Министерство образования Республики Беларусь

Министерство образования и науки РФ

БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ

Кафедра "ПОИТ"

**"** **Объектно-ориентированное программирование "**

Отчет

по лабораторной работе №4

Классы, свойства, индексаторы. Одномерные, прямоугольные и

Ступенчатые массивы

Вариант 2

Выполнил ст.гр. ПИР-161

Кашица Е.М.

Проверил Горбатенко Н.Н.

г. Могилёв 2018

Задание.

## **Вариант 3. Требования к программе**

Определить интерфейс

interface INameAndCopy

{

string Name

{get; set;}

object DeepCopy();

}

Определить новые версии классов **Person**, **Paper** и **ResearchTeam** из лабораторной работы № 1. Класс ResearchTeam определить как производный от класса Team. В классы Team и ResearchTeam добавить реализацию интерфейса INameAndCopy.

В классе **Paper** определить виртуальный метод object DeepCopy().

В новой версии класса **Person** дополнительно

* переопределить метод virtial bool Equals(object obj) и определить операции == и != так, чтобы равенство объектов типа Person трактовалось как совпадение всех данных объектов, а не ссылок на объекты Person;
* переопределить виртуальный метод int GetHashCode();
* определить виртуальный метод object DeepCopy().

Определить класс **Team**. Класс **Team** имеет

* защищенное (protected) поле типа string с названием организации;
* защищенное поле типа int − регистрационный номер.

В классе **Team** определить:

* конструктор с параметрами типа string и int для инициализации полей класса;
* конструктор без параметров для инициализации по умолчанию;
* свойство типа string для доступа к полю с названием организации;
* свойство типа int для доступа к полю с номером регистрации; в методе set бросить исключение, если присваиваемое значение меньше или равно 0; при создании объекта-исключения использовать один из определенных в библиотеке CLR классов-исключений, инициализировать объект-исключение с помощью конструктора с параметром типа string.

В классе **Team**

* определить виртуальный метод object DeepCopy();
* реализовать интерфейс INameAndCopy.

В классе **Team** переопределить (override):

* виртуальный метод virtial bool Equals (object obj) и определить операции == и != так, чтобы равенство объектов типа Team трактовалось как совпадение всех данных объектов, а не ссылок на объекты Team;
* виртуальный метод int GetHashCode();
* виртуальный метод string ToString() для формирования строки со значениями всех полей класса.

Новая версия класса **ResearchTeam** имеет базовый класс **Team** и следующие поля:

* закрытое поле типа string c названием темы исследований;
* закрытое поле типа TimeFrame с информацией о продолжительности исследований;
* закрытое поле типа System.Collections.ArrayList со списком участников проекта (объектов типа Person);
* закрытое поле типа System.Collections.ArrayList для списка публикаций (объектов типа Paper).

Код следующих конструкторов, методов и свойств из старой версии класса **ResearchTeam** необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс **Team**, и в новой версии класса **ResearchTeam** для списка публикаций используется тип System.Collections.ArrayList:

* конструктор c параметрами типа string, string, int, TimeFrame для инициализации соответствующих полей класса;
* конструктор без параметров для инициализации по умолчанию;
* свойство типа System.Collections.ArrayList для доступа к полю со списком публикаций;
* свойство типа Paper (только с методом get), которое возвращает ссылку на публикацию с самой поздней датой выхода; если список публикаций пустой, свойство возвращает значение null;
* метод void AddPapers (params Paper[]) для добавления элементов в список публикаций;
* перегруженная версия виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список публикаций и список участников проекта;
* метод string ToShortString(), который формирует строку со значениями всех полей класса без списка публикаций и списка участников проекта.

Дополнительно в новой версии класса **ResearchTeam** определить

* перегруженную версию виртуального метода object DeepCopy();
* свойство типа System.Collections.ArrayList для доступа к полю со списком участников проекта;
* метод void AddMembers (params Person[]) для добавления элементов в список участников проекта;
* 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;
* реализовать интерфейс INameAndCopy.

В новой версии класса **ResearchTeam** определить

* итератор для последовательного перебора участников проекта (объектов типа Person), не имеющих публикаций;
* итератор с параметром типа int для перебора публикаций, вышедших за последние n лет, в котором число n передается через параметр итератора.

В методе **Main()**

1. Создать два объекта типа Team с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэш-кодов для объектов.
2. В блоке try/catch присвоить свойству с номером регистрации некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.
3. Создать объект типа ResearchTeam, добавить элементы в список публикаций и список участников проекта и вывести данные объекта ResearchTeam.
4. Вывести значение свойства Team для объекта типа ResearchTeam.
5. С помощью метода DeepCopy() создать полную копию объекта ResearchTeam. Изменить данные в исходном объекте ResearchTeam и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.
6. С помощью оператора foreach для итератора, определенного в классе ResearchTeam, вывести список участников проекта, которые не имеют публикаций.
7. С помощью оператора foreach для итератора с параметром, определенного в классе ResearchTeam, вывести список всех публикаций, вышедших за последние два года.

