Н. Культин

C#
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.06
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 C# в задачах и примерах. — СПб.:
БХВ-Петербург, 2007. — 240 с.: ил. + CD-ROM

ISBN 978-5-9775-0115-6

Книга представляет собой сборник задач и программ на языке C#. Примеры и задачи различной сложности — от простейших до приложений работы с графикой, мультимедиа и базами данных — демонстрируют назначение базовых компонентов, раскрывают тонкости разработки .NET-приложений в Borland Developer Studio или Turbo C#. Уделено внимание технологии ASP.NET. Справочник содержит описание базовых компонентов и часто используемых функций. Компакт-диск содержит дистрибутив Turbo C# Explorer, размещенный с разрешения Borland Software Corporation, а также проекты, рассматриваемые в книге.

Для начинающих программистов

УДК 681.3.06
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Римма Смоляк*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.05.07.
Формат 60×90$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 15.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0115-6

Оглавление

Предисловие **5**

Часть 1. Примеры и задачи **7**

БАЗОВЫЕ КОМПОНЕНТЫ 7

Конвертер 8
Электроэнергия 10
Автозаправка 14
Кафе 16
Жалюзи 19
Стеклопакет 22
Калькулятор 25
Просмотр иллюстраций 30
Секундомер 34
Таймер 37
Угадай число 40
Справочная система 44
Работа с несколькими формами 47

ФАЙЛЫ 50

Погода 50
Средняя температура 53
Редактор текста 56

ГРАФИКА 66

Приветствие 66
Диаграмма 68
График 72
Круговая диаграмма 77
Использование кистей 83
Фоновый рисунок 85
Бегущая строка 87
Полет в облаках 90

БАЗЫ ДАННЫХ 95

Записная книжка 95
Контакты 101
Ежедневник 109
Чтение данных из xml-файла 120
Конвертер базы данных Microsoft Access 122
Каталог 126

ИГРЫ И ДРУГИЕ ПОЛЕЗНЫЕ ПРОГРАММЫ 133

Парные картинки 133
Собери картинку 141
Сапер 147
Master Mind 157
Будильник 169
Экзаменатор 174

ASP.NET ... 184
Добро пожаловать ... 184
Хорошие окна ... 186
Авторизация ... 188
Контакты ... 193

Часть 2. Краткий справочник ... 197

Форма ... 197
Компоненты ... 199
Button ... 199
ComboBox ... 201
ContextMenu ... 202
CheckBox ... 202
CheckedListBox ... 204
GroupBox ... 205
ImageList ... 206
Label ... 206
ListBox ... 207
MainMenu ... 208
NotifyIcon ... 209
NumericUpDown ... 209
OpenFileDialog ... 210
Panel ... 211
PictureBox ... 212
RadioButton ... 213
ProgressBar ... 215
SaveFileDialog ... 215
StatusBar ... 216
TextBox ... 218
ToolBar ... 219
ToolTip ... 220
Timer ... 220
Графика ... 221
Карандаш ... 221
Кисть ... 222
Графические примитивы ... 224
Типы данных ... 226
Целый тип ... 226
Вещественный тип ... 226
Символьный тип ... 227
Строковый тип ... 227
Функции ... 227
Функции преобразования ... 227
Функции манипулирования строками ... 229
Функции манипулирования датами и временем ... 230
Функции манипулирования каталогами и файлами ... 232
Математические функции ... 234
События ... 236
Исключения ... 237

Приложение. Описание прилагаемого компакт-диска ... 238

Предметный указатель ... 239

Предисловие

Среда разработки Borland® C#Builder® for the Microsoft® .NET Framework является одним из популярнейших средств разработки компьютерных программ.

Microsoft .NET — это новая технология разработки программного обеспечения. В ее основе лежит идея универсального программного кода, который может быть выполнен любым устройством (компьютером), вне зависимости от используемой этим устройством операционной системы (операционная система устройства должна поддерживать технологию .NET). Универсальность программного кода обеспечивается за счет предварительной (на этапе разработки) компиляции исходной программы в промежуточный код, который во время загрузки (выполнения) транслируется в выполняемую программу.

Чтобы понять, что такое .NET, какими преимуществами обладает эта технология, необходимо опробовать ее в деле. Для этого нужно изучить среду разработки, понять технологию разработки, назначение и возможности компонентов. И здесь хорошим подспорьем могут стать примеры, программы, разработанные другими программистами.

В книге, которую вы держите в руках, собраны разнообразные примеры, демонстрирующие назначение базовых компонентов, технологию работы с графикой, базами данных, разработку ASP.NET-приложений. Следует обратить внимание, что большинство примеров не являются учебными в чистом смысле, это — вполне работоспособные программы.

Состоит книга из двух частей.

Первая часть содержит примеры. Примеры представлены в виде краткого описания, диалоговых окон и хорошо документированных текстов программ.

Вторая часть книги — это краткий справочник. В нем можно найти описание базовых компонентов и наиболее часто используемых функций.

Научиться программировать можно только программируя, решая конкретные задачи. Поэтому чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте (читайте) листинги, старайтесь понять, как работают программы. Не бойтесь экспериментировать — совершенствуйте программы, вносите в них изменения.

ЧАСТЬ 1

Примеры и задачи

БАЗОВЫЕ КОМПОНЕНТЫ

В этом разделе приведены примеры, демонстрирующие назначение и технологию работы с базовыми компонентами.

Общие замечания

- Процесс создания программы состоит из двух шагов: создание формы программы (диалоговое окно) и функций обработки *событий*. Форма *приложения* (прикладная программа, работающая в Windows) создается путем добавления в форму *компонентов* и последующей их настройки.
- В форме практически любого приложения есть компоненты, обеспечивающие интерфейс (взаимодействие) между программой и пользователем. Такие компоненты называют базовыми. К базовым компонентам можно отнести:
  - `Label` — поле вывода текста;
  - `TextBox` — поле редактирования текста;
  - `Button` — командная кнопка;
  - `CheckBox` — независимая кнопка выбора;
  - `RadioButton` — зависимая кнопка выбора;
  - `ListBox` — список выбора;
  - `ComboBox` — комбинированный список выбора;
  - `MainMenu` — главное меню программы;
  - `ContextMenu` — контекстное меню.

- Вид компонента, его размер и поведение определяют значения *свойств* (характеристик) компонента (описание свойств базовых компонентов можно найти в справочнике — во второй части книги).
- Основную работу в программе выполняют функции обработки *событий* (описание основных событий можно найти в справочнике — во второй части книги).
- Исходную информацию программа может получить, например, из полей редактирования (компонент `Edit`), списка выбора (компонент `ListBox`), комбинированного списка (компонент `ComboBox`). Для ввода значений логического типа можно использовать компоненты `CheckBox` и `RadoiButton`.
- Результат программа может вывести в поле вывода текста (компонент `Label`), в поле редактирования текста (компонент `TextBox`), в окно сообщения (функция `MessageBox.Show()`).
- Для преобразования текста (например, находящегося в поле редактирования) в целое число нужно использовать функцию `Convert.ToInt16()`, для преобразования в дробное — `Convert.ToDouble()`. Для преобразования численного значения переменной в строку нужно использовать функцию `ToString()`.

Конвертер

Программа **Конвертер** пересчитывает цену из долларов в рубли. Демонстрирует использование компонентов `TextBox` и `Label` для ввода и отображения числовых данных. Программа спроектирована таким образом, что пользователь может ввести в поля редактирования только правильные данные (числа). Форма программы приведена на рис. 1.1.

```
// нажатие клавиши в поле Цена
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
```

```
            e.Handled = true;
    }
}

// нажатие клавиши в поле Курс
private void textBox2_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox2.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
}

// щелчок на кнопке Пересчет
private void button1_Click(object sender, System.EventArgs e)
{
    double k;    // курс ( отношение рубля к доллару )
    double usd;    // цена в долларах
    double rub;    // цена в рублях

    label4.Text = "";

    // конструкция try { ... } catch { ... }
    // позволяет отлавливать возникающие программные ошибки
    try{
        // исходные данные
        usd = System.Convert.ToDouble(textBox1.Text);
        k   = System.Convert.ToDouble(textBox2.Text);

        // пересчет цены из долларов в рубли
        rub = usd * k;

        // вывод результата
        label4.Text = usd.ToString("N") +
            " USD = " + rub.ToString("C");
    }
```

```
    catch{
        if ((textBox1.Text == "") || (textBox2.Text == "")){
            MessageBox.Show("Ошибка исходных данных.\n" +
                "Необходимо ввести данные в оба поля.",
                "Конвертер", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        else
            MessageBox.Show("Ошибка исходных данных.\n" +
                "Неверный формат данных в одном из полей.",
                "Конвертер", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
    }
}

// щелчок на кнопке Завершить
private void button2_Click(object sender, System.EventArgs e)
{
    this.Close();
}
```

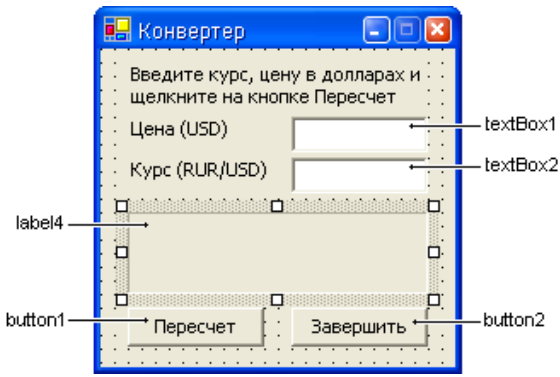

Рис. 1.1. Форма программы **Конвертер**

Электроэнергия

Программа **Электроэнергия** показывает, как одна функция может обрабатывать события разных, но однотипных компонентов. В данном примере — это поля ввода цены за кВт электроэнергии

(предыдущее и текущее показания счетчика). Обрабатываемое событие — это событие отпускания клавиши `KeyUp`. Функция обработки события `KeyUp` создается для поля `textBox1`, а затем назначается как функция обработки событий для полей `textBox2` и `textBox3`. Форма программы приведена на рис. 1.2.

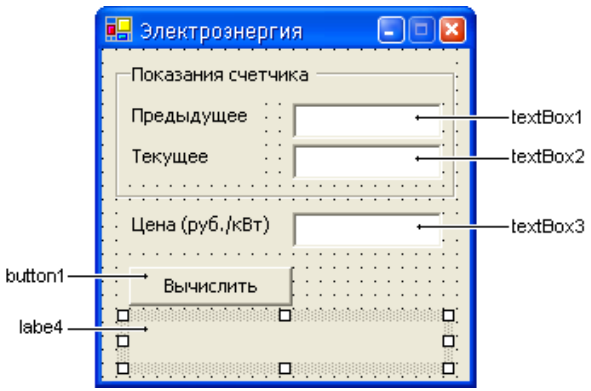

Рис. 1.2. Форма программы **Электроэнергия**

```
// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // блокируем кнопку Вычислить. Она становится доступной
    // только тогда, когда введены данные во все поля
    button1.Enabled = false;
}

// нажатие клавиши в поле Предыдущее показание счетчика
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
```

```
    // если нажата клавиша <Enter> - фокусируемся
    // на поле ввода Текущего показания счетчика
    if (e.KeyChar.Equals((char)13))
        textBox2.Focus();
}

// нажатие клавиши в поле Текущее показание счетчика
private void textBox2_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox2.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
    // если нажата клавиша <Enter> - фокусируемся
    // на поле ввода Цены за кВт электроэнергии
    if (e.KeyChar.Equals((char)13))
        textBox3.Focus();
}

// нажатие клавиши в поле Цена (руб./кВт)
private void textBox3_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox3.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
    // если нажата клавиша <Enter> -
    // фокусируемся на кнопке Вычислить
    if (e.KeyChar.Equals((char)13))
        button1.Focus();
}
```

```
// обработка события отпускания клавиши для полей Цена,
// Предыдущее показание счетчика, Текущее показание счетчика
private void textBox1_KeyUp(object sender,
   System.Windows.Forms.KeyEventArgs e)
{
    // контроль доступности кнопки Вычислить
    if ((textBox1.Text.Length > 0) &&
        (textBox2.Text.Length > 0) &&
        (textBox3.Text.Length > 0))
        button1.Enabled = true;
    else button1.Enabled = false;
}

// щелчок на кнопке Вычислить
private void button1_Click(object sender, System.EventArgs e)
{
    float curr;         // текущее показание счетчика
    float prev;         // предыдущее показание счетчика
    float traf;         // цена за кВт
    float price;    // сумма к оплате

    label4.Text = "";

    try{
        // исходные данные
        prev = Convert.ToSingle(textBox1.Text);
        curr = Convert.ToSingle(textBox2.Text);
        traf = Convert.ToSingle(textBox3.Text);

        if(curr >= prev){
            // вычисляем сумму к оплате
            price = (curr - prev) * traf;

            // вывод результата
            label4.Text = "Сумма к оплате: " +
                          price.ToString("C");
        }
        else{
            MessageBox.Show("Ошибка исходных данных.\n" +
                "Текущее значение показания счетчика\n" +
                "меньше предыдущего.", "Электроэнергия",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
    catch(Exception exc){
        MessageBox.Show("Ошибка исходных данных.\n" +
            "Исходные данные имеют неверный формат.\n" +
            exc.Message, "Электроэнергия",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

Автозаправка

Программа **Автозапрвка** вычисляет количество литров бензина, которое можно купить на заданную сумму. Демонстрирует работу компонента `ComboBox`. Форма программы приведена на рис. 1.3.

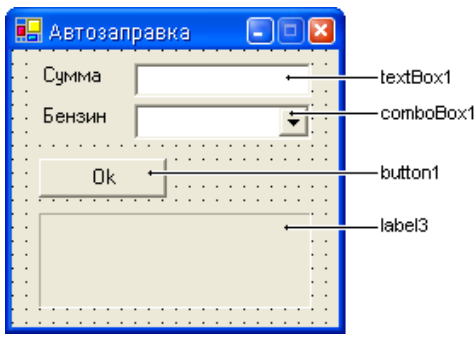

Рис. 1.3. Форма программы **Автозаправка**

