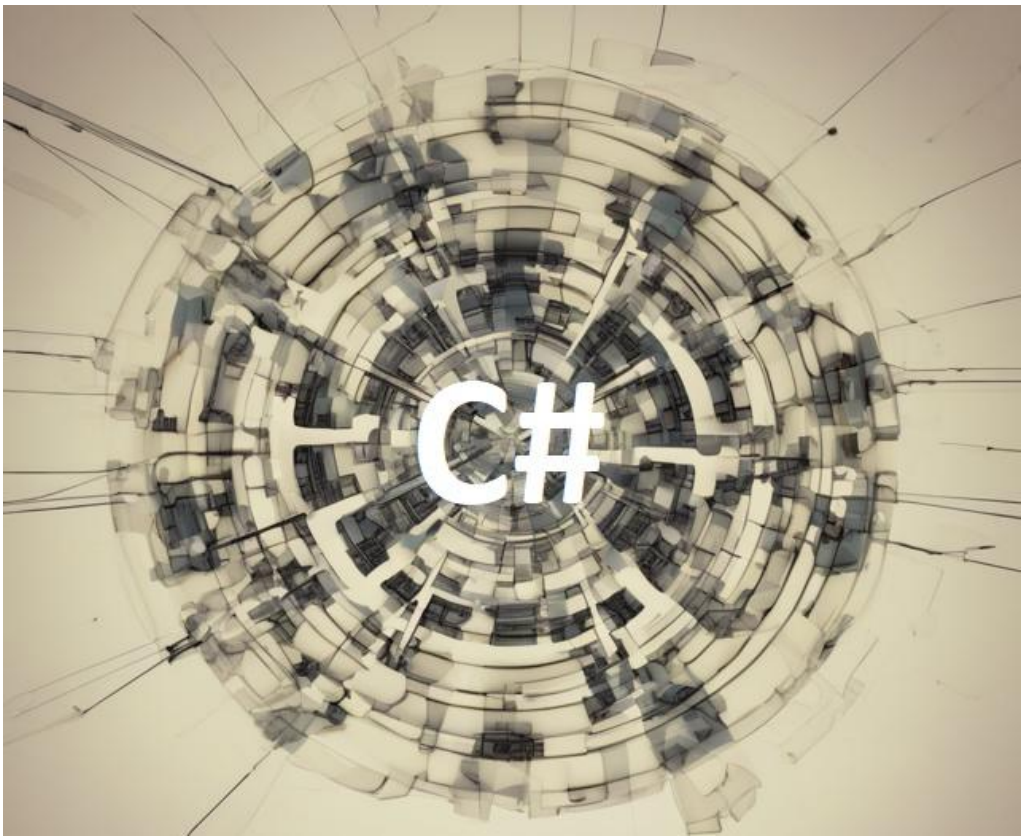
С. Г. ТОЛСТЫХ, И. Л. КОРОБОВА, Н. В. МАЙСТРЕНКО

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ C#



Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2023

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Тамбовский государственный технический университет»

С. Г. ТОЛСТЫХ, И. Л. КОРОБОВА, Н. В. МАЙСТРЕНКО

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ C#

Утверждено Ученым советом университета в качестве учебного пособия для бакалавров 2, 3 курсов направления подготовки 09.03.01 «Информатика и вычислительная техника», изучающих дисциплину «Объектно-ориентированное программирование», очной и заочной форм обучения

Учебное электронное издание

Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2023

УДК 004.432.2
ББК 32.972.13
Т52

Рецензенты:

Кандидат педагогических наук, доцент,
доцент кафедры «Математическое моделирование
и информационные технологии» Института математики,
физики и информационных технологий
ФГБОУ ВО «ТГУ имени Г.Р. Державина»
Е. В. Клыгина

Кандидат технических наук,
доцент кафедры «Информационные процессы и управление»
ФГБОУ ВО «ТГТУ»
И. А. Дьяков

Толстых, С. Г.

Т52 Объектно-ориентированное программирование с использованием C# [Электронное издание] : учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2023. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод ; 3,26 Mb ; RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана.

ISBN 978-5-8265-2615-6

Рассматриваются основные понятия о языке программирования C#, концепция объектно-ориентированного программирования с использованием C#, приводятся общие сведения по разработке графических приложений.

Предназначено для бакалавров 2, 3 курсов направления подготовки 09.03.01 «Информатика и вычислительная техника», изучающих дисциплину «Объектно-ориентированное программирование», очной и заочной форм обучения.

УДК 004.432.2
ББК 32.972.13

ISBN 978-5-8265-2615-6

ВВЕДЕНИЕ

Язык программирования C# − это современный объектно-ориентированный язык. Необходимость его создания была вызвана рядом проблем, возникающих при использовании C и C++:

– сложности в переносимости и выполнении на рабочих местах пользователей;

– отсутствие проверки программного кода до его компиляции;

– существенная сложность изучения для новичков;

– сложность в поиске и отладке ошибок без использования дорогих отладчиков;

– довольно длительный процесс разработки и высокие риски при модификации;

– слишком сложная и неудобная взаимосвязь с современными базами данных.

В связи с этим C# разрабатывался для получения возможности:

– создания одновременно простых в использовании и мощных по возможностям инструментов для разработки взаимодействующих, масштабируемых и надежных пользовательских приложений;

– обеспечения легкого перехода на C# разработчиков, владеющих C и C++;

– создания полностью объектно-ориентированной современной архитектуры;

– обеспечения мощной компонентно-ориентированной поддержки разработки;

– обеспечение применения возможностей C++ в рамках простого в использовании и быстрого инструментария разработки, как в Visual Basic;

– написания приложений не только для настольных компьютеров, но и для мобильных устройств.

1. ПОНЯТИЕ О C#, ОПИСАНИЕ ТИПОВ ДАННЫХ

1.1. ПРОГРАММНАЯ ПЛАТФОРМА MICROSOFT.NET FRAMEWORK

Для обеспечения легкой переносимости приложений, написанных на различных языках программирования, включая C#, на самые разнообразные компьютеры пользователей была разработана новая платформа – Microsoft.NET Framework. Разработанные для данной платформы приложения получили возможность применения сетевых технологий и Интернета.

Задачами создания .NET Framework являлись:

– минимизация усилий при развертывании системного и пользовательского программного обеспечения;

– обеспечение современной и полной объектно-ориентрованной среды программирования;

– снижение конфликтов между версиями и обеспечение безопасного выполнения программного кода;

– обеспечение унификации принципов разработки и отладки для различных типов Windows- и Web-приложений.

В структуру .NET Framework входят два основных компонента: Общеязыковая исполняющая среда – CLR (Common Language Runtime) и Библиотека классов .NET Framework – FCL(Framework Class Library).

CLR выполняет различные вспомогательные функции, обеспечивающие удобство процесса разработки и отладки приложений. К ним относятся: управление памятью, выполнение кода, обработка ошибок, проверка безопасности кода, сборка мусора (автоматическое освобождения памяти).

FCL представляет собой всеобъемлющую объектно-ориентированную библиотеку классов. Она задает набор примитивных типов и классов, позволяющих разрабатывать приложения различного уровня, от консольных до приложений с графическим интерфейсом пользователя – GUI (Graphical User Interface) с возможностью использования в интернете. При этом программисты, разрабатывающие приложения, могут использовать любой из удобных для них языков, поддерживаемых .NET.

Помимо CLR и FCL в .NET Framework входят и другие важные компоненты:

– Windows Forms. Предоставляют набор классов для разработки форм Windows-приложений, включающих стандартные компоненты GUI.

– Web Forms. Предоставляют набор классов для разработки форм Web-страниц.

– Web Services. Включают в себя набор классов для разработки приложений, включающих стандартный набор сетевых протоколов.

– ASP.NET. Предоставляет набор классов для создания Web-приложений по технологии ASP.

– ADO.NET. Предоставляет классы для взаимодействия с базами данных.

– XML Classes. Классы, предоставляющие возможность задания комментариев к приложению в формате XML с последующей манипуляцией ими.

– Base Framework Classes. Базовые классы, обеспечивающие основную функциональность, такую как ввод/вывод, операции со строками, управление безопасностью, сетевые подключения и т.д.

– Common Language Specification. Общие спецификации языка представляют собой набор правил для любого языка .NET, которому он должен следовать, чтобы создавать приложения, совместимые с другими языками.

– Common Type System. Общая система типов описывает объявление типов данных, используемых и управляемых в системе исполнения, что облегчает использование типов в случае различных языков программирования.

В платформе .NET каждый из языков программирования может иметь собственный компилятор и среду выполнения. Но его собственный компилятор преобразует исходный код в исполняемый код на едином промежуточном языке, который может быть запущен пользователем при наличии на его компьютере .NET Framework соответствующей версии. Такой промежуточный код получил название Microsoft Intermediate Language (MSIL). Он состоит из специального набора инструкций, которые указывают на способ выполнения кода.

При первом выполнении код MSIL преобразуется в код известный для операционной системы. Это называется компиляцией по требованию (Just-In-Time – JIT), которая входит в структуру CLR. Таким образом, CLR предоставляет некую виртуальную машину для .NET приложений, которая используется для преобразования кода MSIL в машинный код, понятный и выполнимый процессором компьютера.

Контрольные вопросы

1. Перечислите цели создания языка C#.
2. Какие задачи были решены с помощью среды .NET Framework?
3. Что входит в структуру .NET Framework?
4. Для чего используется Microsoft Intermediate Language?

1.2. ТИПЫ ДАННЫХ C#

Типы данных в C#, как и в C++, можно поделить на две группы: типы-значения и ссылочные типы. Стандартные типы значений имеют предопределенный диапазон и размер, приведенный в табл. 1.

Классификация ссылочных типов представлена ниже:

– Объект (object) – встроенный тип данных, являющийся базовым классом для всех предопределенных и определяемых пользователем типов данных.

– Строка (string) – встроенный тип, ссылающийся на набор значений из символов Unicode, заключенных в двойные кавычки.

– Класс – логическая структура, которая отражает некоторый объект окружающего мира, подлежащий информатизации.

– Интерфейс – особый тип определяемого пользователем класса, который может использоваться для множественного наследования, но не имеет объектов.

– Делегат – определяемый пользователем ссылочный тип, который может ссылаться на один или более методов.

– Массив – определяемая пользователем многомерная структура данных, которая ссылается на набор значений какого-либо типа данных.

1. Размер и диапазон значений типов данных в C#

Тип данных	Размер	Диапазон
Bool	8-битовое логическое	true/false (истина/ложь)
Byte	8-битовое беззнаковое целое	[0, 255]
Short	16-битовое знаковое целое	[–32 768, 32 767]
Int	32-битовое знаковое целое	$[-2^{31}, 2^{31}]$
Long	64-битовое знаковое целое	$[-2^{63}, 2^{63}]$
Float	32-битовое с плавающей точкой с точностью 7 значащих цифр	$[10^{-45}, 10^{38}]$
Double	64-битовое с плавающей точкой с точностью 15-16 значащих цифр	$[10^{-324}, 10^{308}]$
Decimal	128-битовое с фиксированной точкой с точностью 28–29 значащих цифр	$[10^{-28}, 10^{28}]$
Char	16-битовый символ Unicode	[U+0000, U+ffff]

На переменные и константы в C# можно назначать литералы шести типов:

– Булевы литералы имеют два значения – true и false. Например, boolval = true.

– Целочисленные литералы могут быть назначены на типы данных int, uint, long и ulong. Суффиксы для целого числа литерала включают U, L или UL. U – беззнаковое целое, L – длинное целое. Например, longval = 223L.

– Вещественные литералы назначаются на двоичные и десятичные типы данных с плавающей и фиксированной точкой. Они обозначаются суффиксами F, D и M. F соответствует числу типа float, D обозначает тип double, M обозначает тип decimal. Например, floatval = 2.23F.

– Символьный литерал назначается типу данных char. Символьный литерал должен быть заключен в одиночные кавычки. Например, charval = `A`.

– Строковый литерал. В C# имеются два типа строковых литералов: регулярные и дословные. Регулярный строковый литерал – обычный литерал, состоящий из набора символов Unicode, заключенных в двойные кавычки. Дословный строковый литерал аналогичен регулярному строковому литералу, но предварительно установлен символ @. Например, stringPath = “c:\file.txt”. Дословные литералы удобно использовать для указания текста, содержащего символы управляющей последовательности (в примере это символ «\»).

– Нулевой литерал может быть только null. Например, stringPath = null.

Контрольные вопросы

1. Назовите типы данных языка C#.
2. Чем ссылочные типы отличаются от типов данных?
3. Что такое делегат в C#?
4. Перечислите типы литералов, применяемых в C#.

2. ОСНОВНЫЕ КОНСТРУКЦИИ ДЛЯ ПРОГРАММИРОВАНИЯ В C#

2.1. КОНСОЛЬНЫЕ ОПЕРАЦИИ

Консольные операции удобно применять в приложениях, так как они не требуют организации пользовательского интерфейса (GUI) и зависят от устройств ввода и вывода компьютерной системы. Консольные приложения состоят из трех потоков, которые присоединены к устройствам ввода и вывода и сообщениям об ошибках на устройстве вывода.

Методы консольного ввода/вывода в C# сгруппированы в пространстве имен Console. Подробнее о понятии «Пространство имен» в C# далее в 1.10.

К методам консольного вывода информации относятся:

– Console.Write() –выводят указанные данные на экран;

– Console.WriteLine() –выводят указанные данные на экран по строкам.

Синтаксис: Console.Write(“<СтрокаДанных>” + <Переменные>),

где <СтрокаДанных> – определяет строки и символы управляющей последовательности, заключенные в двойные кавычки; <Переменные> – определяет переменные, значения которых должны быть выведены на консоль.

Маркерами {0}, {1} и т.д. можно указать место подстановки значений первой, второй и т.д. переменной в управляющую последовательность.

Методы ввода информации из консоли (напр., клавиатуры):

– Console.Read() – считывает единичный символ;

– Console.ReadLine() – считывает строку символов.

Пример вывода на экран «Введите имя» и его считывания в переменную:

```
Console.WriteLine(“Введитеимя”);
stringname = Console.ReadLine().
```

Для преобразования введенных данных к тому или иному типу можно использовать методы из пространства имен System.Convert. Например,

```
intheight = Convert.ToInt32(Console.ReadLine());
```

преобразует введенный пользователем текст с весом человека к переменной «height»типа «int».

Упаковка – это процесс, предусмотренный для преобразования типа значения, например целых чисел, к его базовому ссылочному типу «object». Это

может применяться, например, для уменьшения нагрузки на систему во время выполнения. Все системные типы значений имеют неявный тип «object».

Пример явной упаковки:

```
float R = 4.5F; double dlina; dlina = 2 * 3.14 * R;
object dlina_boxed = (object)dlina;
```

В процессе обратного распаковывания происходит преобразование ссылочного типа «object» к типу значения. При этом фактические значение типаobject должно совпадать с типом значения адресата. Это явное преобразование, без которого компиляторвыдаст ошибку. Пример распаковки:

```
intdlina = 5; intshirina = 10; intploshad;
ploshad = dlina * shirina;
object ploshad_boxed = ploshad;
intploshad_unboxed = (int)ploshad_boxed;
```

Контрольные вопросы

1. Для чего применяются консольные операции?
2. Каким образом в C# реализуется консольный вывод?
3. Как осуществить считывание информации, введенной в консоли?
4. Для чего применяется упаковка данных?

2.2. УТВЕРЖДЕНИЯ И ОПЕРАТОРЫ

Утверждение – логически сгруппированные переменные, операторы и ключевые слова C #, которые выполняют поставленную задачу. Оно, как и в C++, должно заканчиваться символом «;». Утверждения можно заключать в блоки с помощью символов «{» и «}», соответственно – начало и конец блока.

Утверждения C# похожи на утверждения в C и C++. Утверждения C# можно разделить на семь категорий в соответствии с функциями, которые они выполняют:

– Утверждения выбора – это утверждения решения типа истина/ложь для заданных условий. Для них, как и в C++, применяются конструкции с ключевыми словамиif, else, switch и case.

– Утверждения повторения позволяют повторно выполнить заданный блок кода. Здесь используются ключевые слова циклов do, for, foreach, in и while.

– Утверждения перехода позволяют передавать поток управления от одного блока кода программы к другому. При этом используются такие ключевые слова, как break, continue, default, goto, return и yield.

– Утверждения обработки исключений предназначены для принятия решения по управлению потоком выполнения программы при возникновении нештатных ситуаций. Здесь применяются ключевые слова throw, try-catch-finally.

– Проверяемые и непроверяемые утверждения управляют арифметическим переполнением, когда результирующее значение больше, чем диапазон целевого типа данных переменной. Проверяемое утверждение останавливает выполнение программы, а непроверяемое утверждение содержит данные остатка на целевую переменную. Применяются слова checked и unchecked.

– Утверждения установки предназначены для управления действиями сборщика мусора, указывая, что не нужно перемещать объект во время выполнения. Ключевыми словами для этого служатfixed и unsafe.

– Утверждение блокировки помогает изолировать выполнение важных блоков программы от вмешательства других потоков или процессов в машинной памяти. Оно работает только со ссылочными типами. Ключевое слово для этого lock.

Операторы C# почти совпадают с аналогичными операторами языка C++.К ним относятся следующие категории: арифметические операторы, операторы отношения, логические операторы, условные операторы, операторы приращения и декремента, операторы присваивания. Синтаксис конструкций выбора if…else и switch…case, циклов while, do..while, for также аналогичен применяемым в C++.

Отличием C# является добавление нового типа цикла foreach, в котором осуществляется перемещение по каждому значению из указанного списка и выполнение для него блока заданных операторов. Число значений в списке определяет, сколько раз цикл foreach будет выполняться. Каждое значение в списке упоминается как элемент, причем они доступны только для чтения:

```
foreach (<ТипДанных><Идентификатор>in<Список>)
{ ; // одно и более выражений, применяемых к списку }
```

где <ТипДанных> – тип данных элементов в списке;<Идентификатор> – имя для обращения к элементам набора; <Список> – имя списка, содержащего данные для обработки в цикле.

Следующий код отображает имена студентов при помощи цикла foreach:

```
public string[] studentNames = {"Маша", "Вася", "Миша", "Таня"};
Console.WriteLine ("Имена студентов");
foreach (string names in studentNames)
{     Console.WriteLine("{0} ", names); }
```

Результат: Имена студентов Маша Вася Миша Таня

С # поддерживает четыре типа операторов перехода: break, continue, goto, return. Их применение также аналогично С++.

Контрольные вопросы

1. Какие символы в С# используются для выделения блоков операторов?
2. Перечислите операторы циклов в С#.
3. Когда удобно использовать оператор «foreach»?

2.3. МАССИВЫ

Работа с массивами в С# схожа с работой в С++. Но есть и ряд отличий. Как и в С++, здесь для размещения памяти под массив используется ключевое слово new. Отличие в том, что освобождение памяти, занятой уже ненужными массивами, происходит автоматически сборщиком мусора из CLR.NET Framework. Если значения явно не указаны при размещении памяти, им автоматически будут присвоены значения по умолчанию в соответствии с их типом данных: int 0, float 0.0, double 0.0 ,char `\0` , stringnull.

В С# поддерживается работа с одномерными, многомерными (как в С++), а также с зубчатыми массивами. Зубчатый массив – это многомерный массив, где у одной из указанных размерностей могут быть переменные размеры. У зубчатых массивов может быть неравное число столбцов для каждой строки.

Синтаксис прямоугольного массива:

```
type[,] <arrayName>; //объявление
arrayName = newtype[value1 , value2]; //инициализация
```

где type – тип данных элементов массива; <arrayName> – имя массива; value1 – определяет число строк; value2 – определяет число столбцов.

Если в квадратных скобках рядом с типом данных (type) массива стоит одна и более запятых, это означает наличие в массиве 2 – 8 размерностей. В тексте, изображенном выше, одна запятая соответствует 2 размерностям массива.

Следующий фрагмент кода иллюстрирует использование зубчатого массива для хранения данных о студентах.

```
string[][] students = new string[3][];
students[0] = new string[] {“Маша”, “Петя”};
students[1] = new string[] {“Вася”, “Оля”, “Света”};
students[2] = new string[] {“Коля”, “Галя”, “Таня”, “Ваня”};
for (int i=0; i < students.GetLength(0); i++)
{
    Console.Write(“List of students in group “ + (i+1) + “:\t”);
    for (int j=0; j< students[i].GetLength(0); j++)
    {
        Console.Write(students[i][j] + “ “);
    }
 Console.WriteLine();
}
```

Класс «Array» – встроенный класс из пространства имен «System». Он является базовым классом для всех массивов в C #. В нем реализованы методы для стандартных задач создания, поиска, копирования и сортировки массивов. Для создания массива используется метод «CreateInstance». Методы «SetValue» и «GetValue» служат для изменения или считывания элементов массива.

Контрольные вопросы

1. Какие типы массивов поддерживаются в C#?
2. Опишите синтаксис объявления двумерного массива?
3. Как реализовано освобождение памяти, занимаемой массивом, когда он уже не нужен?

2.4. Управление исключениями

Управление исключениями в C# реализовано с помощью блока try-catch.

Синтаксис следующий:

try { // код программы } catch [(<КлассОшибки><objException>)] {// код обработчика ошибки }

<КлассОшибки> – имя класса исключений (ошибок). Он является опциональным и позволяет обрабатывать каждый тип ошибок в отдельности. Общий блок catch с неуказанным классом исключения обрабатывает все типы ошибок, если иное (отдельные catch-обработчики) не указано. В этом случае классом ошибки является класс «Exception» из пространства «System.Exception».

Оператор «throw» в C# позволяет программно генерировать исключения. Он принимает экземпляр соответствующего класса исключений в качестве параметра. Например, `throw new DivideByZeroException();`. Класс System. DivideByZeroException соответствует случаю ошибки деления на ноль.

