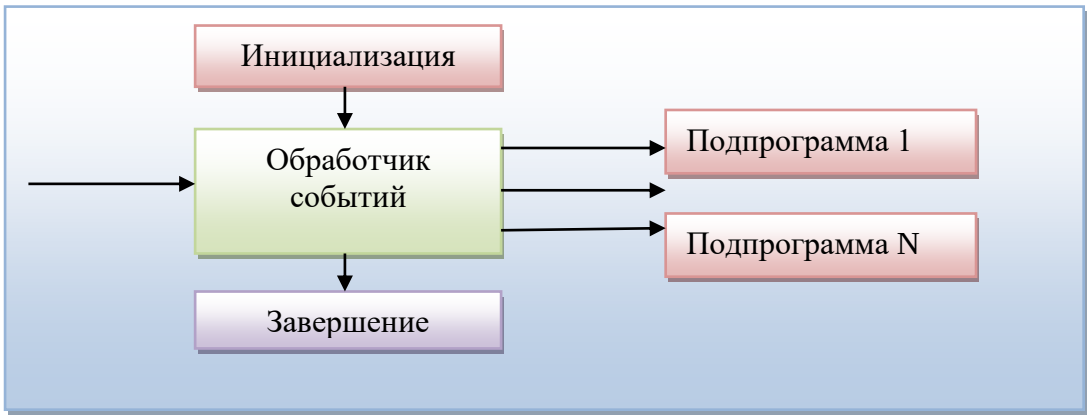
Раздел 3 Разработка приложений в Visual Studio2008
Тема 3.1 Работа в среде Visual Studio2008

В основу Windows положен принцип *событийного управления.* Это значит, что и сама система, и приложения после запуска ожидают действий пользователя и реагируют на них заранее заданным образом. Любое действие пользователя (нажатие клавиши на клавиатуре, щелчок кнопкой мыши, перемещение мыши) называется *событием.* Структура программы, управляемой событиями, изображена на рис. 1.



Событие воспринимается Windows и преобразуется в *сообщение* - запись, содержащую необходимую информацию о событии (например, какая клавиша была нажата, в каком месте экрана на произошел щелчок мышью). Сообщения могут поступать не только от пользователя, но и от самой системы, а также от активного или других приложении. Определен достаточно широкий круг стандартных сообщений, образующий иерархию, кроме того, можно определять собственные сообщения.

Сообщения поступают в общую очередь, откуда распределяются по очередям приложений. Каждое приложение содержит *цикл обработки сообщений,* который выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки (рис. 14.2). Таким образом, Windows-приложение состоит из главной программы, обеспечивающей инициализацию и завершение приложения, цикла обработки сообщений и набора *обработчиков событий.*

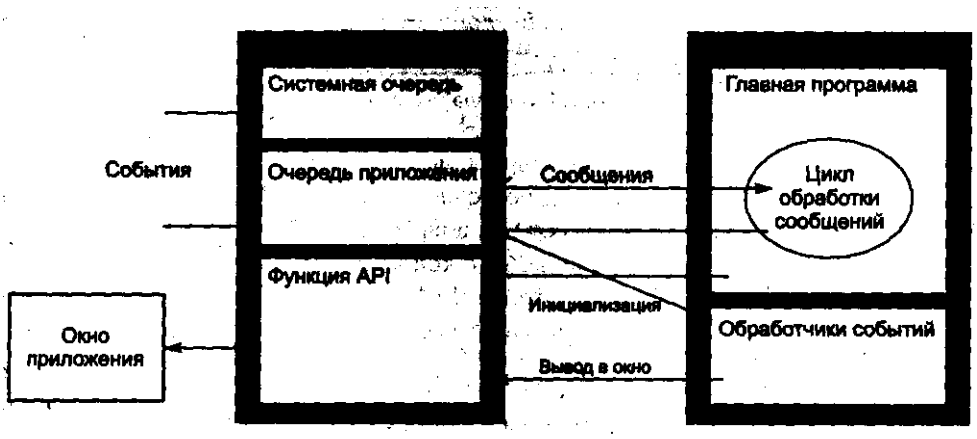


Рис. 14.2. Структура Windows-приложения

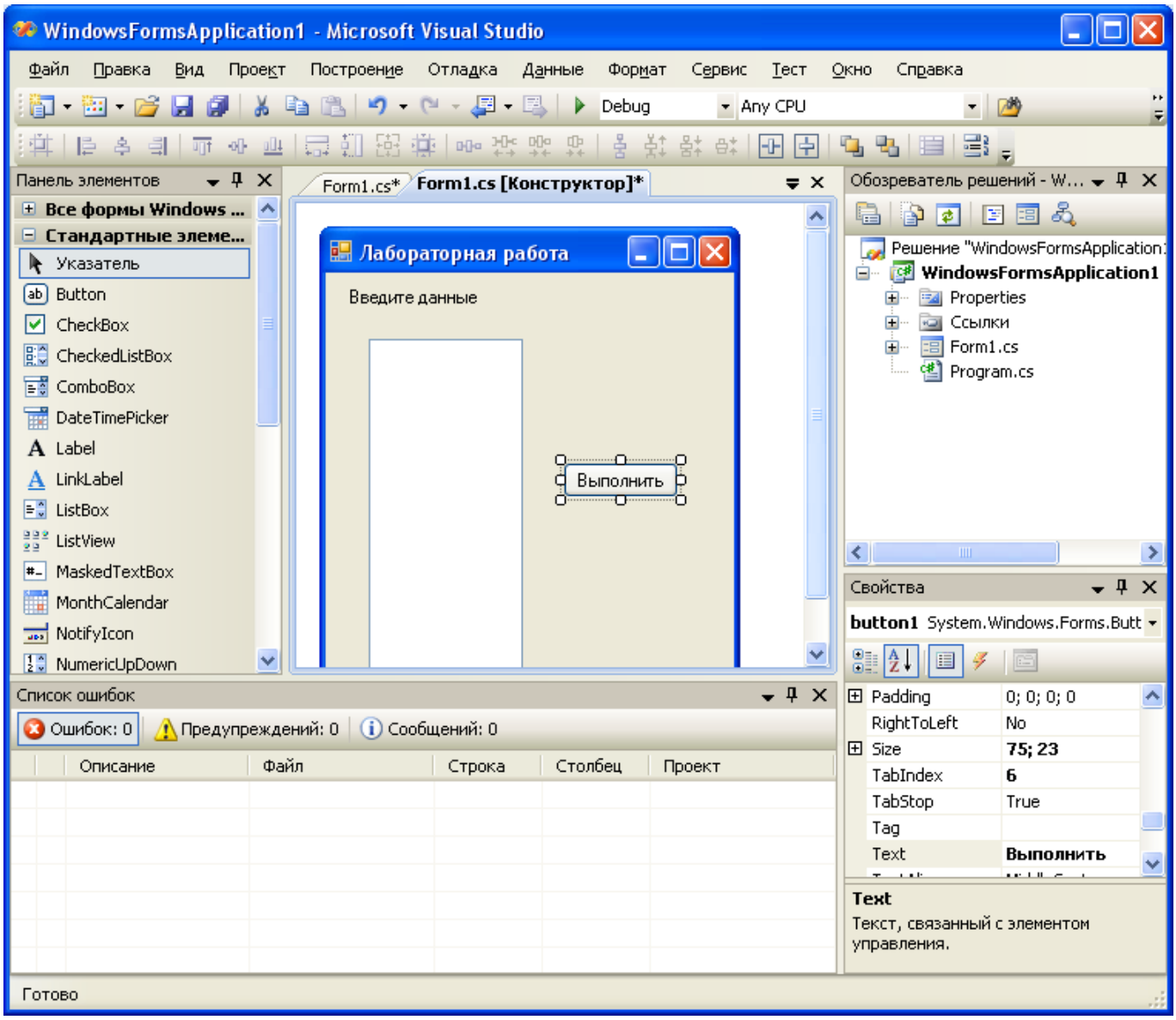
Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу — создание шаблонов

приложения и форм, заготовок обработчиков событий, организацию цикла обработки сообщений и т. д.

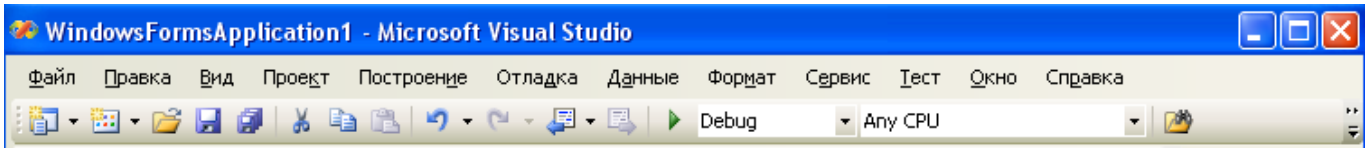
Любая современная система программирования (и Turbo C# в том числе) сопровождается средствами отладки программы синтаксического контроля правильности вводимого кода и массой других вспомогательных механизмов, которые в совокупности образу интегрированную среду разработки (IDE - Integrated Development Evironment). В этом разделе вы познакомитесь с основными возможностями IDE Turbo C#. Для более полного знакомства со средой берите ссылку Tour of the IDE на странице Welcome Page.

Основные панели среды

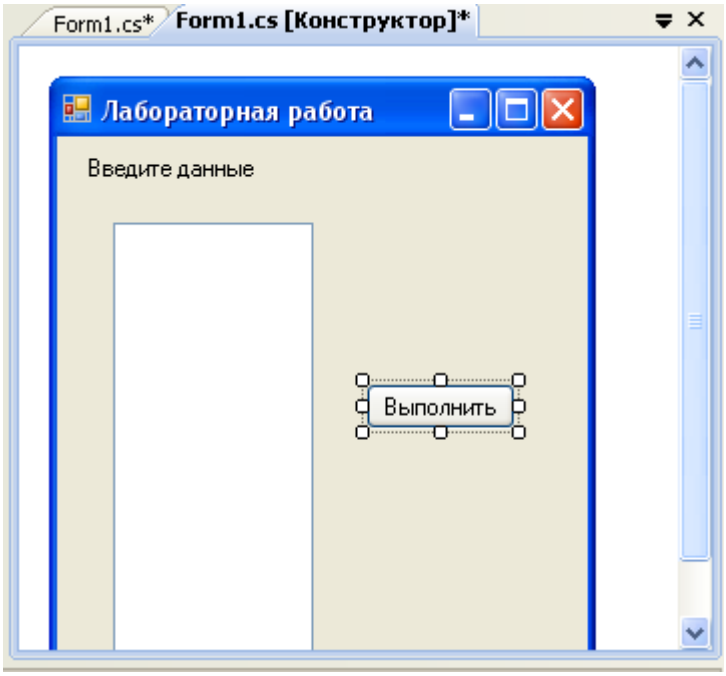
Окно среды программирования содержит шесть панелей (см. с. 16.3).

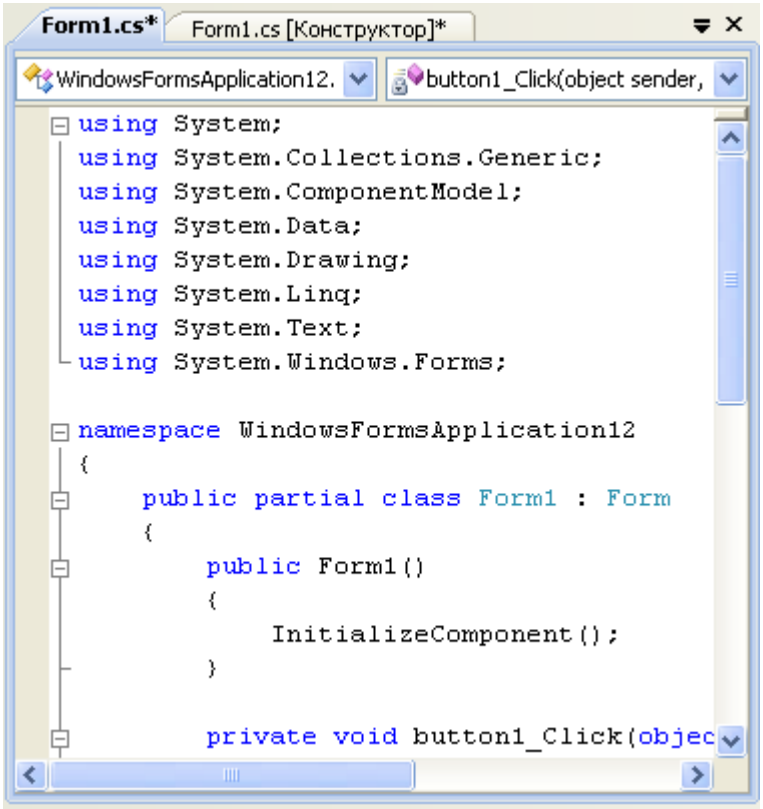


На *управляющей панели* находятся заголовок, главное меню и инструментальные кнопки. Все управление средой доступно с помощью команд главного меню, но некоторые часто используемые команды можно дать, щелкнув на соответствующих инструментальных кнопках. В верхней правой части управляющей панели имеется список выбора, определяющий режим отображения остальных панелей.

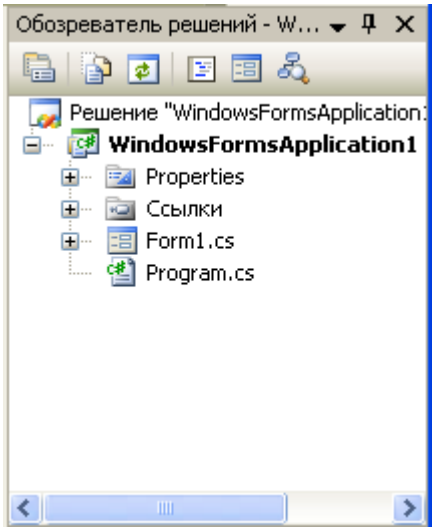


Центральная *панель формы/кода* предназначена для отображения конструктора формы и программного кода.

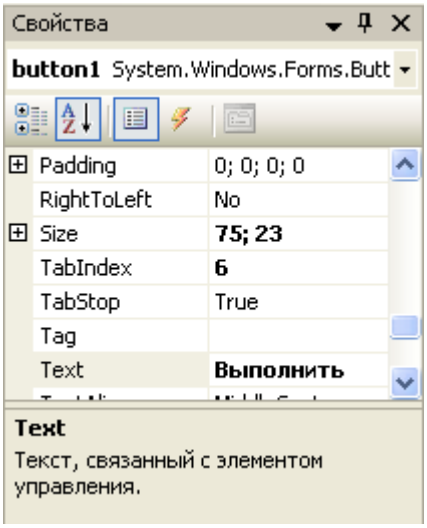




Панель *обозревателя решений* снабжена отображает все связанные с проектом файлы. Двойной щелчок по имени файла приводит к показу содержимого в окне кода.



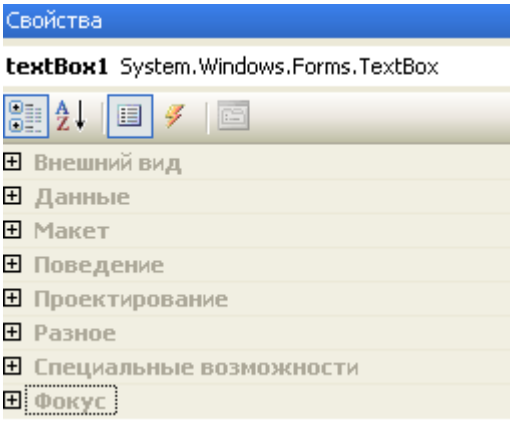
Панель *инспектора объектов* имеет инструментальную панель на которой представлены следующие возможности:



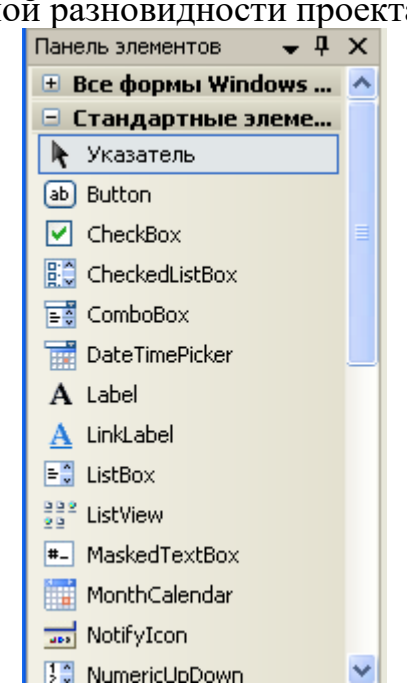
1) Сортировка по категориям
2) Сортировка по имени
3) Отображение свойств компонент
4) Отображение событий компонент

   При отображении свойств компонент они разбиты на группы:

   1) Внешний вид
   2) Данные
   3) Макет
   4) Поведение
   5) Проектирование
   6) Разное
   7) Специальные возможности
   8) Фокус
   9)

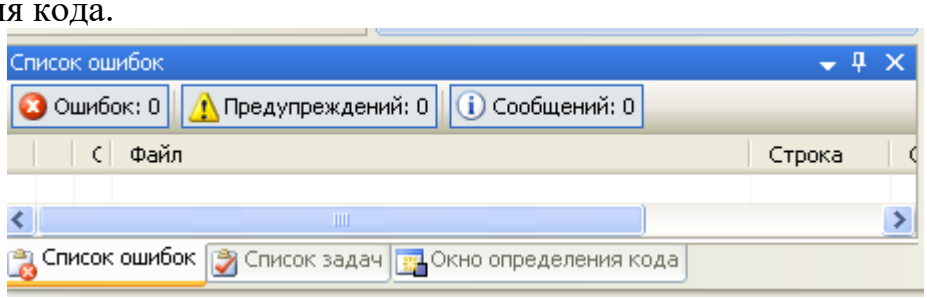


Наконец, правая нижняя *панель палитры компонентов* содержит все доступные компоненты для данной разновидности проекта.



Поскольку в палитре содержится несколько десятков компонентов, в верхней части панели имеет вложенный список отражающий все виды компонент. Щелкните сначала по знаку «минус», а затем выберите нужный компонент.

Во *вспомогательном окне* отражены 3 вкладки: список ошибок, список задач и окно определения кода.



Некоторые возможности редактора кода.

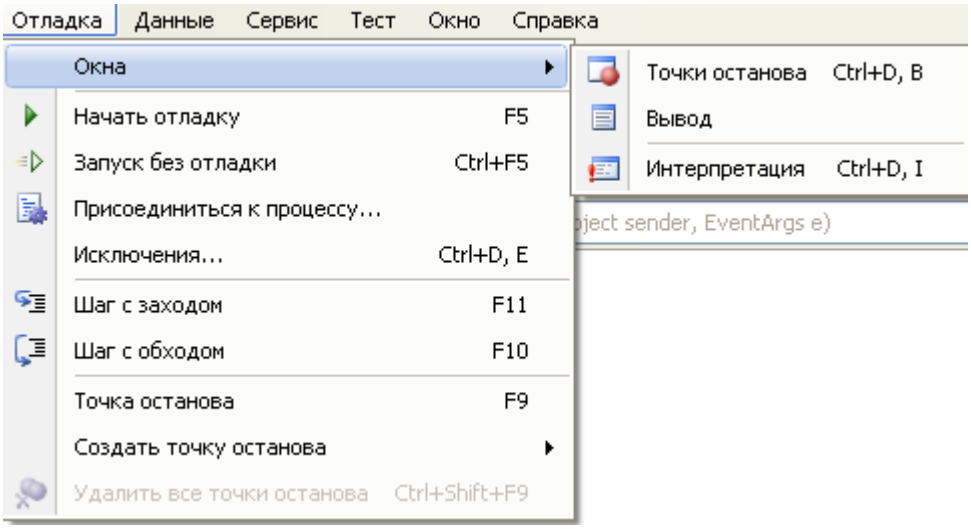
1) для вывода подсказки по именам компонент и свойствам используется клавиша: Ctrl+пробел

2) для вывода подсказки по структурам – Ctrl+J (например: if, while, for)

3) для сворачивания части кода можно использовать квадратики слева от текста проекта

Компиляция проекта

Компиляция проекта осуществляется с помощью горячих клавиш, либо при помощи пункта меню «Отладка» на управляющей панели:



	Горячая клавиша	Описание
	F5	Отладка и компиляция проекта
	Ctrl+F5	Запуск без отладки
	F11	Отладка пошаговая с заходом в процедуры
	F10	Отладка пошаговая без захода в процедуры
	F9	Создание точки останова
	Окна---> Вывод	Вывод значений переменных во вспомогательном окне

Тема 3. 2 Свойства, события и методы

Любой объект содержит свойства, методы и события.

Свойство является важным атрибутом компонента. Свойства служат двум главным целям. Во-первых, они определяют внешний вид формы или компонента. А во-вторых, свойства определяют поведение формы или компонента.

Существует несколько типов свойств, в зависимости от их “природы”, т.е. внутреннего устройства.