```
// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // установим стиль списка Бензин, DropDownList
    comboBox1.DropDownStyle =
        System.Windows.Forms.ComboBoxStyle.DropDownList;
```

```
    // добавим элементы в список Бензин
    comboBox1.Items.Add("92");
    comboBox1.Items.Add("95");
    comboBox1.Items.Add("98");
    comboBox1.Items.Add("ДТ");

    // кнопка Ok недоступна
    button1.Enabled = false;
}

// нажатие клавиши в поле Сумма
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
}

// содержимое поля Сумма изменилось
private void textBox1_TextChanged(object sender,
   System.EventArgs e)
{
    if ((textBox1.Text != ",") && (textBox1.TextLength > 0) &&
        (comboBox1.SelectedIndex != -1))
        button1.Enabled = true;
    else
        if (!button1.Enabled) button1.Enabled = false;
}

// изменился индекс выбранного типа топлива
private void comboBox1_SelectedIndexChanged(object sender,
   System.EventArgs e)
{
    if ((textBox1.Text != ",") && (textBox1.TextLength > 0))
        button1.Enabled = true;
}
```

```
// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    double cl = 17.12F;  // цена за литр
                         // (указанное значение
                         // инициализирует переменную)
    double lt;           // количество литров
    double cash;         // наличные
    double ch;           // сдача

    // цена за литр для выбранного типа топлива
    switch (comboBox1.SelectedIndex)
    {
        case 0: cl = 17.12F;  break;
        case 1: cl = 19.45F;  break;
        case 2: cl = 20.25F;  break;
        case 3: cl = 17.00F;  break;
    }

    // количество литров считается с точностью до 0,1 л
    cash = Convert.ToSingle(textBox1.Text);
    lt   = (double)Decimal.Truncate(
           (Decimal)(cash * 10 / cl)) / 10;
    ch   = cash - lt * cl;

    label3.Text = "Литров: " + lt.ToString("N") +
        "\nСумма: " + cash.ToString("C") +
        "\nСдача: " + ch.ToString("C") +
        "\nЦена за литр: " + cl.ToString("C");
}
```

Кафе

Программа **Кафе** демонстрирует использование компонента CheckBox. Форма программы приведена на рис. 1.4.

```
// переменная  summ объявляется внутри класса WinForm
private double summ; // стоимость заказа

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
```

```
{
    // сделать недоступным переключатель Соус
    checkBox3.Enabled = false;
}

// щелчок на переключателе Сэндвич
private void checkBox1_CheckedChanged(object sender,
   System.EventArgs e)
{
    if (checkBox1.Checked) summ += 54.00;
    else summ -= 54.00;

    label1.Refresh();
}

// щелчок на переключателе Картошка
private void checkBox2_CheckedChanged(object sender,
   System.EventArgs e)
{
    if (checkBox2.Checked) {
        summ += 24.50;

        // сделать доступным переключатель Соус
        if (!checkBox3.Enabled) checkBox3.Enabled = true;
    }
    else{
        summ -= 24.50;

        // сбросить переключатель Соус
        if (checkBox3.Checked) checkBox3.Checked = false;
        // сделать его недоступным
        checkBox3.Enabled = false;
    }

    label1.Refresh();
}

// щелчок на переключателе Соус
private void checkBox3_CheckedChanged(object sender,
   System.EventArgs e)
```

```
{
    if (checkBox3.Checked) summ += 10.50;
    else summ -= 10.50;

    label1.Refresh();
}

// щелчок на переключателе Coca-Cola
private void checkBox4_CheckedChanged(object sender,
   System.EventArgs e)
{
    if (checkBox4.Checked) summ += 18.00;
    else summ -= 18.00;

    label1.Refresh();
}

// обработка события Paint компонента label1
private void label1_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
    // вывести в поле компонента сумму заказа
    label1.Text = summ.ToString("C");
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    if (checkBox1.Checked && checkBox2.Checked &&
        checkBox3.Checked && checkBox4.Checked) {
        // пользователь заказал полный набор,
        // предоставляется скидка 10%
        MessageBox.Show("Вам предоставляется скидка 10%\n" +
            "Стоимость заказа: " + (summ*0.9).ToString("C"),
            "Кафе");
    }
    else {
        if (checkBox1.Checked || checkBox2.Checked ||
            checkBox3.Checked || checkBox4.Checked)
```

```
            MessageBox.Show("Стоимость заказа: " +
                summ.ToString("C"),"Кафе");
        else
            MessageBox.Show("Вы ничего не заказали!",
                "Кафе");
    }
}
```

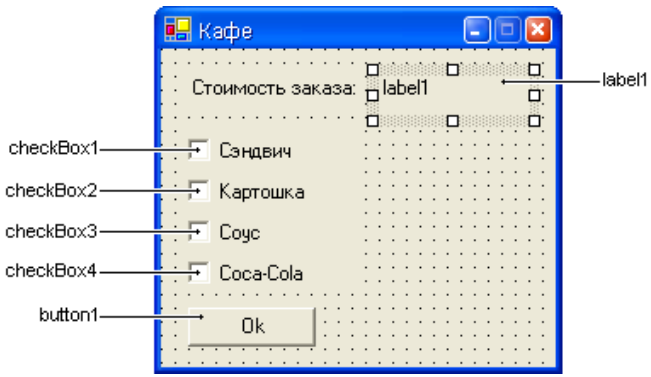

Рис. 1.4. Форма программы **Кафе**

Жалюзи

Программа **Жалюзи** демонстрирует использование компонента `RadioButton`. Форма программы приведена на рис. 1.5.

```
// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // делаем недоступной кнопку Ok
    button1.Enabled = false;

    // материал по умолчанию - алюминий
    radioButton1.Checked = true;
}

// нажатие клавиши в поле Ширина
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
```

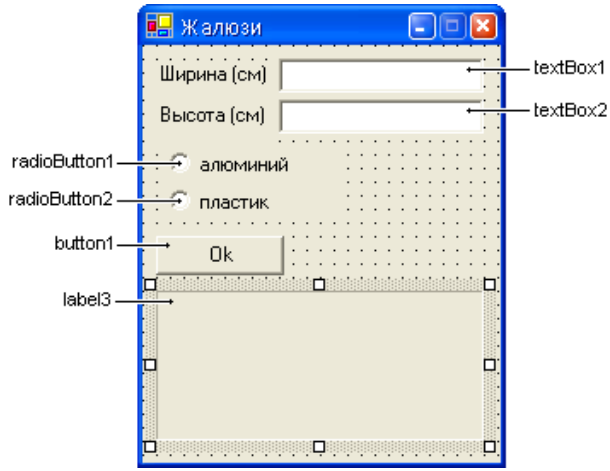

Рис. 1.5. Форма программы **Жалюзи**

```
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar)))
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
            e.Handled = true;
}

// нажатие клавиши в поле Высота
private void textBox2_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar)))
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox2.Text.IndexOf(",") == -1)))
            e.Handled = true;
}

// изменение содержимого полей Ширина и Высота;
// процедура обработки создается для поля Ширина,
```

```
// после чего назначается как процедура обработки
// и для поля Высота
private void textBox1_TextChanged(object sender,
   System.EventArgs e)
{
    // проверка, нужно ли блокировать кнопку Ok
    if ((textBox1.TextLength == 0) ||
        (textBox2.TextLength == 0) ||
        (textBox1.Text == ",") ||
        (textBox2.Text == ","))
        button1.Enabled = false;
    else button1.Enabled = true;
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    Single w, h, s,   // ширина, высота и площадь
           c,         // цена за 1 кв.м.
           cst;       // стоимость

    w = Convert.ToSingle(textBox1.Text);
    h = Convert.ToSingle(textBox2.Text);
    s = w * h / 10000;

    if (radioButton1.Checked)
        c = 3600;     // выбран переключатель "алюминий"
    else
        c = 1800;     // выбран переключатель "пластик"

    cst = s * c;

    if (radioButton1.Checked)
        label3.Text = "Размер:  " + w.ToString("N") +
            " x " + h.ToString("N") + " см\n" +
            "Материал: " + radioButton1.Text +
            "\nСтоимость:  " + cst.ToString("C");
    else
        label3.Text = "Размер:  " + w.ToString("N") +
```

```
            " x " + h.ToString("N") + " см\n" +
            "Материал:  " + radioButton2.Text +
            "\nСтоимость:  " + cst.ToString("C");
}
```

Стеклопакет

Программа **Стеклопакет** позволяет вычислить стоимость окна (стеклопакета). Демонстрирует работу с компонентами `TextBox`, `RadioButton`, `CheckBox`. Форма программы приведена на рис. 1.6.

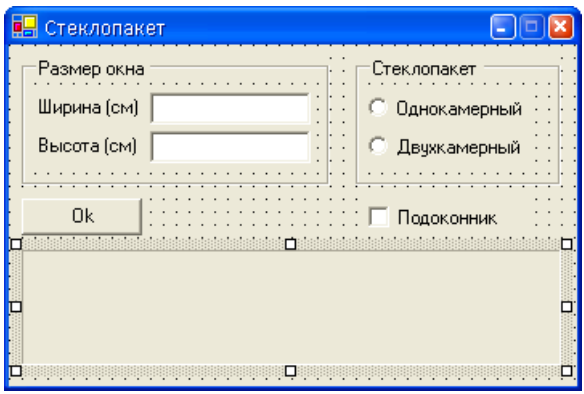

Рис. 1.6. Форма программы **Стеклопакет**

```
// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // делаем недоступной кнопку Ok
    button1.Enabled = false;

    // по умолчанию выбранный
    // тип стеклопакета - однокамерный
    radioButton1.Checked = true;
}

// нажатие клавиши в поле Ширина
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
```

```
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar)))
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
            e.Handled = true;
}

// нажатие клавиши в поле Высота
private void textBox2_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // контроль правильности вводимых данных
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar)))
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox2.Text.IndexOf(",") == -1)))
            e.Handled = true;
}

// изменение типа стеклопакета, Однокамерный/Двухкамерный,
// установка/сброс флажка Подоконник;
// процедура обработки события CheckedChanged создается
// для компонента radioButton1, после чего назначается
// как процедура обработки этого же события
// и для компонентов radioButton2 и checkBox1
private void radioButton1_CheckedChanged(object sender,
   System.EventArgs e)
{
    if (label1.Text != string.Empty)
        label1.Text = string.Empty;
}

// изменение содержимого полей Ширина и Высота
private void textBox1_TextChanged(object sender,
   System.EventArgs e)
{
    // исходные данные изменились, очистим компонент
    // вывода результирующих данных label3 от данных
    // предыдущего расчета, если он производился
```

```
    if (label1.Text != string.Empty)
        label1.Text = string.Empty;

