с методом get), возвращающее минимальное значение номера регистрации для элементов списка List<ResearchTeam>; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска минимального значения номера регистрации надо использовать метод Min класса System.Linq.Enumerable;

свойство типа IEnumerable<ResearchTeam> (только с методом get), возвращающее подмножество элементов списка List<ResearchTeam> с продолжительностью исследований TimeFrame.TwoYears; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

метод List<ResearchTeam> NGroup(int value), который возвращает список, в который входят элементы ResearchTeam из списка List<ResearchTeam> с заданным числом участников исследования; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

Определить класс **TestCollections**, в котором в качестве типа TKey используется класс Team, а в качестве типа TValue - класс ResearchTeam. Класс содержит закрытые поля с коллекциями типов

* System.Collections.Generic.List<Team>;
* System.Collections.Generic.List<string>;
* System.Collections.Generic.Dictionary <Team, ResearchTeam>;
* System.Collections.Generic.Dictionary <string, ResearchTeam>.

В классе **TestCollections** определить

статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа ResearchTeam и используется для автоматической генерации элементов коллекций;

конструктор c параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;

метод, который вычисляет время поиска элемента в списках List<Team> и List<string>, время поиска элемента по ключу и время поиска значения элемента в коллекциях-словарях Dictionary<Team, ResearchTeam> и Dictionary <string, ResearchTeam>.

В методе **Main()**

1. Создать объект типа ResearchTeamCollection. Добавить в коллекцию несколько элементов типа ResearchTeam с разными значениями полей и вывести объект ResearchTeamCollection.
2. Для созданного объекта ResearchTeamCollection вызвать методы, выполняющие сортировку списка List<ResearchTeam> по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку

* по номеру регистрации;
* по названию темы исследований;
* по числу публикаций.

1. Вызвать методы класса ResearchTeamCollection, выполняющие операции со списком List<ResearchTeam>, и после каждой операции вывести результат операции. Выполнить

* вычисление минимального значения номера регистрации для элементов списка; вывести минимальное значение;
* фильтрацию проектов с продолжительностью исследований TimeFrame.TwoYears, вывести результат фильтрации;

Код:

**Main:**

class Program

{

static void Main(string[] args)

{

//Console.BackgroundColor = ConsoleColor.White;

//Console.ForegroundColor = ConsoleColor.Green;

//1.Создать объект типа ResearchTeamCollection. Добавить в коллекцию несколько элементов типа ResearchTeam с разными значениями полей и вывести объект ResearchTeamCollection

ResearchTeamCollection MyCollection = new ResearchTeamCollection();

MyCollection.AddResearchTeams(new ResearchTeam("Котология", "НИИ иследования котов", 23, TimeFrame.TwoYears));

MyCollection.AddResearchTeams(new ResearchTeam("Пушистология", "НИИ иследования котов", 17, TimeFrame.Year));

MyCollection.AddResearchTeams(new ResearchTeam("Лежание на боку", "НИИ иследования котов", 18, TimeFrame.Year));

MyCollection.AddResearchTeams(new ResearchTeam("Мурчание", "НИИ иследования котов", 20, TimeFrame.Year));

//Для созданного объекта ResearchTeamCollection вызвать методы, выполняющие сортировку списка List<ResearchTeam> по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку

foreach (ResearchTeam t in MyCollection)

{

Console.WriteLine(t.ToShortString());

}

MyCollection.ToSortByRegistrNumber();

Console.WriteLine("отсортированио по номеру регистрации");

foreach (ResearchTeam t in MyCollection)

{

Console.WriteLine(t.ToShortString());

}

MyCollection.SortByString();

Console.WriteLine("отсортировано по строкам");

foreach (ResearchTeam t in MyCollection)

{

Console.WriteLine(t.ToShortString());

}

MyCollection.SortByPublications();

Console.WriteLine("отсортировано по публикациям");

foreach (ResearchTeam t in MyCollection)

{

Console.WriteLine(t.ToShortString());

}

Console.WriteLine();

//вычисление минимального значения номера регистрации для элементов списка; вывести минимальное значение;