- **Простые** свойства - это те, значения которых являются числами или строками.
- **Перечислимые** свойства - это те, которые могут принимать значения из предопределенного набора (списка). Простейший пример - это свойство типа Boolean, которое может принимать значения True или False.
- **Вложенные** свойства - это те, которые поддерживают вложенные значения (или объекты). Инспектор объектов изображает знак “+” слева от названия таких свойств. Имеется два вида таких свойств: множества и комбинированные значения. Инспектор объектов изображает множества в квадратных скобках. Если множество

пусто, оно отображается как []. Установки для вложенных свойств вида “множество” обычно имеют значения типа Boolean. Наиболее распространенным примером такого свойства является свойство Style с вложенным множеством булевых значений. Комбинированные значения отображаются в Инспекторе Объектов как коллекция некоторых величин, каждый со своим типом данных. Некоторые свойства, например, Font, для изменения своих значений имеют возможность вызвать диалоговое окно. Для этого достаточно щелкнуть маленькую кнопку с тремя точками в правой части строки Инспектора Объектов, показывающей данное свойство.

Все изменения значений свойств компонент в режиме выполнения должны осуществляться путем прямой записи строк кода на языке Паскаль. В режиме выполнения невозможно использовать Инспектор объектов . Однако, доступ к свойствам компонентов довольно легко получить программным путем. Все, что Вы должны сделать для изменения какого-либо свойства - это написать простую строчку кода аналогично следующей:

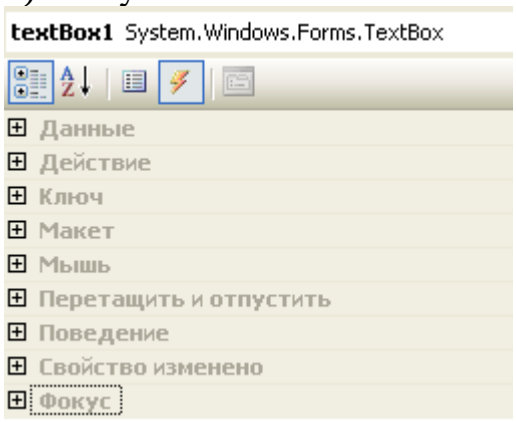
Имя_компоненты.Свойство = Значение;

Пример:
Button1.Text = “Выход”; // надпись на кнопке

События – процедуры обработки, создаются в Инспекторе Объектов, во вкладке Events. Для того чтобы создать обработчик события необходимо напротив строки данного события щелкнуть два раза. Появиться процедура обработки в окне кода программы.

При выборе событий в инспекторе объектов они сгруппированы по категориям:

1) Данные
2) Действие
3) Ключ
4) Макет
5) Мышь
6) Перетащить и опустить
7) Поведение
8) Свойство изменено
9) Фокус



Основные события компонент

Название события	Описание
Click	Щелчок мыши на компоненте и некоторые другие действия пользователя
MouseDown	Нажатие клавиши мыши над компонентом. Возможно распознавание нажатой кнопки и координат курсора мыши
MouseMove	Перемещении курсора мыши над компонентом. Возможно распознавание нажатой кнопки и координат курсора мыши
MouseUp	Отпускание ранее нажатой кнопки мыши над компонентом. Возможно распознавание нажатой кнопки и координат курсора мыши
KeyDown	Событие наступает при нажатии пользователем любой клавиши. Можно распознать нажатые клавиши, включая функциональные, и кнопки мыши, но нельзя распознать символ нажатой клавиши
KeyPress	Событие наступает при нажатии пользователем клавиши символа. Можно распознать только нажатую клавишу символа, различить символ в верхнем и нижнем регистре, различить символы кириллицы и латинские, но нельзя распознать функциональные клавиши и кнопки
KeyUp	Событие наступает при отпускании пользователем любой клавиши. Можно распознать нажатые клавиши, включая функциональные, и кнопки мыши, но нельзя распознать символ отпускаемой клавиши |

События имеют следующий вид:

[модификаторы] <Тип> <Имя> ([<Формальные_параметры>])
{<Тело>}

В квадратных скобках указаны необязательные элементы. Модификаторы определяют область видимости подпрограммы. Сейчас лишь поясню два модификатора — **private и public**. Любые члены класса (в том числе методы-подпрограммы), объявленные с модификатором private, доступны только в методах данного класса. Модификатор public - делает метод (подпрограмму) доступным в любом месте программы. Если модификатор не указан, считается, что данный член класса помечен как закрытый (с модификатором private).

Формальные параметры могут отсутствовать, но и в этом случае круглые скобки за именем подпрограммы обязательны.

Тип подпрограммы может быть любым типом данных. В этом случае подпрограмма представляет собой функцию, которая возвращает результат указанного типа. В теле функции обязательно указывается оператор return, который присваивает функции нужное значение. В качестве типа можно указать

зарезервированное слово void, которое означает отсутствие типа. В этом случае подпрограмма представляет собой процедуру, и использование в ней оператора return означает принудительное завершение ее работы.

Имя события состоит из 2 частей: имени компоненты и имени события. Например: button1_Click (компонента Button1 – событие Click)

Тело подпрограммы обязательно реализуется в виде блока операторов, поэтому за закрывающей круглой скобкой должна следовать открывающая фигурная, даже если тело подпрограммы отсутствует или содержит единственный оператор.

Примеры описаний:

```
private void button1_Click(object sender, EventArgs e)
{}
private void listBox1_KeyPress(object sender, KeyPressEventArgs e)
{}
```

Методы – процедуры предназначенные для работы с компонентой.

Общий вид:

Имя компоненты. имя метода[(параметры)];

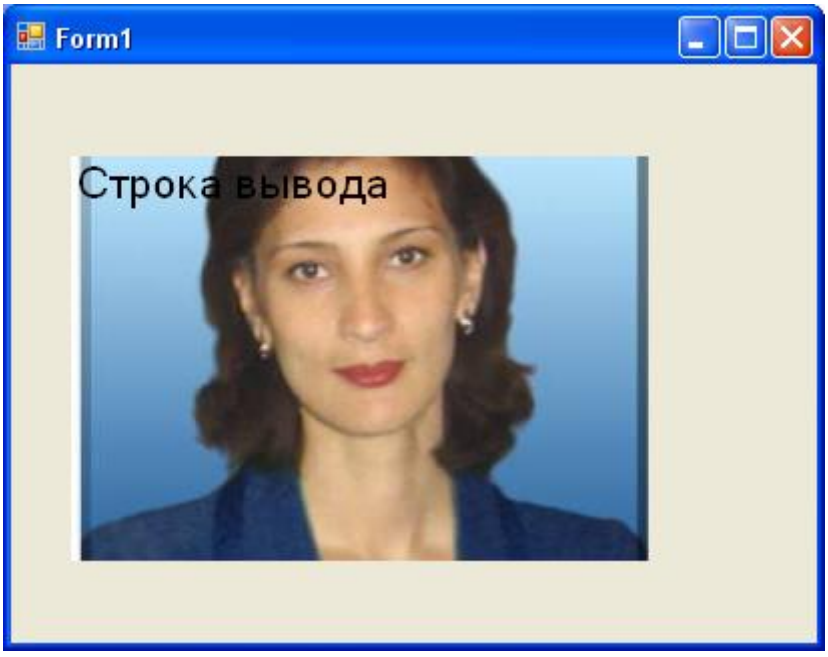
Пример: listBox1.Clear;

Тема 3.3 Работа с текстом (однострочный и многострочный редактор)

Тема 3.3.1 Однострочный и многострочный редактор

Для осуществления ввода и вывода данных используются множество компонент. Рассмотрим некоторые из них: label, textBox, linkLabel.

Label – предназначен для вывода текста в одну или несколько строк, а так же вывод рисунка.



Свойства:

	Наименование свойства	Описание
1.	AutoSize	Автоматическое изменение размера (true/false)
2.	WordWrap	Если true, то слова переносятся на следующую строку. Для того чтобы можно было вывести текст в несколько строк, необходимо изменить свойства: AutoSize – false WordWrap - true

3.	BackColor	Цвет фона:
		label1.BackColor = Color.Red;
		Прозрачность:
		label1.BackColor = Color.Transparent;

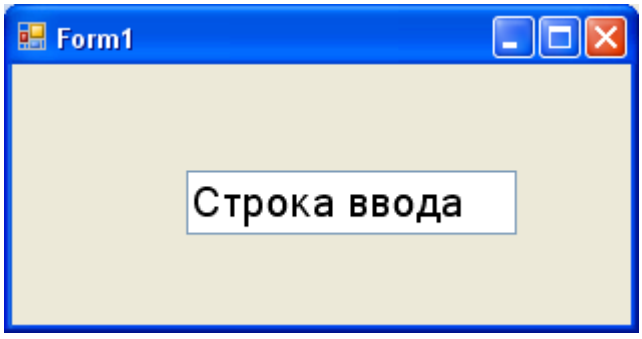
4.	BorderStyle	Рамка
5.	Cursor	Определяет вид курсора при наведении на компоненту
6.	Font	Определяет стиль и размер шрифта текста Name- стиль шрифта Size – размер в пикселях Italic- наклонный шрифт Underline - подчеркнутый
7.	ForeColor	Цвет шрифта label1.ForeColor = Color.Red;
8.	Image	Определяет рисунок
9.	Text	Содержит текст string s=label1.Text;
10.	TextAlign	Выравнивание текста
11.	Enabled	Доступность компоненты (серая) (true/false) label1.Enabled=true;
12.	Visible	Видимость компоненты (true/false) label1. Visible =true;
13.	**ToString**	Форматированный вывод данных: N – для числовых (2 знака после запятой) C- для цены(после числа – тенге или рубли) double Sum; label2.Text = "Сумма = " + Sum.ToSring("N");

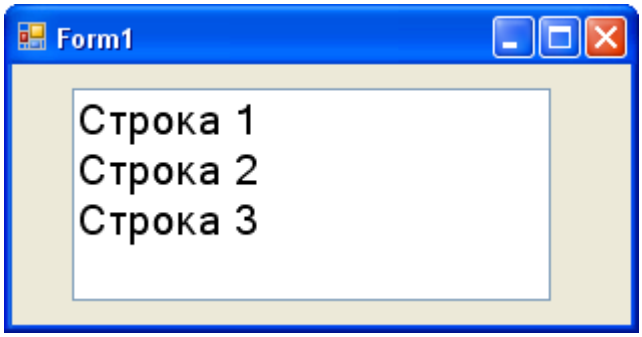
linkLabel- осуществляет вывод строковой информации, а так же существует возможность создания гиперссылок на определенные части текста.

Свойства:

	Наименование свойства	Описание
1.	ActiveLinkColor	Определяет цвет активной гиперссылки
2.	BackColor	Цвет фона
3.	BorderStyle	Рамка
4.	Cursor	Определяет вид курсора при наведении на компоненту
5.	Font	Определяет стиль и размер шрифта текста
6.	ForeColor	Цвет шрифта
7.	Image	Определяет рисунок
8.	LinkColor	Определяет цвет гиперссылки в тексте
9.	Text	Содержит текст
10.	TextAling	Выравнивание текста
11.	Enabled	Доступность компоненты (true/false)
12.	+ **LinkArea**	Определяет часть текста на который будет гиперссылка Start 5 Length 6 // 6 символов начиная с 5 ТОЛЬКО для одной гиперссылки
13.	**Links**	Содержит части гиперссылок в виде LinkArea
14.	**LinkData**	Содержит URL – адреса для гиперссылок
15.	**LinksVisited**	Если true, то автоматический доступ к ресурсу по щелчке по гиперссылке
16.	Visible	Видимость компоненты (true/false) label1. Visible =true;

TextBox – позволяет пользователю вводить и редактировать текст в одну или несколько строк.





Свойства:

	Наименование свойства	Описание

1.	BackColor	Цвет фона: textBox1.BackColor = Color.Blue;
2.	BorderStyle	Рамка
3.	Cursor	Определяет вид курсора при наведении на компоненту
4.	Font	Определяет стиль и размер шрифта текста Name- стиль шрифта Size – размер в пикселях Italic- наклонный шрифт Underline - подчеркнутый
5.	ForeColor	Цвет шрифта textBox1.ForeColor = Color.Red;
6.	Focus	Передает компоненте фокус ввода textBox1.Focus = true;
7.	**Lines**	Содержит текст в виде массива строк, нумерация с нуля textBox1.Lines[5]="Привет";
8.	ScrollBars	Полосы прокрутки

9.	Text	Содержит текст
		string s=textBox1.Text; // ввод данных
		textBox1.Text = “строка”+s; // вывод данных

10.	**CharacterCasing**	Определяет вид ввода данных: прописными(Upper), строчными(Lower), нормальными (Normal)
11.	Enabled	Доступность компоненты (серая) (true/false) label1.Enabled=true;
12.	Visible	Видимость компоненты (true/false) label1. Visible =true;
13.	**MaxLength**	Максимальная длина ввода и вывода данных в строке
14.	**Multiline**	Если true, то разрешена работа с многострочным редактором (по умолчанию false)
15.	**PasswordChar**	Определяет символ при вводе пароля
16.	**ReadOnly**	Если true, то запрещено редактирование
17.	WordWrap	Если true, то слова переносятся автоматически в начало следующей строки

Методы

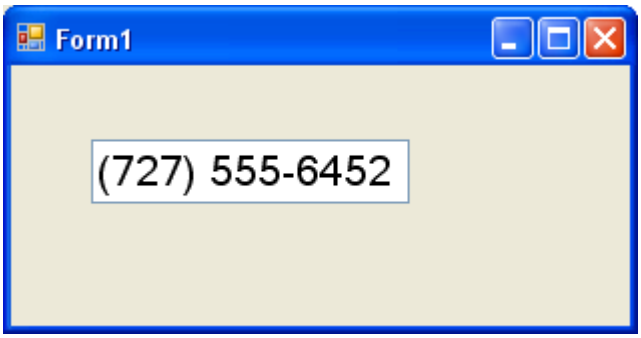
	Наименование метода	Описание
1.	AppendText	Добавляет строку в конец текста textBox1.AppendText("новый текст");
2.	Clear	Очистка содержимого компоненты: textBox1.Clear;
3.	ClearUndo	Удаляет из буфера сведения при последней операции
4.	Copy	Копирование части текста
5.	Cut	Вырезание части текста
6.	Hide	Скрывает элемент управления
7.	Paste	Вставка из буфера обмена
8.	Select	Выбирает диапазон текста textBox1.Select(5,7); // 7 символов начиная с 5

9.	SelectAll	Выбирает весь текст
10.	Show	Показывает элемент управления
11.	Undo	Отменяет последнюю операцию

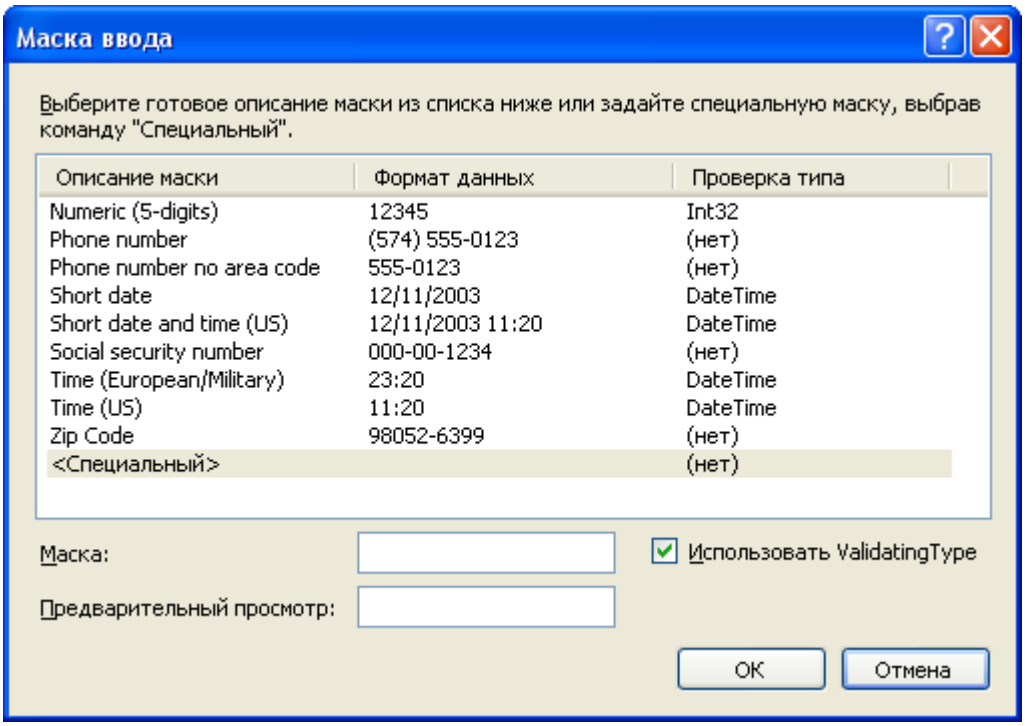
События:

1) **Действия -** Click, DoubleClick, MouseClick, MouseDoubleClick;
2) **Ключ** – KeyDown, KeyPress, KeyUp
3) **Мышь –** Mouse +(Down,Enter,Move,Up)

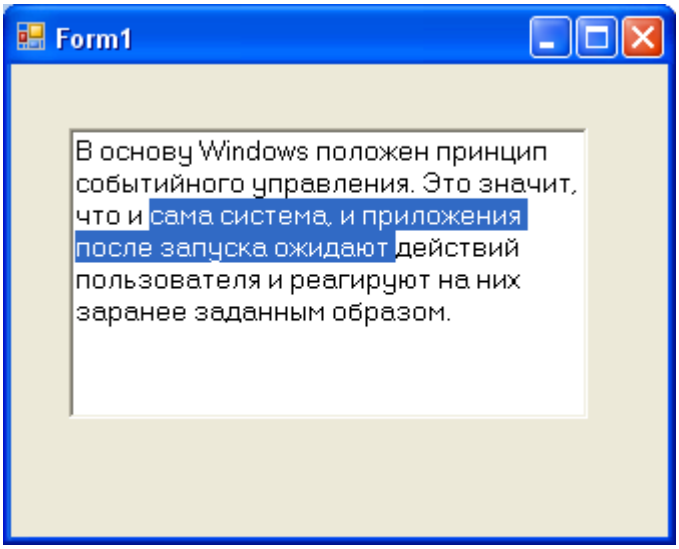
maskedTextBox – осуществляет ввод и редактирование текста по маске.



Схожа с компонентой **textBox,** но имеет следующие отличие:
Свойство Mask – определяет маску ввода



richTextBox – осуществляет ввод/вывод данных более одной строки.



Свойства:

	Наименование свойства	Описание
1.	BackColor	Цвет фона
2.	BorderStyle	Рамка
3.	Cursor	Определяет вид курсора при наведении на компоненту
4.	Font	Определяет стиль и размер шрифта текста
5.	ForeColor	Цвет шрифта
6.	Focus	Передает компоненте фокус ввода
7.	Lines	Содержит текст в виде массива строк, нумерация с нуля
8.	ScrollBars	Полосы прокрутки
9.	Text	Содержит текст
10.	Enabled	Доступность компоненты (серая) (true/false) label1.Enabled=true;
11.	Visible	Видимость компоненты (true/false) label1. Visible =true;
12.	MaxLength	Максимальная длина ввода и вывода данных в строке (2 147 483 647 символов)
13.	**Multiline**	Если true, то разрешена работа с многострочным редактором (по умолчанию стоит true)
14.	ReadOnly	Если true, то запрещено редактирование
15.	WordWrap	Если true, то слова переносятся автоматически в начало следующей строки

Методы

	Наименование метода	Описание
1.	AppendText	Добавляет строку в конец текста
2.	Clear	Очистка содержимого компоненты
3.	ClearUndo	Удаляет из буфера сведения при последней операции
4.	Copy	Копирование части текста
5.	Cut	Вырезание части текста
6.	Hide	Скрывает элемент управления
7.	Paste	Вставка из буфера обмена

8.	Select	Выбирает диапазон текста
9.	SelectAll	Выбирает весь текст
10.	Show	Показывает элемент управления
11.	Undo	Отменяет последнюю операцию

События:

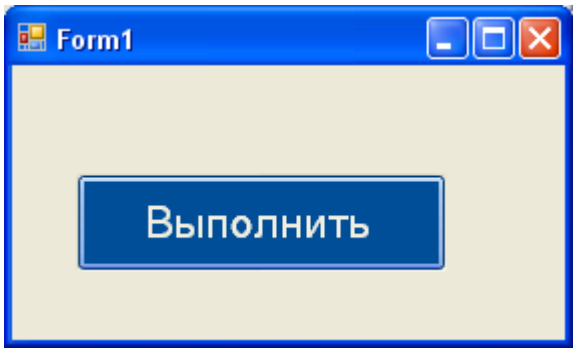
4) Действия - Click, DoubleClick, MouseClick, MouseDoubleClick;

5) Ключ – KeyDown, KeyPress, KeyUp

6) Мышь – Mouse +(Down,Enter,Move,Up)

Для создания простейшего приложения необходимо рассмотреть 2 компоненты: button и form.

button- командная кнопка с рисунком.



Свойства:

	Наименование свойства	Описание
1.	BackColor	Цвет фона: button1.BackColor = Color.Blue;
2.	Cursor	Определяет вид курсора при наведении на компоненту
3.	Image	Рисунок на кнопке
4.	Font	Определяет стиль и размер шрифта надписи
5.	ForeColor	Цвет шрифта
6.	Focus	Передает компоненте фокус ввода button 1.Focus = true;
7.	Text	Содержит надпись
8.	Enabled	Доступность компоненты (серая) (true/false)
9.	Visible	Видимость компоненты (true/false)

События:

7) Действия - Click, MouseClick

8) Ключ – KeyDown, KeyPress, KeyUp

9) Мышь – Mouse +(Down,Enter,Move,Up)

form- это оконный ресурс формы – макет.