    // проверка, нужно ли блокировать кнопку Ok
    if ((textBox1.TextLength == 0) ||
        (textBox2.TextLength == 0) ||
        (textBox1.Text == ",") ||
        (textBox2.Text == ","))
        button1.Enabled = false;
    else button1.Enabled = true;
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    Single w, h, s,  // ширина, высота и площадь
           c,        // цена за 1 кв.м.
           cst;      // стоимость

    w = Convert.ToSingle(textBox1.Text);
    h = Convert.ToSingle(textBox2.Text);
    s = w * h / 10000;

    if (radioButton1.Checked)
        c = 5000;     // однокамерный стеклопакет
    else
        c = 6000;     // двухкамерный стеклопакет

    cst = s * c;

    // если установлен флажок Подоконник
    if (checkBox1.Checked) cst += 20*w;

    if (radioButton1.Checked)
        label1.Text = "Размер окна:  " + w.ToString("N") +
            " x " + h.ToString("N") + " см\n" +
            "Стеклопакет: " + radioButton1.Text +
            "\nСтоимость:  " + cst.ToString("C");
```

```
    else
        label1.Text = "Размер окна:  " + w.ToString("N") +
            " x " + h.ToString("N") + " см\n" +
            "Стеклопакет:  " + radioButton2.Text +
            "\nСтоимость:  " + cst.ToString("C");
}
```

Калькулятор

Программа **Калькулятор** демонстрирует создание компонентов в коде программы. Кнопки калькулятора — это элементы массива компонентов `Button`. Создание и настройка кнопок осуществляется в конструкторе формы, там же назначаются процедуры обработки события `Click`. Форма и окно программы приведены на рис. 1.7.

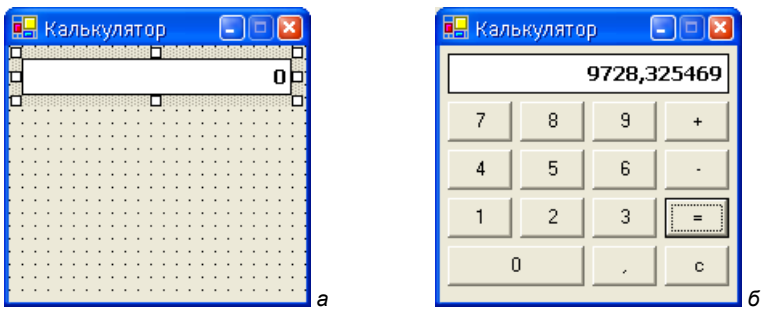

Рис. 1.7. Программа **Калькулятор** (*а* — форма, *б* — окно)

```
private const int
    bw  = 40, bh  = 25,    // ширина и высота кнопок
    bdx = 5,  bdy = 5;     // расстояние между кнопками

// массив цифровых кнопок 0..9 и запятая
private Button[] d  = new Button[11];

// массив операционных кнопок: +, -, =, c
private Button[] op = new Button[4];
```

```
private double ac;  // аккумулятор
private string co;  // текущая операция
private Boolean fd; // fd == true - ждем первую цифру числа,
                    // например, после того как была нажата
                    // кнопка +;
                    // fd == false - ждем ввода следующей
                    // цифры или нажатия кнопки операции;

// конструктор формы
public WinForm()
{
    InitializeComponent();

    int x, y;    // координаты размещения кнопок

    // ширина и высота рабочей области формы
    this.ClientSize =
        new System.Drawing.Size(
            4*bw + 5*bdx, 5*bh + 7*bdy);

    // параметры цифрового индикатора
    label1.SetBounds(bdx, bdy, 4*bw + 3*bdx, bh);
    label1.Text = "0";

    // создание цифровых кнопок
    for(int i = 0; i <= 10; i++) {
        d[i] = new Button();

        // цифровые кнопки 0 - 9
        if (i < 10) {
            d[i].Name = "Button" + Convert.ToString(i);
            d[i].Text = i.ToString();

            if (i != 0) {
                x = ((i-1) % 3)*bw +
                    (((i-1) % 3) + 1)*bdx;
                y = ((int)((9-i)/3) + 1)*bh +
                    ((int)((9-i)/3) + 2)*bdy;
                d[i].SetBounds(x, y, bw, bh);
            }
```

```
            else        // кнопка 0
                d[i].SetBounds(
                    bdx, 4*bh + 5*bdy,
                    2*bw + bdx, bh);
        }
        // кнопка Знак запятой
        else {
            d[i].Name = "ButtonComma";
            d[i].Text = ",";
            d[i].SetBounds(
                2*bw + 3*bdx, 4*bh + 5*bdy,
                bw, bh);
        }

        // назначение процедуры обработки
        // события нажатия кнопки
        this.d[i].Click += new
            System.EventHandler(this.ButtonN_Click);

        // добавление сформированной кнопки на форму
        this.Controls.Add(this.d[i]);
    }

    // создание операционных кнопок
    for(int i = 0; i <= 3; i++) {
        op[i] = new Button();

        // кнопка +
        if (i == 0) {
            op[i].Name = "ButtonPlus";
            op[i].Text = "+";
        }

        // кнопка -
        if (i == 1) {
            op[i].Name = "ButtonMinus";
            op[i].Text = "-";
        }
```

```
        // кнопка =
        if (i == 2) {
            op[i].Name = "ButtonResult";
            op[i].Text = "=";
        }

        // кнопка c
        if (i == 3) {
            op[i].Name = "ButtonClear";
            op[i].Text = "c";
        }

        op[i].SetBounds(
            3*bw + 4*bdx, (i + 1)*bh + (i + 2)*bdy,
            bw, bh);

        // назначение процедуры обработки
        // события нажатия кнопки
        this.op[i].Click += new
            System.EventHandler(this.ButtonOp_Click);

        // добавление сформированной кнопки на форму
        this.Controls.Add(this.op[i]);
    }

    // ждем первую цифру
    fd = true;
    co = "ButtonResult";
}

// нажатие цифровых кнопок или запятой
private void ButtonN_Click(object sender, System.EventArgs e)
{
    // нажатая кнопка
    Button btn_c = (Button)sender;

    if (btn_c.Name != "ButtonComma"){
        // цифры
        if (btn_c.Name != "Button0") {
```

```
            // кнопки 1..9
            if (fd) {
                label1.Text = btn_c.Text;
                fd = false;
            }
            else
                label1.Text += btn_c.Text;
        }
        else {
            // кнопка 0
            if (fd) label1.Text = btn_c.Text;
            if (label1.Text != "0")
                label1.Text += btn_c.Text;
        }
    }
    else
        // запятая
        if (fd) {
            label1.Text = "0,";
            fd = false;
        }
        else
            if (label1.Text.IndexOf(",") == -1)
                label1.Text += btn_c.Text;
}

// нажатие операционных кнопок
private void ButtonOp_Click(object sender, System.EventArgs e)
{
    // нажатая кнопка
    Button btn_c = (Button)sender;

    // число на индикаторе
    double ind_n;

    if (btn_c.Name != "ButtonClear"){
        // кнопки +, -, =
        ind_n = Convert.ToDouble(label1.Text);
```

```
        // выполняем предыдущую операцию
        if (fd == false) {
            if (co.Equals("ButtonPlus"))   ac += ind_n;
            if (co.Equals("ButtonMinus"))  ac -= ind_n;
            if (co.Equals("ButtonResult")) ac  = ind_n;
        }

        if (btn_c.Name == "ButtonPlus")
            co = "ButtonPlus";
        if (btn_c.Name == "ButtonMinus")
            co = "ButtonMinus";
        if (btn_c.Name == "ButtonResult")
            co = "ButtonResult";

        label1.Text = ac.ToString();
    }
    else {
        // кнопка очистить (c)
        ac = 0;
        label1.Text = "0";
        co = "ButtonResult";
    }

    fd = true;
}
```

Просмотр иллюстраций

Программа просмотра иллюстраций ImageViewer демонстрирует использование компонентов `ListBox`, `FolderBrowserDialog` и `PictureBox`. Выбор каталога просматриваемых иллюстраций осуществляется в стандартном окне выбора Windows. Отображение иллюстраций осуществляется компонентом `PictureBox`. Окно программы приведено на рис. 1.8.

```
// для получения прямого доступа к типам DirectoryInfo
// и FileInfo необходимо в директиву using программы
// добавить пространство имен System.IO
```

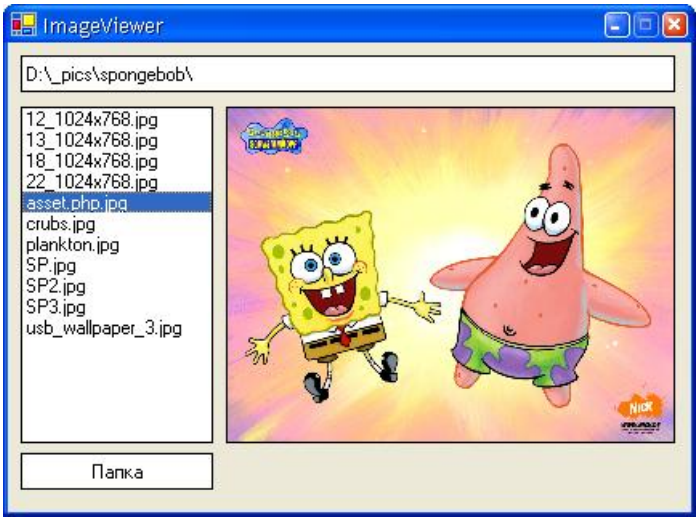

Рис. 1.8. Окно программы просмотра иллюстраций

```
private int pbh,     // исходная высота и
            pbw;     // ширина элемента pictureBox1

// формирует список иллюстраций в ListBox,
// aPath - путь к папке с файлами
private Boolean FillListBox(string aPath) {
    // информация о каталоге
    DirectoryInfo di =
        new DirectoryInfo(aPath);

    // массив информации о файлах
    FileInfo[] fi = di.GetFiles("*.jpg");

    // очищаем ранее полученный список файлов
    listBox1.Items.Clear();

    // добавляем в listBox1 имена jpg-файлов,
    // содержащихся в каталоге aPath
    foreach(FileInfo fc in fi) {
        listBox1.Items.Add(fc.Name);
    }
```

```
    label1.Text = aPath;

    if (fi.Length == 0) return false;
    else {
        // выбираем первый файл из полученного списка
        listBox1.SelectedIndex = 0;
        return true;
    }
}

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // запоминаем исходные значения высоты и ширины
    // компонента pictureBox1
    pbh = pictureBox1.Height;
    pbw = pictureBox1.Width;

    // элементы listBox1 сортируются в
    // алфавитном порядке
    listBox1.Sorted = true;

    // Application.StartupPath возвращает путь к каталогу,
    // из которого была запущена программа;
    // заполняем listBox1 списком иллюстраций
    FillListBox(Application.StartupPath + "\\");
}

// щелчок на кнопке Папка
private void button1_Click(object sender, System.EventArgs e)
{
    // диалоговое окно выбора каталога
    FolderBrowserDialog fb
        = new FolderBrowserDialog();

    fb.Description = "Выберите папку";
    fb.ShowNewFolderButton = false;
```

```
    // отображаем диалоговое окно
    if (fb.ShowDialog() == DialogResult.OK)
        // пользователь выбрал каталог и
        // щелкнул на кнопке OK
        if (!FillListBox(fb.SelectedPath + "\\"))
            // в каталоге нет файлов, выгружаем
            // из pictureBox1 ранее отображаемый файл
            pictureBox1.Image = null;
}

// пользователь выбрал файл щелчком кнопки мыши
// или перемещением по списку при помощи клавиатуры
private void listBox1_SelectedIndexChanged(object sender,
   System.EventArgs e)
{
    double mh, mw;   // коэффициенты масштабирования

    // загружаем изображение в pictureBox1
    pictureBox1.Image =
        new Bitmap(label1.Text +
                   listBox1.SelectedItem.ToString());

    // масштабируем, если нужно
    if ((pictureBox1.Image.Width > pbw) ||
        (pictureBox1.Image.Height > pbh)) {

        pictureBox1.SizeMode =
            PictureBoxSizeMode.StretchImage;

        mh = (double)pbh/(double)pictureBox1.Image.Height;
        mw = (double)pbw/(double)pictureBox1.Image.Width;

        if (mh < mw) {
            // масштабируем по ширине
            pictureBox1.Width  = Convert.ToInt16(
                    pictureBox1.Image.Width * mh);
            pictureBox1.Height = pbh;
        }
```