Если любой из операторов в блоке try вызывает исключение, блок catch выполняется, и остальная часть операторов в блоке try игнорируется. Если нужно выполнить некоторые операции независимо от наличия исключений в try, применяется блок finally.

Пример:

```
class DivisionError { static void Main(string[] args)
{
    int numOne = 12;
    int numTwo = 0;
    int result;
    try {
          result = numOne / numTwo;
        }
    catch (DivideByZeroException objDivide)
    {
        Console.WriteLine("Тип ошибки: " + objDivide);
    }
    finally { Console.WriteLine("Этот блок всегда выполняется"};
    }
}
```

Блок try может состоять из множества вложенных конструкций catch-try. Если внутренний блок try генерирует исключение, управление передается к внутреннему блоку catch. Однако, если внутренний блок catch не содержит соответствующего обработчика ошибок, управление передается на внешний catch.

Контрольные вопросы

1. Для чего нужна обработка исключений?
2. Как в программе можно сгенерировать исключение?
3. Какой системный класс является базовым для всех типов исключений?

3. КОНЦЕПЦИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В C#

3.1. КЛАССЫ И МЕТОДЫ

Концепция объектно-ориентированного программирования (ООП) в C# основывается на понятиях абстракции, инкапсуляции, наследования и полиморфизма из C++. В данном разделе пособия ограничимся описанием лишь новинок и отличий, появившихся именно в C#.

Ключевое слово «ref» заставляет параметры быть переданными по ссылке в вызванный метод. В вызванном методе данный параметр будет соответствовать тому же блоку памяти, что и в вызывающем методе (методе-запросе). Причем оба эти методы должны объявить данный параметр с ключевым словом «ref». Аналогичным образом может быть использовано ключевое слово «out», но при этом от переменных, которые будут переданы по ссылке, не требуется инициализация.

```
class OutClass { static void OutMethod(out int val)
{
    val = 1;
    int OutPole = val * 100;
    Console.WriteLine("Out Pole: " + OutPole);
 }
    static void Main(string[] args)
    {
        int value;
        OutMethod(out value);
    }
}
```

ВC# у двух методов может быть одно и то же название, но различные параметры (по типу или количеству). В итоге при вызове метода происходит перегрузка, определяющая, какой именно метод класса следует вызвать в данном случае (при текущих передаваемых данных). Возвращаемое значение не является частью сигнатуры метода, влияющей на ее перегрузку.

Ключево слово «this» используется для ссылки на текущие объекты класса. Оно применяется для разрешения конфликтов между переменными, имеющими одинаковые имена и передающими текущий объект в качестве параметра. Нельзя использовать ключевое слово this со статическими переменными и методами.

Конструктор – это метод, имеющий то же самое имя, что и класс. Конструкторы могут инициализировать переменные класса или выполнить операции запуска только однажды при создании объекта класса. Они автоматически выполняются всякий раз, когда создан экземпляр класса.

У конструкторов нет никакого явного типа возвращаемого значения, так как неявный тип возвращаемого значения – сам класс. Аналогично методам класса, конструкторы можно перегружать.

Если никакой конструктор не определен в пределах класса, автоматически создается заданный по умолчанию конструктор, который инициализирует все числовые переменные экземпляра класса нулевыми значениями. Если же в классе был определен конструктор, то конструктор по умолчанию не используется.

Статический конструктор (обозначенный ключевым словом «static») используется, чтобы инициализировать статические переменные класса и выполнить специфическое действие только однажды. Он вызывается прежде, чем к любому статическому члену класса обращаются. В классе может быть только один статический конструктор. Также такой конструктор не может иметь модификаторов доступа и параметров.

Деструктор – специальный метод, который имеет то же самое название, как и класс, но начинается с символа «~» перед именем класса. Деструкторы немедленно освобождают память об объектах, которые больше не требуются. В отличие от C++, они вызываются автоматически сборщиком мусора, когда объекты больше не используются. Можно самостоятельно определить только один деструктор в классе. Кроме того, деструкторы не могут: быть перегружены и унаследованы, быть явно вызваны, определить модификаторы доступа, иметь параметры.

Классы могут быть распределены по множественным местоположениям, чтобы хранить различные члены. Для этого применяется ключевое слово «partial». Это удобно, если класс имеет сложную структуру или разрабатывается несколькими программистами. Отдельные части определения объединяются во время сборки.

Пример 1.

Создать новый класс Tel, содержащий поля день разговора (задается целым числом в интервале от 1 до 7), время разговора (вещественное число) и стоимость (вещественное число). Создать новый метод для подсчета стоимости телефонного разговора в зависимости от дня недели: в выходные дни стоимость разговора меньше на 10%.

Код программы

```
namespace ConsoleApplication1
{
    Ссылок: 3
    public class Tel
    {
        public int день;
        public double время;
        public double стоимость;

        ссылка: 1
        public static double Stoim(int d, double v, double s)
        {
            double stoim;
            if ((d == 6) || (d == 7)) stoim = 0.9 * v * s;
            else stoim = v * s;
            return (stoim);
        }
    }

    Ссылок: 0
    class Program
    {
        Ссылок: 0
        static void Main(string[] args)
        {
            Tel t1 = new Tel();
            Console.Write("Введите день разговора\n1 - пн\n2 - вт\n3 - " +
                "ср\n4 - чт\n5 - пт\n6 - сб\n7 - вс\n ");
            t1.день = Convert.ToInt32(Console.ReadLine());
            Console.Write("Введите время разговора ");
            t1.время = Convert.ToDouble(Console.ReadLine());
            Console.Write("Введите стоимость разговора ");
            t1.стоимость = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Стоимость разговора составляет = " +
                "{0,5:####0.00} рублей", Tel.Stoim(t1.день, t1.время, t1.стоимость));
        }
    }
}
```

Результат

```
Введите день разговора
1 - пн
2 - вт
3 - ср
4 - чт
5 - пт
6 - сб
7 - вс
 6
Введите время разговора 25
Введите стоимость разговора 10
Стоимость разговора составляет = 225,00 рублей
Для продолжения нажмите любую клавишу . . .
```

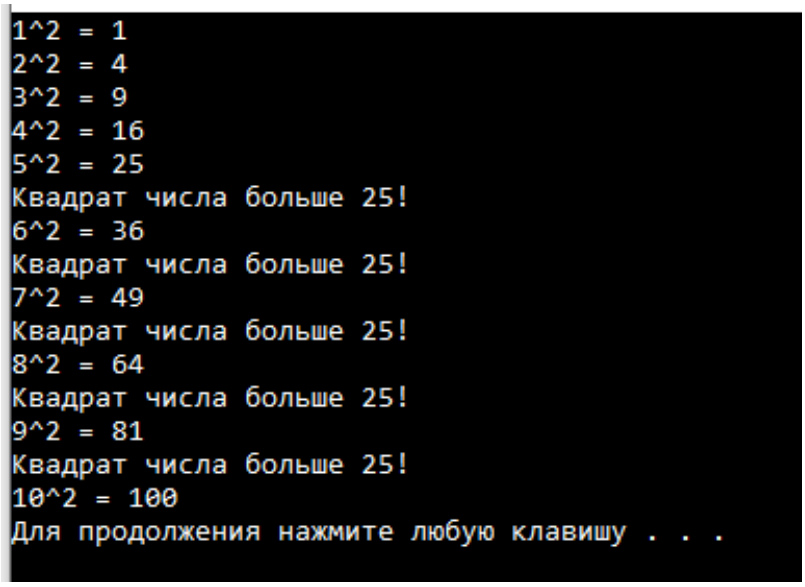
Пример 2.

Определено два класса. Первый класс будет находить квадрат числа от 1 до 10, используя цикл. Другой класс будет ждать, когда в первом классе квадрат числа станет больше 25, и после этого выведет в консоль фразу: «Квадрат числа больше 25!».

Код программы

```
7   namespace ConsoleApplication1
8   {
        Ссылок: 2
9       class kv
10      {
11          public delegate void MyDelegate();
12          public event MyDelegate MyEvent;
13
            ссылка: 1
14          public void S()
15          {
16              int i, ss;
17              for (i = 1; i <= 10; i++)
18              {
19                  ss = i * i;
20                  if (ss > 25) MyEvent();
21                  Console.WriteLine("{0}^2 = {1}", i, ss);
22              }
23
24          }
25      }
26
27      class Message
28      {
            ссылка: 1
29          public void Mes()
30          {
31              Console.WriteLine("Квадрат числа больше 25!");
32          }
33      }
        Ссылок: 0
34      class Program
35      {
            Ссылок: 0
36          static void Main(string[] args)
37          {
38              kv kv1 = new kv();
39              Message message1 = new Message();
40              kv1.MyEvent += message1.Mes;
41              kv1.S();
42
43          }
44      }
45  }
```

Результат



Контрольные вопросы

1. Опишите понятие класса в C#.
2. Для чего используется зарезервированное слово «this»?
3. Что представляет собой передача параметров метода класса по ссылке?
4. Как можно разделить описание класса на части?

3.2. НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

ВC# для наследования класса, как и в C++, используется следующий синтаксис: <ИмяКлассаПотомка> : <ИмяБазовогоКласса>.

Для вызова метода базового класса в классе-потомке синтаксис аналогичен вызову метода текущего дочернего, но объект должен быть базового класса.

В случае, если методы или свойства базового класса и потомка совпадают, для доступа к элементам базового класса используется ключевое слово «base».

В C# применяются четыре модификатора доступа: public (открытый для наследования), protected (возможность наследования только классами-потомками), private (защищенный от наследования), internal (наследование в пределах одной сборки).

В C# существует возможность отмены метода, что позволяет производному классу отменять или переопределять методы базового класса. Для этого в базовом классе при описании метода следует указать ключевое слово «virtual»: <МодификаторДоступа> virtual <ТипВозвращаемогоЗначения> <ИмяМетода> (<СписокПараметров>).

В методе класса-потомка следует применить ключевое слово «override» при описании метода, заменяющего соответствующий метод базового класса: <МодификаторДоступа> override <ТипВозвращаемогоЗначения> <ИмяМетода> (<СписокПараметров>).

Пример:

```
 class Animal
  {
      public virtual void Eat()
     {
        Console.WriteLine("Все животные что то едят");
     }
      protected void DoSomething()
     {
         Console.WriteLine("Все животные что то делают");
     }
  }
class Cat:Animal
{// заменяет метод Eat() класса Animal
     public override void Eat()
  {
      Console.WriteLine("Кошки любят есть мышей");
  }
static void Main(string[] args)
{
     Cat objCat = new Cat();
     objCat.Eat();
}
}
```

Если производный класс пытается отменить невиртуальный метод, компилятор языка C# сгенерирует ошибку.

Запечатанный класс – класс, который предотвращает наследование. Для этого перед именем класса используется ключевое слово «sealed». У запечатанного класса не может быть никаких protected членов.

Концепция полиморфизма в C# бывает двух типов: полиморфизм во время компиляции и полиморфизм во время выполнения. Первый случай (статический полиморфизм) реализуется с помощью перегрузки методов с помощью отличий в сигнатуре методов. Полиморфизм во время выполнения (или динамический полиморфизм) реализуется с помощью процедуры отмены метода (указание методов базового класса и класса-потомка с ключевыми словами «virtual» и «override» соответственно), описанной выше.

Пример 1.

Создать базовый класс Student, который будет содержать информацию о студенте (фамилия, курс обучения, номер зачетной книги). С помощью механизма наследования реализовать класс Aspirant.

```
namespace ConsoleApplication1
{
    // Базовый класс Student, содержит информацию о студенте
    Ссылок: 5
    public class Student
    {
        // 1. Поля класса – объявленные как protected – видимы в производных классах,
        // и невидимы из экземпляра класса
        protected string name; // Фамилия и имя студента
        protected int course; // Курс обучения
        protected string gradeBook; // Номер зачетной книги

        // 2. Конструктор класса с 3 параметрами
        Ссылок: 2
        public Student(string Name, int course, string gradeBook)
        {
            this.Name = Name;
            this.course = course;
            this.gradeBook = gradeBook;
        }

        // 3. Свойства доступа к полям класса
        Ссылок: 2
        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        ссылка: 1
        public int Course
        {
            get { return course; }
            set { course = value; }
        }

        ссылка: 1
        public string GradeBook
        {
            get { return gradeBook; }
            set { gradeBook = value; }
        }

        // 4. Метод Print() – вывести значения полей на экран
        Ссылок: 2
        public void Print()
        {
            Console.WriteLine("Значения полей:");
            Console.WriteLine($"Имя = {name}");
            Console.WriteLine($"Курс = {course}");
            Console.WriteLine($"Зачетка = {gradeBook}");
        }
    }

    // Класс Aspirant – наследует возможности класса Student
    Ссылок: 3
    public class Aspirant : Student
    {
```

```
58          // 1. Внутреннее поле класса
59          protected string topic; // Тема кандидатской диссертации
60
61          // 2. Конструктор класса Aspirant - с помощью ключевого слова base обращается
62          // к конструктору базового класса Student
            ссылка: 1
63          public Aspirant(string name, int course, string gradeBook, string topic) :
64              base(name, course, gradeBook)
65          {
66              // Можно изменять protected-члены базового класса
67              base.name = name; // доступ к полю name класса Student с помощью base
68              this.course = course; // доступ к полю course класса Student с помощью this
69              this.gradeBook = gradeBook;
70
71              this.topic = topic; // инициализация внутреннего поля класса Aspirant
72          }
73
74          // 3. Свойство для доступа к полю topic
            Ссылок: 0
75          public string Topic
76          {
77              get { return topic; }
78              set { topic = value; }
79          }
81          // 4. Метод Print() - печать полей класса Aspirant
82          // Имя данного метода перекрывает имя метода Student.Print(),
83          // поэтому перед именем метода указывается new
            ссылка: 1
84          public new void Print() // new - переопределение метода базового класса
85          {
86              base.Print(); // вызвать метод Print() базового класса
87              Console.WriteLine($"Тема кандидатской диссертации = {topic}");
88          }
89      }
90
            Ссылок: 0
91      class Program
92      {
                Ссылок: 0
93          static void Main(string[] args)
94          {
95              // Демонстрация работы с классами Student и Aspirant
96
97              // 1. Объявить экземпляр класса Student
98              Student st1 = new Student("Иванов С.А.", 2, "0519");
99
100             // 2. Вывести поля класса Student
101             Console.WriteLine("Вывод поля класса Student:");
102             st1.Print();
103
104             // 3. Объявить экземпляр класса Aspirant
105             // При объявлении используется свойство get экземпляра st1
106             Aspirant asp1 = new Aspirant(st1.Name, st1.Course, st1.GradeBook, "Hello world!");
107
108             // 4. Вызвать метод Print() экземпляра asp1
109             Console.WriteLine("--------------------");
110             Console.WriteLine("Вывод поля класса Aspirant:");
111             asp1.Print();
112         }
113     }
114 }
115
```

Результат

```
Вывод поля класса Student:
Значения полей:
Имя = Иванов С.А.
Курс = 2
Зачетка = 0519
--------------------
Вывод поля класса Aspirant:
Значения полей:
Имя = Иванов С.А.
Курс = 2
Зачетка = 0519
Тема кандидатской диссертации = Hello world!
Для продолжения нажмите любую клавишу . . .
```

Пример 2.

Создать базовый класс точка на плоскости. С помощью механизма наследования реализовать классы точка в пространстве, отрезок на плоскости и треугольник.

Код программы

```
7   namespace ConsoleApp2
8   {
        Ссылок: 7
9       class DemoPoint
10      {
11          protected int x;
12          protected int y;
                ссылка: 1
13          public void Show()
14
15          {
16              Console.WriteLine("точка на плоскости: ({0}, {1})", x, y);
17          }
18
                Ссылок: 3
19          public DemoPoint(int x, int y)
20          {
21              this.x = x; this.y = y;
22          }
23      }
        Ссылок: 3
24      class DemoShape : DemoPoint
25      {
26          protected int z;
                ссылка: 1
27          new public void Show()
28          {
29              Console.WriteLine("точка в пространстве: ({0}, {1}, {2})", x, y, z);
30          }
                ссылка: 1
31          public DemoShape(int x, int y, int z) : base(x, y)
32          {
33              this.z = z;
34          }
35      }
```

```
36      class DemoLine : DemoPoint
37      {
38          protected int x2;
39          protected int y2;
            ссылка: 1
40          new public void Show()
41          {
42              Console.WriteLine("отрезок на плоскости: ({0}, {1})-({2},{3})", x, y, x2, y2);
43          }
            Ссылок: 2
44          public DemoLine(int x1, int y1, int x2, int y2) : base(x1, y1)
45          {
46              this.x2 = x2; this.y2 = y2;
47          }
48      }
        Ссылок: 3
49      class DemoTriangle : DemoLine
50      {
51          protected int x3;
52          protected int y3;
            ссылка: 1
53          new public void Show()
54          {
55              Console.WriteLine("треугольник на плоскости: ({0}, {1})-({2},{3})-({4},{5})",
56                  x, y, x2, y2,x3, y3);
57          }
            ссылка: 1
58          public DemoTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
59              : base(x1, y1, x2, y2)
60          {
61              this.x3 = x3;
62              this.y3 = y3;
63          }
64      }
65      class Program
66      {
            Ссылок: 0
67          static void Main()
68          {
69              DemoPoint point = new DemoPoint(1, 1);
70              point.Show();
71              DemoShape pointShape = new DemoShape(1, 1, 1);
72              pointShape.Show();
73              DemoLine line = new DemoLine(2, 2, 10, 10);
74              line.Show();
75              DemoTriangle triangle = new DemoTriangle(0, 0, 0, 3, 4, 0); triangle.Show();
76          }
77      }
78  }
```

Результат

```
точка на плоскости: (1, 1)
точка в пространстве: (1, 1, 1)
отрезок на плоскости: (2, 2)-(10,10)
треугольник на плоскости: (0, 0)-(0,3)-(4,0)
Для продолжения нажмите любую клавишу . . .
```

Контрольные вопросы

1. Для чего применяется наследование классов?
2. В чем сущность полиморфизма?
3. Поддерживается ли в C# множественное наследование классов?

3.3. Абстрактные классы и интерфейсы

Абстрактный класс – это по сути неполный базовый класс. Он не может иметь объекты, но участвует в процедуре наследования и уточнения классами-потомками, которые в свою очередь уже могут являться обычными классами и иметь объекты. Абстрактный класс может реализовать методы, которые похожи для всех классов-потомков. Кроме того, в нем можно объявить без реализации методы, которые являются различными для классов-потомков. Такие методы также помечаются как абстрактные с помощью ключевого слова «abstract»: <МодификаторДоступа> abstractclass <ИмяКласса> { <МодификаторДоступа> abstract <ТипВозвращаемогоЗначения> <ИмяМетода> (<СписокПараметров>); }.

Пример абстрактного класса Animal:

```
public abstract class Animal
{
    public void Eat()
    {
        Console.WriteLine("Все животные что то едят");
    } //Описание обычного неабстрактного метода
     public abstract void AnimalSound() //Объявление абстрактного метода
}
```

Абстрактный класс не может быть запечатан. Класс-потомок (подкласс), наследующий абстрактный класс, должен отменить и реализовать все абстрактные методы базового класса, иначе компилятор C# посчитает его также абстрактным. Подкласс должен реализовать методы, реализованные в абстрактном классе с тем же самым именем и параметрами.

Подкласс в C# не может наследовать два и более базовых класса. Это происходит из-за того, что C#, в отличие от C++, не поддерживает множественное наследование. Тем не менее, множественное наследование в C# существует и реализовано с помощью интерфейсов.

Интерфейс, в отличие от абстрактного класса, может содержать только абстрактные методы без реализации. Поэтому интерфейс может лишь участвовать в наследовании классами или интерфейсами и не имеет объектов.

Следующий синтаксис используется для объявления интерфейсов:

Interface <ИмяИнтерфейса> { //члены интерфейса }

Пример:

```
interfaceIAnimal{voidAnimalType(); }
```

Реализовывая интерфейс в классе, необходимо, как и при наследовании от абстрактного класса, реализовать все абстрактные методы, объявленные в

интерфейсе. Но, если не все методы реализованы, класс не может быть откомпилирован. Методы, реализованные в классе, должны быть объявлены с тем же самым именем и сигнатурой, как и в интерфейсе.

Следующий синтаксис используется для множественного наследования интерфейсов:

```
class <ИмяКласса> :<ИмяИнтерфейса1>, <ИмяИнтерфейса2>
{ //Реализацияметодовинтерфейсов }
```

Зарезервированное слово «override» не используется при реализации абстрактных методов интерфейса.

Пример объявления и использования интерфейсов:

```
public interface IMyInterface
{
    int MyGetInt(); // метод, возвращающий число типа int
    double MyGetPi(); // метод, возвращающий число Pi
    int MySquare(int x); // метод, возвращающий x в квадрате
    double MySqrt(double x); // метод, возвращающий корень квадратный из x
}
public interface IMyInterface2
{
    double MySqrt2(double x); // корень квадратный из x
}
public class MyClass : IMyInterface, IMyInterface2
{
    // методы из интерфейса MyInterface
    public int MyGetInt()
    {
        return 25;
    }    public double MyGetPi()
    {
        return Math.PI;
    }
    public int MySquare(int x)
    {
        return (int)(x * x);
    }
    public double MySqrt(double x)
    {
        return (double)Math.Sqrt(x);
    }
    // метод из интерфейса MyInterface2
    public double MySqrt2(double x)
    {
        return (double)Math.Sqrt(x);
    }
}
```

Класс должен явно реализовать множественные интерфейсы, если у этих интерфейсов есть методы с идентичными именами. Кроме того, если у интерфейса есть имя метода, идентичное имени метода, объявленного в наследующем классе, этот интерфейс должен быть явно реализован.