Свойства:

	Наименование свойства	Описание

1.	BackColor	Цвет фона
2.	Font	Определяет стиль и размер шрифта текста
3.	ForeColor	Цвет шрифта
4.	BackGroundImage	Фоновое изображение
5.	**Text**	Заголовок окна
6.	Enabled	Доступность компоненты (серая) (true/false)
7.	Visible	Видимость компоненты (true/false), только для дочерних форм
8.	**ControlBox**	Если true, то кнопки свернуть, развернуть и закрыть - активны.
9.	**Icon**	Определяет значок у формы
10.	**ShowIcon**	Если true, то значок виден
11.	**Size.Width**	Ширина формы
12.	**Size.Height**	Высота формы
13.	**Location.X**	Расстояние от верхней границы формы до верхней границы экрана
14.	**Location.Y**	Расстояние от левой границы формы до левой границы экрана
15.	**MaximazeBox**	Доступность кнопки «Развернуть»
16.	**MinimazeBox**	Доступность кнопки «Свернуть»

Методы

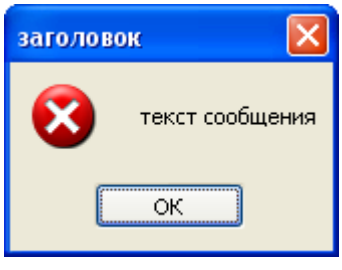
	Наименование метода	Описание
1.	Hide	Скрывает элемент управления
2.	Show	Показывает элемент управления
3.	Close	Закрытие формы
4.	Activate	Активизирует форму и помещает в нее фокус ввода
5.	CenterToScreen	Помещает форму в центр экрана

События: Click, MouseClick, Load, Activated, Closed и другие.

Для вывода сообщений используется диалоговое окно MessageBox. Формат команды:

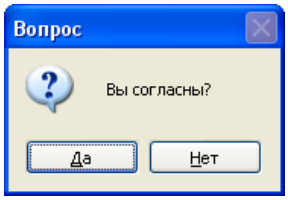
MessageBox.Show(сообщение, заголовок, кнопки, вид окна);

Пример 1: MessageBox.Show(" текст сообщения", "заголовок", MessageBoxButtons.OK, MessageBoxIcon.Error);

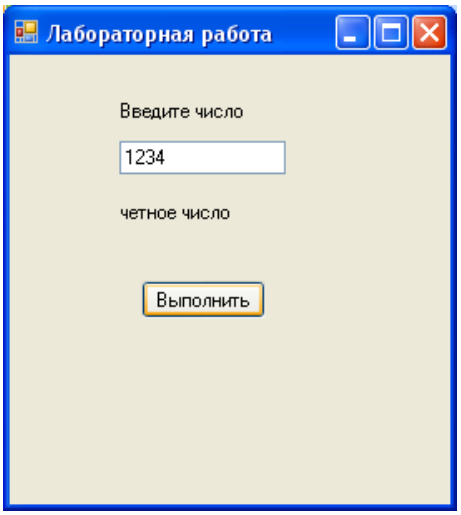


Пример 2: MessageBox.Show(" Вы согласны?","Вопрос",

MessageBoxButtons.YesNo, MessageBoxIcon.Question);

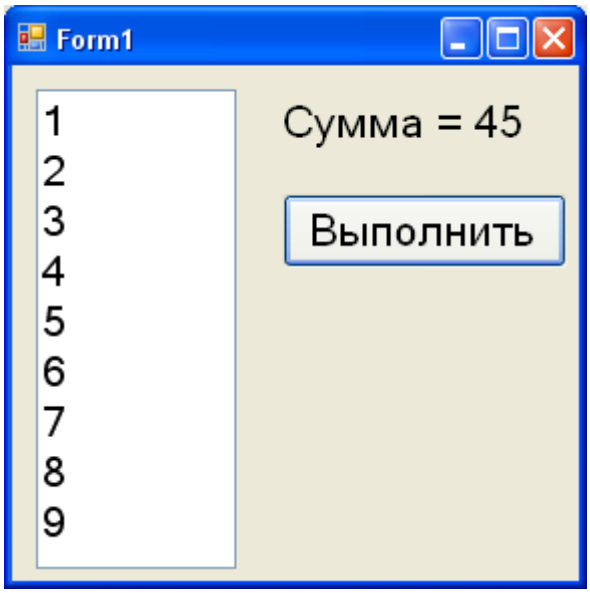


Пример 1: Дано число, проверить четное ли оно.




```
{
    int k = Convert.ToInt32(textBox1.Text);
    if (k%2==0)
        label2.Text= "четное число";
    else
        label2.Text = "НЕчетное число";
}
```

Пример 2: Найти сумму N целых чисел (компонента TextBox).



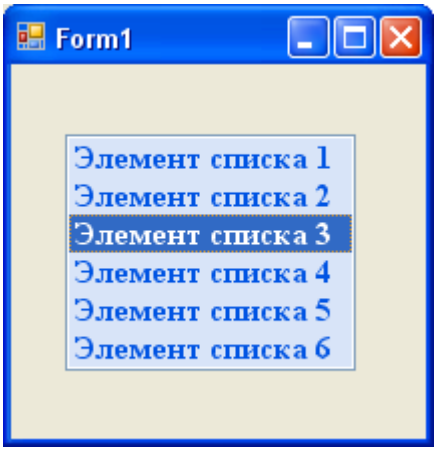

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
       {
            double Sum = 0.0; string c;
            int N = textBox1.Lines.Count(); byte b;
            for (int i = 0; i <= N-1; ++i)
            {  c = textBox1.Lines[i];
                b = Convert.ToByte(c);
                Sum += b;
             }
            label2.Text = "Сумма = " + Sum;
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Multiline = true; // многострочный редактор
        }
    }
}
}
```

Тема 3.3.2 Работа со списками

Для работы со списками существуют следующие компоненты: listBox и comboBox.

listBox – предназначена на создания и просмотра списка выбора.



Свойства:

	Наименование свойства	Описание
1.	BackColor	Цвет фона: listtBox1.BackColor = Color.Green;
2.	BorderStyle	Рамка
3.	Cursor	Определяет вид курсора при наведении на компоненту

4.	Font	Определяет стиль и размер шрифта текста
5.	ForeColor	Цвет шрифта listBox1.ForeColor = Color.Red;
6.	Focus	Передает компоненте фокус ввода listBox1.Focus = true;
7.	**ColumnWidth**	Ширина столбцов
8.	Count	Определяет количество строк в списке int k = listBox1.Items.Count;
9.	Enabled	Доступность компоненты (серая) (true/false)
10.	**HorizontalScrollbar**	Горизонтальные полосы прокрутки
11.	**MultiColumn**	Возможность вывода в несколько столбцов, то есть если количество строк больше компоненты, то будет 2 столбец и так далее (true/false)
12.	**Items**	Массив строк списка, нумерация с 0
13.	**Sorted**	Если true, то сортировка списка
14.	Visible	Видимость компоненты (true/false)
15.	SelectionMode	Если One, то только 1 элемент для выбора если MultiSimple, то несколько
16.	**SelectedText**	Возвращает выбранный элемент string k = listBox1.SelectedText;
17.	**SelectedIndex**	Содержит номер выбранного элемента int k = listBox1.SelectedIndex;

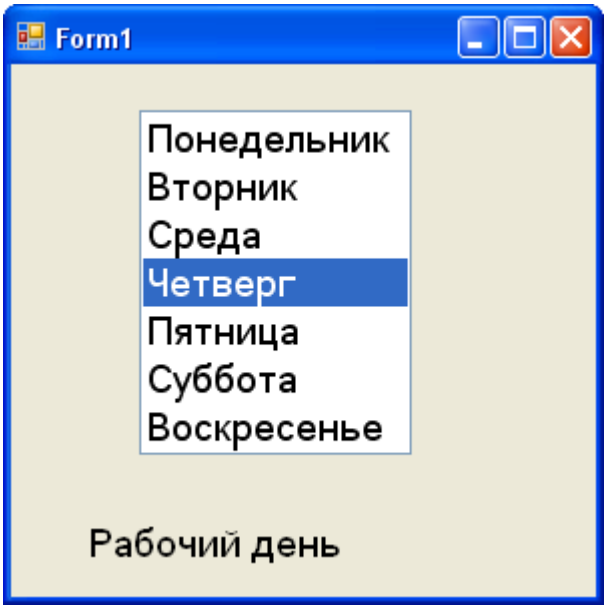
Методы

	Наименование метода	Описание
1.	Add	Добавляет новый элемент в список listBox1.Items.Add("новая строка");
2.	Clear	Очистка содержимого компоненты: listBox1.Clear;
3.	CopyTo	Копирование части списка
4.	Hide	Скрывает элемент управления
5.	Show	Показывает элемент управления
6.	Insert	Вставка listBox1.Items.Insert(5, "новый элемент");
7.	RemoveAt	Удаление N элемента списка listBox1.Items.RemoveAt(3);

События:

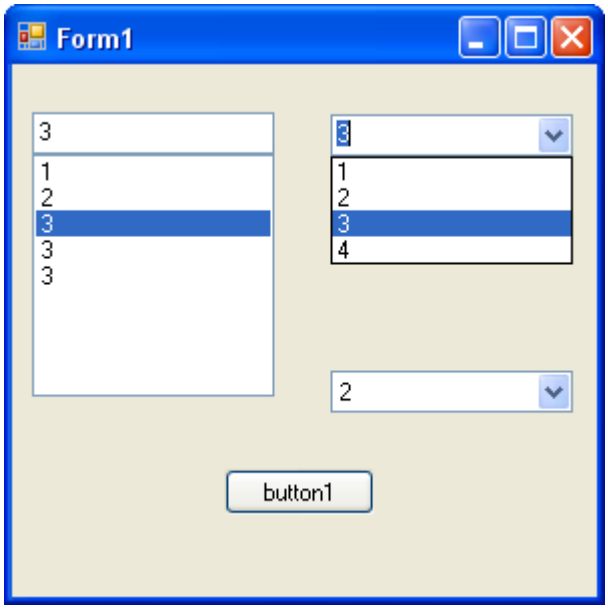
10) **Действия -** Click, DoubleClick, MouseClick, MouseDoubleClick;
11) **Ключ** – KeyDown, KeyPress, KeyUp
12) **Мышь –** Mouse +(Down,Enter,Move,Up)

Пример: по выбранному дню недели вывести выходной или рабочий день.




```
private void listBox1_Click(object sender, EventArgs e)
    {
       int a = listBox1.SelectedIndex;
       int x = 0;
       if ((a >= 0) && (a <= 4)) x = 1;
       if ((a >= 5) && (a <= 6)) x = 2;
       switch (x)
       {
          case 1: label1.Text ="Рабочий день"; break;
          case 2: label1.Text ="Выходной  день"; break;
          default: label1.Text ="Не верный ввод данных"; break;
       }
    }
private void Form1_Load(object sender, EventArgs e)
    {
       listBox1.Items.Add("Понедельник");
       listBox1.Items.Add("Вторник");
       listBox1.Items.Add("Среда");
       listBox1.Items.Add("Четверг");
       listBox1.Items.Add("Пятница");
       listBox1.Items.Add("Суббота");
       listBox1.Items.Add("Воскресенье");
    }
```

comboBox – представляет собой список с окном редактирования (комбинированный).



Свойства:

	Наименование свойства	Описание
1.	BackColor	Цвет фона
2.	Cursor	Определяет вид курсора при наведении на компоненту
3.	Font	Определяет стиль и размер шрифта текста
4.	ForeColor	Цвет шрифта listBox1.ForeColor = Color.Red;
5.	**Text**	Содержит текст
6.	Focus	Передает компоненте фокус ввода listBox1.Focus = true;
7.	**Sorted**	Если true, то сортировка списка
8.	Enabled	Доступность компоненты (серая) (true/false)
9.	MaxLength	Максимальная длина строк списка
10.	Visible	Видимость компоненты (true/false)
11.	**DropDownStyle**	Определяет стиль окна списка: Simple – без окна ввода DropDown – с окном ввода, может получать фокус ввода DropDownList – с окном ввода, ввод собственного варианта не возможен
12.	**Items**	Массив строк списка, нумерация с 0
13.	**Count**	Определяет количество строк в списке int k = comboBox1.Items.Count;
14.	**SelectedText**	Возвращает выбранный элемент string k = comboBox1.SelectedText;
15.	**SelectedIndex**	Содержит номер выбранного элемента int k = comboBox1.SelectedIndex;

Методы

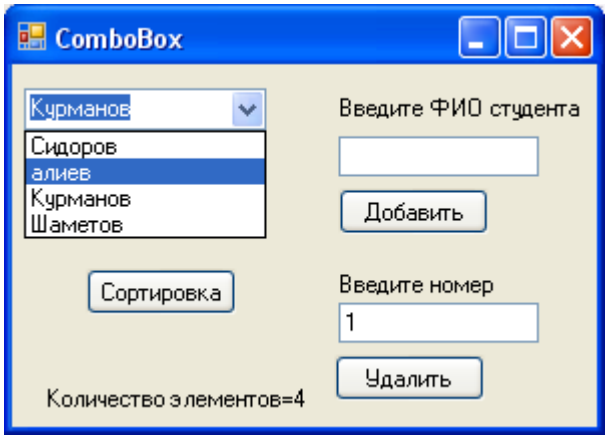
	Наименование метода	Описание
1.	Add	Добавляет новый элемент в список
2.	Clear	Очистка содержимого компоненты

3.	CopyTo	Копирование части текста
4.	Hide	Скрывает элемент управления
5.	Show	Показывает элемент управления
6.	Insert	Вставка
7.	RemoveAt	Удаление N элемента списка

События:

13) **Действия -** Click, DoubleClick, MouseClick, MouseDoubleClick;
14) **Ключ** – KeyDown, KeyPress, KeyUp
15) **Мышь –** Mouse +(Down,Enter,Move,Up)

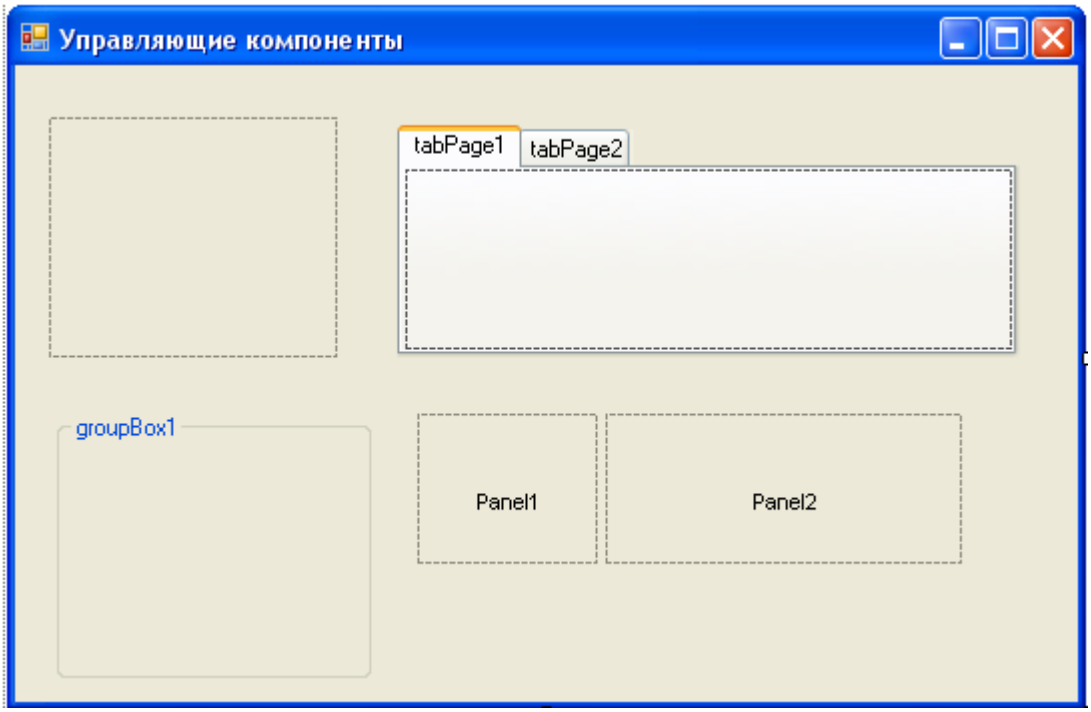
Пример: Дан список студентов, осуществить возможность добавления нового студента с помощью диалогового окна, удаление и редактирование N студента.



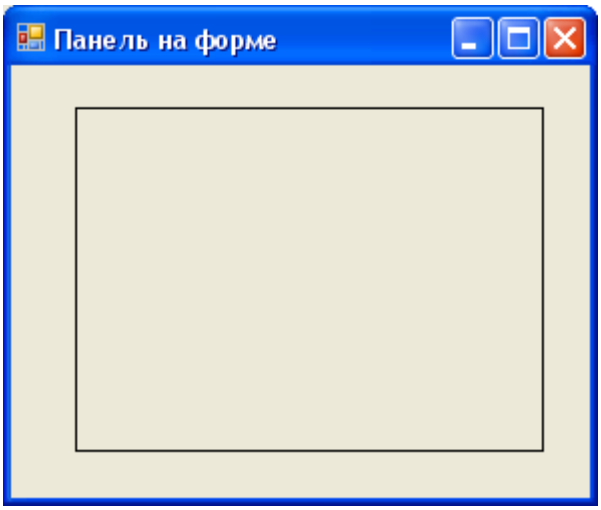