```
        else {
            // масштабируем по высоте
            pictureBox1.Width  = pbw;
            pictureBox1.Height = Convert.ToInt16(
                    pictureBox1.Image.Height * mw);
        }
    }
    else
        // если предыдущее изображение масштабировалось
        if (pictureBox1.SizeMode ==
                PictureBoxSizeMode.StretchImage)
            pictureBox1.SizeMode =
                PictureBoxSizeMode.AutoSize;
}
```

Секундомер

Программа **Секундомер** демонстрирует использование компонента `Timer`. Форма программы приведены на рис. 1.9.

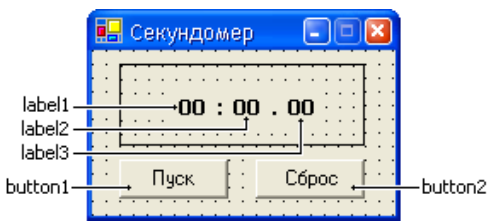

Рис. 1.9. Форма программы **Секундомер**

```
// минуты, секунды, миллисекунды
private int m, s, ms;

// таймер
private System.Windows.Forms.Timer timer1;

// конструктор формы
public WinForm()
{
    InitializeComponent();
```

```
    // обнуление показаний
    m  = 0; s  = 0; ms = 0;

    label1.Text = "00";
    label2.Text = "00";
    label3.Text = "00";

    // период обработки события таймера
    timer1.Interval = 10;
}

// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, System.EventArgs e)
{
    if (timer1.Enabled) {
        // останавливаем таймер
        timer1.Enabled = false;

        button1.Text = "Пуск";
        button2.Enabled = true;
    }
    else {
        // запускаем таймер
        timer1.Enabled = true;
        button1.Text = "Стоп";
        button2.Enabled = false;
    }
}

// щелчок на кнопке Сброс
private void button2_Click(object sender, System.EventArgs e)
{
    // обнуление показаний
    m = 0;  s = 0;  ms = 0;

    label1.Text = "00";
    label2.Text = "00";
    label3.Text = "00";
```

```
    label4.Text = ":";
}

// обработка события таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    if (ms == 99) {
        if (s == 59) {
            if (m == 99) m = 0;
            else m++;
            s = 0;
        }
        else s++;
        ms = 0;
    }
    else ms++;

    // форматирование данных индикатора s:m.ms
    if (m.ToString().Length == 1)
        label1.Text = "0" + m.ToString();
    else
        label1.Text = m.ToString();

    if (s.ToString().Length == 1)
        label2.Text = "0" + s.ToString();
    else
        label2.Text = s.ToString();

    if (ms.ToString().Length == 1)
        label3.Text = "0" + ms.ToString();
    else
        label3.Text = ms.ToString();

    // мигание разделителя минуты/секунды
    if (ms == 1) label4.Text = ":";
    if (ms == 50) label4.Text = "";
}
```

Таймер

Программа **Таймер** демонстрирует использование компонентов `NumericUpDown` и `Timer`. Форма программы приведена на рис. 1.10.

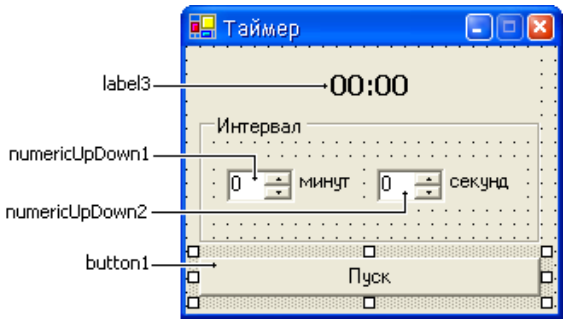

Рис. 1.10. Форма программы **Таймер**

```
private DateTime t1,     // точка отсчета времени
                 t2;     // время срабатывания таймера

// конструктор формы
public WinForm()
{
    InitializeComponent();

    // параметры компонентов numericUpDown
    numericUpDown1.Maximum = 59;
    numericUpDown1.Minimum = 0;

    numericUpDown2.Maximum = 59;
    numericUpDown2.Minimum = 0;

    // кнопка Пуск/Стоп недоступна
    button1.Enabled = false;
}

// изменилось значение поля Value компонента
// numericUpDown1 или numericUpDown2
private void numericUpDown1_ValueChanged(object sender,
   System.EventArgs e)
```

```
{
    if ((numericUpDown1.Value == 0) &&
        (numericUpDown2.Value == 0))
        button1.Enabled = false;
    else
        button1.Enabled = true;
}

// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, System.EventArgs e)
{
    if (!timer1.Enabled) {
        // при помощи t1 инициализируем переменную типа
        // DateTime, содержащую время 00:00:00 (ч:м:с);
        // t2 = t1 + установленное значение таймера
        t1 = new DateTime(DateTime.Now.Year,
                 DateTime.Now.Month, DateTime.Now.Day);
        t2 = t1.AddMinutes((double)numericUpDown1.Value);
        t2 = t2.AddSeconds((double)numericUpDown2.Value);

        groupBox1.Enabled = false;
        button1.Text = "Стоп";

        if (t2.Minute < 9)
            label3.Text = "0" + t2.Minute.ToString() + ":";
        else
            label3.Text = t2.Minute.ToString() + ":";

        if (t2.Second < 9)
            label3.Text += "0" + t2.Second.ToString();
        else
            label3.Text += t2.Second.ToString();

        // сигнал от таймера поступает каждую секунду
        timer1.Interval = 1000;

        // пуск таймера
        timer1.Enabled = true;
    }
```

```
    else {
        // таймер работает, останавливаем
        timer1.Enabled = false;
        button1.Text = "Пуск";
        groupBox1.Enabled = true;
        numericUpDown1.Value = t2.Minute;
        numericUpDown2.Value = t2.Second;
    }
}

// обработка события таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    t2 = t2.AddSeconds(-1);

    if (t2.Minute < 9)
        label3.Text = "0" + t2.Minute.ToString() + ":";
    else
        label3.Text = t2.Minute.ToString() + ":";

    if (t2.Second < 9)
        label3.Text += "0" + t2.Second.ToString();
    else
        label3.Text += t2.Second.ToString();

    if (Equals(t1,t2)) {
        timer1.Enabled = false;
        MessageBox.Show(
            "Заданный интервал времени истек.", "Таймер",
               MessageBoxButtons.OK,
               MessageBoxIcon.Information);

        button1.Text = "Пуск";
        groupBox1.Enabled = true;
        numericUpDown1.Value = 0;
        numericUpDown2.Value = 0;
    }
}
```

Угадай число

Программа **Угадай число** демонстрирует использование компонента `StatusBar`. В панелях состояния приложения отображается количество попыток угадать число и время, затраченное на решение задачи. Форма программы приведена на рис. 1.11.

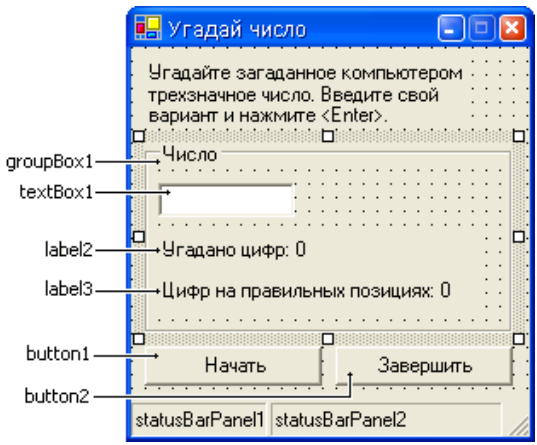

Рис. 1.11. Форма программы **Угадай число**

```
// Для отображения элементов statusBarPannel1 и
// statusBarPannel2 их необходимо добавить в коллекцию Panels
// компонента statusBar1, а также свойству ShowPanels
// присвоить значение true

// количество знаков загаданного числа
private const int CN = 3;

private int t,         // количество сделанных попыток
            s;         // количество прошедших секунд

// загаданное число
private int[] n = new int[CN];

// конструктор формы
public WinForm()
```

```
{
    InitializeComponent();

    statusBar1.ShowPanels = true;
    statusBar1.SizingGrip = false;

    statusBar1.Panels[0].Text = " Попыток: 0";
    statusBar1.Panels[1].Text = " Затрачено времени: 0 сек.";

    groupBox1.Enabled = false;

    timer1.Interval = 1000;
    timer1.Enabled = false;

    textBox1.MaxLength = CN;
}

// щелчок на кнопке Начать
private void button1_Click(object sender, System.EventArgs e)
{
    if (!timer1.Enabled) {
        // генерация числа
        Random rnd = new Random();

        n[0] = rnd.Next(9) + 1;

        for (int i = 1; i < CN; i++)
            n[i] = rnd.Next(10);

        t = 0; s = 0;

        textBox1.Text = string.Empty;
        button1.Text = "Стоп";

        textBox1.Select();

        groupBox1.Enabled = true;
        timer1.Enabled = true;
    }
```

```
    else {
        timer1.Enabled = false;
        groupBox1.Enabled = false;

        textBox1.Text = string.Empty;
        button1.Text = "Начать";

        label2.Text = "Угадано цифр: 0";
        label3.Text = "Цифр на правильных позициях: 0";

        statusBar1.Panels[0].Text = " Попыток: 0";
        statusBar1.Panels[1].Text =
            " Затрачено времени: 0 сек.";
    }
}

// нажатие клавиши в поле ввода числа
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    // учет угаданных цифр
    Boolean[] r = new Boolean[CN];

    int pn = 0,         // цифры на правильных позициях
        rn = 0;         // угаданные цифры

    // допустим ввод только цифр и использование
    // управляющих клавиш (<Enter>, <Backspace> и др.)
    if (Char.IsDigit(e.KeyChar) ||
        Char.IsControl(e.KeyChar)) {

        // была нажата клавиша <Enter>
        if (e.KeyChar.Equals((char)13)) {
            t++;    // счетчик сделанных попыток

            statusBar1.Panels[0].Text =
                " Попыток: " + t.ToString();
```

```
// считаем угаданные цифры и позиции
for(int i = 0; i < textBox1.TextLength; i++) {
    for(int j = 0; j < CN; j++)
        if ((Convert.ToInt16(
            textBox1.Text.Substring(i,1)) == n[j])
            && (!r[j])) {
            rn++;
            r[j] = true;
            break;
        }

    if (Convert.ToInt16(
        textBox1.Text.Substring(i,1)) == n[i])
        pn++;
}

label2.Text = "Угадано цифр: " + rn.ToString();
label3.Text = "Цифр на правильных позициях: " +
              pn.ToString();

// загаданное число угадано
if (pn == CN) {
    timer1.Enabled = false;
    groupBox1.Enabled = false;

    MessageBox.Show("Вы угадали!\n" +
        "Загаданное число: " + textBox1.Text +
        "\nСовершено попыток: " + t.ToString() +
        "\nЗатрачено времени: " + s.ToString() +
        " сек.", "Угадай число",
        MessageBoxButtons.OK,
        MessageBoxIcon. Information);

        textBox1.Text = string.Empty;
        button1.Text = "Начать";

        label2.Text = "Угадано цифр: 0";
        label3.Text =
            "Цифр на правильных позициях: 0";
```

```
                    statusBar1.Panels[0].Text = " Попыток: 0";
                    statusBar1.Panels[1].Text =
                        " Затрачено времени: 0 сек.";
                }
            }
        }
        else e.Handled = true;
}

// щелчок на кнопке Завершить
private void button2_Click(object sender, System.EventArgs e)
{
    this.Close();
}

// обработка события таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    s++;
    statusBar1.Panels[1].Text = " Затрачено времени: " +
        s.ToString() + " сек.";
}
```

Справочная система

Программа демонстрирует использование компонента `HelpProvider` для отображения справочной информации из файла справки на примере программы **Конвертер**. Нажатие клавиши <F1> вызывает файл справки, при этом, если курсор располагается в поле **Цена**, откроется соответствующий раздел справки **Цена**, нажатие клавиши <F1> в поле **Курс** откроет соответствующий раздел **Курс**. Форма программы приведена на рис. 1.12.

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    // назначаем файл справки
    helpProvider1.HelpNamespace = "usd2rub.chm";
```

