**Лабораторная работа 2**

**Наследование. Исключения. Интерфейсы.  Итераторы и блоки итераторов**

**Информация для всех вариантов:**

В классе **Person** из лабораторной работы 1 и в классах, дополнительно  указанных в вариантах, надо

∙ переопределить (override) виртуальный метод bool Equals (object obj); ∙ определить операции == и != ;

∙ переопределить виртуальный метод int GetHashCode();

Реализация виртуального метода bool Equals (object obj) в классе  System.Object определяет равенство объектов как равенство ссылок на  объекты. Некоторые классы из базовой библиотеки BCL переопределяют метод  Equals(). В классе System.String этот метод переопределен так, что равными  считаются строки, которые совпадают посимвольно. Реализация метода  Equals() в структурном типе DateTime равенство объектов DateTime определяет  как равенство значений.

В лабораторной работе требуется переопределить метод Equals так,  чтобы объекты считались равными, если равны все данные объектов. Для  класса Person это означает, что равны даты рождения и посимвольно  совпадают строки с именем и фамилией.

Определение операций == и != должно быть согласовано с  переопределенным методом Equals, т.е. критерии, по которым проверяется  равенство объектов в методе Equals, должны использоваться и при проверке  равенства объектов в операциях == и !=.

Переопределение виртуального метода int GetHashCode() также должно  быть согласовано с операциями == и !=. Виртуальный метод GetHashCode()  используется некоторыми классами базовой библиотеки, например,  коллекциями-словарями. Классы базовой библиотеки, вызывающие метод  GetHashCode() из пользовательского типа, предполагают, что равным объектам  отвечают равные значения хэш-кодов. Поэтому в случае, когда под равенством  объектов понимается совпадение данных (а не ссылок), реализация метода GetHashCode() должна для объектов с совпадающими данными возвращать  равные значения хэш-кодов.

В классах, указанных в вариантах лабораторной работы, требуется  определить метод object DeepCopy() для создания полной копии объекта.  Определенные в некоторых классах базовой библиотеки методы Clone() и  Copy() создают ограниченную (shallow) копию объекта – при копировании  объекта копии создаются только для полей структурных типов, для полей  ссылочных типов копируются только ссылки. В результате в ограниченной  копии объекта поля-ссылки указывают на те же объекты, что и в исходном  объекте.

Метод DeepCopy() должен создать полные копии всех объектов, ссылки  на которые содержат поля типа. После создания полная копия не зависит от  исходного объекта - изменение любого поля или свойства исходного объекта  не должно приводить к изменению копии.

При реализации метода DeepCopy() в классе, который имеет поле типа  System.Collections.ArrayList, следует иметь в виду, что определенные в классе  ArrayList конструктор ArrayList(ICollection) и метод Clone() при создании копии  коллекции, состоящей из элементов ссылочных типов, копируют только ссылки.

Метод DeepCopy() должен создать как копии элементов коллекции  ArrayList, так и полные копии объектов, на которые ссылаются элементы  коллекции. Для типов, содержащих коллекции, реализация метода DeepCopy()  упрощается, если в типах элементов коллекций также определить метод  DeepCopy().

**Вариант 2. Требования к программе**

Определить интерфейс

**interface IRateAndCopy**

**{**

**double Rating { get;}**

**object DeepCopy();**

**}**

Определить новые версии классов **Person**, **Article** и **Magazine** из лабораторной  работы 1. Класс **Magazine** определить как производный от класса **Edition**. В  классы **Article** и **Magazine** добавить реализацию интерфейса IRateAndCopy.

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().

В новой версии класса **Article** дополнительно

∙ определить виртуальный метод object DeepCopy();

∙ реализовать интерфейс IRateAndCopy.

Определить класс **Edition**. Класс **Edition** имеет

∙ защищенное(protected) поле типа string c названием издания; ∙ защищенное поле типа DateTime c датой выхода издания;

∙ защищенное поле типа int с тиражом издания;

В классе **Edition** определить:

∙ конструктор с параметрами типа string, DateTime, int для инициализации  соответствующих полей класса;

∙ конструктор без параметров для инициализации по умолчанию; ∙ свойства c методами get и set для доступа к полям типа;

∙ виртуальный метод object DeepCopy();

∙ свойство типа int с методами get и set для доступа к полю с тиражом  издания; в методе set свойства бросить исключение, если присваиваемое  значение отрицательно. При создании объекта-исключения использовать  один из определенных в библиотеке CLR классов-исключений,  инициализировать объект-исключение с помощью конструктора с  параметром типа string, в сообщении передать информацию о допустимых  значениях свойства.

