Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»

Л.Е. Потапова

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ C#

Учебно-методическое пособие

Репозиторий ВГУ

Витебск
УО «ВГУ им. П.М. Машерова»,
2012

УДК 004.438(075.8)
ББК 32.973.26-018.1я73

Автор*:* доцент кафедры информатики и информационных технологий УО «ВГУ им. П.М. Машерова»,
кандидат физ.-мат. наук, доцент **Л.Е. Потапова**

Рецензент:
заведущий кафедры прикладной математики и механики УО «ВГУ им. П.М. Машерова»,
кандидат физ.-мат. наук, доцент С.А. Ермаченко

Потапова Л.Е.
Объектно-ориентированное программирование на языке C# / авт. Л.Е. Потапова. –
Витебск : УО «ВГУ им. П.М. Машерова», 2012. – 122 с.

Учебно-методическое пособие содержит основной материал по программированию на языке C#, необходимую справочную информацию по использованию среды разработки **Visual Studio.NET**, вопросы и индивидуальные задания для лабораторных занятий и самостоятельной работы.

Настоящее издание ориентировано на обучение методам и приемам объектно-ориентированного программирования, которые проиллюстрированы большим количеством примеров с подробными комментариями.

Рассчитано на студентов дневной и заочной формы обучения, изучающих программирование.

УДК 004.43(075)

ББК 32.973-018.1я73

©Потапова Л.Е., 2012
© УО «ВГУ им. П.М. Машерова , 2012

Репозиторий ВГУ

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ 5
1. СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ. ВВОД И ВЫВОД ДАННЫХ 6
1.1. Консольный ввод-вывод 6
1.1.1 Методы вывода 6
1.1.2 Ввод с клавиатуры 8
1.2. Стандартные методы 9
Лабораторная работа № 1 11
2. УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ. ОБЪЯВЛЕНИЕ И ОПРЕДЕЛЕНИЕ МЕТОДОВ 14
2.1. Операторы выбора 14
2.1.1 Операторы if, if ... else 14
2.1.2 Инструкция switch 14
2.2. Итеративные операторы (цикла). 16
2.2.1 Оператор цикла с предусловием 16
2.2.2 Оператор цикла с постусловием 17
2.2.3 Оператор цикла с параметром 17
2.3. Методы 18
Лабораторная работа № 2. 22
3. ПРОСТЕЙШИЕ КЛАССЫ. ИНКАПСУЛЯЦИЯ И СВОЙСТВА. 25
3.1. Классы: основные понятия 25
3.1.1 Данные: поля и константы 26
3.2. Методы 27
3.3. Конструкторы 29
3.4. Свойства 30
3.5. Сбор "мусора" и использование деструкторов 32
Лабораторная работа № 3. 33
4. МАССИВЫ. ИНДЕКСАТОРЫ. 36
4.1. Структура массива в C# 36
4.1.1 Одномерный массив 36
4.1.2 Многомерные массивы 37
4.2. Цикл foreach. 40
4.3. Индексаторы. 41
Лабораторная работа № 4. 45
5. ОПЕРАЦИИ КЛАССА. СТРОКИ. 47
5.1. Перегрузка операций 47
5.1.1 Унарные операции 48
5.1.2 Бинарные операции 50
5.2. Строковые величины 51
5.2.1 Создание строк 52
5.2.2 Работа со строками 52
5.2.3 Класс StringBuilder 54
Лабораторная работа № 5. 55
6. ИЕРАРХИИ КЛАССОВ. НАСЛЕДОВАНИЕ 57
6.1. Наследование 57

Репозиторий ВГУ

6.1.1 Использование защищенного доступа ... 59
6.2. Вызов конструкторов базового класса ... 60
6.3. Наследование и сокрытие имен ... 63
Лабораторная работа № 6. Наследование ... 65
7. ВИРТУАЛЬНЫЕ МЕТОДЫ И ПОЛИМОРФИЗМ. ... 68
7.1. Виртуальные методы ... 68
7.2. Абстрактные классы ... 70
Лабораторная работа № 7. ... 72
8. ИНТЕРФЕЙСЫ И СТРУКТУРНЫЕ ТИПЫ ... 75
8.1. Синтаксис интерфейса ... 75
8.2. Реализация интерфейса ... 76
8.3. Использование интерфейсных ссылок ... 79
8.4. Явная реализация интерфейса ... 82
8.4.1 Закрытая реализация ... 82
8.4.2 Использование явной реализации при множественном наследовании ... 83
8.5. Наследование интерфейсов ... 85
8.6. Реализация интерфейса виртуальными методами ... 87
8.7. Стандартные интерфейсы среды .NET Framework ... 90
8.7.1 Интерфейсы IFormatProviderb и IFormattable ... 90
8.7.2 Интерфейс IComparable ... 91
8.7.3 Интерфейс IComparer ... 93
Лабораторная работа № 8 Интерфейсы. ... 95
ЛИТЕРАТУРА ... 98
ПРИЛОЖЕНИЯ ... 100
Приложение 1. Создание консольного приложения. ... 100
Приложение 2. Встроенные типы в C#. ... 103
Приложение 3. Операции и отношения. ... 105
Приложение 4. Основные методы пространства имен System. ... 108
Приложение 5. Специфекаторы. ... 116
Приложение 6. Интерфейсные коллекции и методы некоторых интерфейсов. ... 118
Предметный указатель ... 121

Репозиторий ВГУ

ВВЕДЕНИЕ

В процессе эволюция языков программирования происходило изменение вычислительной среды, способа мышления и самого подхода к программированию. В настоящее время широко применяются технологии объектно-ориентированного и компонентного программирования. Одним из самых перспективных современных языков программирования является язык C#, реализованный на платформе новой технологии разработки приложений .NET. Его достоинства – структурность, строгий синтаксис, дисциплина типов данных, логичность и удобство конструкций. Мощная библиотека классов платформы .NET позволяет эффективно решать сложные задачи, используя готовые программные компоненты. Немаловажно, что C# является профессиональным языком, предназначенным для решения широкого спектра задач, и в первую очередь – в быстро развивающейся области создания распределенных приложений. Овладение современными технологиями программирования способствует выявлению и развитию у студентов системных подходов к конструированию объектных моделей из разных предметных областей, логического и объектно-ориентированного мышления.

Основная содержательная линия пособия – изучение основ языка C#, принципов объектно-ориентированного программирования, знакомство с динамическими структурами данных и коллекциями. Изучение основ программирования ведется с позиций объектной технологии, где понятие класса рассматривается как тип данных и как модуль – архитектурная единица построения программных систем.

Пособие включает контрольные вопросы и задания для лабораторных занятий и самостоятельной работы, а также большое количество примеров программ с подробными комментариями и объяснениями, направленными на обучение разработке и освоению приемов объектно-ориентированного программирования. В приложениях приводится необходимая справочная информация по операциям, отношениям, методам стандартных классов и интерфейсов, типам языка C#, а также краткое описание создания приложений в среде разработки Visual Studio.NET.

Учебно-методическое пособие написано на основе отработанной методики и лекционного материала, который использовался в процессе обучения студентов математического факультета УО «ВГУ им. П.М. Машерова» в течение ряда лет.

Репозиторий ВГУ

1. СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ. ВВОД И ВЫВОД ДАННЫХ

Данный раздел посвящен знакомству с назначением и возможностями среды разработки **Visual Studio.NET**, формированию представления о создании консольных приложений в данной среде, об использовании методов пространства имен **System**, средствах ввода – вывода данных.

1.1. Консольный ввод-вывод

Совокупность стандартных устройств ввода и вывода, то есть клавиатуры и экрана, называется консолью.

В языке C#, как и во многих других, нет операторов ввода и вывода. Вместо них для обмена с внешними устройствами применяются стандартные объекты. Для работы с консолью в языке C# применяется класс ***Console***, определенный в пространстве имен ***System***.

1.1.1 Методы вывода

Методы класса ***Console*** для вывода данных – ***Write*** и ***WriteLine***.

Метод ***WriteLine*** используется для вывода значений переменных и литералов различных встроенных типов. Это возможно благодаря тому, что в классе ***Console*** существует несколько вариантов методов с именами ***Write*** и ***WriteLine***, предназначенных для вывода значений различных типов. Методы с одинаковыми именами, но разными параметрами называются *перегруженными.* Компилятор определяет, какой из методов вызван, по типу передаваемых в него величин. Методы вывода в классе ***Console*** перегружены для всех встроенных типов данных, кроме того, предусмотрены варианты форматного вывода.

Если метод ***WriteLine*** вызван с одним параметром, он может быть любого встроенного типа, например, числом, символом или строкой. Если требуется вывести несколько величин, их необходимо склеить в одну строку с помощью операции +. Перед объединением строки с числом надо преобразовать число из его внутренней формы представления в последовательность символов, то есть в строку. Преоразование в строку определено во всех стандартных классах C# – для этого служит метод ***ToString***(). Он выполняется неявно, но можно вызвать его и явным образом:

```
Console.WriteLine("i = " + i);
Console.WriteLine("i = " + i.ToString());
```

Для управления форматированием числовых данных необходимо ввести информацию о форматировании:

```
WriteLine("строка_форматирования",arg0,arg1,...,argN);
```


Репозиторий ВГУ

В этой версии метода ***WriteLine***() передаваемые ему аргументы разделяются запятыми, а не знаками "+". Элемент *строка_форматирования* содержит две составляющие: "постоянную" и "переменную". Постоянная составляющая представляет собой печатные символы, отображаемые на экране, а переменная состоит из спецификаторов формата. Общая форма записи спецификатора формата имеет следующий вид:

```
{номер_аргумента, ширина: формат}
```

Здесь элемент *номер_аргумента* определяет порядковый номер отображаемого аргумента, начиная с нулевого. Элемент *ширина* указывает минимальную ширину поля вывода, а *формат* задается элементом формата. Различные спецификации формата в применении к числовым данным представлены в таблице 1.

Таблица 1. Спецификации формата в применении к целому числу.

Тип форматирования	Код формата		
Currency (денежные суммы)	C	C1	C7
Decimal (десятичный)	D	D1	D7
Exponential (экспоненциальный)	E	E1	E7
Fixed point (с фиксированной точкой)	F	F1	F7
General (общий)	G	Gl	G7
Number (числовой)	N	N1	N7
Percent (процент)	P	P1	P7
Hexadecimal (шестнадцатеричный)	X	X1	X7

Если при выполнении метода ***WriteLine***()в строке форматирования встречается спецификатор формата, вместо него подставляется и отображается аргумент, соответствующий заданному элементу *номер_аргумента*. Элементы *ширина* и *формат* указывать необязательно.

В спецификаторе формата часто используются так называемые *пользовательские шаблоны форматирования.* После двоеточия задается вид выводимого значения посимвольно, причем на месте каждого символа может стоять либо #, либо 0. Если указан знак #, на этом месте будет выведена цифра числа, если она не равна нулю. Если указан 0, будет выведена любая цифра, в том числе и 0.

При выводе можно использовать управляющие последовательности.

Пример 1.

```
double a=67.78;
Cosole.WriteLine("a={0,7:G}\nsin={1,10:E}",a,
Math.Sin(a));
Cosole.WriteLine("a={0,7}\nsin={1,10:#.####}",
a,Math.Sin(a);
```

На экране будет сгенерирован следующий результат:

Репозиторий ВГУ

```
67,22
a=  67,22
sin=-9,478925E-001
a=  67,22
sin=    -,94789
```

1.1.2 Ввод с клавиатуры.

В классе ***Console*** определены методы ввода строки и отдельного символа, но нет методов, которые позволяют непосредственно считывать с клавиатуры числа.

Для считывания строки символов используется метод ***ReadLine()***.

Метод ***ReadLine()*** считывает символы до тех пор, пока не будет нажата клавиша <***Enter***>, и возвращает объект типа ***string***. При неудачном завершении метод генерирует исключение типа ***IOException***.

Ввод числовых данных выполняется в два этапа:

- Символы, представляющие собой число, вводятся с клавиатуры в строковую переменную.
- Выполняется преобразование из строки в переменную соответствующего типа.

Преобразование можно выполнить либо с помощью специального класса ***Convert***, определенного в пространстве имен ***System***, либо с помощью метода ***Parse***, имеющегося в каждом стандартном арифметическом классе.

При вводе вещественных чисел дробная часть отделяется от целой с помощью запятой. Допускается задавать числа в экспоненциальной форме.

Для считывания одного символа используется метод ***Read()***. Считанный символ возвращается как значение типа **`int`**, представляющее собой код символа, или -1, если символов во входном потоке нет (например, пользователь нажал клавишу Enter). Затем это значение должно быть явно приведено к типу **`char`**.

По умолчанию консольный ввод данных буферизован с ориентацией на строки. Фактически, данные сначала заносятся в буфер, а затем считываются оттуда процедурами ввода. Занесение в буфер выполняется по нажатию клавиши <***Enter***>, куда заносится и ее код. Метод ***ReadLine*** считывает данные и очищает буфер. Метод ***Read*** считывает из буфера только один символ, и в отличие от ***ReadLine***, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий, т.е. будет читаться код клавиши <***Enter***>. Поэтому необходимо прочитать остаток строки методом ***ReadLine()***.

Репозиторий ВГУ

В случае неудачного исхода операции метод ***Read()*** генерирует исключение типа ***IOException***.

Пример 2. Методы ввода.

```
using System;
namespace ConsoleApplication1
 {class Class1
  {static void Main()
   {
    Console.WriteLine( "Введите строку" );
    string s =Console.ReadLine() ; //ввод строки
    Console.WriteLine("s = " + s );
    Console.WriteLine( "Введите символ" );
    char c =(char)Console.Read(); //ввод символа
    Console.ReadLine();      //считывает остаток строки
    Console.WriteLine( "c = " + c );
    string buf;    //строка-буфер для ввода чисел
    Console.WriteLine("Введите целое число");
    buf = Console.ReadLine(); // ввод с клавиатуры
    int i=Convert.ToInt32( buf ); //преобразование в целое чис-
    ло
    Console.WriteLine( i );
    Console.WriteLine("Введите вещественное число");
    buf = Console.ReadLine(); // ввод с клавиатуры
    double x = double.Parse(buf); //преобразование в веществен-
    ное число
    Console.WriteLine( x );
   }
  }
 }
```

1.2. Стандартные методы

В C# стандартные методы можно использовать без предваряющего их описания. Математические функции реализованы в классе ***Math***, определенном в пространстве имен ***System***. (см. **Таблица 4.1.** Приложение 4). Вызов функций осуществляется использованием ее в качестве операнда в выражениях.

Пример 4. Вычислить выражение:

Репозиторий ВГУ

$$y = k^2 + \frac{\ln(|a|)}{t^3 + 1};$$

Результаты вывести на экран.

Листинг 2. Программа расчета по заданной формуле

```
using System;
namespace ConsoleApplicationl
{class Classl
 {static void Main()
  {
   string buf;    //строка-буфер для ввода чисел
   Console.WnteLine( "Введите k" );
   buf = Console.ReadLine(); // ввод с клавиатуры
    double k = Convert.ToDouble(buf);
                          //преобразование в вещественное число
   buf = Console.ReadLine();    //ввод строки
   double a = double.Parse(buf);
   Console.WriteLine( "Введите a" );
   buf = Console.ReadLine();   //ввод строки
   double t = double.Parse( buf ); //преобразование в вещественное число
   double
   y=k*k+Math.Log(Math.Abs(a))/(Math.Pow(t,3)+1);
   Console.WriteLine("Для k={0}, a={1}и t={2}",k,a,t);
   Console.WriteLine("Результат = " +y );
  }
 }
}
```

Результат работы программы:
Для k=3, a=4,3и t=1,9
Результат = 9,18559804335151

Контрольные вопросы:

1. Основные принципы технологии .NET.
2. Что представляет собой платформа Visual Studio.NET?
3. Как создать консольное приложение?
4. Принципы объектно-ориентированного программирования.
5. Литералы. Как определяются типы литералов?
6. Какие типы относятся к встроенным?
7. Чем отличаются типы-значения и ссылочные типы?
8. Какие типы числовых переменных имеются?

Репозиторий ВГУ

9. Что такое объявление и инициализация?
10. Для чего используется упаковка и распаковка?
11. Как в C# выполняется преобразование типа?
12. Как осуществляется консольный ввод?
13. Чем отличаются методы `Read` и `ReadLine`?
14. Как обеспечить вывод данных на экран?
15. Для чего предназначен и как используется форматный вывод данных?
16. Каковы основные правила использования стандартных функций?
17. Основные приемы работы в среде разработки Visual Studio.NET:
    - Как создать консольное приложение?
    - Как сохранить проект с заданным именем?
    - Как загрузить проект?
    - Как выполнить отладку программы?
    - Как откомпилировать и выполнить программу?
    - Как просмотреть результаты выполнения программы?

Лабораторная работа № 1

Задание 1. Для создания нового проекта выполните следующие действия (см. Приложение 1):

1. Создайте новый проект**: File ► New Project**, либо **Create Project** в окне **Start Page**
2. В окне **New project** в левой части выберите *Visual C# Projects*, в правой – пункт *Console Application*
3. В поле ***Name*** введите имя проекта, в поле ***Location*** – место его сохранения на диске
4. Ознакомьтесь с основными окнами среды.
5. Рассмотрите каждую строку заготовки программы.
6. Наберите приведенный пример программы (Листинг 1).Вставьте свои значения соответствующих типов в пропущенных местах операторов.

Листинг 1

```
using System;
namespace matemat1
{
 class Program
 {
  public static void Main(string[] args)
  //после знака = для x,y,z вставьте Ваши данные
```


Репозиторий ВГУ

```
{int x= ;
 double y= ;
 Console.WriteLine("x: " + x);
 double b = ;
 Console.WriteLine ("y: " + y);
 }
}
}
```

7. Сохраните весь проект на диске: **File ► Save All**
8. Для выполнения программы: **Debug ► Start Without Debugging**

Задание 2. Создайте новое консольное приложение для решения задачи. Введите вещественные числа x, y, z из области допустимых значений исходных данных. Для преобразования к числовой форме используйте класс Convert и метод Parse. Вычислите a, b. Результаты выведите на экран с использованием формата и шаблонов

Варианты заданий:

1. $a = \dfrac{1}{1 + \dfrac{x^2}{2} + \dfrac{y^4}{4}}; b = x\left(tg(z) + e^y\right);$

2. $a = \dfrac{3 + e}{1 + x^2}; b = 1 + x^3 + \dfrac{|y - x|^3}{3};$

3. $a = (1 + y)^2 \dfrac{x^2 + 4}{e^{-x} + x^2 + 4}; b = \dfrac{1}{x^4/2 + \sin^4 z + 1};$

4. $a = y + \dfrac{x^3}{y^2 + \left|\dfrac{x}{y + 3}\right|}; \mathrm{b} = \left(1 + tg^2 z\right);$

5. $a = \dfrac{\cos(x - \pi/6)}{x^2 + 1}; b = 1 + \dfrac{z}{2x^3 + y};$

6. $a = \dfrac{4 + \sin(x + y)}{2 + \left|x - 1 + x^2 y^2\right|}; \mathrm{b} = \cos\left(tg(z)\right);$

7. $a = \sqrt{|x|}\left(x - \dfrac{y^3}{z + x^2}\right); b = x - \dfrac{x^2}{3} + \dfrac{x^5}{5};$

8. $a = \dfrac{1 + \sin(x + y)}{2 + \left|\pi - 1 + \sin(x + y)\right|}; b = x - \dfrac{2}{1 + \sin(x + y)};$

Репозиторий ВГУ

9. $a = (y - \sqrt{|x|})(x - \sin(x + y)); b = \cos(z^2 + x^2/4);$

10. $a = \dfrac{\sin(x)}{|x| + 1}; b = \dfrac{-\sqrt{|\sin x|}}{2 + y^2 + z^2};$

11. $a = \dfrac{2tg(x)}{1 - 2\cos(3y) + tg(z)}; b = (1 + y)\dfrac{\sin x}{2} - \cos(4z);$

12. $a = \dfrac{10 - y^3}{\sqrt{e^x + 1}}; b = \dfrac{\ln|x^2 + 1|}{|z + x + y| + 1};$

13. $a = \dfrac{\sin x - y}{|x| + \cos^2 z + 1}; b = \dfrac{1 - \sqrt{1 + |\sin x|}}{2 + y^2 + z^{-2}};$

14. $a = \dfrac{10^x - y^3}{\sqrt{e^{z^2}}}; b = \dfrac{\ln|x| + 6^y \sqrt{e^z}}{|z| + 10 + \ln \dfrac{x^2 + 1}{y^4 + 3}};$

15. $a = (1 + y)^{1/3} \dfrac{x + y(x^2 + 4)}{e^{-x-2} + 1/(x^2 + 4)};$ $b = \dfrac{1 + \cos(y - 2)}{x^4/2 + \sin^4 z};$

Репозиторий ВГУ

2. УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ. ОБЪЯВЛЕНИЕ И ОПРЕДЕЛЕНИЕ МЕТОДОВ

Цель: формирование умения использования управляющих операторов в методах класса.