Console.WriteLine("min "+MyCollection.MinRegNumber.ToString());

//фильтрацию проектов с продолжительностью исследований TimeFrame.TwoYears, вывести результат фильтрации;

Console.WriteLine(MyCollection.TwoYearsLong.ToString());

Console.ReadKey();

}

**INameAndCopy**

namespace test

{

interface INameAndCopy

{

string Name { get; set; }

object DeepCopy();

}

}

**ResearchTeamCollection**

namespace test

{

class ResearchTeamCollection : IEnumerable

{

private List<ResearchTeam> SomeResearchTeams = new List<ResearchTeam>();// акрытое поле типа System.Collections.Generic.List<ResearchTeam>;

//метод void AddDefaults (), c помощью которого в список List<ResearchTeam> можно добавить некоторое число элементов типа ResearchTeam для инициализации коллекции по умолчанию;

public void AddDefaults()

{

for (int i = 0; i < 3; i++)

{

SomeResearchTeams.Add(new ResearchTeam());

}

}

//• метод void AddResearchTeams ( params ResearchTeam []) для добавления элементов в список List<ResearchTeam>;

public void AddResearchTeams(params ResearchTeam[] AdditionalTeams)

{

SomeResearchTeams.AddRange(AdditionalTeams);

}

//перегруженную версию виртуального метода string ToString() для формирования строки c информацией обо всех элементах списка List<ResearchTeam>, которая содержит значения всех полей, список

public override string ToString()

{

string ResTeamString = "";

foreach (ResearchTeam team in SomeResearchTeams)

{

ResTeamString += team.ToString();

}

return ResTeamString;

}

//виртуальный метод string ToShortString(), который формирует строку с информацией обо всех элементах списка List<ResearchTeam>, включающую значения всех полей, число участников проекта и число публикаций для каждого элемента ResearchTeam, но без списков участников и публикаций.

public string ToShortString()

{

string ResTeamString = "";

foreach (ResearchTeam team in SomeResearchTeams)

{

ResTeamString += team.ToShortString();

}

return ResTeamString;

}

//по номеру регистрации с использованием интерфейса IComparable, реализованного в классе Team;

public void ToSortByRegistrNumber()

{

SomeResearchTeams.Sort((x, y) => x.RegistrationNumber.CompareTo(y.RegistrationNumber));

}

//по названию темы исследований с использованием интерфейса IComparer<ResearchTeam>, реализованного в классе ResearchTeam;

public void SortByString()

{

SomeResearchTeams.Sort();

}

//по числу публикаций с использованием интерфейса IComparer<ResearchTeam>, реализованного во вспомогательном классе.

public void SortByPublications()

{

ResearchTeamComparer comp = new ResearchTeamComparer();

SomeResearchTeams.Sort(comp);

}

// свойство типа int (только с методом get), возвращающее минимальное значение номера регистрации для элементов списка List<ResearchTeam>;

public int MinRegNumber

{

get

{

if (SomeResearchTeams.Count == 0)

{

return 0;

}

return SomeResearchTeams.Min(teams => teams.RegistrationNumber);

}

}

//свойство типа IEnumerable<ResearchTeam> (только с методом get), возвращающее подмножество элементов списка List<ResearchTeam> с продолжительностью исследований TimeFrame.TwoYears; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

public IEnumerable<ResearchTeam> TwoYearsLong

{

get

{

IEnumerable<ResearchTeam> TwoTearsL = SomeResearchTeams.Where(time => time.ResearchDuration == TimeFrame.TwoYears);

return TwoTearsL;

}

}

//• метод List<ResearchTeam> NGroup(int value), который возвращает список, в который входят элементы ResearchTeam из списка List<ResearchTeam> с заданным числом участников исследования; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

public IEnumerator GetEnumerator()

{

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

{

yield return SomeResearchTeams[i];

}

}

public List<ResearchTeam> NGroup(int value)

{

IEnumerable<IGrouping<int, ResearchTeam>> someGroup = SomeResearchTeams.GroupBy(team => team.ListOfParticipants.Count);

foreach (IGrouping<int, ResearchTeam> teams in someGroup)