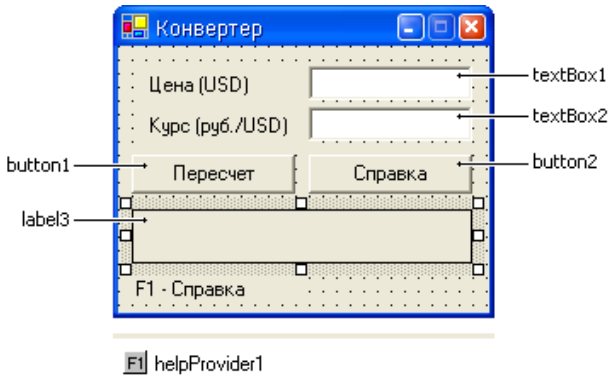

Рис. 1.12. Форма программы **Конвертер**

```
    // задаем раздел справки, который будет выводиться:
    // для основной формы
    helpProvider1.SetHelpKeyword(this, "usd2rub_01.htm");
    helpProvider1.SetHelpNavigator(this, HelpNavigator.Topic);
    helpProvider1.SetShowHelp(this, true);

    // для поля Цена
    helpProvider1.SetHelpKeyword(this.textBox1,
                                 "usd2rub_03.htm");
    helpProvider1.SetHelpNavigator(this.textBox1,
                                   HelpNavigator.Topic);
    helpProvider1.SetShowHelp(this.textBox1, true);

    // для поля Курс
    helpProvider1.SetHelpKeyword(this.textBox2,
                                 "usd2rub_04.htm");
    helpProvider1.SetHelpNavigator(this.textBox2,
                                   HelpNavigator.Topic);
    helpProvider1.SetShowHelp(this.textBox2, true);
}

// нажатие клавиши в поле Цена
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
```

```
{
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox1.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
}

// нажатие клавиши в поле Курс
private void textBox2_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) &&
        !(Char.IsControl(e.KeyChar))) {
        if (!((e.KeyChar.ToString() == ",") &&
            (textBox2.Text.IndexOf(",") == -1)))
            e.Handled = true;
    }
}

// щелчок на кнопке Пересчет
private void button1_Click(object sender, System.EventArgs e)
{
    double k,       // курс (отношение рубля к доллару)
           usd,     // цена в долларах
           rub;     // цена в рублях

    label3.Text = string.Empty;

    try{
        // исходные данные
        usd = System.Convert.ToDouble(textBox1.Text);
        k   = System.Convert.ToDouble(textBox2.Text);

        // пересчет цены из долларов в рубли
        rub = usd * k;

        // вывод результата
        label3.Text = usd.ToString("N") +
            " USD = " + rub.ToString("C");
    }
```

```
    catch{
        if ((textBox1.Text == "") || (textBox2.Text == "")){
            MessageBox.Show("Ошибка исходных данных.\n" +
                "Необходимо ввести данные в оба поля.",
                "Конвертер", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
        else
            MessageBox.Show("Ошибка исходных данных.\n" +
                "Неверный формат данных в одном из полей.",
                "Конвертер", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
    }
}

// щелчок на кнопке Справка
private void button2_Click(object sender, System.EventArgs e)
{
    // вызываем файл справки, раздел Конвертер
    Help.ShowHelp(this, helpProvider1.HelpNamespace,
                  "usd2rub_01.htm");
}
```

Работа с несколькими формами

Программа демонстрирует возможность работы с несколькими формами. Форма **Введите сообщение** (рис. 1.13) позволяет передать сообщение главной форме программы (рис. 1.14). Окно **О программе** приведено на рис. 1.15.

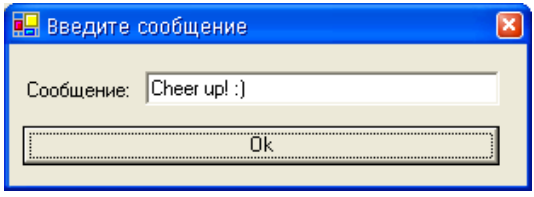

Рис. 1.13. Окно **Введите сообщение**

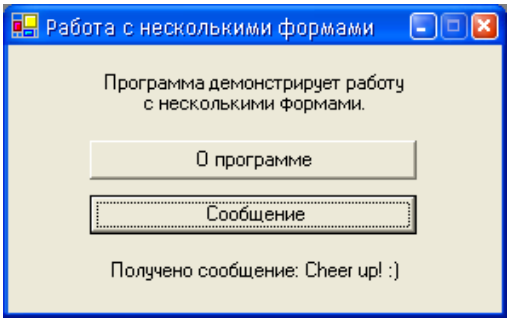

Рис. 1.14. Отображение полученного сообщения в главном окне программы

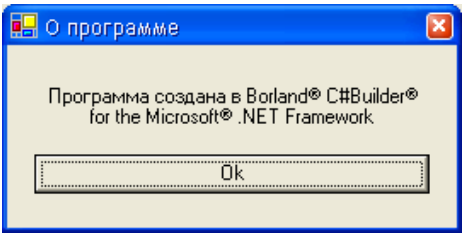

Рис. 1.15. Окно **О программе**

```
// главная форма программы (WinForm):
// щелчок на кнопке О программе
private void button1_Click(object sender, System.EventArgs e)
{
    WinFormAbout fa = new WinFormAbout();
    fa.ShowDialog();
}

// щелчок на кнопке Сообщение
private void button2_Click(object sender, System.EventArgs e)
{
    WinFormMessage fm = new WinFormMessage();
    fm.ShowDialog();
    label2.Text = "Получено сообщение: " + fm.msg;
}
```

```
// форма Введите сообщение (WinFormMessage):
// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    // переменная msg объявлена как
    // public string msg = string.Enpty;
    // внутри класса WinFormMessage;
    // записываем в нее сообщение
    this.msg = textBox1.Text;

    this.Close();
}


// форма О программе (WinFormAbout):
// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    this.Close();
}
```

ФАЙЛЫ

В этом разделе приведены программы, демонстрирующие выполнение операций с файлами.

Погода

Программа **Погода** добавляет в базу данных, представляющую собой текстовый файл, сведения о температуре воздуха. Каждая строка данных содержит дату и значение температуры. Если файла данных нет, программа его создает. Кнопка **Добавить** доступна только в том случае, если заполнено поле **Температура**. Окно программы приведено на рис. 1.16.

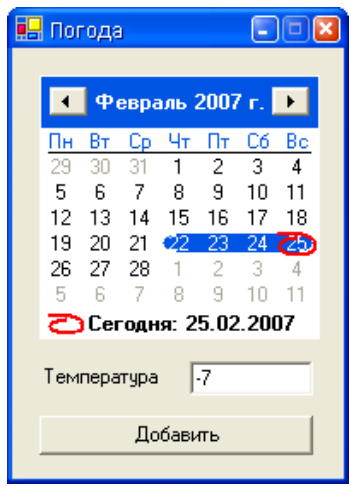

Рис. 1.16. Окно программы **Погода**

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    // устанавливаем текущую дату
    monthCalendar1.TodayDate = System.DateTime.Now;
```

```
    // выбранная на календаре дата - текущая
    monthCalendar1.SelectionRange =
        new SelectionRange(monthCalendar1.TodayDate,
            monthCalendar1.TodayDate);

    button1.Enabled = false;
}

// нажатие клавиши в поле Температура
private void textBox1_KeyPress(object sender,
   System.Windows.Forms.KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) &&
        !Char.IsControl(e.KeyChar)) {
        if (!(e.KeyChar.ToString() == "," &&
              textBox1.Text.IndexOf(",") == -1) &&
            !(e.KeyChar.ToString() == "-" &&
              textBox1.Text.Length == 0))
            e.Handled = true;
    }
}

// содержимое поля Температура изменилось
private void textBox1_TextChanged(object sender,
   System.EventArgs e)
{
    if (textBox1.Text != "" &&
        textBox1.Text != "-" &&
        textBox1.Text != "," &&
        textBox1.Text != "-,")
        button1.Enabled = true;
    else
        button1.Enabled = false;
}

// щелчок на кнопке Добавить
private void button1_Click(object sender, System.EventArgs e)
{
    // введенное показание температуры
    double tp;
```

```
// проверяем правильность вводимых значений
try {
    tp = Convert.ToDouble(textBox1.Text);
}
catch {
    MessageBox.Show(
        "Введенное значение имеет неверный формат.",
        "Погода", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    return;
}

// получим информацию о файле "meteo.txt"
System.IO.FileInfo fi =
    new System.IO.FileInfo(
        Application.StartupPath + "\\meteo.txt");

// поток для записи
System.IO.StreamWriter sw;

// если файл записей показаний "meteo.txt"
// не найден, создаем его
if (fi.Exists)
    sw = fi.AppendText();
else
    sw = fi.CreateText();

// если выбран диапазон дат, записываем введенное
// показание для каждой из дат диапазона
DateTime dt = monthCalendar1.SelectionStart;
do
{
    sw.WriteLine(dt.ToShortDateString() +
                 " " + tp.ToString("N"));
    dt = dt.AddDays(1);
} while(DateTime.Compare(
    monthCalendar1.SelectionEnd, dt) >= 0);
```

```
    // закрываем поток
    sw.Close();

    textBox1.Text = string.Empty;
}
```

Средняя температура

Программа **Средняя температура** демонстрирует чтение данных из текстового файла. Данные считываются из файла, сформированного программой **Погода**. После того как данные считаны, для каждого выбранного месяца можно вычислить среднюю температуру. Окно программы приведено на рис. 1.17.

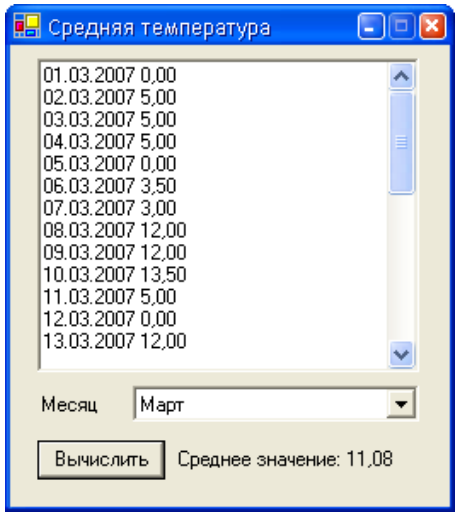

Рис. 1.17. Окно программы **Средняя температура**

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    textBox1.ReadOnly  = true;
    textBox1.BackColor = Color.White;
    textBox1.ScrollBars = ScrollBars.Both;
```

```
label2.Text = string.Empty;

// список месяцев
comboBox1.DropDownStyle =
    ComboBoxStyle.DropDownList;

comboBox1.Items.Add("Январь");
comboBox1.Items.Add("Февраль");
comboBox1.Items.Add("Март");
comboBox1.Items.Add("Апрель");
comboBox1.Items.Add("Май");
comboBox1.Items.Add("Июнь");
comboBox1.Items.Add("Июль");
comboBox1.Items.Add("Август");
comboBox1.Items.Add("Сентябрь");
comboBox1.Items.Add("Октябрь");
comboBox1.Items.Add("Ноябрь");
comboBox1.Items.Add("Декабрь");

comboBox1.SelectedIndex = 0;
comboBox1.Select();

System.IO.StreamReader sr;
try{
    // считываем данные из файла
    sr = new System.IO.StreamReader(
        Application.StartupPath + "\\meteo.txt",
        System.Text.Encoding.GetEncoding(1251));

    textBox1.Text = sr.ReadToEnd();

    sr.Close();
}
catch(Exception exc) {
    MessageBox.Show("Файл исходных данных не найден.\n" +
        exc.ToString(), "Температура",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
```

```
        button1.Enabled = false;
    }
}

// щелчок на кнопке Вычислить
private void button1_Click(object sender, System.EventArgs e)
{
    // номер выбранного месяца
    int nm = comboBox1.SelectedIndex + 1;

    // количество записей
    int n = 0;

    // суммарное значение показаний
    double s = 0;

    for(int i=0; i<textBox1.Lines.Length; i++){
        // смотрим, к какому месяцу относится
        // значение показания температуры
        if (textBox1.Lines[i].Length > 0)
            if (Convert.ToInt16(
                textBox1.Lines[i].Substring(3,2)) == nm){
                n++;
                s += Convert.ToDouble(
                    textBox1.Lines[i].Substring(
                    textBox1.Lines[i].IndexOf(" ")));
            }
    }

    // вывод результата
    if (n == 0) {
        MessageBox.Show("Файл не содержит данных\n" +
            "о температуре за " + comboBox1.Text,
            "Температура",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);

            label2.Text = string.Empty;
    }
```

```
    else
        label2.Text = "Среднее значение: " +
            ((double)s/(double)n).ToString("N");
}
```

Редактор текста

Программа **MEdit** (редактор текста) демонстрирует использование компонентов `MainMenu`, `OpenDialog`, `SaveDialog`, `FontDialog`, `PrintDocument`, а также отображение информации о программе в отдельном окне. Главная форма программы приведена на рис. 1.18. Информация о программе отображается в окне **О программе**, которое появляется в результате выбора соответствующей команды в пункте меню **Справка**. Форма окна приведена на рис. 1.19.