Управляющие операторы применяются в рамках методов. Они определяют последовательность выполнения операторов в программе и являются основным средством реализации алгоритмов.

2.1. Операторы выбора

Операторы выбора вводятся ключевыми словами **if**, **if else**, **switch**.

2.1.1 Операторы if, if ... else ...

Структура условного оператора:

```
if (<условное выражение>) <оператор1>; [else <оператор2>];
```

После ключевого слова **if** располагается взятое в круглые скобки условное выражение, за которым следует оператор (блок операторов), который выполняется в случае истинности условного выражения. В противном случае выполняется оператор (блок операторов) после ключевого слова **else**. Часть оператора в квадратных скобках может отсутствовать.

Пример 1:

```
if (i == 10)
{
 if (j<20) a=b;
  if(k >100) c=d;
  else a=c;  //Эта else-инструкция относится к if(k >100)
}
else a = d; // Эта else-инструкция относится к if(i==10)
```

2.1.2 Инструкция switch

Инструкция **switch** (переключатель) обеспечивает многонаправленное ветвление. Она позволяет делать выбор одной из множества альтернатив.

Общий формат записи инструкции **switch**:

```
switch(выражение) {
```


Репозиторий ВГУ

```
  case <константа1>: <оператор1>; break;
  case <константа2>: <оператор2>;break;
  case <константа3>: <оператор3>;break;
  [default: <оператор>;break;]
}
```

Элемент *выражение* инструкции **switch** должен иметь целочисленный тип (например, **char**, **byte**, **short**, **int**) или тип **string**. Очень часто в качестве управляющего **switch**-выражения используется просто переменная, **case**-константы должны быть литералами, тип которых совместим с типом заданного выражения. При этом никакие две **case**-константы в одной **switch**-инструкции не могут иметь идентичных значений.

Инструкция **switch** работает следующим образом. Значение выражения последовательно сравнивается с константами из заданного списка. При обнаружении совпадения для одного из условий сравнения выполняется последовательность инструкций, связанная с этим условием, до тех пор, пока не встретится инструкция **break**. Последовательность инструкций **default**-ветви выполняется в том случае, если ни одна из заданных case-констант не совпадет с результатом вычисления **switch**-выражения. Ветвь **default** необязательна. Если она отсутствует, то при несовпадении результата выражения ни с одной из case-констант никакое действие выполнено не будет.

Использование **switch**-инструкции демонстрируется в следующей программе.

Пример 2. С клавиатуры вводится номер месяца, выдать сообщение о соответствующем времени года.

```
using System;
namespace ConsoleApplicationl
{class Classl
 {static void Main()
  {
   string buf:
   Console.WnteLine( "Введите номер месяца" );
   buf = Console. ReadLine();
   int m = Convert.ToInt( buf );
   switch (m)
   {1, 2, 12 : Console.WriteLine("зима");break;
    3, 4, 5 : Console.WriteLine("весна");break;
    6, 7, 8 : Console.WriteLine("лето");break;
    9, 10, 11 : Console.WriteLine("осень");break;
    default Console.WriteLine ("неверный номер
    месяца");break;
```

Репозиторий ВГУ

```
      }
    }
  }
}
```

2.2. Итеративные операторы (цикла).

Операторы цикла вводятся ключевыми словами **while, do...while, for, foreach**.

2.2.1 Оператор цикла с предусловием

Общая форма цикла **while** имеет вид:

```
while (<условие>) <оператор>;
```

Работой цикла управляет элемент *условие*, который представляет собой любое допустимое выражение типа **bool**. Под элементом *оператор* понимается либо одиночная инструкция, либо блок инструкций. Инструкция выполняется до тех пор, пока условное выражение возвращает значение ИСТИНА. Как только это условие становится ложным, управление передается инструкции, которая следует за этим оператором.

Пример 3. Вычислить порядок целого числа.

```
using System;
class WhileDemo
{
 public static void Main()
 {
  int num;
  int mag;
  num = 435679;   //данное число
  mag = 0;        //переменная для порядка
  Console.WriteLine("Число: " + num);
  while(num > 0)
  {
   mag++;
   num = num /10;
  }
  Console.WriteLine("Порядок: " + mag);
 }
}
```

Результат работы программы:

Число: 435679

Порядок: 6

Репозиторий ВГУ

2.2.2 Оператор цикла с постусловием

Общий формат цикла с постусловием имеет вид:

```
do {<оператор>;} while (<условие>);
```

Цикл **do-while** выполняется до тех пор, пока остается истинным элемент *условие*, который представляет собой условное выражение. В отличие от цикла **while**, в котором условие проверяется при входе, цикл **do-while** проверяет условие при выходе из цикла. Это значит, что цикл **do-while** всегда выполняется хотя бы один раз.

Фигурные скобки необязательны, если элемент *оператор* состоит только из одной инструкции, но их желательно использовать для улучшения читабельности конструкции **do-while**.

Пример 4. Найти сумму ряда с точностью $\varepsilon = 10^{-4}$, общий член которого:

$$a_n = \frac{1}{n}$$

```
using System;
 class DoWhile
 {
  public static void Main()
  {
   double e=0.0001;
   double s=0;  //s – сумма
   int i=0;
   double a=0; // a – слагаемое
   {подсчёт суммы в цикле do-while}
   do {s += a; i++; a=1/i;}
   while (a>=e);
   Console.WriteLine("Ответ do-while
   {0,10:0.00000}"+s);
   }
  }
 }
```

Результат работы программы:

Ответ do-while   8,78761

2.2.3 Оператор цикла с параметром

Общий формат записи цикла **for** имеет следующий вид:

```
for (<инициализация>;<условие>;<итерация>) <оператор>;
```

Элемент *инициализация* представляет собой инструкцию присваивания *управляющей переменной цикла* начального значения. Эта переменная действует в качестве счетчика, который управляет работой цикла.

Репозиторий ВГУ

Элемент *условие* представляет собой выражение типа **bool**, в котором тестируется значение управляющей переменной цикла. Цикл **for** будет выполняться до тех пор, пока условие истинно. Как только условие станет ложным, выполнение программы продолжится с инструкции, следующей за циклом **for**. Элемент *итерация* – это выражение, которое определяет, как изменяется значение управляющей переменной цикла после каждой итерации. Все элементы цикла **for** должны отделяться точкой с запятой. Тело цикла может быть пустым.

Пример 5. Найти сумму чисел от 1 до 10.

```
using System;
 class Class3
 {
  public static void Main() {
  int s = 0;
  // Суммируем числа от 1 до 10.
  for(int i = 1; i <= 10; s += i++;)
  Console.WriteLine("Сумма равна " + sum);
 }
}
```

Результат работы программы:

Сумма равна 0
Сумма равна 1
Сумма равна 3
Сумма равна 6
Сумма равна 10
Сумма равна 15
Сумма равна 21
Сумма равна 28
Сумма равна 36

2.3. Методы

Метод – это функциональный элемент класса, который реализует вычисления или другие действия, выполняемые классом или экземпляром. Методы определяют поведение класса.

Метод представляет собой законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

Синтаксис метода:

```
[<спецификаторы>] <тип> <имя_метода>([<параметры>])
  {тело_метода}
```


Репозиторий ВГУ

Первая строка представляет гобой *заголовок* метода. *Тело метода,* задающее действия, выполняемые методом, чаще всего представляет собой блок – последовательность операторов в фигурных скобках.

При описании методов можно использовать спецификаторы 1-7 из табл. 5.1 приложения 5. Чаще всего для методов задается спецификатор доступа **public**, так как методы составляют интерфейс класса, поэтому они должны быть доступны.

Тип определяет, значение какого типа вычисляется с помощью метода. После выполнения метода происходит возврат в то место вызывающей функции, откуда был вызван метод, и передача туда значения выражения, записанного в операторе **return**. Если метод не возвращает никакого значения, в его заголовке задается тип **void**, а оператор return отсутствует.

Параметры, описываемые в заголовке метода, определяют множество значений *аргументов,* которые можно передавать в метод. Параметр представляет собой локальную переменную, которая при вызове метода принимает значение соответствующего аргумента. Область действия параметра – весь метод. Для каждого параметра должны задаваться его *тип* и *имя.*

Главное требование при передаче параметров состоит в том, что аргументы при вызове метода должны записываться в том же порядке, что и в заголовке метода, и должно существовать неявное преобразование типа каждого аргумента к типу соответствующего параметра. При несоответствии типов выдается диагностическое сообщение.

Метод, не возвращающий значение, вызывается отдельным оператором, а метод, возвращающий значение, – в составе выражения в правой части оператора присваивания.

Статические (**static**) методы, или методы класса, можно вызывать, не создавая экземпляр объекта. Именно таким образом используется метод ***Main***.

Например, заголовок метода Sin выглядит следующим образом:

```
public static double Sin( double a );
```

Имя метода с количеством, типами и спецификаторами его параметров представляет собой сигнатуру метода – то, чем один метод отличают от других. В классе не должно быть методов с одинаковыми сигнатурами.

Пример 6. Введите символ. Преобразуйте его к знаковому типу. Если результат отрицательный, выведите его значение. Если положительный и является прописной буквой латинского алфавита, выведите букву, если знак препинания – сообщение, иначе комментарий.

```
using System;
namespace симв_выбор
{
```

Репозиторий ВГУ

```
class SwitchDemo1
{
 public static void Main()
 {
   Console.WriteLine( "Введите символ" );
   char c = (char)Console.Read(); //ввод символа и
                                  //приведение к типу char
   Console.ReadLine();       //считывает остаток строки
   Console.WriteLine("символ "+c); //вывод символа
   sbyte z =(sbyte)c;         //приведение к типу sbyte
   int c1=Math.Sign(z);   // Знак числа
   Console.WriteLine("значение числа"+ z +"знак"+c1);
   switch (c1)
    {
      case -1:
      Console.WriteLine("значение отрицательно "+z);
      break;
      case 1 : if (c>='A'& c<= 'Z')
      Console.WriteLine("прописная латинская буква
      "+c);
      else if (Char.IsPunctuation(c))
      Console.WriteLine("знак препинания "+c);
      else Console.WriteLine("другой " +c); break;
     }
  }
 }
}
```

Пример 7. Для каждого числа в заданном диапазоне найти наименьший множитель: 2,3,5 или 7

```
using System;
namespace primer_7
{
 class Class1
 {static int delit(int n) //метод нахождения множителя
   {
   if((n % 2) == 0) return 2;
   else if((n % 3) == 0) return 3;
   else if((n % 5) == 0) return 5;
   else if((n % 7) == 0) return 7;
   else return -1;
```

Репозиторий ВГУ

```
    }
  public static void Main()
    {
    string buf;
    buf=Console.ReadLine();
    int m=int.Parse(buf);          // нижняя граница диапазона
    buf=Console.ReadLine();
    int k= Convert.ToInt32(buf); //верхняя граница диапазона
    Console.WriteLine("Нижняя граница диапазона" +m);
    Console.WriteLine("Верхняя граница диапазона"+k);
    for(int num = m; num < k; num++)
      {int d=delit(num);
      if (d!=-1)
      Console.WriteLine("Наименьший множитель числа " +num + "
      равен "+ d);
      else Console.WriteLine(num +  "не делится на 2,3,5 или
      7");

      }
    }
  }
}
```

Результат работы программы:

Нижняя граница диапазона 23
Верхняя граница диапазона 32
23 не делится на 2, 3, 5 или 7.
Наименьший множитель числа 24 равен2
Наименьший множитель числа 25 равен5
Наименьший множитель числа 26 равен2
Наименьший множитель числа 27 равен3
Наименьший множитель числа 28 равен2
29 не делится на 2, 3, 5 или 7.
Наименьший множитель числа 30 равен2
31 не делится на 2, 3, 5 или 7

Контрольные вопросы:

1. Выражением какого типа является условие в операторе **if**? Какие значения оно может принимать?
2. Как работает оператор **if,** если отсутствует часть **else** <оператор 2>?
3. В каких случаях используется оператор **switch***?*

Репозиторий ВГУ

4. Какого типа может быть <выражение> в операторе **`switch`**?
5. В каком случае выполняется последовательность инструкций **`default`**-ветви?
6. В чем отличие операторов **`while и do ... while`**
7. Что представляет собой элемент <инициализация> в операторе **`for`** ?
8. Какого типа может быть элемент <*условие*> в цикле **`for?`**.
9. Назначение *управляющей переменной цикла* **`for`**?
10. Назначение управляющих операторов **`goto, break, continue, return`**.
11. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?
12. Что представляют собой методы?
13. Как объявляется метод?
14. Какова область действия параметров метода?
15. Как вызываются методы?
16. Общие (статические) методы класса.

Лабораторная работа № 2.

Объявление и определение методов

Задание 1. Создайте проект для решения задачи: На экран выводить исходные данные и результаты. В работе использовать только стандартные типы: числовые, символьный и булевский.

Варианты заданий:

1. Напишите программу, которая по введенным целочисленным значениям определяет являются ли они строчными буквами русского алфавита или прописными. Вывести число, букву и комментарий.
2. Введите целое число. Если оно больше 255, вычислите корень квадратный из него, если меньше и соответствует отображаемым символам кодовой таблицы, выведите символ и его код, и комментарий.
3. Напишите программу для определения, какому алфавиту (латинскому или русскому) принадлежит введенный с клавиатуры символ. На экран вывести символ и комментарий. Предусмотреть возможность ввода неверного символа.
4. Введите символьное название дня недели (например, ’п’ – понедельник). Выведите сообщение, сколько дней осталось до выходного дня (воскресения).
5. Напишите программу для ввода букв латинского алфавита. Если введенный символ не является буквой латинского алфавита, замените его на знак ‘?’. Если буква является прописной, замените ее на строчную.

Репозиторий ВГУ

6. Напишите программу, которая выполняет конверсию из сантиметров в дюймы и футы и наоборот. При вводе величин укажите единицу измерения – ‘i’ для дюймов, ’c’ для см., f – для футов.
7. В компьютер поступают результаты по плаванию трех спортсменов. Составьте программу, печатающую по выбору пользователя:
   а) лучший результат
   б) второй результат
   в) результаты в порядке возрастания
8. Введите целое число. Если оно является кодом отображаемого символа, выведите символ и код, если нет выведите число и комментарий.
9. По введенному символу и значению аргумента выведите значение одной из функций sin(x), cos(x), tg(x), ctg(x) и соответствующий комментарий.
10. Составьте программу, определяющую для буквы английского алфавита, является ли буква гласной или нет. Если буква гласная, определите следующую за ней букву. Выведите введенный символ, результат и комментарий.
11. Введите 3 символа. Если символы – цифры, найдите и выведите их сумму и цифры, иначе выведите код символа.
12. Введите символ. Если символ – цифра, выведите ее значение, увеличенное на единицу, если буква латинского алфавита, выведите следующий за ней символ, иначе код введенного символа. Выведите соответствующий комментарий.

Задание 2. Напишите функции в виде методов. Напишите тестирующую программу с выдачей результатов на экран.

Варианты заданий:

1. Напишите программу, определяющую являются ли вводимые N чисел простыми.
2. Напишите программу последовательного ввода чисел из диапазона 0…255. Запрещается последовательно вводить два числа, разность между которыми меньше 7. Программа заканчивает работу после обнаружения первой ошибки.
3. В последовательности чисел вводимых с клавиатуры исключить все цифры 1 и 3, оставив прежним порядок оставшихся цифр.
4. С клавиатуры вводится последовательность трехзначных чисел. Преобразовать каждое число так, чтобы получалось наибольшее число, записанное теми же цифрами.
5. Вывести таблицу значений функций $\sqrt{x^2+1}$ и $\sqrt{x^2-1}$ с шагом h.
6. Найти первый элемент, больший K, последовательностей {x} и {y}, определяемых рекуррентными соотношениями:

Репозиторий ВГУ

$x_i = x_{i-1} + 2y_{i-1}$
$y_i = y_{i-1} - 2x_{i-1}$
$x_1 = 1,\ y_1 = 0.$

7. Найдите все простые делители в заданном диапазоне целого числа.
8. Напишите программу перевода десятичного числа в систему счисления с другим основанием.
9. Напишите программу, которая вычисляет сумму цифр натурального числа, если число - четное, и остаток от деления на 10, если нечетное.
10. Вычислить при данном n и действительном x

$$\sin x + \sin\sin x + \ldots + \underbrace{\sin\sin\ldots\sin x}_{n}$$

11. Дано натуральное число n. Определить, сколько в числе нулей.
12. Найти степень числа N, у которой две последние цифры одинаковы.

Репозиторий ВГУ

3. ПРОСТЕЙШИЕ КЛАССЫ. ИНКАПСУЛЯЦИЯ И СВОЙСТВА.

Данный раздел посвящен усвоению понятия класса, определения членов класса, умению выполнять обработку исключительных ситуаций, овладению приемами разработки простейших программ в объектно-ориентированном стиле программирования.

3.1. Классы: основные понятия

Класс является типом данных, определяемым пользователем. Он должен представлять собой одну логическую сущность, например, являться моделью реального объекта или процесса. Элементами класса являются данные и функции, 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 C#. *Тело класса* – это список описаний его элементов, заключенный в фигурные скобки. Список может быть пустым, если класс не содержит ни одного элемента.

Спецификаторы определяют свойства класса, а также доступность класса для других элементов программы (приложение 5). Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. В последнем случае класс называется вложенным. В зависимости от места описания класса некоторые из этих спецификаторов могут быть запрещены.

Для классов, которые описываются в пространстве имен непосредственно, допускаются только два спецификатора: **public** и **internal**. По умолчанию подразумевается спецификатор **internal**.

Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов, и используется для их создания. Объекты также называются экземплярами класса. Объекты создаются явным или неявным образом, то есть либо программистом, либо системой. Программист создает экземпляр класса с помощью операции **new**:

```
<переменная_типа_класса> = new <имя_класса>();
```

Например:

Репозиторий ВГУ

```
Demo a = new Demo();  // создание экземпляра класса Demo
```

Класс относится к ссылочным типам данных, память под которые выделяется в хипе. Если достаточный для хранения объекта объем памяти выделить не удалось, операция **new** генерирует исключение ***OutOfMemoryException***. Рекомендуется предусматривать обработку этого исключения в программах, работающих с объектами большого объема.

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.

