ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»



Кафедра

«Программное обеспечение информационных технологий»

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ   
И ПРОЕКТИРОВАНИЕ**

Могилев 2020

**СОДЕРЖАНИЕ**

[1. ЛАБОРАТОРНАЯ РАБОТА № 1. ПРОСТЕЙШИЕ КЛАССЫ. ИНКАПСУЛЯЦИЯ. СВОЙСТВА 4](#_Toc35888357)

[1.1. Классы: основные понятия 4](#_Toc35888358)

[1.2. Данные: поля и константы 4](#_Toc35888359)

[1.3. Методы 5](#_Toc35888360)

[1.4. Конструкторы 6](#_Toc35888361)

[1.5. Свойства 7](#_Toc35888362)

[1.6. Индивидуальное задание 9](#_Toc35888363)

[Варианты заданий: 9](#_Toc35888364)

[2. ЛАБОРАТОРНАЯ РАБОТА 2. МАССИВЫ. ИНДЕКСАТОРЫ 11](#_Toc35888365)

[2.1. Структура массива в С# 11](#_Toc35888366)

[2.2. Цикл foreach 11](#_Toc35888367)

[2.3. Индексаторы 12](#_Toc35888368)

[2.4. Индивидуальные задания 14](#_Toc35888369)

[Варианты заданий: 14](#_Toc35888370)

[3. ЛАБОРАТОРНАЯ РАБОТА №3. ПЕРЕГРУЗКА МЕТОДОВ И ОПЕРАЦИЙ 16](#_Toc35888371)

[3.1. Перегрузка методов 16](#_Toc35888372)

[3.2. Методы с переменным количеством аргументов 17](#_Toc35888373)

[3.3. Перегрузка операций 17](#_Toc35888374)

[3.4. Индивидуальные задания 20](#_Toc35888375)

[Варианты заданий: 21](#_Toc35888376)

[4. ЛАБОРАТОРНАЯ РАБОТА №4. ИЕРАРХИИ КЛАССОВ. НАСЛЕДОВАНИЕ 23](#_Toc35888377)

[4.1. Наследование 23](#_Toc35888378)

[4.2. Вызов конструкторов базового класса 24](#_Toc35888379)

[4.3. Наследование и сокрытие имен 25](#_Toc35888380)

[4.4. Индивидуальные задания 26](#_Toc35888381)

[Варианты заданий: 27](#_Toc35888382)

[5. ЛАБОРАТОРНАЯ РАБОТА №5. ВИРТУАЛЬНЫЕ МЕТОДЫ И ПОЛИМОРФИЗМ 29](#_Toc35888383)

[5.1. Виртуальные методы 29](#_Toc35888384)

[5.2. Абстрактные классы 30](#_Toc35888385)

[5.3. Индивидуальные задания 31](#_Toc35888386)

[6. ЛАБОРАТОРНАЯ РАБОТА №6. ИНТЕРФЕЙСЫ И КОЛЛЕКЦИИ 33](#_Toc35888387)

[6.1. Понятие интерфейса 33](#_Toc35888388)

[6.2. Наследование интерфейсов 35](#_Toc35888389)

[6.3. Стандартные интерфейсы среды .NET Framework 36](#_Toc35888390)

[6.4. Класс ArrayList 36](#_Toc35888391)

[6.5. Индивидуальные задания 39](#_Toc35888392)

[Варианты заданий: 39](#_Toc35888393)

[7. ЛАБОРАТОРНАЯ РАБОТА № 7. СИМВОЛЫ И СТРОКИ 44](#_Toc35888394)

[7.1. Строковый тип string 44](#_Toc35888395)

[7.2. Строковый тип StringBuilder 50](#_Toc35888396)

[7.3. Сравнение классов String и StringBuilder 55](#_Toc35888397)

[7.4. Индивидуальные задания 56](#_Toc35888398)

[8. ЛАБОРАТОРНАЯ РАБОТА № 8. ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ 58](#_Toc35888399)

[8.1. Исключительные ситуации 58](#_Toc35888400)

[8.2. Индивидуальные задания 63](#_Toc35888401)

[9. ЛАБОРАТОРНАЯ РАБОТА № 9. ДЕЛЕГАТЫ. СОБЫТИЯ 65](#_Toc35888402)

[9.1. Делегаты 65](#_Toc35888403)

[9.2. События 66](#_Toc35888404)

[9.3. Индивидуальные задания 69](#_Toc35888405)

[10. ЛАБОРАТОРНАЯ РАБОТА № 10. ФАЙЛЫ 71](#_Toc35888407)

[10.1. Файловый ввод-вывод 71](#_Toc35888408)

[10.2. Индивидуальные задания 73](#_Toc35888409)

[11. ЛАБОРАТОРНАЯ РАБОТА № 11. РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЙ 74](#_Toc35888410)

[11.1. Разработка графического интерфейса для Windows-приложений с помощью технологии Windows Forms 74](#_Toc35888411)

[11.2. Индивидуальные задания 75](#_Toc35888412)

[ЛИТЕРАТУРА 81](#_Toc35888413)

# ЛАБОРАТОРНАЯ РАБОТА № 1. ПРОСТЕЙШИЕ КЛАССЫ. ИНКАПСУЛЯЦИЯ. СВОЙСТВА. 2 – часа

Цель работы – усвоение понятий класса, объекта, определения членов класса, овладение приемами разработки простейших программ в объектно-ориентированном стиле программирования.

## Классы: основные понятия

Класс является типом данных, определяемым пользователем; он описывает признаки состояния и поведение множества схожих объектов окружающей действительности.

Элементами класса являются данные и методы, предназначенные для их обработки.

Описание класса имеет вид:

[атрибуты][спецификаторы] class<имя\_класса> [: предки]

{ тело\_класса }

Обязательными являются только ключевое слово class, а также имя и тело класса. Имя класса задается программистом по общим правилам С#. Тело класса – это список описаний его элементов, заключенный в фигурные скобки. Спецификаторы определяют свойства класса, а также доступность класса для других элементов программы. Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. Для классов, которые описываются в пространстве имен непосредственно, допускаются только два спецификатора: public и internal. По умолчанию подразумевается спецификатор internal.

Класс используется для создания объектов в программе. Объекты также называются экземплярами класса. Объекты могут создаться явным или неявным образом, то есть либо программистом, либо системой.

Класс относится к ссылочным типам данных, память под которые выделяется в хипе. Для каждого объекта при его создании в памяти выделяется отдельная область, в которой хранятся его данные.

В классе могут присутствовать элементы, которые существуют в единственном экземпляре для всех объектов класса – статические элементы.

Функциональные элементы класса не тиражируются, то есть всегда хранятся в единственном экземпляре. Для работы со статическими данными класса используются статические методы, для работы с данными экземпляра – методы экземпляра, или просто методы.

## Данные: поля и константы

Данные, содержащиеся в классе, могут быть переменными или константами. Переменные, описанные в классе, называются полями класса. Синтаксис описания элемента данных:

[атрибуты][спецификаторы][const] <тип> <имя> [= <начальное значение>]

По умолчанию элементы класса считаются закрытыми (private). Все методы класса имеют непосредственный доступ к его закрытым полям.

Все поля сначала автоматически инициализируются нулем соответствующего типа. После этого полю присваивается значение, заданное при его явной инициализации. Задание начальных значений для статических полей выполняется при инициализации класса, а обычных – при создании экземпляра.

Поля, описанные со спецификатором static, а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются через имя класса. Обращение к полю класса выполняется с помощью операции доступа – точки:

для обычных полей: <имя экземпляра>.<имя поля>

для статических: <имя класса>.<имя поля>

**Пример 1**. Класс Demo, содержащий поля и константу.

using System;

namespace ConsoleApplication {

class Demo {

public int a = 1; //поле данных

public const double с = 1.66; //константа

public static string s = "Demo"; //статическое. поле класса

private double y; //закрытое поле данных

}

class Class1 {

static void Main(){

Demo x = new Demo(); //создание экземпляра класса Demo Console.WrteLine(x.a); //обращение к полю класса Console.WriteLine(Demo.с); //обращение к константе Console.WriteLine(Demo.s); //обращение к статическому полю

}}}

Поле y вывести на экран аналогичным образом нельзя – оно является закрытым, т.е. недоступно из класса Class1. Поскольку значение этому полю явным образом не присвоено, компилятор присваивает ему значение ноль.

## Методы

Методы класса имеют непосредственный доступ к его полям. Метод, описанный со спецификатором static, должен обращаться только к статическим полям класса. Статический метод вызывается через имя класса, а обычный – через имя экземпляра.

Например, для статического метода класса Class1 с сигнатурой

static void Poisk(int a, ref int b, out int с);

вызов может быть следующим: Poisk(1, ref x, out y);

Для метода с сигнатурой void Out()необходимо создать объект:

Class1 myob = new Class1();

а затем вызвать метод myob.Out();

## Конструкторы

Конструктор предназначен для инициализации полей объекта. Он вызывается автоматически при создании объекта класса с помощью операции new. Формат записи конструктора:

[спецификатор] <имя\_класса>()

{тело конструктора}

Обычно в качестве элемента спецификаториспользуется модификатор доступа public, поскольку конструкторы, как правило, вызываются вне их класса.

Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации. Конструктор, вызываемый без параметров, называется конструктором по умолчанию. Если в классе не указан ни один конструктор или какие-то поля не были инициализированы, полям значимых типов присваивается нуль соответствующего типа, полям ссылочных типов – значение null.

Создание объекта выполняется операцией

<имя\_переменной\_типа\_класса> = new <имя\_класса>();

Имя класса вместе со следующей за ним парой круглых скобок – это конструктор класса. Если в классе конструктор не определен явным образом, оператор new будет использовать конструктор по умолчанию, который предоставляется средствами языка С#.

**Пример 2**. В программе создаются два объекта с различными значениями полей с помощью конструкторов по умолчанию и с параметрами.

using System;

namespace ConsoleApplication{

class Demo{

public int a;

public double y;

//конструктор с параметром

public Demo(int a,double у1){

this.a = a; //полю а присваив. значен. парам. а

у = y1;

}

// конструктор по умолчанию

public Demo(){

a = 276;

у = 5.5;

}}

class Classl{

static void Main(){

//вызов конструктора с параметрами

Demo ob1 = new Demo(300, 0.002);

Console.WriteLine(ob1.a); //результат: 300

Console.WriteLine(ob1.y); //результат:0,002

//вызов конструктора без параметров

Demo ob2 = new Demo();

Console.Writeline(ob2.a); //результат: 276

Console.WriteLine(ob2.y); //результат: 5.5

}}}

## Свойства

Свойство – это специальный тип членов класса, который предназначен для организации доступа к закрытым полям класса и определяет методы его получения и установки.

Формат записи свойства:

[атрибуты][спецификаторы] <тип> <имя\_свойства>

{[ get {<код аксессора чтения поля> //код доступа}]

[ set {<код аксессора записи поля> // код доступа}]}

Значения спецификаторов для свойств и методов аналогичны. Чаще всего свойства объявляются как открытые (со спецификатором public). После определения свойства любое использование его имени означает вызов соответствующего аксессора. Код доступапредставляет собой блоки операторов, которые выполняются при получении (get) или установке (set) свойства. Может отсутствовать либо частьget, либо set, но не обе одновременно. Если отсутствует часть set, свойство доступно только для чтения, если отсутствует часть get, свойство доступно только для записи.

Аксессор set автоматически принимает параметр с именем value, который содержит значение, присваиваемое свойству.

Имя свойства можно использовать в выражениях и инструкциях присваивания подобно обычной переменной, хотя в действительности здесь будут автоматически вызываться get- и set-аксессоры.

**Пример 3.** Использование свойства. Это свойство позволяет присваивать полю только положительные числа.

using System;

class SimpProp {

private int prop; //это закрытое поле управляется

//свойством Prop

public SimpProp() {

prop = 0 ;

}

public int Prop{ /\* Это свойство поддерживает доступ к закрытому полю prop. Оно позволяет присваивать ему только положительные числа\*/

get {return prop;} //способ получения свойства

set {

if(value >= 0)

prop = value;

} //способ установки свойства

}}

// Демонстрация использования свойств

class PropertyDemo {

public static void Main(){

SimpProp ob = new SimpProp ();

Console.WriteLine("Исходное значение b.myp:"+ob.Prop);

ob.Prop = 100; // вызывается метод установки свойства

Console.WriteLine("Значение ob.Prop: " + ob.Prop);

Console.WriteLine("Присваиваем -10 свойству ob.Prop");

ob.Prop = -10;

Console.WriteLine("Значение ob.Prop: " + ob.Prop);

}}

**Контрольные вопросы**

1. Что такое класс и объект класса?
2. Какое отличие между описанием класса и объектом класса?
3. Что объявляется в классе?
4. Что называется полями класса?
5. Какие элементы языка C# могут относиться к функциям-членам класса?
6. Какая общая форма описания класса?
7. Какие существуют типы (модификаторы) доступа к членам класса?
8. Может ли отсутствовать тип (модификатор) доступа при описании членов класса?
9. Приведите примеры описания простейших классов, содержащих только данные.
10. Как создать объект (экземпляр класса)?
11. К каким типам данных принадлежат классы: к типам значения или к ссылочным типам?
12. Какие особенности использования ключевого слова this в классе?
13. Что такое статические члены класса?
14. Как используются статические и экземплярные методы?
15. Что представляет собой конструктор? Для чего он используется?
16. Какие бывают конструкторы?
17. Для чего предназначены свойства?
18. Какие бывают свойства?

## Индивидуальное задание

**Задание 1**. Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Указания. Разрабатываемый класс должен содержать следующие элементы: скрытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), свойства, метод вывода полей и указанный в таблице метод.

Составьте тестирующую программу с выдачей результатов. В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта.

### Варианты заданий:

|  |  |  |
| --- | --- | --- |
| № | **Класс** | **Метод** |
| 1. | Сотрудник  (поля: имя, р – минимальная зарплата) | Доход: k \*p, где k – по-  вышающий коэффициент |
| 2. | Квартира  (поля: номер, стоимость 1м2, площадь) | Стоимость квартиры |
| 3. | Агрополе (поля: название, вес r посеянных семян на единицу площади, S – площадь в га) | Урожай:  k\*r, где k- коэффициент, зависящий от культуры |
| 4. | Стол  (поля: название, площадь S в см) | Стоимость:  S2/3+500000 |
| 5. | Автобус (поля: количество пассажиров,  стоимость билета) | Выручка |
| 6. | Транспортное средство (поля: название,  расстояние, цена за 1 км) | Стоимость проезда |
| 7. | Пособие (поля: повышающий коэффициент  k, минимальное пособие r) | Размер пособия:  k\*r |
| 8. | Телефон (поля: марка, количество функций –  k) | Стоимость: 40ln(k) |
| 9. | Автомобиль  (поля: марка, расход горючего на 100км N, расстояние R в км) | Объем горючего: N\*R |
| 10. | Одежда  (поле – модель, ширина ткани H м, норма расхода – L м) | Расход ткани: (2+H)\*L |
| 11. | Постройка  (поля: название, высота здания V м) | Высота фундамента:  0,03\*V |
| 12. | Аудитория (поля: номер, площадь S м2) | Количество мест:  S/1,2 |
| 13. | Книга  (поля: название книги; количество страниц N; стоимость одной страницы, Ст руб) | Стоимость книги:  Ст \* ln(N) |
|  | Фрукты  (поля: вес фруктов B, кг; выход сока с одного килограмма фруктов С, л ) | Выход сока:  (В\*0,1 + ln(С)) |
|  | Яблоня  (поля: название, продолжительность жизни П) | Урожайность:  25\*ln(П) |

## ЛАБОРАТОРНАЯ РАБОТА 2. МАССИВЫ. ИНДЕКСАТОРЫ. 2 – часа

Цель работы – усвоение основных приемов создания классов с полями типа - массив; формированию знаний о возможности использования индексаторов для доступа к элементам закрытых массивов.

## Структура массива в С#

Массив  это коллекция переменных одинакового типа, обращение к которым происходит с использованием общего для всех имени. Коллекция  это группа объектов. С# определяет несколько типов коллекций, и одним из них является массив.

Кроме рассмотренных в [11], С# позволяет создавать двумерный массив специального типа, именуемый рваным*,* или с рваными краями, или ступенчатым. У такого массива строки могут иметь различную длину.

Рваные массивы объявляются с помощью наборов квадратных скобок, обозначающих размерности массива. Например:

<тип>[][] <имя> = new <тип>[размер][];

Здесь элемент размерозначает количество строк в массиве. Для самих строк память выделяется индивидуально, что позволяет строкам иметь разную длину.

**Пример 1**.

int [][] gg = new int [3][];

gg[0] = new int [4];

gg[1] = new int [3];

gg[2] = new int [5];

Доступ к элементу осуществляется посредством задания индекса внутри собственного набора квадратных скобок. Например:

gg[2][1] = 10;

Поскольку рваные массивы  по сути массивы массивов, то "внутренние" массивы (строки) могут иметь разный тип.

Можно присваивать одной ссылочной переменной массива другую. При этом не делается копия массива и не копируется содержимое одного массива в другой, а просто изменяете объект, на который ссылается эта переменная.

## Цикл foreach

Цикл foreach используется для опроса элементов коллекции. Формат записи цикла имеет вид:

foreach (<тип> <имя\_переменной> in <коллекция>)

<тело цикла;>

Здесь элементы тип и имя\_переменной задают тип и имя итерационной переменной, которая при выполнении цикла fоreach будет последовательно получать значения элементов из коллекции.

Элемент коллекцияслужит для указания опрашиваемой коллекции. Таким образом, элемент типдолжен совпадать (или быть совместимым) с базовым типом массива. С помощью foreach, невозможно изменить содержимое коллекции.

**Пример 2.** Создать массив для хранения целых чисел и присвоить его элементам начальные значения. Затем вывести элементы массива, и вычислить их сумму.

using System;

class ForeachDemo {// Использование цикла foreach.

public static void Main(){

int sum = 0;

int[] n = new int[10];

for(int i = 0; i < 10; i++)

n[i] = i; //Присваив. элементам массива n значения

//Используем цикл foreach для вывода

//значений элементов массива и их суммирования,

foreach(int x in n){

Console.WriteLine("Значение элемента равно: "+х);

sum += х;

}

Console.WriteLine("Сумма равна: " + sum);

}}

Цикл foreach работает и с многомерными массивами. В этом случае он возвращает элементы в порядке следования строк: от первой до последней.

