Белорусский национальный технический университет
Факультет Международный институт дистанционного образования
Кафедра «Информационные системы и технологии»

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«Объектно-ориентированные технологии программирования и стандарты проектирования»

для специальности:
I - 40 01 01 «Программное обеспечение информационных технологий»

Составитель:
Радкевич Андрей Сергеевич, старший преподаватель каф. ИСиТ

Минск БНТУ 2023

Перечень материалов

Электронный учебно-методический комплекс включает:

- теоретический раздел (конспект лекций),
- практический раздел (лабораторные занятия),
- контроль знаний (задания контрольной работе),
- вспомогательный раздел (программа дисциплины, литература, список вопросов)

Пояснительная записка

Электронный учебно-методический комплекс разработан для студентов специальности I - 1-40 01 01 «Программное обеспечение информационных технологий». Информационное наполнение ЭУМК соответствует программе дисциплины «Объектно-ориентированные технологии программирования и стандарты проектирования».

ЭУМК может использоваться как при проведении занятий по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования», так и для организации самостоятельной работы студентов. Внедрение ЭУМК будет способствовать более эффективному овладению методами, инструментальными средствами и системами разработки объектно-ориентированных программ.

Информация в ЭУМК хорошо структурирована. Теоретический раздел включает основные темы курса. Лабораторный практикум содержит необходимый базовый методический материал (цель лабораторной работы, краткие теоретические сведения, задания на лабораторную работу, контрольные вопросы). В ЭУМК приводится также список литературы и актуальная программа дисциплины. Модуль контроля знаний состоит из экзаменационных вопросов.

ЭУМК разработан в виде pdf-файла и не требует установки специального программного обеспечения

СОДЕРЖАНИЕ

РАЗДЕЛ 1. ТЕОРЕТИЧЕСКИЙ

1. Язык программирования C# и платформа .NET

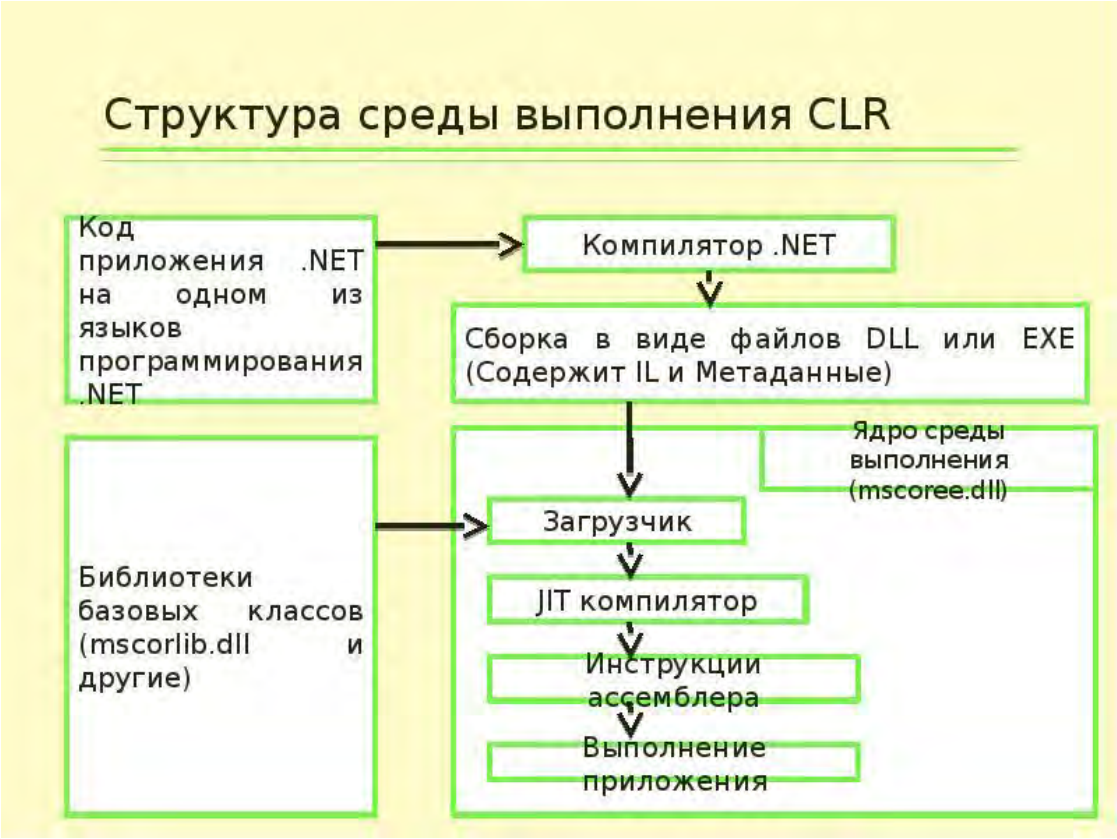
1.1. Состав .NET Framework. Структура среды выполнения CLR

NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду. Состоит из **2x** частей:

- **FCL**
- **CLR**

Base Class Library, или так называемая **.NET FCL** (англ. Framework Class Library), сокращённо **BCL** — стандартная библиотека классов платформы «.NET Framework». Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL — создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и т. д.

Common Language Runtime (англ. **CLR** — общеязыковая исполняющая среда) — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования (C#, Managed C++, Visual Basic .NET, F# и прочие). CLR является одним из основных компонентов пакета Microsoft .NET Framework. Среда CLR является реализацией спецификации CLI (англ. Common Language Infrastructure), спецификации общеязыковой инфраструктуры компании Microsoft.



1.2. Структура управляемого модуля. Понятие и исполнение сборки. CIL

Portable Executable (PE, «переносимый исполняемый») — формат исполняемых файлов, объектного код и динамических библиотек, используемый в 32- и 64-разрядных версиях операционной системы Microsoft Windows. Формат PE представляет собой структуру данных, содержащую всю информацию, необходимую PE-загрузчику для отображения файла в память. Исполняемый код включает в себя ссылки для связывания динамически загружаемых библиотек, таблицы экспорта и импорта API функций, данные для управления ресурсами и данные локальной памяти потока (TLS). В операционных системах семейства Windows NT формат PE используется для EXE, DLL, SYS (драйверов устройств) и других типов исполняемых файлов.

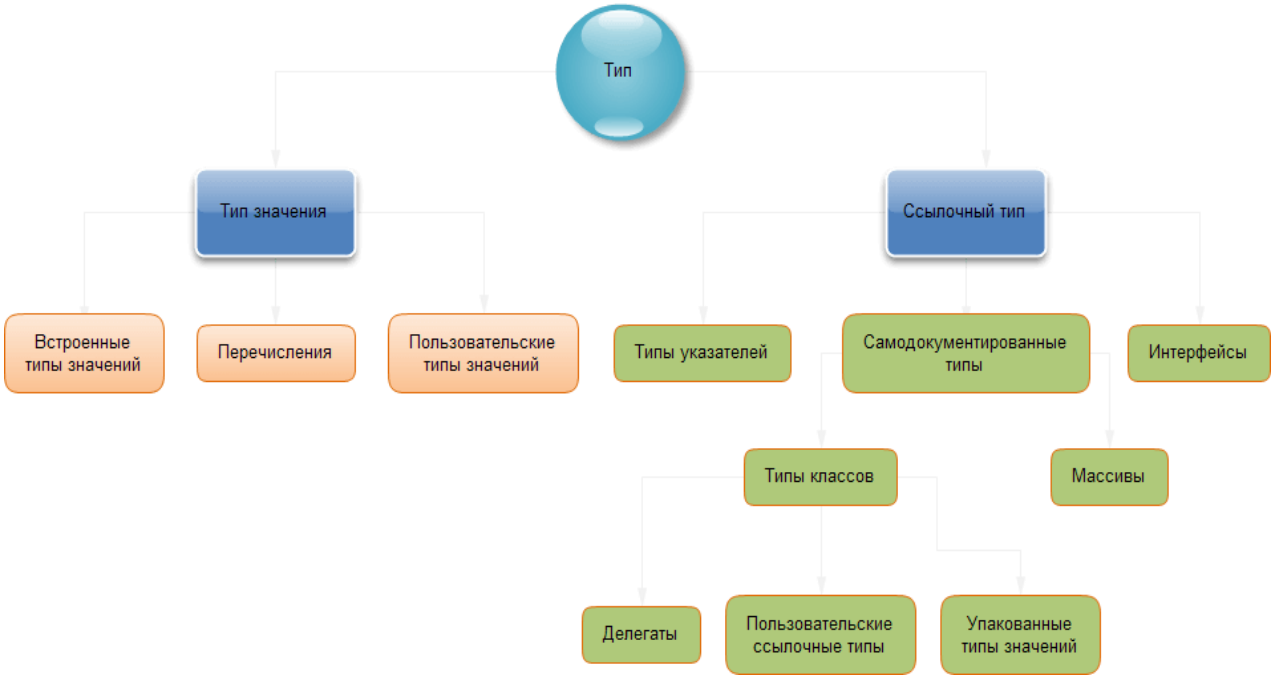
Секция данных **CLR** содержит два важных сегмента: сегмент метаданных и сегмент кода промежуточного языка (IL): Метаданные содержат информацию, относящуюся к сборке, включая манифест сборки. Манифест подробно описывает сборку, включая уникальный идентификатор (с помощью хэша, номера версии, и т. д.), данные об экспортируемых компонентах, расширенную информацию о типе (поддерживаемую общей системой типов (Common Type System — CTS)), внешние ссылки, и список файлов в сборке. Среда CLR широко использует метаданные.

Код промежуточного языка (Intermediate Language — IL) — абстрактный, независимый от языка код, который удовлетворяет требованиям общего промежуточного языка (Common Intermediate Language — CIL) .NET CLR. Термин «промежуточный» относится к природе кода IL, обладающего межъязыковой и кроссплатформенной совместимостью. Этот промежуточный язык, подобный байт-коду Java, позволяет платформам и языкам поддерживать общую среду .NET CLR. IL поддерживает объектно-ориентированное программирование (полиморфизм, наследование, абстрактные типы, и т. д.), исключения, события, и различные структуры данных.

Сборка – это абстрактное понятие, для логической группировки одного или нескольких управляемых модулей или файлов ресурсов, дискретная единица многократно используемого кода внутри CLR.

1.3. CTS (Common Type System). Типы данных C#. Ссылочные и типы значений

CTS (общая система типов) представляет собой формальную спецификацию, в которой описано то, как должны быть определены типы для того, чтобы они могли обслуживаться в CLR-среде. Внутренние детали CTS обычно интересуют только тех, кто занимается разработкой инструментов и/или компиляторов для платформы .NET. Т.е. CTS описывает не просто примитивные типы данных, а целую развитую иерархию типов, включающую хорошо определенные точки, в которых код может определять свои собственные типы. Иерархическая структура общей системы типов (CTS) отражает объектно-ориентированную методологию одиночного наследования IL и показана на следующей схеме:



2. Главные конструкции программирования на C#

2.1. Понятие упаковки и распаковки типов. Типы Nullable

Ссылочные типы (object, dynamic, string, class, interface, delegate) хранятся в управляемой куче, типы значений (struct, enum; bool, byte, char, int, float, double) — в стеке приложения (кроме случая, когда тип значения является полем класса). Преобразование типа значений к ссылочному типу сопровождается неявной операцией упаковки (boxing) — помещение копии типа значений в класс-обёртку, экземпляр которого сохраняется в куче. Упаковочный тип генерируется CLR и реализует интерфейсы сохраняемого типа значения. Преобразование ссылочного типа к типу значений вызывает операцию распаковки (unboxing) — извлечение из упаковки копии типа значения и помещение её в стек.

Преобразование типов Nullable:

- явное
- неявное
- неявные расширяющие преобразования
- явные сужающие преобразования

Null-объединение

Оператор **??** называется оператором **null-объединения**. Он применяется для установки значений по умолчанию для типов значений и ссылочных типов, которые допускают значение null. Оператор ?? возвращает левый операнд, если этот операнд не равен null. Иначе возвращается правый операнд. При этом левый операнд должен принимать null.

При проверке объектов на равенство следует учитывать, что они равны не только, когда они имеют ненулевые значения, которые совпадают, но и когда оба объекта равны null

2.2. Тип данных String: операции, литералы, форматированный вывод

Создание строк

Создавать сроки можно, как используя переменную типа string и присваивая ей значение, так и применяя один из конструкторов класса String. Конструктор String имеет различное число версий. Так, вызов конструктора new String('a', 6) создаст строку "aaaaaa". И так как строка представляет ссылочный тип, то может хранить значение null.

Основные методы строк

Основная функциональность класса String раскрывается через его методы, среди которых можно выделить следующие:

- Compare: сравнивает две строки с учетом текущей культуры (локали) пользователя
- CompareOrdinal: сравнивает две строки без учета локали
- Contains: определяет, содержится ли подстрока в строке
- Concat: соединяет строки
- CopyTo: копирует часть строки или всю строку в другую строку
- EndsWith: определяет, совпадает ли конец строки с подстрокой
- Format: форматирует строку
- IndexOf: находит индекс первого вхождения символа или подстроки в строке
- Insert: вставляет в строку подстроку
- Join: соединяет элементы массива строк
- LastIndexOf: находит индекс последнего вхождения символа или подстроки в строке
- Replace: замещает в строке символ или подстроку другим символом или подстрокой
- Split: разделяет одну строку на массив строк
- Substring: извлекает из строки подстроку, начиная с указанной позиции
- ToLower: переводит все символы строки в нижний регистр
- ToUpper: переводит все символы строки в верхний регистр
- Trim: удаляет начальные и конечные пробелы из строки

Метод String.Format

Преобразует значения объектов в строки на основе указанных форматов и вставляет их в другую строку.

```
Decimal pricePerOunce = 17.36m;
String s = String.Format("The current price is {0} per ounce.", pricePerOunce);
// Result: The current price is 17.36 per ounce.

Decimal pricePerOunce = 17.36m;
String s = String.Format("The current price is {0:C2} per ounce.", pricePerOunce);
// Result if current culture is en-US:
// The current price is $17.36 per ounce.
```

Строковый литерал представляет собой набор символов, заключенных в двойные кавычки. Например следующий фрагмент кода:

"This is text"

Помимо обычных символов, строковый литерал может содержать одну или несколько управляющих последовательностей символов.

Также можно указать буквальный строковый литерал. Такой литерал начинается с символа @, после которого следует строка в кавычках. Содержимое строки в кавычках воспринимается без изменений и может быть расширено до двух и более строк. Это означает, что в буквальный строковый литерал можно включить символы новой строки, табуляции и прочие, не прибегая к управляющим последовательностям. Единственное исключение составляют двойные кавычки ("), для указания которых необходимо использовать двойные кавычки с обратным слэшем ("\").

Управляющая последовательность	Описание
\a	звуковой сигнал (звонок)
\b	возврат на одну позицию
\f	перевод страницы (переход на новую страницу)
\n	новая строка (перевод строки)
\r	возврат каретки
\t	горизонтальная табуляция
\v	вертикальная табуляция
\0	пустой символ
\'	одинарная кавычка
\"	двойная кавычка
\\	обратная косая черта

2.3. Неявная типизация – назначение и использование

Локальные переменные можно объявлять без указания конкретного типа. Ключевое слово **var** указывает, что компилятор должен вывести тип переменной из выражения справа от оператора инициализации. Выведенный тип может быть встроенным, анонимным

```
var i = 5;

// s is compiled as a string
```

```
var s = "Hello";

// a is compiled as int[]
var a = new[] { 0, 1, 2 };

// expr is compiled as IEnumerable<Customer>
// or perhaps IQueryable<Customer>
var expr =
    from c in customers
    where c.City == "London"
    select c;
```

В операторе инициализации for.

```
for(var x = 1; x < 10; x++)
```

2.4. Массивы C# одномерные, прямоугольные и ступенчатые

Одномерные массивы

тип[] имя;
тип[] имя = new тип [размерность];
тип[] имя = { список инициализаторов };
тип[] имя = new тип [] { список инициализаторов };
тип[] имя = new тип [размерность] { список инициализаторов };

```
int[] a;
int[] b;
int[] c = { 61, 2, 5, -9 };
int[] d = new int[] { 6, 2, 5, -9 };
int[] e = new int[4] { 61, 2, 5, -9 };
```

ПРЯМОУГОЛЬНЫЕ

```
int[,] a = { { 1, 2 }, { 2, 3 } };
            foreach (var x in a)
                Console.Write("\t" + x);
```

СТУПЕНЧАТЫЕ

```
int[][] a = new int[3][]; // выделение памяти под ссылки на три строки
a[0] = new int[5]; // выделение памяти под 0-ю строку (5 элементов)
a[1] = new int[3]; // выделение памяти под 1-ю строку (3 элемента)
a[2] = new int[4]; // выделение памяти под 2-ю строку (4 элемента)


int[][] b = { new int[5], new int[3], new int[4] };
```

2.5. Понятие кортежей. Свойства, создание

Кортежи (tuple) комбинируют объекты различных типов (от одного до восьми). Типы и выражения

Свойства:

►создается один раз и остается неименным (все свойства доступны только для чтения)

►позволяют использовать методы CompareTo, Equals, GetHashCode и ToString, свойство Size

►реализуют интерфейсы IStructuralEquatable, IStructuralComparable и IComparable (можно сравнивать)

Создание

```
static void Main(string[] args)
{
   var tuple = (5, 10);
   Console.WriteLine(tuple.Item1); // 5
   Console.WriteLine(tuple.Item2); // 10
   tuple.Item1 += 26;
   Console.WriteLine(tuple.Item1); // 31
   Console.Read();
}
```

3. Объектно-ориентированное программирование на C#

3.1. Принципы ООП. Классы. Элементы класса

Инкапсуляция - механизм связывающий вместе данные и код обрабатывающий эти данные и сохраняющий их от внешнего воздействия и ошибочного использования:

1) никто не знает, что внутри

2) никто не может менять данные снаружи

Свойства инкапсуляции:

Совместное хранение данных

Сокрытие внутренней информации от пользователя

Абстрактные типы данных

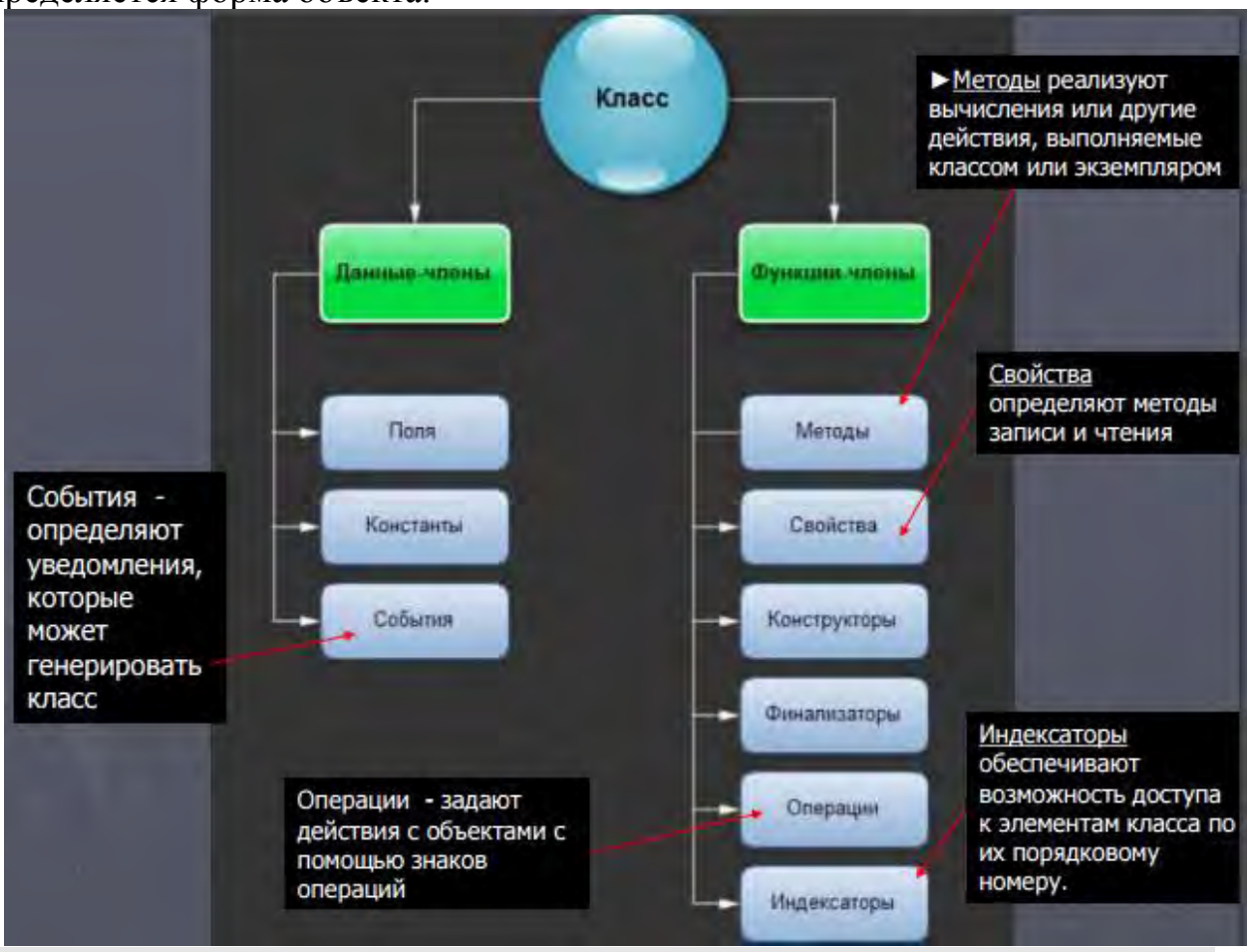
Подразумевают разделение и независимое рассмотрение интерфейса и реализации

Абстракция - уровень описания модели чего-либо.

Наследование – процесс, благодаря которому один объект может наследовать свойства от другого объекта

Полиморфизм- способность вызывать метод потомка через экземпляр предка. Поддержка осуществляется через виртуальные функции, перегрузки функций, а также обобщения.

Класс – это некоторое абстрактное понятие-шаблон, по которому определяется форма объекта.



Индексатор – перегружаем []. Инициализатор – в main в {} инициализируем объект класса.

Индексаторы (свойства с параметрами)

- Позволяют индексировать объекты таким же способом, как массив или коллекцию
- «умный» индекс для объектов
- средство, позволяющее разработчику перегружать оператор []

атрибуты спецификаторы тип this
[список_параметров]
get код доступа
set код доступа

Ограничения на индексаторы:

1) значение, выдаваемое индексатором, нельзя передавать методу в качестве параметра ref или out
2) индексатор не может быть объявлен как static

Свойства класса

- Свойства – специальные методы класса, служат для организации доступа к полям класса.
- Как правило, свойство связано с закрытым полем класса и определяет методы его получения и установки (предоставляет инкапсуляцию).
- Синтаксис свойства:

не void

```
[атрибуты]  [спецификаторы]  тип имясвойства
   {
       [get код_доступа]
       [set код_доступа]
    }
```

аксессоры

Ограничения свойств:

1) не может быть передано методу в качестве параметра ref или out.

2) не подлежит перегрузке

3) не должно изменять состояние базовой переменной при вызове аксессора get

4) могут быть статическими, экземплярными, абстрактными и виртуальными

5) могут иметь модификатор доступа

6) могут определяться в интерфейсах

Set мы сможем использовать только в данном классе - в его методах, свойствах, конструкторе

```
public int Len
     {
                    get { return Length; }
        private     set { Length = value; }
     } // только один модификатор доступен
```

Класс в C# - *user defined type, UDT*

[атрибуты] [спецификаторы]
class имякласса [: предок]
{
тело-класса
}

```
public const int y;
```

Константы

```
const int CC =100;
// значение не может изменено
readonly int FC;
```

1) компилятор сохраняет значение константы в метаданных модуля → константы можно определять только для таких типов, которые компилятор считает примитивными (или не примитивный но тогда = null)

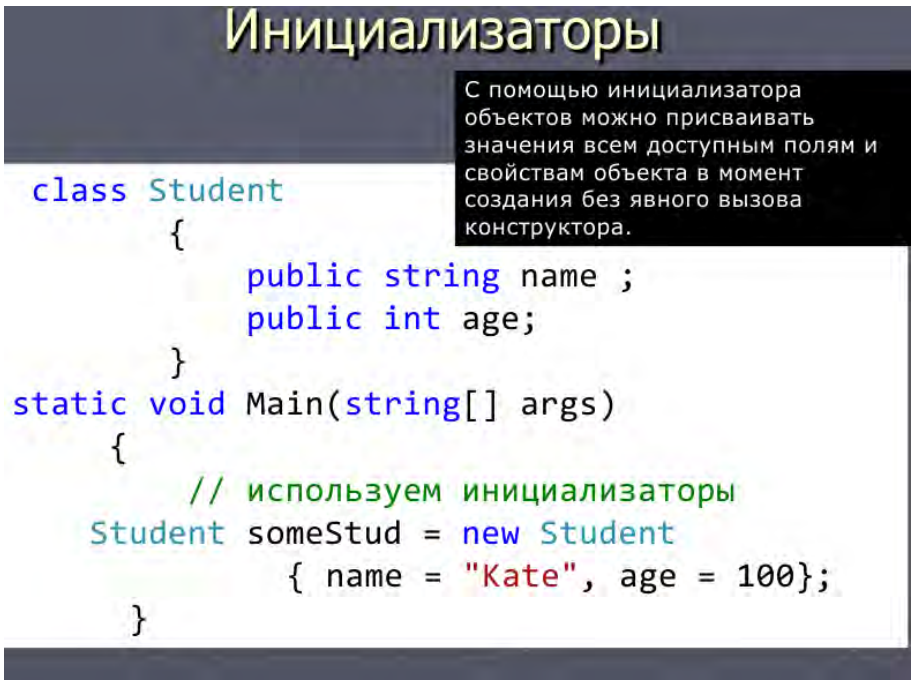
2) константы считаются не явно статическими, всегда связаны с типом, а не с экземпляром типа
3) нельзя получать адрес константы и передавать ее по ссылке
4) объявить можем один раз
5)к моменту компиляции они должны быть определены.

Поля для чтения

readonly - инициализация времени испол.

1) Запись в поле разрешается при объявлении или в коде конструктора
2) Инициализировать или изменять их значение в других местах нельзя, можно только считывать их значение.

```
class Point
    {
        public int x;
        public readonly int y = 0; // можно так инициализировать
        public Point (int _y)
            {
                y = _y; //может быть инициализировано
            } //или изменено в конструкторе после компиляции
        public void ChangeY(int _y)
            {
                y = _y; //  нельзя
            }
}
```



3.2. Спецификаторы доступа C#. Видимость типов

Видимость типа

- может быть открытым (public) или внутренним (internal).

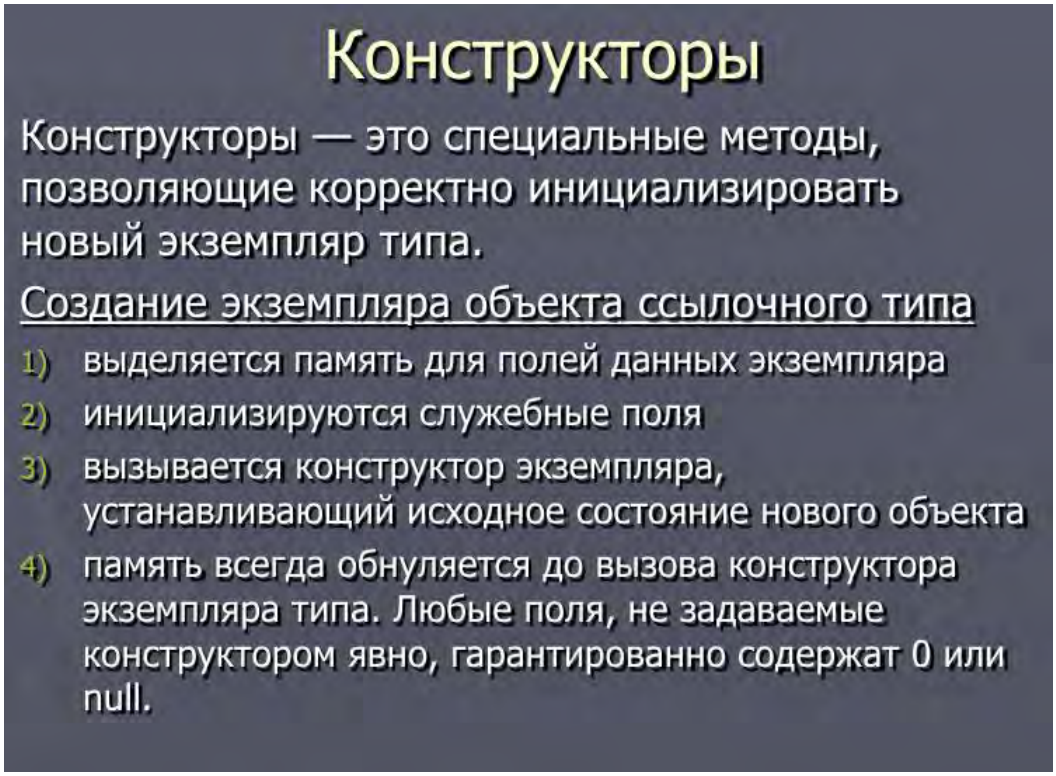
По умолчанию для класса

```
// Открытый тип доступен из любой сборки
    public class Машина {  }

// Внутренний тип доступен только из собственной
сборки
    internal class Колесо {  }

// Это внутренний тип, так как модификатор доступ
а не указан явно
    class Двигатель {  }
```

```
public class TestAccess
    {
      int age; // == private int age;
      private int bithday;
                  //  доступно только из текущего класса
      protected int date;
        // доступно из текущего класса и производных классов
        internal int sum;
                  // доступно в любом месте программы
        protected internal int email;
                   // доступно в любом месте программы
                     //и из классов-наследников
        public int adres;
                   // доступно в любом месте программы,
                 //а также для других программ и сборок (dll
    }
```

Инкапсуляция - скрытие некоторых моментов реализации класса от других частей программы.

Класс. Конструкторы и их свойства. Деструкторы



1. Не имеет возвращаемого значения.

2. Имя такое же как и имя типа(класса).

3. Не наследуется

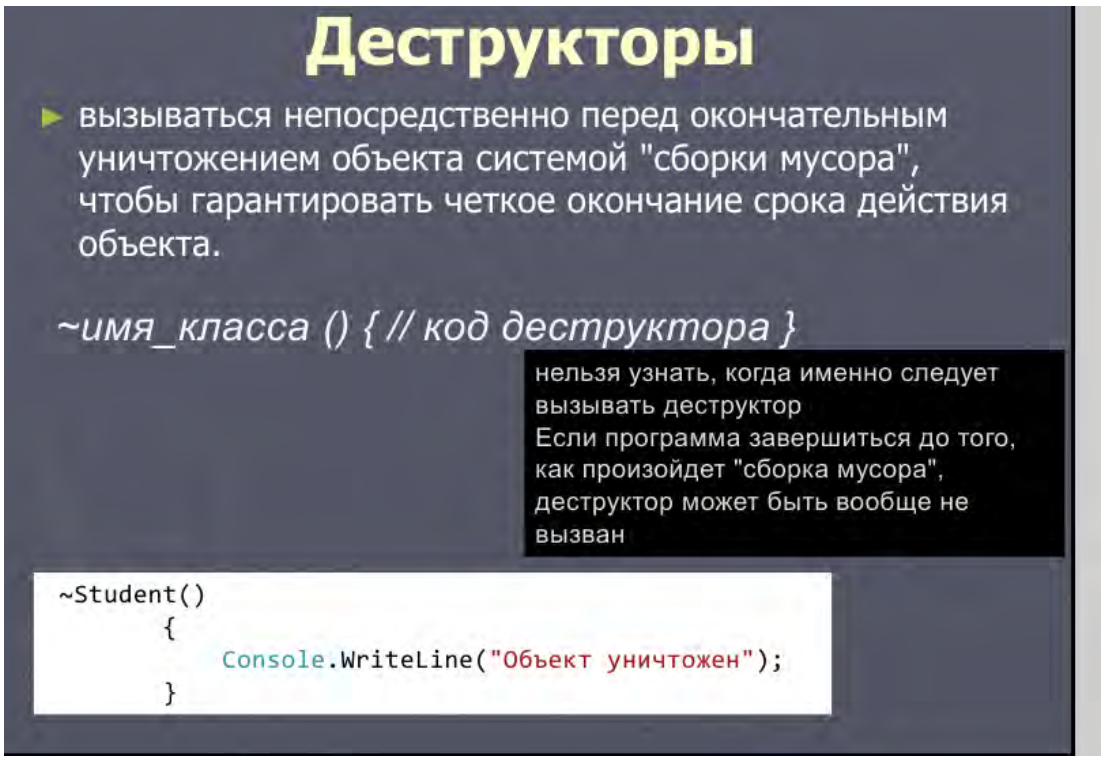
4. Нельзя применять модификаторы virtual,new,override,sealed и abstract.