Следующая запись используется для явной реализации интерфейса:

class<ИмяКласса> : <ИмяИнтерфейса1>, <ИмяИнтерфейса2>
{ <МодификаторДоступа><ИмяИнтерфейса1>.<ИмяМетода1>(); { }
<МодификаторДоступа><ИмяИнтерфейса2>.<ИмяМетода1>(); { } }

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Например, 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.

Интерфейс IComparable определен в пространстве имен System, содержит единственный метод CompareTo, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра:

interface IComparable
{
intCompareTo(objectobj);
}

Реализация данного метода должна возвращать: 1) 0 – если текущий объект и параметр равны; 2) отрицательное число, если текущий объект меньше параметра; 3) положительное число, если текущий объект больше параметра.

Пример реализации интерфейса IComparable.

```
7   namespace ConsoleApp3
8   {
9       // класс DemoPoint реализует стандартный интерфейс IComparable
            Ссылок: 10
10      class DemoPoint : IComparable
11      {
12          protected int x; protected int y;
                Ссылок: 4
13          public DemoPoint(int x, int y)
14          { this.x = x; this.y = y; }
                ссылка: 1
15          public void Show()
16          { Console.WriteLine("точка на плоскости: ({0}, {1})", x, y); }
                Ссылок: 5
17          public double Dlina()
18          { return Math.Sqrt(x * x + y * y); }
19          //реализация метода CompareTo
                Ссылок: 0
20          public int CompareTo(object obj)
21          {
```

```
22              DemoPoint b = (DemoPoint)obj;   //преобразуем к типу DemoPoint
23                                              //определяем критерии сравнения
24                                              //текущего объекта с параметром
25                                              //в  зависимости от удаленности
26                                              //точки от начала координат
27              if (this.Dlina() == b.Dlina()) return 0;
28              else if (this.Dlina() > b.Dlina()) return 1;
29              else return -1;
30          }
31      }
        Ссылок: 0
32      class Program
33      {
            Ссылок: 0
34          static void Main()
35          {
36              //создаем массив ссылок
37              DemoPoint[] a = new DemoPoint[4];
38              a[0] = new DemoPoint(5, -1); a[1] = new DemoPoint(-3, 3);
39              a[2] = new DemoPoint(3, 4); a[3] = new DemoPoint(0, 1);
40              //сортируем массив точек, при этом в качестве
41              //критерия сортировки будет использоваться
42              //собственная реализация метода CompareTo
43              Array.Sort(a); Console.WriteLine();
44              foreach (DemoPoint x in a)
45              { x.Show(); Console.WriteLine("Dlina={0:f2} ", x.Dlina()); }
46          }
47      }
48  }
```

Результат

```
точка на плоскости: (0, 1)
Dlina=1,00
точка на плоскости: (-3, 3)
Dlina=4,24
точка на плоскости: (3, 4)
Dlina=5,00
точка на плоскости: (5, -1)
Dlina=5,10
Для продолжения нажмите любую клавишу . . .
```

Контрольные вопросы

1. Чем абстрактный класс отличается от обычного класса?
2. Что такое интерфейс?
3. Опишите синтаксис множественного наследования с помощью интерфейсов.

3.4. СВОЙСТВА И МЕТОДЫ СТРОК В C#

Строки в C# – это объекты класса String, значением которых является текст.

Пример:

```
static void Main(string[] args) {
string s = "Hello, World!";
Console.WriteLine(s);}
```

Для объединения строк используется "+".

```
string s = "Hello," + " World!";
```

Оператор "[]" используется для доступа к символу строки по индексу:

```
string s = "Hello, World!"; char c = s[1]; // ' e'
```

Свойство Length возвращает длину строки, s.Length

Метод Compare() – сравнивает строки. Строка "a" <строки "b".

Если строки равны – метод возвращает "0", если первая строка меньше второй – "-1", если первая больше второй – "1".

Методы ToUpper() и ToLower() – переводят строку в верхний/нижний регистр:

Пример:

```
string s = "Hello";
Console.WriteLine(s.ToUpper()); //выводит "HELLO"
Console.WriteLine(s.ToLower()); //выводит "hello"
```

Метод Contains() – проверяет содержит ли строка подстроку. Возвращает True или False. Пример:

```
string s = "Hello, World”;
if (s.Contains("Hello") )
Console.WriteLine("Содержит");
```

Метод IndexOf() возвращает индекс первого вхождения подстроки в строке:

Пример:

```
string s = "Hello, World";
Console.WriteLine(s.IndexOf("World")); //7
Console.WriteLine(s.IndexOf("123") ) ; //-1
```

Методы StartsWith() и EndsWith() проверяют начинается/заканчивается ли строка указанной подстрокой. Возвращает True или False. Пример:

```
string s = "Hello, World";
Console.WriteLine(s.StartsWith("Hello")); //True
Console.WriteLine(s.StartsWith("World")); //False
```

Метод Insert() используется для вставки подстроки в строку, начиная с указанной позиции:

```
Console.WriteLine(s.Insert (5,",")); // вставляет запятую на 5 позицию
```

Метод Replace() – заменяет в строке все подстроки указанной новой подстрокой.

Пример:

```
string s = "Hello, World, Hello";
Console.WriteLine(s.Replace("Hello", "World")); //выведет "World, World, World"
```

Метод Remove() – обрезает строку, начиная с указанной позиции:

Пример:

```
string s = "Hello, World";
Console.WriteLine(s.Remove(5));      // удаляем символы, начиная с 5 позиции, выведется "Hello"
```

Метод Substring() используется получения подстроки из строки, начиная с указанной позиции:

Пример:

```
string s = "Hello, World";
Console.WriteLine(s.Substring(7));   // получаем строку начиная с 7 позиции, выведет "World"
```

Метод ToCharArray() возвращает массив символов указанной строки:

Пример:

```
string s = "Hello, World";
char[] array = s.ToCharArray(); // элементымассива – 'H', 'e', 'l'...
```

Метод ToCharArray() возвращает массив символов указанной строки:

Пример:

```
string s = "Hello, World";
char[] array = s.ToCharArray(); // элементымассива – ' H', ' e', '1'…
```

Метод Split() – возвращает массив строк, разбитый по символу:

Пример:

```
string s = "Arsenal,Milan,RealMadrid,Barcelona";
string[] array = s.Split(‘,’); // элементымассива – "Arsenal","Milan","Real Madrid","Barcelona"
```

В C# также есть класс StringBuilder, который позволяет изменять строки. Пример:

```
StringBuilderhello = newStringBuilder("Привет, менязовутИван",120);
```

120 – количество символов в строке. По умолчанию устанавливается емкость в 16 символов.

Пример 1.

Есть некий текст. Необходимо заменить в этом тексте все слова «C++» на «C#».

Код программы

```
7   namespace ConsoleApp4
8   {
        Ссылок: 0
9       internal class Program
10      {
            Ссылок: 0
11          static void Main(string[] args)
12          {
13              Console.WriteLine("Введите строку:");
14              string s = Console.ReadLine();
15              Console.WriteLine("Новая строка:\n" + s.Replace("C++", "C#"));
16              Console.ReadKey();
17          }
18      }
19  }
20
```

Результат

```
Введите строку:
C++ является объектно-ориентированным языком программирования
Новая строка:
C# является объектно-ориентированным языком программирования
```

Пример 2.

Дан текст – «Сегодня мы с вами рассмотрели, как работать со строками в C#. Были описаны основные операторы и методы, которые используются для работы со строками». Обрежьте этот текст так, чтобы осталась только часть «Были описаны основные операторы и методы».

Код программы

```
7   namespace ConsoleApp5
8   {
        Ссылок: 0
9       internal class Program
10      {
            Ссылок: 0
11          static void Main(string[] args)
12          {
13              string s = "Сегодня мы с вами рассмотрели, как работать со строками в Си-шарп. " +
14                  "Были описаны основные операторы и методы, которые используются для работы " +
15                  "со строками";
16              string s1 = (s.Substring(67)).Remove(40);
17              Console.WriteLine(s1);
18              Console.ReadKey();
19          }
20      }
21  }
```

Результат

```
Были описаны основные операторы и методы
```

Пример 3.

Дана строка, которая содержит имена пользователей, разделенные запятой – "Login1,LOgin2,login3,loGin4". Необходимо разбить эту строку на массив строк (чтобы отдельно были логины), и перевести их все в нижний регистр.

Код программы

```
7   namespace ConsoleApp6
8   {
        Ссылок: 0
9       internal class Program
10      {
            Ссылок: 0
11          static void Main(string[] args)
12          {
13              string s = "Login1,LOgin2,login3,loGin4";
14              string[] mas = s.Split(',');
15              Lower(mas);
16              Console.ReadKey();
17          }
18
            ссылка: 1
19          static void Lower(string[] mas)
20          {
21              for (int i = 0; i < mas.Length; i++)
22              {
23                  mas[i] = mas[i].ToLower();
24                  Console.Write("{0} ", mas[i]);
25              }
26          }
27      }
28  }
```

Результат

```
login1 login2 login3 login4
```

Контрольные вопросы

1. Как объявить строку в C# ?
2. Перечислите основные методы для работы со строками в C# и приведите примеры

4. НЕКОТОРЫЕ НОВЫЕ ВОЗМОЖНОСТИ, ПРЕДОСТАВЛЯЕМЫЕ C#

4.1. СВОЙСТВА И ИНДЕКСАТОРЫ

Свойства позволяют скрыть поле класса от чтения или записи. При этом они предоставляют механизм доступа к закрытым (private) полям класса, которые иначе были бы недоступны. Они позволяют проверить правильность передаваемых значений прежде, чем выполнить какие-либо действия с ними, что обеспечивает безопасность кода и скрывает его реализацию.

Синтаксис для объявления свойств в C#: <МодификаторДоступа> <ТипВозвращаемогоЗначения> <ИмяСвойства> { //тело свойства }

Для чтения или записи значений свойств используются специальные методы «get» и «set» соответственно.

Пример:

```
using System;

public class Program
{
    class SalaryDetails { private string _empName;
              public string EmployeeName
             {  get   { return _empName; }
             set         { _empName = value; } }
             static void Main (string [] args)     {
             SalaryDetails objSal = new SalaryDetails();
             objSal.EmployeeName = "Петр Иванов";
             Console.WriteLine("Имя работника: " + objSal.EmployeeName); } }
}
```

Здесь «value» является зарезервированным словом, применяемым для получения значения свойства, в примере «_empName».

Типы свойства определяются наличием или отсутствием прописанных средств доступа get и set, т.е. на чтение, запись и чтение/запись (get и set).

Индексаторы – компоненты данных, которые позволяют обращаться к данным в пределах объектов способом, который похож на доступ к элементам массивов. Индексаторы позволяют индексировать класс, структуру или интерфейс.

Следующий синтаксис показывает создание индексатора:

<МодификаторДоступа> <ТипВозвращаемогоЗначения>
this [<параметр>]
{ get{ // возвращаемое значение }
set{ // записываемое значение } }

Следующий пример демонстрирует использование индексаторов:

```
class EmployeeDetails
{
    public string[] empName = new string[2];
    public string this[int index]
    {
        get
        {
            return empName[index];
        }
        set
        {
            empName[index] = value;
        }
    }
    static void Main(string[] args)
    {
        EmployeeDetails objEmp = new EmployeeDetails();
        objEmp[0] = "Вася Иванов";
        objEmp[1] = "Катя Петрова";
        Console.WriteLine("Имена сотрудников: ");
        for (int i=0; i<2; i++)
        {
            Console.Write(objEmp[i] + "\t");
        }
    }
}
```

У индексаторов должен быть, по крайней мере, один параметр. Параметр обозначает индексную позицию, используя которую соответствующее значение будет записано или считано. У индексаторов могут также быть множественные параметры, что означает доступ к объекту как к многомерному массиву.

Контрольные вопросы

1. Для чего применяются свойства в C#?
2. Какие типы свойств бывают?
3. Что такое индексаторы в C#?
4. Для чего используется зарезервированное слово «value»?

4.2. ПРОСТРАНСТВО ИМЕН

Работая над большим проектом, можно столкнуться с ситуациями, когда у классов есть идентичные имена. Эта проблема может быть решена при наличии нескольких отдельных модулей в рамках проекта. В итоге каждый класс характеризуется не только именем, но и пространством, в котором он был создан:

```
namespace Russia
{
    class Regions     {  }
    class Enterprises    {  }
}
namespace Belorussia
{
    class Regions     {  }
    class Enterprises     {  }
}
```

В пространстве имен могут быть объявлены следующие типы конструкций: классы, интерфейсы, структуры, перечисления, делегаты и другие пространства имен. За счет вложенности пространств имен достигается их иерархичность, что привносит в сетевую структуру связанных наследованием классов и интерфейсов некоторую модульность.

В .NET Framework входят следующие основные пространства имен, базовым для которых является пространство имен «System»:

– System.Collections – содержит классы и интерфейсы, которые определяют комплексные структуры данных, например, списки, очереди, разрядные массивы, хэш-таблицы и словари;

– System.Data – содержит классы, которые составляют архитектуру ADO.NET для работы с базами данных;

– System.Diagnostics – содержит классы, которые используются, чтобы взаимодействовать с системными процессами;

– System.IO – содержит классы, которые дают возможность читать и записывать данные в файловые потоки;

– System.Net – содержит классы, которые позволяют создавать Web-приложения;

– System.Web – содержит классы и интерфейсы, которые позволяют осуществлять взаимодействие между браузером и сервером.

Для ссылки на классы и методы из пространства имен применяется следующий синтаксис: <ИмяПространства>.<ИмяКласса>.<ИмяМетода>.

Импорт пространств имен в приложение осуществляется с помощью ключевого слова «using». Пример вызова метода из «System.Console»:

```
using System;
class World
{
    static void Main(string[] args)
   {
       Console.WriteLine("Hello World");
   }
}
```

Области имен всегда общедоступны (public). Для создания псевдонима (alias) пространства имен служит следующая конструкция: using <ИмяПсевдонима> = <ИмяПространстваИмен>.

Контрольные вопросы

1. Какова основная цель создания пространств имен в C#?
2. Для чего служит пространство имен «System.Data»?
3. Как осуществить импорт пространства имен в программу?

4.3. ДЕЛЕГАТЫ И СОБЫТИЯ

Делегаты – объекты, ссылающиеся на определенные пользователем методы, и определяющие нужный метод только во время выполнения. Чтобы связать делегат со специфическим методом, у метода должен быть тот же самый тип возвращаемого значения и параметры, что и у делегата.

Методы могут передаться в качестве параметров для делегата. Кроме того, делегат может принимать блок программы в качестве параметра. Такие блоки получили название «анонимные методы», так как у них нет никакого имени метода.

Делегат может вызвать множество методов одновременно. Это известно как мультивещание. Делегат может вызывать статические методы.

Для полной реализации делегата следует выполнить следующие шаги:

1. Объявить делегат.
2. Создать метод, на который будет ссылаться делегат.
3. Связать данный метод с делегатом.
4. Вызвать метод с использованием объекта делегата.

Синтаксис для объявления делегатов: <МодификаторДоступа> delegate <ТипВозвращаемогоЗначения> <ИмяДелегата> ([<СписокПараметров>]).

При этом, в отличие от метода для делегата, не нужна реализация.

Синтаксис для связывания делегата с вызываемым им методом:

<ИмяДелегата><ИмяОбъекта> = new<ИмяДелегата>(<ИмяМетода>).

Любой делегат может быть объявлен перед созданием класса с методом, на который он будет ссылаться, либо в пределах этого класса.

Пример реализации делегата:

```
class DelegatesDemo
{
    public delegate double Temperature(double temp);
    public static double FahrenheitToCelsius(double temp)
    {
        return ((temp-32)/9)*5;
    }
    public static void Main()
    {
        Temperature tempConversion = new Temperature(FahrenheitToCelsius);
        double tempF = 100;
        double tempC = tempConversion(tempF);
        Console.WriteLine("Температура по Фаренгейту = {0:F}", tempF);
        Console.WriteLine("Температура по Цельсию = {0:F}", tempC);
    }
}
```

События в C# позволяют объекту – источнику события уведомлять другие объекты – подписчики о том, что событие произошло. Например, о нажатии пользователем кнопки мыши, нажатии клавиши «Enter» и др.

События в C# могут:

– объявляться в классах и интерфейсах;
– объявляться как абстрактные или запечатанные (sealed);
– объявляться как виртуальные (virtual);
– быть реализованными с помощью делегатов (delegates).

Механизм событий широко применяется в Windows-приложениях, поддерживающих графический интерфейс пользователя (GUI).

Чтобы реализовать события в C#, следует выполнить следующие шаги:

1. Определить делегат, который будет связан с событием.

2. Создать событие, используя делегат.

3. Подписаться на событие, чтобы слушать и обрабатывать событие при его возникновении.

4. Сгенерировать событие.

Синтаксис объявления делегата и события: <МодификаторДоступа> delegate <ТипВозвращаемогоЗначения> <Идентификатор> (<Параметры>);

<МодификаторДоступа> event <ИмяДелегата> <ИмяСобытия>.

Синтаксис создания в классе получателя события: <МодификаторДоступа> <ТипВозвращаемогоЗначения> <ИмяМетода> (<Параметры>).

Синтаксис связывания метода и события: <имяОбъекта>.<ИмяСобытия> += new <ИмяДелегата> (ИмяМетода), где <имяОбъекта> – объект класса, в котором определен обработчик событий.Для генерации события его следует вызвать как метод без параметров.Пример:

```
public delegate void Display(); //объявление делегата Display
class Event
{
    event PrintDetails Prin //объявление события Print
    void Show()
    {
        Console.WriteLine("Это событие");
    //описание метода, на который будет подписано событие
    }
    static void Main(string[] args)
    {
    Event objEvents = new Event();
    //подписывание метода Show на событие Print
      objEvents.Print += new Display(objEvents.Show);
     objEvents.Print()//генерация события Print
    }
}
```

Контрольные вопросы

1. Перечислите шаги полной реализации делегата в C#?
2. Для чего применяются события в C#?
3. Как связать делегат и событие в C#?
4. Что нужно, чтобы сгенерировать событие?

5. СОЗДАНИЕ ГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ WINDOWS FORMS

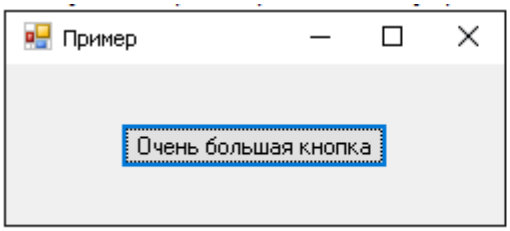
5.1. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ КНОПОК

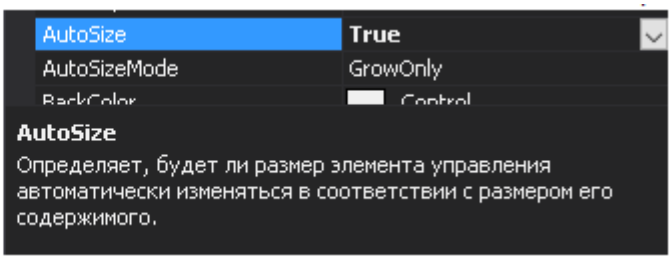
Button – элемент-кнопка. Для помещения ее на форму необходимо выбрать на панели элементов пункт Button.

Рассмотрим основные свойства кнопок:

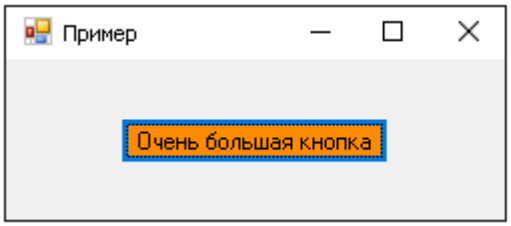
Anchor – возвращает или задает границы контейнера, с которым связан элемент управления, и определяет способ изменения размеров элемента управления при изменении размеров его родительского элемента.

AutoSize – определяет, будет ли размер элемента управления автоматически изменятся в соответствии с размером.

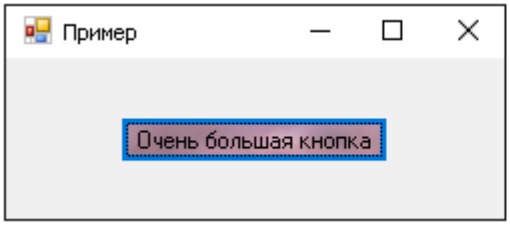


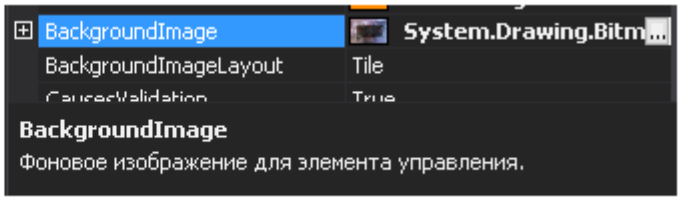


BackColor – фоновый цвет компонента.



BackgroundImage – фоновое изображение для элемента управления.

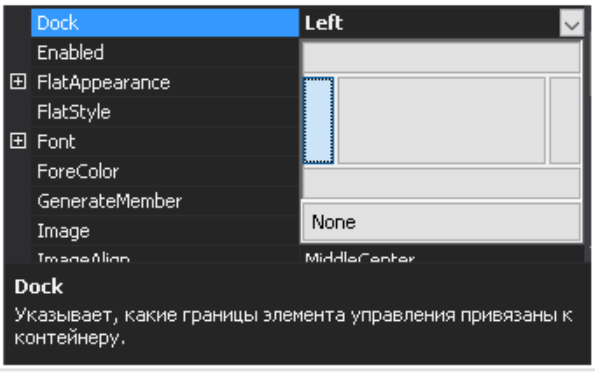




Context Menu Strip – включает меню быстрого доступа, отображаемое при щелчке правой кнопкой мыши на данном элементе управления.