{

if (teams.Key == value)

{

return teams.ToList<ResearchTeam>();

}

else

{

throw new ArgumentNullException("There are no such number!");

}

}

return null;

}

}

}

**Test Collections**

namespace test

{

class TestCollections

{

//Определить класс TestCollections, в котором в качестве типа TKey используется класс Team, а в качестве типа TValue - класс ResearchTeam. Класс содержит закрытые поля с коллекциями типов

private List<Team> ListOfTeam = new List<Team>();

private List<string> ListOfString = new List<string>();

private Dictionary<Team, ResearchTeam> Team\_Rea\_Dict = new Dictionary<Team, ResearchTeam>();

private Dictionary<string, ResearchTeam> String\_Rea\_Dict = new Dictionary<string, ResearchTeam>();

public List<Team> ListTeam { get { return ListOfTeam; } set { ListOfTeam = value; } }

private List<string> ListString { get { return ListOfString; } set { ListOfString = value; } }

private Dictionary<Team, ResearchTeam> Team\_Re\_Dict { get { return Team\_Rea\_Dict; } set { Team\_Rea\_Dict = value; } }

private Dictionary<string, ResearchTeam> String\_Re\_Dict { get { return String\_Rea\_Dict; } set { String\_Rea\_Dict = value; } }

// статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа ResearchTeam и используется для автоматической генерации элементов коллекций;

public static ResearchTeam GenerateElement(int value)

{

ResearchTeam a = new ResearchTeam();

a.RegistrationNumber = value;

return a;

}

//• конструктор c параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;

public TestCollections(int Count)

{

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

{

ResearchTeam rs = GenerateElement(i);

ListOfTeam.Add(rs.Team);

ListOfString.Add(rs.Team.ToString());

Team\_Rea\_Dict.Add(rs.Team, rs);

String\_Rea\_Dict.Add(rs.Team.ToString(), rs);

}

}

//метод, который вычисляет время поиска элемента в списках List<Team> и List<string>, время поиска элемента по ключу и время поиска значения элемента в коллекциях-словарях Dictionary<Team, ResearchTeam> и Dictionary <string, ResearchTeam>.

public void TimeOfSearching(string str, Team team, ResearchTeam resT)

{

int time1 = Environment.TickCount;

ListOfTeam.BinarySearch(team);

Console.WriteLine((time1 - Environment.TickCount).ToString());

int time2 = Environment.TickCount;

ListOfString.BinarySearch(str);

Console.WriteLine((time2 - Environment.TickCount).ToString());

int time3 = Environment.TickCount;

ResearchTeam a = Team\_Rea\_Dict[team];

Console.WriteLine((time3 - Environment.TickCount).ToString());

int time4 = Environment.TickCount;

ResearchTeam b = String\_Rea\_Dict[str];

Console.WriteLine((time4 - Environment.TickCount).ToString());

Console.WriteLine((time4 - Environment.TickCount).ToString());

}

}

}

**Person**

namespace test

{

class Person//Определить класс Person, который имеет

{

private string \_Name; //закрытое поле типа string, в котором хранится имя;

private string \_Surname;//закрытое поле типа string, в котором хранится фамилия

private System.DateTime \_Birthday;//закрытое поле типа System.DateTime для даты рождения

//конструктор c тремя параметрами типа string, string, DateTime для инициализации всех полей класса

public Person(string Name, string Surname, DateTime Birthday)

{

\_Name = Name;

\_Surname = Surname;

\_Birthday = Birthday;

}

//конструктор без параметров, инициализирующий все поля класса некоторыми значениями по умолчанию.

public Person() : this("Джек", "Лондон", new DateTime(1998, 11, 12))

{

}

// В классе Person определить свойства c методами get и set:

//свойство типа string для доступа к полю с именем;

public string PersName

{

get

{

return \_Name;

}

}

// свойство типа string для доступа к полю с фамилией;

public string PersonSurname

{

get

{

return \_Surname;