## Индексаторы

Индексатор предназначен для обращения к скрытому полю класса, представляющему собой массив, используя имя объекта и номер элемента массива в квадратных скобках.

Синтаксис индексатора:

<атрибуты><спецификаторы> <тип> this[<спиcок\_параметров>]

{

get код\_доступа

set код\_доступа

}

Спецификаторы аналогичны спецификаторам свойств и методов. Здесь тип– базовый тип индексатора. Он соответствует базовому типу массива. Код доступапредставляет собой блоки операторов, которые выполняются при получении (get) или установке значения (set) элемента массива. Может отсутствовать либо часть get, либо set. Если отсутствует часть set, индексатор доступен только для чтения, если отсутствует часть get, индексатор доступен только для записи. Список параметровсодержит одно или несколько описаний индексов, по которым выполняется доступ к элементу.

**Пример 3.** Создайте класс с закрытым массивом, элементы которого должны находиться в диапазоне [0, 100]. Кроме того, при доступе к элементу проверяется, не вышел ли индекс за допустимые границы. Используйте индексатор.

using System;

namespace ConsoleApplicationl{

class SafeArray{

public bool error =false; //открытый признак ошибки

private int[] a; //закрытый массив

private int length; //закрытая размерность

public SafeArray(int size){ // конструктор класса

a = new int[size];

length = size;

}

public int Length {// свойство - размерность

get { return length; }

}

public int this[int i]{ // индексатор

get{

if (i >= 0 && i < length) return a[i];

else { error = true; return 0; }

}

set{

if(i>=0 && i<length && value>=0&& value <= 100 )

a[i] = value;

else error = true;

}

}}

class Classl{

static void Main(){

int n = 100;

SafeArray sa = new SafeArray(n);//создание объекта

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

sa[i] = i \* 2; // использование индексатора

Console.Write(sa[i]);

}

if (sa.error) Console.Write ("Были ошибки!");

}}}

Язык С# допускает использование многомерных индексаторов. Они описываются аналогично обычным и применяются в основном для контроля за занесением данных в многомерные массивы и выборке данных из многомерных массивов, оформленных в виде классов.

Например, если внутри класса объявлен двумерный массив int[,] a;, то заголовок индексатора должен иметь вид:

public int this[int i, int j]

**Контрольные вопросы:**

1. Что понимается под массивом?
2. В каких случаях целесообразно описывать двумерный массив с помощью одномерных?
3. Для чего предназначен цикл foreach? Можно ли использовать этот цикл для ввода элементов массива?
4. Для чего предназначены инексаторы?
5. Как определяется базовый тип индексатора?
6. Что записывается в качестве имени индексатора?
7. Что содержит список параметров индексатора?
8. Можно ли в классе объявить более одного индексатора? Если да, то при каком условии.
9. Может ли аргумент индексатора иметь тип, отличный от int ?

## Индивидуальные задания

**Задание 1.**Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Класс должен содержать закрытое поле двумерного массива, конструктор без параметров и три конструктора с параметрами, свойства, индексаторы, методы (ввода, вывода, обработки массива). Обработку массива в соответствии с заданием варианта осуществлять в одном методе, исходные данные и результаты работы метода передавать параметрами. В программе должны проверятся все элементы разработанного класса.

### Варианты заданий:

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Тип массива** | **Размер- ность** | **Метод** |
| 1. | вещественный | NxN | Проверить является ли матрица симметричной. |
| 2. | символьный из цифр | NxM | Определить сумму четных и нечетных цифр в каждой строке. |
| 3. | целочисленный | NxM | Найти суммы элементов, стоящих до и после  введенного с клавиатуры значения |
| 4. | вещественный | NxM | Найти количество столбцов, начинающихся с  отрицательного числа. |
| 5. | целочисленный | NxN | Проверить, является ли матрицы верхней тре-  угольной. |
| 6. | вещественный | NxN | Преобразовать матрицу следующим образом: каждый элемент строки разделить на макси- мальный элемент этой строки, если он не  равна 0. В противном случае элементы строки оставить без изменений. |
| 7. | строковый | NxN | Подсчитать, сколько элементов массива, на-  чинаются с прописной буквы. |
| 8. | целочисленный | Nx2 | Массив состоит из номеров зачетной книж- ки и годов рождений студентов. Вычислить сколько лет студенту и вывести на экран информацию вида «номер зачетной книжки  – количество лет». |
| 9. | целочисленный | NxN | Установить, является ли упорядоченной за-  данная своим номером строка. |
| 10. | строковый | NxM | Проверить, является ли заданный индексами элемент массива палиндромом. Палиндром принимает одно и то же значение при чтении  его как справа налево, так и слева направо. |
| 11. | целочисленный | NxM | Найти элемент в массиве по заданному значе-  нию. (его индексы) |
| 12. | вещественный | NxM | Найти в заданном своим номером столбце  максимальный по модулю элемент. |

# ЛАБОРАТОРНАЯ РАБОТА №3. ПЕРЕГРУЗКА МЕТОДОВ И ОПЕРАЦИЙ. 4 – часа

Цель работы: освоить приемы создания операций класса и их использование.

## Перегрузка методов

Использование нескольких методов с одним и тем же именем, но различными типами параметров и их способами передачи называется перегрузкой методов.

Компилятор определяет, какой именно метод требуется вызвать, по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого методом значения в разрешении не участвует. Допустим, имеется четыре варианта метода, определяющего наибольшее значение:

// Возвращает наибольшее из двух целых:

int max( int a, int b )

// Возвращает наибольшее из трех целых:

int max( int a, int b, int с )

// Возвращает наибольшее из первого параметра и длины второго:

int max ( int a, string b )

// Возвращает наибольшее из второго параметра и длины первого:

int max ( string b, int a )

Console.WriteLine( max( 1, 2 ) );

Console.WriteLine( max( 1, 2, 3 ) );

Console.WriteLine( max( 1, "2") );

Console.WriteLine( max("1", 2 ) );

При вызове метода max компилятор выбирает вариант метода, соответствующий типу передаваемых в метод аргументов (в приведенном примере будут последовательно вызваны все четыре варианта метода).

Если точного соответствия не найдено, выполняются неявные преобразования типов в соответствии с общими правилами. Если преобразование невозможно, выдается сообщение об ошибке. Если соответствие на одном и том же этапе может быть получено более чем одним способом, выбирается «лучший» из вариантов, то есть вариант, содержащий меньшие количество и длину преобразований. Если существует несколько вариантов, из которых невозможно выбрать лучший, выдается сообщение об ошибке.

Например, методы, заголовки которых приведены ниже, имеют различные сигнатуры и считаются перегруженными:

int max( int a, int b )

int max( int a, ref int b )

Перегрузка методов является проявлением полиморфизма*,* одно го из основных свойств ООП.

## Методы с переменным количеством аргументов

Язык С# предоставляет возможность создать метод, в который можно передавать разное количество аргументов. Для этого используется ключевое слово params. Параметр, помеченный этим ключевым словом, может быть только один и размещается в списке параметров последним и обозначает массив заданного типа неопределенной длины. Соответствующие ему аргументы должны иметь типы, для которых возможно неявное преобразование к базовому типу массива. Например:

public int Calculate( int a, params int[] d ) ...

В этот метод можно передать три и более параметров. Внутри метода к параметрам, начиная с третьего, обращаются как к обычным элементам массива. Количество элементов массива получают с помощью его свойства Length.

int[] a = { 10, 20, 30 };

Console.WriteLine(Calculate (5, a ) );

int[] b = { -11, -4, 12, 14, 32, -1, 28 }

Console.WriteLine(Calculate (3, b ) );

short z = 1, e = 12;

byte v = 100;

Console.WriteLine(Calculate ( z, e, v ) );

Console.WriteLine(Calculate () ); //ошибка

Когда компилятор пытается разрешить вызов метода, он выполняет поиск метода, список аргументов которого соответствует вызываемому методу. Если не удается найти перегрузку метода с соответствующим списком аргументов, но имеется подходящая версия с параметром paramsнужного типа, выполняется вызов этого метода с добавлением в массив дополнительных аргументов.

## Перегрузка операций

C# позволяет переопределить действие большинства операций так, чтобы при использовании с объектами конкретного класса они выполняли заданные функции. Определение собственных операций класса называют перегрузкой операций.

При перегрузке операции ни одно из его исходных значений не теряется. Перегрузку операции можно расценивать как введение новой операции для класса.

Например, можно применять операции таким же образом, как стандартные:

MyObject a, b, с;

с = а + b;// используется операция сложения для класса MyObject

Операции класса описываются с помощью методов специального вида (функций-операций)*.* Синтаксис операции:

<спецификаторы> <тип\_возврата> operator

ор(тип\_параметра операнд1[,тип\_параметра операнд2])

{<тело операции>}

В качестве спецификатороводновременно используются ключевые слова public и static. Элемент ор – это оператор (например " + " или " / "), который перегружается. Телооперации определяет действия, которые выполняются при использовании операции в выражении.

При описании операций необходимо соблюдать следующие правила:

* операция должна быть описана как открытый статический метод класса (спецификаторы public static);
* параметры в операцию должны передаваться по значению (то есть не должны предваряться ключевыми словами ref или out);
* сигнатуры всех операций класса должны различаться; типы, используемые в операции, должны иметь не меньшие права доступа, чем сама операция (то есть должны быть доступны при использовании операции).

В С# существуют три вида операций класса: унарные, бинарные и операции преобразования типа.

**Унарные операции**

Можно определять в классе следующие унарные операции:

+, - , ! , ~ , ++ , -- , true, false

Примеры заголовков унарных операций:

public static int operator +( MyObject m )

public static MyObject operator -- ( MyObject m )

public static bool operator true( MyObject m )

Параметр*,* передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция должна возвращать:

* для операций +, –, ! и ~ величину любого типа;
* для операций ++ и -- величину типа класса, для которого она определяется;
* для операций true и false величину типа bool.

Операции не должны изменять значение передаваемого им операнда. Операция, возвращающая величину типа класса, для которого она определяется, должна создать новый объект этого класса, выполнить с ним необходимые действия и передать его в качестве результата.

**Бинарные операции**

Можно определять в классе следующие бинарные операции:

+ , - , \* , / , | , & , | | , && , == , =! , > , < , >= , <=

Примеры заголовков бинарных операций:

public static MyObject operator + (MyObject ml, MyObject m2 )

public static bool operator == (MyObject ml, MyObject m2 )

Хотя бы один параметр, передаваемый в операцию, должен иметь тип класса, для которого она определяется. Операция может возвращать величину любого типа.

Операции == и !=, > и <, >= и <= определяются только парами и обычно возвращают логическое значение.

**Пример 1.** Определение операции отношения (сравнение длин векторов) для класса TD.

class TD{ // Класс трехмерных координат

double x, у, z; // 3-х-мерные координаты

public TD() { x = у = z = 0; }

public TD(double i, double j, double k){

x = i; у = j; z = k;

}

static double r(TD op){ //Вычисление длины вектора

return Math.Sqrt(op.x\*op.x+op.у\*op.у+op.z\*op.z);

}

// Перегрузка оператора ">"

public static bool operator >(TD op1, TD op2){

return r(op1) > r(op2);

}

// Перегрузка оператора "<"

public static bool operator <(TD op1, TD op2){

return r(op1) < r(op2);

}

public void show(TD td){ // Отображаем координаты X, Y, Z

Console.WriteLine(td.x + ", " + td.у + ", " + td.z);

}

// Перегрузка унарного оператора "++"

public static TD operator ++(TD op){

op.x++;

op.y++;

op.z++;

return op;}

}

class TDDemo{

public static void Main(){

TD a = new TD(1, 5, 3);

TD с = new TD(3, 2, 1);

Console.Write("Координаты точки а: ");

a.show(a);

Console.Write("Координаты точки c: ");

с.show(c);

Console.WriteLine();

if (a > с)

Console.WriteLine("a>c ");

else

Console.WriteLine("a<c);

а++; // Инкрементирование а

Console.Write("Результат +:");

а.show(a);

}

}

**Пример 2.** Описать класс для работы с двумерным массивом целых чисел. Реализовать возможность выполнение операции нахождения остатков от деления всех элементов массива на заданное число.

class Massiv{

private int[,] massiv;

private int row, col; // число строк, столбцов матрицы

public Massiv(int row, int col){//конструктор

this.row = row;

this.col = col;

massiv = new int[row, col];

for (int i = 0; i < row; i++){

for (int j = 0; j < col; j++){

massiv[i, j] = (i+10) \* (j+10);

}}}

public int this[int i, int j]{ //Индексатор

set { massiv[i, j] = value; }

get { return massiv[i, j]; }

}

public static Massiv operator %(Massiv ar, int nbr){

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

for (int j = 0; j < 3; j++){

ar.massiv[i,j] = ar.massiv[i,j] % nbr;

}}

return ar;

}

static void Main(){

int row = 3;

int col = 3;

Massiv mas = new Massiv(row, col);

mas = mas % 3;%остаток от целочисленного деления матрицы на 3

for (int i = 0; i < row; i++){

for (int j = 0; j < col; j++){

Console.Write(" {0} ", mas[i,j]);

}

Console.WriteLine();

}

Console.ReadLine();

}

**Пример 3.** Создать класс с закрытыми полями а и b, строковой переменной, означающей операцию арифметического выражения и свойством С. Свойство – значение выражения над полями а и b. Типы полей а и b – float; операции арифметического выражения +=,--; равенство объектов – по а свойству С для --. Выполнить переопределение метода Equals() для сравнения объектов и метода ToString() для вывода состояния объекта.

class MyClass{

public float A { set; get; }

public float B { set; get; }

public string Operation { set; get; }

public MyClass(int a, int b, string operation){

A = a;

B = b;

Operation = operation;

}

public float C{

get{

if (Operation == "+="){

return A + B;

}

else if (Operation == "--")

return A--;

else

return 0;

}

}

public override bool Equals(object obj){

if (this.A.ToString() == obj.ToString())

return true;

else

return false;

}

public override string ToString(){

return string.Format(" A = {0} B = {1} Operation = {2}", A,

B, Operation);

}

}

static void Main(string[] args){

MyClass mc = new MyClass(10, 20, "--");

Console.WriteLine(mc.C);

bool b = mc.Equals(mc.C);

Console.WriteLine(b);

Console.WriteLine(mc.ToString());

Console.ReadLine();

}

**Контрольные вопросы:**

1. Что понимается под перегрузкой методов?

## Индивидуальные задания

**Задание 1.** Перегрузка операций.

Создайте проект, в котором опишите класс для решения задачи Вашего варианта. Разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы, перегруженные операции. В программе должна выполняться проверка всех разработанных элементов класса.

**Варианты заданий:**

1. Описать класс для работы с двумерными массивами чисел. Реализовать возможность выполнения для согласованных массивов комбинированных операций присваивания (+=, -=).
2. Описать класс для работы с двумерным массивом строк фиксированной длины. Обеспечить сравнение массивов на равенство (перегрузку операции == для поэлементного сравнения).
3. Описать класс для работы с двумерным массивом целых чисел. Реализовать возможность выполнение операции нахождения умножения всех элементов массива на заданное число.
4. Описать класс, реализующий тип данных «вещественная матрица». Реализовать вычитание строки заданного номера из всех остальных строк, кроме данной строки.
5. Описать класс для работы с двумерным массивом строк. Обеспечить перегрузку операции + для построчного соединения элементов.
6. Описать класс для работы с двумерным массивом. Реализовать перегруженные операции отношений (>,<), выполняющие сравнение сумм элементов главной диагонали.
7. Описать класс для работы с двумерным массивом чисел. Реализовать возможность выполнения операции умножения согласованных массивов.
8. Описать класс для работы с двумерным массивом целых чисел. Реализовать возможность уменьшения количества столбцов массива на заданное число (перегрузка операции --) с удалением их с начала массива.
9. Описать класс для работы с двумерным массивом вещественных чисел. Обеспечить добавление к первому столбцу столбца заданного номера (перегрузка операции +).
10. Описать класс, реализующий тип данных «вещественная матрица». Реализовать операцию (++) для выполнения циклического сдвига столбцов матрицы.
11. Описать класс, реализующий тип данных «вещественная матрица». Реализовать изменение значений ненулевых элементов матрицы на противоположные, нули заменить 1.
12. Описать класс для работы с двумерным массивом целых чисел. Реализовать возможность нахождения числа, полученного перемножением положительных элементов массива, меньших 10.

**Задание 2.** Переопределение методов класса Object.

Создать класс с закрытыми полями а и b, строковой переменной, означающей операцию арифметического выражения и свойством С. Свойство – значение выражения над полями а и b (выражение и типы полей – см. в таблице). Поля инициализировать при создании объекта. Выполнить переопределение метода Equals(с одним и двумя параметрами) для сравнения объектов и метода ToString()для вывода состояния объекта.

### Варианты заданий:

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Тип полей а, b** | **Выражения** | **Равенство объектов** |
| 1. | float | /,-,+= | По а и свойству C для /, - |
| 2. | double | /=,+,\* | По b и свойству C для /=, \* |
| 3. | int | -,++,/ | По свойству С для всех операций |
| 4. | short | \*=,+,/ | По а и свойству С для \*= |
| 5. | long | ++,\*=,- | По b и свойству С для ++, - |
| 6. | ushort | %=,+,-- | По любой операции свойства и а |
| 7. | float | /=,-,\* | По а и свойству С для -- |
| 8. | double | \*=,%,/ | По свойству С для \*= и ++ |
| 9. | decimal | \*=,/,- | По свойству С для \*= , – и b |
| 10. | int | +=,--,- | По b и свойству С для -- |
| 11. | long | -=,++,- | По а, свойству С для -= и ++ |
| 12. | float | /=,\*,% | По свойству С для всех операций |

# ЛАБОРАТОРНАЯ РАБОТА №4. ИЕРАРХИИ КЛАССОВ. НАСЛЕДОВАНИЕ. 2 – часа

Цель работы: изучение приемов создания иерархии классов, выделение общих признаков объектов в базовый класс, организации доступа к элементам базового и производных классов.