КОД:

class Program

{

static void Main(string[] args)

{

Team team3 = new Team("1", "1", 1);

Team team4 = (Team)team3.DeepCopy();

Console.WriteLine("До:");

Console.WriteLine(team3.ToString() + " " + team3.GetHashCode());

Console.WriteLine(team4.ToString() + " " + team4.GetHashCode());

team4.RegNumber = 2;

team4.Name = "2";

Console.WriteLine("После:");

Console.WriteLine(team3.ToString()+" "+team3.GetHashCode());

Console.WriteLine(team4.ToString() + " " + team4.GetHashCode());

try

{

team3.RegNumber = -1;

}

catch (ArgumentException)

{

Console.WriteLine("Wrong numb");

}

ResearchTeam research = new ResearchTeam("Team1", "BRU", 1, "Math", TimeFrame.Long);

Person per1 = new Person("Ikar", "Duran", DateTime.Today);

Person per2 = new Person("PersonWhithoutPaper","Ivanov", DateTime.Today);

DateTime time = new DateTime(1999, 01, 01);

Paper[] papers = {

new Paper("pap1", per1, DateTime.Now),

new Paper("pap2", per1, time),

new Paper("pap3", per1, time) };

ArrayList arrayPaper = new ArrayList(papers);//here

research.Persons.Add(per1);

research.Persons.Add(per2);

research.Papers = arrayPaper;

Console.WriteLine("Данные ResearchTeam: {0}",research.ToString());

Console.WriteLine("Значение свойства Team для объекта типа ResearchTeam: {0}",research.Team.ToString());

ResearchTeam research1 = (ResearchTeam)research.DeepCopy();

research1.Papers[0] = null;

Console.WriteLine(((Paper)research.Papers[0]).ToString());

Console.WriteLine("список участников проекта, которые не имеют публикаций:");

foreach (string item in research)

{

Console.WriteLine(item);

}

Console.WriteLine("вывести список всех публикаций, вышедших за последние два года:");

foreach (var item in research.Publications(2016))

{

Console.WriteLine(item);//here

}

Console.Read();

}

}

class Team : INameAndCopy

{

public string Name { get; set ; }

protected string nameOfOrganisation;

protected int regNumber;

public string NameOfOrganisation { get; set; }

public int RegNumber

{

get { return regNumber; }

set {

if (value <=0)

{

throw new ArgumentException(" <=0 ");

}

regNumber = value;

}

}

public Team(string name="", string nameOfOrganisation="", int regNumber=0)

{

Name = name;

this.NameOfOrganisation = nameOfOrganisation;

this.RegNumber = regNumber;

}

public override bool Equals(object obj)

{

if (obj is Team)

{

return this == (Team)obj;

}

return false;

}

static public bool operator ==(Team obj1, Team obj2)

{

return obj1.Name == obj1.Name && obj1.NameOfOrganisation == obj2.NameOfOrganisation&&obj1.RegNumber==obj2.RegNumber ;

}

static public bool operator !=(Team obj1, Team obj2)

{

return !(obj1 == obj2);

}

public virtual object DeepCopy()

{

return this.MemberwiseClone() ;

}

public override string ToString()

{

return this.Name+" "+this.NameOfOrganisation+" "+this.regNumber;

}

public override int GetHashCode()

{

return Name.GetHashCode()\*NameOfOrganisation.GetHashCode()\*regNumber.GetHashCode();

}

}

class ResearchTeam : Team,IEnumerable

{

TimeFrame lengthOfResearch;

ArrayList papers;

ArrayList persons;

string subject;

public TimeFrame LengthOfResearch

{

get { return lengthOfResearch; }

set { lengthOfResearch = value; }

}

public ArrayList Papers

{

get { return papers; }

set { papers = value; }

}

public ArrayList Persons

{

get { return persons; }

set { persons = value; }

}

public Paper LastPaper

{

get {

if (papers.Count != 0)

{

Paper minPaper = (Paper)Papers[0];

foreach (Paper item in papers)

{

if (item.Date > minPaper.Date)

{

minPaper = item;

}

}

return minPaper;

}

else

{

return null;

}

}

}

public Team Team

{

get { return new Team(Name,NameOfOrganisation,regNumber); }

set {

Name = value.Name;

nameOfOrganisation = value.NameOfOrganisation;

regNumber = value.RegNumber;

}

}

public ResearchTeam(string name, string organization, int number, string subject = "", TimeFrame lengthOfResearch=TimeFrame.Year) : base(name, organization, number)

{

this.lengthOfResearch = lengthOfResearch;

this.subject = subject;

papers = new ArrayList();

persons = new ArrayList();

}

public IEnumerator GetEnumerator()

{

int n = papers.Count;

int n1 = persons.Count;

for (int i = 0; i < n1; i++)

{

int k = 0;

for (int j = 0; j < n; j++)