Cursor – курсор, отображаемый при наведении указателя мыши на данный элемент управления.

Dock – указывает, какие элементы управления привязаны к контейнеру.



Enabled – указывает включен ли элемент управления.

Font – шрифт, используемый для отображения текста на элементе управления.

ForeColor – основной цвет для отображения текста в данном элементе управления.

Image – изображение, которое будет отображаться на данном элементе управления.

Location – координаты левого верхнего угла элемента управления относительно левого верхнего угла контейнера.

Locked – определяет, можно ли перемещать элемент управления или изменять его размеры.

Size – размер элемента управления в точках.

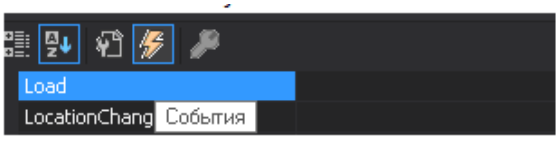
Tablndex – задает индекс порядка переключения Tab для данного элемента управления.

Text – текст, связанный с элементом управления.

TextAlign – выравнивание текста, который будет отображаться на данном элемента управления.

Visible – определяет, отображается или скрыт данный элемент управления.

Чтобы задать кнопке событие, необходимо в окне свойств кликнуть на значке «События».



Рассмотрим некоторые из них:

Click – происходит при щелчке элемента управления.

KeyDown – происходит при нажатии клавиши, если элемент управления имеет фокус.

KeyUp – происходит, когда отпускается клавиша, если элемент управления имеет фокус.

MouseClick – вызывается при щелчке мышью элемента управления.

MouseDown – происходит при нажатии кнопки мыши, если указатель мыши находится на элементе управления.

MouseHower – происходит, если указатель мыши остается неподвижным в течение нескольких секунд в пределах данного элемента управления.

MouseEnter – происходит, когда указатель мыши оказывается на элементе управления.

MouseLeave – происходит, когда указатель мыши покидает элемент управления.

MouseMove – происходит при перемещении указателя мыши по элементу управления.

MouseUp – происходит при отпускании кнопки мыши, когда указатель мыши находится над элементом управления.

Пример 1

Создадим в оконном приложении следующие события для кнопки:

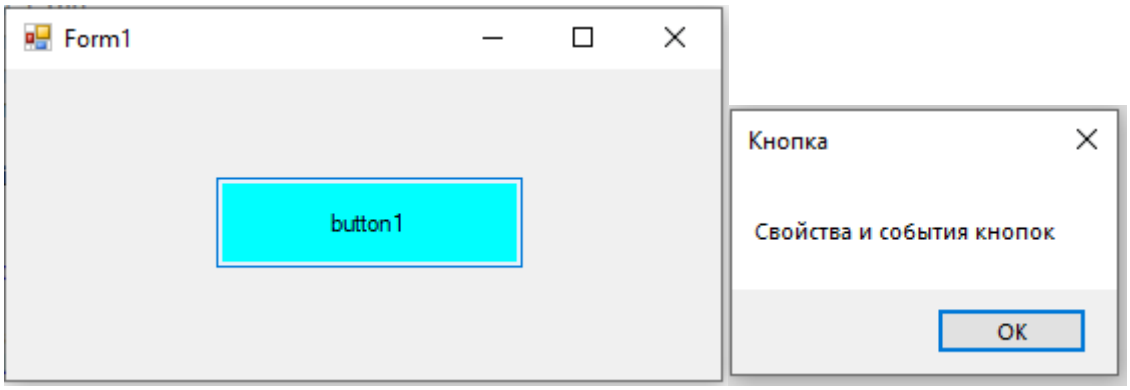
1) при клике на кнопку выводится окно сообщения MessageBox с текстом «Свойства и события кнопок» и заголовком;

2) при наведении мыши на кнопку меняется цвет кнопки.

Код программы

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        ссылка: 1
        public Form1()
        {
            InitializeComponent();
        }

        ссылка: 1
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Свойства и события кнопок", "Кнопка");
        }

        ссылка: 1
        private void button1_MouseMove(object sender, MouseEventArgs e)
        {
            button1.BackColor = Color.Aqua;
        }
    }
}
```

Результат:



Пример 2.

Создадим оконное приложение, состоящее из двух форм, осуществим переход от одной формы к другой. На каждой из форм организуем завершение работы программы.

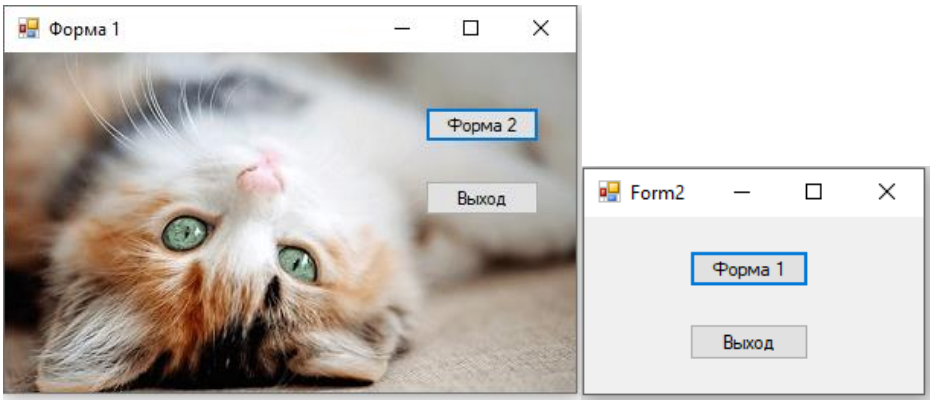
Код программы Формы 1:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp3
{
    Ссылок: 6
    public partial class Form1 : Form
    {
        Ссылок: 2
        public Form1()
        {
            InitializeComponent();
        }

        ссылка: 1
        private void button1_Click(object sender, EventArgs e)
        {
            Form2 f2 = new Form2();
            f2.Show();
            this.Hide();
        }

        ссылка: 1
        private void button2_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

Код программы Формы 2:

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace WindowsFormsApp3
12  {
        Ссылок: 4
13      public partial class Form2 : Form
14      {
            ссылка: 1
15          public Form2()
16          {
17              InitializeComponent();
18          }
19
            ссылка: 1
20          private void button1_Click(object sender, EventArgs e)
21          {
22              this.Hide();
23              Form1 f1 = new Form1();
24              f1.Show();
25          }
26
            ссылка: 1
27          private void button2_Click(object sender, EventArgs e)
28          {
29              Application.Exit();
30          }
31      }
32  }
```

Результат:

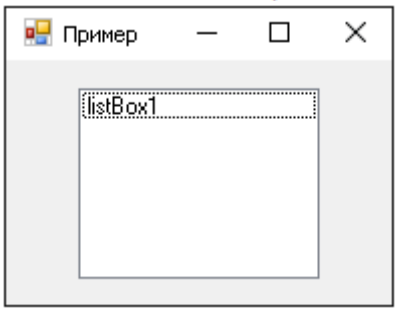


Контрольные вопросы

1. Каким образом можно использовать компонент Button?
2. Перечислите основные свойства кнопок
3. Как создать обработчик события?
4. Перечислите основные события кнопок

5.2. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ СПИСКОВ, ФЛАЖКОВ И ПЕРЕКЛЮЧАТЕЛЕЙ

ListBox – элемент-список, из которого пользователь может выбирать элементы (нумерация ведется с 0).



Horizontal Scrollbar – указывает, должна ли в ListBox отображаться горизонтальная полоса прокрутки для элементов, выходящих за правый край ListBox

Scroll AlwaysVisible – указывает, должен ли список всегда содержать полосу прокрутки вне зависимости от числа элементов в нем.

Selection Mode – указывает сколько элементов можно выбрать в поле списка (один, несколько или ни одного).

Items – элементы в данном списке.

Для добавления элемента x в список: listBoxI .Items.Add(x);

Для добавления массива элемента a в список: listBoxI .Items.AddRange(a);

Для вставки элемента m на позицию k: listBoxI.Items.Insert(k, m);

Для очищения списка: listBoxI.Items.Clear();

Для удаления элемента по тексту: listBoxI. Items. Remove(x);

Для удаления элемента по его индексу a: listBoxI. Items. RemoveAt(a);

Для определения количества элементов в списке: listBoxI. Items. Count();

Для доступа к элементам списка используется его индекс: listBoxI. ltems[0].

listBoxl.Selectedlndex – возвращает или устанавливает номер выделенного элемента списка, если выделенные элементы отсутствуют, то значение равно –1.

Для удаления элемента по его индексу a:listBoxl.Selectedltem – возвращает или устанавливает текст выделенного элемента.

listBox1.SetSelected(n, m) – выделяет элемент под номером n. если m=true, то элемент выделяется, если m=false элемент скрывается.

Элемент CheckBox или флажок предназначен для установки одного из двух значений: отмечен или не отмечен. Чтобы отметить флажок, нужно задать свойству Cheked значение True.

Свойство AutoCheck (по умолчанию true) если оно имеет значение false, то мы не можем изменять состояние флажка.

Свойство CheckBox.Checked указывает, какое значение принимает флажок (true или false).

Элемент RadioButton или переключатель похож на флажок CheckBox, но переключатели располагаются группами, и включение одного переключателя означает отключение всех остальных. Чтобы установить у переключателя включенное состояние, надо присвоить его свойству Checked значение true.

Для создания группы переключателей нужно поместить несколько переключателей в какой-нибудь контейнер (GroupBox или Panel). Переключатели, находящиеся в разных контейнерах, будут относиться к разным группам. Для работы с группой переключателей удобно использовать событие CheckedChanged:

RadioButton<имя переменной> = (RadioButton)sender; //обращаемся к элементу RadioButton в общем виде

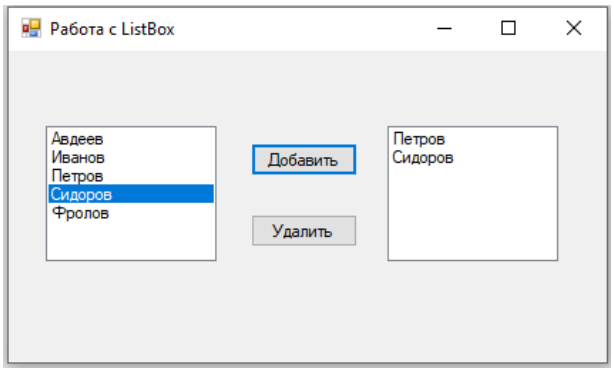
Пример 1.

Поместим на форму 2 списка ListBox, кнопки «Добавить» и «Удалить». Создадим добавление во 2-ой список выбранных элементов из первого списка (при повторном добавлении одного и того же элемента добавление не происходит) и удаление элементов из второго списка.

Код программы

```
namespace WindowsFormsApp4
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        ссылка: 1
        public Form1()
        {
            InitializeComponent();
        }

        ссылка: 1
        private void Form1_Load(object sender, EventArgs e)
        {
            listBox2.Items.Clear();
        }

        ссылка: 1
        private void button1_Click(object sender, EventArgs e)
        {
            if (listBox1.SelectedIndex == -1)
                MessageBox.Show("Выберите один пункт из первого списка", "Ошибка!");
            else
            {
                listBox2.Items.Add(listBox1.SelectedItem);
                for (int i = 0; i < listBox2.Items.Count - 1; i++)
                {
                    if (listBox2.Items[i] == listBox2.Items[listBox2.Items.Count - 1])
                    { listBox2.Items.RemoveAt(listBox2.Items.Count - 1); }
                }
            }
        }

        ссылка: 1
        private void button2_Click(object sender, EventArgs e)
        {
            listBox2.Items.Remove(listBox2.SelectedItem);
        }
    }
}
```

Результат:



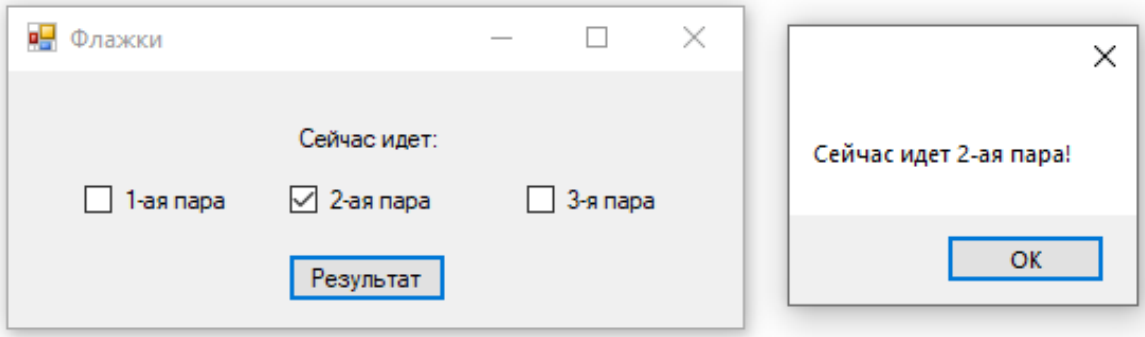
Пример 2.

Поместим на форму три флажка и кнопку «Результат». После каждого нажатия кнопки «Результат» выводится сообщение о том, какая пара сейчас идет и обнуляются значения всех флажков. Если выбраны сразу несколько флажков или не выбран ни один из них, то выдается сообщение об ошибке.

Код программы

```
11  namespace WindowsFormsApp5
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
            ссылка: 1
15          public Form1()
16          {
17              InitializeComponent();
18          }
            ссылка: 1
19          private void button1_Click(object sender, EventArgs e)
20          {
21              if (((checkBox1.Checked == true) && (checkBox2.Checked == true)) ||
22                  ((checkBox1.Checked == true) && (checkBox3.Checked == true)) ||
23                  ((checkBox3.Checked == true) && (checkBox2.Checked == true)))
24              {
25                  MessageBox.Show("Выберите только один флажок!", "Ошибка!!!");
26                  checkBox2.Checked = false;
27                  checkBox3.Checked = false;
28                  checkBox1.Checked = false;
29              }
30              else
31              {
32                  if (checkBox1.Checked == true)
33                  {
34                      MessageBox.Show("Сейчас идет " + checkBox1.Text + "!");
35                      checkBox2.Checked = false;
36                      checkBox3.Checked = false;
37                      checkBox1.Checked = false;
38                  }
39                  if (checkBox2.Checked == true)
40                  {
41                      MessageBox.Show("Сейчас идет " + checkBox2.Text + "!");
42                      checkBox1.Checked = false;
43                      checkBox2.Checked = false;
44                      checkBox3.Checked = false;
45                  }
46                  if (checkBox3.Checked == true)
47                  {
48                      MessageBox.Show("Сейчас идет " + checkBox3.Text + "!");
49                      checkBox1.Checked = false;
50                      checkBox2.Checked = false;
51                      checkBox3.Checked = false;
52                  }
53              }
54          }
55      } }
```

Результат



Пример 3.

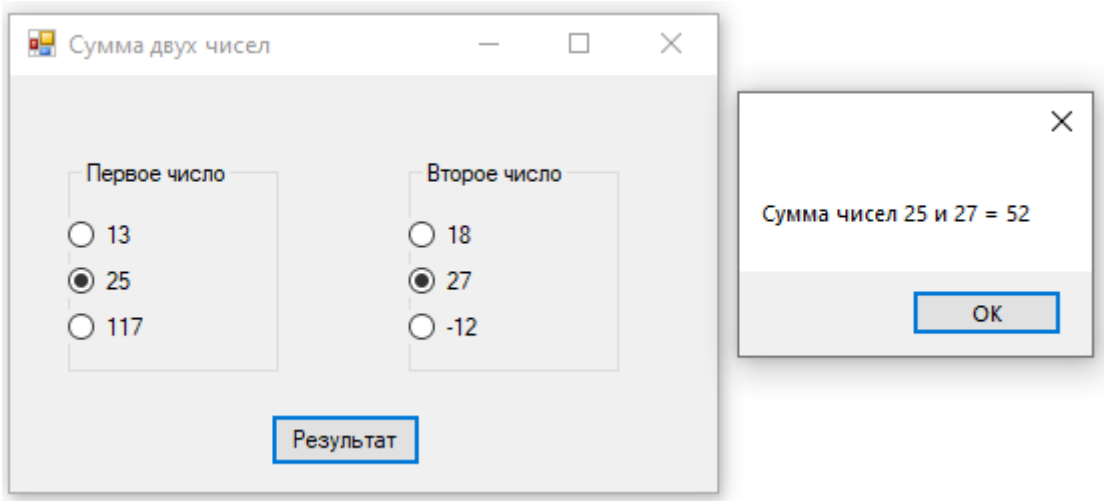
Поместим на форму две группы переключателей и кнопку «Результат». После каждого нажатия кнопки «Результат» выводится сумма чисел, выбранных с помощью переключателей.

Код программы

```
11  namespace WindowsFormsApp6
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
15          int a, b;
            ссылка: 1
16          public Form1()
17          {
18              InitializeComponent();
19          }
            ссылка: 1
20          private void radioButton1_CheckedChanged(object sender, EventArgs e)
21          {
22              RadioButton rb1 = (RadioButton)sender;
23              if (rb1.Checked == true)
24                  a = Convert.ToInt32(rb1.Text);
25          }
            ссылка: 1
26          private void radioButton10_CheckedChanged(object sender, EventArgs e)
27          {
28              RadioButton rb2 = (RadioButton)sender;
29              if (rb2.Checked == true)
30                  b = Convert.ToInt32(rb2.Text);
31          }
            ссылка: 1
32          private void button1_Click(object sender, EventArgs e)
33          {
34              MessageBox.Show("Сумма чисел " + a + " и " + b + " = " + (a + b));
35          }
            ссылка: 1
36          private void radioButton2_CheckedChanged(object sender, EventArgs e)
37          {
38              RadioButton rb1 = (RadioButton)sender;
39              if (rb1.Checked == true)
40                  a = Convert.ToInt32(rb1.Text);
41          }
```

```
42          private void radioButton3_CheckedChanged(object sender, EventArgs e)
43          {
44              RadioButton rb1 = (RadioButton)sender;
45              if (rb1.Checked == true)
46                  a = Convert.ToInt32(rb1.Text);
47          }
            ссылка: 1
48          private void radioButton9_CheckedChanged(object sender, EventArgs e)
49          {
50              RadioButton rb2 = (RadioButton)sender;
51              if (rb2.Checked == true)
52                  b = Convert.ToInt32(rb2.Text);
53          }
            ссылка: 1
54          private void radioButton8_CheckedChanged(object sender, EventArgs e)
55          {
56              RadioButton rb2 = (RadioButton)sender;
57              if (rb2.Checked == true)
58                  b = Convert.ToInt32(rb2.Text);
59          }
            ссылка: 1
60          private void Form1_Load(object sender, EventArgs e)
61          {
62              radioButton1.Checked = true;
63              radioButton10.Checked = true;
64          }
65      }
66  }
```

Результат

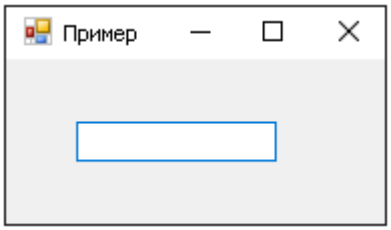


Контрольные вопросы

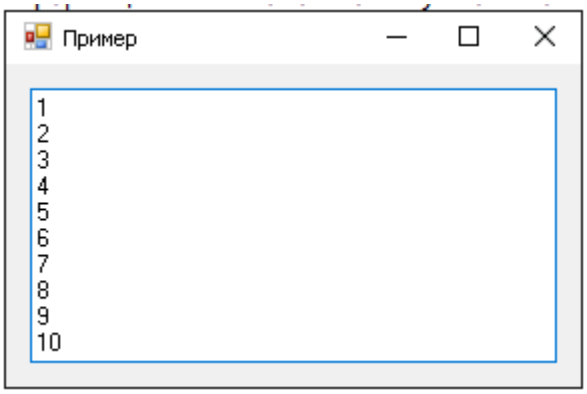
1. Что такое ListBox ?
2. Перечислите основные свойства списка
3. Какие основные методы для работы со списоком вы знаете?
4. Что позволяет делать элемент управления CheckBox?
5. Для чего предназначены радиокнопки?

5.3. СВОЙСТВА И СОБЫТИЯ ТЕКСТОВОГО ПОЛЯ

TextBox – текстовое поле для ввода и редактирования текста. Текст элемента TextBox можно установить или получить с помощью свойства Text.



Для отображения больших объемов информации в текстовом поле нужно использовать его свойства Multiline и ScrollBars.



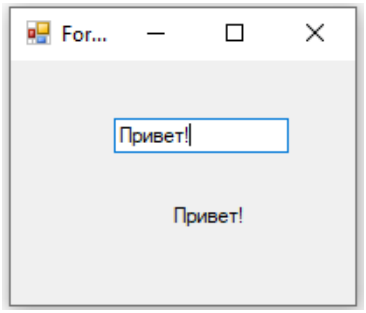
Чтобы текст в элементе TextBox переносился по словам, надо установить свойство Wordwrap, равным true. Данное свойство будет работать только для многострочных текстовых полей.

Событие TextChanged срабатывает при изменении текста в элементе. Поместим на форму текстовое поле и метку. Пусть при изменении текста в текстовом поле меняется текст на метке.

Код программы

```
11  namespace WindowsFormsApp7
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
            ссылка: 1
15          public Form1()
16          {
17              InitializeComponent();
18              textBox1.TextChanged += textBox1_TextChanged;
19          }
20
            Ссылок: 2
21          private void textBox1_TextChanged(object sender, EventArgs e)
22          {
23              label1.Text = textBox1.Text;
24          }
25      }
26  }
```

Результат



Пример 1.