```
private void button3_Click(object sender, EventArgs e)
      {
        comboBox1.Sorted = true;                    // сортировка
      }
private void button1_Click(object sender, EventArgs e)
      {
        comboBox1.Items.Add(textBox1.Text);  // добавление
      }
private void button2_Click(object sender, EventArgs e)
      {
        int k = Convert.ToInt32(textBox2.Text);
        comboBox1.Items.RemoveAt(k-1);                // удаление
        label3.Text = "Количество элементов="+comboBox1.Items.Count;
       }
private void comboBox1_Click(object sender, EventArgs e)
      {
        comboBox1.Text = comboBox1.SelectedText;     // вывод  выбранного элем
                                                     // в строку
       }
```

Тема 3.4 Компоненты управления

Для управления проектом используются следующие компоненты, размещенные в группе «контейнеры»: Panel, GroupBox, TabControl, FlowLayoutPanel, TableLayoutPanel, SplitContainer.



Panel – представляет собой окно, которое визуально отделяет часть формы (контейнер).



Свойства:

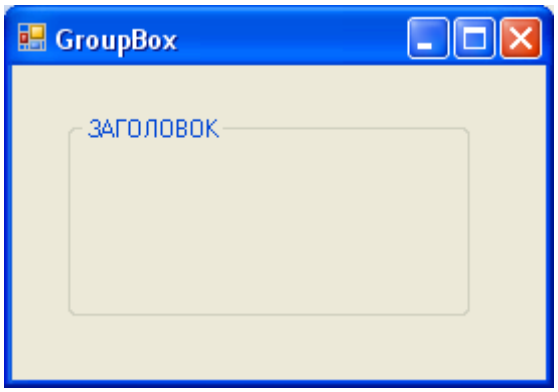
	Наименование свойства	Описание
1.	BorderStyle	Стиль рамки
2.	Enabled	Доступность компоненты (серая) (true/false)
3.	Visible	Видимость компоненты (true/false)

Методы

	Наименование метода	Описание

1.	Hide	Скрывает элемент управления
2.	Show	Показывает элемент управления

GroupBox – представляет собой контейнер для других компонент с заголовком.



Свойства:

	Наименование свойства	Описание
1.	Text	Заголовок контейнера
2.	BorderStyle	Стиль рамки
3.	Enabled	Доступность компоненты (серая) (true/false)
4.	Visible	Видимость компоненты (true/false)

Методы

	Наименование метода	Описание
1.	Hide	Скрывает элемент управления
2.	Show	Показывает элемент управления

tabControl – содержит контейнеры в виде вкладок (страниц).

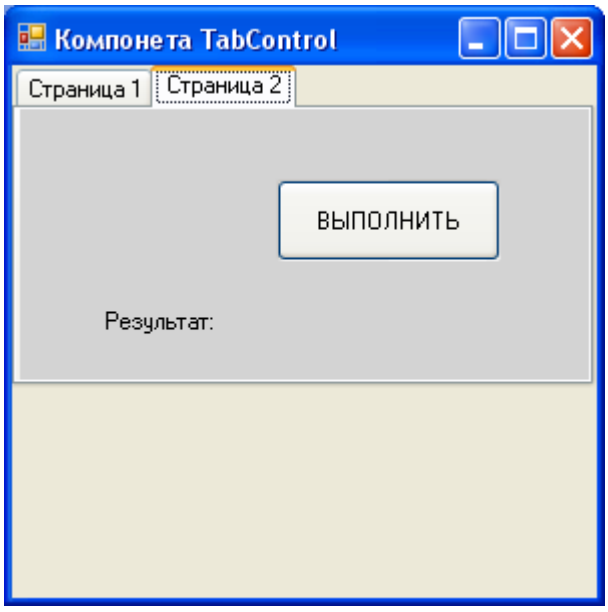
Свойства:

	Наименование	Описание

	свойства	
1.	Aligment	Определяет местоположение вкладок: снизу(bottom), сверху(top), слева(left) и справа(right)
2.	Apperance	Стиль вкладки: кнопкой, не выпуклой кнопкой или простой
3.	Dock	Местоположение компоненты на форме: сверху, снизу, по центру, слева или справа
4.	TabPages	Список заголовков страниц: Name – имя страницы (по умолчанию tabPage1) Font – стиль и размер шрифта BackColor- цвет фона ForeColor – цвет шрифта BorderStyle - Стиль рамки

Методы

	Наименование метода	Описание
1.	Hide	Скрывает элемент управления
2.	Show	Показывает элемент управления

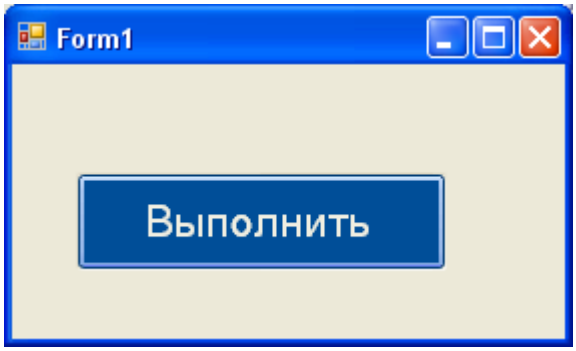


FlowLayoutPanel, TableLayoutPanel – вспомогательные панели в виде таблиц.

SplitContainer – разделяет область компоненты на 2 панели

Тема 3.4.1 Работа с кнопками

button- командная кнопка с рисунком.



Свойства:

	Наименование свойства	Описание
1.	AutoEllipsis	Задает вывод «…» если надпись выходит за пределы компоненты
2.	AutoSize	Если true, то компонента увеличится по размеру надписи
3.	BackColor	Цвет фона: button1.BackColor = Color.Blue;
4.	BackGroudImage	Определяет рисунок фона на кнопке
5.	BackGroudLayout	
6.		
7.	Cursor	Определяет вид курсора при наведении на компоненту
8.	Image	Рисунок на кнопке
9.	Font	Определяет стиль и размер шрифта надписи
10.	ForeColor	Цвет шрифта
11.	Focus	Передает компоненте фокус ввода button 1.Focus = true;
12.	Text	Содержит надпись
13.	Enabled	Доступность компоненты (серая) (true/false)
14.	Visible	Видимость компоненты (true/false)

События:

1) **Действия -** Click, MouseClick
2) **Ключ** – KeyDown, KeyPress, KeyUp
3) **Мышь –** Mouse +(Down,Enter,Move,Up)

Тема 3.4.2 Работа с переключателями и панелями

Для работы с переключателями используются компоненты: зависимый (radioButton) и независимый(checkBox).

В отличие от checkBox компоненты radioButton представляют собой зависимые переключатели, предназначенные для выбора одного из нескольких взаимоисключающих решений. На форму (точнее, в компонент-контейнер) помещается по меньшей мере два таких компонента. Они могут иметь только два состояния, определяемых свойством Checked. Если в одном компоненте это свойство принимает значение True, во всех других компонентах, расположенных в том же контейнере, свойства Checked принимают значения False.

Помимо свойства checked компонент TRadioButton имеет еще одно специфичное свойство - Alignment, аналогичное такому же свойству TCheckBox. Как и в TCheckBox, программист не может изменять размеры и цвет круглого окошка компонента.

radioButton – зависимый переключатель.

Свойства:

	Наименование свойства	Описание
1.	Name	
2.	**Text**	Надпись рядом с кнопкой
3.	AutoEllipsis	Если текст больше компоненты, то отображается кнопка с 3 точками
4.	AutoSize	Разрешение или запрет изменения размера компоненты
5.	WordWrap	Перенос по словам
6.	BackColor	Цвет фона
7.	BackgroundImage	Фоновый рисунок
8.	CanFocus	Получает значение, может ли он получить фокус
9.	CanSelect	Получает значение, доступен ли элемент
10.	**Checked**	Если true, то элемент выбран
11.	ContextMenu	Определяет контекстное меню, которое связано с элементом
12.	Cursor	
13.	Focused	
14.	Font	
15.	ForeColor	
16.	Height	
17.	Enabled	Доступность компоненты (серая) (true/false)
18.	Visible	Видимость компоненты (true/false)

Методы

	Наименование метода	Описание
1.	**Hide**	Скрывает элемент управления
2.	**Show**	Показывает элемент управления

3.	TextImageRelation	
4.	**ToString**	Преобразование в строку
5.	**ResetText**	Сбрасывает значение свойства Text – по умолчанию
6.	**FindForm**	Получает форму на которой находится элемент
7.	Focus	
8.	Select	Активирует элемент управления

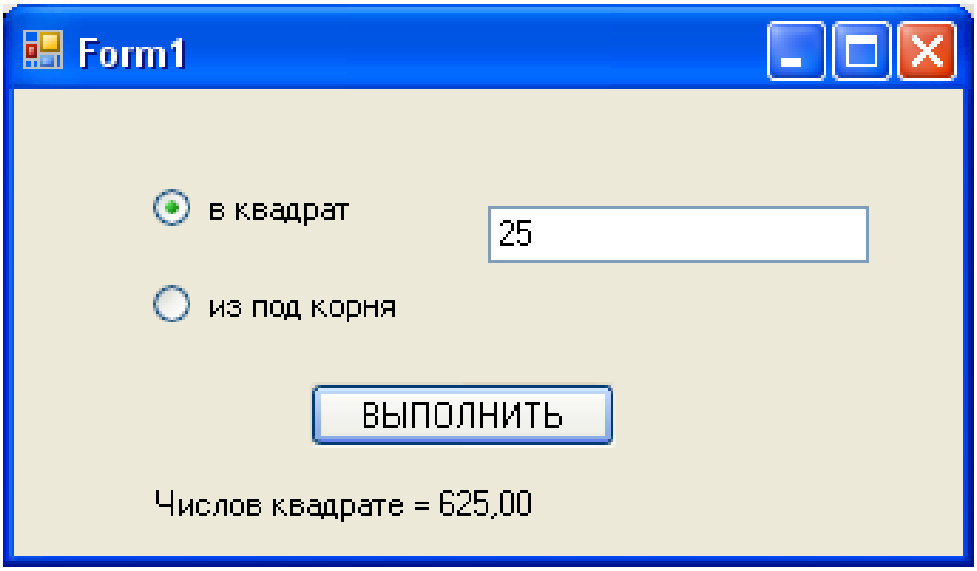
События:

1) Действия - Click, MouseClick

2) Ключ – KeyDown, KeyPress, KeyUp

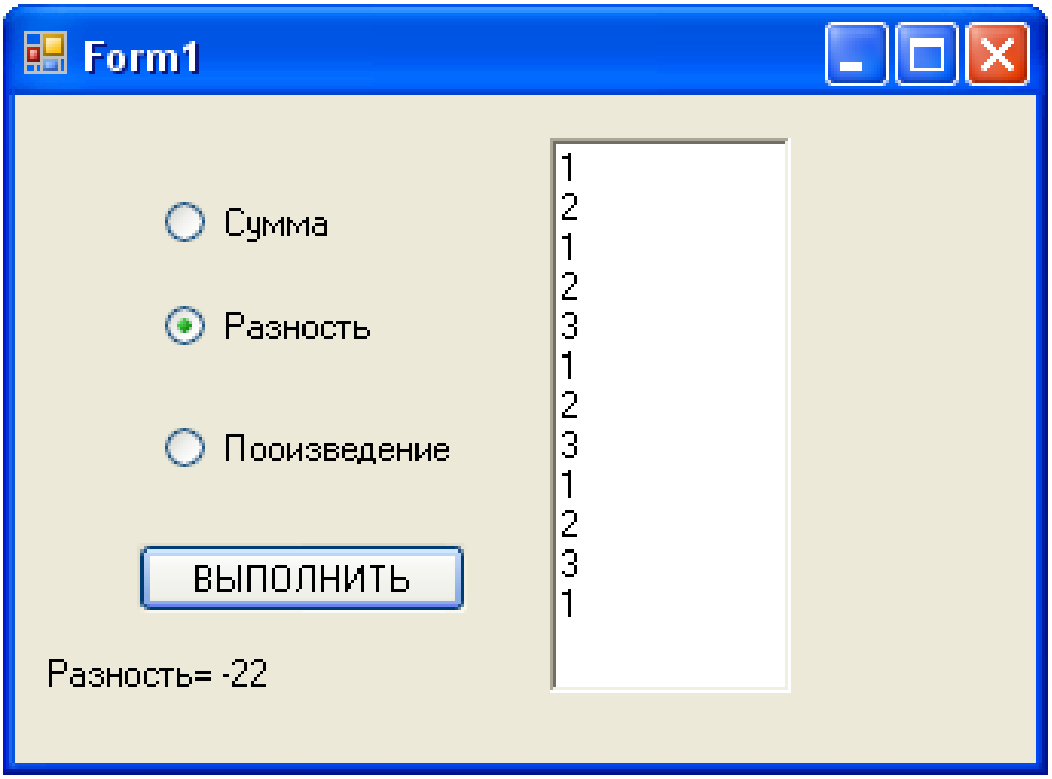
3) Мышь – Mouse +(Down,Enter,Move,Up)

Пример 1: Возвести в квадрат или извлечь из под корня вещественное число.



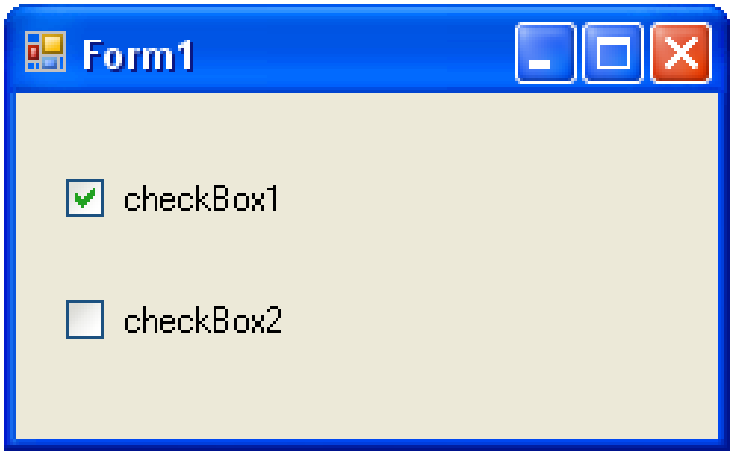

```
private void button1_Click(object sender, EventArgs e)
{

    double b = Convert.ToDouble(textBox1.Text);

  if (radioButton1.Checked)
     label1.Text = "Числов квадрате = " + Math.Pow(b,2).ToString("N");
  if (radioButton2.Checked==true)
     label1.Text = «Число извлеченное из корня= « + Math.Sqrt(b).ToString(«N»);
 }
```

Пример 2: Найти сумму, разность или произведение
чисел одномерного массива.




```
    private void Form1_Load(object sender, EventArgs e)
    {
       radioButton1.Checked=true;  // выбран 1 элемент
    }

    private void button1_Click(object sender, EventArgs e)
    {
       double Sum = 0.0, Raz = 0.0, Proiz = 1.0;
       int N = richTextBox1.Lines.Count();
       int b;
       for (int i = 0; i <= N - 1; ++i)
       {
         b = Convert.ToInt32(richTextBox1.Lines[i]);
          Sum += b;       Raz -= b;        Proiz *= b;
       }
       if (radioButton1.Checked)
                    label1.Text = "Сумма = " + Sum;
       if (radioButton2.Checked)
                     label1.Text = "Разность= " + Raz;
       if (radioButton3.Checked)
                     label1.Text = "Произведение = " + Proiz;
    }
}
```

checkbox – независимый переключатель.



Свойства, такие же как и у зависимого переключателя:

	Наименование свойства	Описание
1.	Name	
2.	**Text**	Надпись

3.	CheckState	Задает состояние компоненты:
		Unchecked – не выбран
		Checked – выбран
		if (checkBox2.CheckState == CheckState.Checked)
		Indeterminate – заполненный квадрат
		checkBox2.CheckState = CheckState.Indeterminate;

4.	**Checked**	Если true, то элемент выбран

Методы

	Наименование метода	Описание
1.	Hide	Скрывает элемент управления
2.	Show	Показывает элемент управления
3.	ToString	
4.	Focus	

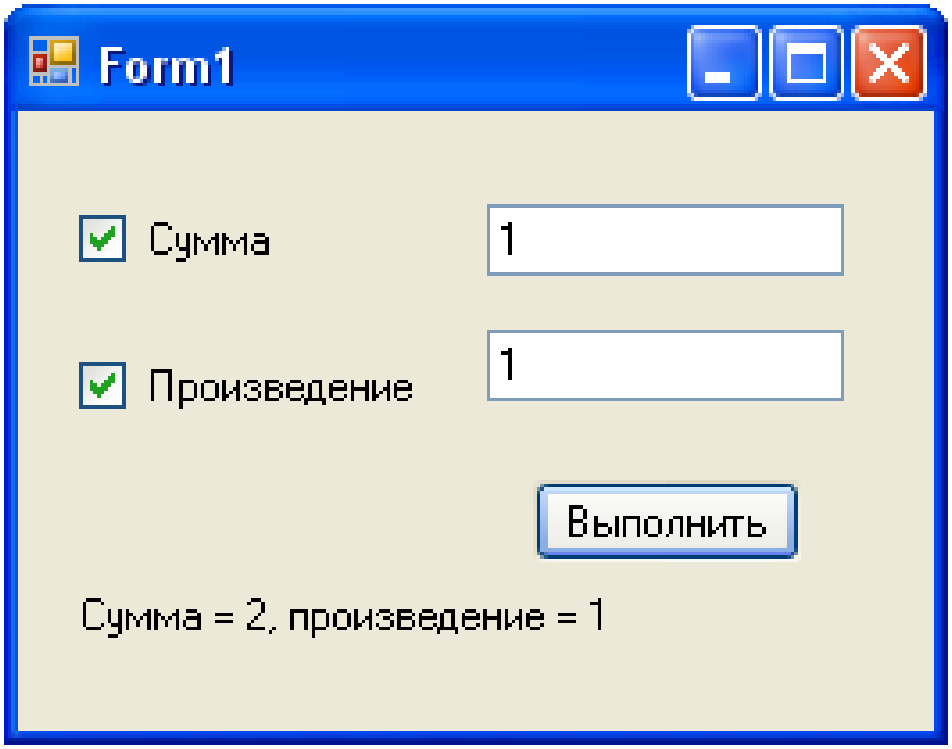
События:

4) Действия - Click, MouseClick

5) Ключ – KeyDown, KeyPress, KeyUp

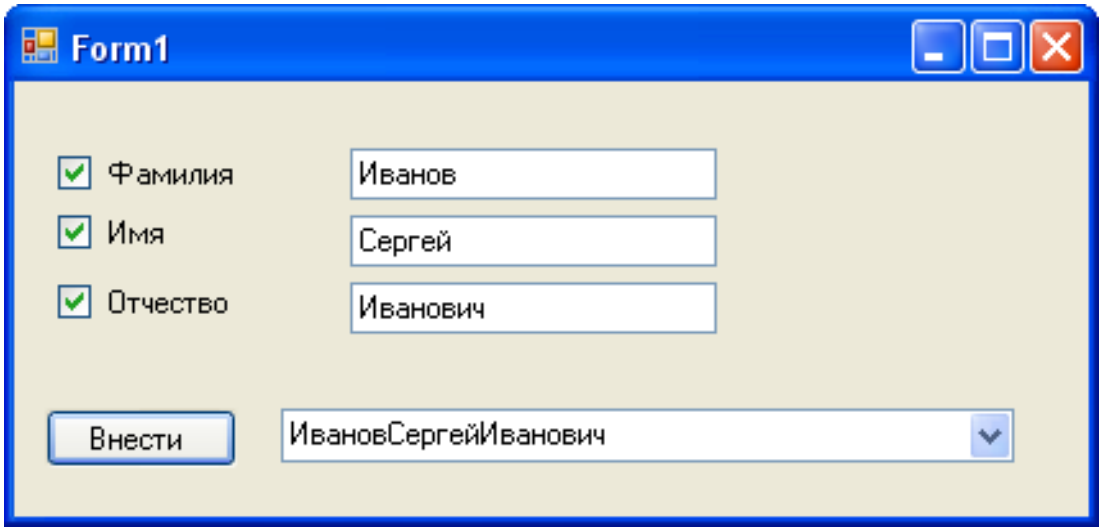
6) Мышь – Mouse +(Down,Enter,Move,Up)

Пример 1: Даны два числа, найти сумму или произведение чисел.



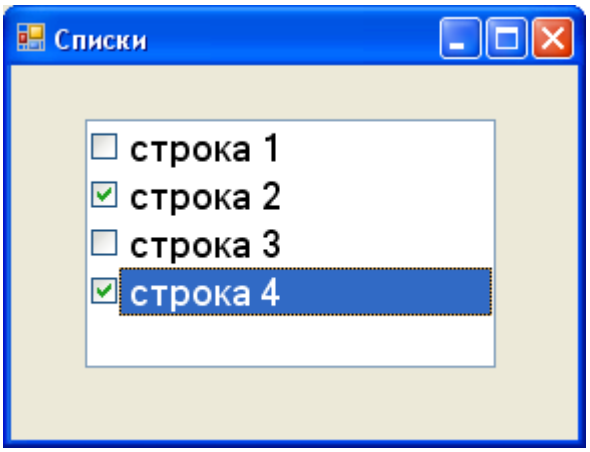

```
private void button1_Click_1(object sender, EventArgs e)
      {
        int a=Convert.ToInt32(textBox1.Text);
        int b=Convert.ToInt32(textBox2.Text);
        int sum=a+b; int p=a*b;
        if ((checkBox1.CheckState == CheckState.Checked) &&
                                    (checkBox2.CheckState == CheckState.Checked))
          label1.Text = "Сумма = " + sum + ", произведение = " + p;
        else
        {
          if (checkBox1.CheckState == CheckState.Checked)
            label1.Text = "Сумма = " + sum;
          if (checkBox2.CheckState == CheckState.Checked)
            label1.Text = "Произведение = " + p;
        }
 }
```

Пример 2: При выборе фамилии, имени или отчества вывести окна ввода, затем внести данные в список (comboBox).