{

if ((Person)persons[i] != ((Paper)papers[j]).Autor)

{

k++;

}

if (k == n)

{

yield return ((Person)persons[i]).ToString();

}

}

}

yield break;

}

public IEnumerable Publications(int t)

{

int n = papers.Count;

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

{

if (((Paper)papers[i]).Date.Year >= t)

{

yield return ((Paper)papers[i]).NameOfPublication;

}

}

yield break;

}

public void AddPapers(params Paper[] paper)

{

paper.CopyTo(papers.ToArray(), 0);

}

public void AddMembers(params Person[] person)

{

person.CopyTo(persons.ToArray(), 0);

}

public override string ToString()

{

string listPaper = "";

foreach (Paper item in papers)

{

listPaper += item.NameOfPublication + " ";

}

return Name + " " + NameOfOrganisation + " " + regNumber + " " + lengthOfResearch + " " + listPaper;

}

public virtual string ToShortString()

{

return Name + " " + NameOfOrganisation + " " + regNumber + " " + lengthOfResearch;

}

public override object DeepCopy()

{

ResearchTeam copy= (ResearchTeam)MemberwiseClone();

copy.Papers = new ArrayList(papers);

copy.Persons = new ArrayList(persons);

return copy;

}

}

class Person

{

private DateTime dateOfBorn;

public string Name { get; set; }

public string Surname { get; set; }

public DateTime DateOfBorn

{

get { return dateOfBorn; }

set { dateOfBorn = value; }

}

public int DateOfBornInt

{

get { return dateOfBorn.Year; }

set {dateOfBorn= new DateTime( value,dateOfBorn.Month,dateOfBorn.Day); }

}

public Person(string name, string surname, DateTime dateOfBorn)

{

this.Name = name;

this.Surname = surname;

this.dateOfBorn = dateOfBorn;

}

public Person() { }

public override string ToString()

{

return Name + " " + Surname + " " + dateOfBorn.Day + "." + dateOfBorn.Month + "." + dateOfBorn.Year;

}

public string ToShortString()

{

return Name + " " + Surname ;

}

public override bool Equals(object obj)

{

if (obj is Person)

{

return this == (Person)obj;

}

return false;

}

static public bool operator ==(Person obj1, Person obj2)

{

return obj1.Name == obj1.Name && obj1.Surname == obj1.Surname && obj1.DateOfBorn == obj1.DateOfBorn; ;

}

static public bool operator !=(Person obj1, Person obj2)

{

return !(obj1==obj2);

}

public override int GetHashCode()

{

return this.Name.GetHashCode()\*dateOfBorn.GetHashCode()\*Surname.GetHashCode();

}

public virtual object DeepCopy()

{

return MemberwiseClone();

}

}

class Paper

{

public string NameOfPublication { get; set; }

public Person Autor { get; set; }

public DateTime Date { get; set; }

public Paper(string nameOfPublication , Person autor,DateTime date)

{

NameOfPublication = nameOfPublication;

Autor = autor;

Date = date;

}

public Paper()

{

NameOfPublication = "";

Autor = new Person();

Date = new DateTime();

}

public override string ToString()

{

return NameOfPublication + " " + Autor + " " + Date.Day + "." + Date.Month + "." + Date.Year;

}

public virtual object DeepCopy()

{

return MemberwiseClone();

}

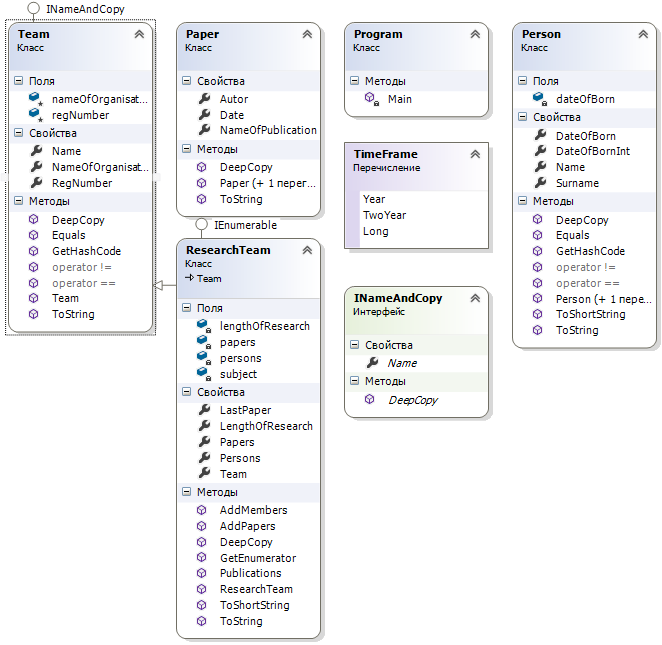
}

enum TimeFrame

{

Year,TwoYear,Long

}



Вывод: в этой лаб работе мной были использованные индексаторы перечислители реализация интерфейса и наследование.