## Наследование

Класс в С# может иметь произвольное количество потомков и только одного предка. При описании класса имя его предка записывается в заголовке класса после двоеточия:

[атрибуты][спецификаторы] class <имя\_класса> [:предки] <тело класса>

Если имя предка не указано, предком считается базовый класс всей иерархии System.Object.

Элементы базового класса, определенные как private, в производном классе недоступны. Поля, определенные со спецификатором protected, будут доступны методам всех классов, производных от базового. Этот модификатор остается со своим членом независимо от реализуемого количества уровней наследования.

**Пример 1.** Рассмотрим в качестве базового класс, в котором описывается печатная продукция (журналы, газеты и др.)

class Press{

//Закрытые поля

string name; //Название

int copies; //Тираж

double price;//Цена

//Конструктор с параметрами

public Press(string name, int copies, double price){

this.name = name;

this.copies = copies;

this.price = price;

}

//Свойства

public string Name {get {return name;}}

public int Copies {get {return copies;} set {copies = value;}}

public double Price {get {return price;}}

//Метод вычисления стоимости тиража

public double Cost(Press pr){return pr.copies\*pr.price;}

//Метод вывода состояния объекта

public void Output(Press pr){

Console.WriteLine("Название: {0} \t Тираж: {1} \t Цена:

{2}", pr.name, pr.copies, pr.price);

}}

## Вызов конструкторов базового класса

Конструкторы не наследуются, поэтому производный класс должен иметь собственные конструкторы. Порядок вызова конструкторов определяется приведенными далее правилами:

Если в конструкторе производного класса явный вызов конструктора базового класса отсутствует, автоматически вызывается конструктор базового класса без параметров.

Для иерархии, состоящей из нескольких уровней, конструкторы базовых классов вызываются, начиная с самого верхнего уровня. После этого выполняются конструкторы тех элементов класса, которые являются объектами, в порядке их объявления в классе, а затем исполняется конструктор класса. Таким образом, каждый конструктор инициализирует свою часть объекта.

Если конструктор базового класса требует указания параметров, он должен быть явным образом вызван в конструкторе производного класса в списке инициализации. Вызов выполняется с помощью ключевого слова base.

Формат расширенного объявления таков:

имя\_производного\_класса(список\_параметров):base (список\_аргументов)

{// тело конструктора}

Здесь с помощью элемента список\_аргументов задаются аргументы, необходимые конструктору в базовом классе. При отсутствии ключевого слова base автоматически вызывается конструктор базового класса, действующий по умолчанию.

**Пример 2.** Дополним базовый класс из примера 1 производным   
Magazine, в котором есть дополнительное поле – качество бумаги.

class Magazine: Press {

string quality; //Качество бумаги

public string Quality{ //Свойство

get { return quality;}

}

//Конструктор

public Magazine(string name, int copies, double

price, string quality):base(name,copies,price){//Вызов базового

//конструктора

this.quality = quality;

} }

Ключевое слово base всегда отсылает к базовому классу, стоящему в иерархии классов непосредственно над вызывающим классом. В примере 2 это класс Press и его конструктор.

## Наследование и сокрытие имен

Новым членам (полям, методам и свойствам) производного класса можно давать имена, совпадающие с именами членов базового класса. В этом случае перед членом производного класса необходимо поставить ключевое слово new. При этом, хотя соответствующие члены базового класса наследуются, они становятся скрытыми в производном классе.

Для обращения к скрытому члену применяется ссылка base, которая указывает на базовый класс производного класса. Формат ее записи такой:

base.<член базового класса>

Здесь в качестве элемента член базового класса можно указывать либо метод, либо переменную экземпляра.

Объекту базового класса можно присваивать объект производного класса, но вызываются для него только методы и свойства, определенные в базовом классе. Иными словами, возможность доступа к элементам класса определяется типом ссылки, а не типом объекта, на который она указывает. Это и понятно: ведь компилятор еще до выполнения программы определяет, какой метод вызывать, и вставляет в код фрагмент, передающий управление на этот метод. Этот процесс называется ранним связыванием.

**Пример 3.** Изменим метод Cost базового класса Press из предыдущего примера в производном классе Magazine следующим образом: в зависимости от его значения (высокое, среднее, низкое) стоимость тиража увеличивается на 10%, не изменяется или уменьшается на 10% соответственно. В метод Output добавим вывод нового поля.

class Magazine: Press {

…

public new double Cost(Press pr){

if (quality == "HIGH")

return base.Cost(pr)\*1.1; //Обращение к методу

//базового класса

else

if (quality == "LOW")

return base.Cost(pr)\*0.9;

else

return base.Cost(pr);

}

public new void Output(Press pr){

base.Output(pr); //Вызов метода класса Press

Console.WriteLine("Качество бумаги:" + quality);}

… }

Использование созданной в примерах 1–3 иерархии классов:

class Program{

static void Main(){

//Объект базовогокласса

Press edu = new Press("Информатика", 250, 1.5);

edu.Output(edu); //обращение к методу базового класса

Console.WriteLine("Стоимость тиража:"+edu.Cost(edu));

//Объект производного класса

Magazine IiO = new Magazine("Информатика и образование", 1500,

3.9, "LOW");

IiO.Output(IiO);//обращение к методу производного класса

Console.WriteLine("Стоимость тиража:"+IiO.Cost(IiO));

//Ссылка типа базового класса на объект производного класса

Press informatica = new Magazine("Мир ПК",1000, 2.1, "HIGH");

informatica.Output(informatica);//обращение к методу базового

//класса

Console.WriteLine("Стоимость

тиража:"+informatica.Cost(informatica));}

}

Поскольку в производном классе определяется собственный метод с именем Cost, он скрывает одноименный метод, определенный в базовом классе. Следовательно, при вызове этого метода для объекта типа Magazine, вычисления выполняются так, как предусмотрено в этом классе, а не в классе Press. Ссылка base позволяет получить доступ к методу в базовом классе. Объект informatica типа Press создается с помощью конструктора класса Magazine. Но так как разрешение, какой метод вызвать, в данном случае осуществляется в момент компиляции (раннее связывание), то обращение происходит к методу базового класса.

**Контрольные вопросы:**

1. В чем состоит принцип наследования?
2. Какие члены класса наследуются?
3. Что представляет собой защищенный доступ?
4. Как происходит вызов конструкторов базового класса?
5. Что такое сокрытие имен при наследовании?
6. Как получить доступ к сокрытому члену базового класса?
7. Когда проявляется механизм раннего связывания?

## Индивидуальные задания

**Задание 1.**Создайте проект, в котором опишите иерархию классов для решения задачи Вашего варианта. Дополните базовый класс, который был создан в лабораторной работе № 1, классами-потомками. Производные классы должны содержать необходимые дополнения и изменения, указанные в варианте задания.

Составить тестирующую программу с выдачей результатов. В программе должна выполняться проверка разработанных элементов всей иерархии классов, вывод состояния различных объектов.

### Варианты заданий:

|  |  |  |
| --- | --- | --- |
| **№** | **Потомки** | **Дополнения и изменения в потомках** |
| 1. | Менеджер (поле объем продаж в тоннах) | Доход менеджера изменить в зависимости от объема продаж (если > H, то увеличить на 1%  от H) |
| Инженер (поле количество разработанных проектов –  n) | Доход инженера увеличить на 4.8\*n. |
| 2. | Квартира в центре (поля  номер этажа, этажность дома) | Увеличить стоимость с учетом надбавки за  расположение в центре на 1%, и уменьшить на 1000$ для квартир на 1 и последнем этажах. |
| Квартира в пригороде (поля расстояние от центра r) | Изменить стоимость в зависимости от расстоя-  ния: если < 10км увеличить на 3%, в против- ном случае уменьшить на 0,01\*r. |
| 3. | Фермерское (поле: количе-  ство внесенных удобрений m) | Найти урожай с учетом увеличения на 0,001\*m на единицу площади. |
| Приусадебное (поле: сроки  посева – ранний, средний, поздний) | Изменить урожай с учетом сроков (+10% для  раннего, –5% для позднего) и площади посе- вов (<0,01 увеличить на 50% от S) |
| 4. | Письменный (поле – ис-  пользуемый материал, стоимость отделки) | Увеличить стоимость на стоимость отделки. |
| Обеденный (поле – форма) | Изменить стоимость в зависимости от формы: увеличить для прямоугольной формы на 10%,  овальной – на 20% при S<0,5 м2 и S>2 м 2. |
| 5. | Экспресс (поле – средняя скорость v, марка автобуса) | Изменить выручку с учетом увеличения на 0.05\*v к цене билета. |
| Пригородный (поле – рас-  стояние r) | Изменить выручку с учетом уменьшения на  0.01\*r к цене билета. |
| 6. | Самолет (высота – h км,  скорость – v км/ч) | Увеличить стоимость проезда на 100\*h\*v |
| Корабль (количество палуб  k, номер палубы n) | Увеличить стоимость проезда на палубах №3-4  на k2% |
| 7. | Инвалид (поле – номер  группы) | Увеличить пособие для инвалидов 1-й группы  на 30%, 2-й – на 20%. |
| Многодетные семьи (поле – количество детей) | Увеличить пособие от 3 до 5 детей на 10%, > 5  – на 20%. |
| 8. | Сотовый (поля – модель, год производства) | Изменить стоимость для телефонов, выпущен- ных: менее 1 года назад на +20%, более 3 лет –  на –60%. |
|  | Стационарный (поле – мо-  бильность: переносной, непереносной) | Увеличить стоимость переносного телефона  на 5.7, уменьшить стоимость непереносного телефона на 3.2. |
| 9. | Грузовой  (поле – грузоподъемность p в т) | Увеличить объем горючего на величину  *M*  *p* \* *R* |
| Легковой  (поле – объем двигателя V в л) | Увеличить объем горючего для объема > 3 на величину  M= 0,005\*V\*R |
| 10. | Пальто  (поле размер V) | Увеличить расход ткани:  для пальто на (V/6.5+0.5)/10 |
| Костюм  (поле рост H) | для костюма на (2\*H+0.3)/8 |
| 11. | Офис  (поле – количество этажей N) | Изменить высоту фундамента: для офиса с N>10 на+ 0,005\*N |
| Завод (поле -– вес G) | для завода на +0,000002\*G |
| 12. | Лекционная  (поле количество ярусов K) | Увеличить количество мест в лекционной аудитории на 2\*K |
| Компьютерная  (поле количество компью- теров P) | Заменить количество мест в компьютерной аудитории на P-1 |

# ЛАБОРАТОРНАЯ РАБОТА №5. ВИРТУАЛЬНЫЕ МЕТОДЫ И ПОЛИМОРФИЗМ. 4 – часа

Цель работы: формирование представления о реализации принципа полиморфизма с помощью механизма позднего связывания.

## Виртуальные методы

В том случае, если ссылке базового класса присваивается объект производного класса, то вызываются для него только методы и свойства, определенные в базовом классе. Чтобы вызываемые методы соответствовали типу объекта, необходимо отложить процесс связывания до этапа выполнения программы, а точнее – до момента вызова метода, когда уже точно известно, на объект какого типа указывает ссылка. Такой механизм называется поздним связываниеми реализуется с помощью так называемых виртуальных методов.

В С# виртуальные методы описываются с использованием ключевого слова virtual. Оно записывается в заголовке метода базового класса, например:

virtual public void Output()…

Если в производном классе требуется переопределить виртуальный метод, используется ключевое слово override, например:

override public void Output()

Переопределенный виртуальный метод должен обладать таким же набором параметров, как и одноименный метод базового класса.

При описании классов рекомендуется определять в качестве виртуальных те методы, которые в производных классах должны реализовываться по-другому. Если во всех классах иерархии метод будет выполняться одинаково, его лучше определить как обычный метод.

Все сказанное о виртуальных методах относится также к свойствам и индексаторам.

**Пример 1.** Опишем методы Cost и Output базового класса Press из примеров 1–3 раздела 4 как виртуальные. В производном классе Magazine переопределим их. Протестируем процесс вызова метода с помощью объектов как базового, так и производного классов.

class Press {

…

//Виртуальный метод вычисления стоимости тиража

virtual public double Cost(){

return copies\*price;

}

//Виртуальный метод вывода состояния объекта

virtual public void Output(){

Console.WriteLine("Название: {0} \t Тираж: {1}

\t Цена: {2}",name,copies,price);

}

}

class Magazine: Press{

…

//Переопределенные методы

override public double Cost(){

if (quality == "HIGH")

return base.Cost()\*1.1;

else

if (quality == "LOW") return base.Cost()\*0.9;

else

return base.Cost();

}

override public void Output(){

base.Output();

Console.WriteLine("Качество бумаги:"+ quality);

}}

class Program{

static void Main(){

const int n = 3;

Base[] st = new Base[n]; //создаем контейнер ссылок

//на объекты базового класса

//создаем объекты базового класса

st[0] = new Press("Информатика", 250, 1.5);

st[l] = new Press("Компьютер", 100, 3.5);

//создаем объект класса-наследника

st[2] = new Magazine("Информатика и образова- ние", 1500, 3.9,

"LOW");

//выводим поля объектов массива

foreach (Press elem in st ){

elem.Output();

Console.WriteLine("Стоимость тиража:"+ st[i].Cost());

}

//обнуляем поля объектов

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

st[i].Copies = 0;

//выводим поля объектов массива

foreach (Press elem in st )

elem.Output();}}

В циклах вызываются методы и свойства, соответствующие типу объекта, помещенного в массив. Это иллюстрирует механизм позднего связывания.

## Абстрактные классы

Абстрактный класс служит только для порождения потомков. Как правило, в нем задается набор методов, которые каждый из потомков будет реализовывать по-своему.

Абстрактный класс задает интерфейс для всей иерархии, при этом метод класса может не содержать никаких конкретных действий. В этом случае методы имеют пустое тело и объявляются со спецификатором abstract. Если в классе есть хотя бы один абстрактный метод, весь класс также должен быть описан как абстрактный. Абстрактный класс может содержать и полностью определенные методы.

**Пример 2.** Абстрактный класс и его потомки.

using System;

namespace ConsoleApplication2 {

abstract class Spirit{

public abstract void Print();

}

class Base : Spirit{

override public void Print(){

Console.WriteLine( "Base ");

}}

class Derivative : Base{

override public void Print(){

Console.WriteLine("Derivative");

}}}

Если класс, производный от абстрактного, не переопределяет все абстрактные методы, он также должен описываться как абстрактный.

Можно создать метод, параметром которого является абстрактный класс. На место этого параметра при выполнении программы может передаваться объект любого производного класса. Это позволяет создавать полиморфные методы, работающие с объектом любого типа в пределах одной иерархии.

**Контрольные вопросы:**

1. Что означает принцип полиморфизма?
2. Для чего используется позднее связывание?
3. В каких случаях используются виртуальные методы?
4. Какие условия необходимо соблюдать при переопределении виртуального метода?
5. Что представляют собой абстрактные классы? Для чего они предназначены?
6. Могут ли в абстрактном классе быть неабстрактные методы?

## Индивидуальные задания

**Задание 1.** Модифицируйте проект из лабораторной работы № 4 следующим образом.

Создайте библиотеку классов и добавьте в нее базовый класс.

Опишите методы в базовом классе как виртуальные.

Составьте новое приложение с двумя потомками базового класса из библиотеки. Тестирующая программа должна содержать полиморфный контейнер – массив ссылок базового класса на объекты базового и производных классов (количество объектов >= 5). В программе должна выполняться проверка всех разработанных элементов.

**Задание 2.**Модифицируйте проект задания 1 следующим образом.

Базовый класс опишите как абстрактный и добавьте его в созданную библиотеку. Указанный в таблице (лаб. раб. №1) метод базового класса не должен содержать никаких конкретных действий.

В тестирующей программе используйте полиморфный контейнер – массив ссылок абстрактного класса на объекты производных классов (количество объектов >= 5 и размещаются в массиве в произвольном порядке) и реализуйте указанное в варианте задания дополнение. В программе должна выполняться проверка всех разработанных элементов.

**Варианты заданий:**

1. Организовать вычисление суммарной величины дохода сотрудников и максимальное для менеджеров.
2. Рассчитать среднюю стоимость квартир в центре и найти наибольшее расстояние от города до квартир в пригороде.
3. Найти суммарный урожай фермеров и суммарный урожай с приусадебных участков.
4. Найти суммарную стоимость всех столов и наименьшую площадь обеденных столов.
5. Рассчитать общую выручку и найти наибольшее расстояние для пригородных автобусов.
6. Организовать вычисление средней стоимости проезда на самолетах и средней стоимости проезда кораблем.
7. Определить, на какой вид пособий потребуется больше средств.
8. Найти максимальные стоимости сотовых и стационарных телефонов.
9. Организовать вычисление суммарного расхода горючего и средний расход для легковых автомобилей.
10. Организовать вычисление суммарного расхода ткани для каждого вида одежды.
11. Найти максимальную высоту фундаментов для офиса и минимальную для заводов.
12. Найти общее количество мест в лекционных аудиториях и общее число компьютеров.

# ЛАБОРАТОРНАЯ РАБОТА №6. ИНТЕРФЕЙСЫ И КОЛЛЕКЦИИ. 4 – часа

Цель работы: сформировать понятие о реализации принципа полиморфизма с помощью интерфейсов, умения использовать пользовательские интерфейсы для задания поведения различных классов.

## Понятие интерфейса

Интерфейс является специальным видом классов. Синтаксис интерфейса аналогичен синтаксису класса:

[атрибуты][спецификаторы] interface

<имя\_интерфейса> [:предки] <тело\_интерфейса> [;]

Для интерфейса могут быть указаны спецификаторы, new, publiс, protected, internal и private. По умолчанию интерфейс доступен только из сборки, в которой он описан (internal). Интерфейс может наследовать свойства нескольких интерфейсов. Тело интерфейса составляют абстрактные методы, шаблоны свойств и индексаторов, а также события.

Интерфейс не может содержать константы, поля, операции, конструкторы, деструкторы, типы и любые статические элементы. В интерфейсе методы неявно являются открытыми (public-методами), при этом не разрешается явным образом указывать спецификатор доступа.