```
private void button1_Click(object sender, EventArgs e)
  {
     comboBox1.Items.Add(textBox1.Text + textBox2.Text + textBox3.Text);
     comboBox1.Text = textBox1.Text + textBox2.Text + textBox3.Text;
  }

  private void checkBox1_Click(object sender, EventArgs e)
  {          textBox1.Visible=true;        }

  private void checkBox3_Click(object sender, EventArgs e)
  {textBox2.Visible=true;       }

  private void checkBox2_Click(object sender, EventArgs e)
  {textBox3.Visible=true;}
```

checkedListBox – компонента представляет собой список, перед каждым элементом которого находится переключатель checkbox.



Свойства:

	Наименование свойства	Описание
1.	Items	“Элементы списка - коллекция строк
2.	Items.Count	Количество элементов списка
3.	Sorted	Сортировка элементов списка

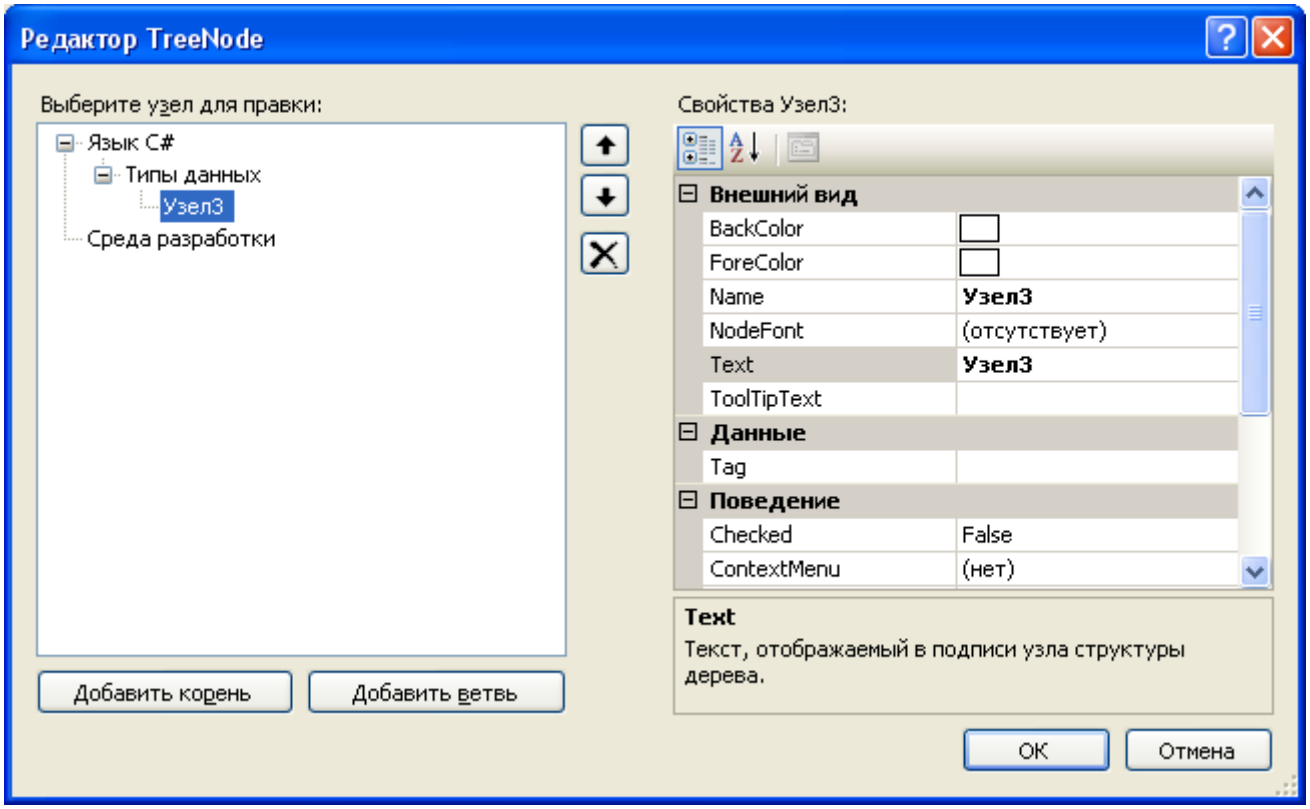
4.	**CheckOnClick**	Способ пометки элемента списка. Если значение свойства равно false, то первый щелчок выделяет элемент списка (строку), а второй устанавливает в выбранное состояние переключатель. Если значение true, то щелчок на элементе списка выделяет элемент и устанавливает состояние переключатель
5.	**CheckedItems**	Список элементов
6.	**CheckedItems.Count**	Количество выбранных элементов списка
7.	**CheckedIndices**	Список выбранных номеров
8.	**MultiColumn**	Если true, то возможен вывод в несколько колонок
9.	**SelectionMode**	Если MultiSimple, то множественный выбор Если One, то одиночный

Методы

	Наименование метода	Описание
1.	ClearSelected	Снимает выделение со всех позиций списка
2.	FindString	Поиск строки с определенной позиции checkedListBox1.FindString("по", 5);
3.	FindStringExact	Поиск точно указанной строки с определенного индекса
4.	Focus	Передает фокус ввода компоненте
5.	GetItemChecked	Возвращает значение true, если выбран N элемент if (checkedListBox1.GetItemChecked(4)) MessageBox.Show("выбран 5 элемент");
6.	GetItemText	Возвращает текстовое представление выбранного элемента
7.	GetSelected	Возвращает значение, определяющее, выбрана ли указанная позиция

treeView –предназначен для отображения иерархических структур: дерева наследования объектов или файловой структуры диска.

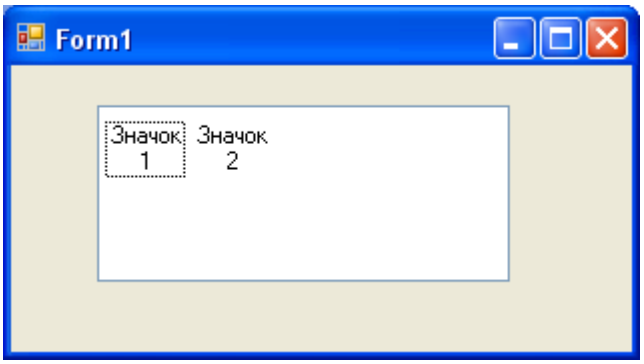
Свойство Nodes – мастер создания структуры.



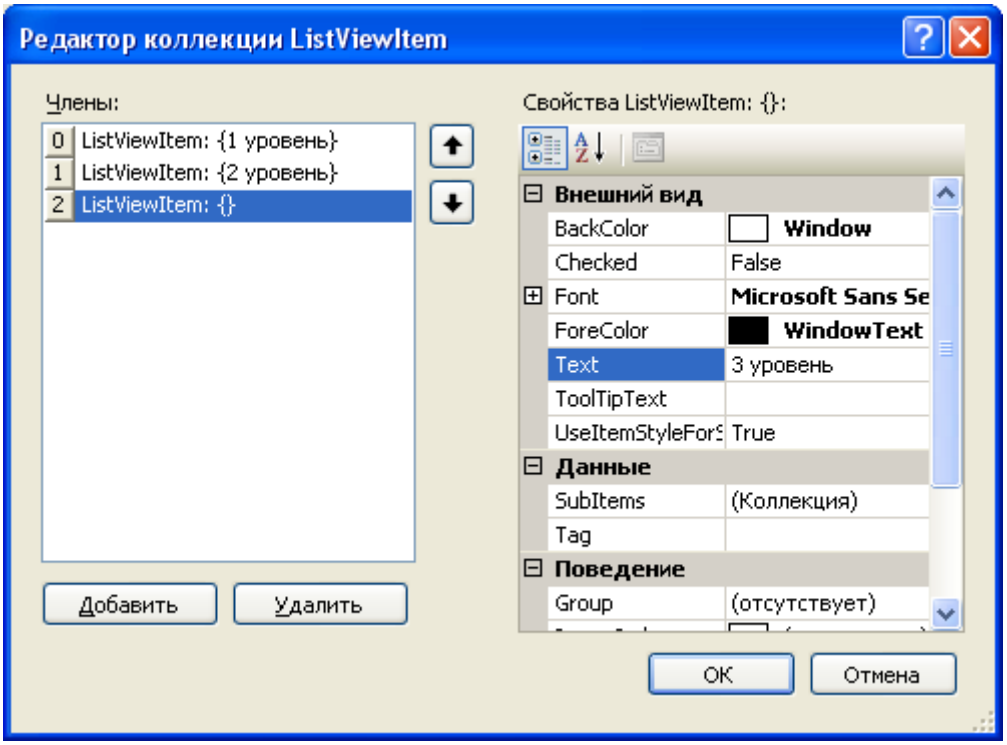
Добавить корень и добавить ветвь – позволяют создать структуру отображения данных.

Для программного наполнения списка используются методы Add, AddRange, Insert, Remove.

listView – список со значками

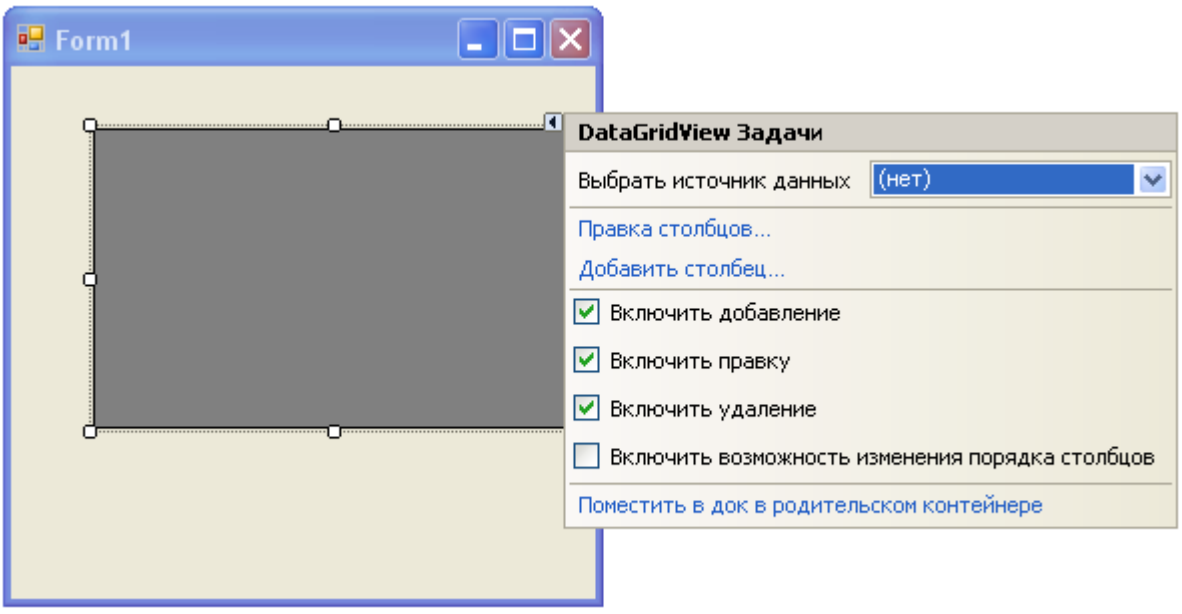


Свойство Items – позволяет создать список значков в виде редактора коллекции.

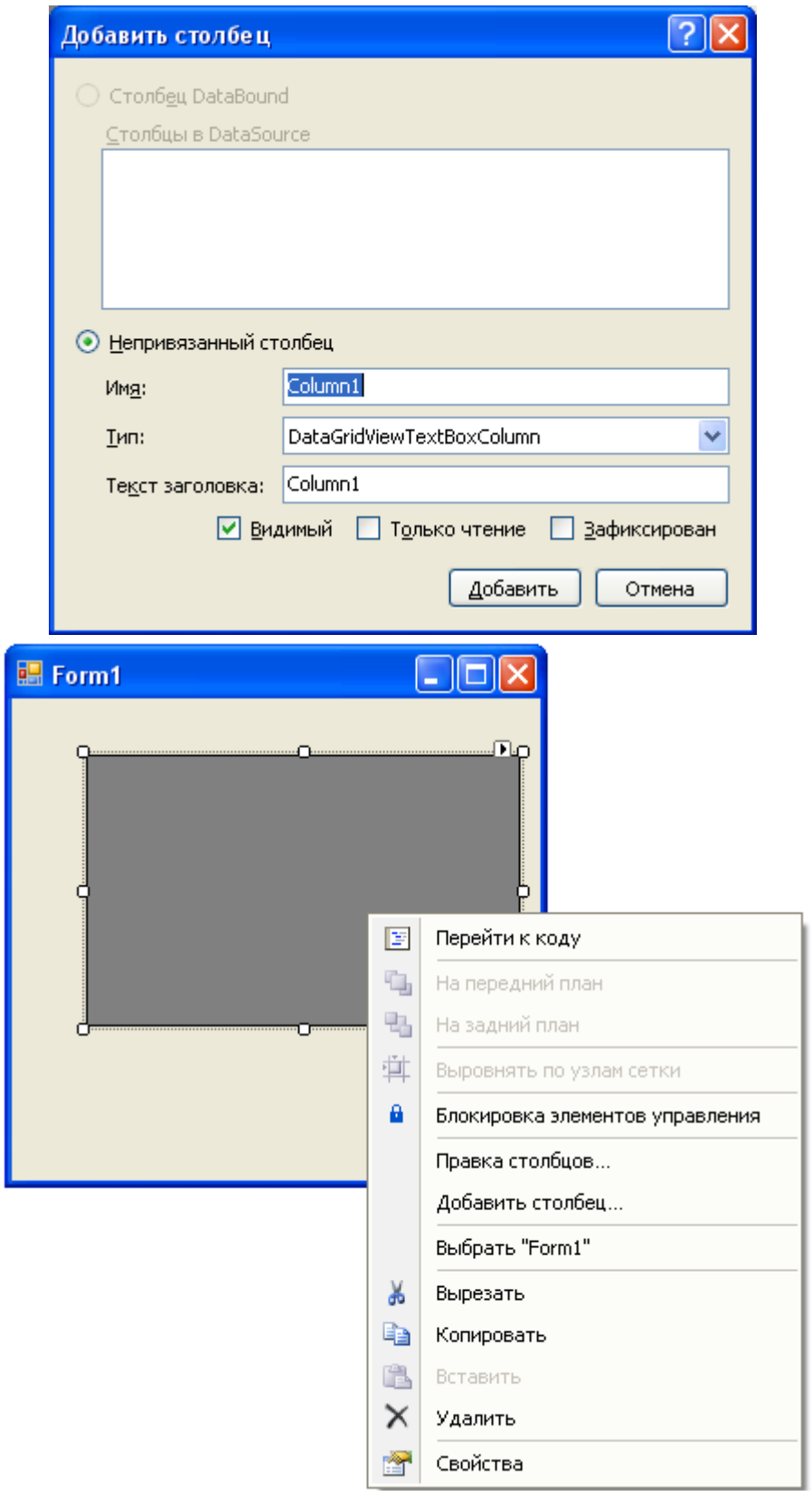


dataGridView – предназначен для работы с двумерным массивом и таблицами БД. Компонента расположена в разделе «Данные» панели компонет. Для того чтобы настроить таблицу: определить количество строк, столбцов, задать формат отображения данных, необходимо использовать вкладку «задачи» (черная стрелка справа от компоненты) или контекстное меню.

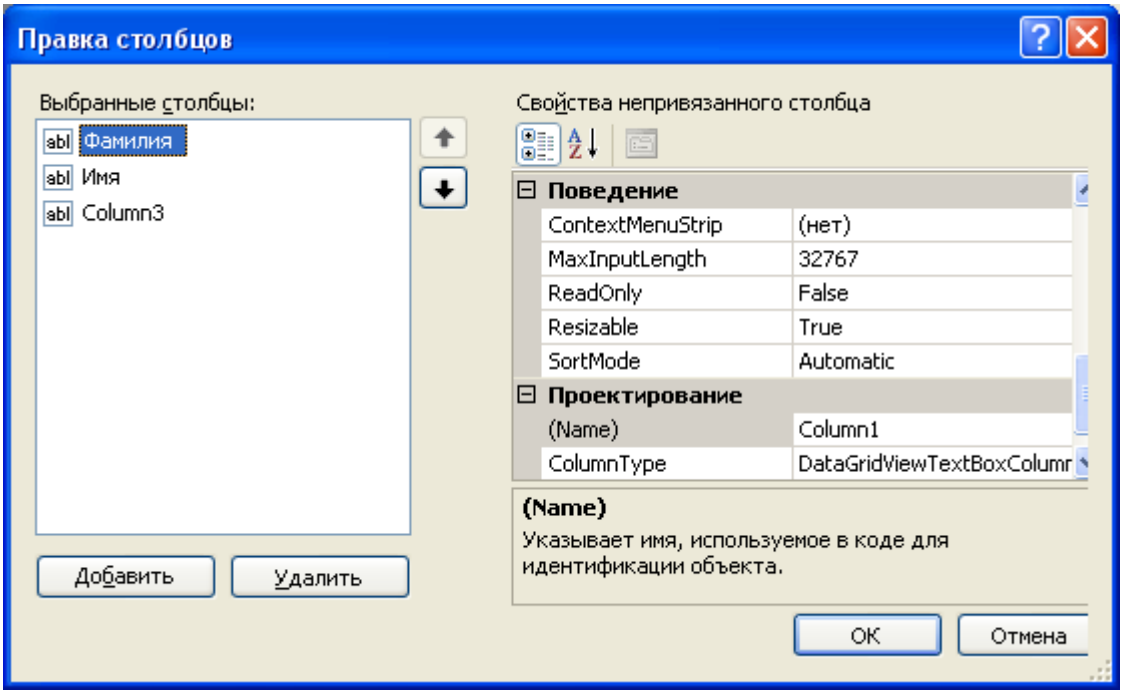
Вкладка «Задачи» позволяет добавлять и исправлять столбцы: задавать заголовки, определять формат текста.



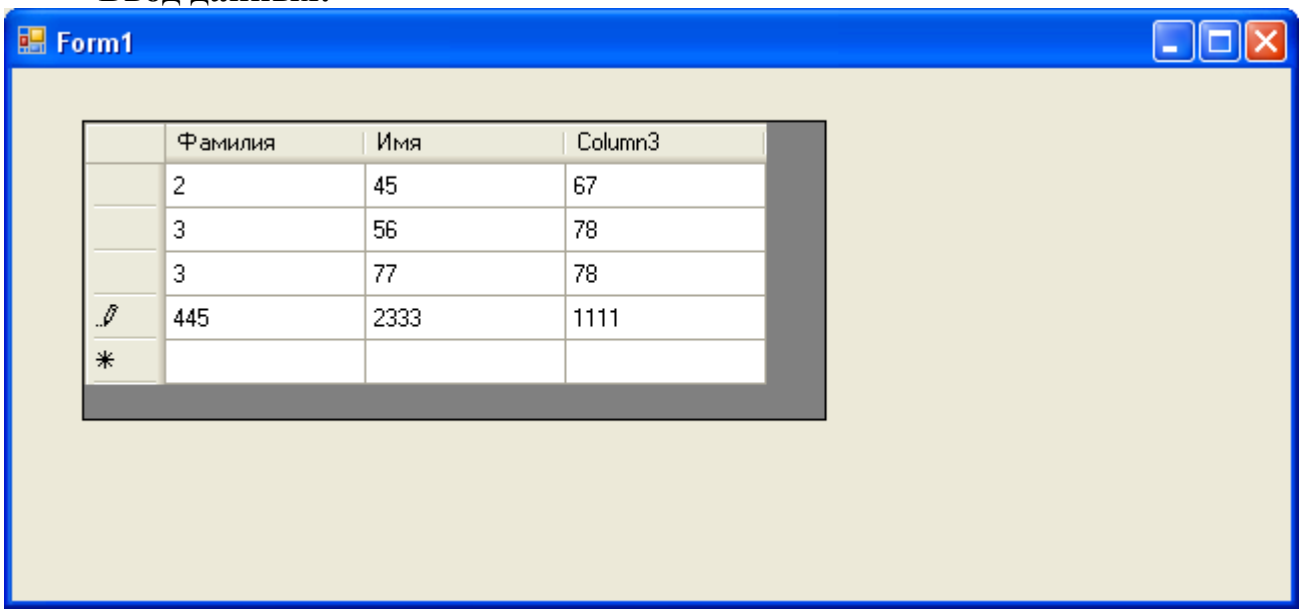
Добавить столбец:



Правка столбцов:



Ввод данных:



Свойства

Свойства:

	Наименование свойства	Описание
1.	Rows	Содержит строки таблицы, подсвойство Add Insert
2.	RowCount	Количество строк
3.	Columns	Содержит столбцы таблицы, подсвойство Add Insert
4.	ColumnCount	Количество столбцов
5.	dataGridView1[I, J]	Обращение к элементам массива I- столбец J – строка dataGridView1[1, 2].Value = "Привет"; //1 столбец 2 строка

		dataGridView1[5, 0].Value = Convert.ToString(d); //5 столбец 0 строка

Методы

	Наименование метода	Описание

1.	Add	Добавление строк или столбцов
		dataGridView1.Rows.Add(1); // 1 строку
		dataGridView1.Columns.Add(3); // 3 столбца

2.	Insert	Вставка строк или столбцов
		dataGridView1.Rows.Insert(5,2); // 2 строк и после 6
		dataGridView1.Columns. Insert (3,1); // 1 столбец после 4

3.	Delete	

Пример: Дан двумерный массив A(3,3). Найти количество положительных.