3.1.1 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>]
```

Спецификаторы полей и констант класса приведены в таблице 5.2 приложения 5 Все поля сначала автоматически инициализируются нулем соответствующего типа (например, полям типа **`int`** присваивается 0, а ссылкам на объекты – значение **`null`**). 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.

По умолчанию элементы класса считаются закрытыми. Поля, описанные со спецификатором **`static`**, а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются через имя класса. Если класс содержит только статические элементы, экземпляр класса создавать не требуется.

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 ConsoleApplication1
```


Репозиторий ВГУ

```
{
   class Demo
  {
    public int a = 1;                   //поле данных
    public const double c = 1.66; //константа
    public static string s = "Demo";//статическое
                                    //поле класса
    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.c); // 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, среда присваивает ему значение ноль.

Поля со спецификатором **`readonly`** предназначены только для чтения. Установить значение такого поля можно либо при его описании, либо в конструкторе.

3.2. Методы

Методы класса имеют непосредственный доступ к его закрытым полям. Метод, описанный со спецификатором **`static`**, должен обращаться только к статическим полям класса. Статический метод вызывается через имя класса, а обычный – через имя экземпляра. При вызове метода из другого метода того же класса имя класса/экземпляра можно не указывать.

Существуют два способа передачи параметров: по значению и по ссылке.

При передаче по значению метод получает копии значений аргументов, и операторы метода работают с этими копиями. Доступа к исходным значениям аргументов у метода нет, а следовательно, нет и возможности их изменить. *При передаче по ссылке (по адресу)* метод получает копии адресов аргу-

Репозиторий ВГУ

ментов, он осуществляет доступ к ячейкам памяти по этим адресам и может изменять исходные значения аргументов, модифицируя параметры.

В C# для обмена данными между вызывающей и вызываемой функциями предусмотрено четыре типа параметров:

- параметры-значения. Описывается в заголовке метода следующим образом: `<тип> <имя>`
- параметры-ссылки – описываются с помощью ключевого слова **`ref`**: **`ref`** `<тип>  <имя>`.Параметры-ссылки используются, если в методе требуется изменить значение каких-либо передаваемых в него величин. При вызове метода на месте параметра-ссылки может находиться только ссылка на инициализированную переменную точно того же типа.
- выходные параметры - описываются с помощью ключевого слова **`out`**. Параметру, имеющему этот спецификатор, должно быть обязательно присвоено значение внутри метода. В вызывающем коде можно использовать переменную без инициализации.
- параметры-массивы – описываются с помощью ключевого слова **`params`**. Параметр-массив может быть только один и должен располагаться последним в списке.

Ключевое слово предшествует описанию типа параметра. Если оно опущено, параметр считается параметром-значением.

Пример 2. Параметры-значения, параметры-ссылки, выходные параметры

```
using System;
namespace ConsoleApplicationl
{class Classl
  {
   static void P(int a, ref int b, out int c)
   {
     a = 44; b = 33; c=55;
     Console.WriteLine("внутри метода {0} {1} {2}",
     a,b,c);
    }
   static void Main()
    {
      int a = 2, b = 4, c;
      Console.WriteLine("до вызова {0} {1}",a, b);
      P(a, ref b, out c);
      Console.WriteLine("после вызова {0} {1} {2}", a,
      b, c);
    }
   }
 }
```

Репозиторий ВГУ

Результаты работы этой программы:

```
до вызова     2 4
внутри метода 44 33
55
после вызова 2 33 55
```

3.3. Конструкторы

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

```
[спецификатор] <имя_класса>()
{ тело конструктора}
```

Обычно в качестве элемента *спецификатор* используется 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 не что иное, как конструктор реализуемого класса. Если в классе конструктор не определен явным образом, оператор **new** будет использовать конструктор по умолчанию, который предоставляется средствами языка C#. Таким образом, оператор **new** можно использовать для создания объекта любого "классового" типа.

Пример 3. В программе создаются два объекта с различными значениями полей с помошью конструктора по умолчанию и с параметрами.

```
using System;
namespace ConsoleApplicationl
{
  class Demo
   {internal int a;
    internal double y;
     public Demo(int a,double y1)  //конструктор с параметрами
     {
```

Репозиторий ВГУ

```
        this.a = a;  //полю а присваиваем значение параметра а
        y = y1;
      }
      public Demo() // конструктор по умолчанию
      {
        a = 276;
        y = 5.5;
      }
    }
  class Class1
   {static void Main()
      {  //вызов конструктора с параметром
      Demo ob1 = new Demo(300, 0.002);
      Console.WriteLine(ob1.a);  //результат: 300
      Console.WriteLine(ob1.y);  //результат:0,002
      Demo ob2 = new Demo();    //вызов конструктора без параметра
      Console.Writeline(ob2.a);  //результат: 276
      Console.WriteLine(ob2.y);  //результат: 5.5
      }
   }
}
```

3.4. Свойства

Свойство – это второй специальный тип членов класса. Свойство служит 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 доступа* представляет собой блоки опера-

Репозиторий ВГУ

торов, которые выполняются при получении (**get**) или установке (**set**) свойства. Может отсутствовать либо часть **get**, либо **set**, но не обе одновременно. Если отсутствует часть **set**, свойство доступно только для чтения (read-only), если отсутствует часть **get**, свойство доступно только для записи (write-only).

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

Свойства не определяют область памяти. Следовательно, свойство управляет доступом к полю, но самого поля не обеспечивает. Это поле должно быть задано независимо от свойства.

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

Можно задавать разные уровни доступа для частей **get** и **set**. Например, во многих классах возникает потребность обеспечить неограниченный доступ для чтения и ограниченный – для записи. Спецификаторы доступа для отдельной части должны задавать либо такой же, либо более ограниченный доступ, чем спецификатор доступа для свойства в целом. Например, если свойство описано как **public**, его части могут иметь любой спецификатор доступа, а если свойство имеет доступ **protected internal**, его части могут объявляться как **internal**, **protected** или **private**. Синтаксис свойства имеет вид

```
[атрибуты][спецификаторы] <тип> <имя_свойства>
{
   [[атрибуты] [спецификаторы] get <код_доступа>]
   [[атрибуты] [спецификаторы] set <код_доступа>]}
```

Пример 4. Использование свойства. Это свойство позволяет присваивать полю только положительные числа.

```
using System;
class SimpProp {
  int prop; // Это закрытое поле управляется свойством myp.
  public SimpProp() { prop = 0 ; }
  public int myp    /* Это свойство поддерживает доступ к закрытой переменной класса prop. Оно позволяет присваивать ей только положительные числа.
  */
  {get {return prop; // способ получения свойства

     }
  set {
        if(value >= 0) prop = value; // способ установки свойства
      }
```


Репозиторий ВГУ

```
  }
}
// Демонстрируем использование свойства
class PropertyDemo {
  public static void Main()
  {
     SimpProp ob = new SimpProp ();
     Console.WriteLine("Исходное значение ob.myp:"+ob.myp);
     ob.myp = 100;  // вызывается метод установки свойства
     Console.WriteLine("Значение ob.myp: " + ob.myp);
     //Переменной prop невозможно присвоить отрицательное значение.
     Console.WriteLine("Попытка присвоить -10 свойству
     ob.myp");
     ob.myp = -10;
     Console.WriteLine("Значение ob.myp: " + ob.myp);
  }
}
```

Результаты выполнения программы:
Исходное значение ob.myp: 0
Значение ob.myp: 100
Попытка присвоить -10 свойству ob.myp
Значение ob.myp: 100

На использование свойств налагаются довольно серьезные ограничения. Во-первых, поскольку в свойстве не определяется область памяти, его нельзя передавать методу в качестве **ref**- или **out**-параметра. Во-вторых, свойство нельзя перегружать.

Свойство не должно изменять состояние базовой переменной при вызове **get**-аксессора, хотя несоблюдение этого правила компилятор обнаружить не в состоянии.

3.5. Сбор "мусора" и использование деструкторов

При использовании оператора **new** объектам динамически выделяется память из пула свободной памяти. Объем буфера динамически выделяемой памяти не бесконечен, и рано или поздно свободная память может исчерпаться. Поэтому одним из ключевых компонентов схемы динамического выделения памяти является восстановление свободной памяти от неиспользуемых объектов, что позволяет делать ее доступной для создания последующих объектов.

В C# эта проблема решается с использованием системы сбора мусора.

Репозиторий ВГУ

Система сбора мусора C# автоматически возвращает память для повторного использования, действуя незаметно и без вмешательства программиста. Ее работа заключается в следующем. Если не существует ни одной ссылки на объект, то предполагается, что этот объект больше не нужен, и занимаемая им память освобождается. Эту память снова можно использовать для размещения других объектов.

Средства языка C# позволяют определить метод, который должен вызываться непосредственно перед тем, как объект будет окончательно разрушен системой сбора мусора. Этот метод называется деструктором, и его можно использовать для обеспечения гарантии "чистоты" ликвидации объекта.

Формат записи деструктора такой:

```
~имя_класса()
{
    // код деструктора
}
```

Контрольные вопросы:

1. Как описываются классы в C#?
2. Что относится к членам класса?
3. Что такое статические члены класса?
4. Данные: поля и константы.
5. Спецификаторы полей и констант класса.
6. Как передаются параметры в методы?
7. Для чего предназначен параметр **params**?
8. Что представляет собой конструктор? Для чего он используется?
9. Какие бывают конструкторы?
10. Может ли класс не иметь конструктора?
11. Для чего предназначена система сбора мусора?

Лабораторная работа № 3.

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

Разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), методы и свойства. Методы и свойства должны обеспечивать непротиворечивый и удобный интерфейс класса.

В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта.

Репозиторий ВГУ

Варианты заданий:

1. Описать класс «здание», содержащий сведения о количестве подъездов и этажей, и количестве квартир на этаже, стоимости квадратного метра. Предусмотреть инициализацию с проверкой допустимости значений полей. Описать методы вычисления количества квартир в подъезде и в доме, общую стоимость квартир в доме.
2. Описать класс, представляющий линейное уравнение вида $ax + b = 0$. Описать метод, вычисляющий решение этого уравнения.
3. Составить описание класса для вектора, заданного его координатами в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора, вычисления скалярного произведения двух векторов, длины вектора.
4. Составить описание класса многочлена вида $\boldsymbol{ax^3 + bx^2 + cx + d}$. Предусмотреть методы, реализующие:
   - вычисление значения многочлена для заданного аргумента;
   - операцию сложения, вычитания многочленов с получением нового объекта-многочлена;
   - вывод на экран описания многочлена.
5. Описать класс, реализующий член геометрической прогрессии. Предусмотреть инициализацию значениями по умолчанию и произвольными значениями. Предусмотреть методы: нахождения следующего члена прогрессии, суммы следующих **k** членов.
6. Описать класс, представляющий полукруг с основанием, параллельным оси **x**. Предусмотреть методы для вычисления площади полукруга, длину обрамляющей линии, и проверки попадания данной точки внутрь полукруга.
7. Составить описание класса прямоугольников со сторонами, параллельными осям координат, и заданного координатами двух его вершин. Предусмотреть вычисление сторон прямоугольника и изменение его размеров.
8. Описать класс дерево, содержащий сведения о названии, высоте и возрасте дерева. Предусмотреть инициализацию полей с проверкой допустимости значений. Разработать метод нахождения среднего прироста дерева в год.
9. Описать класс, представляющий четырехугольник. Разработать методы для определения является ли четырехугольник параллелограммом.
10. Описать класс «аудитория», содержащий сведения о длине и ширине, высоте потолков и количестве компьютеров в аудитории. Предусмотреть инициализацию с проверкой допустимости значений полей. Описать методы вычисления площади и объема аудитории и выдачи сообщения выполняются ли санитарные нормы (площадь на 1 компьютер должна быть не менее 6 м^2).

Репозиторий ВГУ

11. Описать класс для работы с двоичным числом (4 разряда) Предусмотреть инициализацию с проверкой допустимости значений. Описать методы перевода в десятичную систему счисления и сдвига числа влево. Строки не использовать.
12. Составить описание класса дата рождения. Предусмотреть инициализацию с проверкой допустимости значений полей (год, месяц, день). Создать метод вычисления возраста (количества лет, месяцев, дней).

Задание 2. Включите в проект ***Задания 1***обработку исключений.

Репозиторий ВГУ

4. МАССИВЫ. ИНДЕКСАТОРЫ.

Данный раздел посвящен усвоению основных приемов создания классов с полями структурированного типа - массив; формированию знаний о возможности использования индексаторов для доступа к элементам закрытых массивов.

4.1. Структура массива в C#

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

В C# массивы могут быть одномерными или многомерными. C#-массивы реализованы как объекты. Все массивы имеют общий базовый класс ***Array***, определенный в пространстве имен ***System*.**

4.1.1 Одномерный массив

Одномерный массив – это список связанных переменных. Для объявления одномерного массива используется следующая форма записи:

```
<тип>[] <имя_массива>( = new <тип> [<размер>]);
```

Здесь с помощью элемента записи *тип* объявляется базовый тип массива. В круглых скобках выражение может отсутствовать.

Количество элементов, которые будут храниться в массиве, определяется элементом записи *размер* (может быть выражение). Поскольку массивы реализуются как объекты, их создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, а затем для него выделяется память, и переменной массива присваивается ссылка на эту область памяти. Таким образом, в C# массивы динамически размещаются в памяти с помощью оператора **`new`**.

Массивы можно инициализировать в момент их создания. Формат инициализации одномерного массива имеет следующий вид:

```
<тип>[] <имя_массива> = {val1, val2,...,valN};
```

Здесь начальные значения, присваиваемые элементам массива, задаются с помощью последовательности `val1–valN`. Область памяти для массива выделяется автоматически в соответствии с заданными значениями инициализации (инициализаторами). В этом случае нет необходимости использовать в явном виде оператор **`new`**, однако его можно использовать.

Репозиторий ВГУ

Пример 1. Возможные варианты инициализации массива.

```
int[] n = {99,10,100,18,78,23,63,9,8,9};
int[] n = new int[]{99,10,100,18,78,23,63,9,8,9};
int[] n = new int[10]{99,10,100,18,78,23,63,9,8,9};
```

В последнем варианте размер должен соответствовать количеству инициализаторов.

Несмотря на избыточность, *new*-форма инициализации массива оказывается полезной в том случае, когда уже существующей ссылочной переменной массива присваивается новый массив. Например:

```
int [] n;
n = new int[]{99,10,100,18,78,23,63,9,8,9};
```

Доступ к элементу массива осуществляется указанием индекса в квадратных скобках после имени массива, например: `n[6]`. Нумерация индексов начинается с 0.

4.1.2 Многомерные массивы

Многомерным называется такой массив, который характеризуется двумя или более измерениями, а доступ к отдельному элементу осуществляется посредством двух или более индексов.

Простейший многомерный массив – двумерный.

Для объявления двумерного массива используется следующая форма записи.

```
<тип>[,] <имя_массива>( = new <тип> [разм1,разм2]);
```

Выражение в круглых скобках может отсутствовать, например:

```
int [,] table = new int[10, 20];
```

Правая часть этого объявления означает, что создается ссылочная переменная двумерного массива. Для реального выделения памяти для этого массива с помощью оператора **new** используется более конкретный синтаксис: `int[10, 20]`

Чтобы получить доступ к элементу двумерного массива, необходимо указать оба индекса, разделив их запятой. Например:

```
table [3 , 5] = 10;
```

Пример 1. Программа заполняет двумерный массив числами от 1 до 12, а затем выводит содержимое этого массива.

```
using System;
 class TwoD
 {
  public static void Main()
```

Репозиторий ВГУ

```
  {
    int t, i;
    int[,] table = new int[3, 4];
    for(t=0; t < 3; ++t) {
     for(i=0; i < 4; ++i) {
      table[t,i] = (t*4)+i+1;
      Console.Write(table[t,i] + " ") ;
     }
    Console.WriteLine();
    }
  }
 }
```

В этом примере элемент массива table [0,0] получит число 1, элемент table [0,1] – число 2, и т.д.

В общем случае многомерный массив объявляется так:

```
<тип> [,..,] <имя> = new <тип>[размер1,...,размерN];
```

Многомерный массив можно инициализировать, заключив список инициализаторов каждой размерности в собственный набор фигурных скобок.

Формат инициализации двумерного массива:

```
<тип>[,] <имя_массива> = {
{val, val, val, ..., val},
{val, val val, ..., val},
{val, val, val, ..., val}
};
```

Здесь элемент *val* – значение инициализации. Каждый внутренний блок означает строку. В каждой строке первое значение будет сохранено в первой позиции массива, второе значение – во второй и т.д.

Пример 2. Программа инициализирует массив s числами от 1 до 5 и квадратами этих чисел.

```
// Инициализация двумерного массива.
using System;
 class Squares {
  public static void Main() {
   int[,] s= //объявляем двумерный массив с инициализацией
   {
     { 1, 1 },
     { 2, 4 },
     { 3, 9 },
     { 4, 16 },
     { 5, 25 }
   }
```


Репозиторий ВГУ

```
   int i, j;
   for(i=0; i < 5; i++) {
    for(j=0; j < 2; j++)
     Console.Write(s[i,j] + " " ) ;
    Console.WriteLine();
   }
  }
 }
```

Результаты выполнения этой программы:

1 1
2 4
3 9
4 16
5 25

C# 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, чтобы объявить двумерный рваный массив, используется следующий формат записи:

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

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

Пример 3

```
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. Например, эта инструкция создает массив двумерных массивов:

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

Присвоим элементу gg [0] ссылку на массив размером 4x2:

```
gg[0] = new int[4][2];
```

Следующая инструкция присваивает значение переменной i элементу gg[0][1,0].

```
gg[0][1,0] = i;
```

Репозиторий ВГУ

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.

4.2. 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 **foreach** будет последовательно получать значения элементов изколлекции.

Элемент *коллекция* служит для указания опрашиваемой коллекции. Таким образом, элемент *тип* должен совпадать (или быть совместимым) с базовым типом массива. С помощью **foreach**, невозможно изменить содержимое коллекции.

Пример 3**.** Создать массив для хранения целых чисел и присвоить его элементам начальные значения. Затем вывести элементы массива, и попутно вычислить их сумму.

```
// Использование цикла foreach.
using System;
   class ForeachDemo {
     public static void Main() {
      int sum = 0;
      int[] n = new int[10];
      for(int i = 0; i < 10; i++)
      n[i] = i;    //Присваиваем элементам массива n значения
       foreach(int x in n)  //Используем цикл foreach для вывода
                    //значений элементов массива и их суммирования,
      {
       Console.WriteLine("Значение элемента равно: " +
       x);
       sum += x; }
      Console.WriteLine("Сумма равна: " + sum);
   }
 }
```

При выполнении этой программы получим следующие результаты:

Значение элемента равно: 0

Значение элемента равно: 1

Репозиторий ВГУ

…

Значение элемента равно: 9

Сумма равна: 45

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

Пример 4**.** Использование цикла **foreach** с двумерным массивом.

```
using System;
  class ForeachDemo2 {
    public static void Main() {
      int sum = 0;
      int[,] n = new int[3,5];
      // Присваиваем элементам массива n значения
      for(int i = 0; i < 3; i++)
      for(int j =0; j < 5; j++)
        n[i,j] =(i+1)*(j+1);
      // Используем цикл foreach для вывода значений
      // элементов массива и их суммирования,
      foreach(int x in n) {
      Console.WriteLine("Значение элемента равно:  "+x);
      sum += x;
         }
        Console.WriteLine("Сумма равна:  " + sum);
  }
}
```

4.3. Индексаторы.

Индексатор представляет собой разновидность свойства. Он 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 [<спи-
сок_параметров>]
{
  get код_доступа
  set код_доступа }
```

Спецификаторы аналогичны спецификаторам свойств и методов. *Индексаторы* чаще всего объявляются со спецификатором **public**, поскольку

Репозиторий ВГУ

они входят в интерфейс объекта. Атрибуты и спецификаторы могут отсутствовать.

Здесь *тип* – базовый тип индексатора. Он соответствует базовому типу массива.

Код доступа представляет 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 (`read-only`), если отсутствует часть **get**, индексатор доступен только для записи (`write-only`).

Если обращение к объекту встречается в левой части оператора присваивания, автоматически вызывается метод **get**. Если обращение выполняется в составе выражения, вызывается метод **set**.

Список параметров содержит одно или несколько описаний индексов, по которым выполняется доступ к элементу.

Индексаторы в основном применяются для создания специализированных массивов, на работу с которыми накладываются какие-либо ограничения.

Пример 5. 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
  int[] a;                   //закрытый массив
  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; }
```


Репозиторий ВГУ

```
      }
     set{
      if ( i >= 0 && i < length && value >= 0 && value
      <= 100 ) a[i] = value;
      else error = true; }
     }
   }
class Class1
 {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] );//использование индексатора
   }
   if (sa.error) Console.Write ("Были ошибки!");
  }
 }
}
```

Начиная с версии C# 2.0 допускается раздельное указание спецификаторов доступа для блоков получения и установки индексатора, аналогично свойствам.

В классе SafeArray принята следующая стратегия обработки ошибок: если при попытке записи элемента массива его индекс или значение заданы неверно, значение элементу не присваивается; если при попытке чтения элемента индекс не входит в допустимый диапазон, возвращается 0; в обоих случаях формируется значение открытого поля `error`, равное **`true`**.