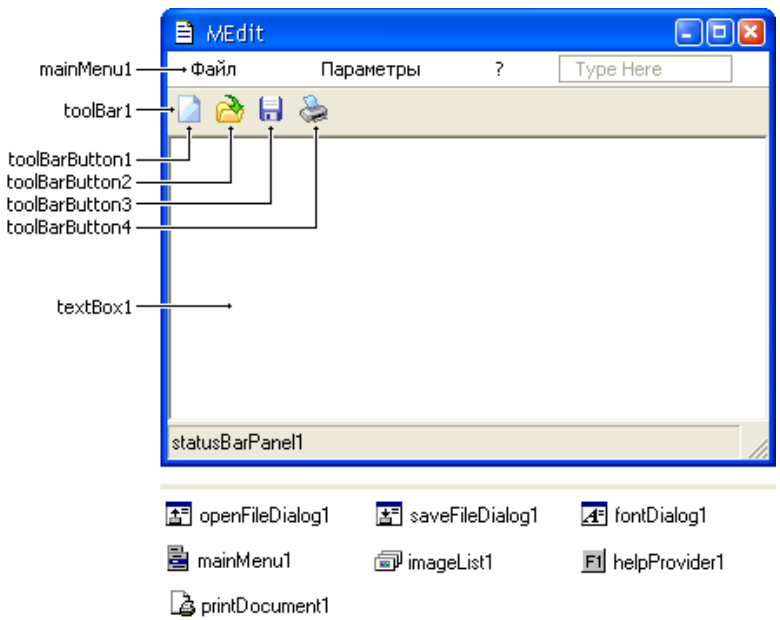

Рис. 1.18. Форма программы **MEdit**

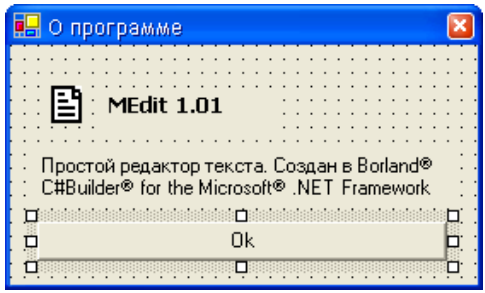

Рис. 1.19. Форма окна **О программе**

```
// полное имя текстового документа (путь к нему)
private string fn = string.Empty;

// открывает документ
private void OpenDocument() {
    openFileDialog1.FileName = string.Empty;

    // отобразить диалог Открыть
    if (openFileDialog1.ShowDialog() ==
                        DialogResult.OK) {
        fn = openFileDialog1.FileName;

        // отобразить имя файла в заголовке окна
        this.Text = fn;

        try{
            // считываем данные из файла
            System.IO.StreamReader sr =
                new System.IO.StreamReader(fn);

            textBox1.Text = sr.ReadToEnd();
            textBox1.SelectionStart = textBox1.TextLength;

            sr.Close();
        }
        catch(Exception exc) {
            MessageBox.Show("Ошибка чтения файла.\n" +
                exc.ToString(), "MEdit",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}

// сохраняет документ
private void SaveDocument() {
    // если имя файла не задано,
    // выводим диалог Сохранить документ
    if (fn == string.Empty) {
        if (saveFileDialog1.ShowDialog() ==
                            DialogResult.OK) {
            fn = saveFileDialog1.FileName;

            // отобразить имя файла в заголовке окна
            this.Text = fn;
        }
    }

    if (fn != string.Empty)
        try {
            // получим информацию о файле fn
            System.IO.FileInfo fi =
                new System.IO.FileInfo(fn);

            // поток для записи (перезаписываем файл)
            System.IO.StreamWriter sw = fi.CreateText();

            // записываем данные
            sw.Write(textBox1.Text);

            // закрываем поток
            sw.Close();
        }
        catch(Exception exc) {
            MessageBox.Show("Ошибка доступа к файлу.\n" +
                exc.ToString(), "MEdit",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
}

// конструктор формы
public WinForm()
{
    InitializeComponent();

    textBox1.ScrollBars = ScrollBars.Vertical;
    textBox1.Text = string.Empty;

    this.Text = "MEdit - Новый документ";

    // отображение галочек панели инструментов и
    // строки состояния в главном меню программы
    menuItem12.Checked = true;
    menuItem13.Checked = true;

    statusBar1.Panels[0].Text = string.Empty;

    // назначаем файл справки
    helpProvider1.HelpNamespace = "m_edit.chm";

    // настройка компонента openDialog1
    openFileDialog1.DefaultExt = "txt";
    openFileDialog1.Filter = "текстовый документ|*.txt";
    openFileDialog1.Title = "Открыть документ";
    openFileDialog1.Multiselect = false;

    // настройка компонента saveDialog1
    saveFileDialog1.DefaultExt = "txt";
    saveFileDialog1.Filter = "текстовый документ|*.txt";
    saveFileDialog1.Title = "Сохранить документ";
}

// печать страницы документа (видимой области)
private void printDocument1_PrintPage(object sender,
   System.Drawing.Printing.PrintPageEventArgs e)
```

```
{
    Rectangle r;    // область вывода

    r = new Rectangle(10, 10,
        textBox1.Width, textBox1.Height);

    if (r.Height + 20 > e.PageBounds.Height)
        r.Height = e.PageBounds.Height - 20;
    if (r.Width  + 20 > e.PageBounds.Width)
        r.Width  = e.PageBounds.Width  - 20;

    // вывести текст на печать
    e.Graphics.DrawString(
        textBox1.Text,
        textBox1.Font,
        Brushes.Black,
        r);
}

// щелчок на кнопке панели инструментов toolBar1
private void toolBar1_ButtonClick(object sender,
   System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
    switch (toolBar1.Buttons.IndexOf(e.Button))
    {
        case 0:         // новый документ
            fn = string.Empty;
            textBox1.Text = string.Empty;
            this.Text = "MEdit - Новый документ";
            break;
        case 1:         // открыть документ
            this.OpenDocument();
            break;
        case 2:         // сохранить документ
            this.SaveDocument();
            break;
        case 3:         // печать страницы документа
            printDocument1.Print();
            break;
    }
}
```

```
// указатель мыши на пункте меню Файл
private void menuItem1_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Файл/Новый
private void menuItem2_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Создать новый документ";
}

// команда Файл/Новый
private void menuItem2_Click(object sender,
             System.EventArgs e)
{
    fn = string.Empty;
    textBox1.Text = "";
    this.Text = "MEdit - Новый документ";
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Файл/Открыть
private void menuItem3_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Открыть существующий
   документ";
}

// команда Файл/Открыть
private void menuItem3_Click(object sender,
             System.EventArgs e)
{
    this.OpenDocument();
    statusBar1.Panels[0].Text = string.Empty;
}
```

```
// указатель мыши на пункте меню Файл/Сохранить
private void menuItem4_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text =
        "Сохранить набранный (отредактированный) документ";
}

// команда Файл/Сохранить
private void menuItem4_Click(object sender,
             System.EventArgs e)
{
    this.SaveDocument();
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Файл/Печать
private void menuItem6_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Распечатать документ";
}

// команда Файл/Печать
private void menuItem6_Click(object sender,
             System.EventArgs e)
{
    printDocument1.Print();
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Файл/Выход
private void menuItem8_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Завершить работу";
}

// команда Файл/Выход
private void menuItem8_Click(object sender, System.EventArgs e)
```

```
{
    this.Close();
}

// указатель мыши на пункте меню Параметры
private void menuItem9_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Параметры/Шрифт
private void menuItem10_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Выбрать шрифт";
}

// команда Параметры/Шрифт
private void menuItem10_Click(object sender,
             System.EventArgs e)
{
    if ( fontDialog1.ShowDialog() == DialogResult.OK )
        textBox1.Font = fontDialog1.Font;
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Параметры/Панель инструментов
private void menuItem12_Select(object sender,
             System.EventArgs e)
{
    if (menuItem12.Checked)
        statusBar1.Panels[0].Text =
            "Скрыть панель инструментов";
    else
        statusBar1.Panels[0].Text =
            "Отобразить панель инструментов";
}
```

```
// команда Параметры/Панель инструментов
private void menuItem12_Click(object sender,
             System.EventArgs e)
{
    // скрыть/показать панель инструментов
    toolBar1.Visible = !toolBar1.Visible;

    // установить/сбросить флажок рядом с командой
    menuItem12.Checked = !menuItem12.Checked;

    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню Параметры/Строка состояния
private void menuItem13_Select(object sender,
             System.EventArgs e)
{
    if (menuItem13.Checked)
        statusBar1.Panels[0].Text =
            "Скрыть строку состояния";
    else
        statusBar1.Panels[0].Text =
            "Отобразить строку состояния";
}

// команда Параметры/Строка состояния
private void menuItem13_Click(object sender,
             System.EventArgs e)
{
    // скрыть/показать строку состояния
    statusBar1.Visible = !statusBar1.Visible;

    // установить/сбросить флажок рядом с командой
    menuItem13.Checked = !menuItem13.Checked;

    statusBar1.Panels[0].Text = string.Empty;
}
```

```
// указатель мыши на пункте меню ?
private void menuItem14_Select(object sender,
                System.EventArgs e)
{
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню ?/Справка
private void menuItem15_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "Справка";
}

// команда ?/Справка
private void menuItem15_Click(object sender,
             System.EventArgs e)
{
    Help.ShowHelp(this,
        this.helpProvider1.HelpNamespace);
    statusBar1.Panels[0].Text = string.Empty;
}

// указатель мыши на пункте меню ?/О программе
private void menuItem16_Select(object sender,
             System.EventArgs e)
{
    statusBar1.Panels[0].Text = "О программе";
}

// команда ?/О программе
private void menuItem16_Click(object sender,
             System.EventArgs e)
{
    // отображаем форму О программе
    WinFormAbout af = new WinFormAbout();
    af.ShowDialog();

    statusBar1.Panels[0].Text = string.Empty;
}
```

ГРАФИКА

В этом разделе собраны программы, демонстрирующие работу с графикой.

Общие замечания

- Вывод графики осуществляется на графическую *поверхность* объекта (графическая поверхность формы, компонента или заданная вручную поверхность).
- Для того чтобы на поверхности объекта появился графический элемент (линия, окружность, прямоугольник и т. д.) или изображение, необходимо применить к `графической поверхности` соответствующий метод.
- Методы `DrawLine`, `DrawRectangle`, `DrawPie` и т. д. выполняют прорисовку контура объекта. Заполнение внутренней области объекта осуществляется при помощи методов `FillRectangle`, `FillPie` и т. д.
- Цвет закраски внутренних областей геометрических фигур, вычерчиваемых методами `FillRectangle`, `FillPie`, определяются параметрами вызываемых методов.
- Характеристики шрифта текста, выводимого методом `DrawString`, определяются параметрами вызова метода. Если они не указаны, используются характеристики шрифта (тип, цвет, размер), указанные для основной формы программы.
- Основную работу по выводу графики на поверхность формы выполняет функция обработки события `Paint`.

Приветствие

Программа демонстрирует вывод текста на поверхность формы. Окно программы приведено на рис. 1.20.

```
// обработка события Paint
private void WinForm_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
    string st;
```

```
    int x, y;
    SizeF aSize;

    Font hFont = new Font("Tahoma", 8, FontStyle.Bold);
    Font tFont = new Font("Tahoma", 8, FontStyle.Underline);

    st = "Borland C#Builder for the Microsoft.NET Framework";

    // определяем размер области, которую будет
    // занимает строка st, написанная шрифтом hFont
    aSize = e.Graphics.MeasureString(st, hFont);

    // координаты верхнего левого угла области,
    // в которую будет выводиться текст
    x = (int)(this.ClientSize.Width - aSize.Width)/2;
    y = 20;

    // выводим строку st на форму
    e.Graphics.DrawString(st, hFont, Brushes.Black, x,y);

    aSize = e.Graphics.MeasureString(st,tFont);

    x = (int)(this.ClientSize.Width - aSize.Width)/2;
    y += (int)aSize.Height;

    e.Graphics.DrawString(st, tFont, Brushes.Black, x,y);
}

// обработка события изменения размера формы
private void WinForm_Resize(object sender, System.EventArgs e)
{
    this.Refresh();
}
```

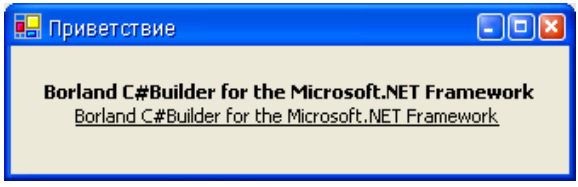

Рис. 1.20. Окно программы **Приветствие**

Диаграмма

В окне программы **Диаграмма** отображается диаграмма, иллюстрирующая изменение курса доллара. Программа спроектирована таким образом, что независимо от размера окна, диаграмма располагается в центре окна и занимает большую его часть, даже если пользователь изменит размер окна. Программа демонстрирует работу с массивами, чтение данных из файла, работу с методами `DrawString`, `DrawRectangle`, `FillRectangle`. Окно программы приведено на рис. 1.21.