5. Для класса без явно заданных конструкторов компилятор создает конструктор по умолчанию(без параметров).

6. Для статических классов(запечатанных и абстрактных)компилятор не создает конструктор по умолчанию.

7. Может определяться несколько конструкторов,сигнатуры и уровни доступа к конструкторам обязательно должны отличаться.

8. Можно явно заставлять один конструктор вызывать другой конструктор посредством зарезервированного слова this.



3.3. Класс и методы System.Object.

Конструкторы

	Имя	Описание
	Object()	Инициализирует новый экземпляр класса Object.

Методы

	Имя	Описание
	Equals(Object)	Определяет, равен ли заданный объект текущему объекту.
S	Equals(Object, Object)	Определяет, следует ли считать равными указанные экземпляры объектов.
	Finalize()	Позволяет объекту попытаться освободить ресурсы и выполнить другие операции по очистке перед тем, как объект будет утилизирован сборщиком мусора.
	GetHashCode()	Служит хэш-функцией по умолчанию.
	GetType()	Возвращает объект Type для текущего экземпляра.
	MemberwiseClone()	Создает неполную копию текущего объекта Object.
S	ReferenceEquals(Object, Object)	Определяет, совпадают ли указанные экземпляры Object.
	ToString()	Возвращает строковое представление текущего объекта.

Все остальные классы в .NET, даже те, которые мы сами создаем, а также базовые типы, такие как System.Int32, являются неявно производными от класса Object. Даже если мы не указываем класс Object в качестве базового, по умолчанию неявно класс Object все равно стоит на вершине иерархии наследования. Поэтому все типы и классы могут реализовать те методы, которые определены в классе System.Object. Рассмотрим эти методы.

Метод **ToString** служит для получения строкового представления данного объекта. Для базовых типов просто будет выводиться их строковое значение:

```
1  int i = 5;
2  Console.WriteLine(i.ToString()); // выведет число 5
3
4  double d = 3.5;
5  Console.WriteLine(d.ToString()); // выведет число 3,5
```

Для классов же этот метод выводит полное название класса с указанием пространства имен, в котором определен этот класс. И мы можем переопределить данный метод. Посмотрим на примере:

```
1    using System;
2
3    namespace FirstApp
4    {
5        class Program
6        {
7            private static void Main(string[] args)
8            {
9                Person person = new Person { Name = "Tom" };
10               Console.WriteLine(person.ToString()); // выведет название класса
11Person
12
13               Clock clock = new Clock { Hours = 15, Minutes = 34, Seconds = 53 };
14               Console.WriteLine(clock.ToString()); // выведет 15:34:53
15
16               Console.Read();
17           }
18       }
19       class Clock
20       {
21           public int Hours { get; set; }
22           public int Minutes { get; set; }
23           public int Seconds { get; set; }
24           public override string ToString()
25           {
26               return $"{Hours}:{Minutes}:{Seconds}";
27           }
28       }
```

```
29      class Person
30      {
31          public string Name { get; set; }
32      }
    }
```

Для переопределения метода ToString в классе Clock, который представляет часы, используется ключевое слово override (как и при обычном переопределении вирутальных или абстрактных методов). В данном случае метод ToString() значение свойств Hours, Minutes, Seconds, объединенные в одну строку.

Класс Person не переопределяет метод ToString, поэтому для этого класса срабатывает стандартная реализация этого метода, которая выводит просто название класса.

Кстати в данном случае мы могли задействовать обе реализации:

```
1   class Person
2   {
3       public string Name { get; set; }
4       public override string ToString()
5       {
6           if (String.IsNullOrEmpty(Name))
7               return base.ToString();
8           return Name;
9       }
10  }
```

То есть если имя - свойство Name не имеет значения, оно представляет пустую строку, то возвращается базовая реализация - название класса. Если же имя установлено, то возвращается значение свойства Name. Для проверки строки на пустоту применяется метод String.IsNullOrEmpty().

Стоит отметить, что различные технологии на платформе .NET активно используют метод ToString для разных целей. В частости, тот же метод Console.WriteLine() по умолчанию выводит именно строковое представление объекта. Поэтому, если нам надо вывести строковое представление объекта на консоль, то при передаче объекта в метод Console.WriteLine необязательно использовать метод ToString() - он вызывается неявно:

```
1   private static void Main(string[] args)
2   {
3       Person person = new Person { Name = "Tom" };
4       Console.WriteLine(person);
5
6       Clock clock = new Clock { Hours = 15, Minutes = 34, Seconds = 53 };
7       Console.WriteLine(clock); // выведет 15:34:53
8
9       Console.Read();
10  }
```

Метод **GetHashCode** позволяет возвратить некоторое числовое значение, которое будет соответствовать данному объекту или его хэш-код. По данному числу, например, можно сравнивать объекты. Можно определять самые разные алгоритмы генерации подобного числа или взять реализаци базового типа:

```
1   class Person
2   {
3       public string Name { get; set; }
4
5       public override int GetHashCode()
6       {
7           return Name.GetHashCode();
8       }
9   }
```

В данном случае метод GetHashCode возвращает то хеш-код для значения свойства Name. То есть два объекта Person, которые имеют одно и то же имя, будут возвращать один и тот же хеш-код. Однако в реальности алгоритм может быть самым различным.

Метод **GetType** позволяет получить тип данного объекта:

```
1   Person person = new Person { Name = "Tom" };
2   Console.WriteLine(person.GetType());    // Person
```

Этот метод возвращает объект Type, то есть тип объекта.

С помощью ключевого слова typeof мы получаем тип класса и сравниваем его с типом объекта. И если этот объект представляет тип Client, то выполняем определенные действия.

```
1   object person = new Person { Name = "Tom" };
2   if (person.GetType() == typeof(Person))
3       Console.WriteLine("Это реально класс Person");
```

Причем поскольку класс Object является базовым типом для всех классов, то мы можем переменной типа object присвоить объект любого типа. Однако для этого переменной метод GetType все равно вернет тот тип, на объект которого ссылается переменная. То есть в данном случае объект типа Person.

В отличие от методов ToString, Equals, GetHashCode метод GetType не переопределяется.

Метод **Equals** позволяет сравнить два объекта на равенство:

```
1     class Person
2     {
3         public string Name { get; set; }
4         public override bool Equals(object obj)
5         {
6             if (obj.GetType() != this.GetType()) return false;
7
8             Person person = (Person)obj;
9             return (this.Name == person.Name);
10        }
```

```
11    }
```

Метод Equals принимает в качестве параметр объект любого типа, который мы затем приводим к текущему, если они являются объектами одного класса. Затем сравниваем по именам. Если имена равны, возвращаем true, что будет говорить, что объекты равны. Однако при необходимости реализацию метода можно сделать более сложной, например, сравнивать по нескольким свойствам при их наличии.

Применение метода:

```
1    Person person1 = new Person { Name = "Tom" };
2    Person person2 = new Person { Name = "Bob" };
3    Person person3 = new Person { Name = "Tom" };
4    bool p1Ep2 = person1.Equals(person2);   // false
5    bool p1Ep3 = person1.Equals(person3);   // true
```

И если следует сравнивать два сложных объекта, как в данном случае, то лучше использовать метод Equals, а не стандартную операцию ==.

3.4. Статические методы и статические конструкторы класса

Иногда требуется определить такой член класса, который будет использоваться независимо от всех остальных объектов этого класса. Как правило, доступ к члену класса организуется посредством объекта этого класса, но в то же время можно создать член класса для самостоятельного применения без ссылки на конкретный экземпляр объекта. Для того чтобы создать такой член класса, достаточно указать в самом начале его объявления **ключевое слово static**.

Если член класса объявляется как static, то он становится доступным до создания любых объектов своего класса и без ссылки на какой-нибудь объект. С помощью ключевого слова static можно объявлять, как переменные, так и методы. Наиболее характерным примером члена типа static служит метод Main(), который объявляется таковым потому, что он должен вызываться операционной системой в самом начале выполняемой программы.

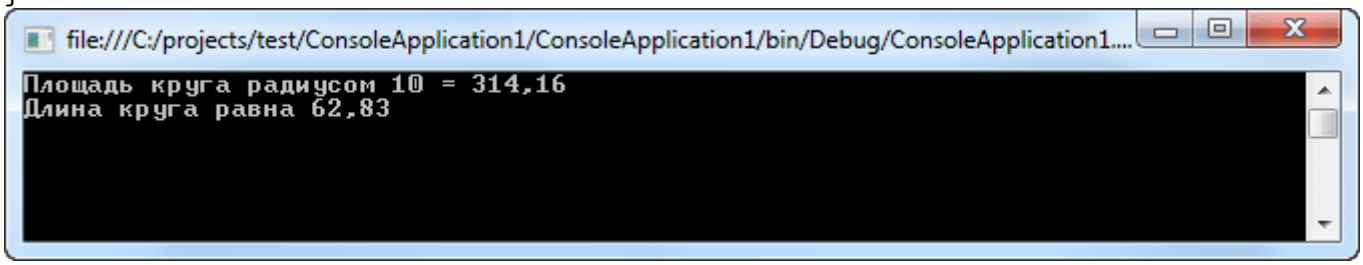
Для того чтобы воспользоваться членом типа static за пределами класса, достаточно указать имя этого класса с оператором-точкой. Но создавать объект для этого не нужно. В действительности член типа static оказывается доступным не по ссылке на объект, а по имени своего класса.

Переменные, объявляемые как static, по существу, являются глобальными. Когда же объекты объявляются в своем классе, то копия переменной типа static не создается. Вместо этого все экземпляры класса совместно пользуются одной и той же переменной типа static. Такая переменная инициализируется перед ее применением в классе.

Пример использования ключевого слова static:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

namespace ConsoleApplication1
{
    class myCircle
    {
        // 2 метода, возвращающие площадь и длину круга
        public static double SqrCircle(int radius)
     {
            return Math.PI * radius * radius;
        }

        public static double LongCircle(int radius)
     {
            return 2 * Math.PI * radius;
        }
    }

    class Program
    {
        static void Main(string[] args)
     {
            int r = 10;
            // Вызов методов из другого класса
            // без создания экземпляра объекта этого класса
            Console.WriteLine("Площадь круга радиусом {0} = {1:#.##}",r,myCircle.SqrCircle(r));
            Console.WriteLine("Длина круга равна {0:#.##}",myCircle.LongCircle(r));

            Console.ReadLine();
        }
    }
}
```



На применение методов типа static накладывается ряд следующих ограничений:

- В методе типа static должна отсутствовать ссылка this, поскольку такой метод не выполняется относительно какого-либо объекта
- В методе типа static допускается непосредственный вызов только других методов типа static, но не метода экземпляра из того самого же класса. Дело в том, что методы экземпляра оперируют конкретными объектами, а метод типа static не вызывается для объекта. Следовательно, у такого метода отсутствуют объекты, которыми он мог бы оперировать
- Аналогичные ограничения накладываются на данные типа static. Для метода типа static непосредственно доступными оказываются только

другие данные типа static, определенные в его классе. Он, в частности, не может оперировать переменной экземпляра своего класса, поскольку у него отсутствуют объекты, которыми он мог бы оперировать

Статические конструкторы

Конструктор можно также объявить как static. Статический конструктор, как правило, используется для инициализации компонентов, применяемых ко всему классу, а не к отдельному экземпляру объекта этого класса. Поэтому члены класса инициализируются статическим конструктором до создания каких-либо объектов этого класса:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class MyClass
    {
        public static int a;
        public int b;

        // Статический конструктор
        static MyClass()
     {
            a = 10;
        }

        // Обычный конструктор
        public MyClass()
     {
            b = 12;
        }
    }

    class Program
    {
        static void Main(string[] args)
     {
            Console.WriteLine("Доступ к экземпляру класса a: " + MyClass.a);

            MyClass obj = new MyClass();
            Console.WriteLine("Доступ к экземпляру класса b: " + obj.b);

            Console.ReadLine();
        }
    }
}
```

Обратите внимание на то, что конструктор типа static вызывается автоматически, когда класс загружается впервые, причем до конструктора экземпляра. Из этого можно сделать более общий вывод: статический конструктор должен выполняться до любого конструктора экземпляра. Более того, у статических конструкторов отсутствуют модификаторы доступа — они

пользуются доступом по умолчанию, а, следовательно, их нельзя вызывать из программы.

Статические члены класса

- переменные и свойства, которые хранят состояние, общее для всех объектов класса, следует определять как статические
- методы, которые определяют общее для всех объектов поведение, также следует объявлять как статические

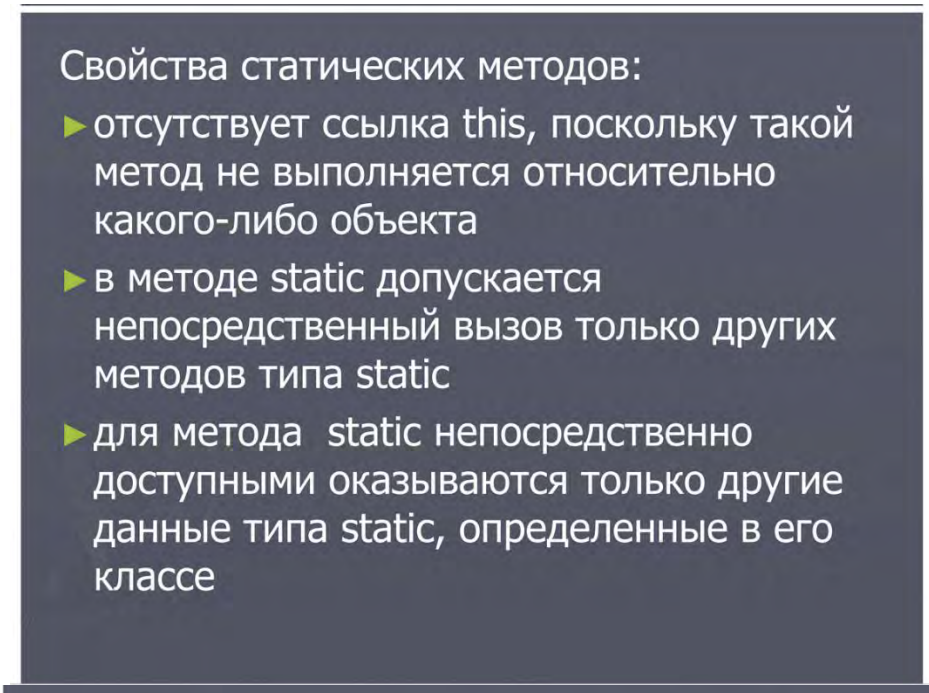
```
public class StudentBSTU
    {
        private static string uo;
         public static string UO { get; set; } = "BSTU";
         public static void getUo (){ Console.WriteLine(UO);
}
    }
```

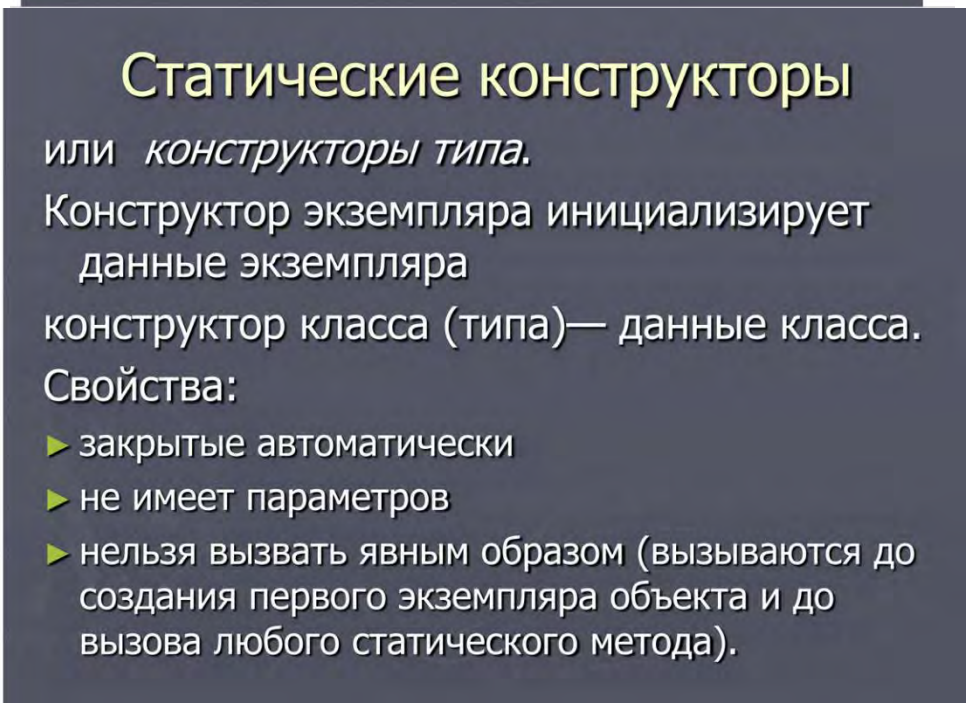
- При использовании статических членов необязательно создавать экземпляр класса

```
StudentBSTU.getUo();

Console.WriteLine(StudentBSTU.UO);
```

```
BSTU
BSTU
```

- Для статических полей будет создаваться участок в памяти, который будет общим для всех объектов класса.





Статические классы

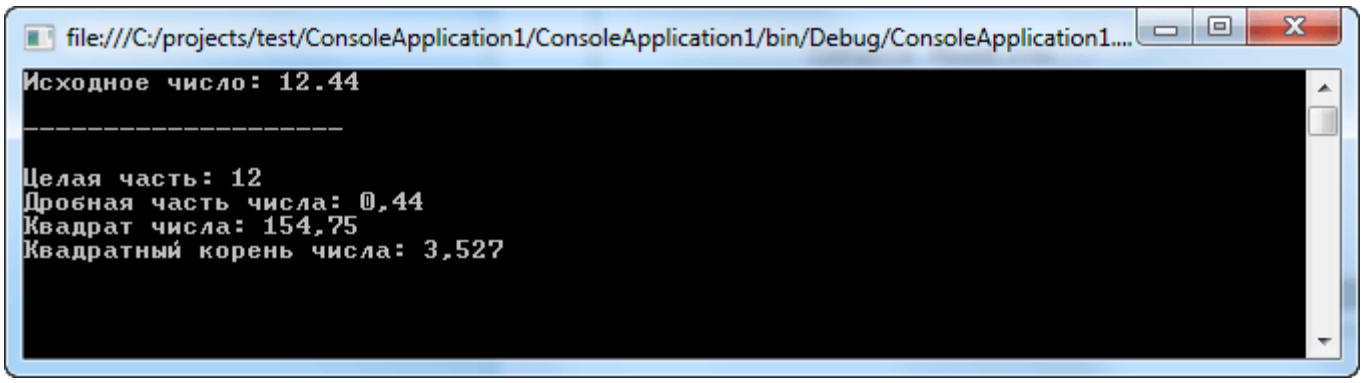
Класс можно объявлять как static. Статический класс обладает двумя основными свойствами. Во-первых, объекты статического класса создавать **нельзя**. И во-вторых, статический класс должен содержать только статические члены. Статический класс создается по приведенной ниже форме объявления класса, видоизмененной с помощью ключевого слова static.

static class имя класса { // ...

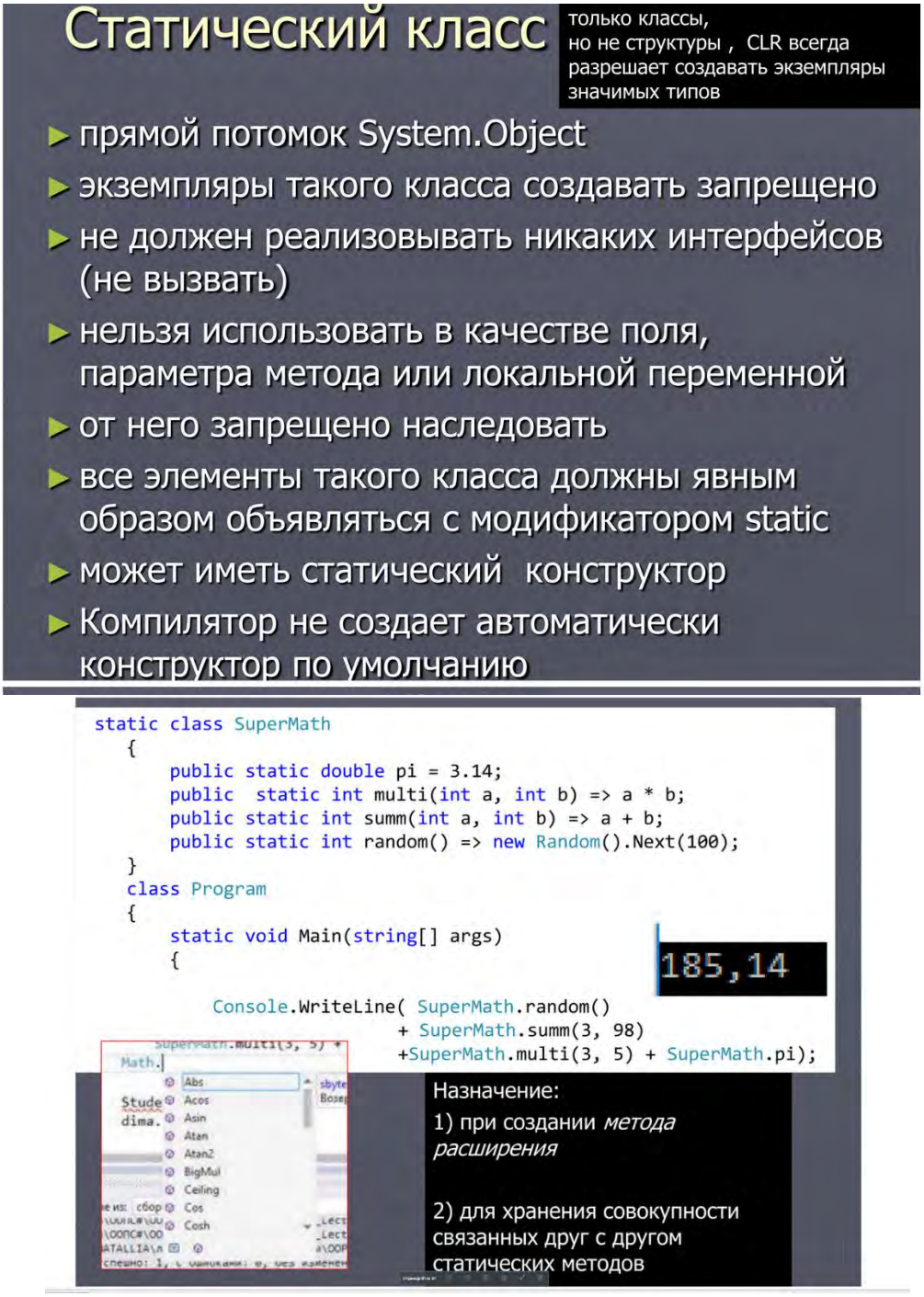
Статические классы применяются главным образом в двух случаях. Во-первых, статический класс требуется при создании *метода расширения*. Методы расширения связаны в основном с языком LINQ. И во-вторых, статический класс

служит для хранения совокупности связанных друг с другом статических методов:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    // В данном классе инкапсулируются статические методы
    // выполняющие простейшие операции
    static class MyMath
    {
        // Целая часть числа
        static public int round(double d)
    {
            return (int)d;
        }

        // Дробная часть числа
        static public double doub(double d)
    {
            return d - (int)d;
        }

        // Квадрат числа
        static public double sqr(double d)
    {
            return d * d;
        }

        // Квадратный корень числа
        static public double sqrt(double d)
    {
            return Math.Sqrt(d);
        }
    }

    class Program
    {
        static void Main(string[] args)
    {
            Console.WriteLine("Исходное число: 12.44\n\n----------\n");
            Console.WriteLine("Целая часть: {0}",MyMath.round(d: 12.44));
            Console.WriteLine("Дробная часть числа: {0}",MyMath.doub(d:
12.44));
            Console.WriteLine("Квадрат числа: {0:#.##}",MyMath.sqr(d:
12.44));
            Console.WriteLine("Квадратный корень числа:
{0:#.###}",MyMath.sqrt(d: 12.44));

            Console.ReadLine();
        }
    }
}
```



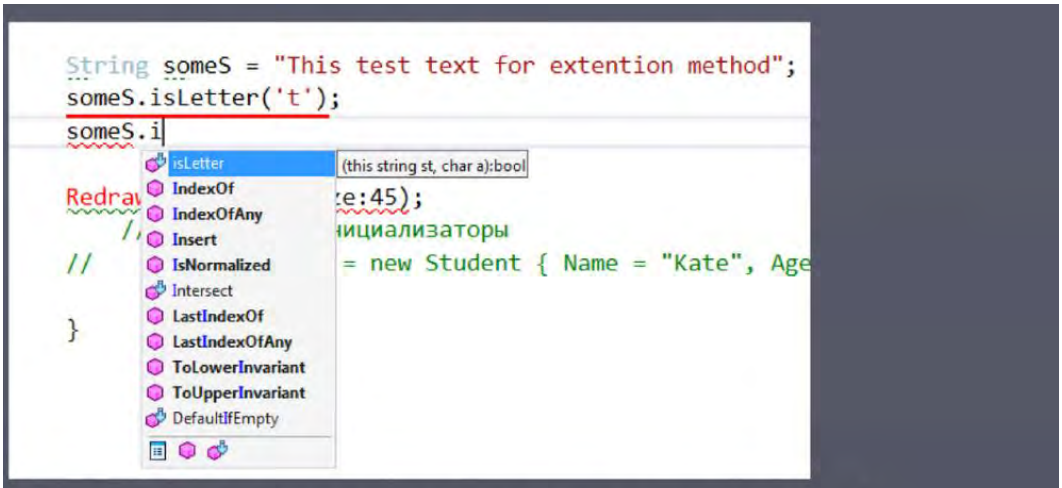
Стоит отметить, что для статического класса не допускается наличие конструктора экземпляра, но у него может быть статический конструктор.



Методы расширения

Методы расширения (extension methods) позволяют добавлять новые методы в уже существующие типы без создания нового производного класса.

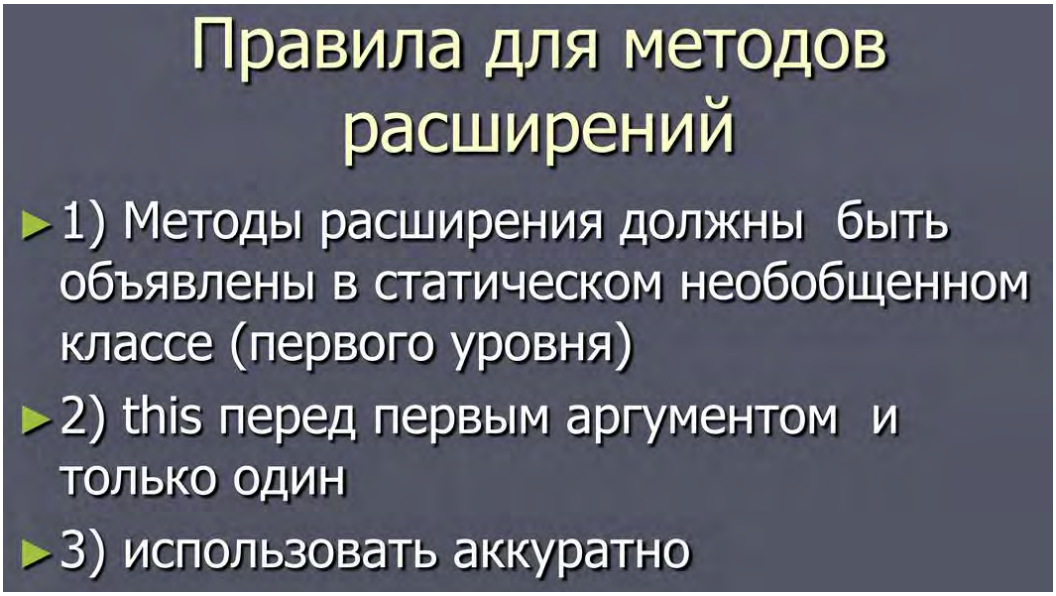
```
public static class NewFromAlex
{
    public static bool isLetter(this String st, char a)
     {
      for (Int32 index = 0; index < st.Length; index++)
                if (st[index] == a) return true;
           return false;
       }
 }
```



1) Проверяется класс и его базовые

2) Ищется любой статический

класс с методом ####, у которого первый параметр соответствует типу выражения (this)



Анонимные типы.

Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса. Анонимный тип определяется с помощью ключевого слова var и инициализатора объектов:

```
1    var user = new { Name = "Tom", Age = 34 };
2    Console.WriteLine(user.Name);
```

В данном случае user - это объект анонимного типа, у которого определены два свойства Name и Age. И мы также можем использовать его свойства, как и у обычных объектов классов. Однако тут есть ограничение - свойства анонимных типов доступны только для чтения.

При этом во время компиляции компилятор сам будет создавать для него имя типа и использовать это имя при обращении к объекту. Нередко анонимные типы имеют имя наподобие "<>f__AnonymousType0'2".

Для исполняющей среды CLR анонимные типы будут также, как и классы, представлять ссылочный тип.

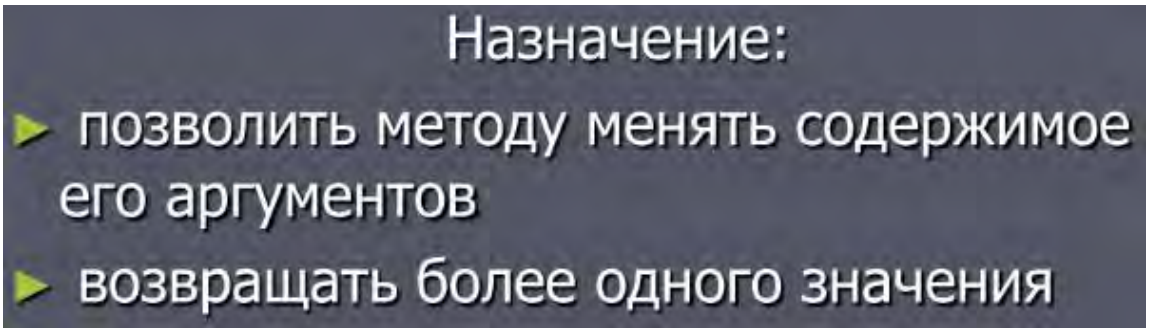
Если в программе используются несколько объектов анонимных типов с одинаковым набором свойств, то для них компилятор создаст одно определение анонимного типа:

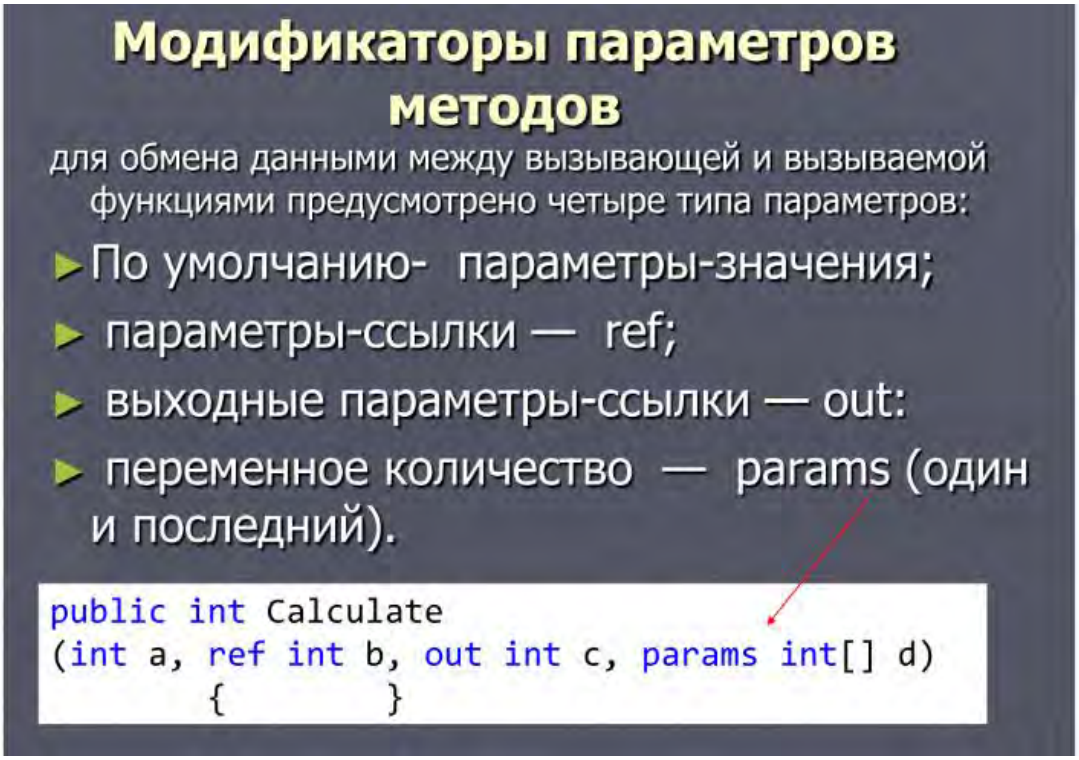
```
1    var user = new { Name = "Tom", Age = 34 };
2    var student = new { Name = "Alice", Age = 21 };
3    var manager = new { Name = "Bob", Age = 26, Company = "Microsoft" };
4
5    Console.WriteLine(user.GetType().Name); // <>f__AnonymousType0'2
6    Console.WriteLine(student.GetType().Name); // <>f__AnonymousType0'2
7    Console.WriteLine(manager.GetType().Name); // <>f__AnonymousType1'3
```

Здесь user и student будут иметь одно и то же определение анонимного типа. Однако подобные объекты нельзя преобразовать к какому-нибудь другому типу, например, классу, даже если он имеет подобный набор свойств.

Зачем нужны анонимные типы? Иногда возникает задача использовать один тип в одном узком контексте или даже один раз. Создание класса для подобного типа может быть избыточным. Если нам захочется добавить свойство, то мы сразу же на месте анонимного объекта это можем сделать. В случае с классом придется изменять еще и класс, который может больше нигде не использоваться. Типичная ситуация - получение результата выборки из базы данных: объекты используются только для получения выборки, часто больше нигде не используются, и классы для них создавать было бы излишне. А вот анонимный объект прекрасно подходит для временного хранения выборки.

Модификаторы параметров - ref, out, params. Необязательные и именованные аргументы.



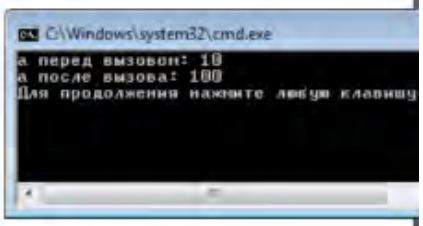


- *ref заставляет C# организовать вместо вызова по значению вызов по ссылке*

```
    class RefTest {
     public void sqr(ref int i)
         {i = i * i;}
}

    public static void Main()
{  RefTest ob = new RefTest();
int a = 10;

ob.sqr(ref a);
}
```

Аргументу, передаваемому методу "в сопровождении" модификатора ref, **должно быть присвоено значение до вызова метода.**

```
// Использование модификатора ref для передачи
// значения нессылочного типа по ссылке.

class RefTest
{
    // Этот метод изменяет свои аргументы.
    //Обратите внимание на использование модификатора ref.
    public void sqr(ref int i)
        {i = i * i;}
}
class RefDemo
    {
public static void Main()
    {
    RefTest ob = new RefTest();
    int a = 10;

    Console.WriteLine("a перед вызовом: " + a);
    ob.sqr(ref a);
    // использование модификатора ref.