Чтобы реализовать интерфейс, нужно указать его имя после имени класса. В списке предков класса сначала указывается его базовый класс, если он есть, а затем через запятую – интерфейсы, которые реализует этот класс.

Методы, которые реализуют интерфейс, должны быть объявлены открытыми. Сигнатура типа в реализации метода должна в точности совпадать с сигнатурой типа, заданной в определении интерфейса. В классах, которые реализуют интерфейсы, можно определять дополнительные члены.

**Пример 1.**

interface IMy\_A{

void meth(int x);

}

// В классе MyClass реализован интерфейс IMy\_A

class MyClass : IMy\_A{

int h;

public MyClass(){

h = 5;

}

public void meth(int x){

h= x + x;

Console.WriteLine("MyClass.meth..."+h);

}

}

class Class1: IMy\_A{

public static void meth(int a){

Console.WriteLine("Class1.meth..."+a\*a);

}

public static void Main(){

MyClass ob = new MyClass();

Console.Write ("Вызов метода ob.meth(): ");

ob.meth(3);

meth(3); //реализация интерфейса IMy\_A в классе Class1

}}

Можно объявить ссылочную переменную интерфейсного типа, которая может ссылаться на любой объект, реализующий ее интерфейс. При вызове метода для объекта посредством интерфейсной ссылки будет выполнена та версия указанного метода, которая реализована этим объектом.

Элементы с одинаковыми именами или сигнатурой могут встречаться более чем в одном интерфейсе. В этом случае при множественном наследовании может возникнуть конфликт из-за неоднозначности ситуации, так как компилятор не может определить из контекста обращения к элементу, элемент какого именно из реализуемых интерфейсов требуется вызвать. Избежать неоднозначности можно с помощью бинарных операций is или as, которые позволяют убедиться, что объект поддерживает данный интерфейс или выполнить приведение к данному интерфейсу.

Для этой же цели можно использовать так называемую явную реализацию, когда имя интерфейса явно указывается перед реализуемым элементом через точку. Такая реализация интерфейса является закрытой. Спецификаторы доступа при этом не указываются. К таким элементам можно обращаться в программе только через объект типа интерфейса.

Кроме того, явная реализация позволяет избежать конфликтов при множественном наследовании, если элементы с одинаковыми именами или сигнатурой встречаются более чем в одном интерфейсе.

**Пример 2.** Реализовано два интерфейса, причем оба объявляют метод с одним именем.

//Использование явной реализации, чтобы избежать неоднозначности.

interface IMy\_A{

void meth(int x);

}

interface IMy\_B{

void meth(int x);

}

// В классе MyClass реализованы оба интерфейса

class MyClass : IMy\_A, IMy\_B {

int h;

public MyClass(){

h = 5;

}

//Явным образом реализуем метод meth() интерфейса IMy\_A

void IMy\_A.meth(int x){

h = x + x;

Console.WriteLine("IMy\_A.meth..." + h);

}

public void meth(int x){

h = x \* x; Console.WriteLine("meth..." + h);

}

}

public static void Main(){

MyClass ob = new MyClass();

//Реализация интерфейса IMy\_В

Console.Write("Вызов метода ob.meth(): ");

ob.meth(3);

Console.WriteLine("Реализация интерфейса IMy\_A");

(ob as IMy\_A).meth(4);

Console.WriteLine("Вызов метода meth инт\_ссылкой");

IMy\_A a\_ob= ob; //интерфейсн. ссылке присваиваем объект класса

Console.WriteLine("Вызов метода IMy\_A.meth()");

a\_ob.meth(3); //реализация интерфейса IMy\_A

}

Метод meth() имеет одинаковую сигнатуру в интерфейсах IMy\_A и IMy\_B. Поскольку единственный способ вызова явно заданного метода состоит в использовании интерфейсной ссылки, метод meth(), объявленный в интерфейсе IMy\_A, создает ссылку на интерфейс IMy\_A, а метод ob.meth(), вызывает метод, объявленный в интерфейсе IMy\_B.

## Наследование интерфейсов

Интерфейс может не иметь или иметь сколько угодно интерфейсов-предков, в последнем случае он наследует все элементы всех своих базовых интерфейсов, начиная с самого верхнего уровня. Базовые интерфейсы должны быть доступны так же, как их потомки.

В интерфейсе-потомке можно также указать элементы, переопределяющие унаследованные элементы с такой же сигнатурой. В этом случае перед элементом указывается ключевое слово new.

Любой класс, который реализует интерфейс, должен реализовать все методы, определенные этим интерфейсом, включая методы, которые унаследованы от других интерфейсов.

Класс наследует все методы своего предка, в том числе те, которые реализовывали интерфейсы. Он может переопределить эти методы с помощью спецификатора new, но обращаться к ним можно будет только через объект класса. Если использовать для обращения ссылку на интерфейс, вызывается не переопределенная версия.

Однако если интерфейс реализуется с помощью виртуального метода класса, после его переопределения в потомке любой вариант обращения (через класс или через интерфейс) приведет к одному и тому же результату.

Метод интерфейса, реализованный явным указанием имени, объявлять виртуальным запрещается.

Если класс наследует от класса и интерфейса, которые содержат методы с одинаковыми сигнатурами, унаследованный метод класса воспринимается как реализация интерфейса.

## Стандартные интерфейсы среды .NET Framework

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Стандартные интерфейсы поддерживаются многими стандартными классами библиотеки. Можно создавать и собственные классы, поддерживающие стандартные интерфейсы, что позволит использовать объекты этих классов стандартными способами.

Во многих классах необходимо реализовать интерфейс IComparableили IComparer, поскольку они позволяют сравнить два объекта.

Интерфейс IComparable состоит только из одного метода:

int CompareTo(object v)

Этот метод сравнивает вызывающий объект со значением параметра v*.* Метод возвращает положительное число, если вызывающий объект больше объекта v*,* нуль, если два сравниваемых объекта равны, и отрицательное число, если вызывающий объект меньше объекта v*.*

Метод CompareTo может сгенерировать исключение типа ArgumentException, если тип объекта v несовместим с вызывающим объектом.

В интерфейсе IComparer определен метод Compare (), который позволяет сравнивать два объекта:

int Compare(object vl, object v2)

Метод Compare() возвращает положительное число, если значение vl больше значения v2, отрицательное, если vl меньше v2, и нуль, если сравниваемые значения равны. Этот интерфейс можно использовать для задания способа сортировки элементов коллекции.

## Класс ArrayList

Класс ArrayList представляет коллекцию объектов, в которой можно сохранить вместе разнотипные объекты - строки, числа и т.д.

Основные методы класса:

int Add(object value): добавляет в список объект value

void AddRange(ICollection col): добавляет в список объекты коллекции col, которая представляет интерфейс ICollection - интерфейс, реализуемый коллекциями.

void Clear(): удаляет из списка все элементы

bool Contains(object value): проверяет, содержится ли в списке объект value. Если содержится, возвращает true, иначе возвращает false

void CopyTo(Array array): копирует текущий список в массив array

ArrayList GetRange(int index, int count): возвращает новый список ArrayList, который содержит count элементов текущего списка, начиная с индекса index

int IndexOf(object value): возвращает индекс элемента value

void Insert(int index, object value): вставляет в список по индексу index объект value

void InsertRange(int index, ICollection col): вставляет в список начиная с индекса index коллекцию ICollection

int LastIndexOf(object value): возвращает индекс последнего вхождения в списке объекта value

void Remove(object value): удаляет из списка объект value

void RemoveAt(int index): удаляет из списка элемент по индексу index

void RemoveRange(int index, int count): удаляет из списка count элементов, начиная с индекса index

void Reverse(): переворачивает список

void SetRange(int index, ICollection col): копирует в список элементы коллекции col, начиная с индекса index

void Sort(): сортирует коллекцию

Кроме того, с помощью свойства Count можно получить количество элементов в списке.

**Пример 3.** Создать класс Student, описывающий работу со студентами. Пусть у него будут поля: fio – фамилия студента; ochenka – средняя оценка. В программе main() создать массив типа ArrayList, и затем отсортировать этот массив. Сортировку сделать по фамилиям в алфавитном порядке, в случае одинаковых фамилий – по возрастанию средних оценок.

using System;

using System.Collections;

namespace Collections

{

public class Student : IComparable, IComparer

{

public string fio;

public double ochenka;

public Student()

{

fio = "NoName";

ochenka = 0;

}

public Student(string f, double o)

{

fio = f;

ochenka = o;

}

public void print()

{

Console.WriteLine("FIO = {0}  средняя оценка = {1}",

fio, ochenka);

}

// Реализация метода Compare() интерфейса IComparer

public int Compare(Object x0, Object y0)

{

Student x = (Student)x0;

Student y = (Student)y0;

if (String.Compare(x.fio, y.fio) < 0)

{

return -1;

}

else if (String.Compare(x.fio, y.fio) == 0)

{

if (x.ochenka < y.ochenka)

return -1;

else if (x.ochenka == y.ochenka)

return 0;

else

return 1;

}

else

return 1;

}

// Реализация метода CompareTo() интерфейса IComparable

public int CompareTo(Object x)

{

return Compare(this, x);

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Тестирование класса Student");

Console.WriteLine("\nСписок студентов до

сортировки:");

ArrayList x = new ArrayList();

x.Add(new Student("Иванов", 3.5));

x.Add(new Student("Петров", 4.2));

x.Add(new Student("Борисов", 4.5));

x.Add(new Student("Сергеев", 4.8));

x.Add(new Student("Борисов", 3.0));

Student st = new Student();

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

{

st = (Student)x[i];

st.print();

}

//Создаём ссылку Comp типа IComparer на объект

//типа Student

IComparer Comp = new Student();

//Передаём ссылку Comp типа IComparer в метод

//сортировки

x.Sort(Comp);

Console.WriteLine("\nСписок студентов после

сортировки:");

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

{

st = (Student)x[i];

st.print();

}

Console.ReadKey(true);

}

}

}

**Контрольные вопросы:**

1. Как описывается интерфейс? Его назначение.
2. Какие члены может содержать интерфейс?
3. Какие спецификаторы допустимы у методов, реализующих интерфейс?
4. В каких случаях используется явная реализация интерфейса?
5. Как осуществляется наследование интерфейсов?
6. Можно ли явно реализованные методы объявлять виртуальными?
7. Можно ли повторно реализовать интерфейс, указав его имя в списке предков класса наряду с классом-предком?
8. Какие стандартные интерфейсы используются для работы с коллекциями?
9. Объясните назначение класса ArrayList.
10. Назовите основные методы и свойства класса ArrayList.

## Индивидуальные задания

**Задание 1.**

1.1. Реализовать неявно интерфейсы в классах 1 и 2, содержащих поле указанного типа.

Переменная w обозначает параметр метода или поле. Результат метода со спецификатором void присвоить полю класса.

В тестирующей программе должны выполняться:

– неявная неоднозначная реализация методов интерфейсов;

– вызов методов с явным приведением к типу интерфейса;

– вызов методов для объекта посредством интерфейсной ссылки.

1.2. Модифицировать проект, используя явную реализацию интерфейса для методов с одинаковой сигнатурой в классе 1.

### Варианты заданий:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **интерфейсы** | **поле** | **классы** | **F0 F1 возвращают:** | |
| неявная реализация | явная реа- лизация |
| 1. | interface Iy  {void F0(double параметр);  void F1();} interface Iz  {void F0(double параметр); float F1(int  параметр);} | double | 1 |  | w2+5 |
| 2 | ew | - |
| 2. | interface IA  {void F0(out double параметр);  int F1(double параметр);} interface IB  {void F0( out double параметр);  void F1();} | double | 1 | w2 | 15/w |
| 2 | w3 | - |
| 3. | interface Ix  {char F0(); void F1(ref char параметр);} interface Iy  {void F0(char параметр);  void F1(ref char  параметр);} | char | 1 | Символ, преобразо- ванный в нижний  регистр | цифру ‘5’, если сим- вол буква |
| 2 | \*, если символ – буква | - |
| 4. | interface Ix  {void F0(); void F1(string параметр);} interface Iy  {void F0(string параметр); void F1(string параметр);} | string | 1 | Строку, удалив два первых символа | Строку, заменив первый символ символом  ‘**-‘** |
| 2 | Строку, удалив два последних  символа | - |
| 5. | interface Ix  {bool F0(char параметр);  void F1(int параметр);} interface Iy  {bool F0(char параметр); void F1(char параметр);} | char | 1 | Определя- ет, являет- ся ли сим- вол (код символа) цифрой | Определя- ет, содер- жится ли значение кода сим- вола в диа- пазоне ко-  дов ASCII |
| 2 | Определя- ет, являет- ся ли сим- вол знаком  препина- ния | - |
| 6. | interface Ix  {string F0(int параметр);  void F1(int параметр);} interface Iy  {void F0(int параметр,out string параметр);  void F1(int параметр);} | string | 1 | Удвоенную строку с удаленным  k-м симво- лом | Cтроку, заменив k- ый символ знаком + |
| 2 | подстроку, начиная с k-ей пози-  ции | - |
| 7. | interface Iy  {void F0(double параметр); void F1(double параметр);} interface Iz  {void F0(double параметр);  double F1();} | double | 1 | sin(w) | w+2 |
| 2 | 2/w\*ln(w  ) | - |
| 8. | interface Ix  {void F0(int k);  void F1(int k, out int параметр);}  interface Iy  {void F0(int k);  void F1(int k);} | int | 1 | k\*w2 | 2w-k |
| 2 | |w|-k | - |
| 9. | interface IA  {double F0(double параметр);  int F1(double параметр);} interface IB  {double F0(double параметр);  void F1();} | double | 1 | 1/ew | w/10 |
| 2 | w-1010\*w | - |
| 10. | interface Ix  {int F0();  void F1();} interface Iy  {int F0();  void F1(char  параметр);} | char | 1 | Если сим- вол являет- ся цифрой найти его числовое значение, иначе его  код | вернуть код симво- ла |
| 2 | Если сим- вол являет- ся буквой х, вывести  число 0 | - |
| 11. | interface Iy  {void F0(double параметр); float F1(int параметр);} interface Iz  {void F0(out double параметр);  float F1(int  параметр);} | float | 1 | 7w/ln(2) | 7w\*2.6 |
| 2 | |w-10| | - |
| 12. | interface Is  {string F0();  void F1();} interface Ir  {void F0(string параметр); void F1();} | string | 1 | Заменить все буквы в строке на прописные символы | изменить регистр всех букв в строке |
| 2 | Заменить на пробелы знаки пре-  пинания | - |

**Задание 2.**

Выполнить задания, используя для хранения экземпляров разработанных классов стандартную коллекцию ArrayList. Во всех классах реализовать интерфейсы IComparable и IComparer. Перегрузить операции отношения для реализации сравнения объектов по указанному полю. Результат вывести на экран. Организовать вывод коллекции.

**Варианты заданий:**

1. Составить багажную ведомость камеры хранения, включив следующие данные: ФИО пассажира, количество вещей, общий вес вещей. Вывести в новый список информацию о тех пассажирах, средний вес багажа которых превышает заданный, отсортировав их по количеству вещей, сданных в камеру хранения.
2. Составить список студентов, включающий ФИО, курс, группу, дату и результат забега. Вывести в новый список информацию о студентах, показавших три лучших результата в забеге. Если окажется, что некоторые студенты получили такие же высокие результаты, то добавить их к списку победителей, отсортировать по результатам.
3. Составить инвентарную ведомость склада, включив следующие данные: вид продукции, стоимость, сорт, количество. Вывести в новый список информацию о той продукции, количество которой меньше заданной величины, отсортировав ее по количеству продукции на складе.
4. Составить список студентов группы, включив следующие данные: ФИО, номер группы, результаты сдачи трех экзаменов. Вывести в новый список информацию о студентах, успешно сдавших сессию, отсортировав по номеру группы.
5. Составить список вкладчиков банка, включив следующие данные: ФИО, № счета, сумма, дату открытия счета. Вывести в новый список информацию о тех вкладчиках, которые открыли вклад в текущем году, отсортировав их по сумме вклада.
6. Составить автомобильную ведомость, включив следующие данные: марка автомобиля, фамилия его владельца, год приобретения, пробег. Вывести в новый список информацию об автомобилях, выпущенных ранее определенного года, отсортировав их по пробегу.
7. Составить инвентарную ведомость игрушек, включив следующие данные: название игрушки, ее стоимость (в руб.), возрастные границы детей, для которых предназначена игрушка. Вывести в новый список информацию о тех игрушках, которые предназначены для детей от N до M лет, отсортировав их по стоимости.
8. Составить список студентов группы, включив следующие данные: ФИО, дату рождения, какую школу окончил. Вывести в новый список информацию о студентах, окончивших заданную школу, отсортировав их по году рождения.
9. В библиотеке имеется список книг. Каждая запись этого списка содержит фамилии авторов, название книги, год издания, цена экземпляра. Определить, имеются ли в данном списке книги, в названии которых встречается некоторое ключевое слово (например, "программирование"). Если имеются, то вывести фамилии авторов, название и год издания всех таких книг, отсортировав их по году издания.
10. Составить список студентов, включающий фамилию, факультет, курс, группу, 5 оценок. Вывести в новый список информацию о тех студентах, которые имеют хотя бы одну двойку, отсортировав их по курсу.
11. Составить список больных отделения, включив следующие данные: ФИО, болезнь, дата поступления, рабочий стаж. Вывести в новый список информацию о больных, находящихся на лечении больше недели, отсортировав их по ФИО.
12. В пресс-центре выставки программных средств хранятся данные о каждом экспонате: название, автор, количество заявок на него. Вывести в новый список экспонаты, получившие больше трех заявок, отсортировав по числу заявок.

# ЛАБОРАТОРНАЯ РАБОТА № 7. СИМВОЛЫ И СТРОКИ. 4 – часа

Целью лабораторной работы является изучение строковых типов; использование строковых типов; закрепление навыков структурного программирования.