```
private void button1_Click(object sender, EventArgs e)
      {
        int[,] a = new int[3, 3]; int poloj = 0;
        for (int i = 0; i < 3; i++)
        {
           for (int j = 0; j < 3; j++)
           {
              a[i, j] = Convert.ToInt32(dataGridView1[i, j].Value);
              if (a[i, j] > 0) poloj++;
           }
        }
        label1.Text = "Количество положительных = " + poloj;  }

 private void Form1_Load(object sender, EventArgs e)
      {
        dataGridView1.ColumnCount = 3;
              dataGridView1.RowCount = 3;
        }
```

Пример2: Дан a(n,m) вещественных чисел. Найти max и min.

Тема 3.5 Формы

В данной теме рассматриваются основополагающие классы любого Windows-приложения — Form и Application. Класс Form служит контейнером для компонентов Windows Forms, а класс Application содержит свойства, методы и события, определяющие приложение в целом.

Класс Form

Конструктор формы Form() не имеет параметров. По умолчанию он создает форму размером 300 × 300 пикселов. Текст конструктора выглядит так:

```
public Form1()
{
   InitializeComponent();
}
```

Конструктор делится на две части: операторы первой части располагаются перед оператором вызова метода **InitializeComponent(),** операторы второй части — после.

Метод **InitializeComponent()** запускает дизайнер форм, который инициализирует все размещенные на форме компоненты — вызывает их конструкторы и устанавливает в свойства значения, определенные на этапе конструирования. Непосредственно перед вызовом **InitializeComponent()** программист может разместить любой код, который может повлиять на работу дизайнера форм. Например, он может проверить легальность программной копии и отменить вызов редактора форм, если копия нелегальна. В этом случае на экране появится пустая форма без компонентов и даже без заголовка. После завершения работы редактора форм (после вызова **InitializeComponent())** программист также может вставить код. В этом месте удобно создавать динамические экземпляры классов.

Свойства формы представлены некоторые свойства класса **Form**.

Таблица 19.1. Свойства класса Form

Свойство	**Назначение**
public IButtonControl AcceptButton{**get**; **set**;}	Наделяет связанную со свойством кнопку способностью перехватывать нажатие клавиши Enter
public Form ActiveForm{**get**;}	Возвращает ссылку на активное окно
public Form ActiveMDIChild {**get**;}	Возвращает ссылку на активное дочернее MDI-окно
public bool AutoScale{**get**; **set**;}	Если содержит **true**, окно будет автоматически изменять свои размеры и размеры своих управляющих компонентов на основе размера текущего шрифта
public bool AutoScroll{**get**; **set**;}	Если содержит **erue**, окно будет автоматически вставлять полосы прокрутки, чтобы получить доступ к компонентам, не умещающимся в пределах текущих размеров формы
public IButtonControl CancelButton{**get**; **set**;}	Содержит ссылку на умалчиваемую кнопку, закрывающее окно с результатом Cancel при нажатии клавиши Esc
public bool ControlBox{**get**; **set**;}	Если содержит **false**, окно не будет содержать системные кнопки в своем заголовке
public Rectangle DesktopBounds{**get**; **set**;}	Содержит положение и размеры окна в координатах экрана
public Point DesktopLocation {**get**; **set**;}	Содержит положение левого верхнего угла окна в координатах экрана
public DialogResult: DialogResult;	Указывает результат, который модальное диалоговое окно вернет вызывающей программе. Все члены перечисления обычно соответствуют

	щелчку на одноименной кнопке. Значение None означает, что диалог еще продолжается
enum FormBorderStyle {Fixed3d, FixedDialog, FixedSingle, FixedToolWindow, None, Sizable, SizableToolWindow} **public** FormBorderStyle FormBorderStyle{**get**; **set**;}	Определяет стиль рамки окна: Fixed3d — фиксированных размеров трехмерная рамка; FixedDialog — фиксированных размеров рамка для диалоговых окон; FixedSingle — фиксированных размеров рамка толщиной в одну линию; FixedToolWindow — фиксированных размеров рамка для инструментального окна; None — окно без рамки; Sizable — обычная трехмерная рамка; SizableToolWindow — рамка для инструментального окна с изменяемыми размерами
public bool HelpButton{**get**; **set**;}	Если имеет значение True, в заголовок окна вставляется кнопка со значком в виде знака вопроса; игнорируется, если в заголовке окна имеются копки свертывания и развертывания
public Icon Icon{**get**; **set**;}	Определяет значок, который появляется в левом верхнем углу заголовка
public bool IsMDIChild{**get**;}	Возвращает **true**, если окно является дочерним для интерфейса MDI
public bool IsMDIContainer {**get**; **set**;}	Если имеет значение **true**, окно является рамочным для интерфейса MDI
public MainMenu Menu{**get**; **set**;}	Содержит указатель на главное меню формы
public MainMenu MergedMenu{**get**; **set**;}	Содержит объединенное меню рамочного и дочернего MDI-окон
public bool KeyPreview{**get**; **set**;}	Если имеет значение **true**, окно получает события от клавиатуры до того, как эти события получит компонент с фокусом ввода
public bool MaximizeBox{**get**; **set**;}	Если имеет значение **true**, в заголовок окна вставляется кнопка развертывания
public Rectangle MaximizedBounds{**get**; **set**;}	Определяет размеры и положение развернутого окна
public Size MaximumSize{**get**; **set**;}	Определяет размеры развернутого окна
public Form[] MDIChildren {**get**;}	Открывает индексированный доступ к дочерним MDI-окнам
public Form MDIParent{**get**; **set**;}	Содержит указатель на рамочное MDI-окно
public bool MinimizeBox{**get**; **set**;}	Если имеет значение True, в заголовок окна вставляется кнопка свертывания
public Rectangle MinimizedBounds{**get**; **set**;}	Определяет размеры и положение свернутого окна
public Size MinimumSize{**get**; **set**;}	Определяет размеры свернутого окна
public bool Modal{**get**;}	Содержит **true**, если окно показывается в модальном режиме

public double Opacity{**get**; **set**;}	Определяет степень прозрачности окна
public Form[] OwnedForms{**get**;}	Содержит коллекцию всех форм, для которых данная форма является родительской
public Form Owner{**get**; **set**;}	Содержит ссылку на родительскую форму
public bool ShowInTaskbar{**get**; **set**;}	Разрешает/запрещает показывать ссылку на форму на панели задач Windows
public bool TopMost{**get**; **set**;}	Если содержит **true**, окно формы располагается поверх всех других окон
public Color TransparencyKey{**get**; **set**;}	Определяет цвет прозрачности (см. далее)

Свойство **AcceptButton** обычно используется в диалоговых окнах для придания кнопке свойства умалчиваемой, то есть реагирующей на нажатие клавиши Enter. Замечу, что реакция на нажатие клавиши Enter реализуется в обработчике события **Click** умалчиваемой кнопки. В отличие от этого, свойство **CancelButton** указывает на кнопку, которая при нажатии клавиши Esc закрывает диалоговое окно с результатом **DialogResult.Cancel**.

В следующем примере демонстрируются оба свойства. На пустую форму поместите компонент TextBox и кнопку Button. В конструкторе формы сделайте кнопку умалчиваемой:

```
public Form1()
{
   InitializeComponent();
   AcceptButton = button1;
   Text = "Демонстрация AcceptButton";
}

private void button1_Click(object sender, EventArgs e)
{
   Form FmDlg = new Form();        // Создаем диалоговое окно
   FmDlg.Text = "Демонстрация CancelButton";
   FmDlg.FormBorderStyle = FormBorderStyle.FixedDialog;

   Button Btn = new Button();      // Создаем кнопку
   Btn.Text = "Cancel";
   // Располагаем ее в центре формы:
   Btn.Left = (Width - Btn.Width) / 2;
   Btn.Top = (Height - Btn.Height) / 2;
   // Создаем текстовое поле:
   TextBox textBox = new TextBox();
   // Располагаеи его над кнопкой:
   textBox.Top = Btn.Top - 2 * textBox.Height;
   textBox.Left = (Width - textBox.Width) / 2;

   FmDlg.Controls.Add(textBox);    // Добавляем к форме поле
   FmDlg.Controls.Add(Btn);        // Добавляем кнопку
   FmDlg.CancelButton = Btn;
```

```
    FmDlg.ShowDialog();
}
```

Заметьте: на пустую форму вначале помещено поле textBox1, которое получает в свойство TabIndex значение 0. Таким образом, это поле в момент старта программы имеет фокус ввода. Тем не менее, нажатие клавиши Enter активизирует кнопку button1, так как она объявлена в свойстве AcceptButton формы. В результате срабатывает обработчик кнопки, который создает и показывает диалоговое окно с кнопкой Cancel. В этом окне также есть поле ввода, которое получает фокус в момент появления окна. Это не мешает при нажатии клавиши Esc сработать кнопке Cancel, закрыв диалоговое окно.

Если в свойство DialogResult диалогового окна поместить значение перечисления DialogResult, окно будет закрыто с указанным результатом. Например, следующий обработчик закроет диалоговое окно и вернет в вызывающую программу значение Abort:

```
private void button1_Click(object sender, EventArgs e);
{
  this.DialogResult = DialogResult.Abort;
}
```

Свойства MaximizeBounds и MinimizeBounds определяют размеры и положение развернутого и свернутого окон. Эти свойства недоступны в окне инспектора объектов, и их нужно устанавливать динамически. Наиболее подходящим местом для этого является конструктор окна. Например:

```
public Form1()
{
    InitializeComponent();
    Rectangle R = new Rectangle(0, 0, 300, 400);
    this.MaximizeBounds = R;
}
```

Свойства MaximizeSize и MinimizeSize определяют размеры развернутого и свернутого окна. Если они имеют размер 0 × 0, эти размеры устанавливает Windows. Эти свойства игнорируются, если заданы свойства MaximizeBounds и MinimizeBounds.

Свойство Opacity определяет степень прозрачности окна. Значение 1 соответствует непрозрачному окну, значение 0 — полностью прозрачному. Это свойство работает только в Windows 2000/XP и при условии, что в видеокарте компьютера на каждый пиксел приходится по два и более байтов видеопамяти (режимы High Color, True Color).

Свойства TransparencyKey задает цвет прозрачности: любые участки формы или размещенных на ней компонентов, окрашенные этим цветом, будут прозрачными. Для воспроизведения примера поместите на пустую форму три панели и каждую из них окрасьте своим цветом (свойство BackColor). Создайте для одной из них обработчик события Click и затем назначьте этот обработчик двум другим панелям:

```
public Form1()
{
   InitializeComponent();
   // Раскрашиваем панели цветами российского флага:
   panel1.BackColor = Color.White;
   panel2.BackColor = Color.Blue;
   panel3.BackColor = Color.Red;
}

private void panel1_Click(object sender, EventArgs e)
{
   this.TransparencyKey = (sender as Panel).BackColor;
}
```

Запустите программу и щелкните на любой панели — она тут же станет прозрачной.

Методы формы представлены наиболее важные методы класса Form, не унаследованные от **Control**.

Таблица 19.2. - Методы класса Form

Метод	Назначение
public void Activate()	Активизирует форму и передает ей фокус ввода
public void AddOwnerForm (Form OwnedForm)	Устанавливает владельца текущей формы
enum MDILayout {ArrangeIcons, Cascade, TileHorizontal, TileVertical}; **public void** LayoutMDI(MDILayout Value)	Устанавливает положение дочерних окон внутри рамочного MDI-окна: ArrangeIcons — упорядочивает значки свернутых дочерних окон; Cascade — располагает дочерние окна каскадом; TileHorizontal — упорядочивает положение и размеры дочерних окон по горизонтали; TileVertical — упорядочивает положение и размеры дочерних окон по вертикали
public void RemoveOwnedForm (Form OwnedForm)	Удаляет указанную форму из списка владельцев текущей формы
public void SetDesktopBounds(**int** X, **int** Y, **int** Width, **int** Height)	Устанавливает положение формы на экране и ее размеры
p[ublic void SetDesktopLocation (**int** X, **int** Y)	Устанавливает положение формы на экране
public DialogResult ShowDialog()	Показывает окно в режиме диалога (информацию о перегруженных методах см. в справочной службе)

Метод AddOwnerForm() устанавливает владельца текущей формы до тех пор, пока не будет вызван метод RemoveOwnerForm(). Форма, имеющая владельца,

свертывается и закрывается, когда свертывается или закрывается ее владелец. Она появляется всегда на поверхности своего владельца. Примером могут быть диалоговые окна поиска и замены. Собственника формы можно удалить или заменить также с помощью свойства Owner.

Метод ShowDialog() показывает модальное диалоговое окно. Это окно блокирует взаимодействие пользователя с другими окнами до тех пор, пока тот не закроет его. Чтобы показать/спрятать обычное (немодальное) окно, используются унаследованные от Control методы Show() и Hide().

События формы

В табл. 19.3 перечислены события формы (кроме унаследованных от класса Control).

Таблица 19.3. События класса Form

Событие	Описание
public event EventHandler Activated	Возникает при активизации окна
public event EventHandler Closed	Возникает при закрытии окна
public event CancelEventArgs Closing	Возникает перед закрытием окна. Параметр e содержит логическую переменную Cancel, с помощью которой программа может отменить действие
public event EventHandler Deactivated	Возникает, когда форма теряет фокус ввода
public event InputLanguageChan-gedEventHandler InputLanguageChanged	Возникает при изменении языка ввода. Параметр e может быть равен одному из следующих значений: CharSet — значение типа Byte, определяющее набор символов; Culture — объект класса CultureInfo, содержащий параметры локализации; InputLanguge — объект класса InputLanuage, определяющий новый язык ввода
public event InputLanguageChan-gingEventHandler InputLanguageChanging	Возникает при попытке пользователя изменить язык ввода. Параметр e может быть равен одному из следующих значений: Cancel — логическая переменная, с помощью которой обработчик может отменить действие; SysCharSet — значение типа Boolean, указывающее, будет ли умалчиваемый системный шрифт поддерживать требуемый набор символов; Culture — объект класса CultureInfo, содержащий параметры локализации; InputLanguge — объект класса InputLanuage, определяющий новый язык ввода
public event EventHandler	Возникает перед появлением формы на

Load	экране
public event EventHandler MDIChildActivate	Возникает при активизации дочернего MDI-окна
public event EventHandler MenuComplete	Возникает, когда меню окна теряет фокус ввода
public event EventHandler MenuStart	Возникает, когда меню окна получает фокус ввода

Интерфейс MDI

Некоторые свойства, методы и события класса Form направлены на поддержку многодокументного интерфейса (Multiple Documents Interface, MDI). В этом интерфейсе создается главное (или рамочное) окно, в клиентской области которого размещаются несколько дочерних окон. Примером программы, использующей этот интерфейс, может служить Microsoft Excel.

При разработке MDI-приложения нужно сначала создать рамочное окно, а затем — клиентские.

Для создания рамочного окна в свойство IsMDIContainer главной формы поместите значение **true**. Дочерние окна создаются динамически, так как для этого нужно использовать свойство формы MdiParent, которое недоступно на этапе конструирования. Вот пример обработчика выбора в меню рамочного окна команды Файл ▸ Новое окно:

```
privite void menuItem1_Click (object sender, EventArgse)
// Создает новое дочернее окно
{
    Form Client = new Form();
    Client.MdiParent = this;
    Client.Show
}
```

Класс Application

Объекты класса Application автоматически создаются для каждого Windows-приложения. Он инкапсулирует общие для всех приложений свойства, методы и события.

Свойства класса Application

В табл. 19.4 перечислены наиболее важные свойства класса Application.

Таблица 19.4. Свойства класса Application

Свойство	Описание
public bool AllowQuit {**get**;}	Возвращает значение, указывающее, может ли обращающийся к этому свойству фрагмент программы закрыть ее
public string CommonAppDataPath {**get**;}	Возвращает маршрут доступа к данным, разделяемым с другими пользователями
public string CommonAppDataRegistery {**get**;}	Возвращает ключ реестра для приложения, которое разделяет данные с другими пользователями
public CultureInfo	Содержит ссылку на объект с параметрами

CurrentCulture{**get**; **set**;}	локализации
public InputLanguage CurrentInputLanguage{**get**;}	Возвращает ссылку на объект, определяющий текущий язык ввода
public string ExecutablePath{**get**;}	Возвращает маршрут доступа и имя файла приложения
public bool MessageLoop{**get**;}	Указывает, будут ли Windows-сообщения обрабатываться в текущем программном потоке
public string StartupPath{**get**;}	Возвращает маршрут доступа к исполняемому файлу приложения (без имени файла)
public string UserAppDataPath{**get**;}	Возвращает маршрут доступа к данным приложения
public string UserAppDataRegistery{**get**;}	Возвращает ключ реестра для приложения

Методы класса Application

В табл. 19.5 перечислены методы класса Application.

Таблица 19.5. Методы класса Application

Метод	Описание
public void AddMessageFilter(IMessageFilter Filter)	Добавляет фильтр для отсева сообщений
public void DoEvents()	Обрабатывает сообщения из очереди Windows-сообщений
public void EnableVisualStyles()	Отрисовывает компоненты приложения в стиле Windows XP
public void Exit()	Информирует Windows о завершении работы приложения и закрывает программу после обработки всех сообщений из очереди
public void ExitTnread()	Завершает работу текущего потока и закрывает все его окна
public void RemoveMessageFilter(IMessageFilter Filter)	Удаляет фильтр для отсева сообщений
public void Run()	Запускает цикл обработки сообщений в текущем потоке (информацию о перегруженных методах см. в справочной службе)

Метод DoEvents() приостанавливает текущую работу до тех пор, пока не будут обработаны все сообщения из очереди сообщений. Например, при изменении текста надписи в очередь помещается соответствующее сообщение, но отрисовка надписи откладывается до завершения текущей работы. Вызов метода позволяет прервать текущую работу и отрисовать надпись заново.

События класса Application

В табл. 19.6 представлены события класса Application.

Таблица 19.6. События класса Application

Событие	Описание
public static event EventHandler ApplicationExit	Возбуждается методом Exit()
public static event EvenentHandler Idle	Возникает при исчерпании очереди сообщений и наступлении паузы в работе приложения
public static event EventHandvler ThreadExit	Возбуждается методом ThreadExit()

Тема 3.5.1 Организация взаимодействия форм

Тема 3.5.2 Стандартные диалоги и сообщения

OpenFileDialog и SaveFileDialog — диалоговые окна открытия и сохранения файлов.

Компоненты OpenFileDialog и SaveFileDialog имеют приблизительно одинаковый набор свойств и методов и поэтому рассматриваются вместе. Первый из них предназначен для создания и обслуживания окна открытия (рис. 18.28), второй — окна сохранения файлов (рис. 18.29).

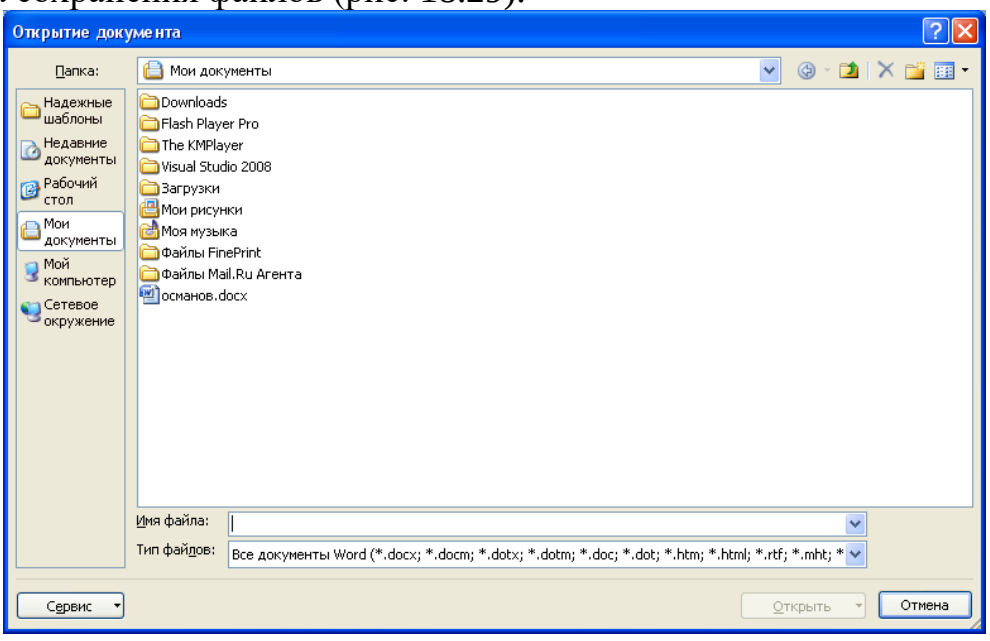


Рис. 18.28. Диалоговое окно открытия файла

Рис. 18.29. Диалоговое окно сохранения файла

Для вызова диалоговых окон используются методы компонентов **ShowDialog**, которые возвращают результат диалога с пользователем в виде члена перечисления **DialogResult.** Если метод возвращает **DialogResult.OK**, это означает, что пользователь выбрал (или ввел) правильное имя файла и щелкнул по кнопке OK диалогового окна открытия или сохранения, то есть может выполняться соответствующая операция.

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {}
```

Оба компонента имеют функцию **OpenFile**, которая возвращает результат в виде потока для чтения/записи файла. Эта функция возвращает объект класса Stream, поэтому необходимо явное приведение типов:

```
FS = OpenFileDialog1.OpenFile() as FileStream;
```

Свойство **Filter** задает одну или несколько масок выбора файлов. Например:

```
OpenFileDialog1.Filter =
"Текстовые файлы (*.txt)|*.txt|" +
"Все файлы (*.*)|*.*";
```

FileName -

В строке используются символы вертикальной черты (|) для отделения маски от ее описания и масок друг от друга. Свойство FilterIndex определяет индекс маски (индексация начинается с 1).

FolderBrowserDialog — диалоговое окно выбора папки (каталога)

Компонент FolderBrowserDialog создает окно, показанное на рис. 18.30.

Использование компонента FolderBrowserDialog мало чем отличается от использования рассмотренных ранее компонентов диалоговых окон открытия и сохранения документов: метод ShowDialog() создает окно, а свойство SelectedPath содержит выбранный маршрут доступа. Например:

```
private void button1_Click(object sender, EventArgs e)
{

   if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
      textBox1.Text = folderBrowserDialog1.SelectedPath;
}
```

Свойство Description компонента определяет произвольный текст, который размещается в верхней части окна. Свойство **RootFolder** определяет папку, с которой начинается просмотр дерева папок. Если в свойство **ShowNewFolderButton** установить значение **true** (это значение установлено по умолчанию), в окно будет помещена кнопка **Создать папку**.

FontDialog — диалоговое окно выбора шрифта

Компонент **FontDialog** обслуживает стандартное окно выбора шрифта (рис. 18.31).