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

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

∙ перегруженную версию виртуального метода string ToString() для  формирования строки со значениями всех полей класса.

Новая версия класса **Magazine** имеет базовый класс **Edition** и следующие поля: ∙ закрытое поле типа Frequency с информацией о периодичности выхода  журнала;

∙ закрытое поле типа System.Collections.ArrayList со списком редакторов  журнала (объектов типа **Person**).

∙ закрытое поле типа System.Collections.ArrayList, в котором хранится список  статей в журнале (объектов типа **Article**).

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

∙ конструктор с параметрами типа string, Frequency, DateTime, int для  инициализации соответствующих полей класса;

∙ конструктор без параметров для инициализации по умолчанию; ∙ свойство типа double (только с методом get), в котором вычисляется  среднее значение рейтинга статей в журнале;

∙ свойство типа System.Collections.ArrayList для доступа к полю со списком  статей в журнале;

∙ метод void AddArticles (params Article[]) для добавления элементов в список статей в журнале;

∙ перегруженная версия виртуального метода string ToString() для  формирования строки со значениями всех полей класса, включая список  статей и список редакторов;

∙ виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка статей и списка редакторов, но со  значением среднего рейтинга статей в журнале.

Дополнительно в новой версии класса **Magazine** реализовать ∙ свойство типа System.Collections.ArrayList для доступа к списку редакторов  журнала;

∙ метод void AddEditors (params Person[]) для добавления элементов в список редакторов;

∙ перегруженную (override) версию виртуального метода object DeepCopy(); ∙ интерфейс IRateAndCopy;

∙ свойство типа Edition; метод get свойства возвращает объект типа Edition,  данные которого совпадают с данными подобъекта базового класса, метод  set присваивает значения полям из подобъекта базового класса.

В новой версии класса **Magazine** определить

∙ итератор с параметром типа double для перебора статей с рейтингом  больше некоторого заданного значения;

∙ итератор с параметром типа string для перебора статей, в названии  которых есть заданная строка.

В методе **Main()**

1. Создать два объекта типа Edition 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 тиражом издания некорректное  значение, в обработчике исключения вывести сообщение, переданное  через объект-исключение.

3. Создать объект типа Magazine, добавить элементы в списки статей и  редакторов журнала и вывести данные объекта Magazine.

4. Вывести значение свойства типа Edition для объекта типа Magazine.

5. С помощью метода DeepCopy() создать полную копию объекта Magazine.  Изменить данные в исходном объекте Magazine и вывести копию и  исходный объект, полная копия исходного объекта должна остаться без  изменений.

6. С помощью оператора foreach для итератора с параметром типа double вывести список всех статей с рейтингом больше некоторого заданного  значения.

7. С помощью оператора foreach для итератора с параметром типа string вывести список статей, в названии которых есть заданная строка.

**Код программы:**

1. Начало программы

class Program

{

const string SEPARATOR\_START = "-----------------------------------------------------------------";

const string SEPARATOR\_END = "-----------------------------------------------------------------\n\n\n";

static void TestArraysRunTime<T>(T item, int nrow, int ncolumn)

{

/\* Одномерный массив \*/

T[] OneDimArray = new T[nrow \* ncolumn];

/\* Двумерный массив \*/

T[,] TwoDimArray = new T[nrow, ncolumn];

/\* Ступенчатый массив \*/

T[][] StepArray = new T[nrow][];

/\* Заполняем ступенчатый массив другими массивами \*/

for (int i = 0; i < StepArray.Length; i++)

StepArray[i] = new T[ncolumn];

int OneDimTimeStart = Environment.TickCount;

/\* Заполняем одномерный массив \*/

for (int i = 0; i < OneDimArray.Length; i++)

OneDimArray[i] = item;

int OneDimTimeEnd = Environment.TickCount;