## Строковый тип string

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом С#. Ему соответствует базовый тип класса System.String библиотеки .Net. Тип string относится к ссылочным типам. Существенной особенностью данного класса является то, что каждый его объект – это неизменяемая (immutable) последовательность символов Unicode. Любое действие со строкой ведет к тому, что создается копия строки, в которой и выполняются все изменения. Исходная же строка не меняется. Такой подход к работе со строками может показаться странным, но он обусловлен необходимостью сделать работу со строками максимально быстрой и безопасной. Например, при наличии нескольких одинаковых строк CLR может хранить их по одному и тому же адресу (данный механизм называется stringinterning), экономя таким образом память.

Создать объект типа string можно несколькими способами:

1) string s; // инициализация отложена

2) strings=''кол около колокола''; //инициализация строковым литералом

3) string s=@''Привет! //символ @ сообщает конструктору string, что строку Сегодня хорошая погода!!! ''// нужно воспринимать буквально, даже если она занимает несколько строк

4) int x = 12344556; //инициализировали целочисленную переменную

strings = x.ToString(); //преобразовали ее к типу string

5) strings=new string (' ', 20); //конструктор создает строку из 20 пробелов

6) char [] a={'a', 'b', 'c', 'd', 'e'}; //создали массив символов

string v=new string (a); //создание строки из массива символов

7) char [] a={'a', 'b', 'c', 'd', 'e'}; //создание строки из части массива символов, при

string v=new string (a, 0, 2) // этом: 0 показывает с какого

// символа, 2 – сколько символов

// использовать для инициализации

Замечание. В примерах 1–4 используется неявный вызов конструктора. В примерах 5–7 конструктор вызывается явным образом через использование операции new.

С объектом типа string можно работать посимвольно, т.е. поэлементно:

static void Main(){

string a ="кол около колокола";

Console.WriteLine("Дана строка: {0}", a);

char b='о';

int k=0;

for (int x=0;x<a.Length; x++){

if (a[x]==b){

k++;

}

}

Console.WriteLine("Символ {0} содержится в ней {1} раз", b, k );

}

Однако при попытке заменить в данной строке все вхождения буквы о на букву а, ожидаемого результата мы не получим:

for (int x=0;x<a.Length; x++){

if (a[x] == b){

a[x] = c; //1

}

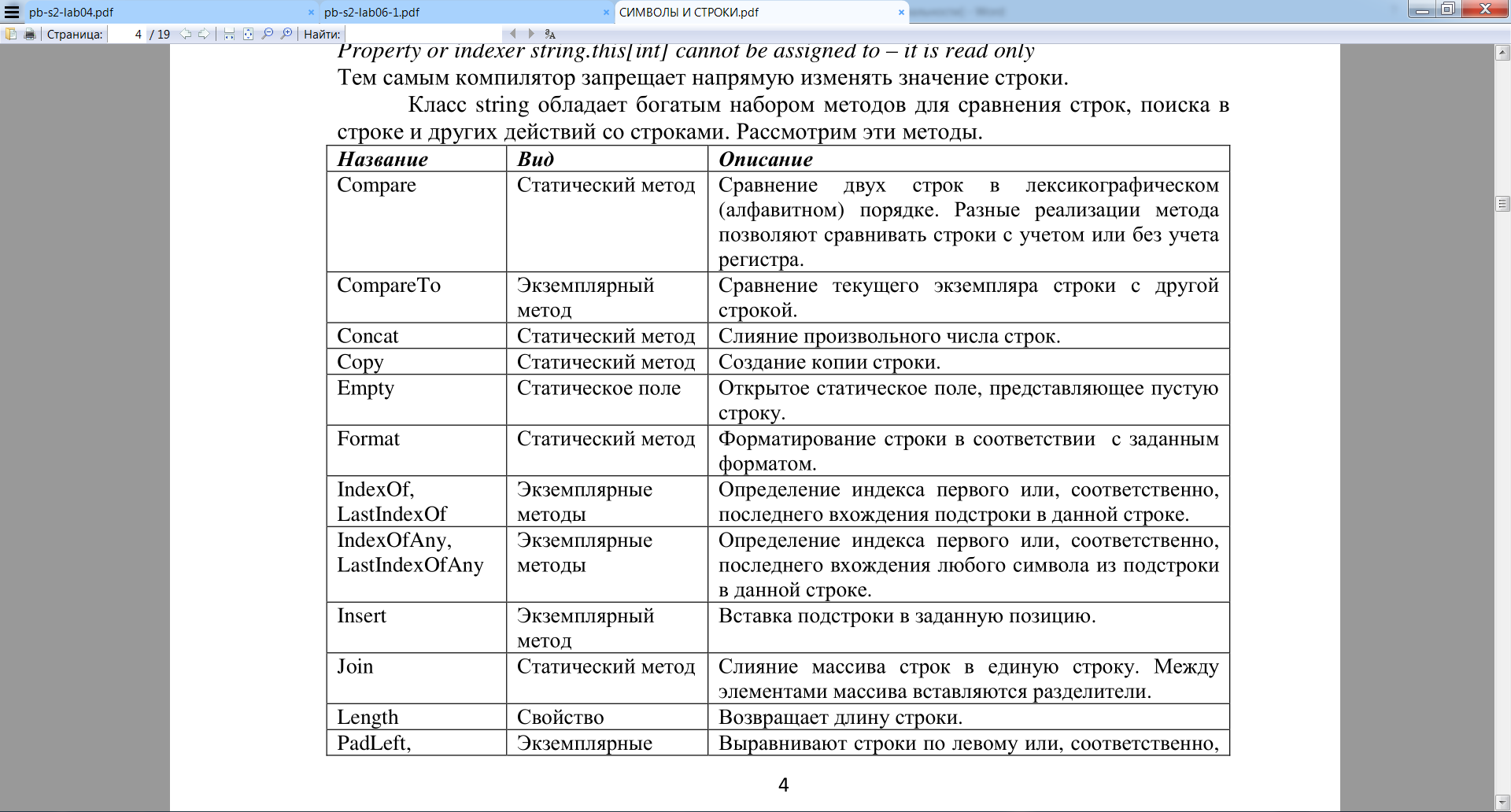
}

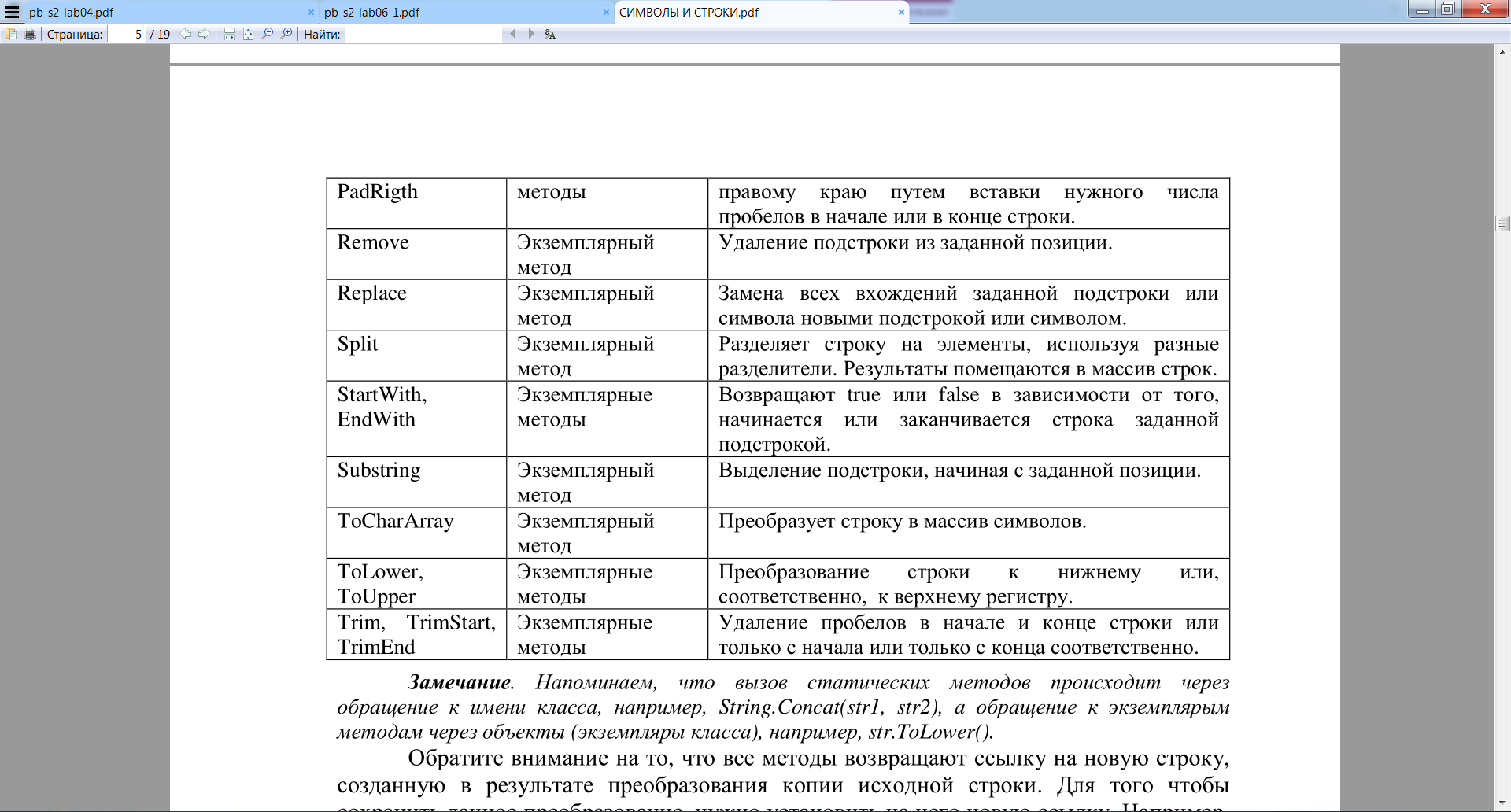
Относительно строки 1 компилятор выдаст сообщение об ошибке:

Property or indexer string.this[int] cannot be assigned to – it is read only

Тем самым компилятор запрещает напрямую изменять значение строки.

Класс string обладает богатым набором методов для сравнения строк, поиска в строке и других действий со строками. Рассмотрим эти методы.





*Замечание. Напоминаем, что вызов статических методов происходит через обращение к имени класса, например, String.Concat(str1, str2), а обращение к экземплярым методам через объекты (экземпляры класса), например, str.ToLower().*

Обратите внимание на то, что все методы возвращают ссылку на новую строку, созданную в результате преобразования копии исходной строки. Для того чтобы сохранить данное преобразование, нужно установить на него новую ссылку. Например, если выполнить следующий фрагмент программы:

string a ="кол около колокола";

Console.WriteLine("Строка а: {0}", a);

a.Remove(0,4);

Console.WriteLine("Строка a: {0}", a);

*Результат работы программы:*

Cтрока a: кол около колокола

Cтрока a: кол около колокола

то компилятор никаких сообщений не выдаст, но мы не увидим никаких преобразований со строкой.

А вот в результате работы следующего фрагмента программы мы получим следующий результат:

string a ="кол около колокола";

Console.WriteLine("Cтрока a: {0}", a);

string b=a.Remove(0,4);

Console.WriteLine("Cтрока a: {0}", a);

Console.WriteLine("Cтрока b: {0}", b);

*Результат работы программы:*

Строка а: кол около колокола

Строка а: кол около колокола

Строка b: около колокола

Результат выполнения метода Remove, можно записать и в саму переменную a:

string a ="кол около колокола";

Console.WriteLine("Cтрока a: {0}", a);

а=a.Remove(0,4);

Console.WriteLine("Cтрока a: {0}", a);

*Результат работы программы:*

Строка а: кол около колокола

Строка а: около колокола

В этом случае будет потеряна ссылка на исходное строковое значение "кол

около колокола", хотя оно и будет занимать память. Освободить занятую память сможет только сборщик мусора.

Рассмотрим следующий фрагмент программы:

string a="";

for (int i = 1; i <= 100; i++)

{

a +="!";

}

Console.WriteLine(a);

В этом случае в памяти компьютера будет сформировано 100 различных строк вида:

!

!!

!!!

…

!!!...!!

И только на последнюю из них будет ссылаться переменная а. Ссылки на все остальные строчки будут потеряны, но как и в предыдущем примере эти строки будут храниться в памяти компьютера и засорять ее. Боротся с таким засорением придется сборщику мусора, что будет сказываться на производительности программы.

Рассмотренные примеры определяют область применения типа string – это поиск, сравнение, извлечение информации из строки. А вот если нужно изменять строку, то лучше пользоваться классом StringBuilder, который мы рассмотрим позже.

Вернемся к методам класса string и рассмотрим их на следующем примере:

class Program{

static void Main(){

string str1 ="Первая строка";

string str2 = string.Copy(str1);

string str3 = "Вторая строка";

string str4 = "ВТОРАЯ строка";

stringstrUp, strLow;

int result, idx;

Console.WriteLine("str1: {0}", str1);

Console.WriteLine("Длина строки str1: {0}",str1.Length);

Console.WriteLine("str2: {0}", str2);

Console.WriteLine("Длина строки str2: {0}",str2.Length);

Console.WriteLine("str3: {0}", str3);

Console.WriteLine("Длина строки str3: {0}",str3.Length);

Console.WriteLine("str4: {0}", str4);

Console.WriteLine("Длина строки str4: {0}",str4.Length);

Console.WriteLine();

// создаем прописную и строчную версии строки str1.

strLow = str1.ToLower();

strUp = str1.ToUpper();

Console.WriteLine("Строчная версия строки str1: {0}",strLow);

Console.WriteLine("Прописная версия строки str1: {0}",strUp);

Console.WriteLine();

// Сравниваем строки,

result = str1.CompareTo(str3);

if (result == 0)

{

Console.WriteLine("str1 и str3 равны.");

}

else

{

if (result < 0)

{

Console.WriteLine("str1 меньше, чем str3");

}

else

{

Console.WriteLine("str1 больше, чем str3");

}

}

Console.WriteLine();

//сравниваем строки без учета регистра

result = String.Compare(str3,str4,true);

if (result == 0)

{

Console.WriteLine("str3 и str4 равны без учета регистра.");

}

else

{

Console.WriteLine("str3 и str4 не равны без учета регистра.");

}

Console.WriteLine();

//сравниваем части строк

result = String.Compare(str1, 4, str2, 4, 2);

if (result == 0)

{

Console.WriteLine("часть str1 и str2 равны");

}

else

{

Console.WriteLine("часть str1 и str2 неравны");

}

Console.WriteLine();

// Поиск строк.

idx = str2.IndexOf("строка");

Console.WriteLine("Индекс первого вхождения подстроки \"строка\" в str2: {0}",idx);

idx = str2.LastIndexOf("о");

Console.WriteLine("Индекс последнего вхождения символа \'о\' в str2: {0}", idx);

Console.WriteLine();

//конкатенация – слияние строк

string str=String.Concat(str1, str2, str3, str4);

Console.WriteLine("Слияние строк: {0}",str);

Console.WriteLine();

//замена подстроки "строка" на пустую подстроку

stringnew str=str.Replace("строка","");

Console.WriteLine("Замена: {0}",newstr);

}}

Очень важными методами обработки строк, являются методы разделения строки на элементы – Split и слияние массива строк в единую строку – Join.

class Program{

static void Main(){

string poems = "тучки небесные вечные странники";

char[] div = { ' '}; //создаем массив разделителей

// Разбиваем строку на части,

string[] parts = poems.Split(div);

Console.WriteLine("Результат разбиения строки на части: ");

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

Console.WriteLine(parts[i]);

}

// собираем эти части в одну строку, в качестве разделителя

// используем символ |

string whole = String.Join(" | ", parts);

Console.WriteLine("Результат сборки: ");

Console.WriteLine(whole);

}}

*Результат работы программы:*

Результат разбиения строки на части:

тучки

небесные

вечные

странники

Результат сборки:

тучки | небесные | вечные |

странники

## Строковый тип StringBuilder

Строковый тип StringBuilder определен в пространстве имен System.Text и предназначен для создания строк, значение которых можно изменять. Объекты данного класса всегда создаются с помощью явного вызова конструктора класса, т.е. через операцию new. Создать объект класса StringBuilder возможно одним из следующих способов:

1) //создание пустой строки, размер которой по умолчанию 16 символов

StringBuilder a =new StringBuilder();

2) //инициализация строки и выделение памяти под 4 символа

StringBuilder b = new StringBuilder("abcd");

3) //создание пустой строки и выделение памяти под 100 символов

StringBuilder с = new StringBuilder(100);

4) //инициализация строки и выделение памяти под 100 символов

StringBuilder d = new StringBuilder("abcd", 100);

5) //инициализация подстрокой "bcd", и выделение памяти под 100 символов

StringBuilder d = new StringBuilder("abcdefg", 1, 3,100);

С объектами класса StringBuilder можно работать посимвольно:

using System;

//подключили пространство имен для работы с классом StringBuilder

using System.Text;

namespace Example{

class Program{

static void Main(){

StringBuilder a = new StringBuilder("кол около колокола");

Console.WriteLine("Дана строка: {0}", a);

char b='о';

int k=0;

for (int x=0;x<a.Length; x++)

{

if (a[x]==b)

{

k++;

}

}

Console.WriteLine("Символ {0} содержится в ней {1} раз", b,k);

}}}

*Результат работы программы:*

Дана строка: кол около колокола

Символ о содержится в ней 7 раз

Результат работы программы ничем не отличается от аналогичного примера для работы с типом string. А вот если теперь мы попытаемся заменить в данной строке вес буквы o на точки, то такая проблема, как со строкой типа string, уже не возникнет:

static void Main(){

StringBuilder a = new StringBuilder("кол около колокола");

Console.WriteLine("Дана строка: {0}", a);

char b='о';

char c='.';

for (int x=0;x<a.Length; x++){

if (a[x]==b)

{

a[x]=c;

}

}

Console.WriteLine("Преобразованная строка: {0}",a );