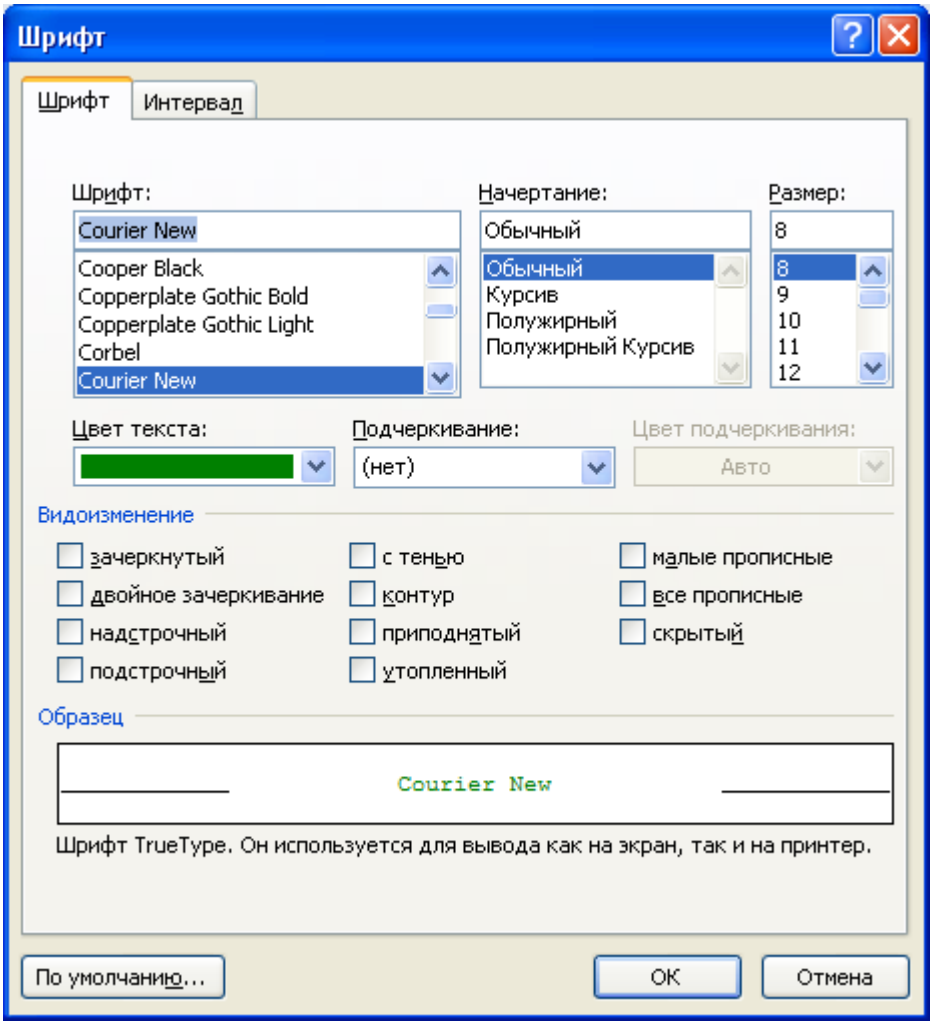


Рис. 18.31. Окно выбора шрифта

Диалоговое окно создается стандартным образом — обращением к его методу **ShowDialog**:

```
if (fontDialog1.ShowDialog() == DialogResult.OK) {}
```

В своем свойстве **Font** компонент возвращает выбранный шрифт, а в свойстве **Color** — его цвет.

Некоторые другие свойства компонента перечислены в табл. 18.3.

Таблица 18.3. Свойства класса FontDialog

Свойство	Назначение
public bool AllowScriptChange {**get**; **set**;}	Если содержит **true**, пользователь может изменять набор символов (в этом случае в окне появляется список Набор символов)
public bool AllowSimulations{**get**; **set**;}	Если содержит **true**, шрифт может моделироваться
public bool AllowVectorFonts{**get**; **set**;}	Если содержит **true**, пользователь может выбирать векторные шрифты
public bool AllowVerticalFonts{**get**; **set**;}	Если содержит **true**, в окне отображаются как горизонтальные, так и вертикальные шрифты
public int	Устанавливает

MaxSize{**get**; **set**;}	максимальный размер шрифта
public int MinSize{**get**; **set**;}	Устанавливает минимальный размер шрифта
public bool ScriptsOnly{**get**; **set**;}	Если содержит **true**, не показываются нестандартные шрифты
public bool ShowApply{**get**; **set**;}	Если содержит **true**, в окне будет кнопка Применить
public bool ShowColor{**get**; **set**;}	Если содержит **true**, в окне будет список выбора цвета шрифта
public bool ShowEffects{**get**; **set**;}	Если содержит **true**, в окне будут флажки для выбора подчеркнутого и зачеркнутого шрифтов

ColorDialog — диалоговое окно выбора цвета

Компонент ColorDialog создает и показывает стандартное диалоговое окно выбора цвета (рис. 18.32).

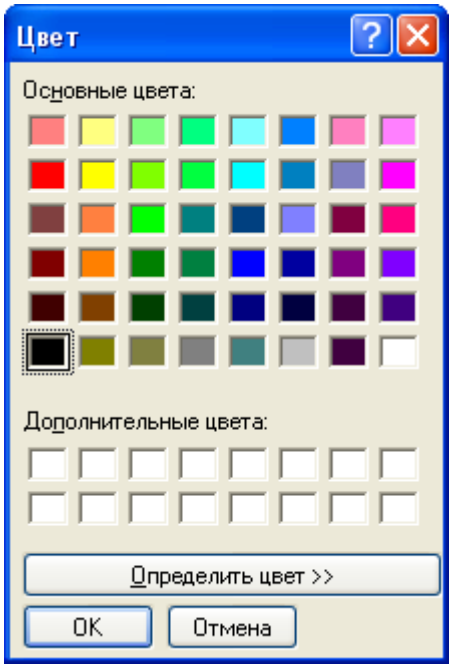


Рис. 18.32. Диалоговое окно выбора цвета

Это окно появляется после обращения к методу ShowDialog компонента:

```
if (colorDialog1.ShowDialog() == DialogResult.OK) {...}
```

После нормального завершения диалога выбранный пользователем цвет содержит свойство **Color.**

Другие свойства компонента представлены в табл. 18.4.

Таблица 18.4. Свойства класса ColorDialog

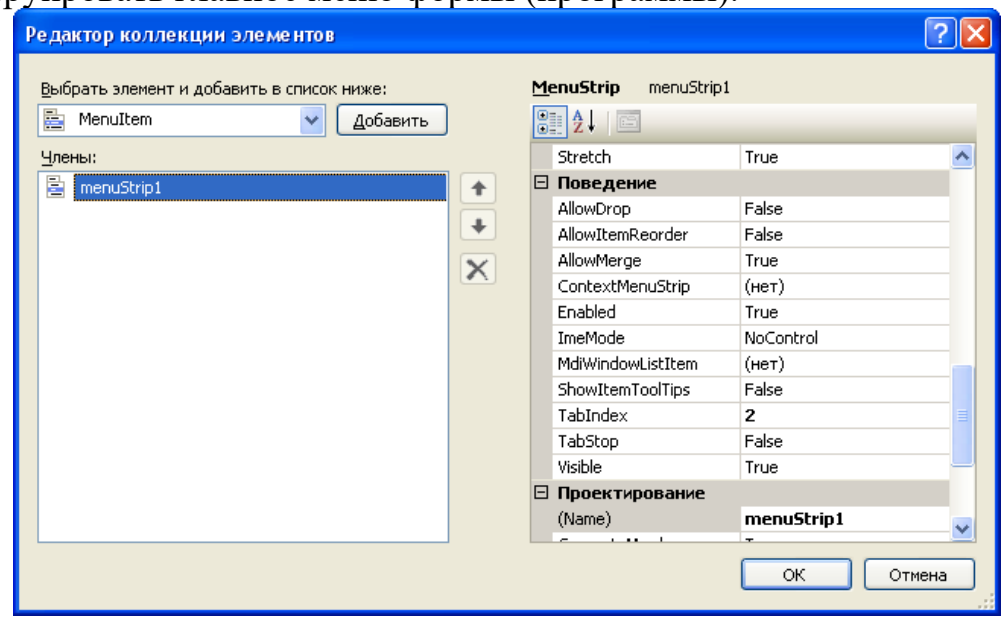
Свойство	**Назначение**
public bool AllowFullOpen{**get**; **set**;}	Если содержит **true**, пользователь может задействовать дополнительное окно определения цвета
public bool AnyColor{**get**; **set**;}	Если содержит **true**, пользователь может выбирать любые цвета (не обязательно

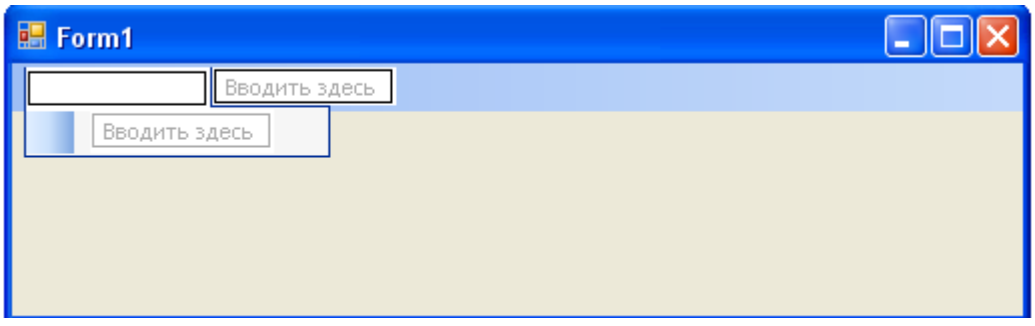
	чистые, то есть относящиеся к цветам из основного набора)
public int[] CustomColors{**get**; **set**;}	Определяет дополнительные цвета
public bool FullOpen{**get**; **set**;}	Если содержит **true**, окно появляется с развернутым дополнительным окном
public bool ShowHelp{**get**; **set**;}	Если содержит **true**, в окне появляется кнопка Справка
public bool SolidCilorOnly{**get**; **set**;}	Если содержит **true**, в окне отображаются только чистые (относящиеся к основному набору) цвета

Тема 3.6 Меню

MenuStrip — главное меню.

Компонент класса **MenuStrip** обладает редактором, позволяющим достаточно просто сконструировать главное меню формы (программы).





Для создания меню поместите значок компонента на форму. У верхней кромки формы появится светлая полоса с приглашением Type Here (вводите здесь).

Как только начнется ввод, справа и снизу от этого места появятся аналогичные надписи (рис. 18.24).

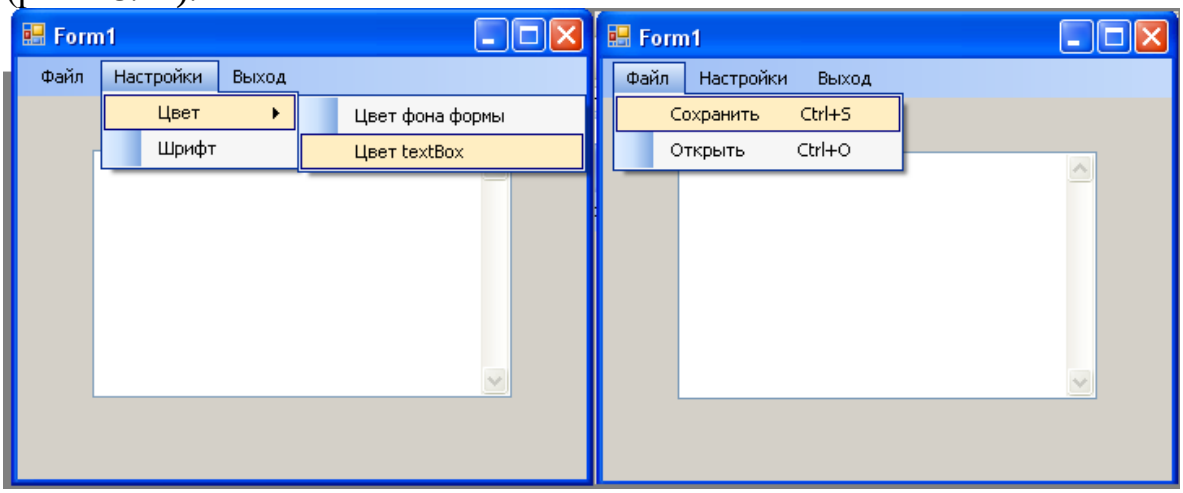


Рис. 18.24. Создание главного меню

Каждая команда меню представляет собой объект класса MenuItem, имеющий событие Click. Обработчик события должен выполнить необходимые действия, связанные с выбранной командой.

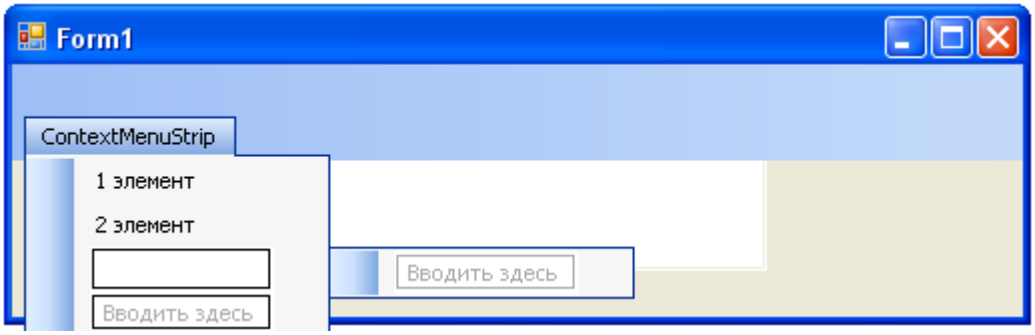
```
private void сохранитьToolStripMenuItem_Click(object sender, EventArgs e)
    { }
```

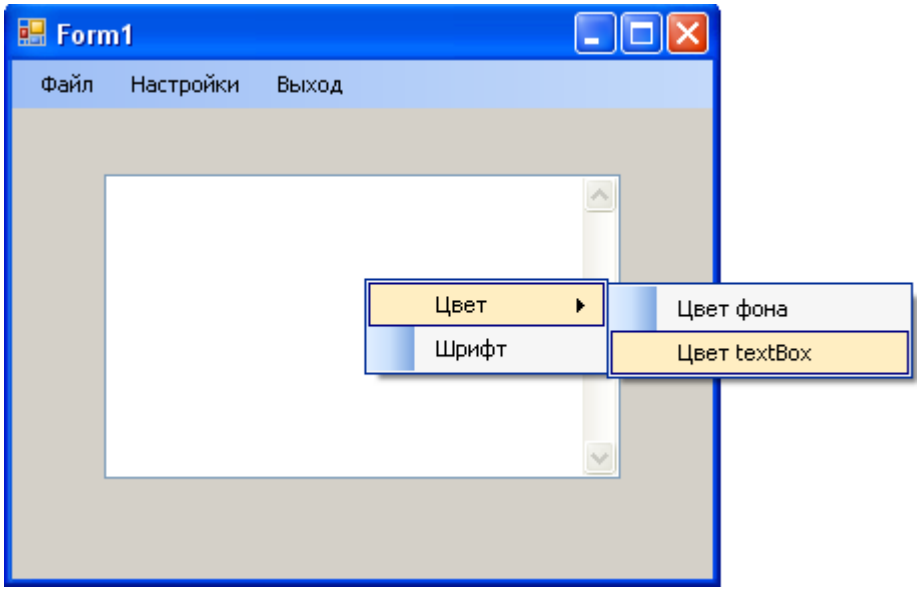
Если в надписи команды какому-то символу предшествует символ амперсанда (&), этот символ в команде подчеркивается, а сама команда выбирается клавишами Alt+<Буква>. Например, нажатие клавиш Alt+Ф для меню, показанного на рис. 18.24, эквивалентно щелчку мыши на команде Файл.

Для одной формы может быть создано сколько угодно главных меню, но рабочим будет только то из них, которое указано в свойстве MainMenuStrip формы. Это позволяет менять главное меню динамически, в зависимости от контекста программы.

ContextMenuStrip — контекстное меню

Контекстное меню ContextMenu отличается от главного только тем, что связано с конкретным элементом управления и появляется при щелчке на нем правой кнопкой мыши. Процесс создания контекстного меню и его связывания с программой ничем не отличается от создания и связывания с программой главного меню.

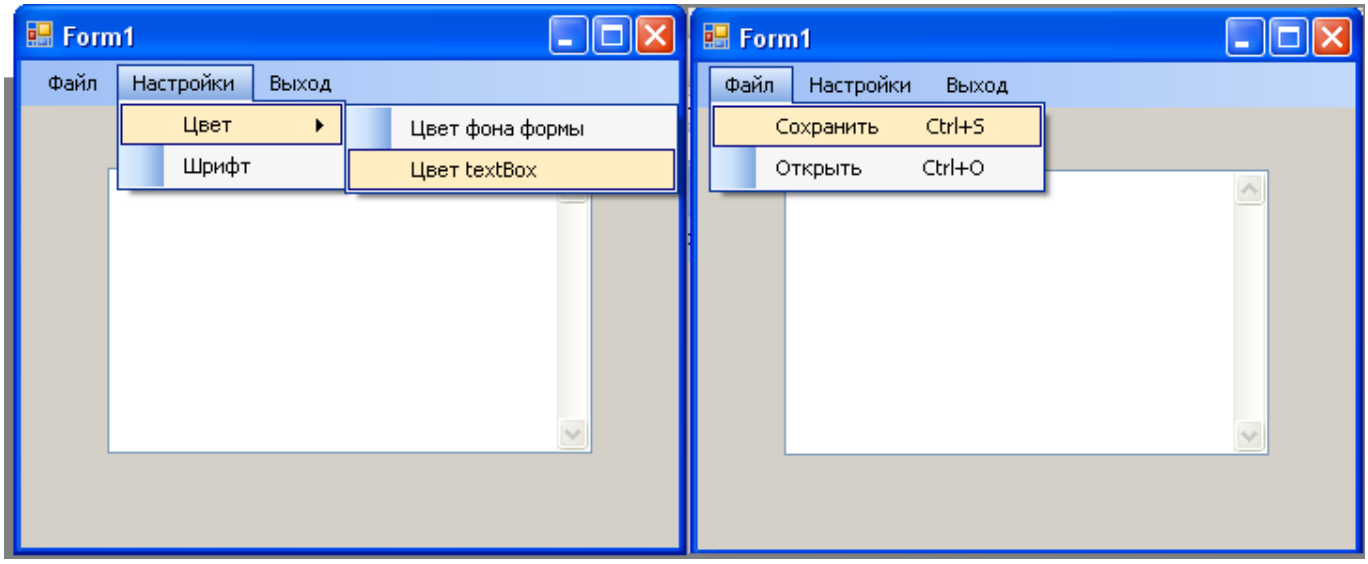


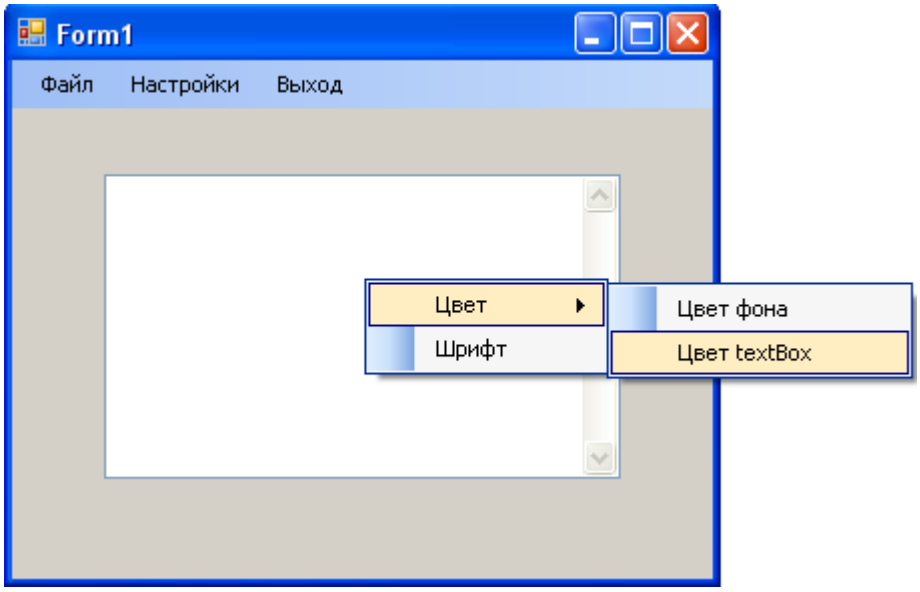


Пример: Создать главное меню с пунктами:

1) Файл: открыть, сохранить
2) Настройки: цвета (фон формы и textbox) и шрифт
3) Выход

А так же контекстное меню с пунктами: сохранить и открыть.






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

namespace WindowsFormsApplication18
{
    public partial class Form1 : Form
    {
        string fn = string.Empty;
        public Form1()
        {
            InitializeComponent();

            openFileDialog1.Filter="текстовые файлы (*.txt)|*.txt|"+
            "Все файлы (*.*)|*.*";      // фильтр для открытия файлов

        }

        private void выходToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();// закрытие формы
        }

        private void сохранитьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                fn = saveFileDialog1.FileName;// загрузка имени файла
                   {
                 // объектный файл для записи
                System.IO.StreamWriter sw = new
                System.IO.StreamWriter(fn);
                sw.Write(textBox1.Text); // запись в файл
                 sw.Close();          // закрытие файла
                    }
                }
        }

        private void открытьToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

```
            {

                fn = openFileDialog1.FileName;// загрузка имени файла
                 // объектный файл для чтения
                System.IO.StreamReader sr = new System.IO.StreamReader(fn);
                textBox1.Text = sr.ReadToEnd();
                sr.Close(); // закрытие файла
            }
        }

        private void цветФонаToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
                            BackColor = colorDialog1.Color;
                        //цвет фона
        }

        private void цветTextBoxToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
              textBox1.BackColor = colorDialog1.Color;    //цвет фона
         }

        private void шрифтToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if ( fontDialog1.ShowDialog() == DialogResult.OK)
                textBox1.Font = fontDialog1.Font; // шрифт
        }

    }
}
```

Тема 3.7 Управление приложениями и экраном

Тема 3.8 Обработка исключений

При разработке сложных программных систем практически невозможно избежать ошибок, которые проявляются не всегда (иначе они были бы обнаружены и устранены), но в большинстве случаев приводят к краху системы или к получению неверного результата. Причиной таких ошибок являются так называемые *исключительные ситуации*, или *исключения*. Они могут быть связаны с попыткой деления на ноль, открытием несуществующего файла, выходом индекса массива из допустимых границ и т. п.