Приложение «Ввод пароля» (пароль). Создадим на форме три кнопки: «Принять пароль» (происходит проверка пароля, если неверный выводится сообщение об ошибке, если верный, происходит переход на форму с информацией об авторе), «Показать пароль» (в текстовом поле отображается введенный пароль) и «Выход». На второй форме содержится кнопка возврата на форму.

Код программы

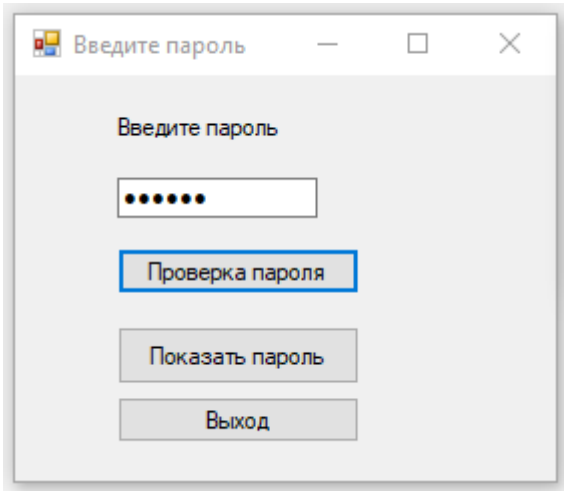
Форма 1

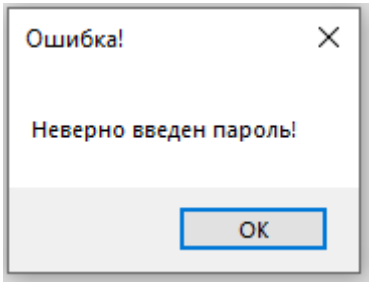
```
11  namespace WindowsFormsApp8
12  {
        Ссылок: 5
13      public partial class Form1 : Form
14      {
            Ссылок: 2
15          public Form1()
16          {
17              InitializeComponent();
18          }
19
            ссылка: 1
20          private void button1_Click(object sender, EventArgs e)
21          {
22              Form2 f2 = new Form2();
23              string s = textBox1.Text;
24              if (s == "qwerty")
25              {
26                  this.Hide();
27                  f2.Show();
28              }
29              else MessageBox.Show("Неверно введен пароль!", "Ошибка!");
30          }
31
            ссылка: 1
32          private void button2_Click(object sender, EventArgs e)
33          {
34              if (textBox1.UseSystemPasswordChar == true)
35                  textBox1.UseSystemPasswordChar = false;
36              else textBox1.UseSystemPasswordChar = true;
37          }
38
            ссылка: 1
39          private void button3_Click(object sender, EventArgs e)
40          {
41              Application.Exit();
42          }
43      }
44  }
```

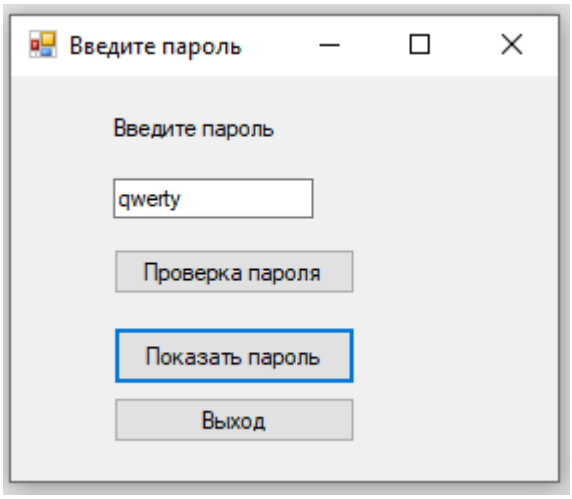
Форма 2

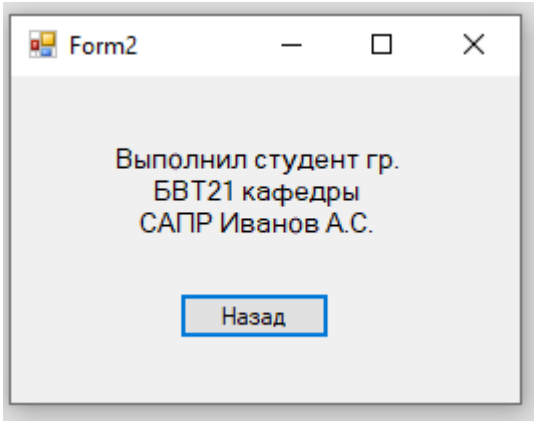
```
11  namespace WindowsFormsApp8
12  {
        Ссылок: 4
13      public partial class Form2 : Form
14      {
            ссылка: 1
15          public Form2()
16          {
17              InitializeComponent();
18          }
19
            ссылка: 1
20          private void button1_Click(object sender, EventArgs e)
21          {
22              Form1 f1 = new Form1();
23              f1.Show();
24              this.Hide();
25          }
26      }
27  }
```

Результат







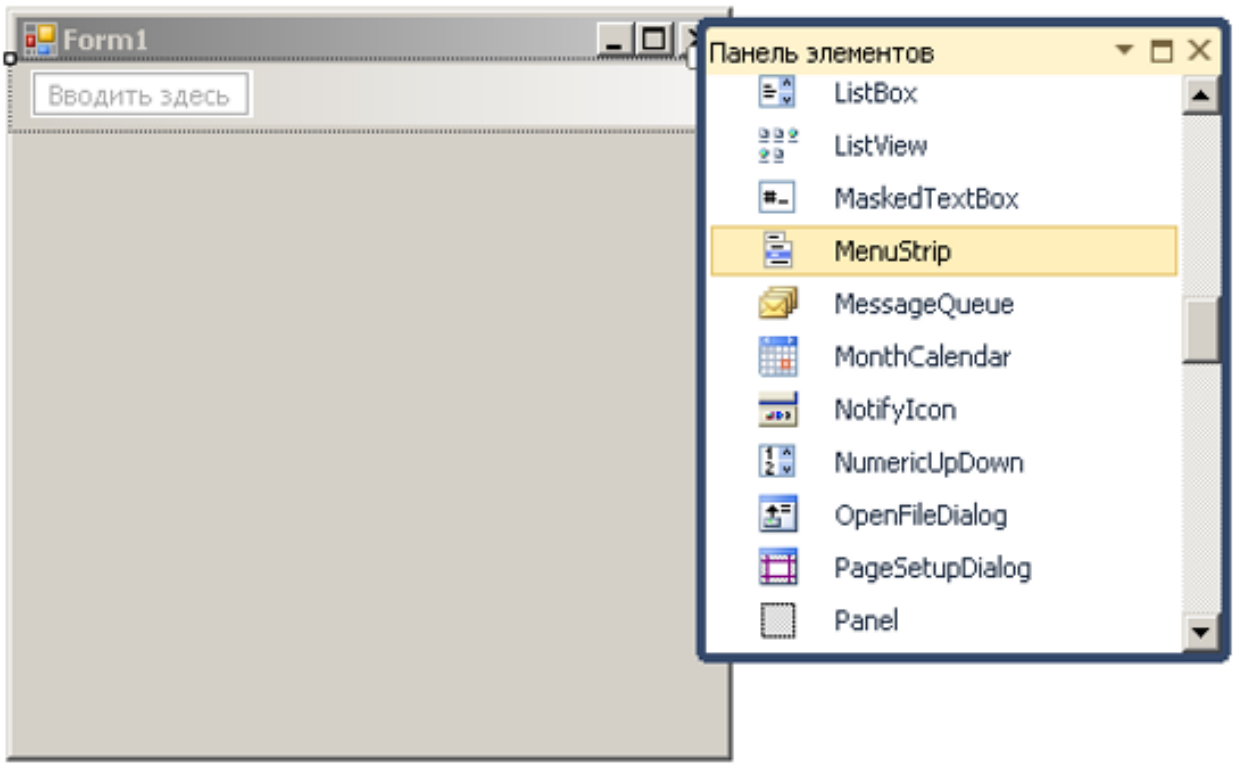


Контрольные вопросы

1. Что такое TextBox?
2. Каким образом можно использовать компонент TextBox?

5.4. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ ВЫПАДАЮЩЕГО МЕНЮ

Для создания выпадающего меню в WindowsForms применяется элемент MenuStrip.



Наиболее важные свойства компонента MenuStrip:

Layoutstyle – ориентация панели меню на форме;

HorizontalStackWithOverflow – расположение по горизонтали с переполнением;

StackWithOverflow – автоматическое расположение с переполнением;

VerticalStackWithOverflow – расположение вертикально с переполнением;

Flow – размещение автоматически без переполнения (выходящие за границы элементы переносятся);

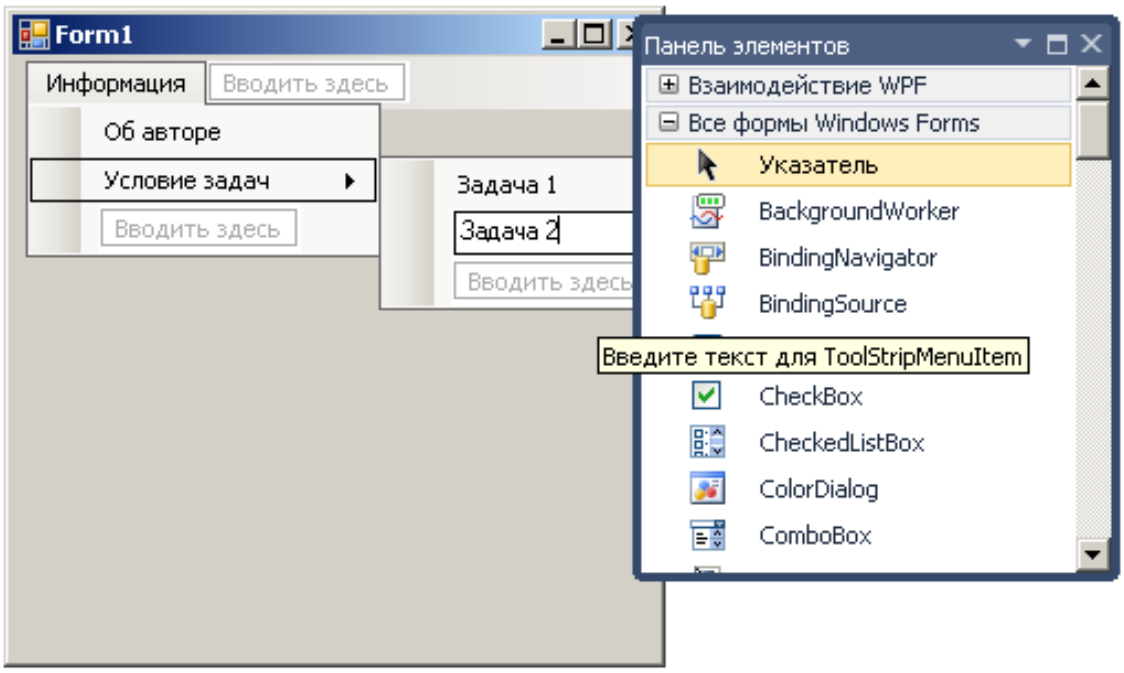
Table – элементы в виде таблицы;

ShowltemToolTips – указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню;

Stretch – позволяет растянуть панель по всей длине контейнера;

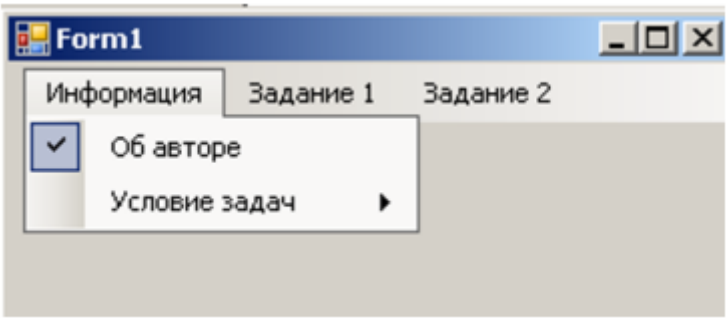
TextDirection – задает направление текста в пунктах меню.

MenuStrip выступает своего рода контейнером для отдельных пунктов меню, которые представлены объектом ToolStripMenultem. Добавить новые элементы в меню можно в режиме дизайнера.

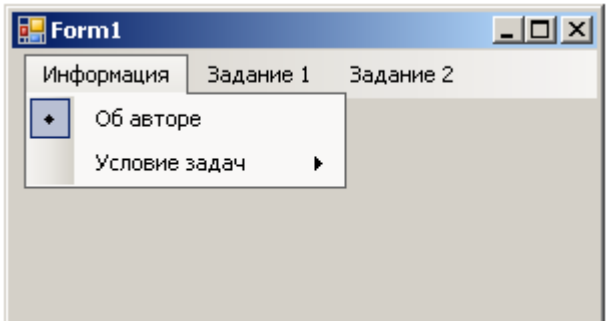


Для добавления доступно три вида элементов: MenuItem (объект ToolStripMenuItem), ComboBox и TextBox. В меню мы можем использовать выпадающие списки и текстовые поля, однако, как правило, эти элементы применяются в основном на панели инструментов. Меню же обычно содержит набор объектов ToolStripMenuItem. ToolStripMenuItem в конструкторе принимает текстовую метку, которая будет использоваться в качестве текста меню. Один элемент ToolStripMenuItem может содержать набор других объектов ToolStripMenuItem. Таким образом, образуется иерархическое меню или структура в виде дерева.

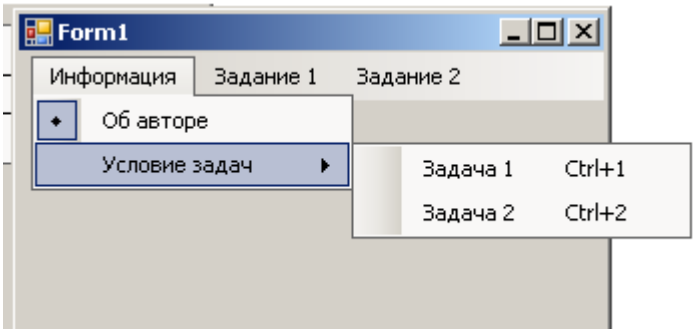
Свойство CheckOnClick при значении True позволяет по клику отметить пункт меню. С помощью свойства Checked можно установить, будет ли пункт меню отмечен при запуске программы.



Свойство CheckState возвращает состояние пункта меню – отмечен он или нет (значения Checked отмечен),Unchecked (не отмечен) и Indeterminate (в неопределенном состоянии).



Если нам надо быстро обратиться к какому-то пункту меню, то мы можем использовать клавиши быстрого доступа. Для задания клавиш быстрого доступа используется свойство ShortcutKeys.

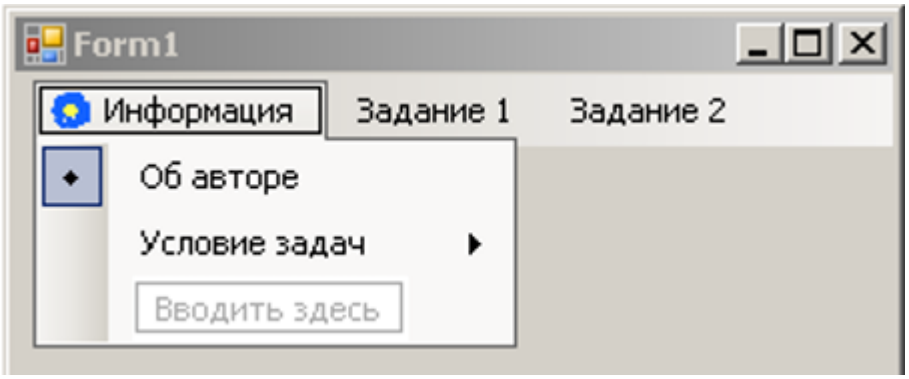


Для изменения внешнего вида меню с помощью изображений следует использовать свойства:

DisplayStyle – определяет, будет ли отображаться на элементе текст, или изображение, или и то, и другое;

Image – указывает на само изображение;

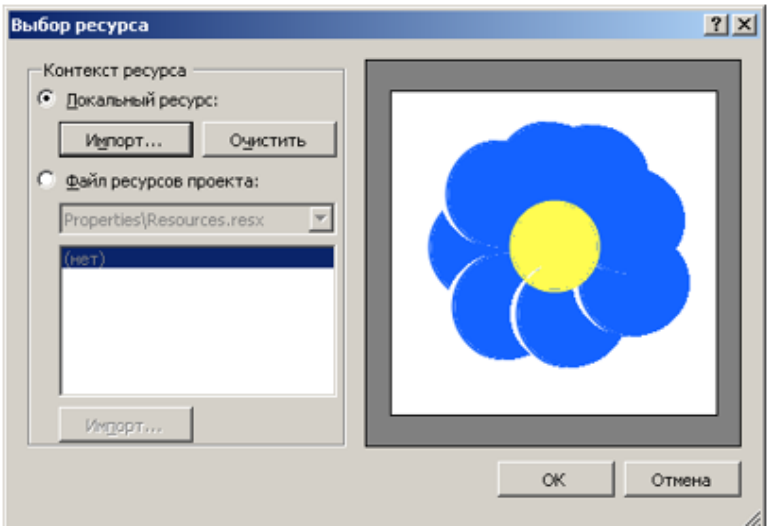
ImageAlign – устанавливает выравнивание изображения относительно элемента.



ImageScaling – указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента;

ImageTransparentColor – указывает, будет ли цвет изображения прозрачным.

Если изображение для пункта меню устанавливается в режиме дизайнера, то надо выбрать в окне свойство пункт Image, после чего откроется окно для импорта ресурса изображения в проект.



Пример 1.

Разработать приложение, содержащее пункты меню со сведениями об авторе и условия задач с решением. Требуется решить две задачи: 1) создать калькулятор вычисления наибольшего элемента массива (массив вводится с клавиатуры в одно текстовое поле); 2) написать программу замены слова или фразы введенного текста на другую.

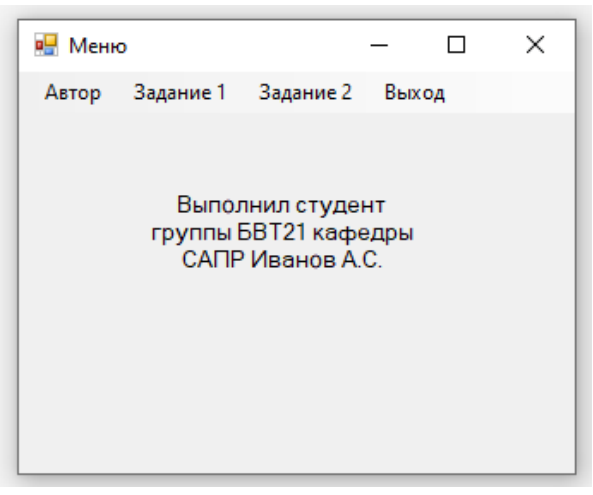
Код программы

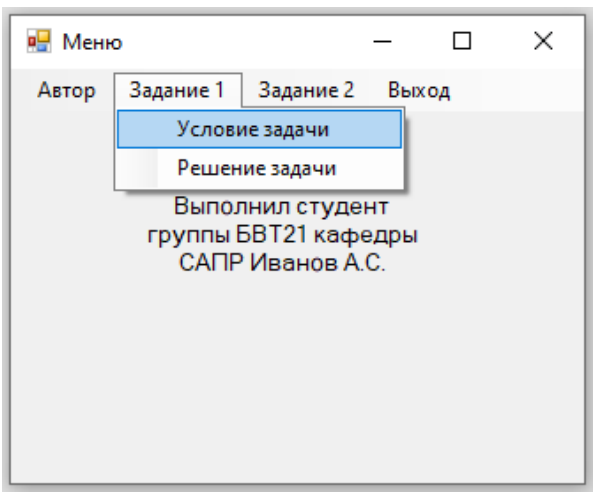
```
11  namespace WindowsFormsApp9
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
15          string s1;
            ссылка: 1
16          public Form1()
17          {
18              InitializeComponent();
19          }
20
            ссылка: 1
21          private void выходToolStripMenuItem_Click(object sender, EventArgs e)
22          {
23              Application.Exit();
24          }
25
            ссылка: 1
26          private void авторToolStripMenuItem_Click(object sender, EventArgs e)
27          {
28              label1.Visible = true;
29              label1.Text = "Выполнил студент группы БВТ21 кафедры САПР Иванов А.С.";
30              label2.Visible = false;
31              label3.Visible = false;
32              textBox1.Visible = false;
33              button1.Visible = false;
34              button2.Visible = false;
35              textBox2.Visible = false;
36              textBox3.Visible = false;
37              textBox4.Visible = false;
38              label4.Visible = false;
39          }
40
```