}

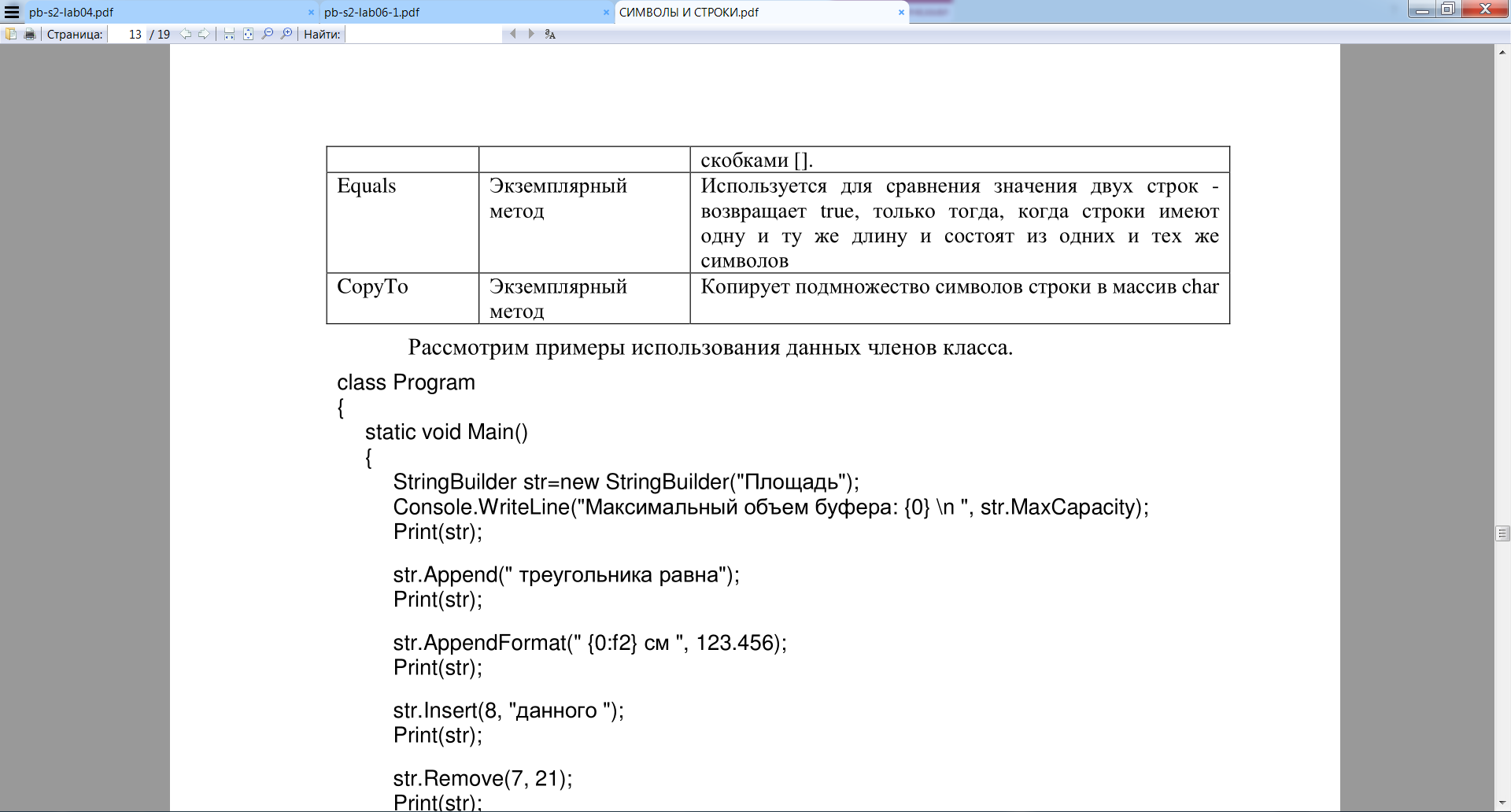
*Результат работы программы:*

Дана строка: кол около колокола

Преобразованная строка: к.л .к.л. к.л.к.ла

Для класса StringBuilder реализованы собственные члены, которые приведены в следующей таблице:





Рассмотрим примеры использования данных членов класса.

class Program{

static void Main(){

StringBuilder str=new StringBuilder("Площадь");

Console.WriteLine("Максимальный объем буфера: {0} \n ",

str.MaxCapacity);

Print(str);

str.Append(" треугольника равна");

Print(str);

str.AppendFormat(" {0:f2} см ", 123.456);

Print(str);

str.Insert(8, "данного ");

Print(str);

str.Remove(7, 21);

Print(str);

str.Replace("а", "…");

Print(str);

str.Length=0;

Print(str);

}

static void Print(StringBuilder a){

Console.WriteLine("Строка: {0} ", a);

Console.WriteLine("Текущая длина строки: {0} ", a.Length);

Console.WriteLine("Объем буфера: {0} ", a.Capacity);

Console.WriteLine();

}}

*Результат работы программы:*

Максимальный объем буфера: 2147483647

Строка: Площадь

Текущая длина строки: 7

Объем буфера: 16

Строка: Площадь треугольника равна

Текущая длина строки: 26

Объем буфера: 32

Строка: Площадь треугольника равна 123,46 см

Текущая длина строки: 37

Объем буфера: 64

Строка: Площадь данного треугольника равна 123,46 см

Текущая длина строки: 45

Объем буфера: 64

Строка: Площадь равна 123,46 см

Текущая длина строки: 24

Объем буфера: 64

Строка: Площ…дь р…вн… 123,46 см

Текущая длина строки: 30

Объем буфера: 64

Строка:

Текущая длина строки: 0

Объем буфера: 64

Все выполняемые действия относились только к одному объекту str. Никаких дополнительных объектов не создавалось. Таким образом, класс StringBuilder применяется тогда, когда необходимо модифицировать исходную строку. Следует обратить внимание на то, что при увеличении текущей длины строки возможно изменение объема буфера, отводимого для хранения значения строки. А именно, если длина строки превышает объем буфера, то он увеличивается в два раза. Обратное не верно, т.е. при уменьшении длины строки буфер остается неизменным.

На практике часто комбинируют работу с изменяемыми и неизменяемыми строками. В качестве примера рассмотрим решение следующей задачи: дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания. Вывести все слова сообщения, которые начинаются и заканчиваются на одну и ту же букву.

class Program{

static void Main(){

Console.WriteLine("Введите строку: ");

StringBuilder a = new StringBuilder(Console.ReadLine());

for (inti=0; i<a.Length;) //удаляем из строк все знаки

//пунктуации

{

if (char.IsPunctuation(a[i]))

{

a.Remove(i,1);

}

else ++i;

}

//преобразуем объект StringBuilder к типу string, и

//разбиваем его на массив слов

string []s=a.ToString().Split(' ');

Console.WriteLine("Искомые слова: ");

//перебираем все слова в массиве слов и выводим на экран те,

//которые начинаются и заканчиваются на одну и туже букву

foreach (string str in s)

{

if (str[0]==str[str.Length-1])

{

Console.WriteLine(str);

}

}}

*Результат работы программы:*

Введите строку:

кол около колокола, а шалаш у ручья.

искомые слова:

около

а

шалаш

у

## Сравнение классов String и StringBuilder

Основное отличие классов String и StringBuilder заключается в том, что при создании строки типа String выделяется ровно столько памяти, сколько необходимо для хранения инициализирующего значения. Если создается строка как объект класса StringBuilder, то выделение памяти происходит с некоторым запасом. По умолчанию, под каждую строку выделяется объем памяти, равный минимальной степени двойки, необходимой для хранения инициализирующего значения, хотя возможно задать эту величину по своему усмотрению. Например, для инициализирующего значения ”это текст” под строку типа String будет выделена память под 9 символов, а под строку типа StringBuilder – под 16 символов, из которых 9 будут использованы непосредственно для хранения данных, а еще 7 – составят запас, который можно будет использовать в случае необходимости.

Если разработчик решит добавить в строку типа StringBuilder еще один символ, например, точку, то новая строка создаваться не будет. Вместо этого будет изменен уже существующий символ, находящийся в конце используемой части строки.

Аналогичные действия будут выполнены при удалении и изменении строки. Если объем добавлений превысит объем созданного запаса, произойдет создание новой строки, длина которой будет в два раза больше, чем длина предыдущей, в нее будет скопировано содержимое старой строки, и добавление продолжится.

У строки типа String такого запас нет, поэтому при попытке добавить новый символ, будет создаваться новая строка требуемой размерности.

Существование запаса памяти привело к тому, что у любого объекта класса StringBuilder есть два свойства, отвечающих за длину строки.

• cвойство Length действует по аналогии со свойством Length класса String и возвращает длину хранящихся в объекте текстовых данных.

• cвойство Capacity возвращает реальный объем, который занимает в памяти объект класса StringBuilder (без служебной информации, разумеется).

**Контрольные вопросы:**

1. Перечислите способы создания объектов типа string.

2. Перечислите операции над строками.

3. В чём особенность операции индексирования для строк?

4. В чём отличия и в чём сходство строк и массивов типа char[ ]?

5. Как выполняется операция присваивания для строк?

6. Какие операции сравнения применимы к строкам?

7. Перечислите особенности конкатенации строк со значениями других типов.

8. В каких случаях метод ToString() вызывается неявно?

9. Каково значение свойства Length для регулярного строкового литерала, содержащего эскейп-последовательности?

10. Как выполняется сравнение строк?

11. Как выполняется метод Join()?

12. Как выполняется метод Split()?

13. Какой тип должна иметь переменная цикла foreach, при-

меняемого к строке?

14. Как инициализировать массив строк?

15. Перечислите особенности использования строковых объектов класса System.String.

16. В чем заключается различие между строковыми объектами классов String и StringBuilder?

17. Объясните назначение класса StringBuilder.

18. В каких случаях рекомендуется использовать строковые объекты класса StringBuilder?

19. В чем заключается главная особенность строковые объекты класса StringBuilder?

20. Перечислите основные методы и свойства класса StringBuilder.

## 7.4. Индивидуальные задания

**Задание 1.**

Замечание. При решении задач использовать тип string.

Разработать программу, которая для заданной строки s:

1. подсчитывает общее число вхождений символов х и y;

2. определяет, какой из двух заданных символов встречается чаще в строке;

3. выводит на экран символы, которые наиболее часто встречается в строке;

4. выводит на экран символы, которые встречаются в строке только один раз;

5. определяет, является ли строка палиндромом;

6. определяет, упорядочены ли по алфавиту символы строки;

7. подсчитывает количество цифр в строке;

8. подсчитывает сумму всех содержащихся в ней цифр;

9. выводит на экран последовательность символов, расположенных до первого двоеточия;

10. выводит на экран последовательность символов, расположенных после последнего двоеточия

**Задание 2.**

Замечание. При решении задач использовать класс StringBuilder.

Разработать программу, которая для заданной строки s:

1. вставляет символ x после каждого вхождения символа y;

2. удваивает каждое вхождение заданной подстроки x;

3. удаляет все символы х;

4. удаляет все подстроки substr;

5. заменяет все вхождения подстроки str1 на подстроку str2 (при этом str1 может являться частью str2);

6. заменяет все группы стоящих рядом точек на многоточие;

7. меняет местами первую букву со второй, третью с четвертой и т.д.

8. меняет местами первую букву с последней, вторую с предпоследней и т.д.

9. удаляет из строки все подстроки, состоящие из цифр;

10. удаляет из строки самую длинную подстроку, состоящую из повторяющегося символа.

**Задание 3.**

Замечание. При решении задач из данного пункта использовать тип string или StringBuilder. Свой выбор обосновать.

Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания.

1. Вывести только те слова сообщения, которые содержат не более чем n букв.

2. Вывести только те слова сообщения, которые содержат хотя бы одну цифру.

3. Удалить из сообщения все слова, которые заканчиваются на заданный символ.

4. Удалить из сообщения все повторяющиеся слова (без учета регистра).

5. Подсчитать сколько раз заданное слово встречается в сообщении.

6. Найти самое длинное слово сообщения.

7. Найти все самые короткие слова сообщения.

8. Вывести на экран все слова-палиндромы, содержащиеся в сообщении.

9. Вывести слова сообщения в алфавитном порядке.

10. Вывести слова сообщения в порядке возрастания их длин.

# ЛАБОРАТОРНАЯ РАБОТА № 8. ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ. 4 – часа

Цель работы: получение навыков использования механизма обработки исключительных ситуаций в прикладных программах.

## Исключительные ситуации

Исключение представляет собой ошибку, происходящую во время выполнения программы. С помощью подсистемы обработки исключений можно обрабатывать эти ошибки, не вызывая аварийного завершения программы. Обработка исключений в С# выполняется с применением следующих ключевых слов: try, catch, throw и finally. Данные ключевые слова взаимосвязаны, так как использование одного из них

влечет за собой использование других. При обработке исключений используются блоки try и catch.

Синтаксис:

try

{

//Блок кода\_в котором контролируется появление ошибок.

}

catch (ExcepType1 ех)

{

//Обработчик исключений ExcepType1.

}

catch (ЕхсерТуре2 ех)

{

//Обработчик исключений ЕхсерТуре2.

}

Основные системные исключения приведены ниже в таблице.

|  |  |
| --- | --- |
| Исключение | Значение |
| FormatException | Некорректный формат операнда или аргумента (при передаче в метод) |
| DivideByZeroException | Предпринята попытка деления на ноль |
| IndexOutOfRangeException | Индекс массива выходит за пределы диапазона массива |
| InvalidCastException | Некорректное преобразование в процессе выполнения |
| OutOfMemoryException | Вызов new был неудачным из-за недостатка памяти |
| OverflowException | Переполнение при выполнении арифметической операции |

Тип исключения в обработчике catch должен соответствовать типу перехватываемого исключения. Не перехваченное исключение приводит к аварийному завершению программы. Для перехвата исключений вне зависимости от их типа (перехват всех исключений) используется оператор catch без параметров. Если исключение не произошло, то блоки catch для данного блока try при выполнении пропускаются.

**Возврат из исключения**

После выполнения блока catch управление не передается обратно оператору программы, в котором возникло исключение. Выполнение программы продолжается с операторов, следующих после блока catch. Можно указать блок кода, который вызывается после выхода из обработчика catch. Для этого необходимо реализовать блок finally в конце последовательности блоков try/catch. Общая форма блоков try/catch, включающей finally следующая:

try

{

// Блок\_кода\_\_контролирующий появление \_ошибок .

}

catch (ExcepType1 ех)

{

// Обработка исключения ExcepType1.

}

catch (ЕхсерТуре2 ех)

{

// Обработка исключения ЕхсерТуре2.

}

finally

{

// Код блока finally.

}

Блок finally выполняется всегда независимо от того, возникло исключение или нет, и независимо от причины возникновения исключения.

**Пример 1.** Написать программу, в которой обрабатываются ошибка деления на ноль и ошибка некорректного ввода числа в консоль.

class Program{

static int MyDel(int x, int y){

return x / y;

}

static void Main(){

try{

Console.Write("Введите x: ");

int x = int.Parse(Console.ReadLine());

Console.Write("Введите y: ");

int y = int.Parse(Console.ReadLine());

int result = MyDel(x, y);

Console.WriteLine("Результат: " + result);

}

// Обрабатываем исключение возникающее при делении на ноль

catch (DivideByZeroException ex){

Console.WriteLine("Деление на 0 detected!!!\n");

Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");

Main();

}

//Обрабатываем исключение при некорректном вводе числа в

//консоль

catch (FormatException ex){

Console.WriteLine("Это НЕ число!!!\n");

Console.WriteLine("ОШИБКА: " + ex.Message + "\n\n");

Main();

}

Console.ReadLine();

}}

**Генерация исключений пользователем**

Как правило, система генерирует исключения автоматически при определенных условиях. Также исключение может быть создано пользователем и вызвано на выполнение с помощью оператора throw. Синтаксис оператора:

throw exceptOb;

где в качестве exceptOb должен быть обозначен объект класса исключений, производного от класса Exception.

**Пример 2.** Написать программу в которой происходит ввод строки, и если длина строки больше 6 символов, возникает исключение.

static void Main(string[] args){

try{

Console.Write("Введите строку: ");

string message = Console.ReadLine();

if (message.Length > 6){

throw new Exception("Длина строки больше 6 символов");

}

}

catch (Exception e){

Console.WriteLine($"Ошибка: {e.Message}");

}

Console.Read();

}

После оператора throw указывается объект исключения, через конструктор которого передается сообщение об ошибке. Вместо типа Exception можно использовать объект любого другого типа исключений. Затем в блоке catch сгенерированное исключение обрабатывается.

Исключение, обработанное одним оператором catch, может быть сгенерировано повторно, и оно может перехватываться другим внешним оператором catch. Для этого используется оператор throw без типа исключения.

**Пример 3.**

try{

try{

Console.Write("Введите строку: ");

string message = Console.ReadLine();

if (message.Length > 6){

throw new Exception("Длина строки больше 6 символов");

}

}

catch{

Console.WriteLine("Возникло исключение");

throw;

}}

catch (Exception ex){

Console.WriteLine(ex.Message);

}

В данном примере при вводе строки с длиной больше 6 символов возникнет исключение, которое будет обработано внутренним блоком catch. Однако поскольку в этом блоке используется оператор throw, то исключение будет передано дальше внешнему блоку catch.

**Пользовательские классы исключений**

Можно создавать свои собственные классы исключений, выполняющие обработку нестандартных ошибок в программе. Для этого необходимо объявить производный класс, который наследует класс Exception. Создаваемые пользователем классы исключений автоматически получают доступ к свойствам и методам, определенным в классе Exception.

**Пример 4.**

//Класс, для описания пользовательского типа ошибок

class PersonException : Exception //Используем наследование

{

//Принимает сообщение с описание ошибки, и код ошибки

public PersonException(string aMessage, int aCode)

: base(aMessage) //Вызываем конструктор базового класса

{

errorCode = aCode;

}

//Возвращает код ошибки

public int ErrorCode { get { return errorCode; } }

//Код ошибки

private int errorCode;

}

В этом примере создан класс наследник класса «Exception» с полем «errorCode», которое хранит код ошибки. Так же, добавлено свойство, возвращающее этот код, и создан конструктор, который принимает описание ошибки и её код. Причем, в конструкторе, вызывается конструктор базового класса, которому передается сообщение об ошибке. Сгенерировать исключение такого типа в конструкторе класса «Person» можно так:

//Конструктор

public Person(string aName, int anAge)

{

name = aName;

//Если указан отрицательный возраст

if (anAge < 0)

{

throw new PersonException("Отрицательный возраст", 20);

//Дальше, конструктор выполняться не будет...