Вообще говоря, индексатор не обязательно должен быть связан с каким-либо внутренним полем данных.

Пример 6. Индексатор без массива для нахождения степени числа 2.

```
using System;
namespace ConsoleApplication1
 {
  class Pow2
   {public ulong this[int i] {
     get
      {
       if ( i >= 0 )
       {
```


Репозиторий ВГУ

```
        ulong res = 1;
        for (int k = 0; k<i; k++) //цикл получения степени
        unchecked { res *= 2; } //1
        return res;
         }
         else return 0;
       }
     }
   }
   class Class1
   {static void Main()
    {
      int n = 5;
      Pow2 pow2 = new Pow2();
      for ( int i = 0; i < n; ++i )
      Console.WriteLine( "{0}\t{1}", i, pow2[i] );
    }
   }
 }
```

В программе исключение, связанное с переполнением, не генерируется, так как используется непроверяемый контекст (**unchecked**. Результат работы программы:

0 1
1 2
2 4
3 8
4 16

Язык C# 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. Каковы возможные способы описания массивов (одномерных и многомерных)?
3. В каких случаях целесообразно описывать двумерный массив с по-

Репозиторий ВГУ

мощью одномерных?
4. Какие типы допустимы для описания индексов массивов?
5. Какие типы могут использоваться в качестве базовых для описания массивов?
6. Как осуществляется ввод и вывод массивов?
7. Для чего предназначен цикл **`foreach?`**
8. Можно ли использовать цикл **`foreach`** для ввода элементов массива?
9. Как определяется базовый тип индексатора?
10. Что записывается в качестве имени индексатора?
11. Что содержит список параметров индексатора?

Лабораторная работа № 4.

Задание 1. Создайте проект, в котором опишите класс для решения задачи Вашего варианта. Каждый разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы с параметрами и без параметров, методы, свойства, индексаторы.

Класс должен реализовывать следующие операции над массивами:

- задание произвольной размерности массива при создании объекта;
- доступ к элементу по индексам с контролем выхода за пределы массива;
- вывод на экран элемента массива по заданному индексу и всего массива.

При возникновении ошибок должны выбрасываться исключения.

В программе должна выполняться проверка всех разработанных элементов класса.

Варианты заданий:

1. Описать класс, реализующий тип данных «вещественный массив» и работу с ним. Класс должен реализовывать метод, проверяющий является ли матрица симметричной.
2. Описать класс для работы с двумерным массивом символов, состоящих из одних цифр. Обеспечить следующие возможности: рассматривая символы как числа, определить сумму четных и нечетных цифр в каждой строке.
3. Описать класс для работы с одномерным массивом целых чисел. Обеспечить возможность нахождения суммы элементов, стоящих по-

Репозиторий ВГУ

сле введенного с клавиатуры значения и вывода элементов, стоящих после введенного с клавиатуры значения.

4. Описать класс «список», состоящий из номеров зачетной книжки и годов рождений студентов. Класс должен реализовывать метод, вычисляющий, сколько лет студенту и вывод на экран информации вида «номер зачетной книжки – количество лет».
5. Описать класс, реализующий тип данных «матрица». Класс должен реализовывать возможность нахождения количества столбцов, начинающихся с отрицательного числа.
6. Описать класс, реализующий тип данных «матрица целых чисел». Класс должен реализовывать метод проверки, является ли матрицы верхней треугольной.
7. Описать класс, реализующий тип данных «вещественная матрица». Класс должен реализовывать возможность преобразования матрицы следующим образом: каждый элемент строки разделить на максимальный элемент этой строки, если он не равна 0. В противном случае элементы строки оставить без изменений.
8. Описать класс для работы с одномерным массивом вещественных чисел. Предусмотреть возможность добавления элемента в массив по заданному индексу.
9. Описать класс для работы с двумерным числовым массивом. Обеспечить возможность проверки, является ли элемент массива палиндромом. Палиндром принимает одно и то же значение при чтении его как справа налево, так и слева направо.
10. Описать класс для работы с двумерным массивом целых чисел. Предусмотреть возможность поиска элемента в массиве по заданному значению.
11. Описать класс для работы с одномерным массивом вещественных чисел. Предусмотреть возможность удаления элемента из массива по заданному индексу.
12. Описать класс для работы с одномерным массивом целых чисел. Обеспечить возможность определения, является ли массив упорядоченным.

Репозиторий ВГУ

5. ОПЕРАЦИИ КЛАССА.
СТРОКИ.

Цель: освоить приемы создания операций класса и их использование; выработать навыков использования строковых переменных для обработки текстовой информации; сформировать умения разрабатывать простейшие проекты, обеспечивающие непротиворечивый и удобный интерфейс класса.

5.1. 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 ее исходных значений не меняется (Приложение 3).

Операции класса описываются с помощью методов специального вида (операций). Перегрузка операций похожа на перегрузку обычных методов. Синтаксис операции:

```
<спецификаторы> <тип> operator op(<тип> <операнд1>
[, <тип> <операнд2]>)
{
//тело операции
}
```

В качестве *спецификаторов* одновременно используются ключевые слова **`public`** и **`static`**. Кроме того, операцию можно объявить как внешнюю (**`extern`**).

По ключевому слову **`operator`** опознается описание операции в классе. Элемент *op* – это операция (например " + " или " / "), которая перегружается. *Тело* операции определяет действия, которые выполняются при использовании операции в выражении. Тело представляет собой блок.

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;
- сигнатуры всех операций класса должны различаться;

Репозиторий ВГУ

- типы, используемые в операции, должны иметь не меньшие права доступа, чем сама операция (то есть должны быть доступны при использовании операции).

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

5.1.1 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 +, -, !, ~ 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.

Префиксный и постфиксный инкременты не различаются (для них может существовать только одна реализация, которая вызывается в обоих случаях).

Пример 1. Определение унарных операций для класса трехмерных координат.

```
using System;
class TD // Класс трехмерных координат,
{
int x, y, z; //  3-х-мерные координаты.
public TD () {x = y = z = 0;} //Конструктор без параметров
public TD(int i, int j, int k)  // Конструктор с параметрами
{
x = i; y = j; z = k; }
    public static TD operator ++(TD op)  // Перегрузка
                                          // унарного оператора "++".
      {
```


Репозиторий ВГУ

```
        op.x++;  // Оператор "++" модифицирует аргумент.
        op.y++;
        op.z++;
        return op;
      }
    public static TD operator -(TD op) //Перегрузка
                                       // унарного оператора "-".
    {
        TD r = new TD();
        r.x = -op.x;
        r.y = -op.y;
        r.z = -op.z;
        return r;
}
public void show()  // Отображаем координаты X, Y, Z.
{
Console.WriteLine(x + ", " + y + ", " + z) ;
}
}
class TDDemo
{
public static void Main()
{
TD a = new TD(1, 2, 3);
TD b = new TD(0, 10, 10);
TD c = new TD();
Console.Write("Координаты точки a: " ) ;
a.show();
Console.Write("Координаты точки b: " ) ;
b. show() ;
  c = -a; // Присваивание -а объекту с.
  Console.Write("Результат присваивания -а: " ) ;
  c.show();
  Console.WriteLine();
  a++;  // Инкрементирование а.
  Console.Write ("Результат инкрементирования a++:  ")  ;
  a.show();
Console.WriteLine();
}
}
```

Репозиторий ВГУ

5.1.2 Бинарные операции

В классе сожно определять следующие бинарные операции:
+ , - , * , / , | , & , | | , && , == , =! , > , < , >= , <=

Примеры заголовков бинарных операций:

```
public static MyObj operator + { MyObj ml. MyObj m2)
public static bool operator == (Ob ml, Ob 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. Чаще всего в классе определяют операции сравнения на равенство и неравенство для того, чтобы обеспечить сравнение объектов, а не их ссылок, как определено по умолчанию для ссылочных типов.

Пример 2. Определение операции сложения и вычитания для класса `TD` (пример 1).

```
using System;
class TD // Класс трехмерных координат,
{
int x, y, z; // 3-х-мерные координаты .
public TD () { x = y = z = 0; }
public TD(int i, int j, int k)
{
x = i; y = j; z = k; }
public static TD operator +(TD opl, TD op2) //Перегрузка
                                            //бинарной операции "+".
{
TD r = new TD();
r.x = opl.x + op2.x;                 // Эти операторы выполняют
r.y = opl.y + op2.y;              // целочисленное сложение двух точек
r.z = opl.z + op2.z;
return r;
}
public static TD operator -(TD opl, TD op2)
    // Перегрузка бинарной операции "-".
{
TD r = new TD();
r.x = opl.x - op2.x; // Эти операторы выполняют
r.y = opl.y - op2.y; // целочисленное вычитание,
r.z = opl.z - op2.z;
return r;
```

Репозиторий ВГУ

```
}
public void show()  // Отображаем координаты X, Y, Z.
{
Console.WriteLine(x + ", " + y + ", " + z) ;
}
}
class TDDemo
{
public static void Main()
{
TD a = new TD(1, 2, 3);
TD b = new TD(0, 10, 10);
TD c = new TD();
Console.Write ("Координаты точки a: " ) ;
a.show();
Console.Write("Координаты точки b: ");
b. show() ;
c = a + b;                 // Складываем а и b.
Console.Write("Результат сложения a + b: ");
c.show();
c = a + b + c;             // Складываем а, b и с.
Console.Write("Результат сложения a+b+c:");
c.show();
c = c - a;                 // Вычитаем а из с.
Console.Write("Результат вычитания c - a: ");
c.show();
c = c - b;                 // Вычитаем b из с.
Console.Write("Результат вычитания c - b: ");
c.show();
Console.WriteLine();
}
}
```

5.2. Строковые величины

В C# строки являются объектами. Предназначены для работы со строками символов в кодировке ***Unicode***, являются встроенным типом ***C#***. Ключевое слово типа – **`string`.** Ему соответствует базовый класс ***System.String*** библиотеки **.NET.** Таким образом, **`string`** – это ссылочный тип. Строки типа **`string`** относятся к так называемым неизменяемым типам данных

Репозиторий ВГУ

5.2.1 Создание строк

Создать строку можно несколькими способами:

string <имя_строки>; // инициализация отложена.

Самый простой способ создать объект типа **string** – использовать строковый литерал. После выполнения инструкции

string <имя_строки>=<строковый литерал>;

будет объявлена ссылочной переменной типа **string**, которой присваивается ссылка на строковый литерал.

Например, **string** str = "test";

Можно также создать *string*-объект из массива типа **char**.

Пример 3:

```
char[] ca = {'t', 'e', 's', 't'};// массив для инициали
                                    зации строки
string str = new string(ca);
```

После создания **string**-объект можно использовать везде, где разрешается использование строки символов, заключенной в кавычки.

5.2.2 Работа со строками

Для строк определены следующие операции*:*

– присваивание (=);
– проверка на равенство (==);
– проверка на неравенство (!=);
– обращение по индексу ([]);
– сцепление (конкатенация) строк (+).

Несмотря на то, что строки являются ссылочным типом данных, на равенство и неравенство проверяются не ссылки, а значения строк. Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.

Как и у массивов, индексация строк начинается с нуля. Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения, так как строки типа **string** являются неизменяемым типом данных.

В классе ***System.String*** предусмотрено множество методов, полей и свойств, позволяющих выполнять со строками практически любые действия (табл. 4.4 приложения 4).

Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые «старые» копии автоматически удаляются сборщиком мусора.

Пример 4. Работа со строками типа string

```
using System;
```


Репозиторий ВГУ

```
namespace ConsoleApplicationl
{class Classl
  {static void Main()
   {int result, idx;
     string s = "прекрасная чашка с кофе";
     Console.WriteLine( s );
     //выделить подстроку с позиции 3 и удалить 2 символа с позиции 14
     string sub = s.Substring(3).Remove(14,2);
     Console.WriteLine( sub );
     Console.WriteLine("Длина строки sub: "
     +sub.Length);
     string[] mas = s.Split(' '); //занести строку в
                                  //массив, разделяя по пробелам
     string j = string.Join("- ", mas); // Слияние
          //массива строк в единую строку, вставляя разделитель "– "
     Console.WriteLine( j );
     Console.WriteLine( "Введите строку" );
     string str2 = "Один Два Три Один";
     // Поиск символа в строке.
     idx = str.IndexOf("Один");
     Console.WriteLine("Индекс первого вхождения под-
     строки Один:"+ idx);
     idx = str.LastIndexOf("Один");
     Console.WriteLine("Индекс последнего вхождения
     подстроки Один: " + idx);
    }
  }
 }
```

Результат работы программы:

прекрасная чашка с кофе
красная чашка кофе
Длина строки sub: 18
красная- чашка- кофе-
Индекс первого вхождения подстроки Один: 0
Индекс последнего вхождения подстроки Один: 13

Подобно другим типам данных, строки могут быть собраны в массивы.

Пример 5 Использование массивов строк.

```
using System;
  class StringArrays {
    public static void Main() {
      string[] str = {"Это","очень","простой","тест"};
```

Репозиторий ВГУ

```
        Console.WriteLine("Исходный массив: " );
        for(int i=0; i < str.Length; i++)
          Console.Write (str[i] + " " ) ;
        Console.WriteLine("\n");
        str[l] = "тоже";        // Изменяем строку.
        str[3) = "тест, не так ли?";
        Console.WriteLine("Модифицированный массив: ");
        for(int i=0; i < str.Length;
          Console.Write (str[i] + " " ) ;
    }
}
```

После выполнения этой программы получаем такие результаты:

Исходный массив:

Это очень простой тест.

Модифицированный массив:

Это тоже простой тест, не так ли?

Для управления **switch**-инструкциями также можно использовать **string**-объекты, причем это единственный тип, который там допускается, помимо типа **int**. Эта возможность в некоторых случаях облегчает обработку.

Ссылочные переменные типа **string** могут менять объекты, на которые они ссылаются. А содержимое созданного **string**- объекта изменить уже невозможно.

5.2.3 Класс StringBuilder

В C# предусмотрен класс **StringBuilder**, который определен в пространстве имен ***System.Text*** и создает объекты, которые можно изменять.

При создании экземпляра обязательно использовать операцию **new** и конструктор, например:

```
StringBuilder a = new StringBuilder();              //1
StringBuilder b = new StringBuilder("text"); //2
StringBuilder c = new StringBuilder( 100 );      //3
StringBuilder d = new StringBuilder("text",100);//4
StringBuilder e =new StringBuilder("text",1,3,10);//5)
```

В конструкторе класса указываются два вида параметров: инициализирующая строка или подстрока и объем памяти, отводимой под экземпляр *(*емкость буфера*)*. Один или оба параметра могут отсутствовать, в этом случае используются их значения по умолчанию.

Если применяется конструктор без параметров (оператор 1), создается пустая строка размера, заданного по умолчанию (16 байт). Другие виды

Репозиторий ВГУ

конструкторов задают объем памяти, выделяемой строке, и/или ее начальное значение. Например, в операторе 5 объект инициализируется подстрокой длиной 3 символа, начиная с первого (подстрока "tex").

В классе **`StringBuilder`** предусмотрены методы, и свойства, позволяющие менять содержимое строк (табл. 4.5 приложения 4).

Контрольные вопросы:

1. Что представляет собой перегрузка методов?
2. Что представляет собой перегрузка операций
3. Формат описания операции класса.
4. Какие операции нельзя перегружать?
5. Что является результатом перегрузки унарных операций?
6. Какие параметры могут быть у бинарных операций класса?
7. Как выполняется перегрузка операций отношения?
8. Чем являются строки в C#?
9. Какие операции определекны для строк?
10. Как создаются строки?
11. Можноли изменять значение строки?

Лабораторная работа № 5.

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

Каждый разрабатываемый класс должен, содержать следующие элементы: скрытые и открытые поля, конструкторы (один из них должен передавать параметром массив), перегруженные операции.

В программе должна выполняться проверка всех разработанных элементов класса.

Варианты заданий:

1. Описать класс для работы с одномерным массивом целых чисел (вектором). Класс должен реализовывать возможность: выполнение операции нахождения остатков от деления всех элементов массива на скаляр.
2. Описать класс для работы с одномерными массивами чисел. Класс должен реализовывать возможность: выполнения для массивов комбинированных операций присваивания (+=, -=).

Репозиторий ВГУ

3. Описать класс для работы с одномерным массивом строк фиксированной длины. Обеспечить следующие возможности: сравнения массивов на равенство (перегрузку операции == для поэлементного сравнения строк).
4. Описать класс, реализующий тип данных «вещественная матрица». Класс должен реализовывать следующие операции над матрицами: вычитание заданной номером строки из всех остальных строк, кроме данной строки.
5. Описать класс для работы с одномерным массивом строк фиксированной длины. Обеспечить следующие возможности: перегрузку операции + для поэлементного соединения массивов.
6. Описать класс для работы с n-мерным вектором. Класс должен реализовывать возможность: перегруженные операции отношений, выполняющие сравнение длин векторов;
7. Описать класс для работы с одномерным массивом чисел. Класс должен реализовывать возможность: выполнение операций поэлементного умножения массивов с одинаковыми границами индексов.
8. Описать класс для работы с одномерным массивом целых чисел (вектором). Класс должен реализовывать возможность: уменьшение количества элементов массива на заданное число (перегрузка операции -).
9. Описать класс для работы с одномерным массивом вещественных чисел. Обеспечить следующие возможности: нахождение суммы элементов массива (перегрузка операции +).
10. Описать класс, реализующий тип данных «вещественная матрица». Класс должен реализовывать следующие операции над матрицами: изменение значений элементов матрицы на противоположные.
11. Описать класс для работы с одномерным массивом целых чисел. Класс должен реализовывать возможность: нахождения числа, полученного перемножением положительных элементов массива.
12. Описать класс для работы с одномерным массивом чисел, позволяющий выполнять основные операции: добавление и удаление элемента в массив – перегруженные операции ++ и --.

Репозиторий ВГУ

6. ИЕРАРХИИ КЛАССОВ. НАСЛЕДОВАНИЕ

Данный раздел посвящен изучению приемов создания иерархии классов, выделению общих признаков объектов в базовый класс, умению организовать доступ к элементам базового и производных классов.

6.1. Наследование

Класс в C# 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.

Рассмотрим класс, в котором определяются двумерные геометрические фигуры (например, квадрат, прямоугольник, треугольник и т.п.).

Пример 1

```
using System;
        // Класс двумерных объектов,
class TwoF
{
  public double w;
  double h;
  public double height {
   get { return h; }
   set { h = value; }
   }
  public void showD()
  {
  Console.WriteLine("Ширина и высота равны "+w + " и "
  + height);
  }
}
        // Класс Treug выводится из класса TwoF.
class Treug:TwoF
```

Репозиторий ВГУ

```
{
 public string style; // Тип треугольника.
 public Treug()
 {w=0; height =0;
  style= "произвольный";
 }
 public double Pl() //Метод возвращает площадь треугольника
  {
  return w * height / 2;
  }
 public void showStyle()  //Отображаем тип треугольника,
 {
  Console.WriteLine("Треугольник " + style);
 }
}
class Class1
{
  public static void Main()
  {
   Treug tl = new Treug();
   Treug t2 = new Treug();
   tl.w = 4.0;
   tl. height = 4.0;
   tl.style = "равнобедренный";
   t2.w = 8.0;
   t2. height = 12.0;
   t2.style = "прямоугольный";
   Console.WriteLine("Информация о tl : ");
   tl.showStyle();
   tl.showD();
   Console.WriteLine ("Площадь равна"+tl.Pl());
   Console.WriteLine();
   Console.WriteLine("Информация о t2: ");
   t2. showStyle() ;
   t2.showD();
   Console.WriteLine("Площадь равна"+t2.Pl());
  }
 }
```

Результаты работы этой программы.

Информация о tl:

Треугольник равнобедренный

Ширина и высота равны 4 и 4

Репозиторий ВГУ

Площадь равна 8
Информация о t2:
Треугольник прямоугольный
Ширина и высота равны 8 и 12
Площадь равна 4 8

Класс Treug содержит все элементы класса TwoF и, кроме того, введены поле style , метод Pl() и метод showStyle(). В переменной style хранится описание типа треугольника.

Член h базового класса TwoF объявлен закрытым, поэтому он передается свойством.

6.1.1 Использование защищенного доступа

Защищенным является член, который открыт для своей иерархии классов, но закрыт вне этой иерархии. Защищенный член создается с помощью модификатора доступа **protected**.

Модификатор **protected** остается со своим членом независимо от реализуемого количества уровней наследования. Следовательно, при использовании производного класса в качестве базового для создания другого производного класса любой защищенный член исходного базового класса, который наследуется первым производным классом, также наследуется в статусе защищенного и вторым производным классом.

Рассмотрим простой пример использования защищенных членов класса.

Пример 2.