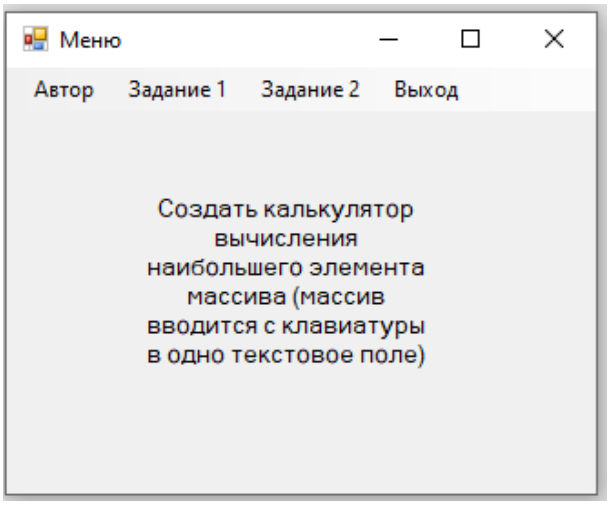
```
41      private void условиеЗадачиToolStripMenuItem_Click(object sender, EventArgs e)
42      {
43          label1.Visible = true;
44          label1.Text = "Создать калькулятор вычисления наибольшего элемента массива " +
45              "(массив вводится с клавиатуры в одно текстовое поле)";
46          label2.Visible = false;
47          label3.Visible = false;
48          textBox1.Visible = false;
49          button1.Visible = false;
50          button2.Visible = false;
51          textBox2.Visible = false;
52          textBox3.Visible = false;
53          textBox4.Visible = false;
54          label4.Visible = false;
55      }
56
            ссылка: 1
57      private void решениеЗадачиToolStripMenuItem_Click(object sender, EventArgs e)
58      {
59          label1.Visible = false;
60          label2.Visible = true;
61          label3.Visible = true;
62          textBox1.Visible = true;
63          button1.Visible = true;
64          button2.Visible = false;
65          textBox2.Visible = false;
66          textBox3.Visible = false;
67          textBox4.Visible = false;
68          label4.Visible = false;
69      }
70
71      private void button1_Click(object sender, EventArgs e)
72      {
73          int max = -100;
74          string s = textBox1.Text;
75          string[] a = s.Split(' ');
76          int[] b = new int[a.Length];
77          for (int i = 0; i < a.Length; i++)
78          {
79              b[i] = Convert.ToInt32(a[i]);
80
81              if (b[i] > max) max = b[i];
82          }
83          label3.Text = "Максимальный элемент массива =" + max;
84      }
85
            ссылка: 1
86      private void условиеЗадачиToolStripMenuItem1_Click(object sender, EventArgs e)
87      {
88          label1.Visible = true;
89          label1.Text = "Напишите программу замены слова или фразы введенного " +
90              "текста на другую ";
91          label2.Visible = false;
92          label3.Visible = false;
93          textBox1.Visible = false;
94          button1.Visible = false;
95          button2.Visible = false;
96          textBox2.Visible = false;
97          textBox3.Visible = false;
98          textBox4.Visible = false;
99          label4.Visible = false;
100     }
101
```

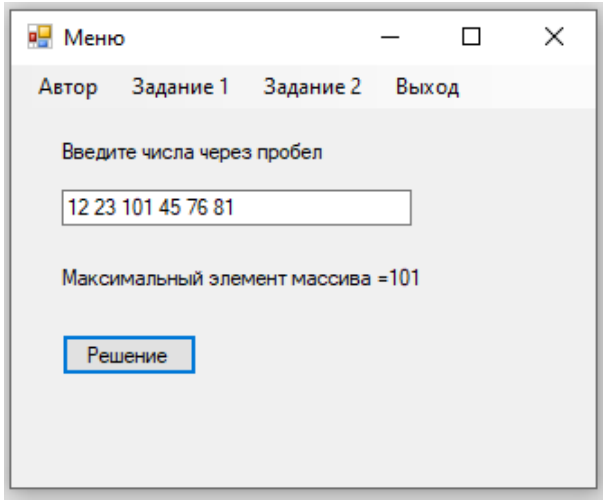
```
102         private void решениеЗадачиToolStripMenuItem1_Click(object sender, EventArgs e)
103         {
104             label1.Visible = false;
105             label2.Visible = true;
106             label2.Text = "Введите слова через пробел:";
107             label3.Visible = false;
108             label4.Visible = true;
109             button2.Visible = true;
110             textBox2.Visible = true;
111             textBox1.Visible = false;
112             textBox3.Visible = true;
113             textBox4.Visible = true;
114             button1.Visible = false;
115         }
116
                ссылка: 1
117         private void button2_Click(object sender, EventArgs e)
118         {
119             s1 = textBox2.Text;
120             textBox2.Text = s1.Replace(textBox3.Text, textBox4.Text);
121         }
122
                ссылка: 1
123         private void Form1_Load(object sender, EventArgs e)
124         {
125             label1.Visible = true;
126             label1.Text = "Задания";
127             label2.Visible = false;
128             label3.Visible = false;
129             textBox1.Visible = false;
130             button1.Visible = false;
131             button2.Visible = false;
132             textBox2.Visible = false;
133             textBox3.Visible = false;
134             textBox4.Visible = false;
135             label4.Visible = false;
136         }
137     }
138 }
```

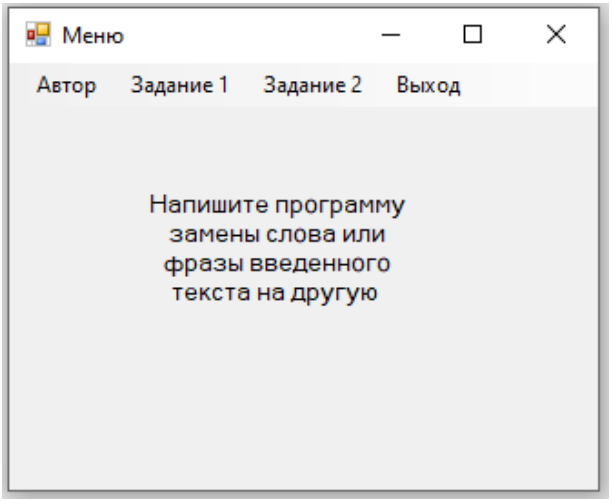
Результат

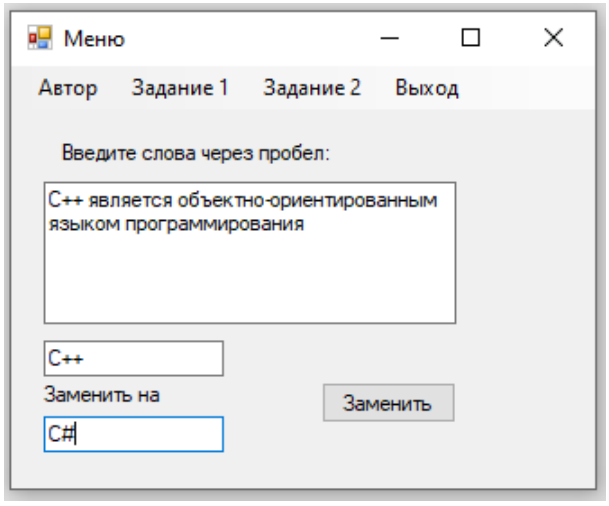


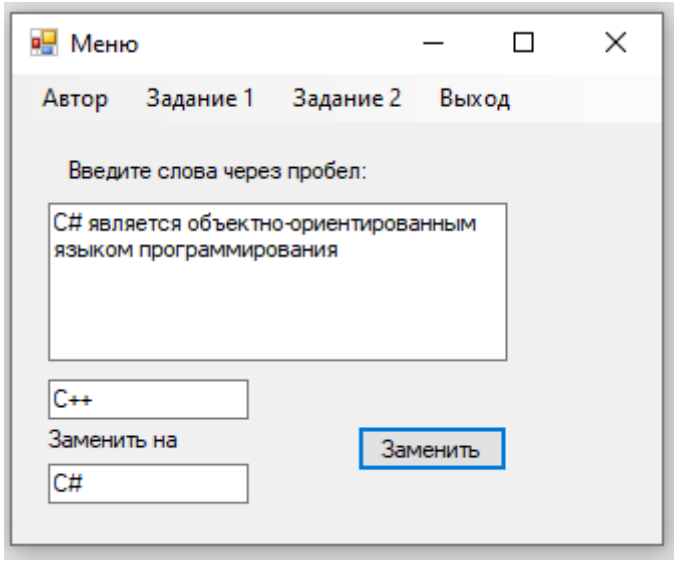












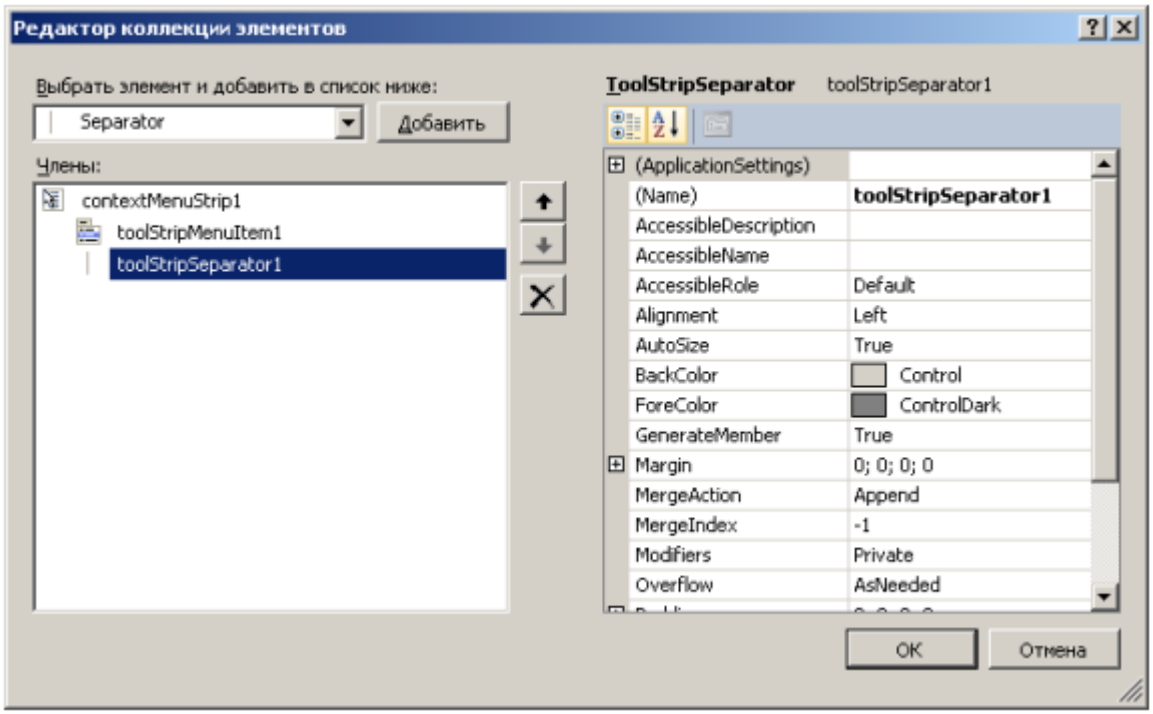
Контрольные вопросы

1. Как создать меню на C#?
2. Перечислите основные свойства компонента MenuStrip.

5.5. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ КОНТЕКСТНОГО МЕНЮ

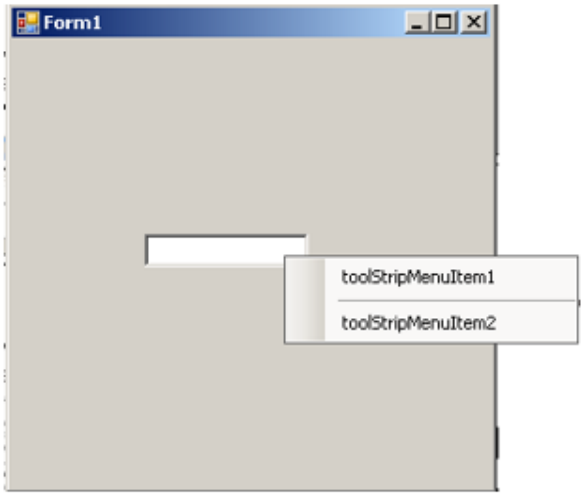
ContextMenuStrip представляет контекстное меню. Данный компонент во многом аналогичен MenuStrip, но контекстное меню обязательно применяется к какому-нибудь другому элементу, например, текстовому полю.

Новые элементы в контекстное меню можно добавить в режиме дизайнера.



При этом мы можем добавить те же элементы, что и в MenuStrip. Но, как правило, использует ToolStripMenuItem, либо элемент ToolStripSeparator, представляющий горизонтальную полоску – разделитель между другими пунктами меню. Либо на панели свойств можно обратиться к свойству Items компонента ContextMenuStrip и в открывшемся окне добавить и настроить все элементы меню.

У многих компонентов есть свойство ContextMenuStrip, которое позволяет ассоциировать контекстное меню с данным элементом, например, textBox1.ContextMenuStrip = contextMenuStrip1. И по нажатию на текстовое поле правой кнопкой мыши мы сможем вызвать ассоциированное контекстное меню.



Пример 1.

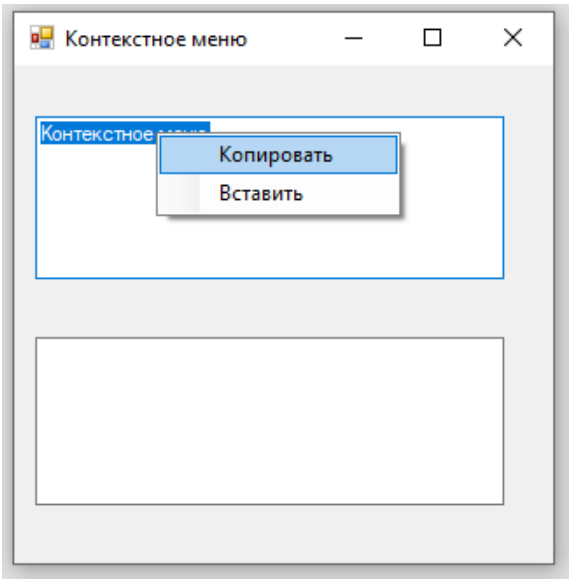
Создадим приложение, позволяющее копировать и вставлять фрагменты текста из одного текстового поля в другое. С помощью другого контекстного меню осуществим очистку полей и выход из программы.

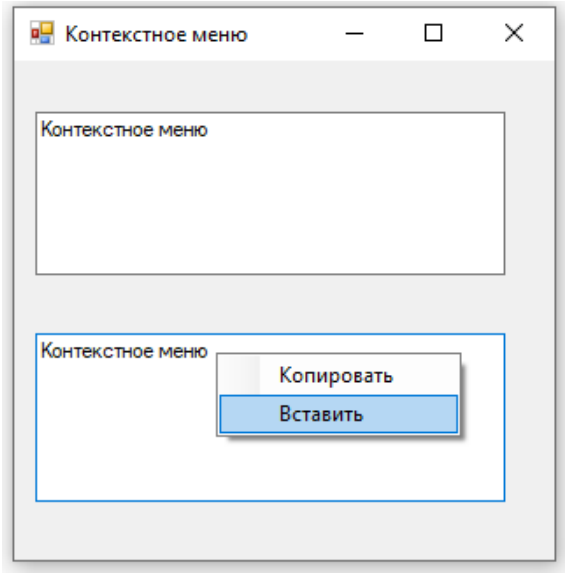
Код программы

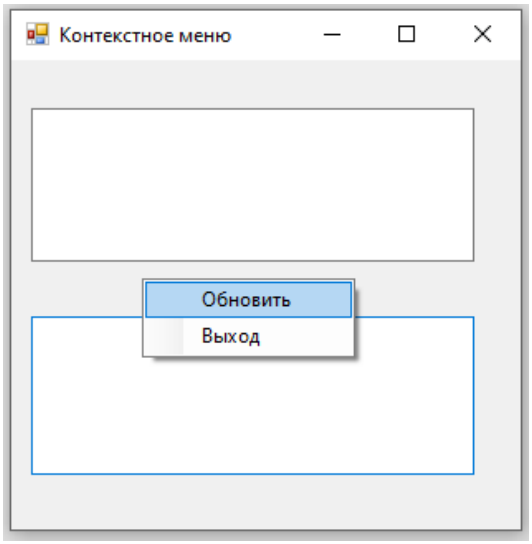
```
11  namespace WindowsFormsApp10
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
            ссылка: 1
15          public Form1()
16          {
17              InitializeComponent();
18          }
19          string bufer;
            ссылка: 1
20          private void Form1_Load(object sender, EventArgs e)
21          {
22              this.ContextMenuStrip = contextMenuStrip2;
23              textBox1.ContextMenuStrip = contextMenuStrip1;
24              textBox2.ContextMenuStrip = contextMenuStrip1;
25          }
26
                ссылка: 1
27          private void выходToolStripMenuItem_Click(object sender, EventArgs e)
28          {
29              Application.Exit();
30          }
31
                ссылка: 1
32          private void обновитьToolStripMenuItem_Click(object sender, EventArgs e)
33          {
34              textBox1.Text = "";
35              textBox2.Text = "";
36          }
```

```
38          private void копироватьToolStripMenuItem_Click(object sender, EventArgs e)
39          {
40              if (textBox1.SelectedText != "")
41                  bufer = textBox1.SelectedText;
42              if (textBox2.SelectedText != "")
43                  bufer = textBox2.SelectedText;
44          }
45
            ссылка: 1
46          private void вставитьToolStripMenuItem_Click(object sender, EventArgs e)
47          {
48              textBox1.Paste(bufer);
49              textBox2.Paste(bufer);
50          }
51      }
52  }
```

Результат





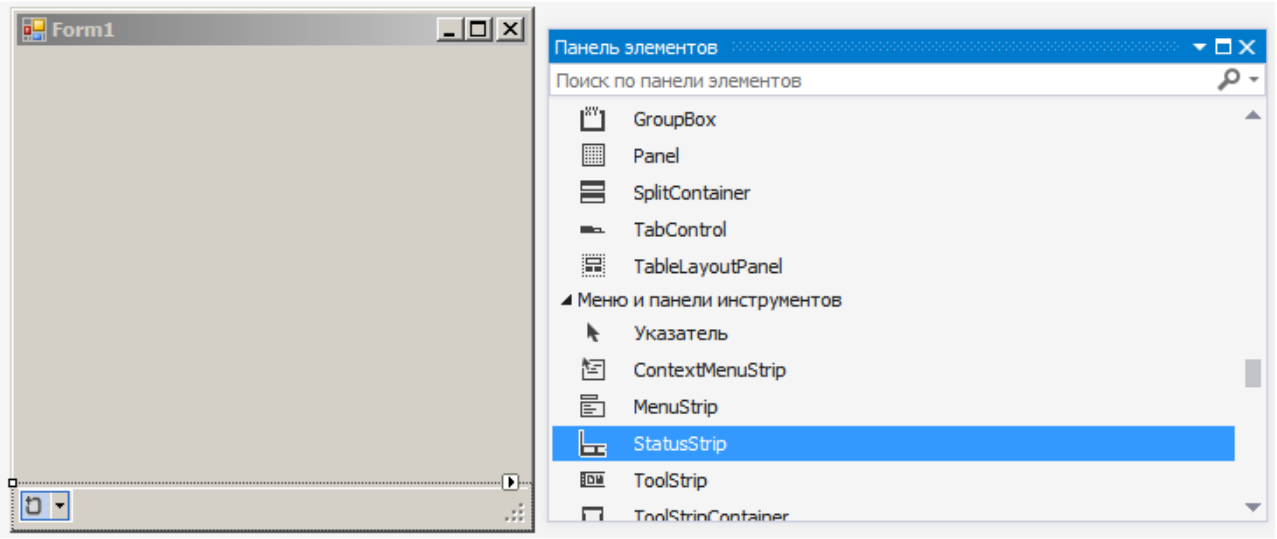


Контрольные вопросы

1. Как создать контекстное меню на C#?
2. Перечислите основные свойства компонента ContextMenuStrip.

5.6. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ СТРОКИ СОСТОЯНИЯ STATUSSTRIP

StatusStrip представляет строку состояния, во многом аналогичную панели инструментов ToolStrip. Строка состояния предназначена для отображения текущей информации о состоянии работы приложения.



При добавлении на форму StatusStrip автоматически размещается в нижней части окна приложения. Однако есть возможность изменения его положения свойством Dock, которое может принимать следующие значения:

Bottom: размещение внизу (значение по умолчанию).

Top: прикрепляет статусную строку к верхней части формы.

Fill: растягивает на всю форму.

Left: размещение в левой части формы.

Right размещение в правой части формы.

None: произвольное положение.

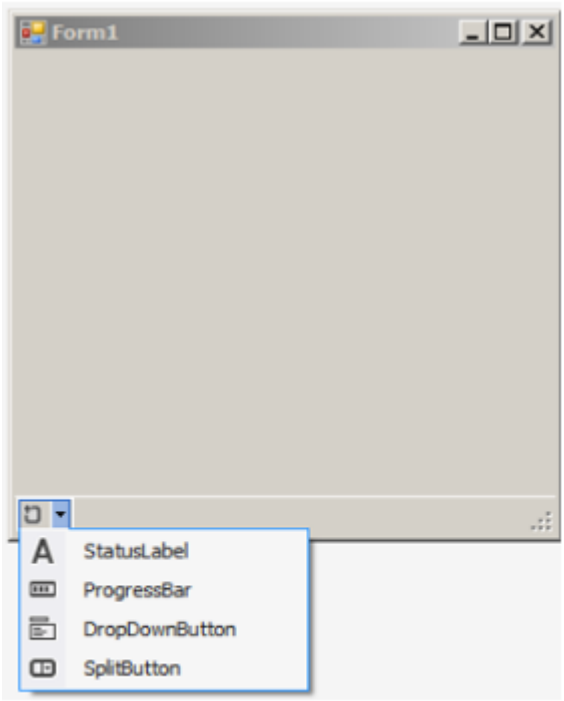
StatusStrip может содержать различные элементы, которые можно добавлятьв режиме дизайнера можно добавить следующие типы элементов:

StatusLabel – метка для вывода текстовой информации. Представляет объект ToolStripLabel.