    Console.WriteLine("a после вызова: " + a) ;

          }
  }
```



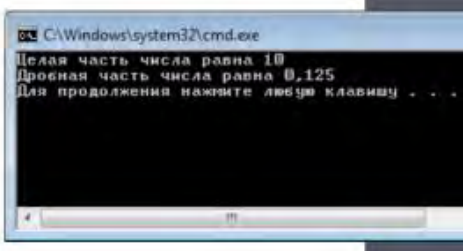
out

- *Модификатор out подобен модификатору ref за одним исключением:*

его можно использовать для передачи значения из метода

out-параметр "поступает" в метод без начального значения, но метод (до своего завершения) **обязательно** должен присвоить этому параметру значение

```
class Decompose
{
//Метод разбивает число с плавающей точкой на
//целую и дробную части
public int parts(double n, out double frac)
{
int whole;
whole = (int) n;
frac = n - whole; // Передаем дробную часть посредством параметра frac.

return whole;  // Возвращаем целую часть числа.
}

class UseOut
    {
        public static void Main()
        {
            Decompose ob = new Decompose();
            int i;
            double f;

            i = ob.parts(10.125, out f);

        Console.WriteLine("Целая часть числа равна " + i);
        Console.WriteLine("Дробная часть числа равна " + f);
                            }
     }
```



```
        // Демонстрация использования двух out-параметров.

class Num {
/*Метод определяет, имеют ли x и y общий делитель.
Если да, метод возвращает наименьший и наибольший
общие делители в out-параметрах. */
    public bool isComDenom(int x, int y,out int least,out int greatest)
    {
        int i;
        int max = x < y ? x : y;
        bool first = true;
        least = 1;
        greatest = 1;
        // Находим наименьший и наибольший общие делители.
        for (i = 2; i <= max / 2 + 1; i++)
        {
            if (((y % i) == 0) & ((x % i) == 0))
            {
                if (first)
                {
                    least = i;
                    first = false;

                }
                greatest = i;
                if (least != 1) return true;
                else return false;
            }
        }
        return false;
    }
```

```
class DemoOut
    {
         public static void Main()
   {
     Num ob = new Num();
     int led, gcd;
     if(ob.isComDenom(231, 105, out led, out gcd))
     {
     Console.WriteLine("Led для чисел 231 и 105 равен " +   led) ;
     Console.WriteLine("Gcd для чисел 231 и 105 равен " +   gcd) ;
     }

     else
         Console.WriteLine("Для чисел общего делителя нет.");

if(ob.isComDenom(35, 51, out led, out gcd))
{
Console.WriteLine("Led для чисел 35 и 51 равен " +     led) ;
Console.WriteLine("Gcd для чисел 35 и 51 равен " +     gcd) ;
}
else   Console.WriteLine( "Для чисел общего делителя нет.");}
          }
}
```

params

- позволяет передавать методу переменное количество аргументов одного типа

```
static void MaxArray(ref int _value, params int[] _arr)
        { if (_arr.Length>0)
            for (int j = 1; j < _arr.Length; j++)
                if (_arr[j] > _value)
                    _value = _arr[j];
        }

static void Check()
        {
            int result = -100;
            MaxArray(ref  result, 2,4,5,6,3,567);
            Console.WriteLine($"Максимум:{result}");
        }
```

```
Максимум:567
```

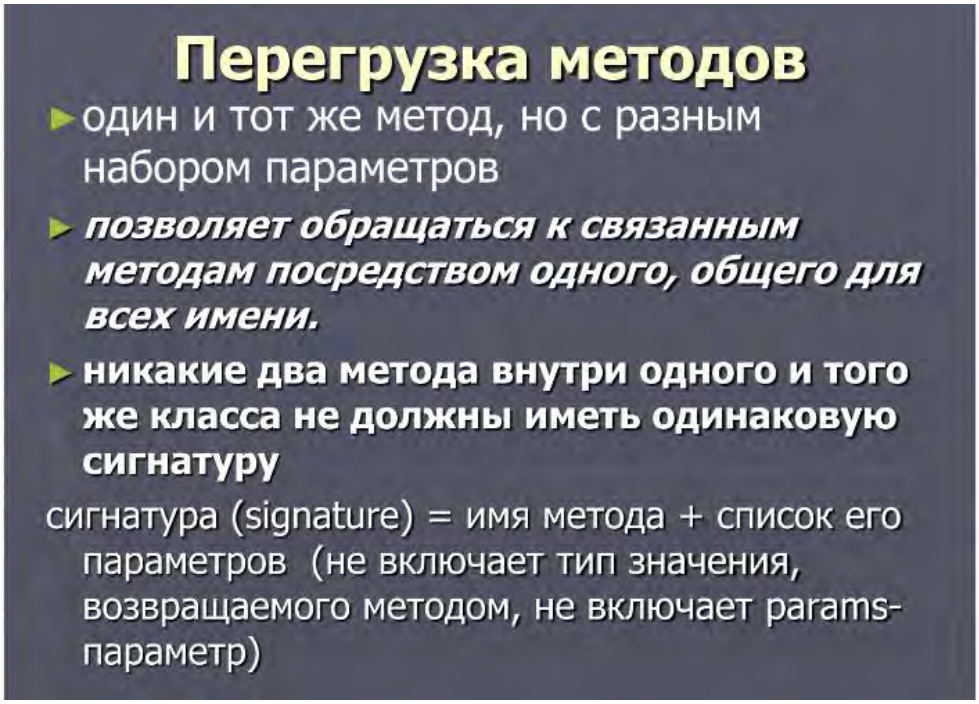
Необязательные аргументы

- позволяет определить используемое по умолчанию значение для параметра метода
- можно применять в конструкторах, индексаторах

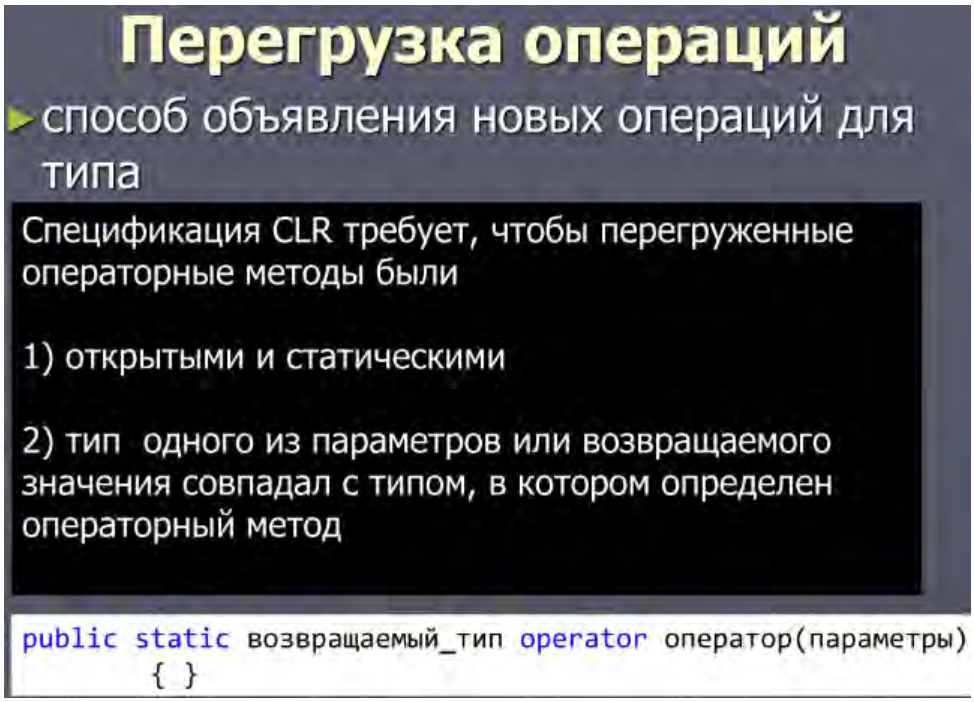
```
static void RedrawButton(int color ,
                         int type = 2 ,
                         int size = 4)
{ }
static void Main(string[] args)
{
    RedrawButton(243);
}
```

должны указываться справа от обязательных

3.5. Перегрузка методов и операторов



```
// -----------------Перегрузка методов--------
        // Возвращает наибольшее из двух целых:
      class Test{
       public int max( int a, int b ) {return 1;}
        // Возвращает наибольшее из трех целых:
       public int max(int a, int b, int c) { return 2; }
        // Возвращает наибольшее из первого параметра и длины второго:
       public int max(int a, string b) { return 3; }
        // Возвращает наибольшее из второго параметра и длины первого:
       public int max(string b, int a) { return 4; }
       public int max(int a, ref int b) { return 5; }
      }
      public static void Main()
      {
          Test q = new Test();
          Console.WriteLine(q.max(1, 2));
          Console.WriteLine(q.max(1, 2, 3));
          Console.WriteLine(q.max(1, "222"));
          Console.WriteLine(q.max("123", 2));

      }
```



```
class BigInt
        {
            public int Value { get; set; }

            public static BigInt operator +(BigInt operand1,
                                            BigInt operand2)
            {
                return new BigInt
                      {Value = operand1.Value + operand2.Value };
            }
            public static bool operator >(BigInt c1, BigInt c2)
            {
                if (c1.Value > c2.Value)
                    return true;
                else
                    return false;
            }
//...
}
```

```
BigInt _i1 = new BigInt { Value = 50 };
BigInt _i2 = new BigInt { Value = 105 };
Console.WriteLine(_i1 > _i2); // false
Console.WriteLine((_i1+_i2).Value);  // 155
```

Операции подлежащие перегрузке

- +, -, !, ++, --
- true, false (попарно)
- +, -, *, /, %, &, |, ^, <<, >>
- ==, !=, <, >, <=, >= (перегрузка парами)

Операции не подлежащие перегрузке

- [] (но есть индексатор)
- () (можно определить новые операторы преобразования)
- +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>= (но получаем автоматически в случае перегрузки бинарной операции)
- &&, ||
- =, ., ?:, ??, ->, =>, f(x), as, checked, unchecked, default, delegate, is, new, sizeof, typeof

правила:

- префиксные операции ++ и – – перегружаются парами;
- операции сравнения перегружаются парами: == и != ; < и >;<= и >=.
- Перегруженные операции обязаны возвращать значения
- Должны объявляться как public и static
- префиксная и постфиксная формы операций ++ и --, в отличие от оригинальных операций, семантически НЕ различаются.

- может быть перегружен (т.к. это метод)

```
public static BigInt operator +(BigInt operand1, BigInt
operand2)
            {
                return new BigInt
               { Value = operand1.Value + operand2.Value };
            }
            public static BigInt operator +(BigInt operand1,
double operand2)
            {
                return new BigInt
                { Value = operand1.Value + (int)operand2};
            }
```

если перегружаются операторы == и !=, то для этого требуется переопределить методы Object.Equals() и Object.GetHashCode().

```
class Point2D
{
    float x, y;
    Point2D()
    {
        x = 0;
        y = 0;
    }
    Point2D(Point2D key)
    {
        x = key.x;
        y = key.y;
    }
    // Перегруженные операции обязаны возвращать значения!
    // Должны объявляться как public и static.
    // При этом префиксная и постфиксная формы операций ++ и --,
    // в отличие от оригинальных операций, семантически НЕ различаются.
    // Каждая из этих операций может быть объявлена либо как префиксная:
    public static Point2D operator ++(Point2D par)
    {
        par.x++;
        par.y++;
        return par;
    }
    // либо как постфиксная!
    public static Point2D operator --(Point2D par)
    {
        Point2D tmp = new Point2D(par);
        // Скопировали старое значение.
        par.x--;
        par.y--;
        // Модифицировали исходное значение. Но возвращаем старое!
        return tmp;
    }
```

```
public static bool operator == ( Point2D a, Point2D b )
{  return a.Equals( b );  }

public static bool operator !=(Point2D a, Point2D b)
{  return ! a.Equals( b );  }
```

```
// Бинарные операции также обязаны возвращать значения!
public static Point2D operator +(Point2D par1, Point2D par2)
{
    return new Point2D(par1.x + par2.x, par1.y + par2.y);
}

// Point2D + float
public static Point2D operator +(Point2D par1, float val)
{
    return new Point2D(par1.x + val, par1.y + val);
}

// float + Point2D
public static Point2D operator +(float val, Point2D par1)
{
    return new Point2D(val + par1.x, val + par1.y);
}
```

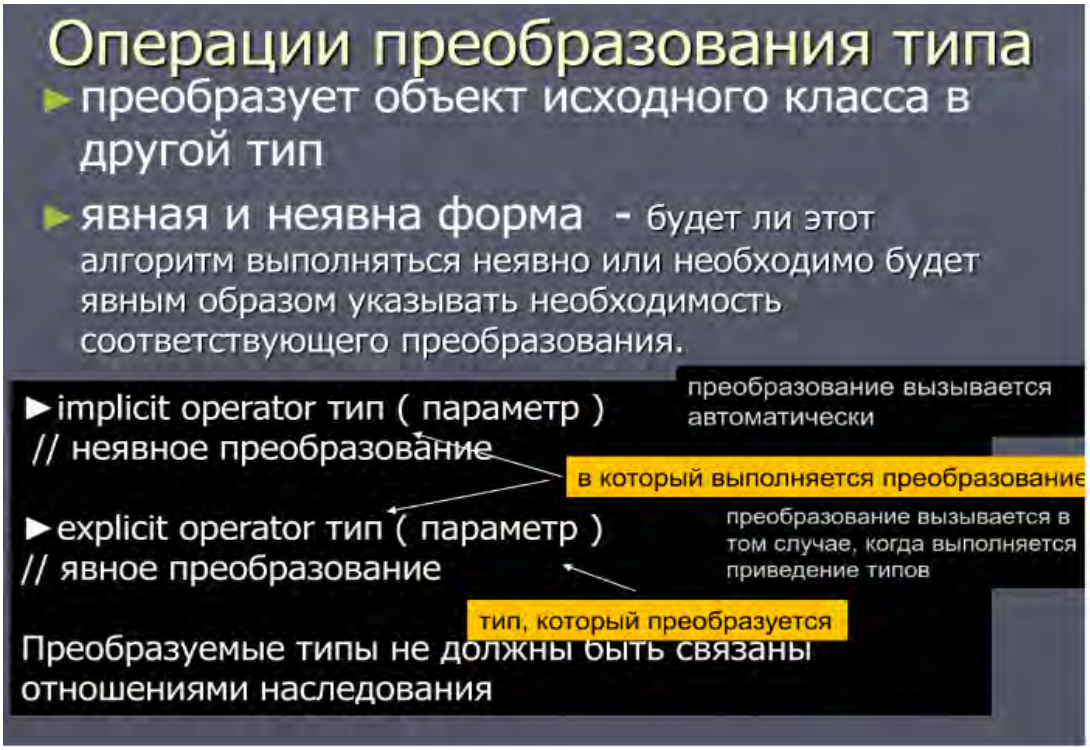
```
// Перегрузка булевских операторов. Это ПАРНЫЕ операторы.

public static bool operator true(Point2D par)
{
    if (par.x == 1.0F && par.y == 1.0F) return true;
    else return false;
}

public static bool operator false(Point2D par)
{
    if (par.x == 0.0F && par.y == 0.0F) return false;
    else return true;
}
```

Point2D.false(x)? x: Point2D.|(x, y)

```
public static Point2D operator | (Point2D par1, Point2D par2)
    {
    if (par1) return par1;
    if (par2) return par2;
    else return new Point2D(-1.0F, -1.0F);
    }

public static Point2D operator & (Point2D par1, Point2D par2)
    {
    if (par1 && par2) return par1;
    else return new Point2D(-1.0F, -1.0F);
    }
```

```
float значение >> 5,6
float значение >> 3,4
**********
float значение >> 3,4
Are You sure to change the y value of object of Point2D? (y/n) >> n
**********
p0.x == 0, p0.y == 0
p1.x == 5,6, p1.y == 3,4
p2.x == 5,6, p2.y == 3,4
##########
false!
true!
true!
true!
true!
p0.x == 1, p0.y == 1
p1.x == 6,6, p1.y == 4,4
p2.x == 5,6, p2.y == 3,4
Для продолжения нажмите любую клавишу . . .
```

3.6. Операции преобразования типа. Явная и неявная форма



```
public int x, y, z;

public Point3D()
{
    x = 0;
    y = 0;
    z = 0;
}

public Point3D(int xKey, int yKey, int zKey)
{
    x = xKey;
    y = yKey;
    z = zKey;
}

// Операторная функция, в которой реализуется алгоритм преобразования
// значения типа Point2D в значение типа Point3D.
// Это преобразование осуществляется НЕЯВНО.
public static implicit operator Point3D(Point2D p2d)
{
    Point3D p3d = new Point3D();
    p3d.x = p2d.x;
    p3d.y = p2d.y;
    p3d.z = 0;
    return p3d;
}
```

```
public Point2D()
{
    x = 0;
    y = 0;
}

public Point2D(int xKey, int yKey)
{
    x = xKey;
    y = yKey;
}

// Операторная функция, в которой реализуется алгоритм преобразования
// значения типа Point3D в значение типа Point2D. Это преобразование
// осуществляется с ЯВНЫМ указанием необходимости преобразования.
// Принятие решения относительно присутствия в объявлении ключевого
// слова explicit вместо implicit оправдывается тем, что это
// преобразование сопровождается потерей информации. По мнению
// разработчика классов об этом обстоятельстве следует напоминать
// каждый раз, когда данное преобразование случается в программе.
public static explicit operator Point2D(Point3D p3d)
{
    Point2D p2d = new Point2D();
    p2d.x = p3d.x;
    p2d.y = p3d.y;
    return p2d;
}
```

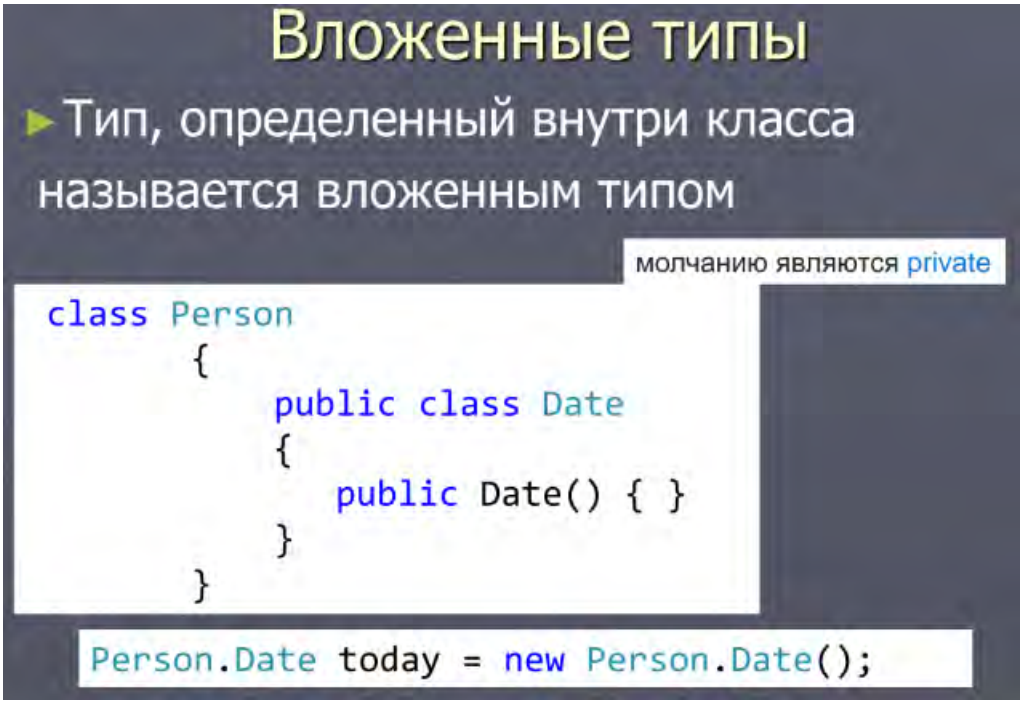
```
class TestClass
{
    static void Main(string[] args)
    {
        Point2D p2d = new Point2D(125, 125);
        Point3D p3d; // Сейчас это только ссылка!
        // Этой ссылке присваивается значение в результате
        // НЕЯВНОГО преобразования значения типа Point2D к типу Point3D
        p3d = p2d;

        // Изменили значения полей объекта.
        p3d.x = p3d.x * 2;
        p3d.y = p3d.y * 2;
        p3d.z = 125; //  появилась новая информация,
        // которая неизбежно будет потеряна в случае присвоения значения типа Point3D
        // значению типа Point2D. Ключевое слово explicit в объявлении соответствующего
        // метода преобразования приводит к тому, что программист всякий раз вынужден
        // повторять, что он в курсе возможных последствий.
        p2d = (Point2D)p3d;
    }
}
```

- Ключевые слова implicit и explicit в сигнатуру не включаются

Ограничения на операторы преобразования

- Исходный или целевой тип преобразования должен относиться к классу, для которого объявлено данное преобразование
- Нельзя указывать преобразование в/из класс object или же из этого класса
- Для одних типов данных нельзя указывать одновременно явное и неявное преобразование
- Нельзя указывать преобразование базового класса в производный класс
- Нельзя указывать преобразование в/из интерфейс

Вложенные типы, объекты



- Вложенный тип может получить доступ к внешнему типу, а внутренний тип — к внешнему
- Вложенный тип имеет доступ ко всем членам, которые доступны вмещающему типу

▶ Вложение или включение классов модель включения-делегирования.

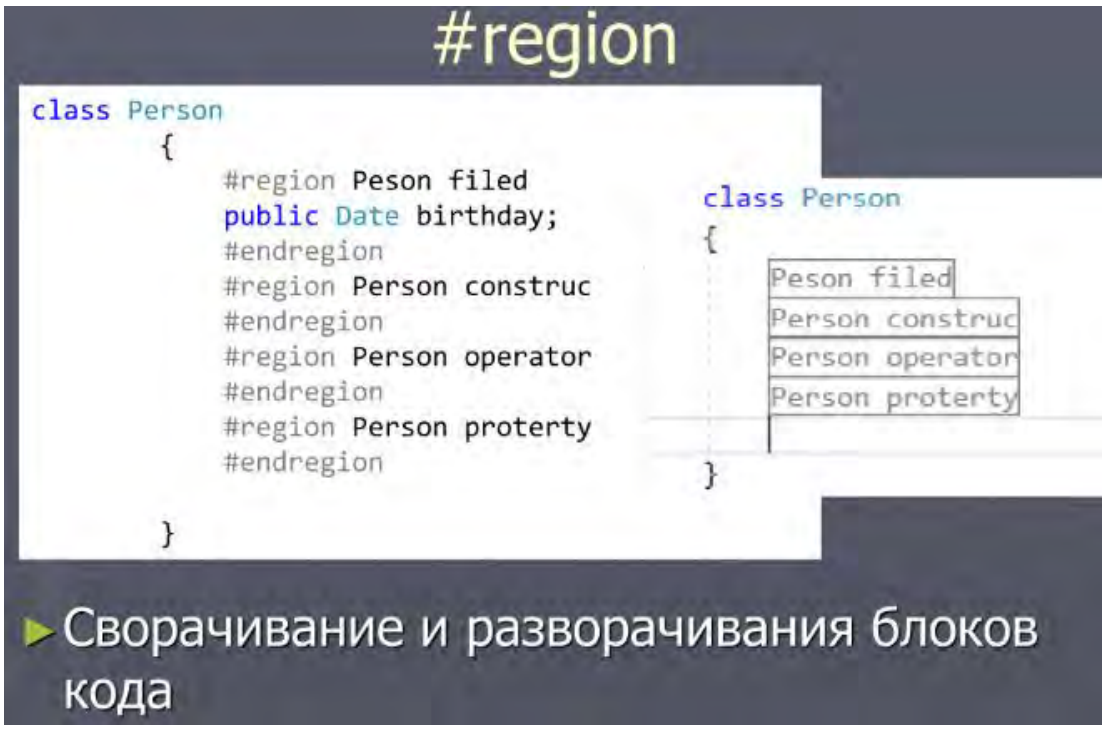
```
class Нога
  {
    public void Марш()
        {
        Console.WriteLine( "топ!!" );
        }
  }
class Человек
        { public Человек()
        {
        левая = new Нога() ;
        правая = new Нога();
        }
public void Побежали()
        {
        левая.Марш();
        правая.Марш();
        }
    Нога левая, правая;
  }

      class Class1
{
          static void Main()
              {
          Человек Вася = new Человек();
           Вася.Побежали();
              }
}
```

```
C:\Windows\system32\cmd.exe
топ!!
топ!!
Для продолжения нажмите любую клавиш
```

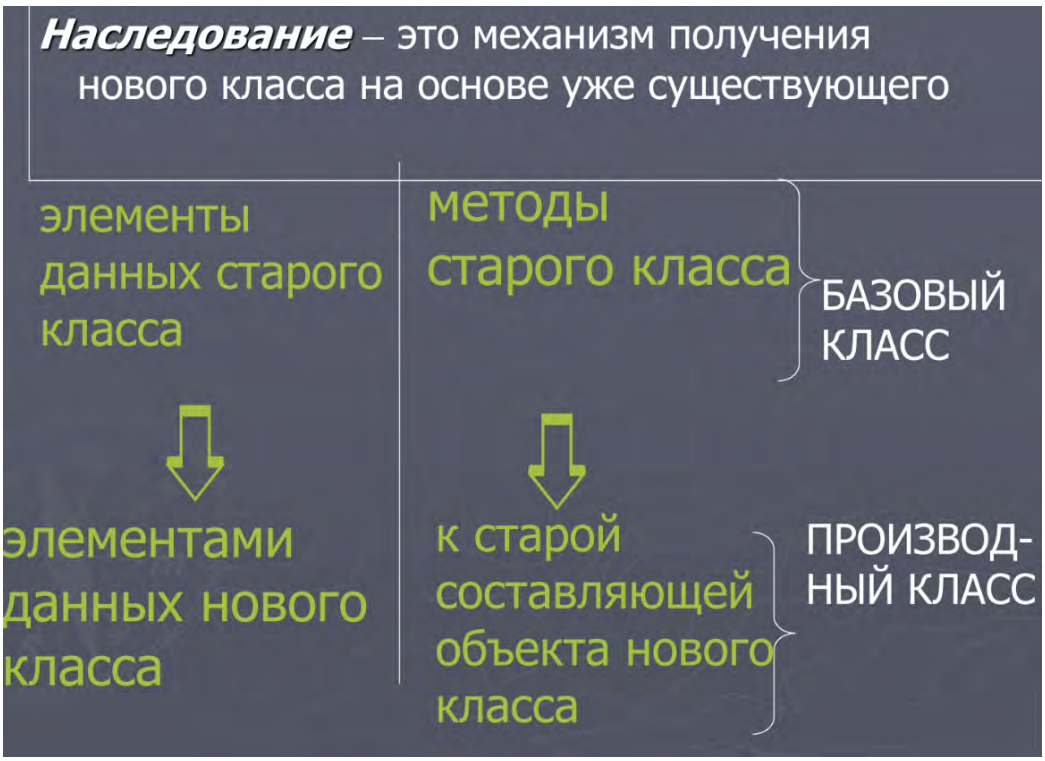
Вложенные объекты

```
public class Date
        {
          public  Date() { }
        }
      class Person
      {
          Date birthday;


      }
```

```
Person anna = new Person();
anna.birthday = null;
```



3.7. Правила наследования



Правила наследования:

1) В C# наследование всегда подразумевается открытым

```
class Student : Person
```

2) Запрещено множественное наследование классов (но не интерфейсов)

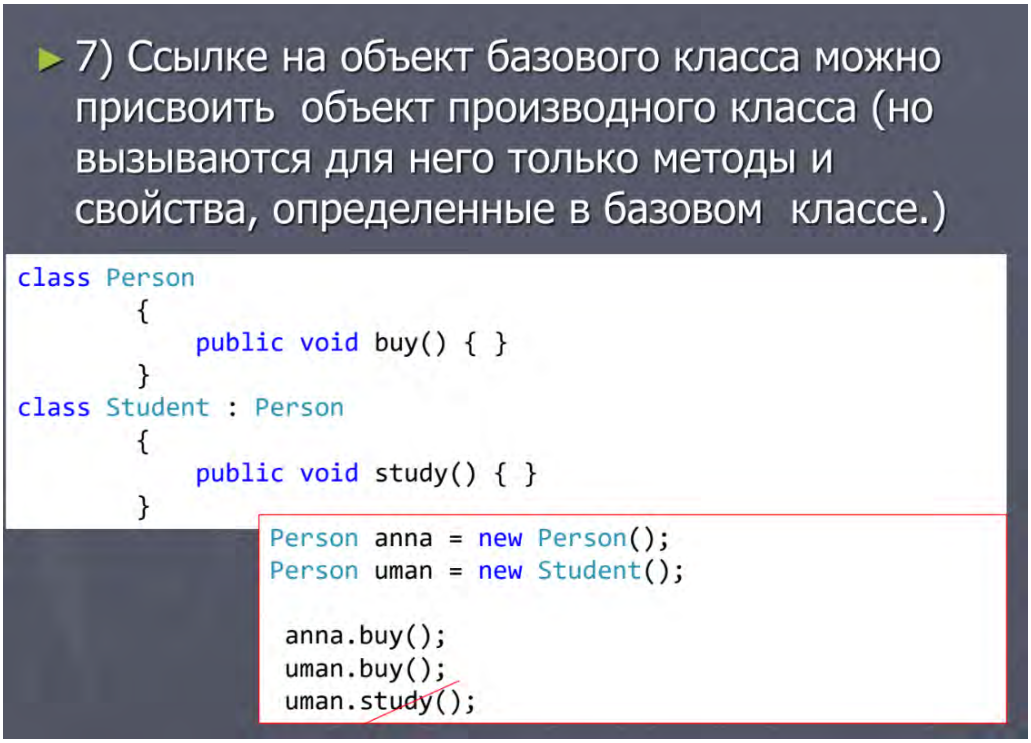
3) наследуются все е свойства, методы, поля, которые есть в базовом классе

4) Производному классу доступны public, internal, protected и protected internal члены базового класса (private – недоступны)

5) не наследуются конструкторы базового класса

6) тип доступа к производному классу должен быть таким же, как и у базового класса или более строгим

```
internal class Машина { }

public class Грузовик :Машина{ }
```



Наследование, вместе с инкапсуляцией и полиморфизмом, является одной из трех основных характеристик объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в другом классе. Класс, члены которого наследуются, называется *базовым классом*, а класс, который наследует эти члены, называется *производным классом*. Производный класс может иметь только один прямой базовый класс. Однако наследование является транзитивным. Если класс ClassC является производным от ClassB, и ClassB является производным от ClassA, ClassC наследует члены, объявленные в ClassB и ClassA.

При определении класса для наследования от другого класса производный класс явно получает все члены базового класса за исключением конструкторов и методов завершения. Производный класс может, таким образом, повторно использовать код в базовом классе без необходимости в его повторной реализации. В производном классе можно добавить больше членов. Таким образом, производный класс расширяет функциональность базового класса.

Абстрактные и виртуальные методы

Когда базовый класс объявляет метод как виртуальный, производный класс может переопределить метод с помощью своей собственной реализации. Если базовый класс объявляет член как абстрактный, этот метод должен быть переопределен в любом неабстрактном классе, который прямо наследует от этого класса. Если производный класс сам является абстрактным, то он наследует абстрактные члены, не реализуя их. Абстрактные и виртуальные члены являются основой для полиморфизма, который является второй основной характеристикой объектно-ориентированного программирования. Дополнительные сведения см.

Абстрактные базовые классы

Можно объявить класс как абстрактный, если необходимо предотвратить прямое создание экземпляров с помощью ключевого слова new. При таком подходе класс можно использовать, только если новый класс является производным от него. Абстрактный класс может содержать один или несколько сигнатур методов, которые сами объявлены в качестве абстрактных. Эти сигнатуры задают параметры и возвращают значение, но не имеют реализации (тела метода). Абстрактному классу необязательно содержать абстрактные члены; однако если класс все же содержит абстрактный член, то сам класс должен быть объявлен в качестве абстрактного. Производные классы, которые сами не являются абстрактными, должны предоставить реализацию для любых абстрактных методов из абстрактного базового класса. Дополнительные сведения см. в разделе Абстрактные и запечатанные классы и члены классов.

Интерфейсы

Интерфейс является ссылочным типом, в чем-то схожим с абстрактным базовым классом, который состоит только из абстрактных членов. Если класс реализует интерфейс, этот класс должен предоставлять реализацию для всех членов интерфейса. В классе может быть реализовано несколько интерфейсов, хотя производным он может быть только от одного прямого базового класса.

Интерфейсы используются для определения определенных возможностей для классов, которые не обязательно имеют отношения тождественности. Например, интерфейс System.IEquatable<T> может быть реализован любым классом или структурой, включающими клиентский код для определения эквивалентности двух объектов типа (однако тип определяет эквивалентность). IEquatable<T> не подразумевает тот же вид отношений тождественности, который существует между базовым и производным классами (например, `Mammal` является `Animal`). Дополнительные сведения см. в статье Интерфейсы.

Сокрытие имён при наследовании. Обращение с скрытым членам.

В производном классе можно определить член с таким же именем, как и у члена его базового класса. В этом случае член базового класса скрывается в производном классе. И хотя формально в C# это не считается ошибкой, компилятор всё же выдаст сообщение-предупреждение о том, что имя скрывается. Если член базового класса требуется скрыть намеренно, то перед его именем следует указать ключевое слово new, чтобы избежать появления подобного сообщения. Следует иметь ввиду, что это совершенно отдельное применение ключевого слова new, не похожее на его применение при создании экземпляров объекта.