```
// Демонстрация использования защищенных членов класса.
using System;
class B
{
  protected int i, j; //Закрыт внутри класса B, но доступен
                                              // для класса D.
  public void set(int a, int b)
  {
    i = a;
    j = b;
  }
  public void show()
  {
    Console.WriteLine(i + " " + j);
  }
class D:B
{
  int k; // Закрытый член.
```

Репозиторий ВГУ

```
  public void setk()  // Класс D получает доступ к членам i и j
                                              //класса B.
   {
     k = i * j;
   }
  public void showk()
   {
     Console.WriteLine(k) ;
    }
  }
class ProtectedDemo
{
  public static void Main()
   {
     D ob = new D();
     ob.set(2, 3);        //D "видит" B-члены i и j .
     ob.show();           //D "видит" B-члены i и j  .
     ob.setk();           //это часть самого класса D .
     ob.showk();          //это часть самого класса D .
   }
  }
}
```

Поскольку в этом примере класс B наследуется классом D и члены i и j объявлены защищенными в классе B (т.е. с использованием модификатора доступа **protected**), метод setk() может получить к ним доступ.

6.2. 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

Репозиторий ВГУ

слова **base**.

Формат расширенного объявления таков:

```
имя_производиого_класса(список_параметров):base (спи-
сок_аргументов)
{
// тело конструктора
}
```

Здесь с помощью элемента *список_аргументов* задаются аргументы, необходимые конструктору в базовом классе.

С помощью ключевого слова **base** можно вызвать конструктор любой формы, определенный в базовом классе. Реально же выполнится тот конструктор, параметры которого будут соответствовать переданным при вызове аргументам. При отсутствии ключевого слова **base** автоматически вызывается конструктор базового класса, действующий по умолчанию.

Пример 3. Рассмотрим в следующей программе еще одну версию классов TwoF и Treug.

```
// Добавляем в класс TwoF конструкторы.
using System;
class TwoF
{
 public double w;        // Открытый член.  ;
 double h;               // Закрытый член .
 public TwoF()           // Конструктор по умолчанию,
 {
  w = h = 0.0;
 }
 public TwoF(double w1, double h1) //Конструктор TwoF с
                                   //параметрами
 {
   w = w1;
   h = h1;
 }
 public TwoF(double x)  // Создаем объект, у которого ширина
                        равна высоте
 {
   w = h = x;
 }
 public double height        // Свойство height.

 {
  get { return h; }
```


Репозиторий ВГУ

```
  set { h = value; }
 }
 public void showD()
 {
  Console.WriteLine("Ширина и высота = "+width+" и
  "+height);
 }
}
class Treug:TwoF //Класс  Treug, производный от класса TwoF.
{
  string style;  // Закрытый член.
    /* Конструктор по умолчанию. Он автоматически вызывает конструктор по
    умолчанию класса TwoF. */
 public Treug()
 {
  style = "null";
 }
      // Конструктор, который принимает три аргумента
 public Treug(string s,double w,double h): base(w,h)
 {
  style = s;
 }
 public Treug(double x):base(x) // Создаем
                                    //равнобедренный треугольник
 {
  style = "равнобедренный";
  }
 public double Pl()  // Метод возвращает площадь треугольника
 {
  return w * height / 2;
  }
 public void showStyle()  // Отображаем тип треугольника,
 {
  Console.WriteLine("Треугольник " + style);
 }
}
class Class4
{
 public static void Main()
 {
  Treug t1 = new Treug();
  Treug t2 = new Treug("прямоугольный", 8.0, 12.0);
```


Репозиторий ВГУ

```
  Treug t3 = new Treug(4.0);
  tl = t2;
  Console.WriteLine("Информация о tl: " ) ;
  tl.showStyle();
  tl.showD();
  Console.WriteLine ("Площадь равна "+tl.Pl());
  Console.WriteLine("Информация о t2: " );
  t2.showStyle();
  t2.showD();
  Console.WriteLine("Площадь равна "+t2.Pl());
  Console.WriteLine("Информация о t3: " );
  t3.showStyle();
  t3.showD ();
  Console.WriteLine("Площадь равна "+t3.Pl());
  }
 }
```

При выполнении этой версии программы получаем такие результаты:
Информация о tl:
Треугольник прямоугольный
Ширина и высота равны 8 и 12
Площадь равна 48
Информация о t2:
Треугольник прямоугольный
Ширина и высота равны 8 и 12 (
Площадь равна 48
Информация о t3:
Треугольник равнобедренный
Ширина и высота равны 4 и 4
Площадь равна 8

Таким образом, ключевое слово **base** всегда отсылает к базовому классу, стоящему в иерархии классов непосредственно над вызывающим классом. Это справедливо и для многоуровневой иерархии.

6.3. 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.

Когда имя члена в производном классе скрывает член с таким же именем в базовом классе, для обращения к последнему применяется

Репозиторий ВГУ

ссылка **base**, которая всегда 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.

Пример 4 Сокрытие имени в связи с наследованием. Использование ссылки **base** для доступа к скрытому имени.

```
using System;
class A                    // Базовый класс
{
 public int i = 0;
 public void show()  // Метод show() в классе A.
  {
   Console.WriteLine("i в базовом классе: "+i);
   }


}
class B:A                 // Создаем производный класс,
{
  new int i;              // Эта переменная i скрывает i класса А.
  public B(int a, int b)      //Конструктор
  {
   base.i = a;           //Так можно обратиться к полю i класса А.
   i = b;                // Переменная i в классе В.
  }
  new public void show()  // Этот метод скрывает метод
                          //show(),определенный в классе А.
  {
   base.show(); //Вызов метода show() класса А.
   Console.WriteLine("i в производном классе: "+i);
                       //Отображаем значение переменной i класса В.
   }
}
class Class2
{
 public static void Main() {
  B ob = new B(1, 2);
  ob.show();
```

Репозиторий ВГУ

```
    }
}
```

Результаты выполнения этой программы выглядят так:
i в базовом классе: 1
i в производном классе: 2

Поскольку в классе В определяется собственная переменная экземпляра с именем i , она скрывает переменную i, определенную в классе А. Следовательно, при вызове метода `show()` для объекта типа В, отображается значение переменной i, соответствующее ее определению в классе В, а не в классе А. Ссылка **base** позволяет получить доступ к полю i в базовом классе. При вызове `base.show()` происходит обращение к версии метода `show ()`, определенной в базовом классе.

Вызов одноименного метода предка из метода потомка всегда позволяет сохранить функции предка и дополнить их, не повторяя фрагмент кода. Помимо уменьшения объема программы это облегчает ее модификацию, поскольку изменения, внесенные в метод предка, автоматически отражаются во всех его потомках.

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?

Лабораторная работа № 6. Наследование

Задание 1. Составить программу с одним родительским классом и потомком. Все поля должны быть закрытыми. Базовый класс должен содержать конструкторы с параметрами, методы доступа к закрытым полям, вывод полей и указанный в таблице метод. Производный класс содержит дополнения и изменения, организовать вывод новых полей потомка, при этом имена методов совпадают с именами методов базового класса. Составить тестирующую программу с выдачей результатов. Создать объекты базового и производного типов. В программе должна выполняться проверка всех разработанных элементов класса.

Варианты заданий:

1. Базовый класс: Квартира (поля: название, стоимость 1м^2, площадь)
   Метод: Стоимость квартиры.

Репозиторий ВГУ

Потомок: Квартира в центре (поле название района)
Изменения в потомках: Увеличить стоимость с учетом надбавки за расположение на 0.01 стоимости квартиры.

2. Базовый класс: Сотрудник (поля: имя, p – минимальная зарплата)
   Метод: Доход k *p, где k – повышающий коэффициент.
   Потомок: Инженер (поле количество разработанных проектов - n)
   Изменения в потомках: Доход инженера увеличить в n/10 раз.
3. Базовый класс: Тетрадь (поле: название, количество листов –k)
   Метод: Стоимость: 15 k
   Потомок: Общая (поле – материал обложки)
   Изменения в потомках: Изменить стоимость с учетом надбавки за обложку на 50р.
4. Базовый класс: Животное (поля: кличка, рост – h в м)
   Метод: Вес животного – $k * h^3$ кг, где k-коэффициент.
   Потомок: Кошка (k= 15, поле – порода животного)
   Изменения в потомках: Перевести вес животного в граммы.
5. Базовый класс: Поле (поля: название, r – вес посеянных семян на единицу площади)
   Метод: Количество урожая с единицы площади:k*r, где k – коэффициент.
   Потомок: Картофельное (поле S – площадь поля)
   Изменения в потомках: Найти урожай со всего поля.
6. Базовый класс: Стол (поля: название, площадь S в см)
   Метод: Стоимость $C=S^2/3+k$, где k – коэффициент.
   Потомок: Письменный (поле – используемый материал, стоимость отделки)
   Изменения в потомках: Найти стоимость отделки, определяемую как 10% от стоимости и полную стоимость.
7. Базовый класс: Автобус (поля: количество пассажиров, стоимость билета)
   Метод: Общая стоимость всех мест
   Потомок: Скорый (поле – скорость, марка автобуса)
   Изменения в потомках: Найти общую стоимость всех мест с учетом увеличения цены билета на 0.05скорости.
8. Базовый класс: Отрезок (поле: название, координаты)
   Метод: Длина отрезка.
   Потомок: Линия (поле – цвет линии, n – коэффициент увеличения)
   Изменения в потомках: Увеличение длины отрезка в n раз.
9. Базовый класс: Студент (поле: имя, средний балл s)
   Метод: Стипендия 300000+10000([s]-5)
   Потомок: Магистр (поле – специальность)
   Изменения в потомках: Увеличить стипендию на m руб.

Репозиторий ВГУ

10. Базовый класс: Телефон (поле: количество функций –k.)
    Метод: Стоимость: $40\ln(k)$
    Потомок: Сотовый (поле – модель)
    Изменения в потомках: Увеличить стоимость в 3 раза.
11. Базовый класс: Квадрат (поле – название, сторона)
    Метод: Площадь квадрата.
    Потомок: Прямоугольник (поле – разность между длинами сторон)
    Изменения в потомках: Изменить вычисление площади фигуры.
12. Базовый класс: Товар (поле – количество, цена.)
    Метод: Стоимость товара.
    Потомок: Фломастеры (поле – название, сорт – s)
    Изменения в потомках: Изменить стоимость фломастеров с учетом сорта в $1/\sqrt{s}$ раз.

Репозиторий ВГУ

7. ВИРТУАЛЬНЫЕ МЕТОДЫ И ПОЛИМОРФИЗМ.

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

Полиморфизм в различных формах является мощным и широко применяемым инструментом объектно-ориентированного программирования. Принцип полиморфизма реализуется с помощью виртуальных методов, которые обеспечивают гибкость и возможность расширения функциональности класса.

7.1. Виртуальные методы

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.

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 связыванием и реализуется с помощью так называемых виртуальных методов.

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

virtual public void P() ...

Слово **virtual** в переводе с английского значит «фактический». Объявление метода виртуальным означает, что все ссылки на этот метод будут разрешаться по факту его вызова, то есть не на стадии компиляции, а во время выполнения программы.

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

override public void P() ...

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

Репозиторий ВГУ

бование вполне естественно, так как одноименные методы, относящиеся к разным классам, могут вызываться из одной и той же точки программы.

Пример 1. Виртуальные методы.

```
using System;
namespace ConsoleApplicationl
{
  class Base                    //базовый класс
  {
   public int a;
   public Base (int i)
   {
    a = i;
   }
   virtual public void Print() //виртуальный метод для вывода поля
   a
   {
    Console.WriteLine( "Base " +a);
   }
  }
  class Potomok : Base      //производный класс от Base
  {
   public Potomok (int j):base(j)
   {
   }
   override public void Print()  //переопределяем метод для вы-
   вода поля потомка
   {
    Console.WriteLine("Potomok "+a);
   }
  }
  class Classl
  {static void Main()
   {
    const int n = 3;
    Base[] st = new Base[n];  //создаем контейнер ссылок
                              //на объекты базового класса
    st[0] = new Base(1);      //создаем объект класса Base
    st[1] = new Base(2);      //создаем объект класса Base
    st[2] = new Potomok(3);   //создаем объект класса Potomok
    //выводим поля объектов массива
    foreach ( Base elem in st ) elem.Print();
    //обнуляем поля объектов
```


Репозиторий ВГУ

```
    for ( int i = 0; i < n; ++i ) st[i].a = 0; Con-
    sole.WriteLine();
    //выводим поля объектов массива
    foreach ( Base elem in st ) elem.Print();
  }
}
```

Результат работы программы:
Base 1
Base 2
Potomok 3

Base 0
Base 0
Potomok 0

В циклах вызывается метод `Print()`, соответствующий типу объекта, помещенного в массив.

Виртуальные методы базового класса определяют интерфейс всей иерархии. Этот интерфейс может расширяться в потомках за счет добавления новых виртуальных методов. Переопределять виртуальный метод в каждом из потомков не обязательно: если он выполняет устраивающие потомка действия, метод наследуется.

Виртуальные методы незаменимы и при передаче объектов в методы в качестве параметров. В параметрах метода описывается объект базового типа, а при вызове в нее передается объект производного класса. В этом случае виртуальные методы, вызываемые для объекта из метода, будут соответствовать типу аргумента, а не параметра.

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.

7.2. Абстрактные классы

При создании иерархии объектов для исключения повторяющегося кода часто бывает логично выделить их общие свойства в один родительский класс. При этом может оказаться, что создавать экземпляры такого класса не имеет смысла, потому что никакие реальные объекты им не соответствуют. Такие классы называют абстрактными.

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

Репозиторий ВГУ

представления общих понятий, которые предполагается конкретизировать в производных классах.

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**.

Абстрактный класс может содержать и полностью определенные методы.

Если в классе есть хотя бы один абстрактный метод, весь класс также должен быть описан как абстрактный.

Пример 2.

```
using System;
namespace ConsoleApplication2
abstract class Spirit
{
  public abstract void Print();
}
class Base : Spirit
{
  override public void Print()
  {
    Console.WriteLine( "Base "+a);
  }
class Potomok : Base
{
  override public void Print()
  {
    Console.WriteLine(" Potomok"+a);
  }
}
```

Абстрактные классы используются при работе со структурами данных, предназначенными для хранения объектов одной иерархии, и в качестве параметров методов. 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.

Полиморфизм в различных формах является мощным и широко применяемым инструментом ООП.

Репозиторий ВГУ

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?

Лабораторная работа № 7. Полиморфизм. Виртуальные методы

Задание 1.

Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с указанием названий и полей и их значений соответствующего объекта. Составить тестирующую программу с выдачей протокола на экран монитора. При этом создать объекты базового и производных типов, используя полиморфный контейнер - массив ссылок базового класса на объекты базового и производных классов (количество объектов >=5).

Варианты заданий:

Варианты	Родительский класс	Потомки	Полиморфные методы
1.	Многочлен (поле название)	Квадратный Кубический (поля – коэффициенты многочленов, неизвестная x)	Значение многочлена
2.	Транспортное средство (поле - название)	Самолет (высота - h, скорость - v) Корабль (количество пассажиров k, порт приписки)	Стоимость транспортного средства Самолет 100*h*v Корабль $5k^2$

Репозиторий ВГУ

3.	Населенный пункт (поле название)	Село (поле количество домов-h, число жителей в доме, площадь села) Город (поле количество жителей- h, площадь города)	Плотность населения
4.	Постройка (поле название)	Офис (поле - количество этажей N) Завод (поле - Вес G)	Высота фундамента Офис 0,05*N Завод 0,000002*G
5.	Автомобиль (поле название)	Грузовой (поле – грузоподъемность p в т) Легковой (поле – объем двигателя V в $см^3$)	Расход горючего на 100км Грузовой $M = \sqrt{p} * 100$ Легковой M=2,5*V
6.	Одежда (поле - название)	Пальто (поле размер V) Костюм (поле рост H)	Расход ткани Пальто V/6.5+0.5 Костюм 2*H+0.3
7.	Мебель (поле - название)	Шкаф (поле объем V в см) Диван (поле площадь S в см)	Стоимость мебели Шкаф $C = \sqrt{V} * 0{,}75$ Диван $C=S^2/3+5000$
8.	Птица (поле название породы)	Аист (поле – размах крыльев L в см) Ворона (поле L –высота в см)	Количество пищи в день аист L*1/2000; ворона 0,8*L
9.	Вектор (поле - название)	Двумерный вектор (поля: компоненты вектора) Трехмерный вектор (поле размерность)	Длина вектора
10.	Печатная продукция (поле - название)	Журнал (поля: тираж, цена), газета (поля количество листов, стоимость листа, тираж)	Стоимость тиража.
11.	Медработник (поле имя, должность)	Медсестра, (поле – количество отработанных часов за неделю - p) Врач (поле–количество	Доход Доход медсестры - - 10*p Доход врача = 8*p

Репозиторий ВГУ

		принятых пациентов за неделю - р)	
12.	Четырехугольник (поле – название)	Прямоугольник (поля – длины сторон), Квадрат (поле – сторона)	Площадь фигуры

Задание 2. Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 1. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов >=5).

Варианты заданий:

1. Организовать нахождение наибольшего значения многочлена.
2. Организовать вычисление средней стоимости самолетов и средней стоимости кораблей.
3. Найти значение наименьшей плотности населения
4. Найти максимальную высоту фундаментов.
5. Организовать вычисление суммарного расхода горючего.
6. Организовать вычисление суммарного расхода ткани.
7. Организовать вычисление средней стоимости шкафов и средней стоимости диванов.
8. Найти количество пищи, необходимой аистам и количество пищи, необходимой воронам.
9. Организовать вычисление суммарной длины векторов.
10. Организовать вычисление средней стоимости журналов и средней стоимости газет.
11. Организовать вычисление суммарной величины дохода.
12. Найти махсимальную из площадей.

Репозиторий ВГУ

8. ИНТЕРФЕЙСЫ И СТРУКТУРНЫЕ ТИПЫ

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

8.1. Синтаксис интерфейса

Интерфейс является специальным видом классов. В нем задается набор абстрактных методов, свойств и индексаторов, которые должны быть реализованы в производных классах. Иными словами, интерфейс определяет поведение, которое поддерживается реализующими этот интерфейс классами. Основная идея использования интерфейса состоит в том, чтобы к объектам таких классов можно было обращаться одинаковым образом.

Каждый класс может определять элементы интерфейса по-своему. Так достигается полиморфизм: объекты разных классов по-разному реагируют на вызовы одного и того же метода. Синтаксис интерфейса аналогичен синтаксису класса:

```
[атрибуты][спецификаторы] interface <имя_интерфейса>
[:предки] <тело_интерфейса> [;]
```

Для интерфейса могут быть указаны спецификаторы, **new**, **public**, **protected**, **internal** и **private**. Спецификатор **new** применяется для вложенных интерфейсов и имеет такой же смысл, как и соответствующий модификатор метода класса. Остальные спецификаторы управляют видимостью интерфейса. По умолчанию интерфейс доступен только из сборки, в которой он описан (**internal**). Интерфейс может наследовать свойства нескольких интерфейсов, в этом случае предки перечисляются через запятую. 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.

В интерфейсе имеет смысл задавать заголовки тех методов и свойств, которые будут по-разному реализованы различными классами разных иерархий. Интерфейсы же чаще используются для задания общих свойств объектов различных иерархий. Отличия интерфейса от абстрактного класса:

• элементы интерфейса по умолчанию имеют спецификатор доступа **public** и не могут иметь спецификаторов, заданных явным образом;

Репозиторий ВГУ

• интерфейс не может содержать полей и обычных методов – все элементы интерфейса должны быть абстрактными;

• класс может иметь в списке предков несколько интерфейсов, при этом он должен определять все их методы.

8.2. Реализация интерфейса

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. Таким образом, в C# поддерживается одиночное наследование для классов и множественное – для интерфейсов. Это позволяет придать производному классу свойства нескольких базовых интерфейсов, реализуя их по своему усмотрению.

Формат записи класса, который реализует интерфейс, таков:

```
class <имя_класса> : <имя__интерфейса>
{
// тело класса
}
```

Методы, которые реализуют интерфейс, должны быть объявлены открытыми. Кроме того, 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.

Рассмотрим пример интерфейса для класса, который генерирует ряд чисел.

