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 интерфейсаINameAndCopy.

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:

**Main:**

class Program

{

static void Main(string[] args)

{

//Console.BackgroundColor = ConsoleColor.White;

//Console.ForegroundColor = ConsoleColor.Green;

//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.

Team MyTeam1 = new Team("Тима", 7);

Team MyTeam2 = new Team("Тима", 7);

Console.WriteLine(MyTeam1.Equals(MyTeam2));//значения хешей

Console.WriteLine(MyTeam1 == MyTeam2);

Console.WriteLine(string.Format(" Тима1: {0}, Тима2: {1} ", MyTeam1.GetHashCode(), MyTeam2.GetHashCode()));

//2.В блоке try/catch присвоить свойству с номером регистрации некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.

try {

MyTeam2.RegistrationNumber = -10;

}

catch (ArgumentOutOfRangeException ex)

{

Console.WriteLine(ex.Message);

}

//3.Создать объект типа ResearchTeam, добавить элементы в список публикаций и список участников проекта и вывести данные объекта ResearchTeam.

ResearchTeam MyTeam3 = new ResearchTeam(); //3.Создать объект типа ResearchTeam, добавить элементы в список публикаций и список участников проекта и вывести данные объекта ResearchTeam.

MyTeam3.AddMembers(new Person[3] { new Person("Дамир", "Гимадеев", new DateTime(1999, 09, 18)), new Person("Некий", "Человек ", new DateTime(1911, 11, 15)), new Person() });

MyTeam3.AddPapers(new Paper[2] { new Paper("SCP", new Person("Дамир", "Гимадеев", new DateTime(1999, 09, 18)), new DateTime(2022, 11, 25)), new Paper() });

Console.WriteLine(MyTeam3.Team.ToString());//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.

ResearchTeam MyTeam4 = (ResearchTeam)MyTeam3.DeepCopy();

MyTeam3.Organisation = "Пушистые Коты";

MyTeam3.RegistrationNumber = 7;

Console.WriteLine("team3 " + MyTeam3.ToString());

Console.WriteLine("team4 " + MyTeam4.ToString());

//6.С помощью оператора foreach для итератора, определенного в классе ResearchTeam, вывести список участников проекта, которые не имеют публикаций.

foreach (Person pers in MyTeam3.MembersWithoutPublications())

{

Console.WriteLine(pers);

}

//7.С помощью оператора foreach для итератора с параметром, определенного в классе ResearchTeam, вывести список всех публикаций, вышедших за последние два года.

foreach (Paper pap in MyTeam3.LastPapers(2))

{

Console.WriteLine(pap);

}

Console.ReadKey();

}

}

**INameAndCopy**

namespace test

{

interface INameAndCopy

{

string Name { get; set; }

object DeepCopy();

}

}

**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

{

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("Регистрационный номер должен быть больше 0 ");

}

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);

}

}

}

**ResearchTeam**

namespace test

{

class ResearchTeam : Team, INameAndCopy //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;

private TimeFrame ResearchDuration; //закрытое поле типа TimeFrame для информации о продолжительности исследований;

private System.Collections.ArrayList participants = new ArrayList(); //закрытое поле типа System.Collections.ArrayList со списком участников проекта (объектов типа Person);

private System.Collections.ArrayList Publications = new ArrayList();//закрытое поле типа System.Collections.ArrayList для списка публикаций (объектов типа Paper).

//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("Котики", "КГЭУ", 13, TimeFrame.Year) {}

//В классе ResearchTeam определить свойства c методами get и set

//свойство типа string для доступа к полю с названием темы исследований

public string theme

{

get

{

return Theme;

}

set

{

Theme = value;

}

}

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

public string nameofogr

{

get

{

return NameOfOrganisation;

}

set

{

NameOfOrganisation = value;

}

}

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

public int number

{

get

{

return RegNumber;

}

set

{

if (value <= 0)

{

throw new ArgumentOutOfRangeException();

}

else

{

RegNumber = value;