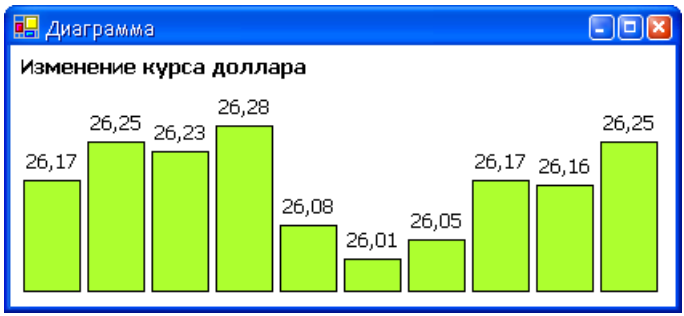

Рис. 1.21. Окно программы **Диаграмма**

```
// массив записей курса доллара
private double[] cv;

// поток для чтения
public System.IO.StreamReader sr;

// процедура рисует диаграмму; имеет такие же параметры,
// что и процедура обработки события Paint
private void drawDiagram(object sender,PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт заголовка диаграммы
    Font hfont = new Font("Tahoma",10,FontStyle.Bold);
```

```
// шрифт подписи значений диаграммы
Font mfont = new Font("Tahoma",10);

// выводим заголовок
g.DrawString("Изменение курса доллара",
    hfont, System.Drawing.Brushes.Black,
    5, 5);

// ширина столбца диаграммы
int sw = (int)
    ((this.ClientSize.Width - 15) / cv.Length) - 5;

// высота столбца диаграммы
int sh;

// максимальное и минимальное значения массива cv
double cvmax = cv[0];
double cvmin = cv[0];

for (int i = 1; i < cv.Length; i++) {
    if (cv[i] > cvmax) cvmax = cv[i];
    if (cv[i] < cvmin) cvmin = cv[i];
}

// рисуем столбцы диаграммы
int x,y;
for(int i=0; i < cv.Length; i++) {
    // высота столбца диаграммы
    sh = (int)((this.ClientSize.Height - 80)*
         (cv[i] - cvmin)/(cvmax - cvmin)) + 20;

    // координаты верхнего правого угла
    // столбца диаграммы
    x = 8 + i*(sw + 5);
    y = this.ClientSize.Height - 10 - sh;

    // столбец диаграммы
    g.FillRectangle(System.Drawing.Brushes.GreenYellow,
                                    x, y, sw, sh);
```

```
        // граница
        g.DrawRectangle(System.Drawing.Pens.Black,
                                        x, y, sw, sh);

        // подпись численного значения
        g.DrawString(cv[i].ToString(),
           mfont, System.Drawing.Brushes.Black,
           x, y-20);
    }
}

// конструктор формы
public WinForm()
{
    InitializeComponent();

    try{
        // создаем поток для чтения и считываем данные;
        // Application.StartupPath возвращает путь
        // к каталогу, из которого была запущена программа
        sr = new System.IO.StreamReader(
            Application.StartupPath + "\\dinput.txt");

        // считываем данные о количестве записей
        // и инициализируем массив
        cv = new double[Convert.ToInt16(sr.ReadLine())];

        // считываем записи курса доллара
        int i=0;
        string t  = sr.ReadLine();
        while (t != null){
            // записываем считанное число в массив
            cv[i++] = Convert.ToDouble(t);

            // считываем следующее значение
            t = sr.ReadLine();
        }

        // закрываем поток
        sr.Close();
```

```
        // если записей нет
        if (cv.Length == 0)
            MessageBox.Show("Файл исходных данных пуст.",
                "Диаграмма", MessageBoxButtons.OK,
                MessageBoxIcon.Error);

        // если исходный файл имеет неверное
        // количество записей
        if (i < cv.Length)
            MessageBox.Show(
                "Файл исходных данных поврежден\n" +
                "или имеет неверное количество записей.",
                "Диаграмма", MessageBoxButtons.OK,
                MessageBoxIcon.Error);

        // если все данные считались и
        // никаких исключений не возникло
        if (cv.Length != 0 && cv.Length == i) {
            // процедура прорисовки диаграммы drawDiagram
            // будет вызываться каждый раз, когда будет
            // происходит событие Paint
            this.Paint += new PaintEventHandler(drawDiagram);
        }
    }
    // обработка исключений:
    // файл исходных данных не найден
    catch(System.IO.FileNotFoundException exc){
        MessageBox.Show("Файл исходных данных не найден.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
            "Диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    // ошибка формата исходных данных
    catch(FormatException exc){
        MessageBox.Show("Ошибка формата исходных данных.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
```

```
            "Диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    // прочие исключения
    catch(Exception exc){
        MessageBox.Show(exc.Message + "\n" +
            exc.GetType().ToString(),
            "Диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }

    // задаем цвет формы
    this.BackColor = System.Drawing.Color.White;
}

// изменение размера окна программы
private void WinForm_SizeChanged(object sender,
   System.EventArgs e)
{
    // метод Refresh перерисовывает форму полностью
    this.Refresh();
}
```

График

В окне программы **График** отображается график изменения курса доллара. Следует обратить внимание на то, что на графике отображаются не значения курса, а изменение между минимальным и максимальным значениями ряда, что делает график наглядным, даже при незначительном разбросе значений. Окно программы приведено на рис. 1.22.

```
// массив записей курса доллара
private double[] cv;

// поток для чтения
public System.IO.StreamReader sr;
```

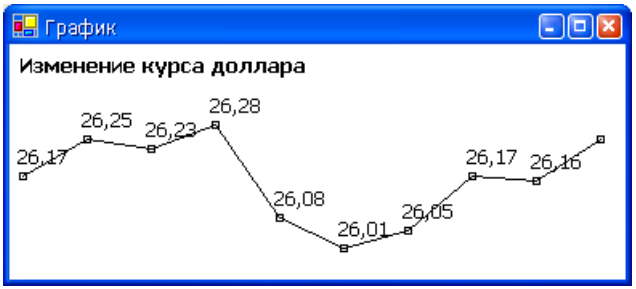

Рис. 1.22. Окно программы **График**

```
// процедура рисует график;
// имеет такие же параметры, что и
// процедура обработки события Paint
private void drawDiagram(object sender,PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт выводимого текста
    Font hfont = new Font("Tahoma",10,FontStyle.Bold);

    // шрифт подписи значений графика
    Font mfont = new Font("Tahoma",10);

    // выводим заголовок
    g.DrawString("Изменение курса доллара",
        hfont, System.Drawing.Brushes.Black,
        5, 5);

    // расстояние между точками графика
    int sw = (int)((this.ClientSize.Width - 20) /
                   (cv.Length -1));

    // максимальное и минимальное значения массива cv
    double cvmax = cv[0];
    double cvmin = cv[0];

    for (int i = 1; i < cv.Length; i++) {
        if (cv[i] > cvmax) cvmax = cv[i];
```

```
        if (cv[i] < cvmin) cvmin = cv[i];
    }

    // рисуем график
    int x1, y1, x2, y2;

    // координата первой точки
    x1 = 8;
    y1 = this.ClientSize.Height - 20 -
        (int)((this.ClientSize.Height - 70)*
        (cv[0] - cvmin)/(cvmax - cvmin));

    // метка первой точки
    g.DrawRectangle(System.Drawing.Pens.Black,
        x1-2, y1-2, 4, 4);

    for(int i=1; i < cv.Length; i++) {
        // координаты следующей точки
        x2 = 8 + i*sw;
        y2 = this.ClientSize.Height - 20 -
            (int)((this.ClientSize.Height - 70)*
            (cv[i] - cvmin)/(cvmax - cvmin));

        // метка следующей точки
        g.DrawRectangle(System.Drawing.Pens.Black,
              x2-2, y2-2, 4, 4);

        // линия
        g.DrawLine(System.Drawing.Pens.Black,
            x1, y1, x2, y2);

        // подпись численного значения
        g.DrawString(Convert.ToString(cv[i-1]),
            mfont, System.Drawing.Brushes.Black,
               x1-5, y1-20);

        x1 = x2;
        y1 = y2;
    }
}
```

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    try{
        // создаем поток для чтения и считываем данные;
        // Application.StartupPath возвращает путь
        // к каталогу, из которого была запущена программа
        sr = new System.IO.StreamReader(
            Application.StartupPath + "\\dinput.txt");

        // считываем данные о количестве записей
        // и инициализируем массив
        cv = new double[Convert.ToInt16(sr.ReadLine())];

        // считываем записи курса доллара
        int i=0;
        string t  = sr.ReadLine();
        while (t != null){
            // записываем считанное число в массив
            cv[i++] = Convert.ToDouble(t);

            // считываем следующее значение
            t = sr.ReadLine();
        }

        // закрываем поток
        sr.Close();

        // если записей нет
        if (cv.Length == 0)
            MessageBox.Show("Файл исходных данных пуст.",
                "График", MessageBoxButtons.OK,
                MessageBoxIcon.Error);

        // если исходный файл имеет
        // неверное количество записей
        if (i < cv.Length)
```

```
            MessageBox.Show(
                "Файл исходных данных поврежден\n"+
                "или имеет неверное количество записей.",
                "График", MessageBoxButtons.OK,
                MessageBoxIcon.Error);

        // если все данные считались и
        // никаких исключений не возникло
        if (cv.Length != 0 && cv.Length == i) {
            // процедура прорисовки диаграммы drawDiagram
            // будет вызываться каждый раз, когда будет
            // происходить событие Paint
            this.Paint += new PaintEventHandler(drawDiagram);
        }
    }
    // обработка исключений:
    // файл исходных данных не найден
    catch(System.IO.FileNotFoundException exc){
        MessageBox.Show("Файл исходных данных не найден.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
            "График",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    // ошибка формата исходных данных
    catch(FormatException exc){
        MessageBox.Show("Ошибка формата исходных данных.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
            "График",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    // прочие исключения
    catch(Exception exc){
        MessageBox.Show(exc.Message + "\n" +
            exc.GetType().ToString(),
            "График",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }

        // задаем цвет формы
        this.BackColor = System.Drawing.Color.White;
}

// изменение размера окна программы
private void WinForm_SizeChanged(object sender,
   System.EventArgs e)
{
        // метод Refresh перерисовывает форму полностью
        this.Refresh();
}
```

Круговая диаграмма

В окне программы **Круговая диаграмма** отображаются результаты опроса в процентном соотношении. Исходные данные считываются из файла. Окно программы приведено на рис. 1.23.

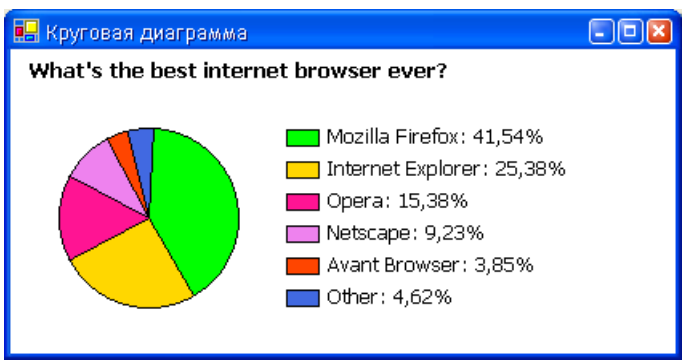

Рис. 1.23. Окно программы **Круговая диаграмма**

```
// заголовок диаграммы
private string hl;

// массив численных значений диаграммы
private double[] cv;
```

```
// количество элементов массива численных значений
private int ne = 0;

// сумма элементов массива значений диаграммы
private double se = 0;

// массив подписей легенды диаграммы
private string[] lv;

// поток для чтения
public System.IO.StreamReader sr;

// процедура рисует круговую диаграмму;
// имеет такие же параметры, что и
// процедура обработки события Paint
private void drawDiagram(object sender,PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт заголовка
    Font hfont = new Font("Tahoma",10,FontStyle.Bold);