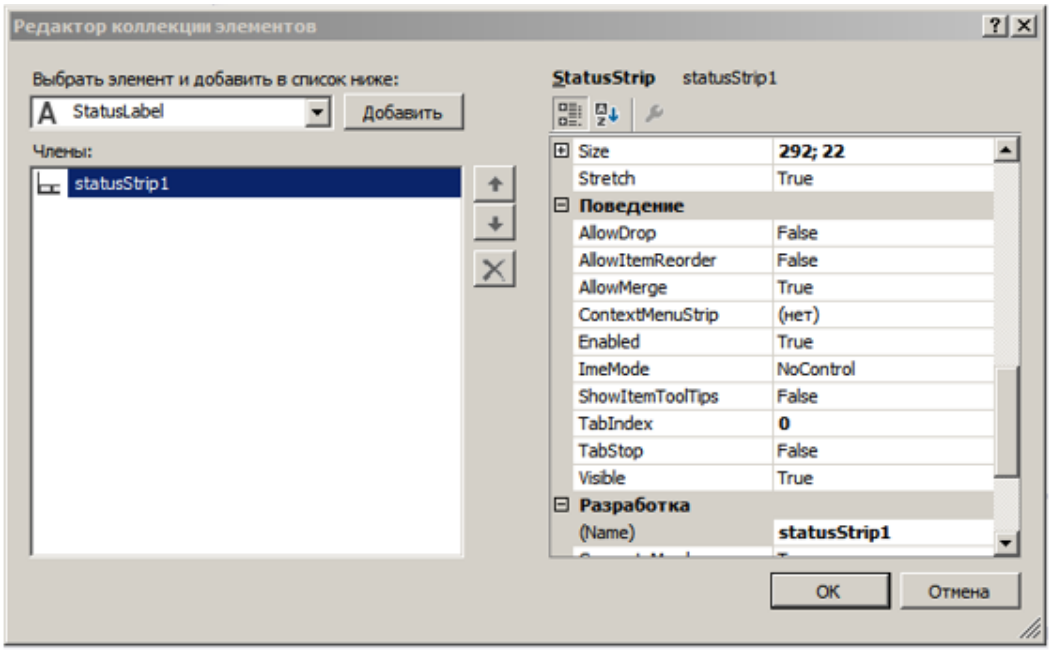
ProgressBar – индикатор прогресса, представляет объект ToolStripProgressBar

DropDownButton – кнопка с выпадающим списком по клику, представляет объект ToolStripDropDownButton.

SplitButton – еще одна кнопка, во многом аналогичная DropDownButton, представляет объект ToolStripSplitButton.



Также можно обратиться на панели свойств к свойству Items компонента StatusStrip и в открывшемся окне добавить и настроить все элементы:

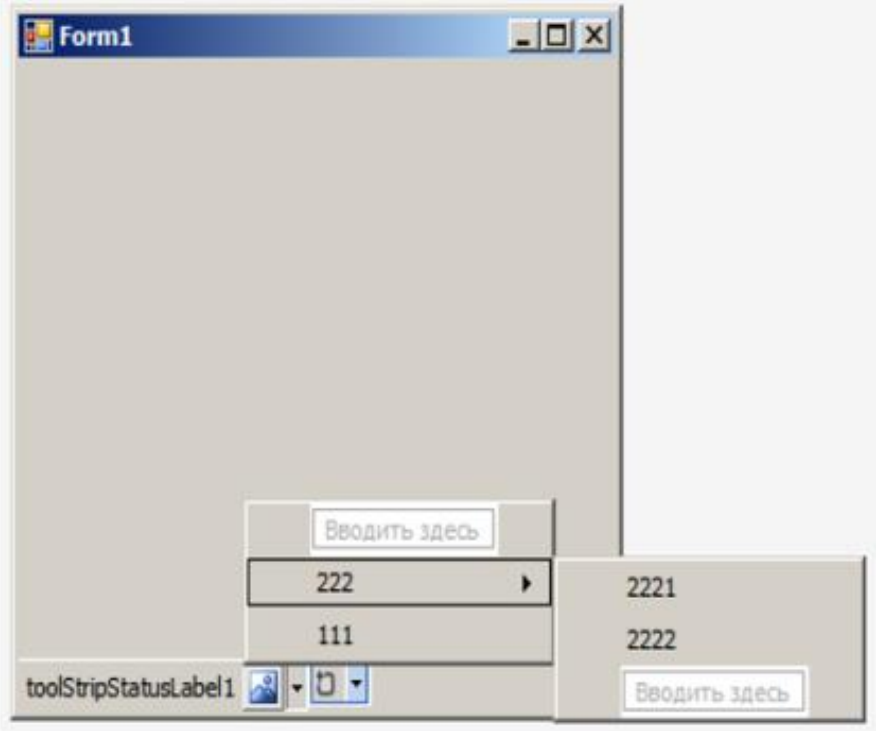


Для отображения даты и времени следует воспользоваться следующим фрагментом кода:

```
toolStripStatusLabel1.Text = String.Format("{0} ({1}),
DateTime.Now.ToString("dd.MM.yyyy - hh:mm",DateTime.Now.DayOfWeek);
```

где DateTime представляет текущее время, обычно выраженное как дата и время суток. Now – возвращает объект DateTime, которому присвоены текущие дата и время данного компьютера, выраженные как местное время. DayOfWeek – возвращает день недели, представленный этим экземпляром.

SplitButton на StatusStrip позволяет создавать раскрывающееся меню.



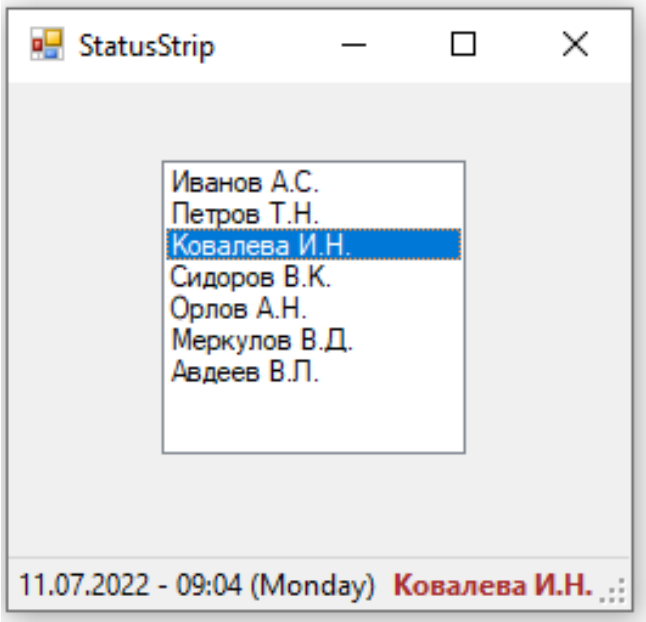
Пример.

Создать оконное приложение, содержащее listbox со списком из фамилий, а также строку состояния (на ней находятся метка, отображающая текущую дату и время и метку, показывающую, какая фамилия выбрана в списке listbox).

Код программы

```
11  namespace WindowsFormsApp11
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
            ссылка: 1
15          public Form1()
16          {
17              InitializeComponent();
18          }
19
            ссылка: 1
20          private void Form1_Load(object sender, EventArgs e)
21          {
22              toolStripStatusLabel1.Text = String.Format("{0} ({1})",
23              DateTime.Now.ToString("dd.MM.yyyy - hh:mm"), DateTime.Now.DayOfWeek);
24              listBox1.SelectedValueChanged += listBox1_SelectedValueChanged;
25          }
26
            ссылка: 1
27          private void listBox1_SelectedValueChanged(object sender, EventArgs e)
28          {
29              toolStripStatusLabel2.Text = listBox1.SelectedItem.ToString();
30          }
31      }
32  }
```

Результат

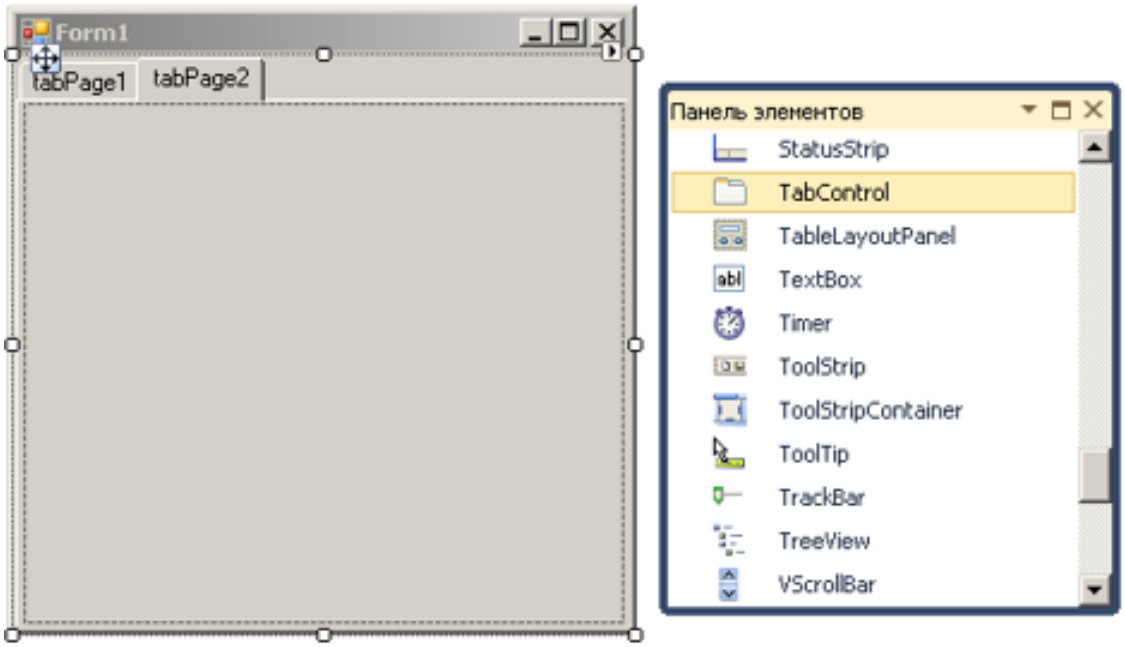


Контрольные вопросы

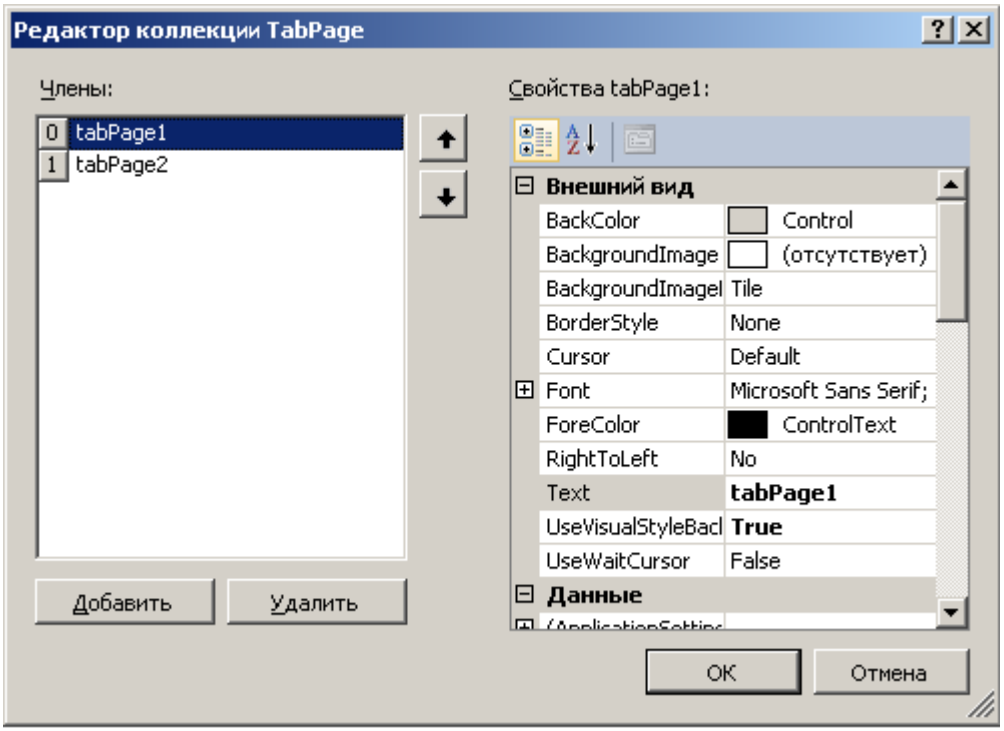
1. Для чего предназначен элемент управления StatusStrip?
2. Перечислите основные свойства компонента StatusStrip.

5.7. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ КОМПОНЕНТА TabCONTROL

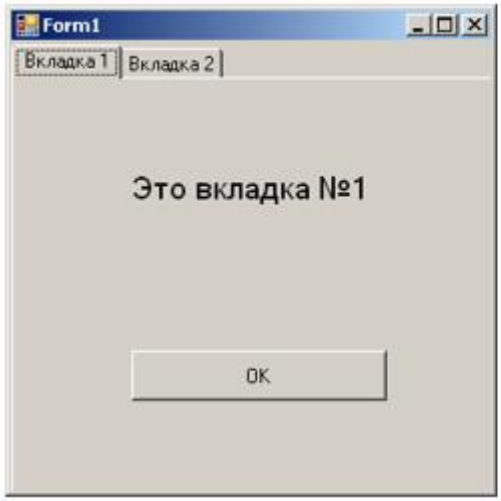
Элемент TabControl позволяет создать элемент управления с несколькими вкладками. И каждая вкладка будет хранить некоторый набор других элементов управления, как кнопки, текстовые поля и др. Каждая вкладка представлена классом TabPage.

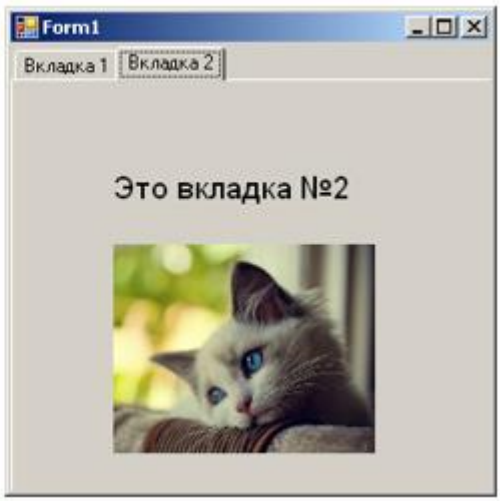


Нам откроется окно редактирования/добавления и удаления вкладок:



Каждая вкладка представляет своего рода панель, на которую мы можем добавить другие элементы управления, а также заголовок, с помощью которого мы можем переключаться по вкладкам. Текст заголовка задается с помощью свойства Text.



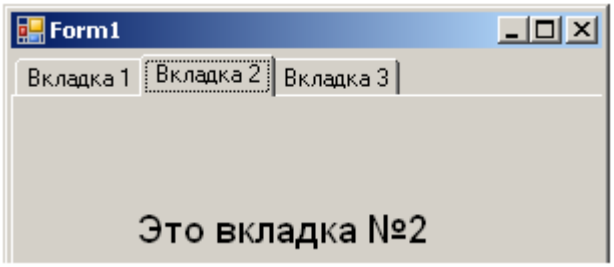


Для добавления новой вкладки нам надо ее создать и добавить в коллекцию tabControl1.TabPages с помощью метода Add

Пример:

```
//добавление вкладки
TabPagenewTabPage = newTabPage();
newTabPage.Text = "Вкладка 3";
```

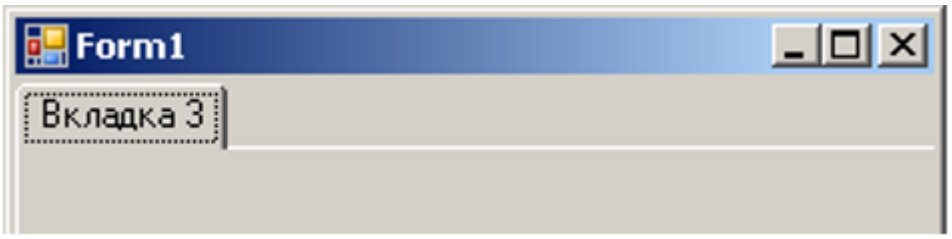
tabControll.TabPages.Add(newTabPage);



Для удаления следует воспользоваться методом Remove или RemoveAt

Пример:

```
// удаление вкладки по индексу
tabControl1.TabPages.RemoveAt(1);
// удаление вкладки по объекту
tabControl1.TabPages.Remove(tabPage1);
```



Получая в коллекции tabContol1.TabPages нужную вкладку, мы можем ей манипулировать по индексу.

Пример:

```
// изменение свойств
tabControl1.TabPages[0].Text = "Единственная вкладка";
```

Пример 1.

Создайте приложение, состоящее из 3 вкладок: «Автор». «Задача1» и «Задача2». На вкладках с задачами решить задачи:

1) Возвести в степень y число x (x и y вводятся с помощью TrackBar);

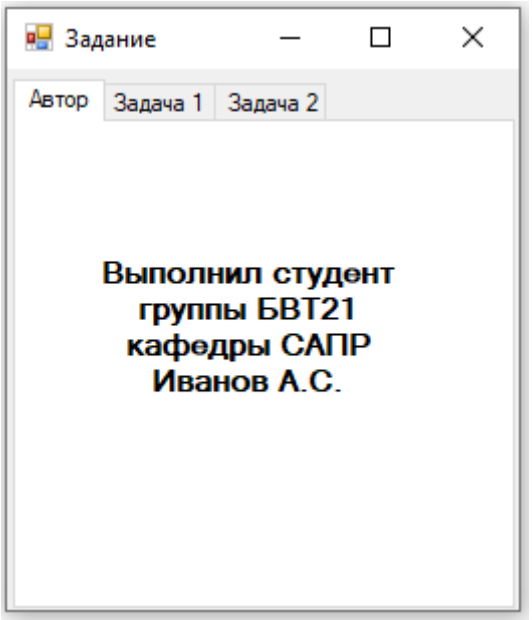
2) Сгенерировать одномерный массив, состоящий из 10 элементов (элементы принимают значения в интервале [–13:11]), вывести его на экран. Найти минимальный элемент массива.

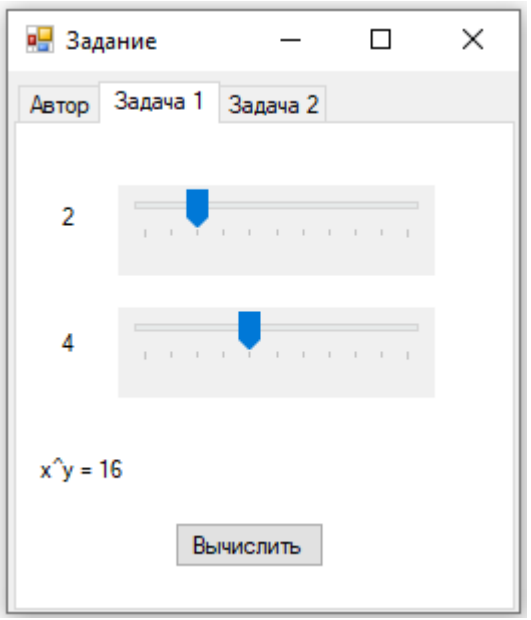
Код программы

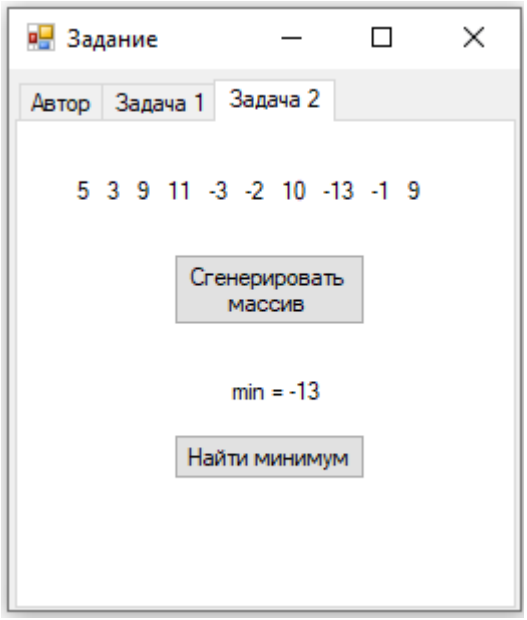
```
11  namespace WindowsFormsApp14
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
15          int[] a = new int[10]; int min = 100;
            ссылка: 1
16          public Form1()
17          {
18              InitializeComponent();
19          }
20
            Ссылок: 2
21          private void trackBar1_ValueChanged(object sender, EventArgs e)
22          {
23              double x = trackBar1.Value;
24              label2.Text = Convert.ToString(x);
25          }
26
27          private void Form1_Load(object sender, EventArgs e)
28          {
29              trackBar1.ValueChanged += trackBar1_ValueChanged;
30              trackBar2.ValueChanged += trackBar2_ValueChanged;
31          }
32
            Ссылок: 2
33          private void trackBar2_ValueChanged(object sender, EventArgs e)
34          {
35              double y = trackBar2.Value;
36              label3.Text = Convert.ToString(y);
37          }
38
            ссылка: 1
39          private void button1_Click(object sender, EventArgs e)
40          {
41              double x = trackBar1.Value;
42              label2.Text = Convert.ToString(x);
43              double y = trackBar2.Value;
44              label3.Text = Convert.ToString(y);
45              double c = Math.Pow(x, y);
46              label4.Text = "x^y = " + c;
47          }
48
49          private void button2_Click(object sender, EventArgs e)
50          {
51              label5.Text = "";
52              Random rn = new Random();
53              int[] a = new int[10];
54              for (int i = 0; i < 10; i++)
55              {
56                  a[i] = rn.Next(25) - 13;
57                  label5.Text += a[i] + "   ";
58                  if (a[i] < min) { min = a[i]; }
59              }
60          }
61
```

```
62          ссылка: 1
            private void button3_Click(object sender, EventArgs e)
63          {
64              label6.Text = "";
65              label6.Text = "min = " + min;
66          }
67      }
68      }
```

Результат





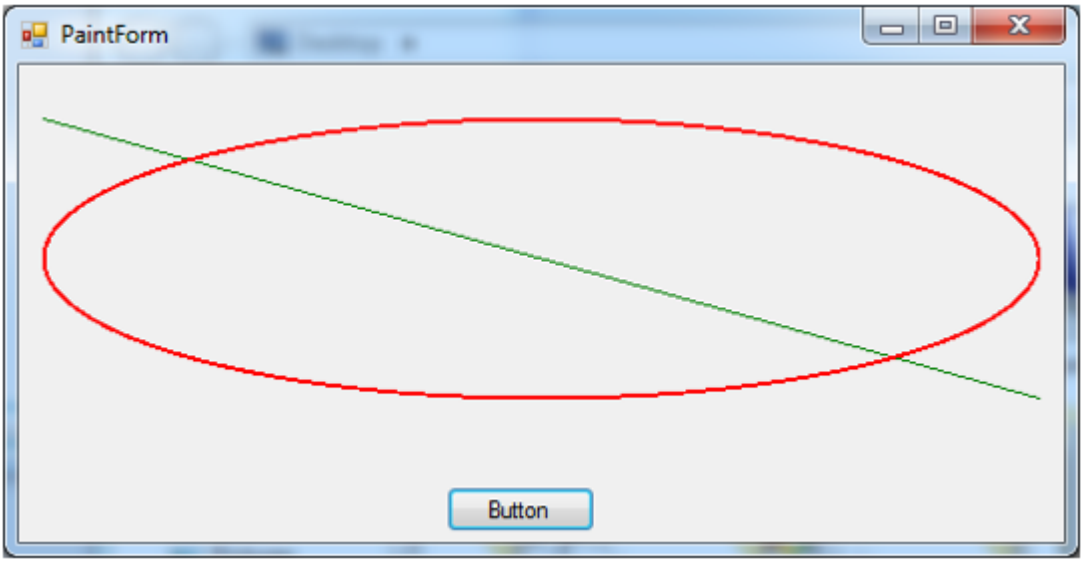


Контрольные вопросы

1. Что такое TabControl?
2. Как добавить TabPage?