}

age = anAge;

}

А обрабатывать так:

try

{

Person somePerson = new Person("Иван", -21);

}

catch (PersonException ex)

{

//Обработка ошибок

Console.WriteLine("Произошла ошибка: " + ex.Message + "; с

кодом: " + ex.ErrorCode);

}

В блоке catch явно указано что он обрабатывает исключения типа «PersonException» и исключения других типов в него не попадут.

**Контрольные вопросы**

1. Что такое «исключительная ситуация (исключение)»?

2. Для чего предназначен механизм исключений в языке C#?

3. Какие операторы используются для обработки исключений? Для чего используется ключевое слово try?

4. Проиллюстрируйте обработку исключительной ситуации фрагментом программы на языке C#.

5. Перечислите основные системные исключения.

6. Требуется ли соблюдение соответствия типа перехватываемого исключения типу исключения в обработчике catch?

7. Как реализуется перехват всех исключений?

8. Что происходит если исключение не обработано?

9. Как выполняется оператор catch без параметров?

10. Может ли быть в программе блок try без блока catch?

11. Куда передается управление после обработки исключительной ситуации?

12. Проиллюстрируйте использование блока finally и обработку исключительной ситуации фрагментом программы на языке C#.

13. Выполняется ли блок finally, если исключение не возникло?

14. Какие причины автоматически генерируют исключения?

15. Как осуществляется явная генерация исключения?

16. Какой оператор используется для явной генерации исключения?

17. Проиллюстрируйте использование оператора throw фрагментом программы на языке C#.

18. Как можно повторно перехватить исключение в программе?

19. Можно ли создавать собственные классы исключений для обработки ошибок в программе пользователя?

20. Какой системный класс является базовым для создания классов исключений пользователя?

21. Как обеспечивается обращение к свойствам и методам системных классов исключений?

22. Получают ли создаваемые пользователем классы исключений доступ к свойствам и методам, определенным в классе Exception?

23. Когда возникает исключение?

24. Для чего предназначен блок catch?

25. Как завершится программа, если возникло исключение, для которого нет соответствующего обработчика?

26. Если в блоке try не возникло исключение, куда передается управление после того, как блок try завершит работу?

27. Что произойдет, если исключение возникло вне блока try?

28. Укажите основной недостаток использования catch (...)?

29. Может ли программа продолжить свое выполнение после возникновения исключительной ситуации?

## Индивидуальные задания

Во всех заданиях реализуемые функции или методы должны генерировать подходящие исключения. Обработку исключений нужно выполнять главной функцией, которая должна демонстрировать обработку всех перехватываемых исключений.

Функции, реализуемые в заданиях, обязаны выполнять проверку передаваемых параметров и генерировать исключение в случае ошибочных. Все функции реализуются в четырех вариантах:

* без спецификации исключений;
* со спецификацией throw();
* с конкретной спецификацией с подходящим стандартным исключением;
* спецификация с собственным реализованным исключением.

Собственное исключение должно быть реализовано в трех вариантах: как пустой класс, как независимый класс с полями-параметрами функции, как наследник от стандартного исключения с полями. Перехват и обработку исключений должна выполнять главная функция.

1. Функция вычисляет площадь треугольника по трем сторонам

 где *р* = (*а* + *b* + *с*) / 2.

1. Функция вычисляет корень линейного уравнения *ах* + *b* = 0.
2. Функция вычисляет периметр треугольника.
3. Функция переводит часы и минуты в секунды.
4. Функция вычисляет корень квадратного уравнения *ах* + *bх* + *с* = 0.
5. Функция вычисляет сумму геометрической профессии  
    *Sn* = (*a*0 − *аnr*) / (1− *r*).
6. Функция выполняет деление комплексных чисел A и В. Комплексные числа представлены структурой-парой.
7. Функция вычисляет целую часть неправильной дроби, представленной числителем и знаменателем − целыми числами.
8. Функция переводит комплексное число *z* = *х* + *i*⋅*у* из алгебраической формы в тригонометрическую *z* = *radius*(cos(*angle*) + *i*⋅sin(*angle*)). Комплексное число представлено структурой-парой. Преобразованное число тоже представляется структурой-парой *(radius*, *angle*):



1. Функция вычисляет разность между двумя датами в днях. Даты представлены структурой с тремя полями: год, месяц, день.
2. Функция вычисляет продолжительность телефонного разговора в минутах, принимая время начала и окончания. Время представлено структурой с тре­мя полями: час, минута, секунда. Неполная минута считается за полную.
3. Функция вычисляет день недели по дате. Даты представлены структурой с тремя полями: год, месяц, день. Первое января считается понедельником.
4. Функция вычисляет углы прямоугольного треугольника. В качестве параметров передаются катеты А и В. (Синус угла А1, противолежащего катету А, вы­числяется по формуле sin (Al) = *а* / *с*, где *с* − гипотенуза треугольника.)
5. Функция проверяет, является ли передаваемая строка палиндромом.
6. Функция определяет, существуют ли прямые A1*x* + В1*у* + С1 = 0 и А2*х* + + В2*у* + С2 = 0, если выражение *d* = А1В2 − А2В1 не равно нулю. Прямые зада­ются структурой с тремя полями.
7. Функция вычисляет расстояние между двумя точками P1(*x*1, *y*1) и

Р2(*x*2, *y*2) по формуле



Исключение генерируется, когда P1 и Р2 − одна и та же точка.

1. Функция вычисляет расстояние от точки Р(*х*1, *у*1) до прямой A*x* + B*y* + С = 0 по формуле



1. Функция выясняет, является ли год високосный. Високосность определяетсяследующим образом: если номер года не делится на 100, то високосным считается тот, который делится на 4 без остатка; если номер года делится на 100,то номер високосного года делится на 400 без остатка.

# ЛАБОРАТОРНАЯ РАБОТА № 9. ДЕЛЕГАТЫ. СОБЫТИЯ. 4 – часа

Цель работы: овладение основными приемами событийного программирования и их программной реализацией.

## Делегаты

Делегат *–* это вид класса, предназначенный для хранения ссылок на методы. Делегат представляет собой тип данных. Его базовым классом является класс System.Delegate. Наследовать от делегата нельзя. Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса.

Описание делегата задает сигнатуру методов, которые могут быть вызваны с его помощью:

[атрибуты] [спецификаторы] delegate <тип>

<имя\_делегата>([параметры])

Спецификаторыделегата имеют тот же смысл, что и для класса, причем допускаются только спецификаторы new, public, protected, internal и private.

Типописывает возвращаемое значение методов, вызываемых с помощью делегата, а необязательными параметрамиделегата являются параметры этих методов.

Делегат может хранить ссылки на несколько методов и вызывать их по очереди.

Пример описания делегата:

public delegate void D(int i);

Делегат может вызывать либо метод экземпляра класса, связанный с объектом, или статический метод, связанный с классом.

При реализации делегата создается ссылка типа делегата и ей присваивается ссылка на метод, который передается делегату в качестве параметра, причем используется только имя метода (параметры не указываются). Затем этот метод вызывается посредством экземпляра делегата.

Таким образом, вызов экземпляра делегата трансформируется в обращение к методу, на который он ссылается при вызове. И, следовательно, решение о вызываемом методе принимается во время выполнения программы, а не в период компиляции.

**Многоадресатная передача**

Многоадресатная передача – это способность создавать список вызовов (или цепочку вызовов) методов, которые должны автоматически вызываться при вызове делегата. Для этого достаточно создать экземпляр делегата, а затем для добавления методов в эту цепочку использовать оператор "+=". Для удаления метода из цепочки используется оператор " -= ".

Делегат с многоадресатной передачей имеет одно ограничение: он должен возвращать тип void.

**Пример 1.**

// Демонстрация использования многоадресатной передачи

using System;

delegate void strMod(ref string str); // Объявляем делегат

class SOps{

// Метод заменяет пробелы дефисами

static void rs(ref string a){

a = a.Replace(' ', '–');

Console.WriteLine("Замена пробелов дефисами." + a);

}

// Метод удаляет символ т.

static void ms(ref string a){

string temp = "";

temp = a.Replace("т", "");

a = temp;

Console.WriteLine("Удаление т ." + a);

}}

public static void Main(){

// Создаем экземпляры делегатов

strMod strOp;

strMod rSp = new strMod(rs);

strMod mSp = new strMod(ms);

string str = "Это простой тест.";

// Организация многоадресатной передачи

strOp = rSp;

strOp += mSp;

// Вызов делегата с многоадресатной передачей

strOp(ref str);

Console.WriteLine("Результирующая строка: " + str);

Console.WriteLine();

str = "Это простой тест."; // Восстановление исходной строки

// Вызов делегата с многоадресатной передачей

strOp(ref str);

Console.WriteLine("Результирующая строка: " + str);

Console.WriteLine();

}

## События

На основе делегатов построено важное средство С#: событие. Синтаксис события:

[атрибуты][спецификаторы]event <тип> <имя>

Для событий применяются спецификаторы new, public, protected, internal, private, static, virtual, sealed, override, abstract и extern. Событие может быть статическим (static), тогда оно связано с классом в целом, или обычным – в этом случае оно связано с экземпляром класса.

Тип события – это тип делегата, на котором основано событие. Пример описания делегата и соответствующего ему события:

public delegate void Del(object о); //объявление делегата

class А

{public event Del Oops; //объявление события

}

События работают следующим образом. Объект, которому необходима информация о некотором событии, регистрирует обработчик для этого события, сигнатура которых соответствует типу делегата. Когда ожидаемое событие происходит, вызываются все зарегистрированные обработчики.

Если в качестве обработчика используется статический метод, уведомление о событии применяется к классу (и неявно ко всем объектам этого класса). Если же в качестве обработчика событий используется метод экземпляра класса, события посылаются к конкретным экземплярам этого класса.

Добавлять обработчики в список или удалять их можно с помощью операторов "+=" и "-=".

Внутри класса, в котором описано событие, с ним можно обращаться, как с обычным полем, имеющим тип делегата: использовать операции отношения, присваивания и т. д. Значение события по умолчанию – null.

Подобно делегатам события могут предназначаться для многоадресатной передачи. В этом случае на одно уведомление о событии может отвечать несколько объектов.

**Пример 2.**

// Демонстрация использования события, предназначенного

// для многоадресатной передачи.

using System;

// Объявляем делегат для события,

delegate void Del();

// Объявляем класс события,

class Et{

public event Del SE;

// Этот метод вызывается для генерирования события,

public void OnSE(){

if(SE != null) SE();

}}

class X{

public void Xh(){

Console.WriteLine("Событие, полученное объектом X.");}

}

class Y{

public void Yh(){

Console.WriteLine("Событие, полученное объектом Y.);

}}

class EventDemo{

static void h(){

Console.WriteLine("Событие, полученное классом EventDemo");

}

public static void Main(){

Et evt = new Et();

X xOb = new X();

Y yOb = new Y();

// Добавляем обработчики в список события,

evt.SE += new Del(h);

evt.SE += new Del(xOb.Xh);

evt.SE += new Del(yOb.Yh);

// Генерируем событие,

evt.OnSE();

Console.WriteLine();

// Удаляем один обработчик.

evt.SE -= new Del(xOb.Xh);

evt.OnSE();

}}

В библиотеке .NET описано огромное число стандартных делегатов, предназначенных для реализации механизма обработки событий. Большинство этих классов оформлено по одним и тем же правилам:

– имя делегата заканчивается суффиксом EventHandler;

– делегат получает два параметра: первый параметр задает источник события и имеет тип object; второй параметр задает аргументы события и имеет тип EventArgs или производный от него.

Если обработчикам события требуется специфическая информация о событии, то для этого создают класс, производный от стандартного класса EventArgs, и добавляют в него необходимую информацию. Если делегат не использует такую информацию, можно обойтись стандартным классом делегата System.EventHandler.

Имя обработчика события принято составлять из префикса On и имени события.

**Контрольные вопросы**

1. Что такое «делегат»?

2. В чем состоит преимущество использования делегатов?

3. Когда осуществляется выбор вызываемого метода в случае использования делегата?

4. Что является значением делегата?

5. Какое ключевое слово используется для описания делегата?

6. Проиллюстрируйте использование делегата фрагментом программы на языке C#.

7. Должна ли сигнатура вызываемого метода, соответствовать сигнатуре делегата?

8. Возможен ли вызов метода в том случае, если его сигнатура не соответствует сигнатуре делегата?

9. Что такое «многоадресный делегат»?

10. Как можно создать последовательность методов для многоадресных делегатов?

11. Какие операторы используются для создания последовательности методов для многоадресных делегатов?

12. Как можно удалить элемент из последовательности методов для многоадресных делегатов?

13. Каким должен быть тип возвращаемого значения для многоадресных делегатов?

14 Как можно подписаться на событие?

15. Каким образом вызывается метод подписавшийся на событие? Он всегда работает пока ждёт события?

16. Что такое событие в языке C# ?

17. Объясните синтаксис оператора посылки сообщения.

18. Приведите формат объявления события.

19. Что такое переменная события?

20. Что определяет делегат, указанный в объявлении события?

21. Какие действия предусматривает подписка на события?

22. Назовите этапы работы с событиями.

## Индивидуальные задания

**Задание 1.**

Разработать проект использования делегата для:

–вызова разных методов одним экземпляром делегата;

–многоадресной передачи.

В методах предусмотреть вывод результата.

### Варианты заданий:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Параметры делегата** | **Методы возвращают:** | | |
| **класс1** | | **класс2** |
| метод\_1 (ста-  тич.) | метод\_2 (экз.) |
| 1. | double |  | h2+5 | ep |
| 2. | double | x2 | 15/y | n3 |
| 3. | char | символ, преобразо- ванный в нижний регистр | цифру ‘5’, если сим- вол – буква | \*, если символ  – буква |
| 4. | string | строку, удалив два первых символа | строку, заменив пер- вый символ символом  ‘**-‘** | строку, удалив два последних  символа |
| 5. | char,bool | определяет, являет- ся ли символ (код символа) цифрой | определяет, содер- жится ли значение кода символа в диа- пазоне кодов ASCII | определяет, яв- ляется ли сим- вол знаком препинания |
| 6. | int, string | удвоенную строку с удаленным k-м символом | строку, заменив  s-ый символ знаком ‘+’ | подстроку, на- чиная с h-ой позиции |
| 7. | double | sin(r) | n+2 | 2/y\*ln(y) |
| 8. | int, int | n\*u2 | 2\*w-t | |x|-k |
| 9. | double | 1/ep | x/10 | w-1010\*w |
| 10. | char,int | если символ является цифрой найти его числовое значение, иначе его код | вернуть код символа | если символ яв- ляется буквой ‘х’, вывести число 0 |
| 11. | int,float | 7\*m/ln(2) | 7\*i\*2.6 | |k-10| |
| 12. | string | заменить все буквы в строке на про-  писные символы | изменить регистр всех букв в строке | заменить на пробелы знаки  препинания |

**Задание 2.**

Создать событие на основе делегата (задание 1). В тестирующем классе организовать цепочку вызовов.

# ЛАБОРАТОРНАЯ РАБОТА № 10. ФАЙЛЫ. 2 – часа

Цель работы: освоить способы работы с файловыми потоками через основные классы для работы с файлами.

## Файловый ввод-вывод

Основными классами для работы с файлами и потоками в С# являются File, FileStream и StreamReader StreamWriter. Класс File предназначен для создания, открытия, удаления, изменения атрибутов файла.

Выполнять обмен с внешними устройствами можно на уровне:

* двоичного представления данных (BinaryReader, BinaryWriter);
* байтов (FileStream);
* текста, то есть символов (StreamWriter, StreamReader).

Рассмотрим простейшие приёмы работы с файловыми потоками.

Класс FileStream – представляет байтовый поток, разработанный для выполнения операции ввода/вывода в файл.

Использование классов потоков в программе предполагает следующие операции:

– создание потока и связывание его с физическим файлом.

– обмен (ввод/вывод).

– закрытие файла.

Каждый класс файловых потоков содержит несколько вариантов конструкторов, с помощью которых можно создавать объекты этих классов различными способами и в различных режимах. Например, файлы можно открывать только для чтения, только для записи или для чтения и записи. Эти режимы доступак файлу содержатся в перечислениях FileMode и FileAccess, определенном в пространстве имен System.I0.

Пример инициализации объекта FileStream:

FileStream myStream. = File.Open("С:\MyFile.rxt", FileMode.Open, FileAccess.Read);

В данном случае режим открытияустановлен как FileMode.Open, что означает открыть файл, если он существует; права доступа установлены FileAccess.Read, что означает возможность только читать файл. Функция Open возвращает объект типа FileStream, посредством которого в дальнейшем происходят чтение или запись в файл. Иногда удобнее работать с файлом с помощью класса StreamReader. StreamReader и StreamWriter связываются с потоком при помощи конструктора инициализации.

Для StreamWriter используйте один из следующих конструкторов:

StreamWriter(string filename) StreamWriter(string filename, bool appendFlag)

Здесь элемент filenameозначает имя открываемого файла, причем имя может включать полный путь к файлу. Во второй форме используется параметр appendFlagтипа bool: если appendFlagравен значению true, выводимые данные добавляются в конец существующего файла. В противном случае заданный файл перезаписывается.

В обоих случаях, если файл не существует, он создается, а при возникновении ошибки ввода-вывода генерируется исключение типа IOException (также возможны и другие исключения).

Для StreamReader чаще всего используется следующий конструктор:

StreamReader (string filename ) ;

Здесь элемент stream означает имя открытого потока. Этот конструктор генерирует исключение типа ArgumentNullException, если поток stream имеет null-значение, и исключение типа ArgumentException, если поток stream не открыт для ввода. После создания объект класса StreamReader автоматически преобразует байты в символы.