Console.WriteLine($"Одномерный массив: {OneDimTimeEnd - OneDimTimeStart}");

int TwoDimTimeStart = Environment.TickCount;

/\* Заполняем двумерный массив \*/

for (int i = 0; i < TwoDimArray.GetLength(0); i++)

for (int j = 0; j < TwoDimArray.GetLength(1); j++)

TwoDimArray[i, j] = item;

int TwoDimTimeEnd = Environment.TickCount;

Console.WriteLine($"Двумерный массив: {TwoDimTimeEnd - TwoDimTimeStart}");

int StepTimeStart = Environment.TickCount;

/\* Заполняем ступенчатый массив \*/

for (int i = 0; i < StepArray.Length; i++)

for (int j = 0; j < StepArray[i].Length; j++)

StepArray[i][j] = item;

int StepTimeEnd = Environment.TickCount;

Console.WriteLine($"Ступенчатый массив: {StepTimeEnd - StepTimeStart}");

}

static void StartLab\_2()

{

Console.WriteLine("Задание №1");

Console.WriteLine(SEPARATOR\_START);

/\*

\* Создать два объекта типа Edition с совпадающими данными и проверить,

\* что ссылки на объекты не равны, а объекты равны, вывести значения хэш-

\* кодов для объектов.

\*/

var first = new Edition("First", new DateTime(), 10);

var second = new Edition("First", new DateTime(), 10);

Console.WriteLine($"Проверка ссылок на объекты (first == second): {first == second}");

Console.WriteLine($"Equals: {first.Equals(second)}");

Console.WriteLine($"First Hash: {first.GetHashCode()}; Second Hash: {second.GetHashCode()}");

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №2");

Console.WriteLine(SEPARATOR\_START);

/\*

\* В блоке try/catch присвоить свойству с тиражом издания некорректное

\* значение, в обработчике исключения вывести сообщение, переданное

\* через объект-исключение.

\*/

try

{

first.EditionCirculation = -1;

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №3");

Console.WriteLine(SEPARATOR\_START);

/\*

\* Создать объект типа Magazine, добавить элементы в списки статей и

\* редакторов журнала и вывести данные объекта Magazine.

\*/

var magazine = new Magazine();

magazine.AddArticles(new Article(new Person("Петров", "Иванович", new DateTime()), "Статья №2342", 23));

magazine.AddEditors(new Person("Петр", "Васильевич", new DateTime()));

Console.WriteLine(magazine);

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №4");

Console.WriteLine(SEPARATOR\_START);

/\*

\* Вывести значение свойства типа Edition для объекта типа Magazine.

\*/

Console.WriteLine(magazine.Edition);

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №5");

Console.WriteLine(SEPARATOR\_START);

/\*

\* С помощью метода DeepCopy() создать полную копию объекта Magazine.

\* Изменить данные в исходном объекте Magazine и вывести копию и

\* исходный объект, полная копия исходного объекта должна остаться без

\* изменений.

\*/

var newMagazine = magazine.DeepCopy() as Magazine;

newMagazine.Name = "test12345";

Console.WriteLine(magazine);

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №6");

Console.WriteLine(SEPARATOR\_START);

/\*

\* С помощью оператора foreach для итератора с параметром типа double

\* вывести список всех статей с рейтингом больше некоторого заданного

\* значения.

\*/

foreach (Article article in magazine.GetArticlesForRating(10))

{

Console.WriteLine(article + "\n");

}

Console.WriteLine(SEPARATOR\_END);

Console.WriteLine("Задание №7");

Console.WriteLine(SEPARATOR\_START);

/\*

\* С помощью оператора foreach для итератора с параметром типа string

\* вывести список статей, в названии которых есть заданная строка.