```
public interface ISeries
{
 int getNext();          // Возвращает следующее число ряда
 void reset ();          // Выполняет перезапуск
 void setStart(int x);  //Устанавливает начальное значение.
}
```

Интерфейс ISeries объявлен открытым, поэтому он может быть реализован любым классом в любой программе.

Пример 1. Создать класс, генерирующий ряд чисел, в котором каждое следующее число больше предыдущего на два, и класс, который генерирует первое простое число после заданного. В обоих классах реализовать интерфейс ISeries.

```
//Использование интерфейса ISeries для реализации арифметической прогрессии с разностью 2
class NaDva:ISeries
{
 int start;    // начальное значение члена ряда
```


репозиторий ВГУ

```
 int val;       // очередное значение членов ряда
 public NaDva()
 {
   start = 0;
   val = 0;
  }
 public int getNext()
 {
   val += 2;
   return val;
  }
 public void reset()
 {
   val = start;
  }
 public void setStart(int x)
 {
   start = x;
   val = start;
  }
}
// Использование интерфейса ISeries для реализации ряда простых чисел.
class Primes : ISeries
{
 int start;
 int val;
 public Primes()
 {
  start = 2;
  val = 2;
 }
 public int getNext()
 {
  int i, j;
  bool b=false;  //признак простого числа
  i = val+1;          // берем число, следующее за val
  while(!b)
  {
   b = true;
   j = 2;              //делители, начиная с числа 2
   while (j < (i / j + 1) && b)
   {
```

Репозиторий ВГУ

```
    if ((i % j) == 0) //если число имеет делитель
    {
     b = false;
    }
    j++;
   }
   if (b) { val = i; } //если число не имеет делителей,
   i++;                // переходим к следующему числу
  }
  return val;
 }
 public void reset()
 {
  val = start;
 }
 public void setStart(int x)
 {
  start = x;
  val = start;
 }
}
```

/*Демонстрация использования интерфейса, реализованного классами NaDva и Primes*/

```
using System;
class SeriesDemo1
{
 public static void Main()
 {
   NaDva ob = new NaDva() ;
   for(int i=0; i < 5;
   Console.WriteLine("Следующее значение равно " +
   ob.getNext());
   Console.WriteLine("Переход в исходное
    состояние.")
   ob.reset();
   for(int i=0; i < 5; i++)
   Console.WriteLine("Следующее значение
   равно"+ob.getNext());
   Console.WriteLine("Начинаем с числа 100.");
   ob.setStart (100);
   for(int i=0; i < 5; i++)
```


Репозиторий ВГУ

```
   Console.WriteLine("Следующее значение
    равно"+ob.getNext());
   Primes ob1 = new Primes();
   ob1.setStart(55);
   for (int i = 0; i < 5; i++)
   {
   Console.WriteLine("Простые числа " +
   ob1.getNext());
  }
}
```

Чтобы скомпилировать программу `SeriesDemo` необходимо включить в процесс компиляции файлы, которые содержат классы `ISeries`, `NaDva` и `SeriesDemo`. Для этого необходимо добавить все эти три файла в свой проект. Вполне допустимо также поместить их в один файл.

Результаты выполнения этой программы:
Следующее значение равно 2
Следующее значение равно 4
Следующее значение равно 6
Следующее значение равно 8
Следующее значение равно 10
Переход в исходное состояние.
Следующее значение равно 2
Начинаем с числа 100.
Следующее значение равно 102
Простые числа 59
Простые числа 61
Простые числа 67
Простые числа 71
Простые числа 73

Здесь важно понимать, что, хотя классы Primes и NaDva генерируют разные ряды чисел, оба они реализуют один и тот же интерфейс **ISeries**.

8.3. Использование интерфейсных ссылок

Можно объявить ссылочную переменную интерфейсного типа. Такая переменная может ссылаться на любой объект, который реализует ее интерфейс. При вызове метода для объекта посредством интерфейсной ссылки будет выполнена та версия указанного метода, которая реализована этим объектом. Этот процесс аналогичен использованию ссылки на базовый класс для доступа к объекту производного класса.

Репозиторий ВГУ

Пример 2. Задание предыдущего примера. Использовать одну интерфейсную переменную-ссылку, для вызова методов объектов как класса NaDva, так и класса Primes.

```
// Демонстрация использования интерфейсных ссылок.
using System;
class SeriesDemo2
{
 public static void Main()
 {
  NaDva twoOb = new NaDva();  //создаем объект класса NaDva
  Primes primOb = new Primes();//создаем объект класса
                                //Primes
  ISeries ob;       //создаем интерфейсную ссылку
  for(int i=0; i < 5;i++){
  ob = twoOb;
  Console.WriteLine("Следующее четное число равно "
  +ob.getNext());
  ob = primOb;
  Console.WriteLine("Следующее простое число равно "
  +ob.getNext());
  ob.setStart(ob.getNext()+1);}
 }
}
```

Вот результаты выполнения этой программы:
Следующее четное число равно 2
Следующее простое число равно 3
Следующее четное число равно 4
Следующее простое число равно 5

В методе Main() объявляется переменная ob как ссылка на интерфейс ISeries. Это означает, что ее можно использовать для хранения ссылок на любой объект, который реализует интерфейс ISeries. В данном случае она служит для ссылки на объекты twoOb и primOb, которые являются экземплярами классов NaDva и Primes, реализующих один и тот же интерфейс ISeries.

Интерфейсную ссылочную переменную нельзя использовать для доступа к другим переменным или методам, которые может определить объект, реализующий этот интерфейс, но необъявленных в этом интерфейсе.

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

Репозиторий ВГУ

элементу, элемент какого именно из реализуемых интерфейсов требуется вызвать. При неявной реализации одноименных методов в разных интерфейсах, избежать неоднозначности можно с помощью бинарных операций **is** или **as**, которые позволяют убедиться, что объект поддерживает данный интерфейс или выполнить приведение к данному интерфейсу.

Пример 3. //Решение проблемы неоднозначности при неявной реализации интерфейсов.

```
using System;
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; }
 // Неявным образом реализуем два метода meth().
 public void meth(int x)
 {
  h= x + x;
  Console.WriteLine("meth..."+h);
 }
}
class Class1
{
 public static void Main()
 {
  MyClass ob = new MyClass();
  Console.Write("Вызов метода ob.meth(): ");
  ob.meth(3);
  Console.WriteLine("Реализация интерфейса IMy_A");
  (ob as IMy_A).meth(7);
  Console.WriteLine("Реализация интерфейса IMy_B");
  (ob as IMy_B).meth(7);
  Console.WriteLine("Вызов метода meth инт_ссылкой");
  IMy_A a_ob;        //интерфейсная ссылка
  a_ob = ob;  //интерфейсной ссылке присваиваем объект класса
```

Репозиторий ВГУ

```
  Console.WriteLine("Вызов метода IMy_A.meth()");
  a_ob.meth(3); //реализация интерфейса IMy_A
  IMy_B b_ob;       //интерфейсная ссылка
  b_ob = ob; //интерфейсной ссылке присваиваем объект класса
  Console.WriteLine("Вызов метода IMy_B.meth()");
  b_ob.meth(3); //реализация интерфейса IMy_B
 }
}
```

Результат работы программы:

Вызов метода ob.meth(): meth...6
Реализация интерфейса IMy_A
meth...14
Реализация интерфейса IMy_B
meth...14
Вызов метода meth инт_ссылкой
Вызов метода IMy_A.meth()
meth...6
Вызов метода IMy_B.meth()
meth...6

8.4. Явная реализация интерфейса

При реализации члена интерфейса можно квалифицировать его имя с использованием имени интерфейса. В этом случае говорят, что член интерфейса реализуется явным образом, или имеет место его явная реализация.

Имя интерфейса явно указывается перед реализуемым элементом через точку. Спецификаторы доступа при этом не указываются. К таким элементам можно обращаться в программе только через объект типа интерфейса.

8.4.1 Закрытая реализация

Реализуя метод с использованием полностью квалифицированного имени, мы обозначаем части закрытой реализации, которые недоступны вне класса, т.е. при явном задании имени реализуемого интерфейса соответствующий метод не входит в интерфейс класса. Это позволяет упростить код программы в том случае, если какие-то элементы интерфейса не требуются конечному пользователю класса.

Пример 4. Описать интерфейс, который определяет два метода: один – устанавливающий заканчиваются ли числа на 0, второй, что число на 0 не заканчивается. Второй метод реализовать явно.

```
// Явная реализация члена интерфейса.
```


Репозиторий ВГУ

```
using System;
interface I10
{
  bool is10(int x );
  bool isne10(int x);
}
class MyClass: I10
{
  bool I10.isne10(int x)  // Явная реализация

   {
    if((x%10) != 0) return true; else return false;
   }
  public bool is10(int x)  // Обычная реализация,
  {
    I10 o = this;        // Ссылка на вызывающий объект .
    return !o.isne10(x);
    }
  }
class Demo
{
  public static void Main()
  {
    MyClass ob = new MyClass();
    bool result;
    result = ob. is10(40);
    if (result) Console.WriteLine("число заканчивается
    0");
    else Console.WriteLine("число не заканчивается
    0");
    result = ob.isne10();  // Ошибка, член не виден.
    }
}
```

Поскольку метод `isne10` реализован в явном виде, он недоступен вне класса MyClass. Такой способ реализации делает его надежно закрытым. Внутри класса MyClass к методу `isne10` можно получить доступ только через ссылку на интерфейс.

8.4.2 Использование явной реализации при множественном наследовании

Кроме того, явное задание имени реализуемого интерфейса перед именем метода позволяет избежать конфликтов при множественном

Репозиторий ВГУ

наследовании, если элементы с одинаковыми именами или сигнатурой встречаются более чем в одном интерфейсе.

Конфликт возникает в том случае, если компилятор не может определить из контекста обращения к элементу, элемент какого именно из реализуемых интерфейсов требуется вызвать. В этой ситуации явная реализация используется для того, чтобы избежать неопределенности.

Пример 5. Реализовано два интерфейса, причем оба объявляют метод с одним именем.

```
//Использование явной реализации, чтобы избежать неоднозначности.
using System;
interface IMy_A
{
  int meth(int x);
}
interface IMy_B
{
  int meth(int x);
}
        // В классе MyClass реализованы оба интерфейса,
class MyClass:IMy_A, IMy_B
{// Явным образом реализуем два метода meth().
  int IMy_A.meth(int x)
  {
    return x + x;
  }
  int IMy_B.meth(int x)
  {
    return x * x;
  }
  // Вызываем метод meth() посредством ссылки на интерфейс,
  public int methA(int x)
  {
   IMy_A a_ob;
   a_ob = this;
   return a_ob.meth(x);  //Имеется в виду интерфейс IMy_A.
  }
  public int methB(int x)
  {
  IMy_B b_ob; i
  b_ob = this;
  return b_ob.meth(x);  // Имеется в виду интерфейс IMy_B
  }
```


Репозиторий ВГУ

```
}
class Demo
{
   public static void Main()
   {
    MyClass ob = new MyClass();
    Console.Write("Вызов метода IMy_A.meth():");
    Console.WriteLine(ob.methA(3));
    Console.Write ("Вызов метода IMy_B.meth(): ");
    Console.WriteLine(ob.methB(3));
   }
 }
```

Результаты выполнения этой программы:

Вызов метода IMy_A.meth(): 6

Вызов метода IMy_B.meth(): 9

Метод `meth()` имеет одинаковую сигнатуру в интерфейсах `IMy_A` и `IMy_B`. Следовательно, если класс `MyClass` реализует оба эти интерфейса, он должен реализовать каждый метод в отдельности, полностью указав его имя (с использованием имени соответствующего интерфейса). 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 `meth`(), объявленный в интерфейсе `IMy_B`, создает ссылку на интерфейс `IMy_B`. Созданные ссылки затем используются при вызове этих методов, благодаря чему можно избежать неоднозначности.

8.5. 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. Например, нельзя использовать интерфейс, описанный со спецификатором **`private`** или **`internal`**, в качестве базового для открытого (**`public`**) интерфейса'.

Как и в обычной иерархии классов, базовые интерфейсы определяют общее поведение, а их потомки конкретизируют и дополняют его. В интерфейсе-потомке можно также указать элементы, переопределяющие унаследованные элементы с такой же сигнатурой. В этом случае перед элементом указывается ключевое слово **`new`**, как и в аналогичной ситуации в классах. С помощью этого слова соответствующий элемент базового интерфейса скрывается.

Репозиторий ВГУ

Любой класс, который реализует интерфейс, должен реализовать все методы, определенные этим интерфейсом, включая методы, которые унаследованы от других интерфейсов.
Пример 6.

```
using System;
public interface A
{
  void methl() ;
  void meth2();
}
// Интерфейс В включает методы methl() и meth2(),
// а также добавляет метод meth3().
public interface B:A
{
  void meth3();
}
//Этот класс должен реализовать все методы интерфейсов А и В.
class MyClass:B
{
  public void methl()
  {
    Console.WriteLine ("Реализация метода methl”);
  }
  public void meth2()
  {
    Console.WriteLine("Реализация метода meth2");
  }
  public void meth3()
  {
    Console.WriteLine("Реализация метода meth3");
  }
}
class Program
{
  public static void Main()
  {
    MyClass ob = new MyClass();
    ob.methl();
    ob.meth2();
    ob.meth3();
   }
}
```

Репозиторий ВГУ

Результат работы программы:
Реализация метода methl
Реализация метода meth2
Реализация метода meth3

8.6. Реализация интерфейса виртуальными методами.

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 непереопределенная версия.

Пример 7. Переопределение методов, реализующих интрфейс.

```
using System;
interface IBase
{
 void A();
}

class Base : IBase
{
 public void A()
 {
  Console.WriteLine("Реализация в классе Base");}
}
class Potomok: Base
{
 new public void A()
 {
  Console.WriteLine("Реализация в классе Potomok");
  base.A(); //Вызывает метод класса Base
 }
 }
namespace интерф_невиртуал
{
 class Program
 {
  static void Main()
  {
   Potomok d = new Potomok();
   d.A(); // вызывается Potomok.A();
   IBase id = d;
   id.A(); // вызывается Base.A();
```


Репозиторий ВГУ

```
   }
  }
}
```

Результат работы программы:
Реализация в классе Potomok
Реализация в классе Base
Реализация в классе Base

Несмотря на то, что ссылке `id` присвоин объект класса Potomok вызывается метод класса Base

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

Пример 8. Использование виртуального метода.

```
using System;
interface IBase
{
 void A();
}

class Base : IBase
{
 public virtual void A()
 {
  Console.WriteLine("Реализация в классе Base");}
}
class Potomok: Base
{
 public override void A()
 {
  Console.WriteLine("Реализация в классе Potomok");
  base.A(); //Вызывает метод класса Base
 }
 }
namespace интерф_невиртуал
{
 class Program
 {
  static void Main()
  {
   Potomok d = new Potomok();
   d.A(); // вызывается Potomok.A();
```


Репозиторий ВГУ

```
    IBase id = d;
    id.A(); // вызывается Base.A();
   }
  }
}
```

Результат работы прграммы:
Реализация в классе Potomok
Реализация в классе Base
Реализация в классе Potomok
Реализация в классе Base

Метод интерфейса, реализованный явным указанием имени, объявлять виртуальным запрещается. При необходимости переопределить в потомках его поведение пользуются следующим приемом: из этого метода вызывается другой, защищенный метод, который объявляется виртуальным.

Пример 9

```
interface IBase
{
  void A();
}
class Base : IBase
{
  void IBase.A() { A_( ) ; }
  protected virtual void A_() { . . . }
}
class Potomok: Base
{
  protected override void A_() { . . . }
  . . .
}
```

В приведенном далее примере метод A() интерфейса IBase реализуется посредством защищенного виртуального метода A_(), который можно переопределять в потомках класса Base. Существует возможность повторно реализовать интерфейс, указав его имя в списке предков класса наряду с классом-предком, уже реализовавшим этот интерфейс. При этом в производном классе реализация переопределенных методов базового класса во внимание не принимается. Например:

```
interface IBase
{
  void A();
}
class Base : IBase
```


Репозиторий ВГУ

```
{
  void IBase.A() { . . . } //не используется в Potomok
}
class Potomok : Base, IBase
{
  public void A() { ... } //реализация метода A() в классе
                                                    Potomok
}
```

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

8.7. Стандартные интерфейсы среды .NET Framework

В библиотеке классов **.NET** определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Стандартные интерфейсы (табл.6.1 приложения 6) поддерживаются многими стандартными классами библиотеки. Так, например, интерфейс ***System.Collections.ICollection*** (табл 6.2 приложения 6) определяет функциональность, общую для всех коллекций, а работа с массивами с помощью цикла **foreach** возможна именно потому, что тип **Array** реализует интерфейсы ***IEnumerable*** и ***IEnumerator***.

Можно создавать и собственные классы, поддерживающие стандартные интерфейсы, что позволит использовать объекты этих классов стандартными способами. При работе с коллекциями часто используется интерфейс ***IComparable***, который задает метод сравнения объектов на больше-меньше, что позволяет выполнять их сортировку.

8.7.1 Интерфейсы IFormatProviderb и IFormattable

Интерфейс ***IFormatProvider*** определяет один метод ***GetFormat*()**, который возвращает объект, управляющий форматированием данных строки, удобной для восприятия человеком. Общий формат метода ***GetFormat*()**:

```
object GetFormat(Type fmt)
```

Здесь параметр *fmt* задает формат объекта.

Интерфейс ***IFormattable*** поддерживает форматирование выводимого результата в удобной для восприятия человеком форме. В интерфейсе ***IFormattable*** определен следующий метод:

```
string ToString(string fmt, IFormatProvider fmtpvdr)
```

Здесь параметр *fmt* задает инструкции форматирования, а параметр *fmtpvdr* – источник формата.

Репозиторий ВГУ

8.7.2 Интерфейс IComparable

Во многих классах необходимо реализовать интерфейс ***IComparable***, поскольку он позволяет 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.

Рассмотрим пример реализации интерфейса ***IComparable***.

Пример 8.Создать класс, содержащий информацию о людях (фамилия, возраст, вес).Вывести на экран информацию о людях, отсортированную по возрасту.

```
      // Реализация интерфейса IComparable.
using System;
using System.Collections;
using System.IO;
using System.Text;
namespace MyProgram
{
 class one : IComparable
 {
  public string f;     //имя
  public int age;      //возраст
  public float massa;   //вес
      //перегружаем метод ToString
  public override string ToString()
   {
    return String.Format("{0,-16}возраст:{1,-8}Вес:
    {2}",f, age, massa);
   }
      // Реализуем интерфейс IComparable .
  public int CompareTo(object obj)
   {
    one b;
    b = (one)obj;
    return age.CompareTo(b.age);  //реализуем метод для
                                  //сравнения возраста
   }
 }
```

Репозиторий ВГУ

```
class Program
{
 static void Main(string[] args)
 {
  ArrayList people = new ArrayList();
  int k = 0;
  Console.WriteLine("Введите начальный список ");
  while ( k<3)          //цикл для организации списка
  {
   one a = new one();
   a.f = Console.ReadLine();
   a.age = int.Parse(Console.ReadLine());
   a.massa = float.Parse(Console.ReadLine());
   people.Add(a); //метод класса ArrayList добавляет объект в
                                          //коллекцию
   k++;
  }
  Console.WriteLine("Информация о людях до
  сортировки:");
  foreach (one i in people)
  { Console.WriteLine(" " + i); }
  Console.WriteLine();
  people.Sort();           // Сртируем список
  Console.WriteLine("Информация о людях после
  сортировки:");
  foreach (one i in people)
  { Console.WriteLine(" " + i); }
 }
}
}
```

Результаты выполнения программы:

Информация о людях до сортировки:
Иван возраст: 23 Вес: 67
Петр возраст: 19 Вес: 60
Николай возраст: 21 Вес: 75

Информация о людях после сортировки:
Петр возраст: 19 Вес: 60
Николай возраст: 21 Вес: 75
Иван возраст: 23 Вес: 67

Репозиторий ВГУ

8.7.3 Интерфейс IComparer

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.

Пример 9. Реализация интерфейса ***IComparer*** для предыдущего примера.