}

}

//свойство типа DateTime для доступа к полю с датой рождения;

public DateTime PersonBirthday

{

get

{

return \_Birthday;

}

}

//свойство типа int c методами get и set для получения информации(get) и изменения(set) года рождения в закрытом поле типа DateTime, в котором хранится дата рождения.

public int intBirthday

{

get

{

return Convert.ToInt32(\_Birthday);

}

set

{

\_Birthday = Convert.ToDateTime(value);

}

}

//перегруженную(override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса;

public override string ToString()

{

return string.Format("{0} {1} was born {2}", \_Name, \_Surname, \_Birthday);

}

// виртуальный метод string ToShortString(), который возвращает строку, содержащую только имя и фамилию.

public string ToShortString()

{

return string.Format("{0} {1}", \_Name, \_Surname);

}

//контент 9 лабы переопределить (override) виртуальный метод bool Equals (object obj);

public override bool Equals(Object obj)

{

return ((Person)obj)?.GetHashCode() == GetHashCode();

}

// переопределить виртуальный метод int GetHashCode();

public override int GetHashCode()

{

return \_Birthday.GetHashCode() + \_Name.GetHashCode() + \_Surname.GetHashCode();

}

public static bool operator ==(Person p1, Person p2)

{

return p1.Equals(p2);

}

public static bool operator !=(Person p1, Person p2)

{

return !p1.Equals(p2);

}

public object DeepCopy()

{

Person person = new Person(PersName, PersonSurname, PersonBirthday);

return person;

}

}

}

**Team**

namespace test

{

class Team : INameAndCopy, IComparable

{

protected string \_Organisation;

protected int \_RegistrationNumber;

// конструктор без параметров для инициализации по умолчанию;

public Team(string Organisation, int RegistrationNumber)

{

\_Organisation = Organisation;

\_RegistrationNumber = RegistrationNumber;

}

//свойство типа string для доступа к полю с названием организации

public string Organisation

{

get

{

return \_Organisation;

}

set

{

\_Organisation = value;

}

}

//свойство типа int для доступа к полю с номером регистрации

public int RegistrationNumber

{

get

{

return \_RegistrationNumber;

}

set

{

if (value <= 0)

{

throw new ArgumentOutOfRangeException("Error! GroupNumber out of range(100, 599).");

}

else

{

\_RegistrationNumber = value;

}

}

}

string INameAndCopy.Name

{

get

{

return string.Format("Группа организации {0} с номером {1}", \_Organisation, \_RegistrationNumber);

}

set

{

throw new NotImplementedException();

}

}

public DateTime Date { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public Team() : this("Неизвестная организация", 1) { }

//определить виртуальный метод object DeepCopy();

public virtual object DeepCopy()

{

return new Team(this.\_Organisation, this.\_RegistrationNumber);

}

// переопределить (override) виртуальный метод bool Equals (object obj)

public virtual bool Equals(object obj)

{

if (obj == null)

{

return false;

}

Team objAsTeam = obj as Team;

if (objAsTeam as Team == null)

{

return false;

}

return objAsTeam.Organisation == this.Organisation && objAsTeam.RegistrationNumber == this.RegistrationNumber;

}

// определить операции == и !=

static public bool operator ==(Team l\_Team, Team r\_Team)

{

if (ReferenceEquals(l\_Team, r\_Team))

{

return true;

}

if ((((object)l\_Team) == null) || (((object)r\_Team) == null))

{

return false;

}

return false;

}

static public bool operator !=(Team l\_Team, Team r\_Team)

{

return !(l\_Team == r\_Team);

}

// переопределить виртуальный метод int GetHashCode();

public virtual new int GetHashCode()

{

int HashCode = 0;

foreach (char ch in \_Organisation.ToCharArray())

{

HashCode += (int)Convert.ToUInt32(ch);

}

HashCode += \_RegistrationNumber;

return HashCode;

}

// перегруженную версию виртуального метода string ToString() для фор-мирования строки со значениями всех полей класса

public virtual new string ToString()

{

return string.Format("Группа организации {0} с номером {1}", \_Organisation, \_RegistrationNumber);