\*/

foreach (Article article in magazine.GetArticlesForSubstr("ест"))

{

Console.WriteLine(article + "\n");

}

Console.WriteLine(SEPARATOR\_END);

}

static void Main(string[] args)

{

StartLab\_2();

Console.ReadLine();

}

}

1. Класс Person – для описания человека

class Person

{

/\* Имя \*/

private string name;

/\* Фамилия \*/

private string secondName;

/\* Дата рождения \*/

private DateTime dateBirth;

/\*\* Имя \*/

public string Name

{

get { return name; }

}

/\* Фамилия \*/

public string SecondName

{

get { return secondName; }

}

/\* Дата рождения \*/

public DateTime DateBirth

{

get { return dateBirth; }

}

public int DateBirthYear

{

get { return dateBirth.Year; }

set { dateBirth.AddYears(value - dateBirth.Year); }

}

public Person(string name, string secondName, DateTime dateBirth)

{

this.name = name;

this.secondName = secondName;

this.dateBirth = dateBirth;

}

public Person()

{

name = "Егор";

secondName = "Дмитриев";

dateBirth = new DateTime(1999, 1, 1);

}

public override string ToString()

{

return $"Имя: {Name}\nФамилия: {SecondName}\nДата рождения: {DateBirth}";

}

public virtual string ToShortString()

{

return $"Имя: {Name}\nФамилия: {SecondName}";

}

public override bool Equals(object obj)

{

var person = obj as Person;

return name == person.name && secondName == person.secondName && dateBirth == person.dateBirth;

}

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

{

return obj1.Equals(obj2);

}

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

{

return !obj1.Equals(obj2);

}

public override int GetHashCode()

{

return ToString().GetHashCode();

}

public Person DeepCopy()

{

return new Person(Name, SecondName, DateBirth);

}

}

1. Класс Article – для описания статьи

class Article: IRateAndCopy

{

/\* Автор статьи \*/

public Person Person { get; set; }

/\* Название статьи \*/

public string Name { get; set; }

/\* Рейтинг статьи \*/

public double Rating { get; set; }

public Article(Person person, string name, double rating)

{

Person = person;

Name = name;

Rating = rating;

}

public Article()

{

Person = new Person();

Name = "Тестовая статья";

Rating = 2.1;

}

public override string ToString()

{

return $"Название статьи: {Name}\nРейтинг статьи: {Rating}\nАвтор статьи: {Person}";

}

public object DeepCopy()

{

return new Article(Person.DeepCopy(), Name, Rating);

}

}

1. Класс Edition – для описания издания

class Edition

{

/\* Название издания \*/

protected string name;

/\* Дата выхода издания \*/

protected DateTime releaseDate;

/\* Тираж издания \*/

protected int editionCirculation;

/\* Название издания \*/

public string Name {

get { return name; }

set { name = value; }

}

/\* Дата выхода издания \*/

public DateTime ReleaseDate {

get { return releaseDate; }

set { releaseDate = value; }

}

/\* Тираж издания \*/

public int EditionCirculation {

get { return editionCirculation; }

set {

if(value < 0)

{

Exception exception = new Exception("EditionCirculation < 0");

throw exception;

}

editionCirculation = value;

}

}

public Edition(string name, DateTime releaseDate, int editionCirculation)

{

this.name = name;

this.releaseDate = releaseDate;

this.editionCirculation = editionCirculation;

}

public Edition()

{

name = "Тестовое издание";

releaseDate = new DateTime();

editionCirculation = 99;

}

public virtual object DeepCopy()

{

return new Edition(Name, releaseDate, editionCirculation);

}

public override bool Equals(object obj)

{

var edition = obj as Edition;

return name == edition.Name && releaseDate == edition.ReleaseDate && editionCirculation == edition.editionCirculation;

}

public override string ToString()

{

return $"Название издания: {name}\n" +

$"Дата выхода издания: {releaseDate}\n" +

$"Тираж издания: {editionCirculation}";

}

public override int GetHashCode()

{

return ToString().GetHashCode();

}

}

1. Класс Magazine – для описания журнала

class Magazine : Edition, IRateAndCopy

{

/\* Переодичность выхода журнала \*/

private EnumFrequency qutputFrequency;

/\* Редакторы журнала \*/

private ArrayList editors;

/\* Cписок статей в журнале \*/

private ArrayList articles;

/\* Переодичность выхода журнала \*/

public EnumFrequency QutputFrequency

{

get { return qutputFrequency; }

set { qutputFrequency = value; }

}

/\* Статьи в журнале \*/

public ArrayList Articles

{

get { return articles; }

set { articles = value; }

}

/\* Редакторы журнала \*/

public ArrayList Editors

{

get { return editors; }

set { editors = value; }

}

/\* Cреднее значение рейтинга в списке статей \*/

public double AvgRatingOfArticles

{

get

{

var \_articles = from Article article in Articles select article;

return \_articles.Sum(a => a.Rating) / \_articles.Count();