```
using System;
using System.Collections;
using System.IO;
using System.Text;
namespace MyProgram
{
 class one
 {
  public string f;     //имя
  public int age;      //возраст
  public float massa;   //вес
  public static void Print(string s, ArrayList a)
  {
   Console.WriteLine(s);
   foreach (one x in a)
   Console.WriteLine(x.f+"\t"+x.age+"\t"+x.massa);
  }
 }
 class SortByAge:IComparer //реализация стандартного
                                    // интерфейса
 {    //переопределение метода Compare
  int IComparer.Compare(object x, object y)
  {
   one t1 = (one)x;
   one t2 = (one)y;
   if (t1.age > t2.age) return 1;
   if (t1.age < t2.age) return -1;
   return 0;
  }
 }
 class Program
 {
  static void Main(string[] args)
```


Репозиторий ВГУ

```
{
 ArrayList people = new ArrayList();
 int k = 0;
 while ( k<3) //цикл для организации списка
 {
  one a = new one();
  a.f = Console.ReadLine();
  a.age = int.Parse(Console.ReadLine());
  a.massa = float.Parse(Console.ReadLine());
  people.Add(a);
  k++;
 }
 one.Print("Исходные данные: ", people);
 people.Sort(new SortByAge()); //вызов сортировки

 one.Print("Отсортированные данные: ",people);
}
}
}
```

Результаты выполнения программы:
Информация о людях до сортировки:
Иван возраст: 23 Вес: 67
Петр возраст: 19 Вес: 60
Николай возраст: 21 Вес: 75

Информация о людях после сортировки:
Петр возраст: 19 Вес: 60
Николай возраст: 21 Вес: 75
Иван возраст: 23 Вес: 67

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 коллек-

Репозиторий ВГУ

циями?

9. Чем отличаются интерфейсы ***IComparable*** и **IComparer?**

Лабораторная работа № 8 Интерфейсы.

Задание 1. Интерфейсы `Ix, Iy, Iz,` содержат объявления методов с одной и той же сигнатурой следующим образом