}

}

}

//Cвойство типа TimeFrame для доступа к полю с продолжительностью исследований

public TimeFrame resDur

{

get

{

return ResearchDuration;

}

set

{

ResearchDuration = value;

}

}

//свойство типа System.Collections.ArrayList для доступа к полю со списком публикаций;

public System.Collections.ArrayList ListOfPublication

{

get

{

return Publications;

}

set

{

Publications = value;

}

}

//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;

public Paper LastPublic

{

get

{

if (Publications.Count == 0)

{

return null;

}

int MaxIndex = 0;

DateTime MaxDateTime = (Publications[0] as Paper).Data;

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

{

if ((Publications[i] as Paper).Data > MaxDateTime)

{

MaxIndex = i;

MaxDateTime = (Publications[i] as Paper).Data;

}

}

Console.WriteLine("paper" + Publications[MaxIndex]);

return Publications[MaxIndex] as Paper;

}

}

//свойство типа System.Collections.ArrayList для доступа к полю со списком участников проекта;

public System.Collections.ArrayList Participants

{

get {

return participants;

}

set {

participants = value;

}

}

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

//индексатор булевского типа (только с методом get) с одним параметром типа TimeFrame; значение индексатора равно true, если значение поля с информацией о продолжительности исследований совпадает со значением индекса, и false в противном случае;

public bool this[TimeFrame timeFrame]

{

get

{

bool rez;

if (timeFrame == ResearchDuration)

{

rez = true;

}

else

{

rez = false;

}

return rez;

}

}

//метод void AddPapers(params Paper[]) для добавления элементов в список публикаций

public void AddPapers(params Paper[] additionalPapers)

{

Publications.AddRange(additionalPapers);

}

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, participants);

}

//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 ArrayList(Publications);

other.participants = new ArrayList(participants);

return other;

}

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

public void AddMembers(params Person[] particips)

{

participants.AddRange(particips);

}

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 participants)

{

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 participants)

{

yield return person;

}

}

}

}

**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();

{

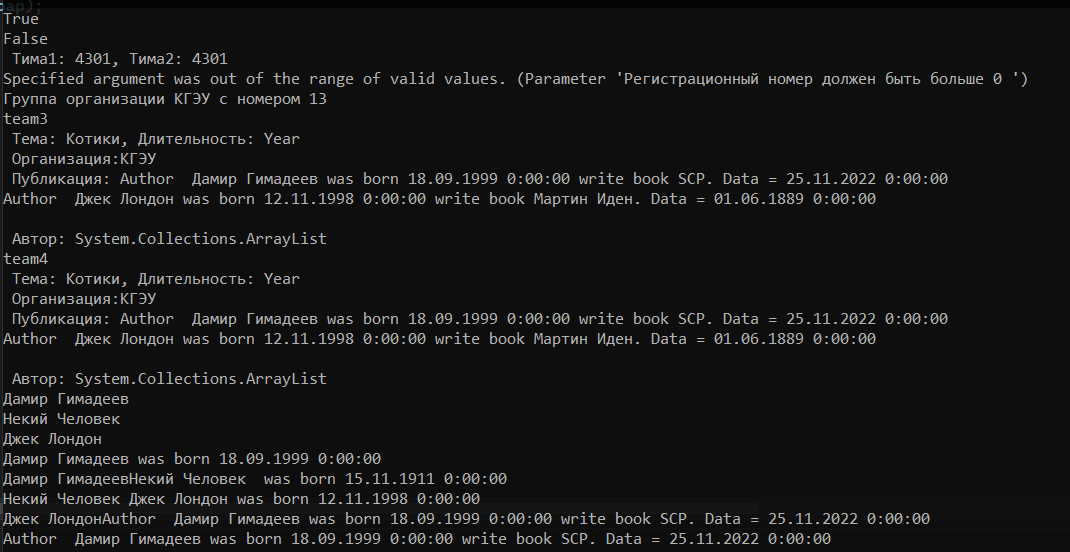
return new Paper(this.Name, this.Author, this.Data);

}

}

}

Результат:

****