}

}

public bool this[EnumFrequency frequency]

{

get => QutputFrequency == frequency;

}

/\* Получаем строковое представление частоты выхода журнала \*/

private string QutputFrequencyString

{

get

{

switch (qutputFrequency)

{

case EnumFrequency.Weekly:

return "Ежедневно";

case EnumFrequency.Monthly:

return "Ежемесячно";

case EnumFrequency.Yearly:

return "Ежегодно";

default:

return "";

}

}

}

/\* Строка с информацией о статьях \*/

private string ArticlesInfoString

{

get {

var \_articles = from Article article in Articles select article;

return \_articles.Select((article) => article.ToString()).Aggregate((current, next) => current + "\n\n" + next);

}

}

/\* Строка с информацией о редакторах \*/

private string EditorsInfoString

{

get

{

var \_editors = from Person editor in Editors select editor;

return \_editors.Select((editor) => editor.ToString()).Aggregate((current, next) => current + "\n\n" + next);

}

}

/\* Рейтинг всех статей \*/

public double Rating

{

get

{

var \_articles = from Article article in Articles select article;

return \_articles.Sum((article) => article.Rating);

}

}

public Edition Edition

{

get

{

return new Edition(Name, ReleaseDate, EditionCirculation);

}

set

{

Name = value.Name;

ReleaseDate = value.ReleaseDate;

EditionCirculation = value.EditionCirculation;

}

}

public IEnumerable<Article> GetArticlesForRating(double min = 0) {

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

{

var article = Articles[i] as Article;

if(article.Rating > min)

{

yield return article;

}

}

}

public IEnumerable<Article> GetArticlesForSubstr(string substring)

{

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

{

var article = Articles[i] as Article;

if (article.Name.Contains(substring))

{

yield return article;

}

}

}

public Magazine(string name, EnumFrequency qutputFrequency, DateTime releaseDate, int circulation, ArrayList articles, ArrayList editors)

{

this.name = name;

this.qutputFrequency = qutputFrequency;

this.releaseDate = releaseDate;

editionCirculation = circulation;

this.articles = articles;

this.editors = editors;

}

public Magazine()

{

name = "Тестовый журнал";

qutputFrequency = EnumFrequency.Monthly;

releaseDate = DateTime.MinValue;

editionCirculation = 0;

articles = new ArrayList() { new Article(), new Article() };

editors = new ArrayList() { new Person(), new Person() };

}

/\* Добавляем статьи в журнал \*/

public void AddArticles(params Article[] newArticles)

{

var \_articles = from Article article in Articles select article;

var concatValues = \_articles.Concat(newArticles).ToArray();

var result = new ArrayList();

for (int i = 0; i < concatValues.Length; i++)

result.Add(concatValues[i]);

articles = result;

}

/\* Добавляем редакторов в журнал \*/

public void AddEditors(params Person[] newEditors)

{

var \_editors = from Person editor in Editors select editor;

var concatValues = \_editors.Concat(newEditors).ToArray();

var result = new ArrayList();

for (int i = 0; i < concatValues.Length; i++)

result.Add(concatValues[i]);

editors = result;

}

public override string ToString()

{

return $"Название журнала: {name}\n" +

$"Периодичность выхода журнала: {QutputFrequencyString}\n" +

$"Дата выхода журнала: {releaseDate}\n" +

$"Тираж: {editionCirculation}\n\n" +

$"Статьи:\n" +

$"{ArticlesInfoString}\n\n" +

$"Редакторы:\n" +

$"{EditorsInfoString}";

}

public virtual string ToShortString()

{

return $"Название журнала: {name}\n" +

$"Периодичность выхода журнала: {QutputFrequencyString}\n" +

$"Дата выхода журнала: {releaseDate}\n" +

$"Тираж: {editionCirculation}\n" +

$"Cреднее значение рейтинга статей: {AvgRatingOfArticles}";

}

public override object DeepCopy()

{

return new Magazine(Name, QutputFrequency, ReleaseDate, EditionCirculation, Articles, Editors);

}

}

**Консольный вывод:**