```
namespace ConsoleApplication1
{
    class MyClass
    {
        public int x = 10, y = 5, z = 6;
    }

    class ClassA : MyClass
    {
        // Скрываем члены класса MyClass
        public new int x = 12, y = -2, z = -5;
    }

    class ClassB : MyClass
    {
        public int x;
    }

    class Program
    {
        static void Main()
        {
            ClassA obj1 = new ClassA();
            ClassB obj2 = new ClassB();

            Console.WriteLine("Координаты объекта obj1: {0} {1} {2}",obj1.x,obj1.y,obj1.z);
            Console.WriteLine("Координаты объекта obj2: {0} {1} {2}", obj2.x, obj2.y, obj2.z);

            Console.ReadLine();
        }
    }
}
```

Применение ключевого слова base для доступа к скрытому имени

Имеется ещё одна форма ключевого слова base, которая действует подобно ключевому слову this, за исключением того, что она всегда ссылается на базовый класс в том производном классе, в котором она используется: base.член где член может обозначать метод или поле экземпляра. Эта форма ключевого слова base чаще всего применяется в тех случаях, когда под именами членов производного класса скрываются члены базового класса с теми же именами. С помощью base могут вызываться скрытые методы

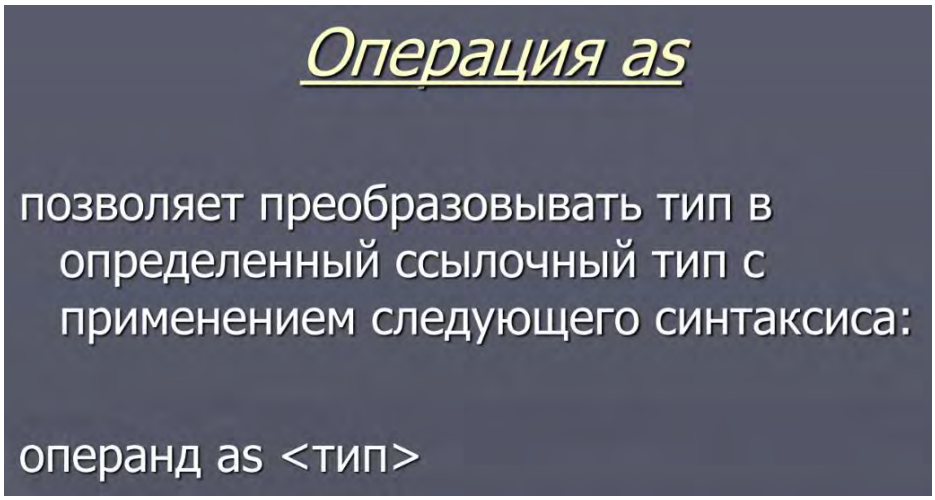
```
class MyClass
{
    public int x;
}

class ClassA : MyClass
{
    new int x = 10;

    public void someMethod(int i1, int i2)
    {
        // Координата x из базового класса MyClass
        base.x = i1;
        Console.WriteLine("x (в базовом классе) = " + base.x);
        // Координата x из класса ClassA
        x = i2;
        Console.WriteLine("x (в производном классе) = " + x);
    }
}

class Program
{
    static void Main()
    {
        ClassA obj1 = new ClassA();
        obj1.someMethod(1,25);

        Console.ReadLine();
    }
}
```

Использование операций is и as

Оператор `as` можно использовать для выполнения определенных типов преобразований между совместимыми ссылочными типами или типами, допускающими значение NULL. Вот пример кода:

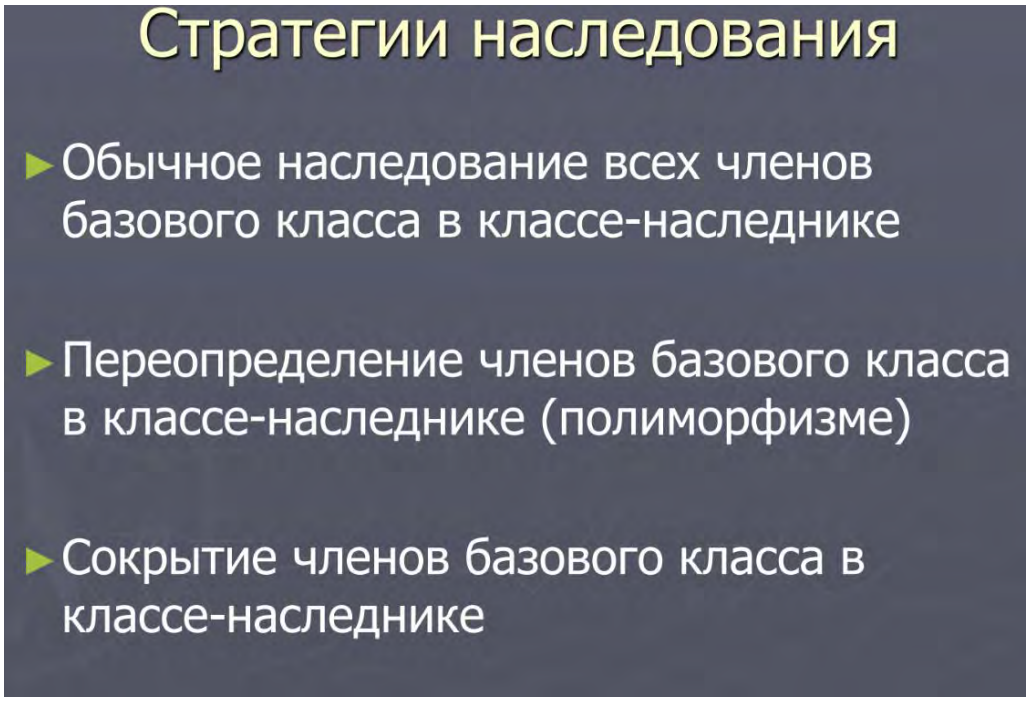
C#

```
class csrefKeywordsOperators
{
    class Base
    {
        public override string  ToString()
        {
           return "Base";
        }
    }
    class Derived : Base
    { }

    class Program
    {
        static void Main()
        {

            Derived d = new Derived();

            Base b = d as Base;
            if (b != null)
            {
                Console.WriteLine(b.ToString());
            }

        }
    }
}
```




```
int j = 123;
            object boxed = j;
            object obj = new Object();
            Boolean chekJ = boxed is int; //true
            Boolean checkObj = obj is int; //false

            Console.WriteLine("boxed {0} System.ValueType",
                      boxed is ValueType ? "is" : "is not");
```

3.8. Полиморфизм. Виртуальные методы, свойства и индексаторы



Полиморфизм

- ключевой аспект объектно-ориентированного программирования
- способность к изменению функционала, унаследованного от базового класса

Виртуальные: методы, свойства, индексаторы

полиморфный интерфейс в базовом классе - набор членов класса, которые могут быть переопределены в классе-наследнике

```
virtual public void A_method() { }
```

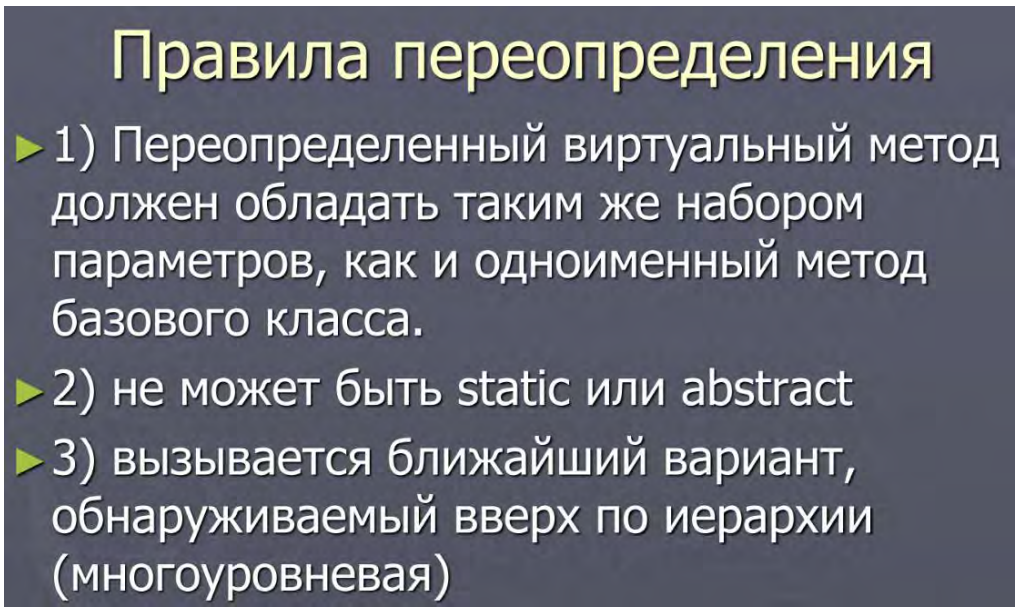
переопределение виртуального метода в производном классе:

```
override public void A_method() { }
```

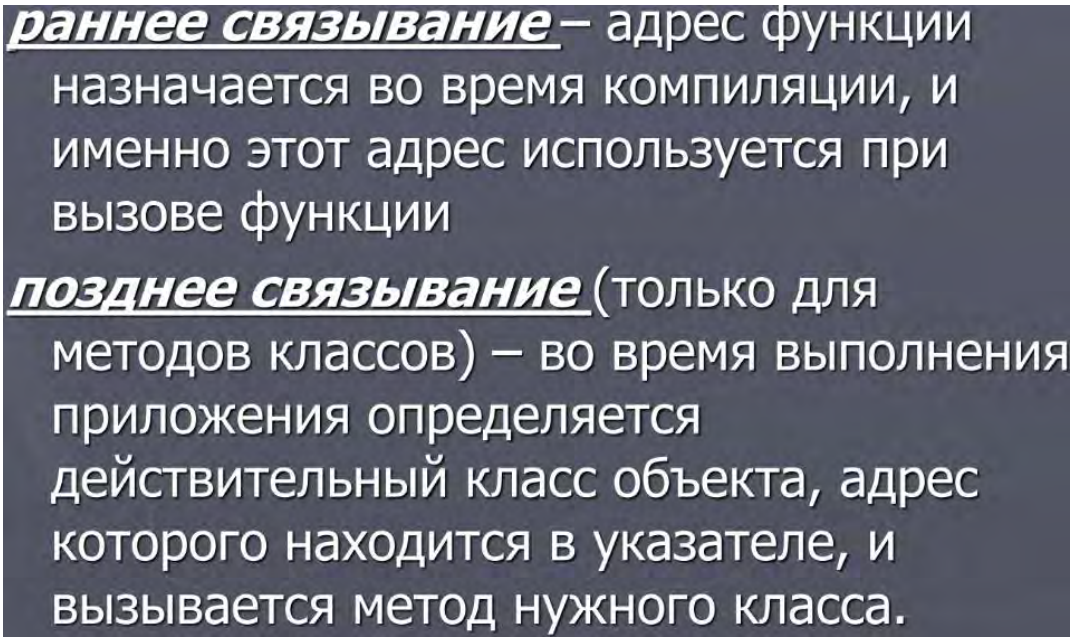
полиморфизм

потенциально виртуальный virtual ставить нельзя

```
public class Point
        {
            public int x = 10;
            public int y = 20;
           virtual public int Sum()  {   return x + y;  }
        }
public class ColorPoint : Point
        {
            public int color = 78;
           override public int Sum() {  return x * y * color; }
        }
 private static void Main(string[] args)
        {
            Point a12 = new Point();
            Console.WriteLine(a12.Sum()); //30

            ColorPoint ca100 = new ColorPoint();
           a12 = ca100;
            Console.WriteLine(a12.Sum());//15600
            //  вызов методов по типу объекта
        }
```

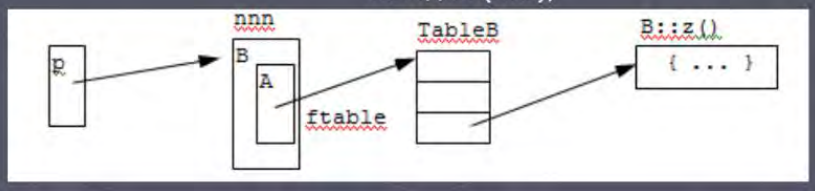


Понятие раннего и позднего связывания



Виртуальные функции предоставляют механизм **позднего** (**отложенного**) или **динамического связывания.**

таблица виртуальных методов (vtbl),



Тип связывания	Достоинства	Недостатки
Ранее	высокое быстродействие получаемых выполнимых программ	снижение гибкости программ
Позднее	высокая гибкость выполняемой программы, возможность реакции на события	относительно низкое быстродействие программы

Абстрактные классы и методы. Бесплодные классы

Абстрактные классы

- Служит только для порождения потомков - предоставляют базовый функционал для классов-наследников.
- Задает интерфейс для всей иерархии
- Может содержать и полностью определенные методы, переменные, конструкторы, свойства

- если класс имеет хотя бы одно абстрактное свойство или метод, то он должен быть определен как абстрактный.

```
public  class Person
        {
            public abstract void work();
        }
```

```
public abstract class Person
        {
            public abstract void work();
        }
        public class Employee : Person
        {
            public override void work()
            {
                //...
            }
        }
```

абстрактный метод
Определяет полиморфный интерфейс

производный класс обязан переопределить и реализовать все абстрактные методы и свойства, которые имеются в базовом абстрактном классе

Свойства abstract меодов

- 1) абстрактные методы автоматически виртуальные (virtual не ставится)
- 2) абстрактные методы не используется со static
- 3) объекты создавать нельзя

```
public abstract class Person
        {
            public abstract void work();
        }
private static void Main(string[] args)
        {
            Person anna = new Person(); //ошибка
      }
```

4) А.К. может быть параметром метода - Полиморфные методы

```
abstract class  Car  { }
 class Cargo : Car {  }
 class BigCar : Car { }
 class Maneger
 {
     public List<Car> createGarag (Car[] All);
 }
```

Бесплодные (запечатанные) классы

- класс, от которого наследовать запрещается

```
    abstract class AAA
{  public abstract void A_method();
}
sealed class A : AAA
{
 override public void A_method()  {}
}
class B : AAA
{
override public void A_method()  {}
}
```

Запрет переопределения методов

```
public class Point
        {
            public int x = 10;
            public int y = 20;
            public  virtual int Sum() { return x + y; }
        }
        public class ColorPoint : Point
        {
            public int color = 78;
            public sealed override int Sum() { return x * y*
color; }
        }
```

метод в незапечатанном классе является запечатанным

не сможем переопределить метод Sum в классе, унаследованном от ColorPoint.

4. Дополнительные конструкции программирования на C#

4.1. Структуры. Интерфейсы. Свойства интерфейсов

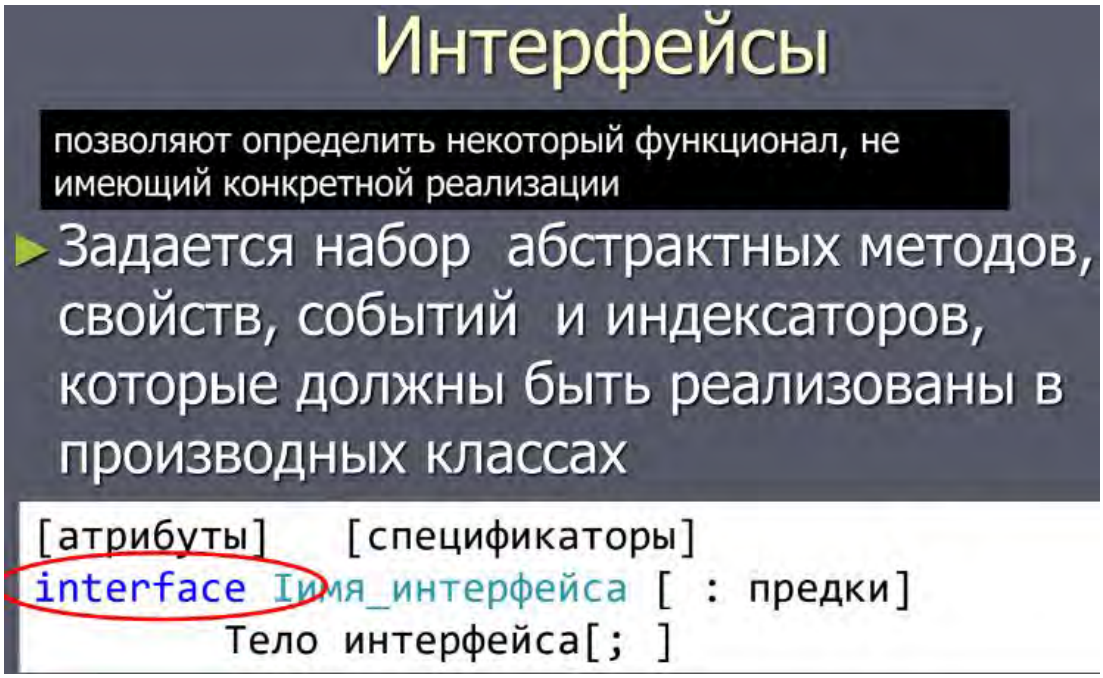
Структура – это составной объект (пользовательский тип данных), совокупность логически связанных данных различного типа, объединенных под одним идентификатором.

Структура относится к типу значения, класс – к ссылочному типу данных, т.е. структуры размещаются в стеке, а классы – в куче. Также структуры не поддерживают наследование.

Может содержать: структура этого же типа, указатель на такую же структуру, указатель на функцию, прототип функции, объединение, перечисление, реализовывать интерфейсы, значение NULL, объявление конструкторов с параметрами.

Не может: инициализированные при объявлении поля (если не константы и не статические), не может объявлять конструктор (деструктор) по умолчанию.

Перечисление – тип данных, чьё множество значений представляет собой ограниченный список идентификаторов.



Свойства	Интерфейс
не может содержать	константы, поля, операции, конструкторы, деструкторы, типы, любые статические элементы
может содержать	абстрактные методы, обобщения свойства и индексаторы, а также события
Доступность методов	**public по умолчанию (не указывается)** (при переопределении тоже public)
наследуются	C# поддерживается одиночное наследование для классов и множественное — для интерфейсов (при реализации интерфейса нужно обеспечить точное совпадение) Сначала всегда указывается имя базового класса, затем указывается интерфейс
Расширение интерфейса	Интерфейс наследуется интерфейсом
Имена	с прописной буквы I

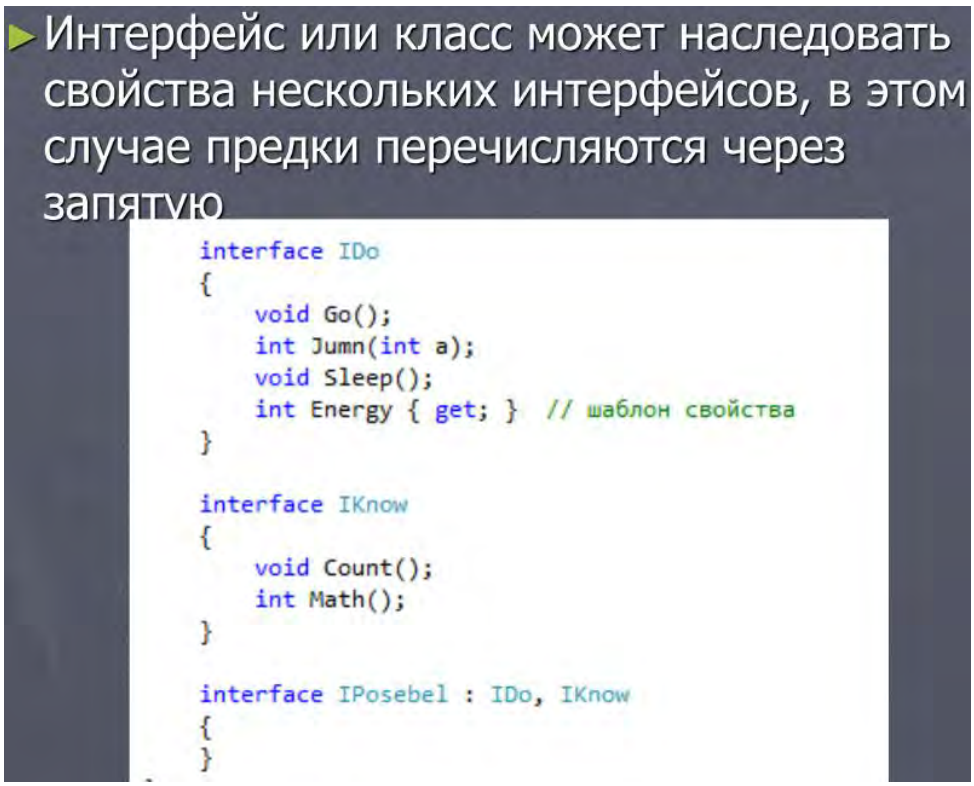
МОГУТ : абстрактные методы, обобщения свойства и индексаторы, а также события

```
interface IDo
{
    void Go();
    int Jumn(int a);
    void Sleep();
    int Energy { get; }  // шаблон свойства
}
```

По умолчанию public, реализации нет

Назначение :

задания общих характеристик объектов различных иерархий – навязывание контракта



Явная и неявная реализация интерфейсов

Критерии	**Неявная (implicit) реализация**	**Явная (explicit) реализация**

Базовый синтаксис	`interface IDo { void Sleep(); }`
	`public class ImplicitDo : IDo`
	`{ public void Sleep() { } }`
		`interface IDo { void Sleep(); }`
		`public class ExplicitDo : IDo`
		`{ void IDo.Sleep() { } }`

Видимость	всегда является открытой (**public**)
	можно обращаться напрямую.
	`var imp = new ImplicitDo();`
	`imp.Sleep();`
		всегда закрыта (**private**)
		чтобы получить доступ необходимо приводить инстанцию класса к интерфейсу (upcast to interface).
		`var exp = new ExplicitDo();`
		`((IDo)exp).Sleep();`

Полиморфизм	может быть виртуальной (**virtual**), что позволяет переписывать эту реализацию в классах-потомках.	всегда **статична** не может быть переопределена (override) или перекрыта (new) в классах-потомках.

Работа с объектами через интерфейсы (преобразования)

- Проверка поддержки данного интерфейса

```
static void Act(object A)
{
    if (A is IDo)
    {
        IDo Actor = (IDo)A;
        Actor.Sleep();
    }
}
```

```
static void Act(object A)
{
    if (A is IDo)
    {
        IDo Actor = (IDo)A;
        Actor.Sleep();
    }

    INotDo Actor2 = A as INotDo;
    if (Actor2 != null) Actor2.Sleep();

}
```

Переменной ссылки на интерфейс доступны только методы, объявленные в ее интерфейсе.

- Класс наследует все методы своего предка (интерфейсы). Он может переопределить (new), но обращаться к ним - через объект класса. Если использовать для обращения ссылку на интерфейс, вызывается не переопределенная версия

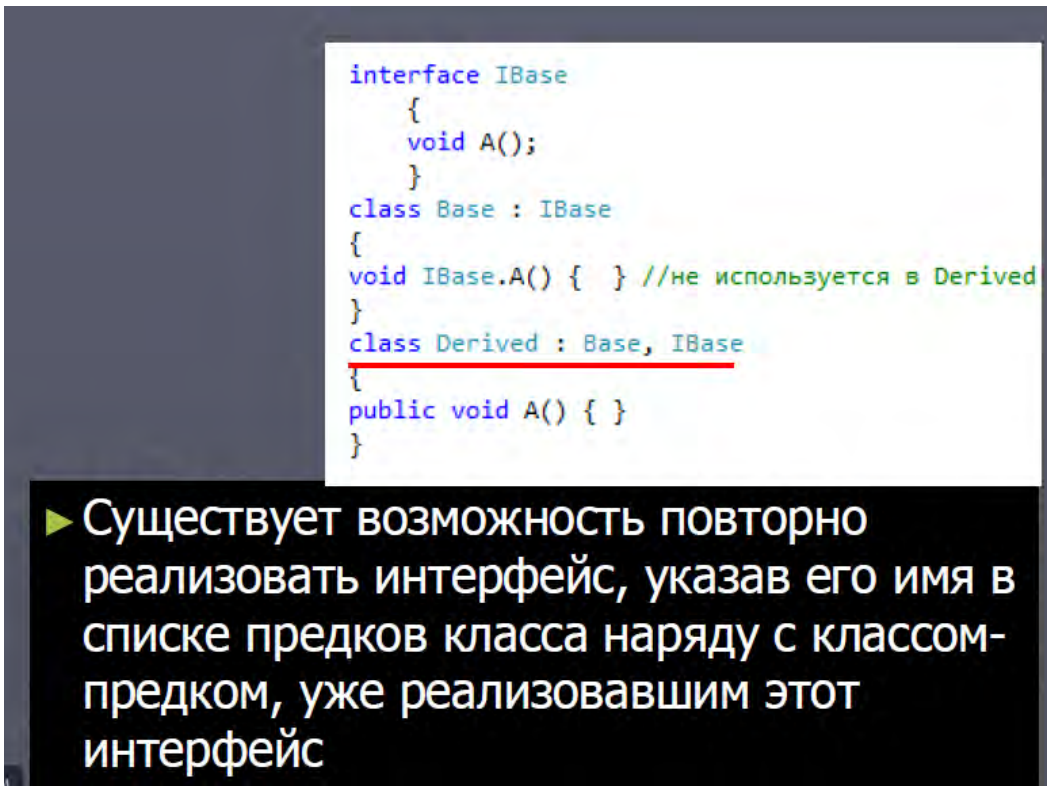
```
interface IBase
{
void A() ;
}
class Base : IBase
{
public void A() {  }
}
class Derived: Base
{
new public void A() { }
}

static void Main()
{
Derived d = new Derived ();
d.A(); // вызывается Derived.AO;
IBase id = d;
id.A(); // вызывается Base.AO;
```

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

```
interface IBase
{
void A();
}
class Base : IBase
{
public virtual void A() { }
}
class Derived: Base
{
public override void A() {  }
}

static void Main()
{

Derived d = new Derived();
d.A(); // вызывается Derived.AO;
IBase id = d;
id.A(); // вызывается Derived.AO;
```

- Метод интерфейса, реализованный явным указанием имени, объявлять виртуальным запрещается. При необходимости →переопределить в потомках его поведение:

```
interface IBase
{
void A();
}
class Base : IBase
{
void IBase.A() { A1(); }
protected virtual void A1() { }
}
class Derived: Base
{
protected override void A1() {}
}
```



4.2. Ковариантность и контравариантность интерфейсов

Понятия ковариантности и контравариантности связаны с возможностью использовать в приложении вместо некоторого типа другой тип, который находится ниже или выше в иерархии наследования.

Имеется три возможных варианта поведения:

- **Ковариантность**: позволяет использовать более конкретный тип, чем заданный изначально
- **Контравариантность**: позволяет использовать более универсальный тип, чем заданный изначально
- **Инвариантность**: позволяет использовать только заданный тип

Начиная с .NET 4.0 в C# была добавлена возможность создания ковариантных и контравариантных обобщенных интерфейсов. Это функциональность повышает гибкость при использовании обобщенных интерфейсов в программе. По умолчанию все обобщенные интерфейсы, например, IAccout<T> являются инвариантными.

Обобщенные интерфейсы могут быть ковариантными, если к универсальному параметру применяется ключевое слово **out**. Например:

```
1   class Account
2   {
3       static Random rnd = new Random();
4
5       public void DoTransfer()
6       {
```

```
7           int sum = rnd.Next(10, 120);
8           Console.WriteLine("Клиент положил на счет {0} долларов", sum);
9       }
10  }
11  class DepositAccount : Account
12  {
13  }
14
15  interface IBank<out T>  where T : Account
16  {
17      T DoOperation();
18  }
19
20  class Bank: IBank<DepositAccount>
21  {
22      public DepositAccount DoOperation()
23      {
24          DepositAccount acc = new DepositAccount();
25          acc.DoTransfer();
26          return acc;
27      }
28  }
```

Определение интерфейса IBank<out T> указывает, что данный интерфейс будет ковариантным. В программе мы могли бы использовать его так:

```
1   static void Main(string[] args)
2   {
3       IBank<DepositAccount> depositBank = new Bank();
4       depositBank.DoOperation();
5
6       IBank<Account> ordinaryBank = depositBank;
7       ordinaryBank.DoOperation();
8
9       Console.ReadLine();
10  }
```

То есть мы можем присвоить более общему типу IBank<Account> объект более конкретного типа IBank<DepositAccount>.

В то же время если бы мы не использовали ключевое слово out:

```
1   interface IBank<out T>  where T : Account
```

то Visual Studio ометила бы строку IBank<Account> ordinaryBank = depositBank; как ошибочную. Поскольку в этом случае невозможно было бы привести объект IBank<DepositAccount> к типу IBank<Account>

При создании ковариантного интерфейса надо учитывать, что универсальный параметр может использоваться только в качестве типа значения, возвращаемого

методами интерфейса. Но не может использоваться в качестве типа аргументов метода или ограничения методов интерфейса.

Для создания контравариантного интерфейса надо использовать ключевое слово **in**. Для его применения изменим интерфейс IBank и класс Bank:

```
interface IBank<in T>  where T : Account
{
    void DoOperation(T account);
}

class Bank<T>: IBank<T> where T : Account
{
    public void DoOperation(T account)
    {
        account.DoTransfer();
    }
}
```

Классы Account и DepositAccount остаются без изменений. Теперь применим интерфейс IBank и классы в программе:

```
static void Main(string[] args)
{
    Account account = new Account();
    IBank<Account> ordinaryBank = new Bank<Account>();
    ordinaryBank.DoOperation(account);

    DepositAccount depositAcc = new DepositAccount();
    IBank<DepositAccount> depositBank = ordinaryBank;
    depositBank.DoOperation(depositAcc);

    Console.ReadLine();
}
```

Так как интерфейс IBank использует универсальный параметр с ключевым словом in, то он является контравариантным, поэтому в коде мы можем объект Bank<Account> привести к типу IBank<DepositAccount>:

```
IBank<DepositAccount> depositBank = ordinaryBank;
```

То есть объект интерфейса с более универсальным типом приводится к объекту интерфейса с более конкретным типом.

При создании контрвариантного интерфейса надо учитывать, что универсальный параметр контрвариантного типа может применяться только к аргументам метода, но не может применяться к аргументам, используемым в качестве возвращаемых типов.

4.3. Стандартные интерфейсы .NET, назначение

Для среды .NET Framework определено немало стандартных интерфейсов, которыми можно пользоваться в программах на C#. Так, в интерфейсе System.IComparable определен метод CompareTo (), применяемый для сравнения объектов, когда требуется соблюдать отношение порядка. Стандартные интерфейсы являются также важной частью классов коллекций, предоставляющих различные средства, в том числе стеки и очереди, для хранения целых групп объектов. Так, в интерфейсе System.Collections. iCollection определяются функции для всей коллекции, а в интерфейсе System.Collections.IEnumerator — способ последовательного обращения к элементам коллекции.

//

В библиотеке .NET определœено множество стандартных интерфейсов, задающих желаемое поведение объектов. К примеру, интерфейс IComparable задает метод сравнения объектов по принципу больше или меньше, что позволяет выполнять их сортировку. Реализация интерфейсов IEnumerable и IEnumerator дает возможность просматривать содержимое объекта с помощью конструкции foreach, а реализация интерфейса ICloneable – клонировать объекты.

Стандартные интерфейсы поддерживаются многими стандартными классами библиотеки. К примеру, 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.

Сравнение объектов (интерфейс IComparable)

Интерфейс IComparable содержит всœего один метод **CompareTo(),**возвращающий результат сравнения двух объектов – текущего и переданного ему в качеству параметра:

```
interface IComparable
{
int CompareTo(object obj)
}
```

Метод должен возвращать:

- o 0, если текущий объект и параметр равны
- o отрицательное число, если текущий объект меньше параметра
- o положительное число, если текущий объект больше параметра

Сортировка объектов по различным критериям (интерфейс IComparer)

Данный интерфейс определœен в пространстве имен System.Collections. Он также содержит один метод**Compare(),**возвращающий результат сравнения двух объектов, переданных ему в качестве параметров.