}

public int CompareTo(object? obj)

{

throw new NotImplementedException();

}

}}

**ResearchTeam**

namespace test

{

class ResearchTeam : Team, INameAndCopy, IComparable<ResearchTeam> //Определить класс ResearchTeam, который имеет

{

private string Theme; // закрытое поле типа string c названием темы исследований;

private string NameOfOrganisation; // закрытое поле типа string с названием организации;

private int RegNumber; //• закрытое поле типа int – регистрационный номер;

public TimeFrame ResearchDuration; //закрытое(уже нет) поле типа TimeFrame для информации о продолжительности исследований;

//закрытое поле типа System.Collections.ArrayList для списка публикаций (объектов типа Paper).

List<Person> ProjectParticipants = new List<Person>();

List<Paper> Publications = new List<Paper>();

public List<Paper> ListOfPublication { get { return Publications; } set { Publications = value; } }

Random rand = new Random();

public List<Person> ListOfParticipants { get { return ProjectParticipants; } set { ProjectParticipants = value; } }

//В классе ResearchTeam определить конструкторы:

public ResearchTeam( string InvestigationTheme, string Organisation, int RegistrationNumber, TimeFrame InvestigationDuration)// конструктор c параметрами типа string, Person, DateTime для инициализации всех свойств класса

{

ResearchDuration = InvestigationDuration;

Theme = InvestigationTheme;

RegNumber = RegistrationNumber;

NameOfOrganisation = Organisation;

}

// конструктор без параметров, инициализирующий поля класса значениями по умолчанию.

public ResearchTeam() {

this.Theme = "Котики";

NameOfOrganisation = "КГЭУ";

ResearchDuration = TimeFrame.Year;

RegNumber = rand.Next(0, 100);

}

public override string ToString()

{

string stringListOfPublications = "";

foreach (Paper pap in Publications)

{

stringListOfPublications += pap.ToString() + "\r\n";

}

return string.Format("\r\n Тема: {0}, Длительность: {1} \r\n Организация:{2} \r\n Публикация: {3} \r\n Автор: {4}", Theme, ResearchDuration, NameOfOrganisation, stringListOfPublications, ProjectParticipants);

}

//виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка публикаций.

public string ToShortString()

{

return string.Format("\r\n Тема: {0}, Длительность: {1} \r\n Организация:{2} \r\n", Theme, ResearchDuration, NameOfOrganisation);

}

public object DeepCopy()

{

ResearchTeam other = (ResearchTeam)MemberwiseClone();

other.NameOfOrganisation = string.Copy(NameOfOrganisation);

other.ResearchDuration = ResearchDuration;

other.Theme = string.Copy(Theme);

other.RegNumber = RegNumber;

other.Publications = new List<Paper>(Publications);

other.ProjectParticipants = new List<Person>(ProjectParticipants);

return other;

}

//метод void AddMembers ( params Person[] ) для добавления элементов в список участников проекта;

public void AddMembers(params Person[] particips)

{

foreach (var item in particips)

{

ProjectParticipants.Add(item);

}

}

public Team Team //свойство типа Team; метод get свойства возвращает объект типа Team, данные которого совпадают с данными подобъекта базового класса, метод set присваивает значения полям из подобъекта базового класса;

{

get

{

return new Team(NameOfOrganisation, RegNumber);

}

set

{

Organisation = value.Organisation;

RegistrationNumber = value.RegistrationNumber;

}

}

//итератор для последовательного перебора участников проекта (объектов типа Person), не имеющих публикаций;

public IEnumerable<Person> MembersWithoutPublications()

{

ArrayList AutorsWithoutP = new ArrayList();

bool Bool;

foreach (Person pers in ProjectParticipants)

{

Bool = true;

foreach (Paper pap in Publications)

{

if (pap.Author == pers)

{

Bool = false;

break;

}

}

if (Bool)

{

AutorsWithoutP.Add(pers);

Console.WriteLine(pers.ToShortString());

}

}

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

{

yield return (Person)AutorsWithoutP[i];