Средства обработки исключений

Как и в большинстве других современных языков программирования, в C# есть средства обработки исключений, то есть средства программной реакции на ошибочную ситуацию. Цель такой обработки — как минимум, известить пользователя о недостоверности получаемого результата, а как максимум — попытаться устранить последствия ошибки и получить достоверный результат.

Для реализации обработки исключительной ситуации в месте возможного обнаружения исключения создается так называемый защищенный блок. При возникновении исключения в защищенном блоке управление передается некоторому обработчику исключения, который может проинформировать

пользователя о некорректной работе программы и (или) попытаться устранить последствия ошибки.

Синтаксис защищенного блока таков:

```
try
{ // Защищеный блок кода
...}
catch (T1 e1)
{ // Обработчик исключения.
Срабатывает, если тип исключения T1
...}
...
catch(Tk ek)
{ // Обработчик исключения.
Срабатывает, если тип исключения Tк
...}
finally
{ // Блок финализации. Срабатывает в любом случае
...}
```

Здесь **try**, **catch**, **finally** — зарезервированные слова.

В C# есть специальный класс Exception, который вместе со своими многочисленными потомками участвует в обработке исключения: если возникло исключение, автоматически прерывается дальнейшее выполнение защищенного кода, создается объект класса Exception или одного из его потомков и в обработчиках **catch**() ищется первый по ходу программы блок, предназначенный для обработки исключения данного класса Ti. Блоку обработчика передается объект ei класса Ti, уточняющий обстоятельства возникновения ошибки. После срабатывания обработчика управление передается в блок финализации, в котором осуществляются некоторые общие действия, связанные с работой защищенного блока. В блок **finally** управление передается и в случае, если в защищенном блоке не возникла исключительная ситуация, то есть этот блок срабатывает в любом случае. В нем, например, могут закрываться открытые файлы или освобождаться ресурсы, выделенные защищенному блоку.

Допускается произвольное количество (в том числе — ни одного) обработчиков **catch**(), разрешается также не использовать блок **finally**. Защищенные блоки могут быть вложенными.

При построении защищенного блока необходимо блокировать выдачу неверного результата, который может ввести пользователя в заблуждение и привести к непредсказуемым последствиям — в этом состоит главное назначение механизма обработки исключительных ситуаций.

Итак, при возникновении исключительной ситуации создается объект класса Exception или одного из его потомков. Класс Exception уведомляет пользователя о возникшей проблеме.

Наиболее важные свойства этого класса показаны в табл. 15.1.

Таблица 15.1. Свойства класса Exception

Свойство	**Назначение**
HiperLink	Возвращает URL файла справки с описанием ошибки
Message	Возвращает текстовое описание ошибки
Source	Возвращает имя объекта или приложения, сгенерировавшего исключение
StackTrace	Возвращает состояние стека на момент возникновения исключения
InnerExcepton	Используется для сохранения сведений о текущем исключении между сериями исключений

Потомки **Exception** специализируются на обработке конкретных исключительных ситуаций. Например, в пространстве имен System определены такие важные классы, как ArgumentOutOfRangeException, IndexOutOfRange Exception, StackOverlowException и многие другие.

Исключение возбуждает в подавляющем большинстве случаев ядро операционной системы, однако в C# это может сделать и сама программа.

Генерация исключения

Для генерации исключения используется зарезервированное слово **throw** (*throw* по-английски означает бросить, поэтому иногда вместо *генерации* исключения говорят о *вбрасывании* исключения).

Продемонстрируем генерацию и технику обработки исключения на примере

Пример: Даны 2 целых числа (a и b), найти a/b.

```
private void button1_Click(object sender, EventArgs e)
    {
      try
      {
         int a = Convert.ToInt32(textBox1.Text);
         int b = Convert.ToInt32(textBox2.Text);
         label1.Text = "Результат = "+Convert.ToString(a / b);
      }
      catch (DivideByZeroException)
      { MessageBox.Show("Ошибка деления на ноль !!!"); }
      catch (FormatException)
      { MessageBox.Show("Ошибка формата исходных данных!!!"); }
      catch (Exception)
      { MessageBox.Show("Ошибка !!!"); }
    }
```

Результат прогона программы показан на рис. 15.1.

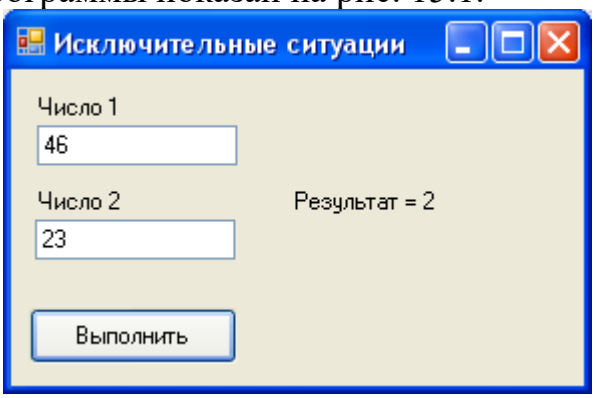


Рис. 15.1. Окно работающей программы

В примере используется объект исключения класса **Exception**. Как уже говорилось, этот класс предназначен для любых исключений. В табл. 15.2 описаны некоторые классы исключений, определенные в пространстве имен **System**.

Таблица 15.2. Некоторые классы исключений

Класс	**Описание ошибки**
AccessViolationException	Попытка доступа к защищенной памяти
ApplicationException	Обнаружение нефатальной ошибки
ArgumentException	Передача методу в одном из параметров вызова недопустимого значения
ArgumentNullException	Передача недопустимого значения **null** в качестве параметра вызова метода
ArgumentOutOfRange Exception	Параметр вызова вышел из допустимых границ
ArithmeticException	Арифметическая ошибка или ошибка приведения типа
ArrayTypeMismatch Exception	Попытка поместить в элемент массива значение недопустимого типа
BadImageFormatException	Попытка воспроизвести изображение из файла с недопустимым форматом
DataMisalignedException	Попытка разместить блок данных в памяти недостаточного размера
DivideByZeroException	Попытка деления на ноль
FieldAccessException	Обращение к закрытому или защищенному полю объекта
FormatException	Обнаружение

	несоответствия параметров и спецификации форматирования
IndexOutOfRangeException	Выход индекса массива из допустимых границ
InsuffisientMemoryException	Исчерпание доступной памяти
InvalidCastException	Недопустимое приведение типов
InvalidProgramException	Обнаружение ошибки в тексте программы на промежуточном языке CIL (обычно из-за ошибки компилятора)
MemberAccessException	Ошибка доступа к члену класса
MissingFieldException	Попытка доступа к несуществующему полю
MissingMemberException	Попытка доступа к несуществующему члену класса
MissingMethodException	Попытка доступа к несуществующему методу класса
MulticastNotSupportedException	Попытка объединения двух несовместимых делегатов
NotFinityNumberException	Значение числа с плавающей точкой равно бесконечности или не является числом
NoySupportedException	Исполняемый метод не поддерживается или делается попытка чтения, записи или смещения в потоке, который не поддерживает эту функциональность
NullReferenceException	Попытка обращения к ссылочному типу по неправильной ссылке
OperationCancelledException	Ошибочное завершение текущего программного потока
OutOfMemoryException	Нехватка памяти
PlatformNotSupportedException	Попытка выполнить CIL-программу на машине, которая не поддерживает платформу .NET

RankException	Передача методу массива недопустимой размерности
StackOverflowException	Переполнение стека
TimeoutException	Исчерпано время, выделенное процессу
FileNotFoundException	
FileLoadException	

В других пространствах имен определены собственные классы исключений. Например, в пространстве System.IO определены такие классы, как DdiveNotFoundException, FileNotFoundException, InvalidDataException.

Если ни один из стандартных классов исключений не удовлетворяет нужным целям, программист может создать собственный класс, объявив для него родителем класс **System.Exception**.

Захват исключения

Как уже говорилось, при вбрасывании исключения нормальное исполнение защищенного блока прерывается и в списке обработчиков **catch** ищется первый обработчик, способный обработать исключение данного типа. Если в защищенном блоке могут генерироваться несколько исключений, обработчики **catch** нужно располагать в определенной последовательности: самыми первыми должны располагаться наиболее специализированные обработчики, а обработчики типа Exception — последними. Если, например, в защищенном блоке возможно деление на ноль или переполнение стека, обрабатывать эти ошибки нужно так:

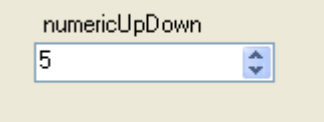
```
try
{...}
catch(DivideByZeroException e){...}  // Обработка деления на 0
catch(StackOverflowException e){...} // Обработка переполнения стека
catch(ArithmeticException e){...}    // Обработка любой арифметической
                                     // ошибки
cath(Exception e){...}               // Обработка любой ошибки
finally {...}
```

Если изменить порядок следования на обратный, ни один из специализированных обработчиков никогда не получит управления.

```
try
{
  if openFileDialog2.ShowDialog();
  { StreamReader a =
       new StreamReader(openFileDialog1.FileName);
   textBox2.AppendText(a.ReadToEnd());
   a.Close();
      }
}
catch (System.IO.FileNotFoundException)
{
   MessageBox.Show("Файл не найден!");
}
```

Тема 3.9 Элементы интерфейса
Тема 3.9.1 Реверсивные счетчики.
Компоненты numericUpDown, monthCalendar, DateTimePicker, NotifyIcon

numericUpDown – Представляет регулятор Windows (также известный как элемент управления "вверх-вниз"), отображающий числовые значения.



Элемент управления **NumericUpDown** содержит одно числовое значение, которое можно увеличить или уменьшить, щелкая кнопки ВВЕРХ или ВНИЗ элемента управления.

Существует возможность настройки отображения значений в элементе управления Windows Forms NumericUpDown.

Свойства

Наименование	Назначение
ReadOnly	Запрет на редактирование если false
DecimalPlaces	определяет количество знаков после десятичной запятой, по умолчанию используется нулевое значение
ThousandsSeparator	определяет, будет ли вставлен разделитель между каждыми тремя десятичными знаками; по умолчанию используется значение **false**
Hexadecimal	Если значение **true**, то этот элемент управления может отображать значения в шестнадцатеричном формате вместо десятичного; по умолчанию используется значение **false**
Minimum Maximum	диапазон значений элемента управления
Increment	задать значение, используемое для увеличения или уменьшения
Accelerations	Скорость изменения чисел в элементе управления при постоянно нажатой пользователем кнопке со стрелкой вверх или вниз

Value	Значение в компоненте
	
	numericUpDown1.Value = 5;
	numericUpDown1.Maximum = 2500;
	numericUpDown1.Minimum = -100;

Форматирование численного отображения осуществляется с помощью свойств DecimalPlaces, Hexadecimal и ThousandsSeparator. Чтобы значения отображались в шестнадцатеричном формате, свойство Hexadecimal должно быть равно **true**. Чтобы в десятичных числах отображался разделитель групп разрядов, свойство ThousandsSeparator должно быть равно **true**. Чтобы задать число десятичных разрядов, которые будут отображаться после разделителя целой и дробной частей, свойству DecimalPlaces нужно присвоить значение, равное этому числу десятичных разрядов.

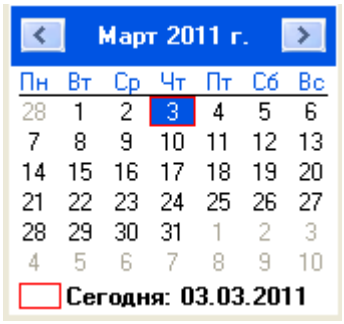
Пример

```
public void InstantiateMyNumericUpDown()
{
   // Create and initialize a NumericUpDown control.
   numericUpDown1 = new NumericUpDown();

   // Dock the control to the top of the form.
   numericUpDown1.Dock = System.Windows.Forms.DockStyle.Top;

   // Set the Minimum, Maximum, and initial Value.
   numericUpDown1.Value = 5;
   numericUpDown1.Maximum = 2500;
   numericUpDown1.Minimum = -100;

   // Add the NumericUpDown to the Form.
   Controls.Add(numericUpDown1);
}

// Check box to toggle decimal places to be displayed.
private void checkBox1_Click(Object sender,
                             EventArgs e)
{
   /* If DecimalPlaces is greater than 0, set them to 0 and round the
      current Value; otherwise, set DecimalPlaces to 2 and change the
      Increment to 0.25. */
   if (numericUpDown1.DecimalPlaces > 0)
   {
      numericUpDown1.DecimalPlaces = 0;
      numericUpDown1.Value = Decimal.Round(numericUpDown1.Value, 0);
   }
   else
   {
      numericUpDown1.DecimalPlaces = 2;
      numericUpDown1.Increment = 0.25M;
   }
}

// Check box to toggle thousands separators to be displayed.
private void checkBox2_Click(Object sender,
                             EventArgs e)
{
   /* If ThousandsSeparator is true, set it to false;
      otherwise, set it to true. */
   if (numericUpDown1.ThousandsSeparator)
   {
      numericUpDown1.ThousandsSeparator = false;
   }
   else
   {
      numericUpDown1.ThousandsSeparator = true;
   }
}

// Check box to toggle hexadecimal to be displayed.
```

```
private void checkBox3_Click(Object sender,
                             EventArgs e)
{
   /* If Hexadecimal is true, set it to false;
      otherwise, set it to true. */
   if (numericUpDown1.Hexadecimal)
   {
      numericUpDown1.Hexadecimal = false;
   }
   else
   {
      numericUpDown1.Hexadecimal = true;
   }
}
```

При вызове метода UpButton или DownButton, как в коде, так и при нажатии кнопки ВВЕРХ или кнопки ВНИЗ, новое значение проверяется, и элемент управления обновляется, принимая новое значение в соответствующем формате. В частности, если свойство UserEdit установлено равным **true**, до проверки или изменения значения вызывается метод ParseEditText. После этого выполняется проверка попадания введенного значения в интервал значений свойств Minimum и Maximum, а затем вызывается метод UpdateEditText.

monthCalendar - Представляет элемент управления Windows, который позволяет выбрать дату с помощью визуального календаря на месяц.



Элемент управления **monthCalendar** позволяет пользователю визуально выбирать дату.

Свойства компоненты **monthCalendar представлены в таблице 15**

Таблица 15

Свойство	Назначение
MinDate	Минимальная выбираемая дата или время monthCalendar1.MinDate = new System.DateTime(1999, 1, 1, 0, 0, 0, 0);
MaxDate	Максимальная выбираемая дата или время monthCalendar1.MaxDate = new System.DateTime(2011, 12, 31, 0, 0, 0, 0);
ForeColor	Цвет фона monthCalendar1. ForeColor = System.Drawing.Color.Yellow;
Font	Размер и тип шрифта

TitleBackColor	monthCalendar1.TitleBackColor = System.Drawing.Color.Blue;
TitleForeColor,	monthCalendar1.TitleForeColor = System.Drawing.Color.Yellow;
TrailingForeColor	monthCalendar1.TrailingForeColor = System.Drawing.Color.Red;
BackColor	
AnnuallyBoldedDates	выделены полужирным шрифтом monthCalendar1.AnnuallyBoldedDates = new System.DateTime[] { new System.DateTime(2011, 3, 20, 0, 0, 0, 0), new System.DateTime(2011, 4, 28, 0, 0, 0, 0),};
BoldedDates	Даты которые будут выделены полужирным шрифтом ежегодно
MonthlyBoldedDates	выделены полужирным шрифтом
FirstDayOfWeek	
MaxSelectionCount	
ShowWeekNumbers	Если true отображение номеров недель

CalendarDimensions	// Конфигурация календаря i строки и j столбца по месяцу
	
	monthCalendar1.CalendarDimensions = new System.Drawing.Size(10, 3);

Элемент управления **MonthCalendar** рисуется операционной системой, поэтому событие Paint никогда не вызывается. Если нужно задать нестандартный вид для элемента управления **MonthCalendar**, следует переопределить метод OnPrint, вызвать базовую реализацию метода OnPrint и выполнить нестандартные операции рисования.

Если необходимо применить пользовательские форматы даты и ограничить выбор только одной датой, рекомендуется использовать элемент управления DateTimePicker вместо **MonthCalendar**. Использование элемента управления DateTimePicker во многом устраняет необходимость проверки значений даты и времени.

Также обрабатываются события DateSelected и DateChanged, состояние которых отображается на форме.

DateTimePicker — ввод и отображение даты или времени.

Компонент DateTimePicker позволяет отображать и (или) вводить дату/время.

Свойство Format управляет форматом отображаемых данных. Оно может иметь одно из следующих значений:

Long — дата с полным названием месяца;

Short — дата в формате ДД.ММ.ГГ;

Time — время;

Custom — формат задается свойством CustomFormat.

Строка свойства CustomFormat формируется с помощью спецификаторов формата даты/времени, показанных в табл. 18.1.

Таблица 18.1. Некоторые спецификаторы формата даты/времени

Спецификатор	**Описание**
d	Показывает день месяца в виде одной (1…9) или двух (10…31) цифр
dd	Показывает день месяца в виде двух цифр (01…31)
ddd	Показывает сокращенное название дня недели (пн…вс)
dddd	Показывает полное название дня недели (понедельник...воскресенье)
h	Показывает одну (1…9) или две (10…12) цифры часа до или после полудня
hh	Показывает две цифры часа (01…12) до или после полудня
H	Показывает одну (0…9) или две (10…23) цифры часа
HH	Показывает две (00…23) цифры часа
m	Показывает одну (0…9) или две (10…59) цифры минуты
mm	Показывает две (00…59) цифры минуты
M	Показывает одну (1…9) или две (10…12) цифры месяца
MM	Показывает две (01…12) цифры месяца
MMM	Показывает сокращенное название месяца (янв…дек)
MMMM	Показывает полное название месяца (Январь…Декабрь)
s	Показывает одну (0…9) или две (10…59) цифры секунды
ss	Показывает две (00…59) цифры секунды
y	Показывает одну (0…9) или две (10…99) последние цифры года
yy	Показывает две (00…99) последние цифры года
yyyy	Показывает все цифры года

Любые другие сочетания символов в свойстве CustomFormat воспроизводятся без их интерпретации.

Свойство ShowUpDown вставляет в правый угол компонента две небольшие стрелки, а свойство ShowCheckBox вставляет в левый угол компонента флажок.

Свойство Value хранит отображаемое компонентом значение даты-времени. В момент размещения компонента на форме в него помещается текущее значение DateTime.Now.

NotifyIcon — извещающий значок

Компонент NoifyIcon (невизуальный компонент) предназначен для создания в правом нижнем углу окна Windows — в так называемой *области уведомлений панели задач* — небольшого значка, с которым можно связать контекстное меню. Подобного рода значки обычно связываются с фоновыми процессами, такими как фоновая печать, сканирование вирусов, обслуживание файловой системы и т. п. На рис. 18.25 показана область уведомлений панели задач Windows со значком в виде косого креста, связанным с запущенной программой WinForm.

Рис. 18.25. Значок компонента NotifyIcon

Этот значок определяется свойством **Icon** компонента. При наведении на него указателя мыши (и в случае активности фоновой программы) появляется всплывающая подсказка, задаваемая свойством **Text**. Чтобы значок был видимым, необходимо в свойство **Visible** компонента поместить значение **true**.

Щелчок правой кнопкой на компоненте вызывает событие Click и может обрабатываться обычным образом (например, закрыть фоновую программу). Однако с ним можно связать контекстное меню (свойство **ContextMenu** компонента) и существенно расширить диапазон возможных действий.

	Тема 3.9.2 Панели инструментов Panel, NoolBar, CoolBar.	2
1.	Тема 3.9.3 Создание панели инструментов на основе компонента Form, строка состояния	2
	Итого по разделу 3	**48/20**
	Раздел 4 Работа с базами данных в Borland C++ Builder	
	Тема 4.1 Реляционные базы данных	-
2.	Тема 4.1.1 Банки данных.Модели данных.	2
3.	Тема 4.1.2 Ключи и индексы. Способы доступа к данным	2
4.	Тема 4.1.3 Связь между таблицами. Форматы таблиц	2
5.	**Лаб № 27** Создание баз данных	2
	Тема 4.2 Создание и средства для работы с базами данных	-
6.	Тема 4.2.1 Инструменты для работы с базами данных	2
7.	Тема 4.2.2 Компоненты для работы с базами данных	2
8.	Тема 4.2.3 Изменение сруктуры таблицы. Создание приложения BDE	2
9.	**Лаб № 28** Создание приложений с БД	2
10.	Тема 4.3 Поиск данных	2
11.	Тема 4.4 Работа с отчетами	2
12.	**Лаб № 29** Создание отчетов и поиск данных	2
13.	**Итого по разделу 4**	**22/6**
	Раздел 5. Основы программирования на языке	

	Java	
14.	Тема 5.1 Назначение языка Java. Средства разработки приложений Java	2
15.	Тема 5.2 Типы данных и операции над ними	2
16.	Тема 5.3 Объектно-ориентированное програмирование в Java	2
17.	Тема 5.4 Работа со строками и массивами	2
18.	Тема 5.5 Создание графического интерфейса	2
19.	**Лаб № 30** Создание приложения на Java	2
20.	Тема 5.6 Создание и работа с апплетами	2
	Итого по разделу 5	**14/2**
	Всего	**144/60**