```
interface IComparer
{
```

int Compare(object obj1,object obj2)
}

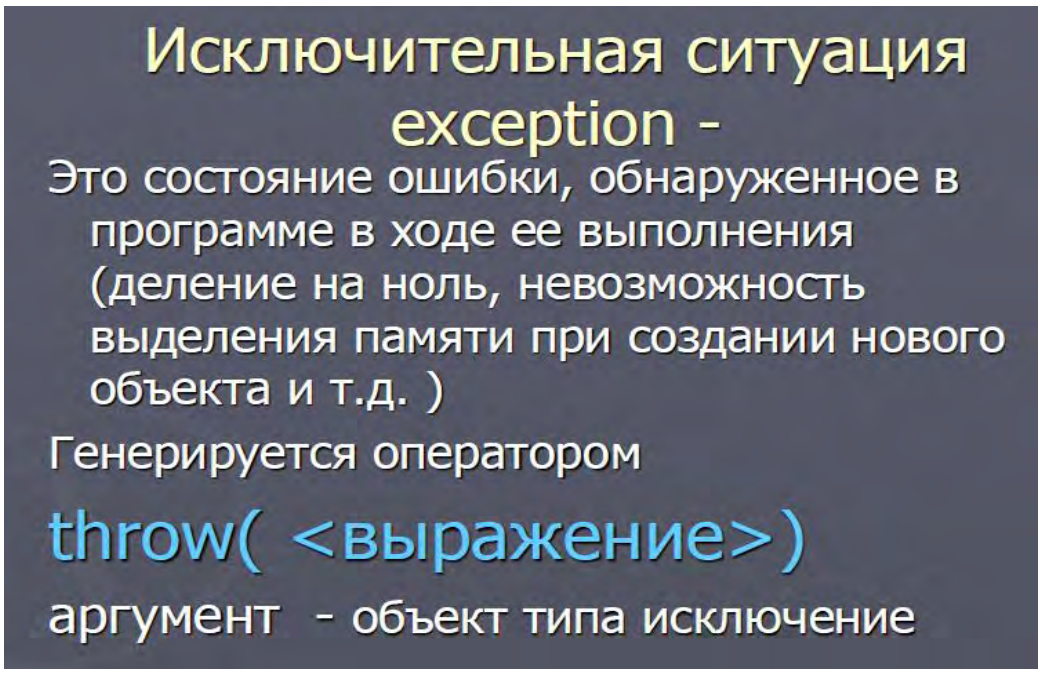
Принцип применения этого интерфейса состоит в том, что для каждого критерия сортировки объектов описывается небольшой вспомогательный класс, реализующий данный интерфейс. Объект этого класса передается в стандартный метод сортировки массива в качестве второго аргумента.

Клонирование объектов (интерфейс ICloneable)

Клонирование - создание копии объекта. Копия объекта принято называть **клоном.** При присваивании одного экземпляра другому копируется ссылка, а не сам объект. В случае если крайне важно скопировать в другую область памяти поля объекта, можно воспользоваться методом **MemberwiseClone(),** который любой объект наследует от класса object. При этом объекты, на которые указывают поля объекта, в свою очередь являющиеся ссылками, не копируются. Это принято называть ***поверхностным клонированием***. Важно заметить, что для создания полностью независимых объектов крайне важно ***глубокое копирование***, когда в памяти создается дубликат всего дерева объектов, то есть объектов, на которые ссылаются поля объекта, поля полей, и т.д. Алгоритм глубокого копирования сложен, требует рекурсивного обхода всех ссылок объекта и отслеживания циклических зависимостей.

Объект, имеющий собственные алгоритмы клонирования, должен объявляться как производный интерфейса ICloneable и переопределять его единственный метод Clone().

4.4. Исключительные ситуации. Генерация и обработка



Генерация исключения

```
if (b==0)
        throw new Exception("Zero devision");
else
          c = a / b;
```

ГЕНЕРИРОВАНИЕ И РАСПОЗНАВАНИЕ ИСКЛЮЧЕНИЙ

- try – контролируемый блок
- throw - генерация искл. ситуации внутри try
- catch – обработчики исключений, идут за try
- finally - код, очищающий ресурсы и др. действия (выполняется всегда)

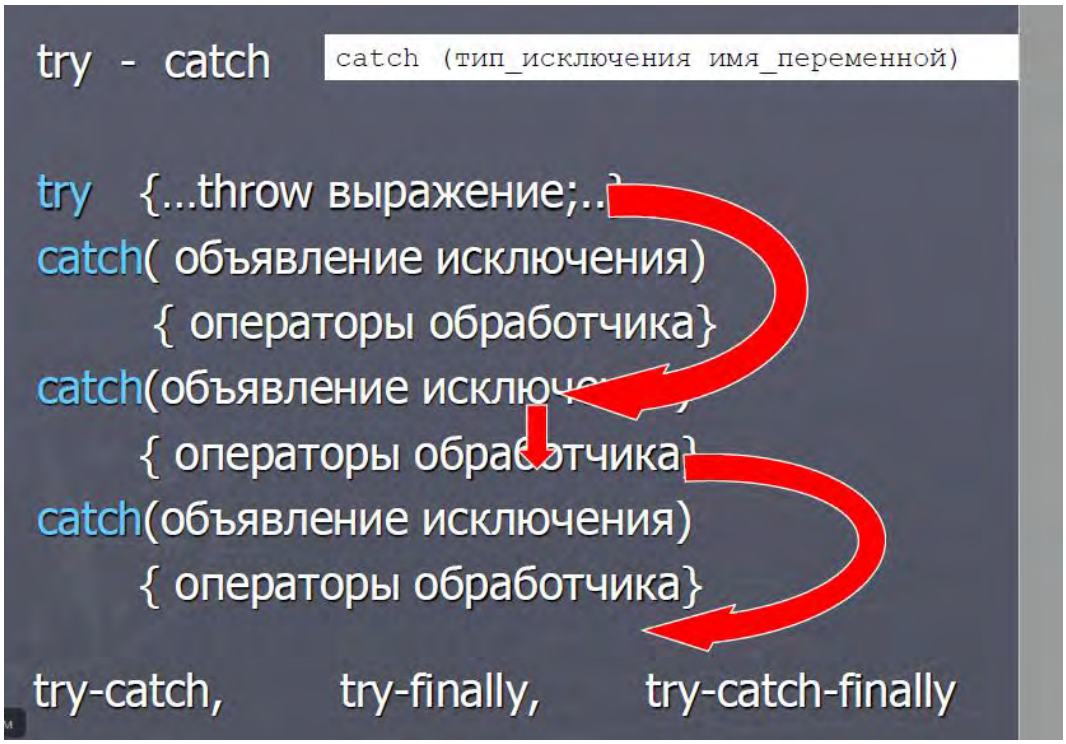
```
try
            {
        // Блок кода, проверяемый на наличие ошибок.
            }

            catch (ExcepType1 exOb)
            {
          // Обработчик исключения типа ExcepType1.
            }
            catch (ExcepType2 exOb)
            {
         // Обработчик исключения типа ExcepType2.
            }
```

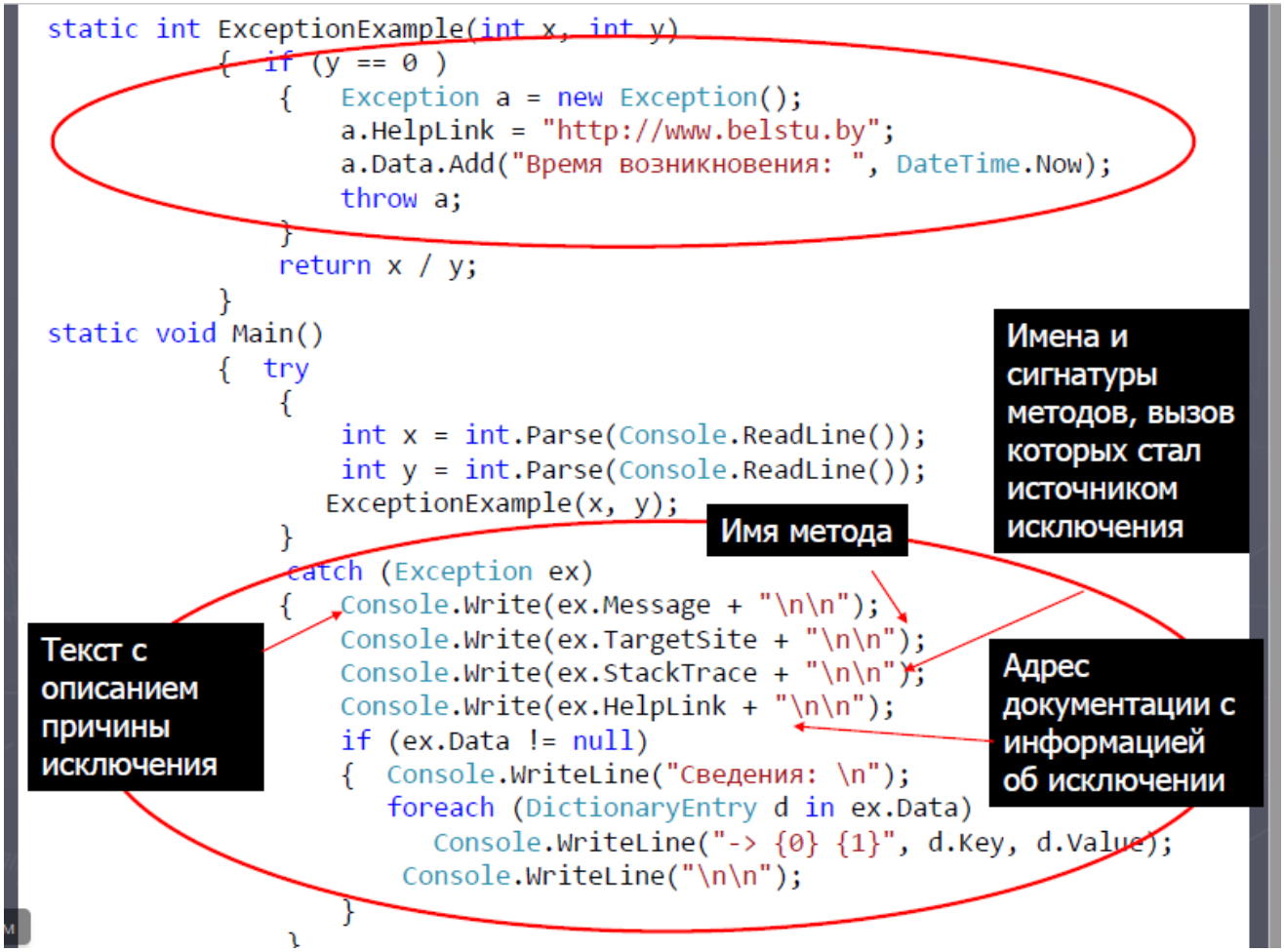
```
FileStream fs = null;
           try
           {
               fs = new FileStream(pathname, FileMode.Open);
               // Обработка данных
           }
           catch (IOException)
           {
               // Код восстановления
           }
           finally
           {
               // Файл  следует закрыть
               if (fs != null) fs.Close();
           }
```

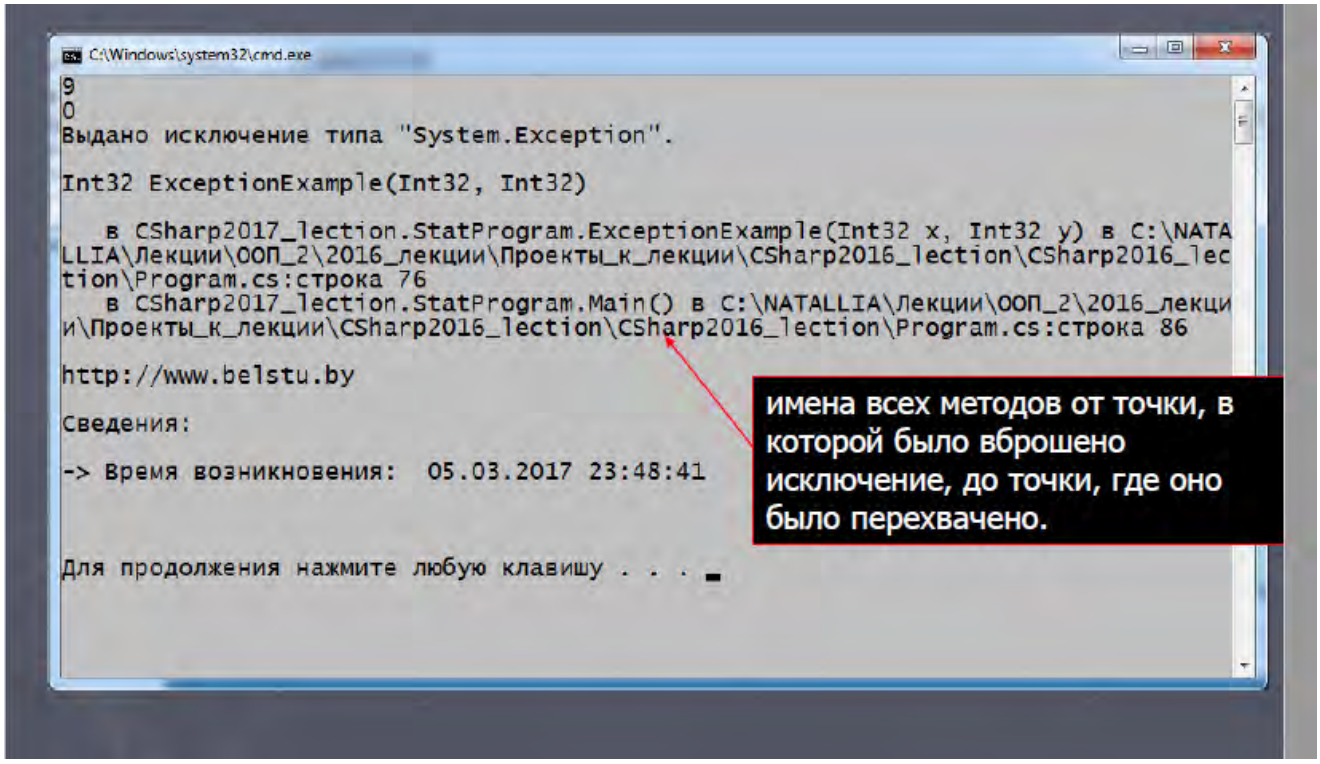
по возможности коротким

только один блок finally

источник исключения → catch или finally, CLR продолжает работу, теряется информация о первом исключении, вброшенном в блоке try. Скорее всего новое исключение останется необработанным → CLR завершает процесс




static int ExceptionExample(int x, int y)
{ if (y == 0)
{ Exception a = new Exception();
a.HelpLink = "http://www.belstu.by";
a.Data.Add("Время возникновения: ", DateTime.Now);
throw a;
}
return x / y;
}
static void Main()
{ try
{
int x = int.Parse(Console.ReadLine());
int y = int.Parse(Console.ReadLine());
ExceptionExample(x, y);
}
catch (Exception ex)
{ Console.Write(ex.Message + "\n\n");
Console.Write(ex.TargetSite + "\n\n");
Console.Write(ex.StackTrace + "\n\n");
Console.Write(ex.HelpLink + "\n\n");
if (ex.Data != null)
{ Console.WriteLine("Сведения: \n");
foreach (DictionaryEntry d in ex.Data)
Console.WriteLine("-> {0} {1}", d.Key, d.Value);
Console.WriteLine("\n\n");
}
Имена и сигнатуры методов, вызов которых стал источником исключения
Имя метода
Текст с описанием причины исключения
Адрес документации с информацией об исключении


C:\Windows\system32\cmd.exe
9
0
Выдано исключение типа "System.Exception".
Int32 ExceptionExample(Int32, Int32)
в CSharp2017_lection.StatProgram.ExceptionExample(Int32 x, Int32 y) в C:\NATALLIA\Лекции\ООП_2\2016_лекции\Проекты_к_лекции\CSharp2016_lection\CSharp2016_lection\Program.cs:строка 76
в CSharp2017_lection.StatProgram.Main() в C:\NATALLIA\Лекции\ООП_2\2016_лекции\Проекты_к_лекции\CSharp2016_lection\CSharp2016_lection\Program.cs:строка 86
http://www.belstu.by
Сведения:
-> Время возникновения: 05.03.2017 23:48:41
Для продолжения нажмите любую клавишу . . .
имена всех методов от точки, в которой было вброшено исключение, до точки, где оно было перехвачено.

```
catch (OverflowException ex)
                {
                  Console.Write("Данное число не входит в диапазон" +
                      ex.Message);
                }
catch (DivideByZeroException ex)
                {
                 Console.WriteLine("Деление на ноль "+ ex.Message);
                }

 catch (IndexOutOfRangeException )
                {
                    Console.WriteLine("Индекс выходит за пределы\n");
                }
 catch //универсальный обработчик
               {}
```

предпочтительней

```
catch (Exception ex) // это общий обработчик
исключений
                {
                    //...
                }
```

Генерация исключений

```
try
                {
                    if (i > 255)
                      // Генерируем исключение
                       throw new OverflowException();
                }
```

Должен существовать catch

Повторная генерация исключения

создание нового объекта посредством повторного использования старого с помощью оператора throw без параметров

```
static void MathOp(int x, int y)
        {
            try
            {
                int result = x / y;
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Деление на ноль!");
                throw;
            }
        }
```

при повторном вызове перехваченного исключения с помощью ключевого слова throw удаления из стека информации о начальной точке не происходит

4.5. Обобщения. Свойства обобщений

```
public class SuperArray<T>
        {
            T[] s;
        }
```

Тип – любой идентификатор **универсальный параметр,** так как вместо него можно подставить любой тип

```
SuperArray<int> iArr = new SuperArray<int>();
SuperArray <Stack<int>> stArr=new SuperArray<Stack<int>>();
SuperArray<Person> perArr = new SuperArray<Person>();
```

Обобщения (generics)

Механизм многократного использования алгоритмов

- *Обобщение* - параметризированный тип
- Определены для CLR – поддержка разных языков
- Открытый тип ⟶ закрытый тип

  Tlist<T> Tlist<int>

  экземпляры a,b,c

В CLR запрещено конструирование экземпляров открытых типов

System.Collections.Generic

свойства

- 1)Обобщенный тип может содержать другой обобщенный тип

```
public class B<T>
{
    private A<T> one;
    private A<int> two;
}
```

- 2) Обобщенные типы перегружаются на основе количества параметров (арность)

```
public class A { }
public class A<T> { }
public class A<T, U> { }
```

- 3) Обобщенными могут быть классы, структуры , интерфейсы, делегаты, методы

public void Method <R> (A<R> iA, B<R,T> iB)

```
public class Animal
{

    public void Move<T>(T distance)
    { }
}

static void Main(){
    Animal Носорог = new Animal();
    Носорог.Move(1);
    Носорог.Move("аршин");
    Носорог.Move<double>(45.6);

}
```

логическое выведение типов (type inference) → используется тип данных переменной, а не фактический тип объекта, на который ссылается

- 4) Могут содержать статические типы
- 5) Доступность конструируемых типов определяется на основе пересечения доступности обобщённого типа и типа в списке аргументов

```
public class lab2
{
    private class People { }
    public class Generic<T> { }
    private Generic<People> one;
    public Generic<People> two;  // ошибка
}
```

- 5) могут использовать несколько универсальных параметров одновременно

```
class Transaction<U, V>
        {
            public U FromAccount { get; set; }
// с какого счета перевод
            public U ToAccount { get; set; }
// на какой счет перевод
            public V Code { get; set; }
// код операции
            public int Sum { get; set; }
// сумма перевода
        }
```

- 6) поддерживает механизм ограничений

Концепция ограничений обобщений

В CLR существует механизм ограничений (constraints) - инструмент определения обобщенного типа с указанием допустимых для него аргументов типа

Ограничение на интерфейс

```
private static T Min<T>(T o1, T o2)
        {
            if (o1.CompareTo(o2) < 0) return o1;
            return o2;
        }
```

'T' не содержит определения для 'CompareTo' и не удалось найти метод расширения 'CompareTo', принимающий тип 'T' в качестве первого аргумент

Ограничение сужает перечень типов, которые можно передать в обобщенном аргументе, и расширяет возможности по работе с этими типами.

```
public static T Min<T>(T o1, T o2) where T : IComparable<T>
        {
            if (o1.CompareTo(o2) < 0) return o1;
            return o2;
        }
```

указанный в T тип должен реализовывать обобщенный интерфейс IComparable того же типа (T).

Ограничение на базовый класс

```
public class Figure { }
public class Rectengle : Figure { }
public class Computer { }

public class LinkedSet<U> where U : Figure { }

static void Main(){
    LinkedSet<Rectengle> бусы = new LinkedSet<Rectengle>();
    LinkedSet<Computer> компкласс = new LinkedSet<Computer>();
```

Ограничение ссылочного типа

```
public class Computer { }

public class LinkedSet<U> where U : class { }

static void Main()
{
    LinkedSet<Computer> компкласс = new LinkedSet<Computer>();
    LinkedSet<int> рядфурье = new LinkedSet<int>();
}
```

Этому ограничению удовлетворяют все типы-классы, типы-интерфейсы, типы-делегаты и типы-массивы

```
internal sealed class Mama<T> where T : class
        {
            public void M()
            {
                T temp = null;// Допустимо
            }
        }
```

При отсутствии у T ограничений код бы не скомпилировался

Ограничение типа значения

гарантирует компилятору, что указанный аргумент типа будет иметь значимый тип

Но значимые типы с поддержкой null (System.Nullable<T>) не подходят под это ограничение

```
internal sealed class Mama<T> where T : struct
        {
            public static T GiveSomething()
            {

// Допускается, потому что у каждого значимого типа неявно
// есть открытый конструктор без параметров
                return new T();
            }
        }
```

Ограничение на конструктор

гарантирует компилятору, что указанный аргумент-тип будет иметь неабстрактный тип, имеющий открытый конструктор без параметров

```
public class Computer {
    public Computer(int h){}
  }

public class LinkedSet<U> where U : new() { }

static void Main()
{

    LinkedSet<Computer> компкласс = new LinkedSet<Computer>();
    LinkedSet<int> рядфурье = new LinkedSet<int>();

}
```

Требование предоставить конструктор без параметров

Ограничение на конструктор

```
class SomeClass { };
       class TList<T> where T :new()
       {
  // Следующий код доступен благодаря ограничению на конструктор
           T obj = new T();

       }
      static class  Run{
           public static void Main()
           {
               TList<SomeClass> Spisok = new TList<SomeClass>();
           }
           }
   }
```

4.6. Делегаты. Определение, назначение. События

Свойства

- Тип данных
- Наследовать от делегата нельзя
- Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса (в любом месте, где может быть определен класс)

```
public delegate void D(int i);

class GGG
{
    public delegate void GGGD(int i);
}
```

- может вызывать только такие методы, у которых тип возвращаемого значения и список параметров совпадают
- Может быть статический метод класса
- Имеет тот же синтаксис, что и вызов метод
- Если делегат хранит ссылки на несколько методов, они вызываются последовательно в том порядке, в котором были добавлены в делегат (цепочки (chaining))

Назначение делегатов

- 1)возможности определять вызываемый метод не при компиляции, а динамически во время выполнения программы;
- 2)обеспечения связи между объектами по типу «источник — наблюдатель»;
- 3)создания универсальных методов, в которые можно передавать другие методы;
- 4)поддержки механизма обратных вызовов.

Пример

```
delegate string strMod(string stx);

class DelegateTest
{
    static string replaceSpaces(string a) { Console.WriteLine("Замена"); return a; }
    static string removeSpaces(string a) { Console.WriteLine("Удаление"); return a; }
    static string reverse(string a) { Console.WriteLine("Реверс"); return a; }

    public static void Main()
    {
      public static void Main()
      {
          strMod strOp = replaceSpaces;
          string str; str = strOp("ЭТО простой тест.");
          strOp = DelegateTest.removeSpaces;
          str = strOp("Это простой тест.");
      }
}
```

Операции над делегатами

- можно *сравнивать на равенство и неравенство* (не содержат ссылок или если ссылки на одни и те же методы в одном и том же порядке)
- *выполнять операции* простого и составного *присваивания* (один тип д.и.)

```
del += HelloI; // добавляем делегат
del -= HelloI; // удаляем делегат
```

- является неизменяемым типом данных, поэтому при любом изменении создается новый экземпляр, а старый впоследствии удаляется сборщиком мусора.

Групповая адресация

Создание списка, или цепочки вызовов, для методов, которые вызываются автоматически при обращении к делегату

```
delegate void OperationWithArray(ref int[] arr);
      public class ArrayOper
      {  public static void
          // Сортировка масс
          public static void
          // Заменяем отрица
          public static void
      }
      class Program
      { static void Main()
          {  int[] someArr = new int[] { 20, -456, 103, 0, -634, 92 };
              // Структуируем делегаты
              OperationWithArray DelegAll;      // Групповая адресация
              DelegAll = ArrayOperation.WriteArray;
              DelegAll += ArrayOperation.IncSort;
              DelegAll += ArrayOperation.WriteArray;
              DelegAll += ArrayOperation.NegatArr;
              DelegAll += ArrayOperation.WriteArray;
              // Выполняем делегат
              DelegAll(ref someArr);
              Console.ReadLine();
```

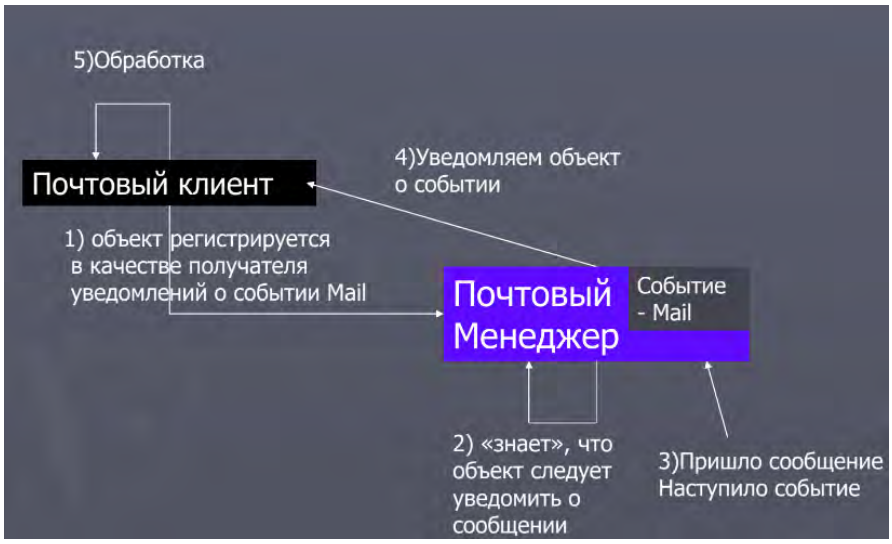
```
C:\Windows\system32\cmd.exe
Исходный массив:
20      -456    103     0       -634    92
Исходный массив:
-634    -456    0       20      92      103
Исходный массив:
0       0       0       20      92      103
```

События

События - элемент класса , позволяющий ему посылать другим объектам уведомления об изменении своего состояния.

Модель “публикация – подписка” или паттерн “наблюдатель” , класс , являющийся отправителем , сообщения , публикует события, которые он может

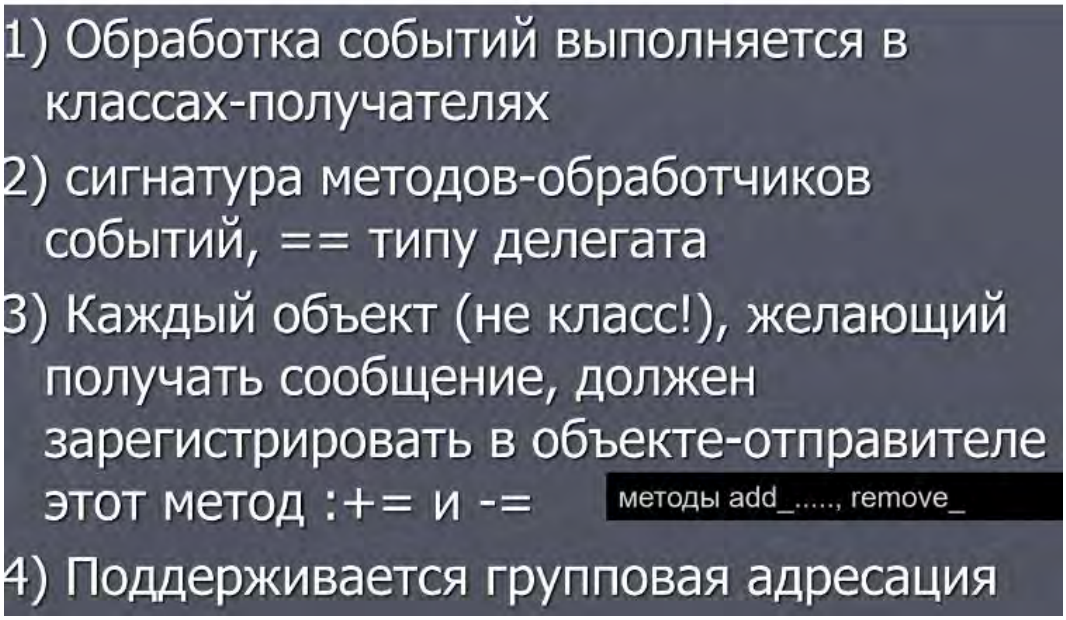
инициировать , а другие классы , являющиеся получателями сообщения , подписываются на получение этих событий.



События построены на основе делегатов: с помощью делегатов вызываются методы-обработчики событий. Поэтому создание события в классе состоит из следующих частей:

- Описание делегата, задающего сигнатуру обработчиков событий
- Описание события
- Описание метода (методов), инициирующих событие

```
{
    public event Doing Oppa; // объявление события
}
```



4.7. Классы для работы с файловой системой

Фреймворк .NET предоставляет большие возможности по управлению и манипуляции файлами и каталогами, которые по большей части сосредоточены в пространстве имен System.IO. Классы, расположенные в этом пространстве имен (такие как Stream, StreamWriter, FileStream и др.), позволяют управлять файловым вводом-выводом.

Работа с дисками

Работу с файловой системой начнем с самого верхнего уровня - дисков. Для представления диска в пространстве имен System.IO имеется класс DriveInfo.

Этот класс имеет статический метод GetDrives, который возвращает имена всех логических дисков компьютера. Также он предоставляет ряд полезных свойств:

- ✓ AvailableFreeSpace: указывает на объем доступного свободного места на диске в байтах
- ✓ DriveFormat: получает имя файловой системы
- ✓ DriveType: представляет тип диска
- ✓ IsReady: готов ли диск (например, DVD-диск может быть не вставлен в дисковод)
- ✓ Name: получает имя диска
- ✓ TotalFreeSpace: получает общий объем свободного места на диске в байтах
- ✓ TotalSize: общий размер диска в байтах
- ✓ VolumeLabel: получает или устанавливает метку тома

Получим имена и свойства всех дисков на компьютере:

```
using System;
using System.Collections.Generic;
using System.IO;
namespace FileApp
{
   class Program
   {
      static void Main(string[] args)
      {
         DriveInfo[] drives = DriveInfo.GetDrives();
         foreach (DriveInfo drive in drives)
         {
            Console.WriteLine("Название: {0}", drive.Name);
            Console.WriteLine("Тип: {0}", drive.DriveType);
            if (drive.IsReady)
            {
               Console.WriteLine("Объем диска: {0}", drive.TotalSize);
               Console.WriteLine("Свободное пространство: {0}",
drive.TotalFreeSpace);
```

```
                Console.WriteLine("Метка: {0}", drive.VolumeLabel);
            }
            Console.WriteLine();
        }
        Console.ReadLine();
    }
  }
}
```

Работа с каталогами

Для работы с каталогами в пространстве имен System.IO предназначены сразу два класса: Directory и DirectoryInfo.

Класс Directory

Класс Directory предоставляет ряд статических методов для управления каталогами. Некоторые из этих методов:

- ✓ CreateDirectory(path): создает каталог по указанному пути path
- ✓ Delete(path): удаляет каталог по указанному пути path
- ✓ Exists(path): определяет, существует ли каталог по указанному пути path. Если существует, возвращается true, если не существует, то false
- ✓ GetDirectories(path): получает список каталогов в каталоге path
- ✓ GetFiles(path): получает список файлов в каталоге path
- ✓ Move(sourceDirName, destDirName): перемещает каталог
- ✓ GetParent(path): получение родительского каталога

Класс DirectoryInfo

Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory. Некоторые из его свойств и методов:

- ✓ Create(): создает каталог
- ✓ CreateSubdirectory(path): создает подкаталог по указанному пути path
- ✓ Delete(): удаляет каталог
- ✓ Свойство Exists: определяет, существует ли каталог
- ✓ GetDirectories(): получает список каталогов
- ✓ GetFiles(): получает список файлов
- ✓ MoveTo(destDirName): перемещает каталог
- ✓ Свойство Parent: получение родительского каталога
- ✓ Свойство Root: получение корневого каталога

Посмотрим на примерах применение этих классов

Получение списка файлов и подкаталогов

```
string dirName = "C:\\";
if (Directory.Exists(dirName))
{
    Console.WriteLine("Подкаталоги:");
    string[] dirs = Directory.GetDirectories(dirName);
    foreach (string s in dirs)
    {
```

```
        Console.WriteLine(s);
    }
    Console.WriteLine();
    Console.WriteLine("Файлы:");
    string[] files = Directory.GetFiles(dirName);
    foreach (string s in files)
    {
        Console.WriteLine(s);
    }
}
```

Обратите внимание на использование слешей в именах файлов. Либо мы используем двойной слеш: "C:\\", либо ординарный, но тогда перед всем путем ставим знак @: @"C:\Program Files"

Создание каталога

```
string path = @"C:\SomeDir";
string subpath = @"program\avalon";
DirectoryInfo dirInfo = new DirectoryInfo(path);
if (!dirInfo.Exists)
{
    dirInfo.Create();
}
dirInfo.CreateSubdirectory(subpath);
```

Вначале проверяем, а нету ли такой директории, так как если она существует, то ее создать будет нельзя, и приложение выбросит ошибку. В итоге у нас получится следующий путь: "C:\SomeDir\program\avalon"

Получение информации о каталоге

```
string dirName = "C:\\Program Files";
DirectoryInfo dirInfo = new DirectoryInfo(dirName);
Console.WriteLine("Название каталога: {0}", dirInfo.Name);
Console.WriteLine("Полное название каталога: {0}", dirInfo.FullName);
Console.WriteLine("Время создания каталога: {0}", dirInfo.CreationTime);
Console.WriteLine("Корневой каталог: {0}", dirInfo.Root);
```

Удаление каталога

Если мы просто применим метод Delete к непустой папке, в которой есть какие-нибудь файлы или подкаталоги, то приложение нам выбросит ошибку. Поэтому нам надо передать в метод Delete дополнительный параметр булевого типа, который укажет, что папку надо удалять со всем содержимым:

```
string dirName = @"C:\SomeFolder";
try
{
    DirectoryInfo dirInfo = new DirectoryInfo(dirName);
    dirInfo.Delete(true);
}
catch (Exception ex)
```

```
{
    Console.WriteLine(ex.Message);
}
```

Или так:

```
string dirName = @"C:\SomeFolder";
Directory.Delete(dirName, true);
```

Перемещение каталога

```
string oldPath = @"C:\SomeFolder";
string newPath = @"C:\SomeDir";
DirectoryInfo dirInfo = new DirectoryInfo(oldPath);
if (dirInfo.Exists && Directory.Exists(newPath) == false)
{
    dirInfo.MoveTo(newPath);
}
```

При перемещении надо учитывать, что новый каталог, в который мы хотим перемесить все содержимое старого каталога, не должен существовать.

Работа с файлами. Классы File и FileInfo

Подобно паре Directory/DirectoryInfo для работы с файлами предназначена пара классов File и FileInfo. С их помощью мы можем создавать, удалять, перемещать файлы, получать их свойства и многое другое.

Некоторые полезные методы и свойства класса FileInfo:

- ✓ CopyTo(path): копирует файл в новое место по указанному пути path
- ✓ Create(): создает файл
- ✓ Delete(): удаляет файл
- ✓ MoveTo(destFileName): перемещает файл в новое место
- ✓ Свойство Directory: получает родительский каталог в виде объекта DirectoryInfo
- ✓ Свойство DirectoryName: получает полный путь к родительскому каталогу
- ✓ Свойство Exists: указывает, существует ли файл
- ✓ Свойство Length: получает размер файла
- ✓ Свойство Extension: получает расширение файла
- ✓ Свойство Name: получает имя файла
- ✓ Свойство FullName: получает полное имя файла

Класс File реализует похожую функциональность с помощью статических методов:

- ✓ Copy(): копирует файл в новое место
- ✓ Create(): создает файл
- ✓ Delete(): удаляет файл
- ✓ Move: перемещает файл в новое место
- ✓ Exists(file): определяет, существует ли файл

Получение информации о файле

```
string path = @"C:\apache\hta.txt";
FileInfo fileInf = new FileInfo(path);
```

```
if (fileInf.Exists)
{
    Console.WriteLine("Имя файла: {0}", fileInf.Name);
    Console.WriteLine("Время создания: {0}", fileInf.CreationTime);
    Console.WriteLine("Размер: {0}", fileInf.Length);
}
```

Удаление файла

```
string path = @"C:\apache\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
  fileInf.Delete();
  // альтернатива с помощью класса File
  // File.Delete(path);
}
```

Перемещение файла

```
string path = @"C:\apache\hta.txt";
string newPath = @"C:\SomeDir\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
  fileInf.MoveTo(newPath);
  // альтернатива с помощью класса File
  // File.Move(path, newPath);
}
```

Копирование файла

```
string path = @"C:\apache\hta.txt";
string newPath = @"C:\SomeDir\hta.txt";
FileInfo fileInf = new FileInfo(path);
if (fileInf.Exists)
{
  fileInf.CopyTo(newPath, true);
  // альтернатива с помощью класса File
  // File.Copy(path, newPath, true);
}
```

Метод CopyTo класса FileInfo принимает два параметра: путь, по которому файл будет копироваться, и булевое значение, которое указывает, надо ли при копировании перезаписывать файл (если true, как в случае выше, файл при копировании перезаписывается).

Если же в качестве последнего параметра передать значение false, то если такой файл уже существует, приложение выдаст ошибку.

Метод Copy класса File принимает три параметра: путь к исходному файлу, путь, по которому файл будет копироваться, и булевое значение, указывающее, будет ли файл перезаписываться.

Чтение и запись файла. Класс FileStream

Класс FileStream представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

Рассмотрим наиболее важные его свойства и методы:

- ✓ Свойство Length: возвращает длину потока в байтах
- ✓ Свойство Position: возвращает текущую позицию в потоке
- ✓ Метод Read: считывает данные из файла в массив байтов. Принимает три параметра: int Read(byte[] array, int offset, int count) и возвращает количество успешно считанных байтов. Здесь используются следующие параметры:

  array - массив байтов, куда будут помещены считываемые из файла данные

  offset представляет смещение в байтах в массиве array, в который считанные байты будут помещены

  count - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны.
- ✓ Метод long Seek(long offset, SeekOrigin origin): устанавливает позицию в потоке со смещением на количество байт, указанных в параметре offset.
- ✓ Метод Write: записывает в файл данные из массива байтов. Принимает три параметра: Write(byte[] array, int offset, int count)

  array - массив байтов, откуда данные будут записываться в файл

  offset - смещение в байтах в массиве array, откуда начинается запись байтов в поток

  count - максимальное число байтов, предназначенных для записи

FileStream представляет доступ к файлам на уровне байтов, поэтому, например, если вам надо считать или записать одну или несколько строк в текстовый файл, то массив байтов надо преобразовать в строки, используя специальные методы. Поэтому для работы с текстовыми файлами применяются другие классы.

В то же время при работе с различными бинарными файлами, имеющими определенную структуру FileStream может быть очень даже полезен для извлечения определенных порций информации и ее обработки.

Посмотрим на примере считывания-записи в текстовый файл:

```
Console.WriteLine("Введите строку для записи в файл:");
string text = Console.ReadLine();
// запись в файл
using (FileStream fstream = new FileStream(@"C:\SomeDir\noname\note.txt", FileMode.OpenOrCreate))
{
    // преобразуем строку в байты
    byte[] array = System.Text.Encoding.Default.GetBytes(text);
    // запись массива байтов в файл
    fstream.Write(array, 0, array.Length);
```

```
    Console.WriteLine("Текст записан в файл");
}
// чтение из файла
using (FileStream fstream = File.OpenRead(@"C:\SomeDir\noname\note.txt"))
{
    // преобразуем строку в байты
    byte[] array = new byte[fstream.Length];
    // считываем данные
    fstream.Read(array, 0, array.Length);
    // декодируем байты в строку
    string textFromFile = System.Text.Encoding.Default.GetString(array);
    Console.WriteLine("Текст из файла: {0}", textFromFile);
}
 Console.ReadLine();
```

Разберем этот пример. И при чтении, и при записи используется оператор using. Не надо путать данный оператор с директивой using, которая подключает пространства имен в начале файла кода. Оператор using позволяет создавать объект в блоке кода, по завершению которого вызывается метод Dispose у этого объекта, и, таким образом, объект уничтожается. В данном случае в качестве такого объекта служит переменная fstream.

Объект fstream создается двумя разными способами: через конструктор и через один из статических методов класса File.

Здесь в конструктор передается два параметра: путь к файлу и перечисление FileMode. Данное перечисление указывает на режим доступа к файлу и может принимать следующие значения:

- ✓ Append: если файл существует, то текст добавляется в конец файл. Если файла нет, то он создается. Файл открывается только для записи.
- ✓ Create: создается новый файл. Если такой файл уже существует, то он перезаписывается
- ✓ CreateNew: создается новый файл. Если такой файл уже существует, то он приложение выбрасывает ошибку
- ✓ Open: открывает файл. Если файл не существует, выбрасывается исключение
- ✓ OpenOrCreate: если файл существует, он открывается, если нет - создается новый
- ✓ Truncate: если файл существует, то он перезаписывается. Файл открывается только для записи.

Статический метод OpenRead класса File открывает файл для чтения и возвращает объект FileStream.

Конструктор класса FileStream также имеет ряд перегруженных версий, позволяющий более точно настроить создаваемый объект.

И при записи, и при чтении применяется объект кодировки Encoding.Default из пространства имен System.Text. В данном случае мы используем два его

метода: GetBytes для получения массива байтов из строки и GetString для получения строки из массива байтов.

В итоге введенная нами строка записывается в файл note.txt. По сути это бинарный файл (не текстовый), хотя если мы в него запишем только строку, то сможем посмотреть в удобочитаемом виде этот файл, открыв его в текстовом редакторе.

Однако если мы в него запишем случайные байты, например:

```
fstream.WriteByte(13);
fstream.WriteByte(103);
```

То у нас могут возникнуть проблемы с его пониманием. Поэтому для работы непосредственно с текстовыми файлами предназначены отдельные классы - StreamReader и StreamWriter.

Произвольный доступ к файлам

Нередко бинарные файлы представляют определенную стрктуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоборот записать в определенном месте файла определенный набор байтов. Например, в wav-файлах непосредственно звуковые данные начинаются с 44 байта, а до 44 байта идут различные метаданные - количество каналов аудио, частота дискретизации и т.д.

С помощью метода Seek() мы можем управлять положением курсора потока, начиная с которого производится считывание или запись в файл. Этот метод принимает два параметра: offset (смещение) и позиция в файле.

Позиция в файле описывается тремя значениями:

- ✓ SeekOrigin.Begin: начало файла
- ✓ SeekOrigin.End: конец файла
- ✓ SeekOrigin.Current: текущая позиция в файле

Курсор потока, с которого начинается чтение или запись, смещается вперед на значение offset относительно позиции, указанной в качестве второго параметра. Смещение может отрицательным, тогда курсор сдвигается назад, если положительное - то вперед.

Рассмотрим на примере:

```
using System.IO;
using System.Text;
class Program
{
    static void Main(string[] args)
    {
        string text = "hello world";

        // запись в файл
        using (FileStream fstream = new FileStream(@"D:\note.dat", FileMode.OpenOrCreate))
        {
            // преобразуем строку в байты
```

```
            byte[] input = Encoding.Default.GetBytes(text);
            // запись массива байтов в файл
            fstream.Write(input, 0, input.Length);
            Console.WriteLine("Текст записан в файл");

            // перемещаем указатель в конец файла, до конца файла- пять байт
            fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока

            // считываем четыре символов с текущей позиции
            byte[] output = new byte[4];
            fstream.Read(output, 0, output.Length);
            // декодируем байты в строку
            string textFromFile = Encoding.Default.GetString(output);
            Console.WriteLine("Текст из файла: {0}", textFromFile); // worl

            // заменим в файле слово world на слово house
            string replaceText = "house";
            fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока
            input = Encoding.Default.GetBytes(replaceText);
            fstream.Write(input, 0, input.Length);

            // считываем весь файл
            // возвращаем указатель в начало файла
            fstream.Seek(0, SeekOrigin.Begin);
            output = new byte[fstream.Length];
            fstream.Read(output, 0, output.Length);
            // декодируем байты в строку
            textFromFile = Encoding.Default.GetString(output);
            Console.WriteLine("Текст из файла: {0}", textFromFile); // hello house
        }
        Console.Read();
    }
}
```

Консольный вывод:

```
Текст записан в файл
Текст из файл: worl
Текст из файла: hello house
```

Вызов fstream.Seek(-5, SeekOrigin.End) перемещает курсор потока в конец файлов назад на пять символов: чтение и запись файлов через FileStream в C#. То есть после записи в новый файл строки "hello world" курсор будет стоять на позиции символа "w".

После этого считываем четыре байта начиная с символа "w". В данной кодировке 1 символ будет представлять 1 байт. Поэтому чтение 4 байтов будет эквивалентно чтению четырех сиволов: "worl".

Затем опять же перемещаемся в конец файла, не доходя до конца пять символов (то есть опять же с позиции символа "w"), и осуществляем запись строки "house". Таким образом, строка "house" заменяет строку "world".

Закрытие потока

В примерах выше для закрытия потока применяется конструкция using. После того как все операторы и выражения в блоке using отработают, объект FileStream уничтожается.

Однако мы можем выбрать и другой способ:

```
FileStream fstream = null;
try
{
    fstream = new FileStream(@"D:\note3.dat", FileMode.OpenOrCreate);
    // операции с потоком
}
catch(Exception ex)
{
}
finally
{
    if (fstream != null)
        fstream.Close();
}
```

Если мы не используем конструкцию using, то нам надо явным образом вызвать метод Close(): fstream.Close()

Работа с бинарными файлами. BinaryWriter и BinaryReader

Для работы с бинарными файлами предназначена пара классов BinaryWriter и BinaryReader. Эти классы позволяют читать и записывать данные в двоичном формате.

Основные метода класса BinaryWriter

- ✓ Close(): закрывает поток и освобождает ресурсы
- ✓ Flush(): очищает буфер, дописывая из него оставшиеся данные в файл
- ✓ Seek(): устанавливает позицию в потоке
- ✓ Write(): записывает данные в поток

Основные метода класса BinaryReader

- ✓ Close(): закрывает поток и освобождает ресурсы
- ✓ ReadBoolean(): считывает значение bool и перемещает указатель на один байт
- ✓ ReadByte(): считывает один байт и перемещает указатель на один байт
- ✓ ReadChar(): считывает значение char, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке

- ✓ ReadDecimal(): считывает значение decimal и перемещает указатель на 16 байт
- ✓ ReadDouble(): считывает значение double и перемещает указатель на 8 байт
- ✓ ReadInt16(): считывает значение short и перемещает указатель на 2 байта
- ✓ ReadInt32(): считывает значение int и перемещает указатель на 4 байта
- ✓ ReadInt64(): считывает значение long и перемещает указатель на 8 байт
- ✓ ReadSingle(): считывает значение float и перемещает указатель на 4 байта
- ✓ ReadString(): считывает значение string. Каждая строка предваряется значением длины строки, которое представляет 7-битное целое число

С чтением бинарных данных все просто: соответствующий метод считывает данные определенного типа и перемещает указатель на размер этого типа в байтах, например, значение типа int занимает 4 байта, поэтому BinaryReader считает 4 байта и переместит указать на эти 4 байта.

Посмотрим на реальной задаче применение этих классов. Попробуем с их помощью записывать и считывать из файла массив структур:

```
struct State
{
    public string name;
    public string capital;
    public int area;
    public double people;
    public State(string n, string c, int a, double p)
    {
        name = n;
        capital = c;
        people = p;
        area = a;
    }
}
class Program
{
    static void Main(string[] args)
    {
        State[] states = new State[2];
        states[0] = new State("Германия", "Берлин",  357168,  80.8);
        states[1] = new State("Франция", "Париж", 640679, 64.7);
        string path= @"C:\SomeDir\states.dat";
    try
        {
```

```
            // создаем объект BinaryWriter
            using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))
            {
                // записываем в файл значение каждого поля структуры
                foreach (State s in states)
                {
                    writer.Write(s.name);
                    writer.Write(s.capital);
                    writer.Write(s.area);
                    writer.Write(s.people);
                }
            }
            // создаем объект BinaryReader
            using (BinaryReader reader = new BinaryReader(File.Open(path, FileMode.Open)))
            {
                // пока не достигнут конец файла
                // считываем каждое значение из файла
                while (reader.PeekChar() > -1)
                {
                    string name = reader.ReadString();
                    string capital = reader.ReadString();
                    int area = reader.ReadInt32();
                    double population = reader.ReadDouble();

                    Console.WriteLine("Страна: {0}  столица: {1}  площадь {2} кв. км численность населения: {3} млн. чел.",
                        name, capital, area, population);
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        Console.ReadLine();
    }
}
```

Итак, у нас есть структура State с некоторым набором полей. В основной программе создаем массив структур и записываем с помощью BinaryWriter. Этот класс в качестве параметра в конструкторе принимает объект Stream, который создается вызовом File.Open(path, FileMode.OpenOrCreate).

Затем в цикле пробегаемся по массиву структур и записываем каждое поле структуры в поток. В том порядке, в каком эти значения полей записываются, в том порядке они и будут размещаться в файле.

Затем считываем из записанного файла. Конструктор класса BinaryReader также в качестве параметра принимает объект потока, только в данном случае устанавливаем в качестве режима FileMode.Open: new BinaryReader(File.Open(path, FileMode.Open))

В цикле while считываем данные. Чтобы узнать окончание потока, вызываем метод PeekChar(). Этот метод считывает следующий символ и возвращает его числовое представление. Если символ отсутствует, то метод возвращает -1, что будет означать, что мы достигли конца файла.

В цикле последовательно считываем значения поле структур в том же порядке, в каком они записывались.

Таким образом, классы BinaryWriter и BinaryReader очень удобны для работы с бинарными файлами, особенно когда нам известна структура этих файлов. В то же время для хранения и считывания более комплексных объектов, например, объектов классов, лучше подходит другое решение - сериализация.

Создание и чтение сжатых файлов. GZipStream и DeflateStream

Кроме классов чтения-записи .NET предоставляет классы, которые позволяют сжимать файлы, а также затем восстанавливать их в исходное состояние.

Это классы DeflateStream и GZipStream, которые находятся в пространстве имен System.IO.Compression и представляют реализацию одного из алгоритмов сжатия Deflate или GZip.

Рассмотрим применение класса GZipStream на примере:

```
using System.IO;
using System.IO.Compression;
class Program
{
    static void Main(string[] args)
    {
        string sourceFile = "D://test/book.pdf"; // исходный файл
        string compressedFile = "D://test/book.gz"; // сжатый файл
        string targetFile = "D://test/book_new.pdf"; // восстановленный файл
        // создание сжатого файла
        Compress(sourceFile, compressedFile);
        // чтение из сжатого файла
        Decompress(compressedFile, targetFile);
        Console.ReadLine();
    }
    public static void Compress(string sourceFile, string compressedFile)
    {
        // поток для чтения исходного файла
```

```
        using (FileStream sourceStream = new FileStream(sourceFile, FileMode.OpenOrCreate))
        {
          // поток для записи сжатого файла
          using (FileStream targetStream = File.Create(compressedFile))
          {
            // поток архивации
            using (GZipStream compressionStream = new GZipStream(targetStream, CompressionMode.Compress))
            {
              sourceStream.CopyTo(compressionStream); // копируем байты из одного потока в другой
              Console.WriteLine("Сжатие файла {0} завершено. Исходный размер: {1}  сжатый размер: {2}.",
                sourceFile, sourceStream.Length.ToString(), targetStream.Length.ToString());
            }
          }
        }
      }
      public static void Decompress(string compressedFile, string targetFile)
      {
        // поток для чтения из сжатого файла
        using (FileStream sourceStream = new FileStream(compressedFile, FileMode.OpenOrCreate))
        {
          // поток для записи восстановленного файла
          using (FileStream targetStream = File.Create(targetFile))
          {
            // поток разархивации
            using (GZipStream decompressionStream = new GZipStream(sourceStream, CompressionMode.Decompress))
            {
              decompressionStream.CopyTo(targetStream);
              Console.WriteLine("Восстановлен файл: {0}", targetFile);
            }
          }
        }
      }
    }
```

Метод Compress получает название исходного файла, который надо архивировать, и название будущего сжатого файла.

Сначала создается поток для чтения из исходного файла - FileStream sourceStream. Затем создается поток для записи в сжатый файл - FileStream

targetStream. Поток архивации GZipStream compressionStream инициализируется потоком targetStream и с помощью метода CopyTo() получает данные от потока sourceStream.

Метод Decompress производит обратную операцию по восстановлению сжатого файла в исходное состояние. Он принимает в качестве параметров пути к сжатому файлу и будущему восстановленному файлу.

Здесь в начале создается поток для чтения из сжатого файла FileStream sourceStream, затем поток для записи в восстанавливаемый файл FileStream targetStream. В конце создается поток GZipStream decompressionStream, который с помощью метода CopyTo() копирует восстановленные данные в поток targetStream.

Чтобы указать потоку GZipStream, для чего именно он предназначен - сжатия или восстановления - ему в конструктор передается параметр CompressionMode, принимающий два значения: Compress и Decompress.

Если бы захотели бы использовать другой класс сжатия - DeflateStream, то мы могли бы просто заменить в коде упоминания GZipStream на DeflateStream, без изменения остального кода. Их использование идентично.

В то же время при использовании этих классов есть некоторые ограничения, в частности, мы можем сжимать только один файл.

Для архивации группы файлы лучше выбрать другие инструменты.

РАЗДЕЛ 2. ПРАКТИЧЕСКИЙ

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1

КЛАСС, СОЗДАНИЕ ОБЪЕКТА КЛАССА.ПОНЯТИЕ ИНКАПСУЛЯЦИИ

Цель работы: получить навыки проектирования простейших классов. Научиться создавать объекты класса. Освоить принцип инкапсуляции.

Краткие теоретические сведения

Класс – это тип, определяемый программистом, в котором объединяются структуры данных и функции их обработки. Переменные типа класс называются экземплярами класса и создаются при помощи оператора new.

Классы могут содержать переменные и константы, называемые полями. В классе могут быть объявлены функции, выполняющие действия над полями и именуемые методами.

Проектирование классов следует выполнять, придерживаясь стратегии минимальной связанности и зависимости между ними. Это достигается за счет использования принципа инкапсуляции. Инкапсуляция – это ограничение доступа к полям и методам при помощи модификаторов доступа. Основные модификаторы доступа в C#: public – доступ без ограничений; private – доступ разрешен только членам класса; protected – доступ разрешен как членам данного класса, так и производного. По умолчанию действует модификатор доступа private.

Пример объявления класса с закрытым полем и открытым методом:

```
class Employee
{
     private string name=«Петров»;
     public void PrintName()
     {
        Console.WriteLine(«Name=»+name);
     }
}
```

Кроме полей и методов в классе можно объявлять свойства. Свойства позволяют объявить два метода, один из которых вызывается при установке значения свойства (метод set), а второй при его чтении (метод get). Обычно код этих методов содержит обращение к полю, хранящему значение свойства. Возможно объявление свойства либо только для чтения, либо только для записи.

В методе записи set для доступа к записываемому значению используют

ключевое слово value.

Пример класса Employee дополненного свойством для чтения и записи:

```
class Employee
{
      private string name=«Петров»;
      public string Name
      {
          get
          {
             return name;
          }
          set
          {
             name = value;
          }
      }
      public void PrintName()
      {
          Console.WriteLine(«Name»=+name);
      }
}
```

После создания объекта класса для доступа к открытым членам класса (полям, методам и свойствам) используют оператор точка. В одной программе можно создать несколько объектов одного класса.

Задание к работе

1. Выбрать предметную область согласно варианту индивидуального задания.
2. Спроектировать класс для выбранной предметной области.
3. Нарисовать диаграмму спроектированного класса.
4. Предусмотреть наличие у объекта полей, методов и свойств.
5. Предусмотреть наличие свойств только для записи.

Индивидуальные задания

1. Предметная область: **АТС**. В классе хранить информацию об адресе АТС, числе абонентов, абонентской плате (для всех абонентов одна). Реализовать метод для подсчета абонентской платы всех клиентов.
2. Предметная область: **Вокзал**. В классе хранить информацию о наименовании станции, стоимости билета (стоимость одинакова для всех

направлений), числе мест, числе проданных билетов. Реализовать метод для подсчета общей стоимости всех непроданных билетов.

3. Предметная область: **ЖЭС**. В классе хранить информацию о районе, к которому принадлежит ЖЭС, номере ЖЭС, числе жильцов, оплате за месяц (для всех жильцов одна), числе оплативших. Реализовать метод для подсчета общей задолженности жильцов.

4. Предметная область: **Аэропорт**. В классе хранить информацию о названии аэропорта, стоимости билета (стоимость одинаковая), общем числе мест во всех самолетах, числе проданных биле- тов. Реализовать метод для подсчета общей стоимости всех проданных билетов.

5. Предметная область: **Банк**. В классе хранить информацию о наименовании банка, числе вкладов, размере вклада (все вклады одинаковые), размере процентной ставки. Реализовать метод для подсчета общей выплаты по процентам.

6. Предметная область: **Отдел кадров**. В классе хранить информацию о наименовании предприятия, числе работников, норме выработки часов в месяц (одна для всех работников), оплате за час, подоходном налоге. Реализовать метод для подсчета общей выплаты по подоходному налогу.

7. Предметная область: **Фирма грузоперевозок**. В классе хранить информацию об оплате за перевозку одной тонны грузов (не зависит от направления), о массе перевезенных грузов, наименовании фирмы. Реализовать метод для подсчета общей выручки фирмы.

8. Предметная область: **Гостиница**. В классе хранить информацию о названии гостиницы, числе заселенных мест, общем числе мест, оплате за день проживания (для всех жильцов одинаковая стоимость). Реализовать метод для подсчета общей выручки гостиницы.

9. Предметная область: **Интернет-оператор**. В классе хранить информацию о стоимости тарифа (одна для всех пользователей), наименовании оператора, числе абонентов. Реализовать метод для подсчета общей выручки.

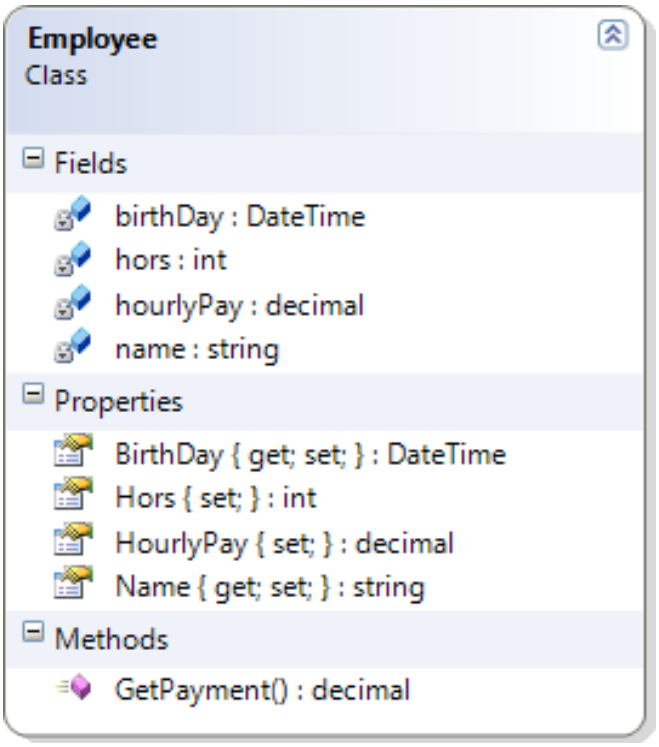
10. Предметная область: **Интернет-магазин** по продаже телевизоров. В классе хранить информацию о стоимости телевизора (одна для всех моделей), наименовании магазина, числе покупок. Реализовать метод для подсчета общей выручки.

Пример выполнения работы

Пусть задана предметная область: **Завод**. У работника завода хранить фамилию, год рождения, размер почасовой оплаты и количество отработанных часов. В классе реализовать метод для подсчета заработной платы работника,

исходя из величины почасовой оплаты и отработанных часов.

Диаграмма спроектированного класса:

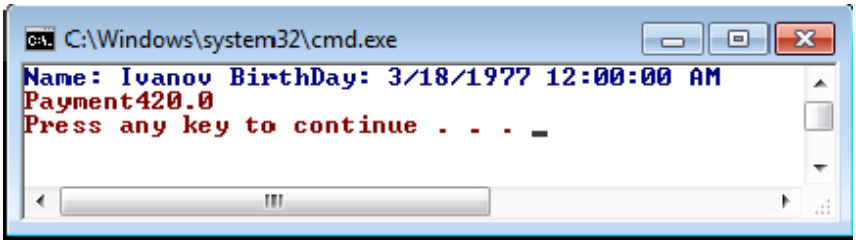


Текст программы:

```
using System;
// объявление пользовательского класса
class Employee
{
    string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
    //объявление закрытого поля
     DateTime birthDay;
    //объявление свойства для чтения и для записи
      public DateTime BirthDay
      {
          get { return birthDay; }
          set { birthDay = value; }
      }
      decimal hourlyPay;
      public decimal HourlyPay
      {
          set { hourlyPay = value; }
      }
      private int hors;
```

```
    public int Hors
    {
        set { hors = value; }
    }
    public decimal GetPayment()
    {
        return hors * hourlyPay;
    }
}
class Program
{
  static void  Main()

    {
    //создание объекта класса
      Employee ivanov = new Employee();
      ivanov.Name = «Ivanov»;
      ivanov.BirthDay = new DateTime(1977, 03,
      18);
      ivanov.Hors = 40;
      ivanov.HourlyPay = 10.5M;
      decimal p = ivanov.GetPayment();
      Console.WriteLine(«Name: {0}
       BirthDay:
      {1}»,ivanov.Name,ivanov.BirthDay);
      Console.ForegroundColor    =    ConsoleCo-
      lor.DarkRed;
      Console.WriteLine(«Payment»     +     iva-
      nov.GetPayment());
  }
}
```

Результат работы:



Контрольные вопросы

1. В чем заключается принцип инкапсуляции?
2. При помощи какого ключевого слова создается объект класса?
3. Как объявить свойство только для чтения?
4. Чем поля класса отличаются от свойств?

Лабораторная работа № 2

КОНСТРУКТОРЫ. СТАТИЧЕСКИЕ ЧЛЕНЫ КЛАССА.ШАБЛОН ПРОЕКТИРОВАНИЯ SINGLETON

Цель работы: изучить работу и назначение конструкторов. Освоить возможности членов класса с модификатором static. Ознакомиться с шаблоном проектирования Singleton.

Краткие теоретические сведения

Конструктор – это метод, имеющий имя такое же, как и имя класса и не возвращающий параметров. Конструктор вызывается при создании объекта класса и служит для начальной инициализации полей и свойств. Как и любой метод класса, можно перегрузить конструктор, при этом вызов соответствующего конструктора будет определяться по списку параметров, который указывается при создании объекта класса.

Пример перегрузки конструктора:

```
class Item
   {
      decimal price;
      public decimal Price
      {
          get { return price; }
          set { price = value; }
      }
      public Item()
      {
          price = 10;
      }
      public Item(decimal p)
      {
          price = p;
      }
}
```

Для вызова конструктора без параметров, именуемого конструктором по умолчанию, следует написать

```
Item i1=new Item();
```

Когда требуется вызвать конструктор с параметрами, передаваемые фактические значения указываются в круглых скобках, как по- казано ниже:

```
Item i1=new Item(50);
```

Если в одном конструкторе следует вызвать другой перегруженный

конструктор, то после объявления первого следует поставить двоеточие и указать ключевое слово this. В круглых скобках после this через запятую указываются фактические значения, передаваемые конструктору.

Статический член класса объявляется с ключевым словом static и является независимым от всех объектов класса. Статическое поле или метод становится доступным до создания объекта класса. Для доступа к статическому члену за переделами класса достаточно указать имя этого класса и через оператор «точка» имя статического члена.

Шаблон проектирования Singleton – это порождающий шаблон, задачей которого является гарантия возможности создания только одного объекта класса и предоставление к этому объекту глобальной точки доступа.

Для применения шаблона проектирования Singleton к конкретному классу требуется: объявить в этом классе закрытый конструктор; объявить закрытую статическую ссылку на данный класс; добавить открытый статический метод, возвращающий ссылку на единственный созданный объект класса. Статический метод возвращает ссылку на единственный созданный объект класса и является глобальной точкой доступа к этому объекту.

Пример класса, написанного в соответствии с шаблоном Singleton:

```
class World
{
  private static World world;

    private World()
    {  }
    public static World GetWorld()
    {
        if (world==null) world = new World();
        return world;
    }
  }
```

Задание к работе

1. Спроектировать классы для выбранной предметной области.
2. Нарисовать диаграмму классов.
3. Применить к одному из классов шаблон проектирования Singleton.

Индивидуальные задания

Разработать два класса: класс-контейнер, управляющий контейнеризируемым классом, и контейнеризируемый класс. Для класса- контейнера применить шаблон проектирования Singleton.

Описание предметной области:

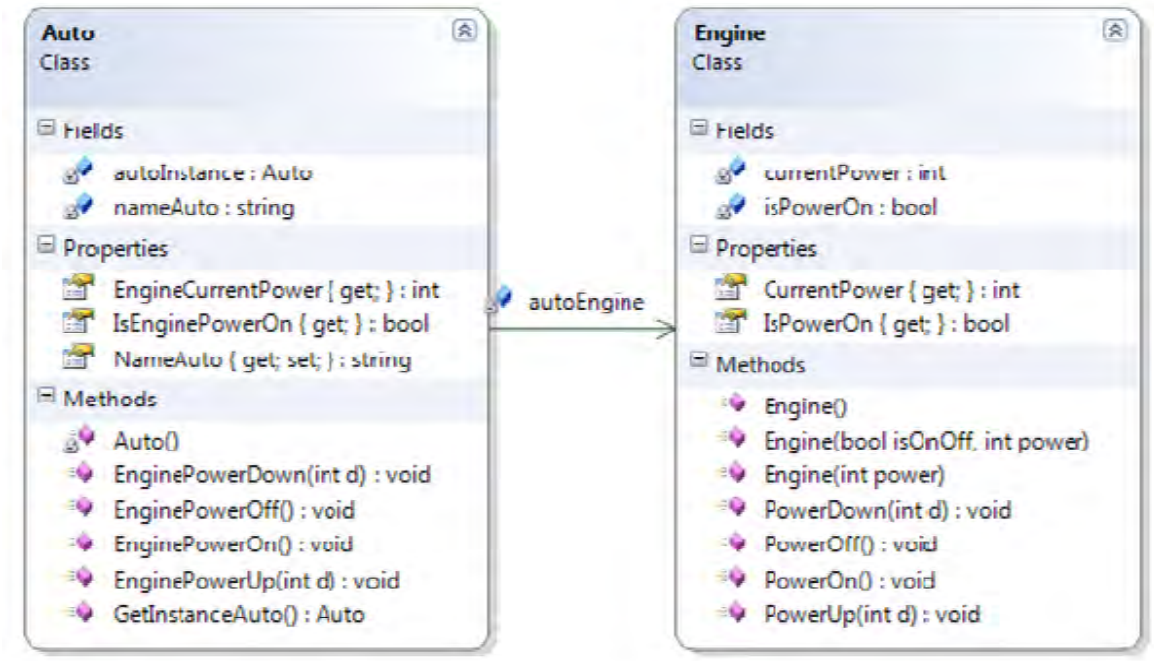
1. Здание – Отопительная система.
2. Компьютер – Винчестер.

3. Больница – Приемное отделение.
4. Завод – Склад деталей.
5. Аэропорт – Взлетная полоса.
6. Вокзал – Богажное отделение.
7. Фирма – Отдел кадров.
8. Ресторан – Кухня.
9. Компьютер – Монитор.
10. Библиотека – Книгохранилище.

Пример выполнения работы

Пусть задана предметная область: Автомобиль – Двигатель автомобиля. Класс «автомобиль» является контейнерным классом, а класс «двигатель» – контейнеризируемым. В классе «двигатель» хранится информация о его состоянии (включен/выключен) и о текущей мощности.

Диаграмма классов:



Текст программы:

```
using System;
//класс двигатель
class Engine
{
//конструктор
    public Engine()
    {
        isPowerOn = true;
        currentPower = 10;
    }
//конструктор
```

```
    public Engine(int power)
    {
        isPowerOn = true;
        currentPower = power;
    }
//конструктор
    public Engine(bool isOnOff, int power)

        {
            isPowerOn = isOnOff;
            if (!isPowerOn)
                currentPower = 0;
            else
                currentPower = power;
        }
   // состояние двигателя
        private bool isPowerOn;

        public bool IsPowerOn
        {
            get { return isPowerOn; }
        }
   //текущая мощность
        private int currentPower;
        public int CurrentPower
        {
            get { return currentPower; }
        }
   // включить двигатель
        public void PowerOn()
        {
            isPowerOn = true;
        }
   //выключить двигатель
        public void PowerOff()
        {
            isPowerOn = false;
            currentPower = 0;
        }
   //поднять мощность
        public void PowerUp(int d)
        {
            if(isPowerOn)
            currentPower += d;
        }
   //убавить мощность

    public void PowerDown(int d)
    {
        currentPower -= d;
    }
```

```
}
//класс автомобиль
class Auto
{
//ссылка на объект класса Автомобиль
    private static Auto autoInstance;
//ссылка на объект класса Двигатель
    private Engine autoEngine;
//закрытый конструктор
    private Auto()
    {
        autoEngine = new Engine(false,0);
    }
    public static Auto GetInstanceAuto()
    {
        if (autoInstance == null)
        {
            autoInstance = new Auto();
        }
        return autoInstance;
    }
//наименование автомобиля
private string nameAuto;
public string NameAuto
    {
        get { return nameAuto; }
        set { nameAuto = value; }
    }
//состояние двигателя
    public bool IsEnginePowerOn
    {
        get
        {         return autoEngine.IsPowerOn;
        }
    }

    public void EnginePowerOn()
    {
        autoEngine.PowerOn();
    }


    public void EnginePowerOff()
    {
        autoEngine.PowerOff();
    }
//текущая мощность автомобиля
    public int EngineCurrentPower
    {
         get
        {      return  autoEngine.CurrentPower;
        }
    }
//прибавить мощность
    public void EnginePowerUp(int d)
```

```
    {
        autoEngine.PowerUp(d);
    }
//убавить мощность
    public void EnginePowerDown(int d)
    {
        autoEngine.PowerDown(d);
    }
}
class Program
{
    static void Main(string[] args)
    {
        //создание объекта класса автомобиль

        Auto auto1 = Auto.GetInstanceAuto();
        auto1.NameAuto = «BMW»;
        PrintInfoOfAuto(auto1);
       //включить двигатель
        auto1.EnginePowerOn();
        //установить мощность
        auto1.EnginePowerUp(20);
        PrintInfoOfAuto(auto1);
        //выключить двигатель
        auto1.EnginePowerOff();
        PrintInfoOfAuto(auto1);
    }
     //  вывод информации о параметрах
     автомобиля
    private static void PrintInfoOfAuto(Autoauto1)
    {
        Console.WriteLine(«{0}двигатель
     включен{1}»,auto1.NameAuto, auto1.IsEnginePowerOn);
        Console.WriteLine(«Текущая мощность(кВт) {0}»,
 auto1.EngineCurrentPower);
    }
}
```

Контрольные вопросы

1. Чем конструктор отличается от обычного метода?
2. Какие преимущества дает перегрузка конструкторов?
3. Возможен ли доступ к статическому методу из экземплярногои наоборот?
4. Какое назначение шаблона проектирования Singleton?

Лабораторная работа № 3

ИСПОЛЬЗОВАНИЕ КОЛЛЕКЦИЙ

Цель работы: получить навыки проектирования приложения, состоящего из нескольких взаимосвязанных классов.

Краткие теоретические сведения

Чтобы хранить набор ссылок на объекты удобно использовать обобщенный класс List, объявленный в пространстве имен System.Collections.Generic. При создании объекта класса List в треугольных скобках указывается тип данных, который будет храниться в коллекции.

Пример объявления коллекции:

List<Employee> listEmployee =new List<Employee>();

Для обращения к элементам коллекции удобно использовать цикл foreach:

```
foreach (Employee emp in listEmployee)
{ //тело цикла       }
```

Класс List унаследован от стандартных интерфейсов IList, ICollection, IEnumerable, что позволяет выполнять основные действия над коллекцией: добавление элемента, удаление элемента, доступ через индекс.

В программе классы могут быть связаны отношением ассоциации, когда один класс содержит ссылки на другие классы. Различают кратность ассоциации «один к одному», когда один класс содержит ссылку на другой класс, и кратность «один ко многим», когда один класс содержит коллекцию ссылок на другой класс.

Ассоциации бывают однонаправленные и двунаправленные. Однонаправленные ассоциации предполагают навигацию от одного объекта к другому только в одном направлении. В случае двунаправленных ассоциаций ссылки на взаимно-ассоциированные объекты присутствуют как в первом, так и во втором классе.

Задание к работе

1. Для заданной предметной области спроектировать программную структуру, состоящую из 3–5 классов.
2. В соответствии с разработанной диаграммой классов выполнить программную реализацию.
3. Предусмотреть использование типа данных – перечисление.
4. Ввод/вывод должен быть реализован вне проектируемого класса.

Индивидуальные задания

1. Предметная область: **АТС**. На АТС хранится информация о всех клиентах станции. АТС имеет список тарифов на междугородние разговоры. Клиент АТС может совершать множество звонков в различные города.

Система должна:

- позволять вводить информацию о тарифах;
- вводить информацию о клиентах и регистрировать звонки;
- по введенной фамилии о клиенте определять стоимость всех сделанных им звонков в соответствии с действующими тарифами;
- вычислять общую стоимость всех выполненных на АТС звонков.

2. Предметная область: **Вокзал**. Касса вокзала имеет список тарифов на различные направления. При покупке билета регистрируются паспортные данные пассажира. Пассажир покупает билеты на различные направления.

Система должна:

- позволять вводить данные о тарифах;
- позволять вводить паспортные данные пассажира и регистрировать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- после ввода наименования направления выводить список всех пассажиров, купивших на него билет.

3. Предметная область: **ЖЭС**. В ЖЭС хранятся тарифы на коммунальные услуги. ЖЭС имеет информацию о всех жильцах. При потреблении жильцами коммунальных услуг информация регистрируется в системе.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- ввод информации о жильцах и потребленных услугах;
- после ввода фамилии выводить сумму всех потребленных услуг;
- выводить стоимость всех оказанных услуг.

4. Предметная область: **Аэропорт**. Касса аэропорта имеет списоктарифов на различные направления. При покупке билета регистрируются паспортные данные.

Система должна:

- позволять вводить данные о тарифах;
- позволять вводить паспортные данные пассажира и регистри-ровать покупку билета;
- рассчитывать стоимость купленных пассажиром билетов;
- рассчитывать стоимость всех проданных билетов.

5. Предметная область: **Банк**. Информационная система банка хранит описание процентов по различным вкладам. Система хранит информацию о вкладчиках и сделанных ими вкладах. Каждый клиент может поместить в банк только один вклад.

Система должна позволять выполнять следующие задачи:

- хранить информацию о процентах по вкладам;
- хранить информацию о клиентах;
- пополнять клиенту величину вклада;
- вычислять общую сумму выплат по процентам для всех вкладов.

6. Предметная область: **Отдел расчета зарплаты**. Информаци- онная система отдела расчета зарплаты на предприятии хранит дан- ные о величине оплаты за различные виды работ. Система хранит информацию о работниках предприятия.

Система должна позволять выполнять следующие задачи:

- вводить информацию о различных видах работ;
- вводить информацию о работниках и выполненных ими работах;
- после ввода фамилии выводить для работника зарплату;
- выводить сумму выплат всем работникам.

7. Предметная область: **Фирма грузоперевозок**. Фирма имеет спи- сок тарифов по перевозке грузов. Клиент регистрируется в системе, после чего может заказать перевозку определенного объема груза.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация клиента и заказ на перевозку грузов;
- вывод суммы заказа для определенного клиента;
- подсчет суммарной стоимости всех заказов.

8. Предметная область: **Гостиница**. Информационная система гостиницы хранит информацию о всех номерах и их стоимости. Система регистрирует клиентов. Каждый клиент может заказать один номер. При попытке заказа номера, который занят, выводится предупреждение.

Система должна позволять выполнять следующие задачи:
- ввод информации о номерах и их стоимости;
- регистрация клиента и заказ номера;
- вывод списка не занятых номеров;
- после ввода фамилии клиента вывод стоимости проживания.

9. Предметная область: **Интернет-оператор**. Провайдер имеет различные тарифы доступа в Интернет за 1 Мбайт в зависимости от величины абонентской платы. Информационная система провайде- ра хранит данные о клиентах.

Система должна позволять выполнять следующие задачи:

- ввод тарифов;
- регистрация пользователя;
- ввод данных о потребленном трафике для конкретного пользователя;
- подсчет общей стоимости реализованного трафика;
- поиск клиента, заплатившего наибольшую стоимость за услуги.

10. Предметная область: **Интернет-магазин**. В информационной системе хранятся данные о товарах. Клиент звонит в магазин и оставляет заказ на товар.

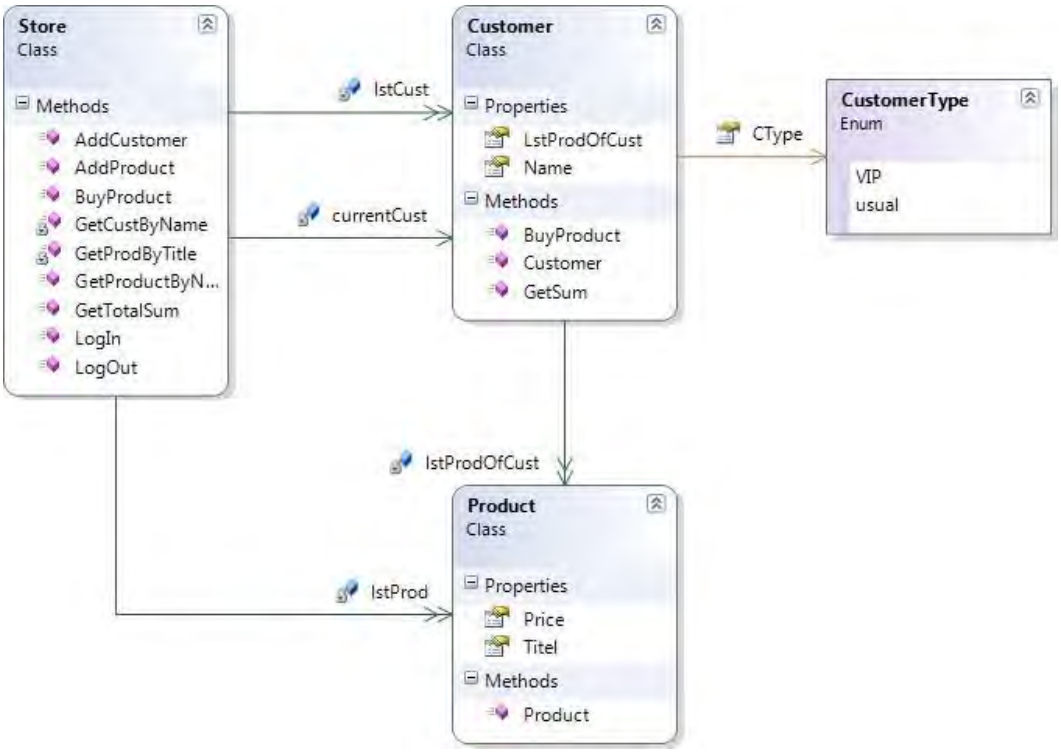
Система должна позволять выполнять следующие задачи:

- ввод информации о товарах;
- регистрация заказа клиента на покупку определенного товара;
- после ввода фамилии покупателя вывод списка заказанных имтоваров;
- после ввода фамилии покупателя вывод суммы заказа.

Пример выполнения работы

Предметная область: **Торговая система**. В информационной системе хранятся данные о товарах и покупателях. Некоторые покупатели имеют статус «VIP». Для приобретения товара вводится фамилия, после чего регистрируются купленные товары. Система должна высчитывать общую сумму проданных товаров.

Диаграмма классов:



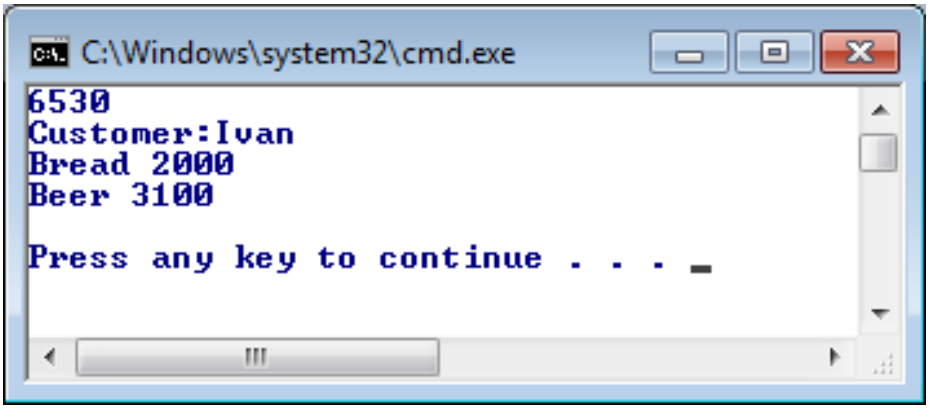
Текст программы:

```
using System;
using System.Collections.Generic;
class Product
{
    public string Titel { get; set; }
    public int Price { get; set; }
    public Product(string t, int p)
    {
        Titel = t;
        Price = p;
    }

}
class Store{

  List<Product>      lstProd      =      new
  List<Product>();
  List<Customer>      lstCust      =      new
  List<Customer>();
  Customer currentCust;
  public void AddProduct(string t, int p)
  {
      lstProd.Add(new Product(t, p));
  }
  public  void  AddCustomer(string  n,  Custo-
  merType ct)
  {
      lstCust.Add(new Customer(n, ct));
  }
  Customer GetCustByName(string n)
  {
      foreach (Customer item in lstCust)
      {
          if (item.Name == n) return item;
      }
      return null;
  }

  Product GetProdByTitle(string t)
  {
      foreach (Product item in lstProd)
      {
          if (item.Titel == t) return item;
      }
      return null;
  }
```

```
    public void LogIn(string n)
    {
        if (currentCust == null)
            currentCust = GetCustByName(n);
    }

    public void LogOut()
    {
        currentCust = null;
    }

    public void BuyProduct(string titel)
    {
        Product p = GetProdByTitle(titel);
        currentCust.BuyProduct(p);
    }

    public int GetTotalSum()
    {
        int sum = 0;
        foreach (Customer c in lstCust)
        {
            foreach    (Product    p     inc.LstProdOfCust)
            {
                sum += p.Price;
            }
        }
        return sum;
    }
    public   string   GetProductsByName(stringname)
    {
        Customer c = GetCustByName(name);
        string  s  =  «Customer:»  +  c.Name  + «\n»;
        foreach (Product p in c.LstProdOfCust)
        {
            s  +=  p.Titel  +  «  «  +  p.Price  + «\n»;
        }
        return s;


    }
}

enum CustomerType { VIP, usual };
class Customer
{
    List<Product>   lstProdOfCust   =    newList<Product>();
    public List<Product> LstProdOfCust
    {
        get { return lstProdOfCust; }
    }
    public string Name { get; set; }
    public CustomerType CType { get; set; }
```

```
    public Customer(string n, CustomerType ct)
    {
        Name = n;       CType = ct;
    }

    public void BuyProduct(Product p)
    {
        lstProdOfCust.Add(p);
    }

    public int GetSum()
    {
        int s = 0;
        foreach  (Product  item  in  lstProdOf-Cust)
        {
            s += item.Price;
        }
        return s;
    }
}