```
interface Ix
{
 void IxF0(параметр);
 void IxF1();
}
interface Iy
{
 void F0(параметр);
 void F1();
}
interface Iz
{
 void F0(параметр);
 void F1();
}
```

Эти интерфейсы наследуются в классе `TestClass`, содержащий член `w` типа `параметр` и реализуются так, как задано в варианте. В каждом методе задать вывод результата.

Рассмотреть случай

- неявной реализации интерфейсов
- явной реализации интерфейса Iz

В программе должна выполняться:

- неявная неоднозначная реализация методов интерфейсов Iy и Iz,
- вызов функций с явным приведением к типу интерфейса,
- вызов метода для объекта посредством интерфейсной ссылки.

Варианты заданий:

№	параметр	`IxF0, IxF1` возвращают	`F0 F1` возвращают	
			Неявная реализация	Явная реализация `Iz`
1	**double**	w^2	$\sqrt{w}$	w^2+5
2	**double**	`Cos(w)`	`Exp(w)`	`1/Exp(w)`
3	**int**	`w+5`	w^3	`7w-2`
4	**char**	Символ, преобразованный в нижний ре-	*, если символ буква,	цифру '5', если символ буква

Репозиторий ВГУ

		гистр		
5	**string**	Строку, удалив два последних символа	Строку, удалив два первых символа	Строку, заменив первый символ символом -
6	**int**	7w-4	w*3	6+w
7	**string**	подстроку, начиная с 3-ей позиции	Удвоенную строку	Удвоенную строку с удаленным последним символом
8	**double**	Abs(w)	Sin(w)	w+2
9	**double**	Log(w)	2/w	w^3
10	**int**	w^2- w	15/w	2w-3
11	**int**	10*w	w-10	w/10
12	**char**	Строку из5 символов	Предыдущий символ, если он отображаемый, в противном случае #.	Следующий символ, если он отображаемый, в противном случае #.

Задание 2. Выполнить задания, используя для хранения экземпляров разработанных классов стандартные параметризованные коллекции. Во всех классах реализовать интерфейсы IComparable и IComparer перегрузить операции отношения для реализации сравнения объектов по указанному полю. Результат вывести на экран.

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, 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 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.
4. Составить инвентарную ведомость склада, включив следующие данные: вид продукции, стоимость, сорт, количество. Вывести в новый

Репозиторий ВГУ

список информацию о той продукции, количество которой менее заданной величины, отсортировав ее по количеству продукции на складе.

5. Составить список вкладчиков банка, включив следующие данные: ФИО, № счета, сумма, год открытия счета. Вывести в новый список информацию о тех вкладчиках, которые открыли вклад в текущем году, отсортировав их по сумме вклада.
6. 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.
7. Составить список студентов группы, включив следующие данные: ФИО, год рождения, какую школу окончил. Вывести в новый список информацию о студентах, окончивших заданную школу, отсортировав их по году рождения.

   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.
8. 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.
9. 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.
10. 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.
11. Составить список сотрудников учреждения, включив следующие данные: ФИО, должность, зарплата, рабочий стаж. Вывести в новый список информацию о сотрудниках, имеющих зарплату ниже определенного уровня, отсортировав их по рабочему стажу.
12. Составить расписание вылетов аэропорта, содержащее следующую информацию: название пункта назначения рейса; номер рейса; тип самолета. Вывести в новый список информацию о пунктах назначения и номерах рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры, отсортировав их по названиям пунктов (если таких рейсов нет, вывести соответствующее сообщение).

Репозиторий ВГУ

ЛИТЕРАТУРА

1. Бен Ватсон С# 4.0 на примерах – СПб.: БХВ-Петербург, 2011, 608с.
2. Биллиг, В. А. Основы программирования на С#. – М.: Изд-во «Интернет-университет информационных технологий – ИНТУИТ.ру», 2006. – 488 с.
3. Брукс, Ф. Мифический человеко-месяц, или как создаются программные комплексы. – М.: Символ-Плюс, 2000. – 304 с.
4. Ватсон, К. С#. – М.: Лори, 2004. – 880 с.
5. Вирт, Н. Алгоритмы и структуры данных. – СПб: Невский диалект, 2001. – 352 с.
6. Гиббонз, П. Платформа .NET для Java-программистов. – СПб.: Питер, 2003. – 336 с.
7. Гуннерсон, Э. Введение в С#. Библиотека программиста. – СПб.: Питер, 2001. – 304 с.
8. Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер. С# 4.0 и платформа .NET 4 для профессионалов – М.: Диалектика, Вильямс, 2011 – 1440с.
9. Либерти, Д. Программирование на С#. – СПб.: Символ-Плюс, 2003. – 688 с.
10. Майо, Д. С#. Искусство программирования: Энциклопедия программиста. – Киев: ДиаСофт, 2002. – 656 с.
11. Майо, Дж. С# Builder. Быстрый старт. – М.: Бином, 2005. – 384 с.
12. Мартынов, Н.Н. С# для начинающих – М.: КУДИЦ-ПРЕСС, 2007, – 272 с.
13. Микелсен, К. Язык программирования С#. Лекции и упражнения: Учебник. – Киев: ДиаСофт, 2002. – 656 с.
14. Оньон, Ф. Основы ASP.NET с примерами на С#. – М.: Издательский дом «Вильямс», 2003. – 304 с.
15. Павловская, Т. А. С#. Программирование на языке высокого уровня. Учебник для вузов – СПб.: Питер, 2007. – 432 с.
16. Уотсон К., Нейгел К.Visual С# 2010. Полный курс – М.: Вильямс, 2011 – 955 с.
17. Чарльз Петцольд Программирование в тональности С#. – М.: Русская Редакция, 2004 – 512 с.

Репозиторий ВГУ

18. Эндрю Троелсен Язык программирования C# 2010 и платформа .NET 4 – М.: Вильямс, 2011 – 1392 с.
19. Мэтью Мак-Дональд Windows Presentation Foundation в .NET 4.0 с примерами на C# 2010 – М.: Вильямс, 2011 – 1020с.
20. Н. Культин Microsoft Visual C# в задачах и примерах – СПб.: БХВ-Петербург, 2009 – 314 с.
21. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C# : – СПб.: Питер, 2007. – 636 с.

Репозиторий ВГУ

ПРИЛОЖЕНИЯ

Приложение 1. Создание консольного приложения.

Основные окна среды.

Для создания проекта следует после запуска ***Visual Studio.NET*** в главном меню выбрать команду ***File* ► *New Project***.... В левой части открывшегося диалогового окна нужно выбрать пункт Visual C# Projects, в правой – пункт Console Application. В поле Name можно ввести имя проекта, а в поле Location – место его сохранения на диске. После щелчка на кнопке ***OK*** среда создаст решение и проект с указанным именем. Примерный вид экрана приведен на рис. 1.1

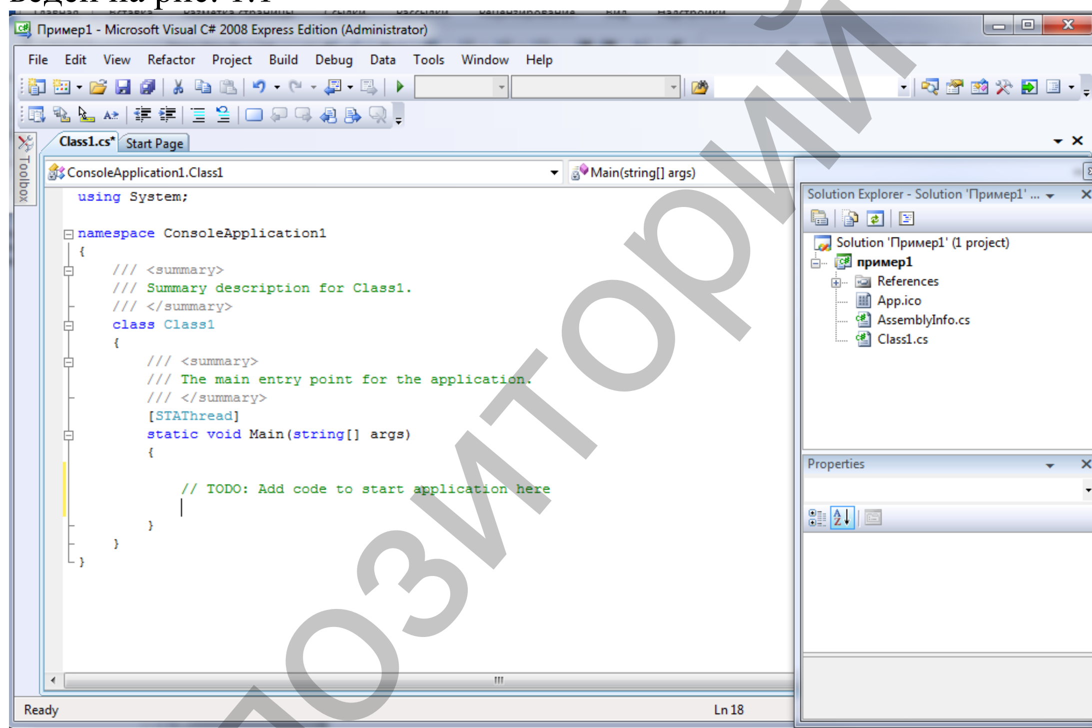

Рис. 1.1. Примерный вид экрана после создания проекта консольного приложения

В верхней части экрана располагается *главное меню* (с разделами File, Edit, View и т. д.) и *панели инструментов* (toolbars). Панелей инструментов в среде достаточно много, их можно включить с помощью пункта меню View ► Toolbars... В верхней правой части экрана располагается *окно управления проектом* Solution Explorer (если оно не отображается, следует воспользоваться командой View ► Solution Explorer главного меню). В окне перечислены все ресурсы, входящие в проект: ссылки на библиотеку (System,

Репозиторий ВГУ

System.Data, System.XML и т. д.), файл ярлыка (App.ico), файл с исходным текстом класса (**Class**1.cs) и информация о сборке (Assemblylnfo.cs).

С помощью проводника Windows можно увидеть, что на заданном диске появилась папка с указанным именем, содержащая несколько других файлов и вложенных папок. Среди них – файл проекта (с расширением csproj), файл решения (с расширением sln) и файл с кодом класса (**Class**1.cs).

В нижней правой части экрана расположено *окно свойств* Properties. В окне свойств отображаются важнейшие характеристики выделенного элемента. Например, чтобы изменить имя файла, в котором хранится класс **Class**l, надо выделить этот файл в окне управления проектом и задать в окне свойств новое значение свойства FileName (ввод заканчивается нажатием клавиши Enter).

Основное пространство экрана занимает *окно редактора,* в котором располагается текст программы, созданный средой автоматически. Ключевые (зарезервированные) слова отображаются синим цветом, комментарии различных типов – серым и темно-зеленым, остальной текст – черным.

Заготовка консольной программы

Рассмотрим каждую строку заготовки программы (Рис. 1.1).

Директива ***using System*** подключает соответствующее пространство имен и разрешает использовать имена стандартных классов этого пространства непосредственно (без указания имени пространства).

Ключевое слово ***namespace*** создает для проекта собственное пространство имен, названное по умолчанию ***ConsoleApplicationl***. Это сделано для того, чтобы и было давать программным объектам имена, не заботясь о том, что они могут совпасть с именами в других пространствах имен.

Строки, начинающиеся с двух или трех косых черт, являются комментариями.

В нашей заготовке программы всего один класс, которому по умолчанию присвоино имя ***Classl***. *Описание класса* начинается с ключевого слова ***class***, за которым следуют его имя и далее в фигурных скобках список элементов класса (его данных и функций, называемых также методами).

Фигурные скобки ограничивают *блок,* внутри которого могут располагаться другие блоки, вложенные в него.

В данном случае внутри класса только один элемент – метод ***Main***. Каждое приложение должно содержать метод ***Main*** – с него начинается выполнение программы.

В методе ***Main*** записывается код, выполняемый при запуске приложения.

Репозиторий ВГУ

Запустить программу можно нажав клавишу ***F5*** или выбрать в меню команду ***Debug* ► *Start***. Результат работы программы будет выведен в консольном окне, консольном окне, которое незамедлительно закроется. Чтобы консольное окно не закрылось сразу, следует воспользоваться клавишами ***Ctrl+F5*** или выбрать в меню команду ***Debug* ► *Start Without Debugging***.

Если в тексте программы обнаружены *синтаксические ошибки,* об этом выводятся сообщения в окне, расположенном в нижней части экрана.

Двойной щелчок на строке с сообщением об ошибке подсвечивает неверную строку в программе. Для получения дополнительных пояснений можно нажать клавишу F1, при этом в окне редактора появится новая вкладка с соответствующей страницей из справочной системы.

Репозиторий ВГУ

Приложение 2. Встроенные типы в C#.

Встроенные типы не требуют предварительного определения. Они однозначно соответствуют стандартным классам библиотеки .NET, определенным в пространстве имен System.

Целые типы, а также символьный, вещественные и финансовый типы можно объединить под названием *арифметических типов.*

Таблица 2.1. Встроенные типы C#

Название	**Ключевое слово**	**Тип .NET**	**Диапазон значений**	**Описание**	**Размер битов**
Логический тип	bool	Boolean	true, false		8
Целые типы	sbyte	SByte	От -128 до 127	Со знаком	8
	byte	Byte	От 0 до 255	Без знака	8
	short	Intl6	От -32768 до 32767	Со знаком	16
	ushort	UIntl6	От 0 до 65535	Без знака	16
	int	Int32	От $-2\cdot10^9$ до $2\cdot10^9$	Со знаком	32
	uint	UInt32	От 0 до $4\cdot10^9$	Без знака	32
	long	Int64	От $-9\cdot10^{18}$ до $9\cdot10^{18}$	Со знаком	64
	ulong	UInt64	От 0 до 18·1018	Без знака	64
Символьный тип	char	Char	От U+0000 до U+ffff	Unicode-символ	16
Вещественные	float	Single	От 1.5·10-45 до 3.4·1038	7 цифр	32
	double	Double	От 5.0·10-324 до 1.7·10308	15-16 цифр	64
Финансовый тип	Decimal	Decimal	От 1.0·10-28 до 7.9·1028	28-29 цифр	128
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	

Репозиторий ВГУ

Тип object	object	Object	Можно хранить все что угодно	Всеобщий предок	

Тип **decimal** предназначен для денежных вычислений, в которых критичны ошибки округления. Величины типа **decimal** позволяют хранить 28-29 десятичных разрядов.

Тип **decimal** не относится к вещественным типам, у них различное внутреннее представление. Величины денежного типа даже нельзя использовать в одном выражении с вещественными без явного преобразования типа. Использование величин финансового типа в одном выражении с целыми допускается.

Любой встроенный тип C# соответствует стандартному классу библиотеки .NET, определенному в пространстве имен **System**. Везде, где используется имя встроенного типа, его можно заменить именем класса библиотеки. Это значит, что у встроенных типов данных C# есть методы и поля. С их помощью можно, например, получить минимальные и максимальные значения для целых, символьных, финансовых и вещественных чисел:

Репозиторий ВГУ

Приложение 3. Операции и отношения.

Таблица 3.1 Арифметические операции

Операция	Назначение	Пример
+	*Сложение*	X = x + y;
–	*Вычитание*	X = x – y;
*	*Умножение*	Z = x * y;
/	*Деление*	Z = x / y;
%	*Деление по модулю*	Z = t % e;

Поразрядные операции

В C# имеются операции, пригодные для обработки отдельных разрядов памяти (например, в видеопамяти графического дисплея). Такие операции называются поразрядными (операции с битами). Они позволяют изменять, считывать и сдвигать разряды в переменных. При этом переменная рассматривается не как число, а как комбинация двоичных разрядов, т.е. как логический код. Операция выполняется отдельно над каждым разрядом.

Таблица 3.2 ***Поразрядные операции***

Операция	Назначение	Пример
&	*Поразрядное И*	i & 16
		Поразрядное ИЛИ	i	12
^	*Поразрядное Исключающее ИЛИ*	k ^ 10
~	*Поразрядное НЕ*	~a
<<	*Поразрядный сдвиг влево*	<< 2
>>	*Поразрядный сдвиг вправо*	>> 2

Таблица 3.3 ***Логические операции***

Операция	Назначение	Пример
&&	*Логическое И*	(R>2) && (R<20)
||	*Логическое ИЛИ*	(L= =12) || (L>15)
!	*Логическое НЕ*	Tab != 13

Репозиторий ВГУ

Таблица 3.4 ***Операции отношения***

Операция	Назначение	Пример
= =	*Равно*	I = = 0
! =	*Не равно*	K!=15
>	*Больше, чем*	Z>15.2
<	*Меньше, чем*	Tab< -132.654
>=	*Больше или равно*	Z11>=0
<=	*Меньше или равно*	Y2<=10

Приоритет операций

Порядок вычисления выражения определяется старшинством (приоритетом) содержащихся в нем операций. В языке Паскаль принят следующий приоритет операций:

Таблица 3.5 ***Приоритет операций***

<table>
<tr><td>Приоритет</td><td>Операции</td></tr>
<tr><td>1</td><td>() [] . (постфикс)++ (постфикс)-- new sizeof typeof unchecked</td></tr>
<tr><td>2</td><td>! ~ (имя типа) +(унарный) -(унарный) ++(префикс) --(префикс)</td></tr>
<tr><td>3</td><td>* / %</td></tr>
<tr><td>4</td><td>+ -</td></tr>
<tr><td>5</td><td><< >></td></tr>
<tr><td>6</td><td>< > <= => is</td></tr>
<tr><td>7</td><td>== !=</td></tr>
<tr><td>8</td><td>&</td></tr>
<tr><td>9</td><td>^</td></tr>
<tr><td>10</td><td>|</td></tr>
<tr><td>11</td><td>&&</td></tr>
<tr><td>12</td><td>||</td></tr>
<tr><td>13</td><td>?:</td></tr>
<tr><td>14</td><td>= += -= *= /= %= &= |= ^= <<= >>=</td></tr>
</table>

Репозиторий ВГУ

Порядок выполнения операций переопределить можно с помощью скобок.Приложение 4. Поля и методы основных классов пространства имен System.

Репозиторий ВГУ

Приложение 4. Основные методы пространства имен System.

Таблица 4.1. Основные поля и статические методы класса Math

Имя	**Описание**	**Результат**	**Пояснения**
Abs	Модуль	Перегружен	записывается как Abs(x)
Acos	Арккосинус[2]	double	Acos(double x)
Asin	Арксинус	double	Asin(double x)
Atari	Арктангенс	double	Atan(double x)
Atan2	Арктангенс	double	Atan2(double x, double y) – угол, тангенс которого есть результат деления y на x
BigMul	Произведение	long	BigMul(int x, int y)
Ceiling	Округление до большего целого	double	Ceiling(double x)
Cos	Косинус	double	Cos(double x)
Cosh	Гиперболический косинус	double	Cosh(double x)
DivRem	Деление и остаток	Перегружен	DivRem(x, y, rem)
E	База натурального логарифма (число *e)*	double	2,71828182845905
Exp	Экспонента	double	e^x записывается как Exp (x)
Floor	Округление до меньшего целого	double	Floor(double x)
IEEERemainder	Остаток от деления	double	IEEERemainder(double x, double y)
Log	Натуральный логарифм	double	$\log_e x$ записывается как Log(x)
Logl0	Десятичный логарифм	double	$\log_{10} x$ записывается как Logl0(x)
Max	Максимум из двух чисел	Перегружен	Max(x, y)
Min	Минимум из двух чисел	Перегружен	Min(x, y)
PI	Значение числа p	double	3,14159265358979
Pow	Возведение в	double	$\mathbf{x^y}$ записывается как Pow(x,

Репозиторий ВГУ

	степень		y)
Round	Округление	Перегружен	Округление до ближайшего целого.
Sign	Знак числа	int	Аргументы перегружены
Sin	Синус	double	Sin(double x)
Sinn	Гиперболический синус	double	Sinh(double x)
Sqrt	Квадратный корень	double	√x записывается как Sqrt(x)
Tan	Тангенс	double	Tan(double x)
Tanh	Гиперболический тангенс	double	Tanh(double x)

Таблица 4.2. Основные элементы класса Array

Элемент	**Вид**	**Описание**
Length	Свойство	Количество элементов массива (по всем размерностям)
Rank	Свойство	Количество размерностей массива
BinarySearch	Статический метод	Двоичный поиск в отсортированном массиве
Clear	Статический метод	Присваивание элементам массива значений по умолчанию
Copy	Статический метод	Копирование заданного диапазона элементов одного массива в другой массив
CopyTo	Метод	Копирование всех элементов текущего одномерного массива в другой одномерный массив
GetValue	Метод	Получение значения элемента массива
IndexOf	Статический метод	Поиск первого вхождения элемента в одномерный массив
LastlndexOf	Статический метод	Поиск последнего вхождения элемента в одномерный массив
Reverse	Статический метод	Изменение порядка следования элементов на обратный
SetValue	Метод	Установка значения элемента массива
Sort	Статический метод	Упорядочивание элементов одномерного массива

Таблица 4.3. Основные методы класса System.Char

Метод	**Описание**

Репозиторий ВГУ

Метод	Описание
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае
GetUnicodeCategory	Возвращает категорию Unicode-символа
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой
IsLower	Возвращает true, если символ задан в нижнем регистре
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным)
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsUpper	Возвращает true, если символ записан в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки и возврат каретки)
Parse	Преобразует строку в символ (строка должна состоять из одного символа)
ToLower	Преобразует символ в нижний регистр
ToUpper	Преобразует символ в верхний регистр
MaxValue, MinValue	Возвращают символы с максимальным и минимальным кодами (эти символы не имеют видимого представления)

Таблица 4.4. Основные элементы класса System.String

Название	Вид	Описание
Compare	Статический метод	Сравнение двух строк в алфавитном порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т. д.

Репозиторий ВГУ

Название	**Вид**	**Описание**
CompareOrdinal	Статический метод	Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки
CompareTo(string str)	Метод	Сравнение текущего экземпляра строки с другой строкой. Возвращает отрицательное значение, если вызывающая строка меньше строки `str`, положительное значение, если вызывающая строка больше строки `str`, и нуль, если сравниваемые строки равны
Concat	Статический метод	Конкатенация строк. Метод допускает сцепление произвольного числа строк
Copy	Статический метод	Создание копии строки
Empty	Статическое поле	Пустая строка (только для чтения)
Format	Статический метод	Форматирование в соответствии с заданными спецификаторами формата
IndexOf(string str), LastIndexOf, IndexOfAny, LastlndexOfAny	Методы	Определение индексов первого и последнего вхождения заданной подстроки `str` или любого символа из заданного набора
Insert	Метод	Вставка подстроки в заданную позицию
Intern, IsInterned	Статические методы	Возвращает ссылку на строку, если такая уже существует. Если строки нет, `Intern` добавляет строку во внутренний пул, `IsIntern` возвращает `null`
Join	Статический метод	Слияние массива строк в единую строку. Между элементами массива вставляются разделители
Length	Свойство	Длина строки (количество символов)

Репозиторий ВГУ

Название	Вид	Описание
PadLeft, PadRight	Методы	Выравнивание строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки
Remove(№, n)	Метод	Удаление подстроки из `n` символов с заданной позиции №
Replace	Метод	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом
Split	Метод	Разделяет строку на элементы, используя заданные разделители. Результаты помещаются в массив строк
StartsWith, EndsWith	Методы	Возвращает `true` или `false` в зависимости от того, начинается или заканчивается строка заданной подстрокой
Substring	Метод	Выделение подстроки, начиная с заданной позиции
ToCharArray	Метод	Преобразование строки в массив символов
ToLower, ToUpper	Методы	Преобразование символов строки к нижнему или верхнему регистру
Trim, TrimStart, TrimEnd	Методы	Удаление пробелов в начале и конце строки или только с одного ее конца (обратные по отношению к методам `PadLeft` и `PadRight` действия)

Таблица 4.5. Основные элементы класса `System.Text.StringBuilder`

Название	Вид	Описание
`Append`	Метод	Добавление в конец строки. Разные варианты метода позволяют добавлять в строку величины любых встроенных типов, массивы символов, строки и подстроки типа string
`AppendFormat`	Метод	Добавление форматированной строки в конец строки
`Capacity`	Свойство	Получение или установка емкости буфера. Если

Репозиторий ВГУ

Название	**Вид**	**Описание**
		устанавливаемое значение меньше текущей длины строки или больше максимального, генерируется исключение ArgumentOutOfRangeExceptlon
`Insert`	Метод	Вставка подстроки в заданную позицию
`Length`	Свойство	Длина строки (количество символов)
`MaxCapacity`	Свойство	Максимальный размер буфера
`Remove`	Метод	Удаление подстроки из заданной позиции
`Replace`	Метод	Замена всех вхождений заданной подстроки или символа новой подстрокой или символом
`ToString`	Метод	Преобразование в строку типа string

Таблица 4.6. Основные методы класса System.Random

Название	**Описание**
`Next ()`	Возвращает целое положительное число во всем положительном диапазоне типа int
`Next(макс)`	Возвращает целое положительное число в диапазоне [0, макс]
`Next(мин, макс)`	Возвращает целое положительное число в диапазоне [мин, макс]
`NextBytes(массив)`	Возвращает массив чисел в диапазоне [0, 255]
`NextDouble()`	Возвращает вещественное положительное число в диапазоне [0, 1]

Таблица 4.7. Основные методы класса ArrayList

Метод	Описание
public virtual void AddRange(ICollection c)	Добавляет элементы из коллекции c в конец вызывающей коллекции
public virtual int BinarySearch(object v)	В вызывающей отсортированной коллекции выполняет поиск значения, заданного параметром v. Возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.
public virtual int BinarySearch(object v, IComparer comp)	В вызывающей отсортированной коллекции выполняет поиск значения, заданного параметром v, на основе метода сравнения объектов, заданного параметром comp. Возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.
public virtual int Bina-	В вызывающей отсортированной коллекции вы-

Репозиторий ВГУ

rySearch(int startIdx, int count, object v, IComparer comp)	полняет поиск значения, заданного параметром v, на основе метода сравнения объектов, заданного параметром comp. Поиск начинается с элемента, индекс которого равен значению startIdx, и включает count элементов. Метод возвращает индекс найденного элемента. Если искомое значение не обнаружено, возвращает отрицательное значение.
public virtual void CopyTo(Array ar, int startIdx)	Копирует содержимое вызывающей коллекции, начиная с элемента, индекс которого равен значению startIdx, в массив, заданный параметром ar. Приемный массив должен быть одномерным и совместимым по типу с элементами коллекции.
public virtual void CopyTo(int srcIdx, Array ar, int destIdx, int count)	Копирует count элементов вызывающей коллекции, начиная с элемента, индекс которого равен значению srcIdx, в массив, заданный параметром ar, начиная с элемента, индекс которого равен значению destIdx. Приемный массив должен быть одномерным и совместимым по типу с элементами коллекции
public virtual ArrayList GetRange(int idx, int count)	Возвращает часть вызывающей коллекции типа ArrayList. Диапазон возвращаемой коллекции начинается с индекса idx и включает count элементов. Возвращаемый объект ссылается на те же элементы, что и вызывающий объект
public static ArrayList FixedSize(ArrayList ar)	Превращает коллекцию ar в ArrayList-массив с фиксированным размером и возвращает результат
public virtual void InsertRange(int startIdx, ICollection c)	Вставляет элементы коллекции, заданной параметром с, в вызывающую коллекцию, начиная с индекса, заданного параметром startIdx
public virtual int LastIndexOf(object v)	Возвращает индекс последнего вхождения объекта v в вызывающей коллекции. Если искомый объект не обнаружен, возвращает отрицательное значение
public static ArrayList Readonly(ArrayList ar)	Превращает коллекцию ar в ArrayList-массив, предназначенный только для чтения
public virtual void RemoveRange(int idx, int count)	Удаляет count элементов из вызывающей коллекции, начиная с элемента, индекс которого равен значению idx
public virtual void Reverse()	Располагает элементы вызывающей коллекции в обратном порядке
public virtual void Reverse(int startIdx, int	Располагает в обратном порядке count элементов вызывающей коллекции, начиная с индекса startIdx

Репозиторий ВГУ

count)	
public virtual void SetRange(int startIdx, ICollection c)	Заменяет элементы вызывающей коллекции, начиная с индекса startIdx, элементами коллекции, заданной параметром с
public virtual void Sort()	Сортирует коллекцию по возрастанию
public virtual void Sort(IComparer comp)	Сортирует вызывающую коллекцию на основе метода сравнения объектов, заданного параметром comp. Если параметр comp имеет нулевое значение, для каждого объекта используется стандартный метод сравнения
public virtual void Sort (int startidx, int endidx, icomparer comp)	Сортирует часть вызывающей коллекции на основе метода сравнения объектов, заданного параметром comp. Сортировка начинается с индекса startidx и заканчивается индексом endidx. Если параметр comp имеет нулевое значение, для каждого объекта используется стандартный метод сравнения
public virtual object [] ToArray ()	Возвращает массив, который содержит копии элементов вызывающего объекта
public virtual Array ToArray (Type type)	Возвращает массив, который содержит копии элементов вызывающего объекта. Тип элементов в этом массиве задается параметром type
public virtual void TrimToSize()	Устанавливает свойство Capacity равным значению свойства Count

Репозиторий ВГУ

Приложение 5. Специфекаторы.

Таблица 5.1. Спецификаторы класса

	Спецификатор	Описание
1.	**new**	Используется для вложенных классов. Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях объектов.
2.	**public**	Доступ не ограничен
3.	**protected**	Используется для вложенных классов. Доступ только из элементов данного и производных классов
4.	**internal**	Доступ только из данной программы (сборки)
5.	**protected internal**	Доступ только из данного и производных классов или из данной программы (сборки)
6.	**private**	Используется для вложенных классов. Доступ только из элементов класса, внутри которого описан данный класс
7.	**abstract**	Абстрактный класс. Применяется в иерархиях объектов.
8.	**sealed**	Бесплодный класс. Применяется в иерархиях объектов.
9.	**static**	Статический класс. Введен в версию языка 2.0. Рассматривается в разделе «Конструкторы»

Спецификаторы 2-6 называются *спецификаторами доступа.* Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут присутствовать в описании только в вариантах, приведенных в таблице, а также могут комбинироваться с остальными спецификаторами.

Таблица 5.2 Спецификаторы полей и констант класса

1	Спецификатор	Описание
1.	**new**	Новое описание поля, скрывающее унаследованный элемент класса
2.	**public**	Доступ к элементу не ограничен
3.	**protected**	Доступ только из данного и производных классов
4.	**internal**	Доступ только из данной программы (сборки)
5.	**protected in-**	Доступ только из данного и производных классов

Репозиторий ВГУ

	ternal	и из данной сборки
6.	**private**	Доступ только из данного класса
7.	**static**	Одно поле для всех экземпляров класса
8.	**readonly**	Поле доступно только для чтения
9.	**volatile**	Поле может изменяться другим процессом или системой

По умолчанию элементы класса считаются закрытыми (***private***). Для полей класса этот вид доступа является предпочтительным, поскольку поля определяют внутреннее строение класса, которое должно быть скрыто от пользователя. Все методы класса имеют непосредственный доступ к его закрытым полям.

Репозиторий ВГУ

Приложение 6. Интерфейсные коллекции и методы некоторых интерфейсов.

Таблица 6.1 Интерфейсные коллекции

Интерфейс	Описание
IEnumerator	Содержит методы, которые позволяют поэлементно получать содержимое коллекции
IEnumerable	Определяет метод GetEnumerator( ), который поддерживает нумератор для любого класса коллекции
ICollection	Определяет элементы, которые должны иметь все коллекции
IComparer	Определяет метод Compare( ) , который выполняет сравнение объектов, хранимых в коллекции
IList	Определяет коллекцию, к которой можно получить доступ посредством индексатора
IDictionary	Определяет коллекцию (словарь), которая состоит из пар ключ/значение
IDictionaryEnumerator	Определяет нумератор для коллекции, которая реализует интерфейс IDictionary
IHashCodeProvider	Определяет хеш-функцию

Таблица 6.2 Основные методы и свойства ICollection

Элемент интерфейса	Его тип	описание
int Count {get;}	Свойство	Определяет количество элементов коллекции в данный момент. Если Count равно нулю, то коллекция пуста.
void CopyTo (Array target, int startIdx)	Метод	Обеспечивает переход от коллекции к стандартному C#-массиву, копируя содержимое коллекции в массив, заданный параметром target, начиная с индекса, заданного параметром startIdx

Таблица 6.3 Собственные методы интерфейса IList

Элемент интерфейса	Его тип	Описание
int Add(object obj)	Метод	Добавляет объект obj в вызывающую коллекцию. Возвращает индекс, по которому этот объект сохранен
void Clear()	Метод	Удаляет все элементы из вызывающей коллекции
bool	Метод	Возвращает значение true, если вызываю-

Репозиторий ВГУ

Contains(object obj)		щая коллекция содержит объект, переданный в параметре obj, и значение false в противном случае
int IndexOf(object obj)	Метод	Возвращает индекс объекта obj, если он (объект) содержится в вызывающей коллекции. Если объект obj не обнаружен, метод возвращает -1
void Insert(int idx, object obj)	Метод	Вставляет в вызывающую коллекцию объект obj по индексу, заданному параметром idx. Элементы, находившиеся до этого по индексу idx и далее, смещаются вперед, чтобы освободить место для вставляемого объекта obj
void Remove(object obj)	Метод	Удаляет первое вхождение объекта obj из вызывающей коллекции. Элементы, находившиеся до этого за удаленным элементом, смещаются назад, чтобы ликвидировать образовавшуюся "брешь"
void RemoveAt(int idx)	Метод	Удаляет из вызывающей коллекции объект, расположенный по индексу, заданному параметром idx. Элементы, находившиеся до этого за удаленным элементом, смещаются, ликвидируя образовавшуюся "брешь"
bool IsFixedSize { get;}	Свойство	Принимает значение true, если коллекция имеет фиксированный размер. Это означает, что в такую коллекцию нельзя вставлять элементы и удалять их из нее.
bool IsReadOnly { get;}	Свойство	Принимает значение true, если коллекция предназначена только для чтения.
object this[int idx] { get; set; }	Индексатор	Используется для считывания или записи значения элемента с индексом idx. Нельзя применить для добавления в коллекцию нового элемента.

Таблица 6.4 Интерфейс IDictionary наследует интерфейс ICollection.

Элемент интерфейса	Его тип	Описание
void Add (object к, object v)	метод	Добавляет в вызывающую коллекцию пару ключ/значение, заданную параметрами к и v. Ключ к не должен быть нулевым. При попытке задать нулевой ключ генерируют исключение типа NotSupportedException Если окажется, что

Репозиторий ВГУ

		ключ к уже хранится в коллекции, генерируется исключение типа ArgumentException
void Clear ()	метод	Удаляет все пары ключ/значение из вызывающей коллекции
bool Contains (object k)	метод	Возвращает значение true , если вызывающая коллекция содержит объект k в качестве ключа. В противном случае возвращает значение false
IDictionaryEnumerator GetEnumerator()	метод	Возвращает нумератор для вызывающей коллекции
void Remove (object к)	метод	Удаляет элемент, ключ которого равен значению к
bool isFixedSize {get}	свойство	Равно значению true , если коллекция имеет фиксированный размер
bool isReadOnly {get}	свойство	Равно значению true, если коллекция предназначена только для чтения
ICollection Keys {get}	свойство	Получает коллекцию ключей
icollection Values {get}	свойство	Получает коллекцию значений
object this[object key] { get; set; }	индексатор	Этот индексатор можно использовать для получения или установки значения элемента, а также для добавления в коллекцию нового элемента. "Индекс" в данном случае является ключом элемента.

Репозиторий ВГУ

Предметный указатель

Индексация строк 52
Convert 8
Icomparable 90
Icomparer 92
Iformatprovider 90
Iformattable 90
New 25
Parse 8
Read() 8
Readline() 8
Stringbuilder 54
Write 6
Writeline 6
Абстрактный класс 70
Абстрактный метод 71
Аксессор 31
Базовые интерфейсы 85
Базовый класс 57
Бинарные операции 49
Виртуальные методы 68
Двумерный массив 39
Закрытая реализация 82
Защищенный член 59
Иерархия классов 57
Индексатор 41
Инициализация массива 37
Инструкция **switch** 14
Интерфейс 70, 71
Интерфейс 75
Интерфейсная ссылка 79
Интерфейс-потомок 85
Итерация 18
Класс 25
Код доступа 30, 42
Константы 26
Конструктор 29
Конструктор базового класса 60
Конструктор по умолчанию 29
Массив 36
Метод 18
Многомерный массив 37
Множественное наследование 83
Наследование 57
Наследование интерфейсов 85
Неявная реализация 80
Объект 25
Одномерный массив 36
Оператор цикла с параметром 17
Оператор цикла с постусловием 17
Оператор цикла с предусловием 16
Операторы if 14
Операторы выбора 14
Операторы цикла 16
Операции класса 47
Параметр 27
Параметр-массив 28
Перегрузка операций 47
Полиморфизм 68
Поля 26
Производный класс 60
Рваные массивы 39
Реализация интерфейса 76
Свойство 30
Связывание 68
Синтаксис интерфейса 75
Система сбора мусора 32
Создание строк 51
Сокрытие имен 63
Спецификаторы 19, 25
Ссылка на интерфейс 87
Стандартные интерфейсы 90
Статические методы 19
Строки 51
Тело интерфейса 75
Типы параметров 27
Унарные операции 48
Управляющие операторы 14
Цикл **do-while** 17
Цикл **for** 18
Цикл **foreach** 40
Цикла **while** 16
Экземпляр класса 25
Явная реализация интерфейса 82

Репозиторий ВГУ