Console.Write(((Person)AutorsWithoutP[i]).ToShortString());

}

}

//итератор с параметром типа int для перебора публикаций, вышедших за последние n лет, в котором число n передается через параметр итератора.

public IEnumerable<Paper> LastPapers(int N\_years)

{

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

{

if (((Paper)Publications[i]).Data.Year >= DateTime.Now.Year - N\_years)

{

yield return (Paper)Publications[i];

Console.Write(((Paper)Publications[i]).ToString());

}

}

}

public IEnumerator GetEnumerator()

{

foreach (Person person in ProjectParticipants)

{

yield return person;

}

}

public int CompareTo(ResearchTeam other)

{

if (other == null)

{

throw new ArgumentException("Wrong argument!");

}

return Theme.CompareTo(other.Theme);

}

delegate int KeySelector<TKey>(ResearchTeam rt);

}

}

**Paper**

namespace test

{

class Paper

{

public string Name { get; set; }

public Person Author { get; set; }

public DateTime Data { get; set; }

public Paper(string name, Person author, DateTime data)

{

Name = name;

Author = author;

Data = data;

}

// конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию

public Paper() : this("Мартин Иден", new Person(), new DateTime(1889, 6, 1))

{ }

// перегруженную версию виртуального метода string To-String() для формирования строки со значениями всех полей класса

public override string ToString()

{

return string.Format("Author {0} write book {1}. Data = {2}", Author, Name, Data);

}

public virtual object DeepCopy() //в классе Paper определить виртуальный метод object DeepCopy();

{

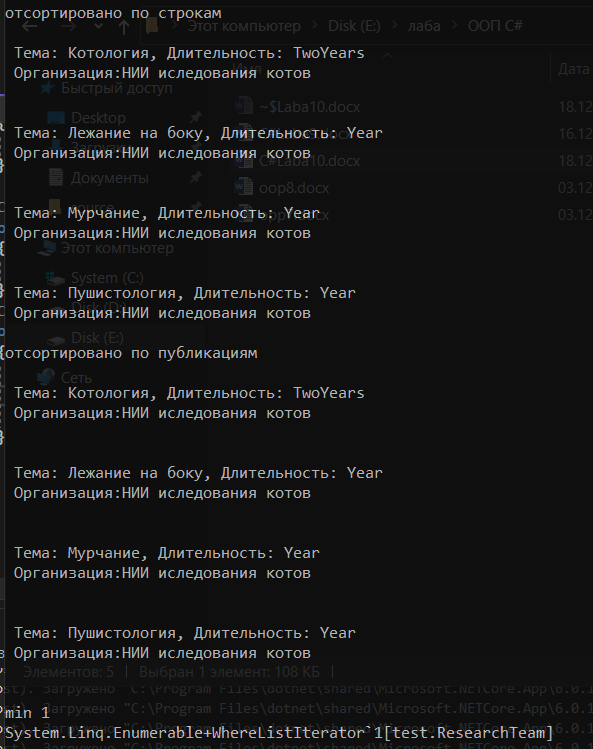
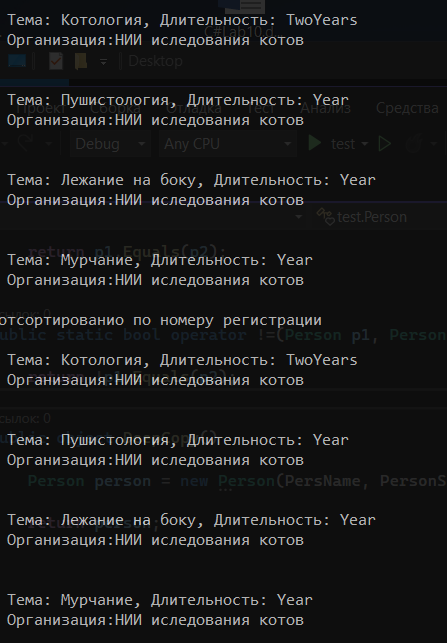
return new Paper(this.Name, this.Author, this.Data);

}

}

}

Результат:

****