Можно указывать кодировку, в которой будут считываться/записываться данные при создании StreamReader/StreamWriter:

static void Main(string[] args){

FileStream file1=new FileStream("d:\\test.txt",FileMode.Open);

StreamReader reader = new StreamReader(file1, Encoding.Unicode);

StreamWriter writer = new StreamWriter(file1, Encoding.UTF8);

}

**Пример 1.** Записать в текстовый файл таблицу из двух столбцов: в первом столбце цифры от 1 до 5 с шагом 1, во втором – квадраты этих чисел. Считать файл и найти сумму квадратов чисел.

class Program{

static void Main(string[] args){

const string testFile = @"dat.data";

FileStream fs = File.Create(testFile);

using (var bw = new StreamWriter(fs)){//Запись данных в файл

bw.WriteLine("{0, 10} {1, 10}", 1, 1);

bw.WriteLine("{0, 10} {1, 10}", 2, 4);

bw.WriteLine("{0, 10} {1, 10}", 3, 9);

bw.WriteLine("{0, 10} {1, 10}", 4, 16);

bw.WriteLine("{0, 10} {1, 10}", 5, 25);

}

Console.WriteLine();

Console.WriteLine();

string[] massivStr;

double summa = 0;

using (var sr = new StreamReader(testFile)){

while (true){

String str = sr.ReadLine();//Чтение одной строки из файла

if (str == null)

break;

//Разделяем строку на элементы, используя разделитель

//пробел. Лишние пробелы удаляем

massivStr = str.Split(new char[] {' '},

StringSplitOptions.RemoveEmptyEntries);

Console.WriteLine("{0} {1}",

Convert.ToDouble(massivStr[0]),

Convert.ToDouble(massivStr[1]));

summa += Convert.ToDouble(massivStr[1]);//Сумма чисел

}}}}

**Контрольные вопросы**

1. Что такое поток данных?

2. Для чего предназначен механизм потоков данных в языке C#?

3.Какие классы иерархии потоков ввода Вы знаете?

4. Какие стандартные классы используются при консольном вводе?

5. Какие стандартные классы используются при файловом вводе?

6. В чем отличие текстового набора данных от бинарного набора?

7. Объясните механизм консольного ввода.

8. Какие классы иерархии потоков вывода Вы знаете?

9.Какие стандартные классы используются при консольном выводе?

10. Какие стандартные классы используются при файловом выводе?

11. Какая передача данных выполняется быстрее: в текстовый набор

данных или в бинарный набор данных? Объясните почему.

12. Стандартные классы, объекты файлового ввода/вывода.

13. Объясните механизм файлового ввода.

14. Объясните механизм консольного вывода.

15. Объясните механизм файлового вывода.

16. Какие операторы используются для открытия текстового потока

данных?

17. Какие операторы используются для открытия бинарного потока

данных?

## Индивидуальные задания

**Задание 1.**

Записать в текстовый файл результат расчета функции f(y). Результат должен быть записан в виде двух столбцов – аргумента и значения функции от данного аргумента. Начало и конец диапазона, имя файла, а также шаг расчета вводить с клавиатуры.

|  |  |
| --- | --- |
| Вариант | Функция |
| 1 | f(y) = y\*y |
| 2 | f(y) = y\*y\*y |
| 3 | f(y) = cos(y) |
| 4 | f(y) = sin(y) |
| 5 | f(y) = sin(y)\*cos(y) |
| 6 | f(y) = sin(y)\*y |
| 7 | f(y) = cos(y)\*y |
| 8 | f(y) = sin(y)\*cos(y)\*y |
| 9 | f(y) = sin(y)\*y\*y |
| 10 | f(y) = cos(y)\*y\*y |

**Задание 2.**

Считать файл, вывести на экран среднее арифметическое.

## ЛАБОРАТОРНАЯ РАБОТА № 11. РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЙ

Цель: разработка графического интерфейса для Windows-приложений с помощью технологии Windows Forms. Работа с диалогами открытия и сохранения файлов.

## Разработка графического интерфейса для Windows-приложений с помощью технологии Windows Forms. 2 – часа

Процесс создания Windows-приложения состоит из двух основных этапов:

– визуальное проектирование, то есть задание внешнего облика приложения;

– определение поведения приложения путем написания процедур обработки событий.

Визуальное проектирование заключается в помещении на форму компонентов (элементов управления) и задании их свойств и свойств самой формы.

Windows-приложение состоит из главной программы, обеспечивающей инициализацию и завершение приложения, цикла обработки сообщений и набора обработчиков событий.

Принципиально важным признаком Windows-приложений является то, что все они – программы, управляемые событиями.

Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу – создание шаблонов приложения и форм, заготовок обработчиков событий, организацию цикла обработки сообщений и т. д.

**Шаблон Windows-приложения**

Среда создает заготовку формы и шаблон текста приложения.

Процесс создания Windows-приложения состоит из двух основных этапов:

1. Визуальное проектирование, то есть задание внешнего облика приложения.

2. Определение поведения приложения путем написания процедур обработки событий.

Визуальное проектирование заключается в помещении на форму компонентов (элементов управления) и задании их свойств и свойств самой формы с помощью окна свойств.

Таким образом, для размещения компонента на форме необходимо выполнить три действия:

1. Создать экземпляр соответствующего класса.

2. Настроить свойства экземпляра, в том числе зарегистрировать обработчик событий.

3. Поместить экземпляр в коллекцию Controls компонентов формы.

В теле обработчиков событий программист может самостоятельно написать код, который будет выполняться при наступлении события.

**Контрольные вопросы**

1. Что понимают под термином событийно-управляемое программирова-ние?

2. Назовите основные структурные элементы Windows-приложения.

3 . Опишите процесс создания Windows-приложения с графическим интер-фейсом на основе Windows Forms.

4 . Что такое обработчик события?

5 . Как создаются обработчики событий?

6 . Опишите синтаксис обработчика событий.

7 . Какие параметры передаются в обработчик события?

8 . Перечислите основные свойства стандартных визуальных элементов управления.

9 . Каким образом можно добавить элемент управления на форму?

10. Объясните назначение класса Form.

11. Назовите основные свойства, методы, события класса Form.

12. Чем диалоговое окно отличается от обычной формы?

13. Какие виды диалоговых окон Вам известны?

14. Объясните последовательность действий, которые надо совершить для отображения диалогового окна.

15. Для чего предназначен класс Application?

16.Перечислите основные элементы класса Application.

## Индивидуальные задания

**Задание 1.**

Работа с формой.

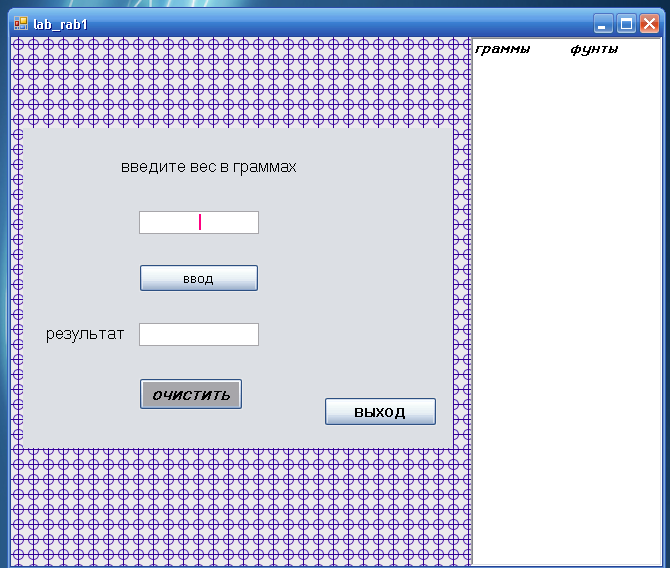


Рисунок 1. Вид окна формы.

Для создания приложения, изображенного на рисунке 1, выполните следующие действия:

1. Создайте новый проект: File/New/Project, либо New в окне Start Page.

2. В окне New project в левой части выбрать Visual C# Projects, в правой – пункт Windows Forms Application.

3. В поле Name введите имя проекта, в поле Location – место его сохранения на диске.

4. Просмотрите свойства формы, представленные на странице Properties.

5. Измените свойство Textс Formlна Лабораторная работа 9.

6. Обратите внимание на свойство Name.Это свойство определяет имя компонента, под которым компонент будет известен программе.

7. Измените свойство BackGroundImage Formlна какой-либо рисунок. Для этого в окне Select Resourceустановите переключатель Local Resource, нажмите Importи выберите файл изображения.

8. Поместите на форму компонент Panel.Осуществите прогон пустой программы. В рабочем приложении максимизируйте окно, а затем закройте его.

9.Задайтесвойству компонента BackColor Panel1значение ControlLight.

10.РазместитеPanel1в центре формы.

11.Отбуксируйтесторону компонента Panel,ухватившись за верхний обрамляющий черный квадратик. Обратите внимание на то, что это значение установилось в свойстве Heightинспектора объектов. Задайте размер Size430; 320.

12. Установите на панель Panel1 метку Labei1.Свойству Textпридайте значение «Введите значение веса в граммах». В свойстве Fontраскройте диалоговое окно настройки шрифта и измените высоту шрифта – 12 пт.

13. Установите на панель Panel1редактор TextBox1.Очистите свойство Text*.*

14. Установите на панель кнопку Buttonl.Задайте свойству Textзначение «Ввод».

15. Поместите на панель Panel1метку Label2.Свойству Textпридайте значение «Результат». Установите высоту шрифта 12 пт.

16. Установитена панель Panel1 редактор TextBox2.Очистите свойство Text.Установите начертание шрифта – курсив, размер 10 пт.

17. Поместите на форму компонент richTextBox1 (многострочный ре- дактор) и задайте свойству Dockзначение Richt.Компонент займет правую часть формы.

18. Свойству многострочного редактора Linesвведите текст «Граммы Фунты» через 2 табуляции.

19. Установите на панель кнопку Button2.Задайте свойству Textзначение «Выход». Задайте высоту шрифта 12 пт. Измените свойство UseVisualStyleBackColorна false.

20. Выделите кнопку Buttonl.В инспекторе щелкните по закладке events. Дважды щелкните по правой части строки события OnClick*.* В ответ C# активизирует окно программы. Событие OnClickвозникает в работающей программе при щелчке мышью по кнопке.

21. Используя линейки прокрутки, просмотрите содержимое окна программы.

22. Занесите в программу (обработчик события) следующий код.

После public partial class Form1 : Form введите описания переменных:

double x; //значение веса в граммах, вводимое в textBox1

double y; // значение веса в фунтах

int i=1; //количество символов, в строке richTextBox1

В методе

private void button1\_Click(object sender, EventArgs e)

между операторными скобками { } введите следующий код:

x = Convert.ToDouble(textBox1.Text);

y = x / 400;

textBox2.Text= Convert.ToString(y);

MessageBox.Show("Нажата кнопка buttonl"); richTextBox1.Multiline=true;

richTextBox1.Text += (x+ "\t\t"+Convert.ToString(y) + "\n");

MessageBox.Show("Нажата клавиша " + e.ToString()); richTextBox1.TabIndex = i;

i += (Convert.ToString(y) + (char)13).Length;

*Пояснения. В первом операторе присваивания содержимое окна редактора преобразуется в вещественное число. Для отражения результата расчета в окне* textBox2 *используется свойство* Text*. этого компонента. Свойство* Text *класса* richTextBox *добавляет новую строку к имеющемуся в* Text *набору строк. Добавленная строка отображается на экране. Свойство* TabIndex *устанавливает номер позиции в тексте* richTextBox1.Text*. Последний оператор изменяет эту позицию на длину добавленной строки.*

23 Создайте обработчик события для кнопки Button2,введя текст Close();

24 Сохраните программу на диске.

25 Выполните программу. Введите в окно редактора любое число, нажмите на кнопку «Ввод», и Вы получите результат в окнах редакторов.

26 Закомментируйте вывод в окна MessageBox.

27 Добавьте на панель кнопку Button3.Задайте свойству Textзначение «Очистить». Напишите обработчик события для очистки окон textBox1 и textBox2, и установки курсора в окно textBox1 (метод Focus()).

**Задание 2.**

Создайте приложение в соответствии с рисунком 2 для вычисления корней квадратного уравнения.

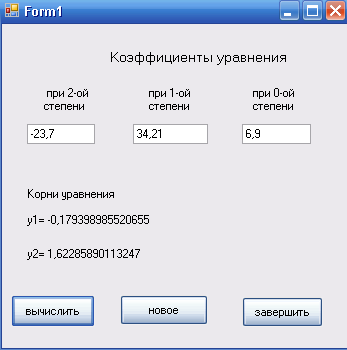


Рисунок 2. Вид окна приложения.

Создайте обработчик событий, который:

– предоставит защиту от ввода недопустимых символов в редакторы Editl, Edit2, Edit3;

– будет вычислять корни квадратного уравнения;

– будет подготавливать окно к новому расчету;

– завершает работу приложения.

**Задание 3.**

Использование элементов управления Common Controls и создание диалоговых окон. Создайте приложение (рисунок 3) для пересчета из одной системы измерения в другую, используя следующие соотношения: 1 фунт = 400 г; 1 пуд = 16380 г; 1 унция = 28,35 г; 1 драхм = 28,35\*16 г; 1 гран = 28,35.437,5 г.

Под расчётом для интервала понимается следующее. При нажатии на радиокнопку «Расчет для интервала» должно появиться три новых текстовых редактора TextBox в одном из которых указывается начальное значение вычисляемой величины, во втором конечное значение вычисляемой величины, а в третьем – шаг приращения от начального к конечному значению. Результаты вычислений для выбранной единицы измерения должны отражаться в белом правом поле – это компонент richTextBox. Описанные действия достаточно сделать только для одной единицы измерения. При активизации поля "Цвет красный" следует изменить цвет единиц измерения с синего на красный.

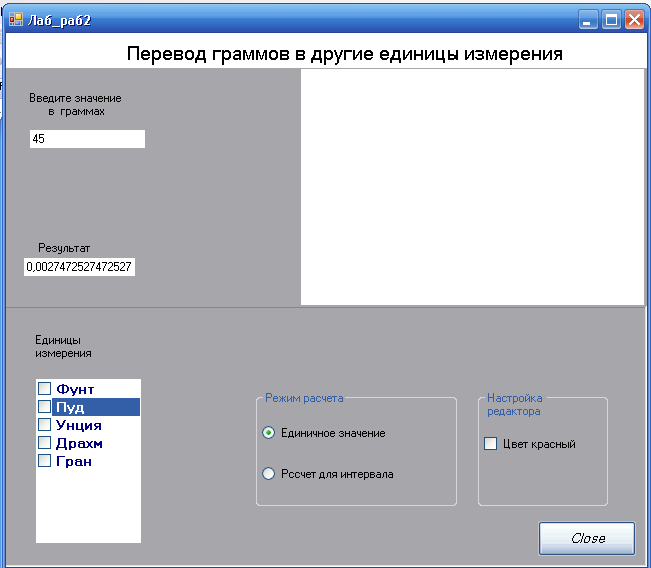


Рисунок 3. Вид окна приложения с элементами управления

**Задание 4.**

Создание Windows-приложений с меню. Разработать приложение, изображенное на рисунке 4.

Менюприложения содержит команды «Файл» и «Редактировать». Команда «Файл» содержит пункты: «Сохранить», «Открыть», «Выход». Команда Редактировать содержит пункты «Вставить», «Удалить».

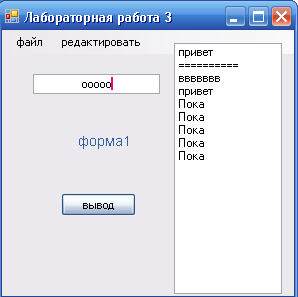


Рисунок 4. Приложение с меню.

Кнопка «Вывод» выводит на форму сообщение, и содержимое редактора TextBox в очередную строку списка ListBox.

Привыборе команды «Файл/Сохранить» открывается диалоговое окно для сохранения в файле содержимого списка ListBox.

При выборе команды «Файл/Открыть» открывается диалоговое окно для чтения файла в список ListBox.

Команда «Редактировать/Вставить» вставляет текст из TextBox в ListBox.

Команда «Редактировать/Удалить» удаляет выделенный диапазон записей из ListBox.

Команда «Файл /Выход» закрывает окно.

## ЛИТЕРАТУРА

1. Орлов, С.А. Теория и практика языков программирования: учеб- ник для вузов. Стандарт 3-го поколения / С.А. Орлов. – СПб.: Пи- тер, 2013. – 688 с.
2. Павловская, Т.А. С#. Программирование на языке высокого уров- ня / Т.А. Павловская. – СПб. [и др.]: Питер: Мир книг, 2013. – 432 с.
3. Троелсен, Э. Язык программирования С# и платформа .NET 4 / Э. Троелсен. – М.: Вильямс, 2007. – 1392 с.
4. Васильев, А. C#. Объектно-ориентированное программирование. Учебный курс. – СПб.: «Питер», 2012. – 316 с.
5. Нортроп, Т. Основы разработки приложений на платформе Microsoft .NET Framework / Т. Нортроп, Ш. Уилдермьюс, Б. Райан. – СПб.: Питер, 2007. – 864 с.
6. Пенкрат, В.В. Методы алгоритмизации: пособие для студентов высших учебных заведений, обучающихся по специальности 1-02 05 03 «Математика. Дополнительная специальность» (1-02 05 03-02 «Математика. Информатика») / В.В. Пенкрат. – Минск: БГПУ, 2012. – 107 с.
7. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# / Дж. Рихтер. – СПб.: Питер, 2013. – 896 с.
8. Севернева, Е.В. Основы алгоритмизации и программирования: учебно-методический комплекс для студентов высших учебных заведений / Е.В. Севернева, Н.М. Жалобкевич. – Минск: БГАТУ, 2009. – 112 с.
9. Шилдт, Г. С# 4.0: полное руководство / Г. Шилдт. – М.: Вильяме, 2012. – 1056 с.
10. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке С# / Дж. Рихтер. – СПб.: Питер, 2007. – 636 c.
11. Потапова, Л.Е. Объектно-ориентированное программирование на языке C#: методические рекомендации к выполнению лабораторных работ / Л.Е. Потапова, Т.Г. Алейникова. – Витебск: ВГУ имени П.М. Машерова, 2016. – 50 с.
12. Справочная онлайн-библиотека для разработчиков на платформе Microsoft.NET. – Режим доступа: [http://msdn.microsoft.com](http://msdn.microsoft.com/)