class Program
{
    static void Main()
    {
        Store riga = new Store();
        riga.AddProduct(«Milk», 1430);
        riga.AddProduct(«Bread», 2000);
        riga.AddProduct(«Beer», 3100);
        riga.AddCustomer(«Masha»,  Customer-Type.VIP);
        riga.AddCustomer(«Ivan»,   Customer-Type.usual);
        riga.LogIn(«Ivan»); riga.BuyProduct(«Bread»);
        riga.BuyProduct(«Beer»); riga.LogOut();
        riga.LogIn(«Masha»); riga.BuyProduct(«Milk»);
        riga.LogOut();
        Console.WriteLine(riga.GetTotalSum());Con-
        sole.WriteLine(riga.GetProductsByName(
        «Ivan»));
    }
}
```

Результат работы программы:



РАЗДЕЛ 3. КОНРОЛЬ ЗНАНИЙ

Общая формулировка заданий к контрольной работе

Номер варианта выбираем по номеру в журнале. При совпадении вариантов обе работы не принимаются.

Требования к отчету по контрольной работе (приложение 1). Оформляется один отчет на два задание.

Задание №1.

Выполнить задание согласно варианту. Продемонстрировать работу программы с помощью **консольного приложения**. Каждый разрабатываемый класс должен содержать следующие элементы: **скрытые поля, конструкторы с параметрами и без параметров, методы, свойства, индексаторы, перегруженные операции**. Функциональные элементы класса должны обеспечивать непротиворечивый, полный и удобный интерфейс класса. При возникновении ошибок должны **выбрасываться исключения**. В программе (**в методе Main()**) должна выполняться проверка всех разработанных элементов класса.

Вариант 1

Создать класс «**Сторона**» для хранения длины стороны фигуры. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Треугольник», обеспечивающий следующие возможности:

• установку и получение длин **сторон** треугольника;

• проверку на существование треугольника с заданными длинами сторон;

• расчет периметра и площади треугольника;

• определение типов треугольника: равнобедренный, равносторонний, разносторонний; остроугольный, тупоугольный, прямоугольный.

Вариант 2

Создать класс «**Число**», содержащий закрытое поле для хранения целого числа в диапазоне от –15 до 45. Обеспечить проверку на правильность ввода чисел, предусмотреть свойства для получения состояния объекта.

Создать класс «Одномерный массив» для работы с массивом целых **чисел** (вектором).

Обеспечить следующие возможности:

• обращение к отдельному элементу массива с контролем выхода за пределы массива;

• выполнение операций поэлементного сложения и вычитания массивов с одинаковыми границами индексов;

• выполнение операций умножения и деления всех элементов массива на скаляр;

• вывод на экран элемента массива по заданному индексу и всего массива.

Вариант 3

Создать класс «**Товар**», содержащий следующие закрытые поля: название товара, количество, стоимость товара в рублях. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Склад», содержащий закрытый массив товаров. Обеспечить:

• вывод информации о **товаре** по номеру с помощью индекса;

• вывод информации о товаре, название которого введено с клавиатуры, если таких товаров нет, выдать соответствующее сообщение;

• сортировку товаров по наименованию, по количеству и по цене.

Вариант 4

Создать класс «**Строка**», содержащий закрытое поле для хранения русскоязычной строки. Обеспечить проверку на правильность ввода русских символьных данных, предусмотреть свойства для получения состояния объекта.

Создать класс «Текст» для работы с массивом русскоязычных **строк**.

Обеспечить следующие возможности:

• обращение к отдельной строке массива по индексу с контролем выхода за пределы массива;

• выполнение операций поэлементного сцепления двух массивов с образованием нового массива;

• выполнение операций слияния двух массивов с исключением повторяющихся элементов;

• вывод на экран элемента массива по заданному индексу и всего массива.

Вариант 5

Создать класс «**Книга**», содержащий следующие закрытые поля: автор, название, год издания, категория. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Домашняя библиотека». Предусмотреть возможность работы с произвольным числом **книг**, поиска книги по какому-либо признаку (по автору, по году издания или категории), добавления книг в библиотеку, удаления книг из нее, доступа к книге по номеру.

Вариант 6

Создать класс, реализующий тип данных «**Вещественная матрица**» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• сложение, вычитание (как с другой матрицей, так и с числом);

• комбинированные операции присваивания (+=, -=);

• операции сравнения на равенство/неравенство;
• операции вычисления обратной и транспонированной матрицы;
• доступ к элементу по индексам.

Вариант 7

Создать класс «**Элемент**», содержащий закрытое поле для хранения символа. Предусмотреть свойства для получения состояния объекта.

Создать класс «Множество», позволяющий выполнять основные операции надо множеством символов: добавление и удаление **элемента**, пересечение, объединение и разность множеств.

Вариант 8

Создать класс «**Автомобиль**», содержащий закрытые поля: госномер, цвет, фамилия владельца. Для каждого автомобиля указывается номер места и признак присутствия на стоянке. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Автостоянка» для хранения сведений об **автомобилях**. Обеспечить возможность поиска автомобиля по разным критериям, вывода списка присутствующих и отсутствующих на стоянке автомобилей, доступа к имеющимся сведениям по номеру места.

Вариант 9

Создать класс «**Студент**», содержащий следующие закрытые поля: фамилия, имя, дата рождения, группа. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Студенческая группа». Предусмотреть возможность работы с переменным числом **студентов**, поиска студента по какому-либо признаку (например, по фамилии, имени, дате рождения), добавления и удаления записей, сортиювки по разным полям, доступа к записи по номеру.

Вариант 10

Создать класс «**Колода карт**», включающий закрытый массив элементов класса «Карта». Предусмотреть свойства для получения состояния объекта. В карте хранятся масть и номер. Обеспечить возможность вывода карты по номеру, вывода всех карт, перемешивания колоды и выдачи всех карт из колоды поодиночке и по 6 штук в случайном порядке.

Написать программу, демонстрирующую все разработанные элементы классов с обеспечением проверки на правильность ввода данных.

Вариант 11

Создать класс «**самолет**», содержащий следующие закрытые поля: название пункта назначения, шестизначный номер рейса, время отправления. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Аэропорт», содержащий закрытый массив **самолетов**. Обеспечить следующие возможности:

• вывод информации о самолете по номеру рейса с помощью индекса;

• вывод информации о самолетах, отправляющихся в течение часа после введенного с клавиатуры времени;

• вывод информации о самолетах, отправляющихся в заданный пункт назначения;

Информация должна быть отсортирована по времени отправления.

Вариант 12

Создать класс, реализующий тип данных «**Вещественная матрица**» и работу с ними. Класс должен реализовывать следующие операции над матрицами:

• методы, реализующие проверку типа матрицы (квадратная, диагональная, нулевая, единичная, симметричная, верхняя треугольная, нижняя треугольная);

• операции сравнения на равенство/неравенство;

• доступ к элементу по индексам.

Вариант 13

Создать класс «**поезд**», содержащий следующие закрытые поля: название пункта, назначения, номер поезда (может содержать буквы и цифры), время отправления. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Вокзал», содержащий закрытый массив **поездов**. Обеспечить следующие возможности:

• вывод информации о поезде по номеру с помощью индекса;

• вывод информации о поездах, отправляющихся после введенного с клавиатуры времени;

• перегруженную операцию сравнения, выполняющую сравнение времени отправления двух поездов;

• вывод информации о поездах, отправляющихся в заданный пункт назначения.

Информация должна быть отсортирована по времени отправления.

Вариант 14

Создать класс «**Точка**», содержащий закрытые поля для хранения координат точки. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Массив точек», который обеспечивает следующие возможности:

• вывод координаты **точки** на экран;

• расчет расстояния от начала координат до точки;

• перемещение точки на плоскости на вектор (a, b).

• упорядочивание точек относительно оси ординат.

Вариант 15

Создать класс «**Жилец**», содержащий закрытые поля для хранения следующей информации: ФИО, город, улица, номер дома, номер квартиры, телефон. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Дом» для хранения информации по всем **жильцам**. Реализовать следующие возможности:

• вывод информации о конкретном жильце дома по заданным критериям (фамилия, номер квартиры);

• сортировка жильцов по фамилиям, по адресу проживания;

• поиск всех жильцов, проживающих по заданному адресу.

Вариант 16

Создать класс «**Деньги**», содержащий закрытые поля для хранения номинала купюры и их количества. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Банк», в котором реализовать закрытый массив **денег**, обеспечивающий следующими возможностями:

• вывод номинала и количества купюр, хранящихся в банке;

• вывод полной суммы денег, хранящейся в банке;

• определение, хватит ли денежных средств на покупку товара на сумму N рублей.

• определение, сколько штук товара стоимости n рублей можно купить на имеющиеся денежные средства.

Вариант 17

Создать класс для работы с восьмеричным числом, хранящимся в виде строки символов. Реализовать конструкторы, свойства, методы и следующие операции:

• операции присваивания, реализующие значимую семантику;

• операции сравнения;

• преобразование в десятичное число;

• форматный вывод;

• доступ к заданной цифре числа по индексу.

Вариант 18

Создать класс для работы с датой. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Класс должен реализовывать следующие возможности:

• вычисление даты предыдущего дня;

• вычисление даты следующего дня;

• определение количества дней до конца месяца;

• определение года високосным;

• определение даты в зависимости от количества дней, пройденных с начала года.

Вариант 19

Создать класс «Предметный указатель». Каждый компонент указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти. Предусмотреть возможность формирования указателя с клавиатуры и из файла, вывода указателя, вывода номеров страниц для заданного слова, удаления элемента из указателя.

Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Вариант 20

Создать класс для работы с регулярными выражениями. Класс должен содержать закрытые поля для хранения шаблона поиска (тип Regex) и текста (тип String). Предусмотреть свойства для получения состояния объекта.

Класс должен реализовывать следующие возможности:

• получение и вывод текста на экран;

• определение, содержит ли текст фрагменты, соответствующие шаблону поиска;

• вывод на экран всех фрагментов текста, соответствующих шаблону поиска;

• удаление из текста всех фрагментов, соответствующих шаблону поля.

Вариант 21

Создать класс «Преподаватель», содержащий следующие закрытые поля: ФИО, предмет, группа, даты занятий. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Расписание», позволяющий хранить сведения о всех преподавателях учебного заведения и их графике работы. Реализовать следующие возможности:

• вывод информации о преподавателе на экран;

• изменение расписания;

• вывод информации о занятиях на заданную дату (преподаватель, группа, предмет);

• упорядочивание списка преподавателей по различным данным (фамилия, предмет, количество групп).

Вариант 22

Создать класс «Число», содержащий закрытое поле для хранения вещественного числа в диапазоне от –35.5 до 35.5. Обеспечить проверку на правильность ввода чисел, предусмотреть свойства для получения состояния объекта.

Создать класс для работы с двухмерным массивом вещественных чисел произвольного размера.

Обеспечить следующие возможности:

• изменение числа строк и столбцов;

• вывод на экран подматрицы любого размера и всей матрицы;

• выполнение операций поэлементного сцепления двух матриц с образованием новой матрицы;

• доступ по индексам к элементу матрицы.

Вариант 23

Создать класс «Англо-русский словарь», обеспечивающий возможность хранения нескольких вариантов перевода для каждого слова. Реализовать доступ по строковому индексу - английскому слову. Обеспечить возможность вывода всех значений слов по заданному префиксу.

Вариант 24

Создать класс «запись», содержащий следующие закрытые поля: ФИО, номер телефона, дата рождения (массив из трех чисел). Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Записная книжка», содержащий закрытый массив записей. Обеспечить:

• вывод на экран информации о человеке, номер телефона которого введен (если такого нет, то выдать соответствующее сообщение);

• поиск людей, день рождения которых сегодня или в заданный день;

• поиск людей, день рождения которых будет в этом месяце;

• поиск людей, номер телефона которых начинается на три заданных цифры.

Вариант 25

Создать класс «Сторона» для хранения длины стороны фигуры. Обеспечить проверку на правильность ввода данных. Предусмотреть свойства для получения состояния объекта.

Создать класс «Прямоугольник», обеспечивающий следующие возможности:

• вывод длин сторон прямоугольника на экран;

• расчет периметр и площади прямоугольника;

• определение, является ли данный прямоугольник квадратом;

• определение, можно ли вписать один заданный прямоугольник в другой.

Задание №2.

Выполнить задание согласно варианту. Продемонстрировать работу программы с помощью **консольного приложения**. В заданиях требуется описать **абстрактный** базовый класс и производные от него, создать **параметризованную коллекцию** объектов производных классов. Обеспечить читабельный вывод значений полей классов на экран. Используя механизм **виртуальных методов**, продемонстрировать единообразную работу с элементами коллекции. Должна быть обработка исключительных ситуаций. Создать диаграмму классов.

Вариант 1

Создать абстрактный класс File, инкапсулирующий в себе методы Open, Close, Seek, Read, Write, GetPosition и GetLength. Создать производные классы MyDataFile1 и MyDataFile2— файлы, содержащие в себе данные некоторого определенного типа MyData1 и MyData2, а также заголовки, облегчающие доступ к этим файлам.

Создать класс Folder, содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода списка имен и длин файлов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 2

Создать абстрактный класс Point (точка). На его основе создать классы ColoredPoint и Line. На основе класса Line создать класс ColoredLine и класс PolyLine (многоугольник). Все классы должны иметь виртуальные методы установки и получения значений всех координат, а также изменения цвета и получения текущего цвета.

Создать класс Picture, содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 3

Создать абстрактный класс Vehicle. На его основе реализовать классы Car (автомобиль), Bicycle (велосипед) и Lorry (грузовик). Классы должны иметь возможность задавать и получать параметры средств передвижения (цена, максимальная скорость, год выпуска и т.д.). Наряду с общими полями и методами, каждый класс должен содержать и специфичные для него поля.

Создать класс Garage, содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 4

Создать абстрактный класс Figure. На его основе реализовать классы Rectangle (прямоугольник), Circle (круг) и Trapezium (трапеция) с возможностью вычисления площади, центра тяжести и периметра.

Создать класс Picture, содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 5

Создать абстрактный класс Number с виртуальными методами, реализующими арифметические операции. На его основе реализовать классы Integer и Real.

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 6

Создать абстрактный класс Body. На его основе реализовать классы Parallelepiped (прямоугольный параллелепипед), Cone (конус) и Ball (шар) с возможностью вычисления площади поверхности и объема.

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 7

Создать абстрактный класс Currency для работы с денежными суммами. Определить в нем методы перевода в рубли и вывода на экран. На его основе реализовать классы Dollar, Euro и Pound (фунт стерлингов) с возможностью пересчета в центы и пенсы соответственно.

Создать класс Purse (кошелек), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода общей суммы, переведенной в рубли, и суммы по каждой из валют. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 8

Создать абстрактный класс Triangle (треугольник), задав в нем длину двух сторон, угол между ними, методы вычисления площади и периметра. На его основе создать классы, описывающие равносторонний, равнобедренный и

прямоугольный треугольники со своими методами вычисления площади и периметра.

Создать класс Picture, содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и получения суммарной площади. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 9

Создать абстрактный класс Solution (решение) с виртуальными методами вычисления корней уравнения и вывода на экран. На его основе реализовать классы Linear (линейное уравнение) и Square (квадратное уравнение).

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 10

Создать абстрактный класс Function (функция) с виртуальными методами вычисления значения функции $y = f(x)$ в заданной точке x и вывода результата на экран. На его основе реализовать классы Ellipse, Hiperbola и Parabola.

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 11

Создать абстрактный класс Triad (тройка) с виртуальными методами увеличения на 1. На его основе реализовать классы Date (дата) и Time (время).

Создать класс Memories, содержащий параметризованную коллекцию пар (дата-время) объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и выборки самого раннего и самого позднего событий. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 12

Описать абстрактный класс Element (элемент логической схемы), задав в нем числовой идентификатор, количество входов, идентификаторы присоединенных к нему элементов (до 10) и двоичные значения на входах и выходе. На его основе реализовать классы AND и OR — двоичные вентили, которые могут иметь различное количество входов и один выход и реализуют логическое умножение и сложение соответственно.

Создать класс Scheme (схема), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка и вычисление значений, формируемых на выходах схемы по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 13

Описать абстрактный класс Element (элемент логической схемы) задав в нем символьный идентификатор, количество входов, идентификаторы присоединенных к нему элементов (до 10) и двоичные значения на входах и выходе. На его основе реализовать классы AND_NOT и OR_NOT — двоичные вентили, которые могут иметь различное количество входов и один выход и реализуют логическое умножение с отрицанием и сложение с отрицанием соответственно.

Создать класс Scheme (схема), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка и вычисление значений, формируемых на выходах схемы по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 14

Описать абстрактный класс Trigger (триггер), задав в нем идентификатор и двоичные значения на входах и выходах. На его основе реализовать классы RS и JK, представляющие собой триггеры соответствующего типа.

Создать класс Register (регистр), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка, общего сброса и установки значений каждого триггера по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 15

Создать абстрактный класс Progression (прогрессия) с виртуальными методами вычисления заданного элемента и суммы прогрессии. На его основе реализовать классы Linear (арифметическая) и Exponential (геометрическая).

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и вывода общей суммы всех прогрессий. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 16

Создать абстрактный класс Pair (пара значений) с виртуальными методами, реализующими арифметические операции. На его основе реализовать классы Fractional (дробное) и LongLong (длинное целое).

В классе Fractional вещественное число представляется в виде двух целых, в которых хранятся целая и дробная часть числа соответственно. В классе LongLong длинное целое число хранится в двух целых полях в виде старшей и младшей части.

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и вывода общей суммы всех значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 17

Создать абстрактный класс Integer (целое) с символьным идентификатором, виртуальными методами, реализующими арифметические операции, и методом вывода на экран. На его основе реализовать классы Decimal (десятичное) и Binary (двоичное). Число представить в виде массива цифр.

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода значений и идентификаторов всех объектов списка и вывода общей суммы всех десятичных значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 18

Создать абстрактный класс Sorting (сортировка) с идентификатором последовательности, виртуальными методами сортировки, получения суммы и вывода на экран. На его основе реализовать классы Choice (метод выбора) и Quick (быстрая сортировка).

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода идентификаторов и сумм элементов каждого объекта списка, а также вывода общей суммы всех значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 19

Создать абстрактный класс Pair (пара значений) с виртуальными методами, реализующими арифметические операции, и методом вывода на экран. На его основе реализовать классы Money (деньги) и Complex (комплексное число).

В классе Money денежная сумма представляется в виде двух целых, в которых хранятся рубли и копейки соответственно. При выводе части числа снабжаются словами «руб.» и «коп.». В классе Complex предусмотреть при выводе символ мнимой части (i).

Создать класс Series (набор), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Вариант 20

Создать абстрактный класс Worker с полями, задающими фамилию работника, фамилии руководителя и подчиненных и виртуальными методами вывода списка обязанностей и списка подчиненных на экран. На его основе реализовать классы Manager (руководитель проекта), Developer (разработчик) и Coder (младший программист).

Создать класс Group (группа), содержащий параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода всех объектов списка и выборки по фамилии с выводом всего дерева подчиненных. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Список литературы.

1. Шилдт, Герберт C# 4.0 Полное руководство. : Пер. с английского. М. – ООО «И.Д. Вильямс», 2011 – 1056 с. : ил.

2. Уотсон, К. Visual C# 2010: полный курс.: Пер. с англ. - М.: ООО "И.Д. Вильяме", 2011. - 960 с. : ил.

3. Троелсен, Эндрю. Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильяме", 2011. — 1392 с. : ил.

Приложение 1

ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА ПО КОНТРОЛЬНОЙ РАБОТЕ

1. Отчет по контрольной работе сдается в бумажном виде после обязательной регистрации. Преподавателю отправляется в электроном виде.
2. Имя файла отчета «**Фамилия_Номер_группы_Номер_варианта**».

Пример: Иванов_4017412_15.docx

3. Версия редактора Word не менее Word 2010. Общие требования к оформлению текста представлены ниже.
4. Содержание отчета
   - титульный лист
   - содержание
   - задание
   - исходный код
   - скриншоты работы приложения
   - заключение
   - список использованной литературы

Требования к разделам: Титульный лист должен обязательно содержать – фамилию, имя, отчество, номер группы, номер варианта. Исходный код, представленный в отчете, должен **обязательно** содержать комментарии к ключевым строкам программы (подписать классы, методы). Скриншоты работы приложения должны полностью раскрывать все функциональные возможности разработанного приложения.

ОБЩИЕ ТРЕБОВАНИЯ ПО ОФОРМЛЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ

Параметры страницы

Формат листа – А4 (размер 210 × 290 мм). Перед набором текста настроить параметры **Microsoft Word**:

- поля – 20 мм;
- номер страницы ставится **снизу**, по центру;
- ориентация – книжная.

Основной текст

- абзац: первая строка – отступ **1,25** мм, междустрочный интервал – «**одинарный**», выравнивание – «**по ширине»**;
- шрифт – **Times New Roman**, **14 пт**;
- перенос слов – **автоматический**;

- выделять (жирным или курсивом) отдельные слова, словосочетания и предложения следует исходя из важности терминов.

ТРЕБОВАНИЯ К ИСХОДНОМУ КОДУ ПРОГРАММ

1. Программа должна быть разработана в **Visual Studio 2017-2022** или в **SharpDevelop** версии 4.3.0 на языке программирования C#, версия .NET 4.0.
2. Программа должна быть скомпилирована и иметь файл с расширением **.exe** для запуска приложения.
3. Имя архива «**Фамилия_Номер_группы_Номер_варианта»**

Пример: Иванов_25412_15.rar

РАБОТЫ, ОФОРМЛЕННЫЕ С НАРУШЕНИЕМ ТРЕБОВАНИЙ, ПРИНЯТЫ НЕ БУДУТ.

РАЗДЕЛ 4. ВСПОМОГАТЕЛЬНЫЙ

ПРОГРАММА ДИСЦИПЛИНЫ

Учебная программа по учебной дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования» разработана для специальности 1-40 01 01 «Программное обеспечение информационных технологий» специализации 1-40 01 01 01 «Веб-технологии и программное обеспечение мобильных систем».

Целью изучения учебной дисциплины является формирование устойчивых теоретических знаний и практических навыков в технологиях, языках и инструментальных средствах объектно-ориентированного программирования. Изучение данной дисциплины является необходимым этапом в профессиональном развитии специалиста в области информационных технологий и позволяет в дальнейшем совершенствовать навыки разработки профессиональных программных средств, отвечающих современному этапу развития компьютерной техники.

Основными задачами преподавания учебной дисциплины являются: освоение возможностей, предоставляемых современными компьютерными технологиями; изучение принципов проектирования, создания, масштабирования объектно-ориентированных приложений; овладение методами, подходам, принципами создания объектно-ориентированных приложений; приобретение знаний и навыков проектирования и создания объектно-ориентированных приложений; формирование навыков программирования с использованием объектно-ориентированных подходов; приобретение навыков работы в интегрированной среде разработки приложений.

Базовыми учебными дисциплинами по курсу «Объектно-ориентированные технологии программирования и стандарты проектирования» являются «Иностранный язык», «Дискретная математика», «Технологии разработки программного обеспечения», «Основы алгоритмизации и программирования». В свою очередь, учебная дисциплина «Объектно-ориентированные технологии программирования и стандарты проектирования» является базой для таких учебных дисциплин, как «Скриптовые языки программирования», «Программирование сетевых приложений», «Программирование мобильных информационных систем».

В результате изучения дисциплины «Объектно-ориентированные технологии программирования и стандарты проектирования» формируются следующие компетенции:

универсальные: владеть основами исследовательской деятельности, осуществлять поиск, анализ и синтез информации; обладать навыками саморазвития и самосовершенствования в профессиональной деятельности; проявлять инициативу и адаптироваться к изменениям в профессиональной деятельности;

базовая профессиональная: применять фундаментальные методы и свойства объектно-ориентированного проектирования и программирования для разработки проектных и программных решений задач в рамках объектно-ориентированной парадигмы.

В результате изучения учебной дисциплины студент должен:

знать:

- базовые понятия и синтаксис объектно-ориентированного языка программирования, технологию объектно-ориентированного проектирования и приемы разработки программ;
- методы создания и использования основных объектов и конструкций объектно-ориентированного языка программирования;
- технологию создания, организации и использования иерархии классов, предопределенных классов и типов данных, методы ограничения доступа и обработки исключительных ситуаций;
- методы параметризации классов и их использование для решения практических задач;
- методы применения шаблонов и контейнерных абстракций;
- методы работы с потоками ввода/вывода и разработки интерактивных приложений;

уметь:

- определять абстракции, модули, строить иерархию классов для реализации программ;
- использовать принципы типизации, инкапсуляции, наследования, полиморфизма для разработки программных продуктов;
- использовать возможности стандартных библиотек объектно-ориентированного языка программирования;
- использовать механизм исключений для создания устойчивых приложений;
- создавать собственные и использовать предоставляемые стандартные библиотеки динамических структур данных;

- использовать технологию объектно-ориентированного проектирования для разработки сложных и масштабируемых программ и систем;

владеть:

- методами, инструментальными средствами и системами разработки объектно-ориентированных программ;
- навыками формализации предметной области с помощью средств объектно-ориентированного анализа;
- техникой создания объектно-ориентированных программных компонент и организацией их взаимодействия в программных проектах.

Освоение данной учебной дисциплины обеспечивает формирование следующих компетенций:

Для специальности 1-40 01 01 «Программное обеспечение информационных технологий»:

БПК-17. Использовать объектно-ориентированный подход в технологии разработки программных систем

Согласно учебному плану для заочной формы получения высшего образования (срок обучения – 4 года) на изучение учебной дисциплины отведено всего 240 ч., из них аудиторных – 52 часа.

Согласно учебному плану для заочной формы получения высшего образования (срок обучения – 5 лет) на изучение учебной дисциплины отведено всего 240 ч., из них аудиторных – 52 часа.

Распределение аудиторных часов по курсам, семестрам и видам занятий приведено в таблицах 1 и 2.

Таблица 1.

Заочная (дистанционная) форма получения высшего образования (срок обучения – 4 года)					
Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Практические занятия, ч.	Форма текущей аттестации
2	3	12	14		экзамен
2	4	12	14		экзамен

Таблица 2.

Заочная (дистанционная) форма получения высшего образования (срок обучения – 5 лет)					
Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Практические занятия, ч.	Форма текущей аттестации
2	3	12	14		экзамен
2	4	12	14		экзамен

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Раздел I. КОНЦЕПЦИЯ И ОСОБЕННОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ И ПРОГРАММИРОВАНИЯ

Тема 1.1. Концептуальные основы объектно-ориентированного проектирования. Сравнение принципов объектно-ориентированного проектирования с другими парадигмами

Предмет учебной дисциплины и ее содержание. Связь учебной дисциплины с другими дисциплинами учебного плана. Парадигмы программирования и проектирования, их особенности. Основные направления в программировании и проектировании программных продуктов. Возникновение объектно-ориентированного программирования и особенности использования в нем принципов объектно-ориентированного проектирования. Базовые принципы объектно-ориентированного программирования.

Тема 1.2. Фундаментальные методы, подходы, свойства объектной модели, ее преимущества, недостатки, особенности использования

Основные положения объектной модели. Ее составные элементы, свойства, преимущества, недостатки. Абстрагирование. Модульность. Иерархия. Типизация. Взаимосвязь основных элементов объектно-ориентированной парадигмы.

Раздел II. БАЗОВЫЕ АБСТРАКЦИИ ОБЪЕКТНО - ОРИЕНТИРОВАННОГО ПРОЕКТИРОВАНИЯ И ПРОГРАММИРОВАНИЯ

Тема 2.1. Базовые конструкции объектно-ориентированных программ. Абстрагирование как один из основных принципов в объектно-ориентированном проектировании и программировании. Особенности использования абстракции для выделения основных элементов проектируемой системы

Классы и объекты в объектно-ориентированном проектировании и программировании. Компоненты класса: поля и методы. Инициализация и разрушение объектов класса. Использование конструкторов и деструкторов класса. Конструктор по умолчанию, конструктор копирования. Перегрузка и

переопределение методов класса.

Тема 2.2. Инкапсуляция как один из основных принципов в объектно-ориентированном проектировании и программировании. Методы и принципы реализации инкапсуляции и организации корректного доступа к элементам объекта

Атрибуты доступа к компонентам класса. Область действия класса и доступ к компонентам класса. Управление доступом к компонентам класса.

Тема 2.3. Структурные элементы класса, методы взаимодействия объектов классов. Особенности создания корректных связей между классами

Организация внешнего доступа к локальным компонентам класса. Понятие интерфейса в объектно-ориентированном проектировании и программировании. Дружественные методы как способ доступа к содержимому класса. Статические и константные компоненты класса. Особенности использования статических полей и методов класса. Сравнение использования статических компонент класса и статических переменных (локальных и глобальных). Вложенные классы, особенности организации доступа к ним. Перегрузка операторов и методов класса. Преобразование типов данных (явное и неявное). Использование указателей и ссылок на объекты. Операторы динамического выделения и освобождения памяти при работе со встроенными и пользовательскими типами данных. Организация ввода/вывода данных. Статические и динамические массивы объектов пользовательских типов данных.

Раздел III. МЕТОДЫ И МЕХАНИЗМЫ РАЗРАБОТКИ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ПРОГРАММ

Тема 3.1. Наследование как один из основных принципов в объектно-ориентированном проектировании и программировании. Механизмы наследования типов и определения собственных типов данных. Принципы и подходы при повторном использования кода

Базовые и производные классы как основа повторного использования кода. Основные правила и принципы построения базовых и производных классов. Атрибуты доступа при наследовании. Работа конструкторов и деструкторов при наследовании. Связь композиции, агрегации, наследования (обобщения) при

проектировании программных продуктов. Переопределение методов базового класса в производном. Простое и множественное наследование. Особенности и проблемы, возникающие при реализации множественного наследования в объектно-ориентированных языках программирования.

Тема 3.2. Полиморфизм как один из основных принципов в объектно-ориентированном проектировании и программировании. Основные проявления, механизмы, способы реализации полиморфизма

Понятие раннего и позднего связывания. Использование виртуального механизма вызова методов при реализации принципа полиморфизма. Перегрузка операторов и методов как полиморфизм этапа компиляции. Переопределение методов как полиморфизм этапа выполнения программы. Виртуальные методы класса и механизм их использования. Абстрактные классы: их назначение, свойства, необходимость создания при проектировании объектно-ориентированной системы. Возможные пути решения неоднозначности при множественном наследовании.

Тема 3.3. Использование параметризированных классов в объектно-ориентированном проектировании и программировании. Особенности использования обобщенного проектирования и программирования в объектно-ориентированном

Параметризированные классы и методы: их свойства, особенности использования. Совместное использование параметризации и принципов наследования. Особенности использования параметров типов в объектно-ориентированном языке программирования. Организация внешнего доступа к компонентам параметризированных классов. Параметризированные классы и статические элементы класса. Создание специализированной версии параметризированного класса. Задание значений параметров класса по умолчанию.

Тема 3.4. Управление совместным использованием ресурсов. Создание собственных механизмов и использование встроенных компонент для реализации управления ресурсами

Реализация концепции RAII как принцип управления ресурсами. Особенности создания и использования умных указателей и механизма транзакций при управлении используемыми ресурсами.

Тема 3.5. Особенности возникновения и обработки исключительных ситуаций

Основы возникновения, создания, обработки исключительных ситуаций. Альтернативные способы обработки исключительных ситуаций. Генерация исключений как способ описания возникших исключительных ситуаций. Перехватывание исключений. Повторная генерация исключения. Обработка неожидаемых типов исключительных ситуаций. Генерация исключений в конструкторах, особенности создания объектов при возникновении исключительных ситуаций. Взаимосвязь возникновения исключений и иерархии классов. Спецификация исключений. Классы исключений стандартной библиотеки. Создание собственных типов исключительных ситуаций.

Тема 3.6. Использование потоков ввода/вывода как основа создания интерактивных программных средств. Объектно-ориентированное проектирование в библиотеках, реализующих ввод/вывод данных

Понятие потока ввода/вывода данных. Особенности создания и закрытия потоков ввода/вывода. Организация ввода данных из потока и вывода данных в поток. Особенности перегрузки операторов при использовании потоков ввода/вывода. Контроль состояния потока, установка битов ошибок, исправление ошибок, возникающих при вводе/выводе данных. Неформатированный ввод/вывод данных. Стандартные и определяемые пользователем манипуляторы потоков как способ управления вводом/выводом данных.

Тема 3.7. Использование потоков файлового ввода/вывода как основа создания программных средств с возможностью долговременного хранения данных. Объектно-ориентированное проектирование в библиотеках, реализующих ввод/вывод данных в/из файлы(ов)

Файловые потоки ввода/вывода данных. Общие свойства потоков ввода/вывода данных. Режимы открытия файловых потоков. Реализация последовательного и произвольного доступа к содержимому файла. Организация ввода/вывода данных переменных примитивных типов и объектов классов.

Тема 3.8. Контейнерные типы данных как возможность делегирования ответственности выделения динамической памяти. Особенности применения стандартных библиотек классов коллекций

Введение в стандартную библиотеку шаблонов (классов коллекций), основные понятия, концепции. Классы контейнеры и итераторы. Их взаимосвязь, особенности использования. Типы контейнерных классов, адаптеры контейнеров. Алгоритмы библиотеки классов: их типы, особенности использования с контейнерными классами.

Тема 3.9. Использование паттернов проектирования при разработке объектно-ориентированных приложений. Особенности и основные принципы применения объектно-ориентированного проектирования при разработке прикладных программ

Особенности использования паттернов проектирования при проектировании и программировании объектно-ориентированных программ. Основные виды паттернов проектирования, особенности использования, решаемые задачи. Взаимосвязь паттернов проектирования и принципов объектно-ориентированного проектирования и программирования.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

заочная (дистанционная) форма получения высшего образования (срок обучения – 4 и 5 лет)[1]

Номер раздела, темы	Название раздела, темы, занятия	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	2	3	4	5	6	7	8	9
	1 семестр							
1.	Концепция и особенности объектно-ориентированного проектирования и программирования							
1.1	Концептуальные основы объектно-ориентированного проектирования. Сравнение принципов объектно-ориентированного проектирования с другими парадигмами	2						
1.2	Фундаментальные методы, подходы, свойства объектной модели, ее преимущества, недостатки, особенности использования	2						
	Лабораторное занятие №1. Разработка набора классов и их объектов. Реализация корректных связей между классами		2					
	Лабораторное занятие №2. Создание массива объектов класса		2					
2.	Базовые абстракции объектно-ориентированного проектирования и программирования							
2.1	Инкапсуляция как один из основных принципов в объектно-ориентированном проектировании и программировании. Методы и принципы реализации инкапсуляции и организации корректного доступа к элементам объекта	2						

[1] Темы учебного материала, не указанные в Учебно-методической карте, отводятся на самостоятельное изучение студента.

2.2	Структурные элементы класса, методы взаимодействия объектов классов. Особенности создания корректных связей между классами	2						
	Лабораторное занятие №3. Организация ввода/вывода. Динамическое выделение памяти.		2					
	Лабораторное занятие №4. Дружественные функции и классы		2					
3.	Методы и механизмы разработки объектно-ориентированных программ							контрольная работа
3.1	Наследование как один из основных принципов в объектно-ориентированном проектировании и программировании. Механизмы наследования типов и определения собственных типов данных. Принципы и подходы при повторном использования кода	2						
3.2	Полиморфизм как один из основных принципов в объектно-ориентированном проектировании и программировании. Основные проявления, механизмы, способы реализации полиморфизма	2						
	Лабораторное занятие №5. Наследование. Простое наследование		2					
	Лабораторное занятие №6. Принцип полиморфизма. Виртуальные функции		2					
	Лабораторное занятие №7. Абстрактные классы		2					
	Итого за семестр	12	14					экзамен
	Всего аудиторных часов	26						

Номер раздела, темы	Название раздела, темы, занятия	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	2	3	4	5	6	7	8	9
	2 семестр							
3.	Методы и механизмы разработки объектно-ориентированных программ							контрольная работа
3.3	Использование параметризированных классов в объектно-ориентированном проектировании и программировании. Особенности использования обобщенного проектирования и программирования в объектно-ориентированном	2						
3.5	Особенности возникновения и обработки исключительных ситуаций	2						
3.6	Использование потоков ввода/вывода как основа создания интерактивных программных средств. Объектно-ориентированное проектирование в библиотеках, реализующих ввод/вывод данных	2						
	Лабораторное занятие №1. Параметризация в объектно-ориентированном проектировании и программировании. Реализация шаблонов классов		2					
	Лабораторное занятие №2. Генерация и обработка исключительных ситуаций		2					
	Лабораторное занятие №3. Потоки ввода/вывода		2					
3.7	Использование потоков файлового ввода/вывода как основа создания программных средств с возможностью долговременного хранения данных. Объектно-ориентированное проектирование в библиотеках, реализующих ввод/вывод данных в/из файлы(ов)	2						
3.8	Контейнерные типы данных как возможность делегирования ответственности выделения динамической памяти. Особенности применения стандартных библиотек классов коллекций	2						
	Лабораторное занятие №4. Организация работы с файлами		2					
	Лабораторное занятие №5. Последовательные классы-контейнеры		2					

3.9	Использование паттернов проектирования при разработке объектно-ориентированных приложений. Особенности и основные принципы применения объектно-ориентированного проектирования при разработке прикладных программ	2						
	Лабораторное занятие №6. Практические приемы использования шаблонов типов и иерархии классов		2					
	Лабораторное занятие №7. Классы-итераторы библиотеки Standard Template Library		2					
	Итого за семестр	12	14					экзамен
	Всего аудиторных часов	26						

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Список литературы

Основная литература

1. Вайсфельд, М. Объектно-ориентированный подход / М. Вайсфельд. – Санкт-Петербург: Питер, 2020. – 256 с.
2. Шилдт, Г. C++. Полное руководство / Г. Шилдт. – Москва: Вильямс, 2019. – 800 с.
3. Лафоре, Р. Объектно-ориентированное программирование в C++. Классика Computer Science / Р. Лафоре. – 4-е изд. – Санкт-Петербург: Питер, 2021. – 928 с.
4. Страуструп, Б. Программирование. Принципы и практика с использованием C++ / Б. Страуструп. – Москва: Вильямс, 2018. – 1328 с.
5. Шилдт, Г. C++: базовый курс / Г. Шилдт. – 3-е издание. – Москва: Диалектика-Вильямс, 2018. – 624 с.
6. Гамма, Э. Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм. – Санкт-Петербург: Питер, 2020. – 448 с.
7. Липпман, Стенли Б. Язык программирования C++. Базовый курс / Стенли Б. Липпман, Жози Му Лажойе, Э. Барбара. – 5-е изд.; пер. с англ. – Москва: Диалектика-Вильямс, 2018. – 1118 с.
8. Страуструп, Б. Язык программирования C++. Краткий курс / Б. Страуструп. – 2-е изд. – Москва: Вильямс, 2019. – 320 с.
9. Липачёв, Е. К. Технология программирования. Базовые конструкции C/C++: учебно-справочное пособие / Е. К. Липачев. – Казань: Казан. ун-т, 2012. – 142 с.
10. Васильев, А. Программирование на C++ в примерах и задачах / А. Васильев. – Москва: Эксмо, 2021. – 368 с.
11. Horton, I. Beginning C++17: From Novice to Professional / I. Horton, P. Van Weert. – New York: Apress Media, 2018. – 804 p.
12. Хорстманн, Кей С. Java. Библиотека профессионала. Т. 1: Основы / Кей С. Хорстманн ; пер. с англ. – Москва : Вильямс, 2019. – 864 с.
13. Эккель, Б. Философия Java / Б. Эккель. – 4-е полное изд. – Санкт-Петербург: Питер, 2019. – 1168 с.
14. Эванс, Э. Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем / Э. Эванс. – Москва: Вильямс, 2020. – 448 с.
15. Маклафлин, Б. Объектно-ориентированный анализ и проектирование / Б. Маклафлин, Д. Уэст. – Санкт-Петербург: Питер, 2019. – 608 с.
16. Вернон, В. Реализация методов предметно-ориентированного проектирования / В. Вернон. – Москва: Вильямс, 2019. – 688 с.

Дополнительная литература

1. Доусон, М. Изучаем C++ через программирование игр / М. Доусон. – Санкт-Петербург: Питер, 2021. – 352 с.
2. Дьюхэрст, К. Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ / К. Дьюхэрст. – Москва: ДМК Пресс, 2017. – 264 с.
3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – Москва: Вильямс, 2018. – 1244 с.
4. Александреску, А. Стандарты программирования на C++ / А. Александреску, Г. Саттер. – Москва: Вильямс, 2019. – 224 с.
5. Кнут, Д. Искусство программирования : в 2 т. Т. 1 : Основные алгоритмы / Д. Кнут. – Москва: Вильямс, 2019. – 720 с.
6. Кнут, Д. Искусство программирования: в 2 т. Т. 2: Получисленные алгоритмы / Д. Кнут. – Москва: Вильямс, 2019. – 832 с.
7. Хайнеман, Д. Алгоритмы справочник с примерами на C, C++, Java и Python / Д. Хайнеман. – Санкт-Петербург: Альфа-книга, 2020. – 432 с.
8. Павловская, Т. C/C++. Процедурное и объектно-ориентированное программирование / Т. Павловская. – СПб.: Питер Мейл, 2018. – 496 с.

Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин – Санкт-Петербург: Питер, 2019. – 464 с.

Экзаменационные вопросы

1. Введение в язык программирования C# и технологию .NET
2. Платформа .NET (CLR, CTS и CLS)
3. Различия между сборками, пространствами имен и типами
4. Система типов языка программирования C# и платформы .NET
5. Память в .NET: стек (stack) и динамическая память (куча, heap). Область видимости (scope) и время жизни переменных и объектов. Изменяемые и неизменяемые типы данных
6. Базовый синтаксис языка программирования C#
7. Основы консольного ввода-вывода в C#. Форматирование числовых данных
8. Конструкции принятия решений и операции равенства/сравнения C#
9. Итерационные конструкции C#
10. Методы и модификаторы параметров в C#
11. Описание и способы вызова динамических и статических методов в C#
12. Работа со строковыми данными в C#
13. Массивы в C#
14. Неявно типизированные локальные переменные вC#
15. Основные принципы и идеи методологии ООП. Реализация ООП в C#
16. Объект в ООП и его основные характеристики. Классы в ООП. Общее определение класса
17. Описание класса и способы создания экземпляров классов (объектов) в C#. Способы инициализации состояния экземпляров класса
18. Конструкторы класса. Цепочки конструкторов
19. Основная концепция архитектурного шаблонаMVC (Model-View-Controller). Преимущества и недостатки использования шаблона MVC
20. Основы унифицированного языка моделирования UML для проектирования классов и объектов, а также их взаимодействия
21. Проектирование UML-диаграммы классов. Отображение связей между классами: ассоциация, композиция, агрегация, наследование
22. Сокрытие реализации в ООП. Инкапсуляция. Способы и средства, обеспечивающие поддержку инкапсуляции в C#
23. Наследование в ООП. Множественное и единственное (одиночное) наследование. Реализация наследования в C#
24. Полиморфное поведение в ООП. Переопределение (overriding) методов как способ реализации полиморфизма в C#
25. Правила приведения классов. Ключевые слова as и is

26. Абстракция (абстрагирование) в ООП. Абстрактные классы и интерфейсы в C#
27. Понятие методов-индексаторов в C#
28. Статические члены класса в C#
29. Статические и вложенные классы в C#. Методы расширения
30. Структуры в C#. Поведение структур
31. Перечислимый тип в C#
32. Понятие упаковки и распаковки в C#
33. Делегаты. Комбинированные (групповые) делегаты
34. Анонимные методы. Лямбда операторы и лямбда выражения
35. Универсальные типы (обобщения) в C#
36. Ковариантность и контравариантность обобщений
37. Понятие типов, допускающих null, в C#
38. Ограничения обобщенных типов
39. Преимущества использования обобщенных коллекций
40. События. Объявление, генерация событий. Подписка на событие
41. Многопоточность в C# и .NET
42. Обработка ошибок и исключительных ситуаций в C#
43. Родительский главный класс System.Object
44. Перегрузка операторов. Перегрузка операторов явного и неявного преобразования типа
45. Анонимные и динамические типы в C#
46. Язык запросов Language Integrated Query (LINQ)