5.8. ГРАФИКА НА C#. СВОЙСТВА, МЕТОДЫ И СОБЫТИЯ КОМПОНЕНТА PictureBox

Среда позволяет загрузить в приложение готовые картинки и фотографии, разработать программы, которые выводят графику на поверхность объекта.



Рисованные изображения отображаются на форме при выполнении программы с помощью различных инструментов. Каждая точка на форме имеет координаты X и Y. Текущая позиция при рисовании определяется горизонтальной (X) и вертикальной (Y) координатами, заданными в пикселях.

Основным классом для «рисования» в языке C# является класс Graphics. Он предназначен для вывода графической информации в клиентскую часть формы приложения. Для того чтобы приложение могло что-нибудь нарисовать в окне, оно должно получить или создать для этого окна объект класса Graphics. В C# инструменты рисования определены в пространстве имен System.Drawing.

Объект Graphics – указатель на место, где будут рисоваться примитивы. Если нужно рисовать в форме Windows. Синтаксис задания ссылки на нее:

Graphics g = Graphics.FromHwnd(this.Handle);

Здесь:

Graphics – тип объекта, g – имя переменной. FromHwnd – метод из класса Graphics, который задает ссылку Handle на форму Windows (т.е. рисование будет на самой форме).

Для создания объекта g типа Graphics для перерисовываемой области (по умолчанию всей формы) используется синтаксис:

Graphics g = e.Graphics; или Graphics g = this.CreateGraphics();

Объекты пера используются в методах рисования линий и графических фигур.

Объекты Pen выбираются из класса Pens (перья).

Класс Pens содержит набор объектов для выбора. У них толщина линии (1 пиксель), стиль линии – сплошная. У каждого объекта свой цвет линии, имя которого определяет объект. Например, создаем объект myPen:

PenmyPen = Pens.Black;

Объекты Pen с изменяемыми свойствами создаются из класса Реп (перо). Основные свойства объекта пера:

Color-цвет линии;

Brush – ссылка на кисть, используемую в качестве пера;

Width – толщина линии;

DashStyle – стиль штриховой линии;

DotStyle – стиль пунктирной линии;

DashDotStyle – штрих пунктир;

DashDotDotStyle – штрих двойной пунктир;

SolidStyle – стиль непрерывная линия;

Сначала объект myPen можно создать с указанием цвета и толщины линии 2 пикселя.

Pen myPen = new Pen(Color.Red, 2);

Затем можно изменить его свойства.

Пример:

myPen.Width =3;

Также можно задать только цвет линии: PenmyPen = newPen(Color.Red);

Объекты кисти используются в методах заливки графических фигур.

Определены кисти разного типа:

Brush – простая кисть, одноцветная заливка.

HatchBrush – кисть со штриховой заливкой.

LinearGradientBrush – кисть с линейной градиентной заливкой, цвет фрагментов фигуры меняется плавно.

PathGradientBrush – кисть с градиентной заливкой, цвет фрагментов фигуры меняется скачкообразно.

Объекты Brush выбираются из класса Brushes, который содержит кисти со сплошной заливкой. Класс Brushes содержит набор объектов для выбора, у которых по умолчанию определен цвет.

Например, создаем объект myBrush с синей заливкой:

BrushmyBrush = Brushes.Blue; // Заливка синим

Объекты HatchBrush выбираются из класса HatchBrushes. Класс HatchBrushes содержит набор объектов для выбора, у которых у которых по умолчанию определены стиль заливки HatchStyle, цвет переднего плана ForeColor и цвет фона BackColor (задний план).

Определены следующие стили заливки:

– сетка (Cross);
– диагональная сетка (DiagonalCross);
– прямая диагональ (ForwardDiagonal);
– обратная диагональ (BackwardDiagonal) и др.

Например, создадим кисть с заливкой сеткой HatchStyle.Cross:

HatchBrush brush2 = new HatchBrush(HatchStyle.Cross, ForeColor, BackColor);

Для вывода на форму текста используется метод DrawString. Рисует заданный текст в заданном прямоугольнике. Синтаксис метода:

DrawString(S, Font, Brush, RectangleF, StringFormat);

S – текст.

Font-шрифт текста.

Brush – задает цвет и текстуру текста.

RectangleF – прямоугольник вывода.

StringFormat – задает атрибуты форматирования (междустрочный интервал и выравнивание и др.) текста.

Формат можно не указывать, будет использован формат по умолчанию.

Font – шрифт текстовой строки. Выбирается с помощью методов класса Font.

Например:

fontМойШрифт = new Font("Arial", 24, FontStyle.Bold);

fontШрифт = new Font("Arial Narrow", 14); // без учета начертания шрифта;

RectangleF – прямоугольник, в котором рисуется строка текста. Задается двумя способами:

– 4 координаты – левого верхнего (X1,Y1) и правого нижнего (X2,Y2) углов;
– точка с координатами левого верхнего угла (P) и размеры (H – ширина, W – высота прямоугольника).

Как и большинство классов C#, класс Graphics имеет свойства и методы. Рассмотрим некоторые из них.

Clear() – этот метод заполняет объект Graphics выбранным пользователем цветом, удаляя его предыдущее содержимое.

Группа методов для «рисования» некоторых геометрических фигур:

DrawArc(), DrawBezler(), DrawBeziers(), DrawCurve(), DrawEllipse(), Drawlcon(), DrawLine(), DrawLines(), DrawPie(),DrawPath(), DrawRectange(), DrawRectangles(), DrawString().

Для «заполнения» внутренних областей геометрических фигур используются методы, перед которыми находится слово Fill, например: FillPie(), FillElllpse() или FillRectangle().

DrawLine – метод рисования линии между двумя точками. Синтаксис метода: g.DrawLine(pen, x[1], y[1], x[2], y[2],);

Здесь: g – где рисуем, DrawLine – метод рисования линии, pen – перо, x[1], y[1], x[2], y[2] – точки границы линии.

DrawRectangle – метод рисования прямоугольника. Синтаксис метода:

g. DrawRectangle (pen, rect);

Здесь: g – где рисуем, DrawRectangle – метод рисования прямоугольника, pen – перо, rect – прямоугольник, свойства которого задаются целыми числами.

rectangleПрямоугольник = new Rectangle(x, y, w, h);

Здесь: x, y – координаты левого верхнего угла, w – ширина прямоугольника, h – высота прямоугольника. Возможен вариант задания свойств числами в формате с плавающей точкой.

DrawEllipse – метод рисования эллипса.

Синтаксис метода: g.DrawEllipse (pen, rect);

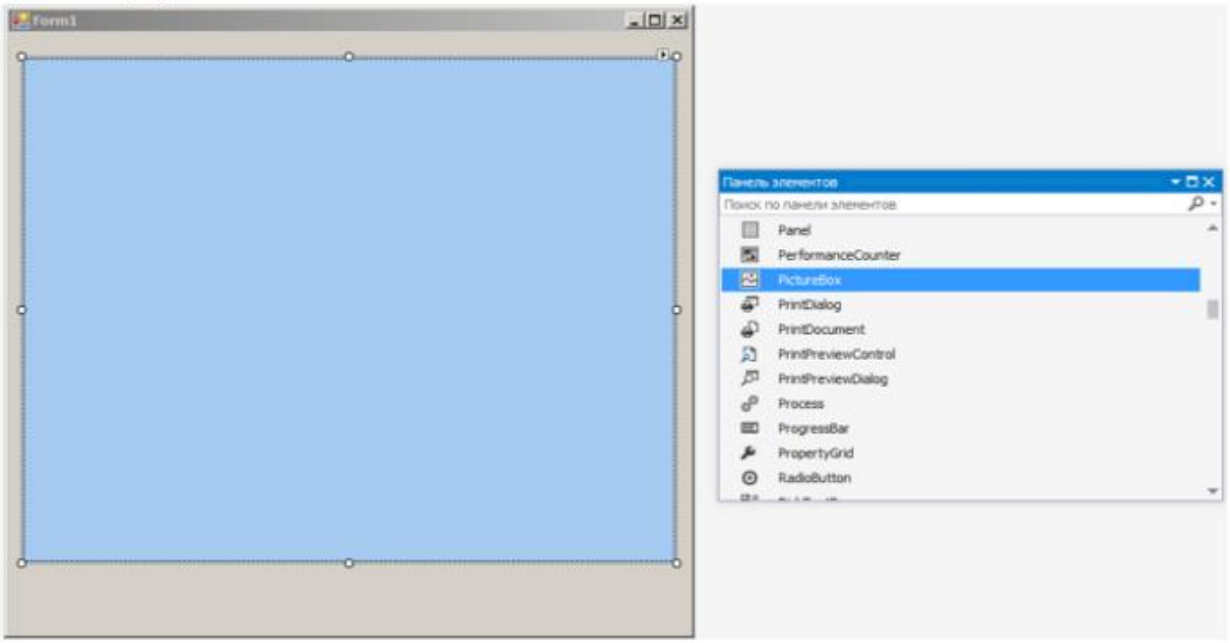
Здесь: g – где рисуем, DrawEllipse – метод рисования эллипса, pen – перо, rect – прямоугольная область, в которую вписывается эллипс.

FillRectangle–метод закрашивания прямоугольника.

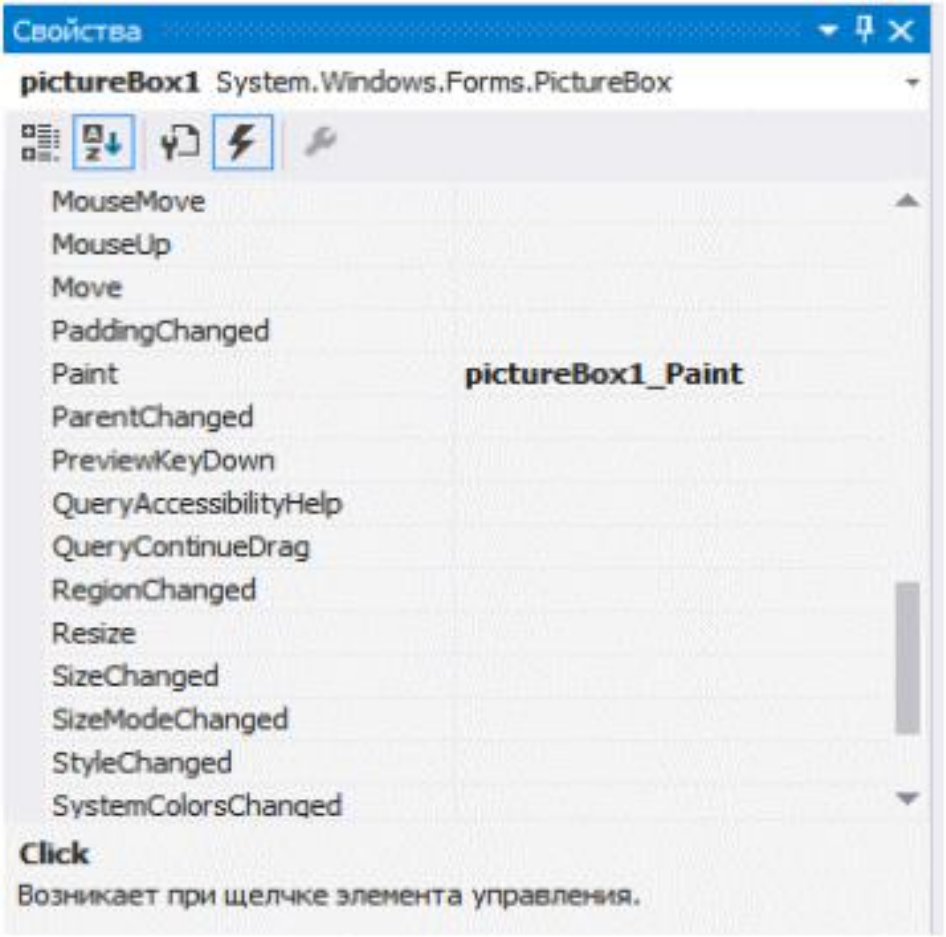
Синтаксис метода: g. FillRectangle (brush, p);

g – где рисуем, FillRectangle – метод рисования залитого прямоугольника, brush – кисть, p – массив точек. Получаемый результат зависит от стиля заливки.

Для работы с графикой в C# также можно использовать компонент PictureBox.



Для рисования в PictureBox необходимо создать для него событие Paint.



Пример рисования эллипса в PictureBox.

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Brush b = Brushes.Blue;
    g.FillEllipse(b, 30, 20, 200, 300);
}
```

Пример.

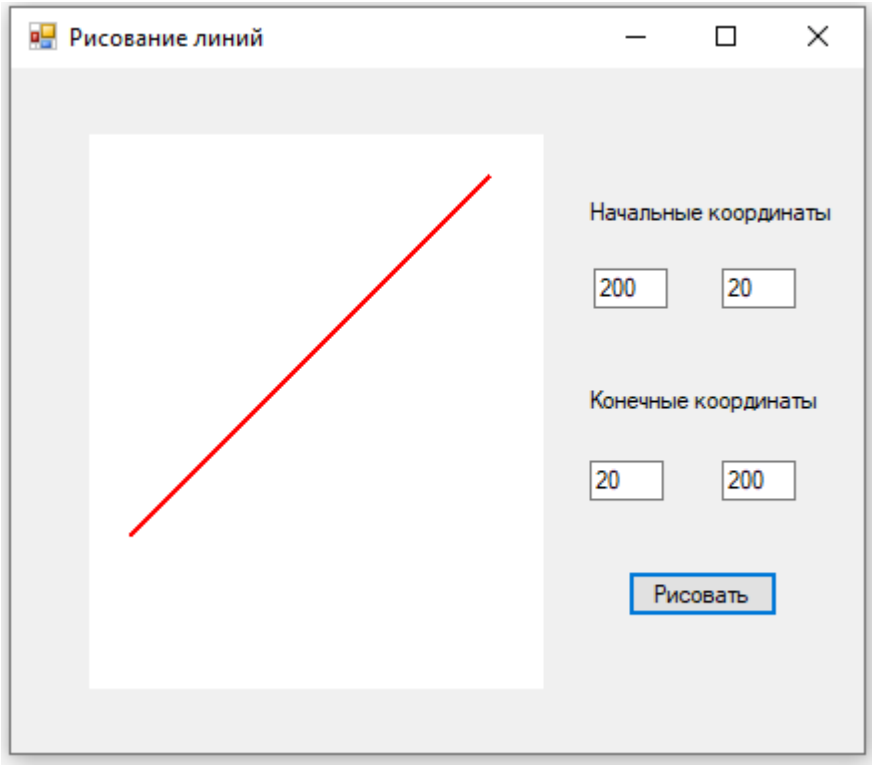
Создать приложение, с помощью которого можно нарисовать линию с заданными цветом и толщиной в заданном месте поля "PictureBox".

Код программы

```
11  namespace WindowsFormsApp13
12  {
        Ссылок: 3
13      public partial class Form1 : Form
14      {
15          int[] m_p = new int[5];
            ссылка: 1
16          public Form1()
17          {
18              InitializeComponent();
19          }
20
```

```
        ссылка: 1
        private void button1_Click(object sender, EventArgs e)
        {
            m_p[1] = Convert.ToInt32(textBox1.Text);
            m_p[2] = Convert.ToInt32(textBox2.Text);
            m_p[3] = Convert.ToInt32(textBox3.Text);
            m_p[4] = Convert.ToInt32(textBox4.Text);
            pictureBox1.Refresh();
        }

        ссылка: 1
        private void pictureBox1_Paint(object sender, PaintEventArgs e)
        {
            // Рисуем линию
            Pen myPen = new Pen(Color.Red, 2);
            e.Graphics.DrawLine(myPen, m_p[1], m_p[2], m_p[3], m_p[4]);
        }
    }
}
```

Результат



Пример.

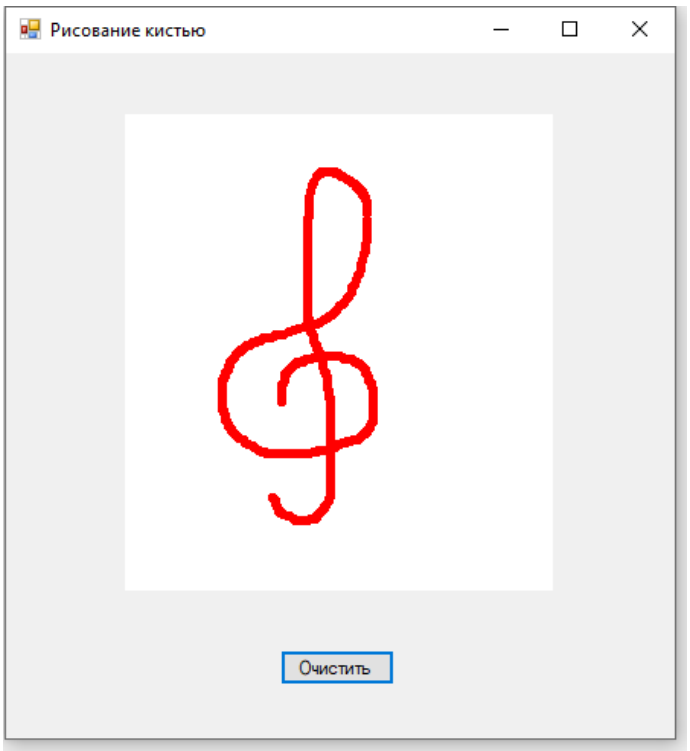
Создать приложение с возможностью рисования произвольных линий кистью. Предусмотреть кнопку для очистки поля рисования.

Код программы

```
namespace WindowsFormsApp12
{
    Ссылок: 3
    public partial class Form1 : Form
    {
        bool Drow;
```

```
16          ссылка: 1
            public Form1()
17          {
18              InitializeComponent();
19          }
            ссылка: 1
20          private void button1_Click(object sender, EventArgs e)
21          {
22              Graphics graf = pictureBox1.CreateGraphics();
23              graf.Clear(SystemColors.Window);
24          }
            ссылка: 1
25          private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
26          {
27              Graphics graf = pictureBox1.CreateGraphics();
28              if (Drow == true)
29              {
30                  graf.FillEllipse(Brushes.Red, e.X, e.Y, 7, 7); // толщина кисти
31              }
32          }
            ссылка: 1
33          private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
34          {
35              Drow = true;
36          }
37
            ссылка: 1
38          private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
39          {
40              Drow = false;
41          }
42      }
43  }
```

Результат



Контрольные вопросы

1. Какой класс используется для «рисования» в языке C#?
2. Перечислите основные методы и свойства класса Graphics.
3. Приведите пример создания одного из геометрических примитивов.
4. Что такое PictureBox?

ЗАКЛЮЧЕНИЕ

В учебном пособии представлены вопросы, касающиеся объектно-ориентированного программирования. Используются знания, полученные на курсе информатики, а именно – основы программирования на языке C#. Учебное пособие ориентировано на самостоятельную работу студентов над различными учебными проектами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. **Кудрина, Е. В.** Основы алгоритмизации и программирования на языке C# : учебное пособие для СПО / Е. В. Кудрина, М. В. Огнева. – М. : Юрайт, 2019. – 322 с.

2. **Зыков, С. В.** Программирование. Объектно-ориентированный подход : учебник и практикум для академического бакалавриата / С. В. Зыков. – М. : Юрайт, 2019. – 155 с.

3. **НОУ ИНТУИТ.** Основы программирования на C#: Информация [Электронный ресурс]. – URL : http://www.intuit.ru/studies/courses/2247/18/info/

4. **Руководство** по программированию в WindowsForms [Электронный ресурс]. – URL : https://metanit.com/sharp/windowsforms/

ОГЛАВЛЕНИЕ

Учебное электронное издание

ТОЛСТЫХ Светлана Германовна
КОРОБОВА Ирина Львовна
МАЙСТРЕНКО Наталья Владимировна

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ C#

Учебное пособие

Редактирование И. В. Калистратовой
Графический и мультимедийный дизайнер Н. И. Кужильная
Обложка, упаковка, тиражирование И. В. Калистратовой

ISBN 978-5-8265-2615-6

9 785826 526156

Подписано к использованию 28.08.2023.
Тираж 50 шт. Заказ № 90

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14
Тел./факс (4752) 63-81-08.
E-mail: izdatelstvo@tstu.ru