    // шрифт легенды
    Font mfont = new Font("Tahoma",10);

    // выводим заголовок
    g.DrawString(hl,
        hfont, System.Drawing.Brushes.Black,
        10, 5);

    // диаметр сектора круговой диаграммы
    int sd = (int)(this.ClientSize.Height*4/5 - 40);

    int x = 30;
    int y = (int)((this.ClientSize.Height - sd)/2 + 10);

    // координаты верхнего левого угла
    // месторасположения легенды
    int lx = 60 + sd;
    int ly = y + (int)((sd - ne*20 + 10)/2);
```

```
// начальная точка дуги сектора
int sta = -90;

// длина дуги сектора
int swa;

// кисть для заливки сектора диаграммы
Brush fBrush = Brushes.White;

// рисуем диаграмму
for(int i = 0; i < cv.Length ;i++) {
    // длинна дуги
    swa = (int)(360 * (cv[i]/se)) + 1;

    // определяем цвет сектора
    switch (i)
    {
        case 0:
            fBrush = Brushes.Lime;
            break;
        case 1:
            fBrush = Brushes.Gold;
            break;
        case 2:
            fBrush = Brushes.DeepPink;
            break;
        case 3:
            fBrush = Brushes.Violet;
            break;
        case 4:
            fBrush = Brushes.OrangeRed;
            break;
        case 5:
            fBrush = Brushes.RoyalBlue;
            break;
        case 6:
            fBrush = Brushes.SteelBlue;
            break;
```

```
            case 7:
                fBrush = Brushes.Chocolate;
                break;
            case 8:
                fBrush = Brushes.LightGray;
                break;
            case 9:
                fBrush = Brushes.Gold;
                break;
        }

        // сектор диаграммы
        g.FillPie(fBrush, x, y, sd, sd, sta, swa);

        // граница сектора
        g.DrawPie(System.Drawing.Pens.Black,
                  x, y, sd, sd, sta, swa);

        // начальная точка дуги для следующего сектора
        sta = sta + (int)(360 * (cv[i]/se)) + 1;

        // легенда
        g.FillRectangle(fBrush,
                  lx, ly + i*20, 20, 10);
        g.DrawRectangle(System.Drawing.Pens.Black,
                  lx, ly + i*20, 20, 10);

        // подпись легенды
        g.DrawString(lv[i] + ": " +
            Convert.ToDouble(cv[i]/se*100).ToString("N") +
            "%", mfont, System.Drawing.Brushes.Black,
            lx + 20, ly + i*20 - 4);
    }
}

// конструктор формы
public WinForm()
{
    InitializeComponent();
```

```
try{
    // создаем поток для чтения и считываем данные;
    // Application.StartupPath возвращает путь
    // к каталогу, из которого была запущена программа;
    // если не указать кодировку, могут возникнуть
    // проблемы с отображением кириллических символов
    sr = new System.IO.StreamReader(
        Application.StartupPath + "\\dinput.txt",
        System.Text.Encoding.GetEncoding(1251));

    // считываем заголовок диаграммы
    hl = sr.ReadLine();

    // считываем данные о количестве записей
    // и инициализируем массивы
    ne = Convert.ToInt16(sr.ReadLine());
    cv = new double[Convert.ToInt16(ne)];
    lv = new string[Convert.ToInt16(ne)];

    // считываем записи
    int i=0;
    string t = sr.ReadLine();
    while (t != null){
        // записываем считанное число
        // в массив численных значений диаграммы
        cv[i] = Convert.ToDouble(t.Substring(0,
                                 t.IndexOf((char)32)));

        // считаем сумму элементов массива
        se += cv[i];

        // записываем комментарий к считанному значению
        // в массив подписей легенды диаграммы
        lv[i] = t.Substring(t.IndexOf((char)32));

        // считываем следующую запись
        t = sr.ReadLine();

        // увеличиваем счетчик считанных записей
        i++;
    }
```

```
        // закрываем поток
        sr.Close();

        // если записей нет
        if (ne == 0)
            MessageBox.Show("Файл исходных данных пуст.",
                "Круговая диаграмма",
                MessageBoxButtons.OK, MessageBoxIcon.Error);

        // если исходный файл имеет
        // неверное количество записей
        if (i < ne)
            MessageBox.Show("Файл исходных данных" +
                "поврежден\n" +
                "или имеет неверное количество записей.",
                "Круговая диаграмма",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);

        // если все данные считались и
        // никаких исключений не возникло
        if (ne != 0 && ne == i) {
            // процедура прорисовки диаграммы drawDiagram
            // будет вызываться каждый раз, когда будет
            // происходит событие Paint
            this.Paint += new PaintEventHandler(drawDiagram);
        }
    }
    // обработка исключений:
    // файл исходных данных не найден
    catch(System.IO.FileNotFoundException exc){
        MessageBox.Show("Файл исходных данных не найден.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
            "Круговая диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
```

```
    // ошибка формата исходных данных
    catch(FormatException exc){
        MessageBox.Show("Ошибка формата исходных данных.\n" +
            exc.Message + "\n" +
            exc.GetType().ToString(),
            "Круговая диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
    // прочие исключения
    catch(Exception exc){
        MessageBox.Show(exc.Message + "\n" +
            exc.GetType().ToString(),
            "Круговая диаграмма",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }

    // задаем цвет формы
    this.BackColor = System.Drawing.Color.White;
}

// изменение размера окна программы
private void WinForm_SizeChanged(object sender,
   System.EventArgs e)
{
    // метод Refresh перерисовывает форму полностью
    this.Refresh();
}
```

Использование кистей

Программа демонстрирует работу с кистями различных типов. Для доступа к `LeanerGradienBrush` и `TextureBrush` в директиву `uses` программы необходимо добавить ссылку на библиотеку `System.Drawing.Drawing2D`. Окно программы приведено на рис. 1.24.

```
// обработка события Paint
private void WinForm_Paint(
object sender, System.Windows.Forms.PaintEventArgs e)
```

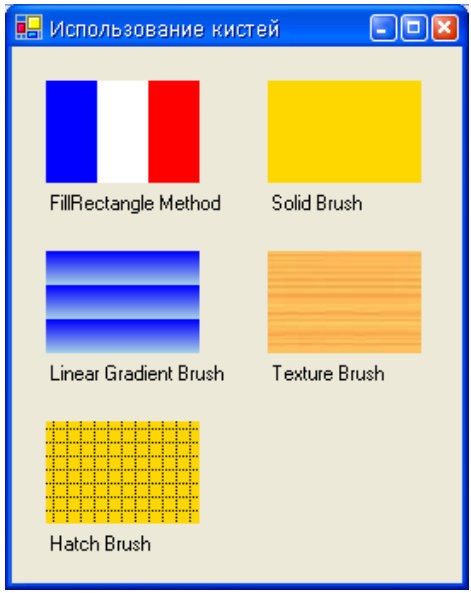

Рис. 1.24. Окно программы **Использование кистей**

```
{
    // использование стандартного метода FillRectangle
    e.Graphics.FillRectangle(Brushes.Blue,20,20,30,60);
    e.Graphics.FillRectangle(Brushes.White,50,20,30,60);
    e.Graphics.FillRectangle(Brushes.Red,80,20,30,60);
    e.Graphics.DrawString("FillRectangle Method", this.Font,
                          Brushes.Black, 20, 85);

    // использование сплошной кисти SolidBrush
    SolidBrush aBrush =
        new SolidBrush(Color.Gold);

    e.Graphics.FillRectangle(aBrush, 150, 20, 90, 60);
    e.Graphics.DrawString("Solid Brush", this.Font,
                          Brushes.Black, 150, 85);

    // использование градиентной кисти LinearGradientBrush
    LinearGradientBrush lgBrush =
        new LinearGradientBrush(
```

```
            new RectangleF(0,0,20,20),
            Color.Blue, Color.LightBlue,
            LinearGradientMode.Vertical);

    e.Graphics.FillRectangle(lgBrush, 20, 120, 90, 60);
    e.Graphics.DrawString("Linear Gradient Brush", this.Font,
                          Brushes.Black, 20, 185);

    // использование штрихованной кисти HatchBrush
    HatchBrush hBrush =
        new HatchBrush(HatchStyle.DottedGrid,
            Color.Black, Color.Gold);

    e.Graphics.FillRectangle(hBrush, 20, 220, 90, 60);
    e.Graphics.DrawString("Hatch Brush", this.Font,
                          Brushes.Black, 20, 285);

    // использование текстурной кисти TextureBrush
    try{
        TextureBrush tBrush =
            new TextureBrush(Image.FromFile("texture01.jpg"));
        e.Graphics.FillRectangle(tBrush, 150, 120, 90, 60);
    }
    catch (System.IO.FileNotFoundException){
        e.Graphics.DrawRectangle(Pens.Black,
               150, 120, 90, 60);
        e.Graphics.DrawString("Source image", this.Font,
               Brushes.Black, 160, 135);
        e.Graphics.DrawString(" not found", this.Font,
               Brushes.Black, 170, 150);
    }
    e.Graphics.DrawString("Texture Brush", this.Font,
                          Brushes.Black, 150, 185);
}
```

Фоновый рисунок

Программа **Фоновый рисунок** демонстрирует, как можно получить фоновый рисунок путем многократного вывода битового

образа на поверхность формы. Битовый образ (рис. 1.25, *а*), используемый для формирования фонового рисунка, загружается из файла. Окно программы приведено на рис. 1.25, *б*.

Рис. 1.25, *а*. Используемый битовый образ

Рис. 1.25, *б*. Окно программы **Фоновый рисунок**

```
// переменная показывает, загружено ли фоновое изображение
private Boolean back = true;

// переменная содержит фоновое изображение
private System.Drawing.Bitmap aBitmap;

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    try{
        aBitmap = new Bitmap("net.bmp");
    }
    catch{
        back = false;
    }
}
```

```
// обработка события Paint
private void WinForm_Paint(object sender,
    System.Windows.Forms.PaintEventArgs e)
{
    // фоновый рисунок не загружен
    if (!back){
        e.Graphics.DrawString(
            "Фоновый рисунок net.bmp недоступен.",
            this.Font, Brushes.Black, 10, 10);
        return;
    }

    // фоновый рисунок загружен
    else {
        Point outP = new Point(0,0);

        // дублирование рисунка по вертикали
        while (outP.Y < this.ClientSize.Height) {
            outP.X = 0;

            // дублирование рисунка по горизонтали
            while (outP.X < this.ClientSize.Width ) {
                e.Graphics.DrawImage(aBitmap,outP);
                outP.X += aBitmap.Width;
            }

            outP.Y += aBitmap.Height;
        }
    }
}
```

Бегущая строка

Программа **Бегущая строка** демонстрирует использование битового образа для вывода баннера в стиле бегущей строки. При наведении указателя мыши на баннер, его движение останавливается. Окно программы приведено на рис. 1.26.

Рис. 1.26. Окно программы **Бегущая строка**

```
// графическая поверхность
private Graphics g;

// переменная содержит графический файл баннера
private System.Drawing.Bitmap psource;

// область вывода баннера
private Rectangle rct;

// таймер
private System.Windows.Forms.Timer timer1;

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    try{
        // загружаем файл баннера
        psource = new Bitmap("banner.bmp");
    }
    catch(Exception exc){
        // выводим сообщение произошедшего исключения
        MessageBox.Show(exc.ToString(),"Бегущая строка");

        // закрываем форму
        this.Close();
        return;
    }

    // определяем графическую поверхность
    g = this.CreateGraphics();

    // определяем область вывода баннера
    rct.X = 0;
```

```
    rct.Y = Convert.ToInt16(
           (this.ClientSize.Height -
            psource.Height) / 2);
    rct.Width  = psource.Width;
    rct.Height = psource.Height;

    // определяем интервал обработки события таймера
    timer1.Interval = 50;
    timer1.Enabled  = true;
}

// обработка события таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    // движение баннера
    rct.X -= 1;

    if (Math.Abs(rct.X) > rct.Width)
        rct.X += rct.Width;

    // рисуем баннер
    for (int i = 0; i <= Convert.ToInt16(
         this.ClientSize.Width / rct.Width) + 1; i++)
        g.DrawImage(psource, rct.X + i*rct.Width, rct.Y);
}

// движение курсора мыши
private void WinForm_MouseMove(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    // при наведении указателя мыши на баннер
    // движение строки останавливается
    if ((e.Y < rct.Y + rct.Height) && (e.Y > rct.Y)) {
        if (timer1.Enabled != false)
            timer1.Enabled = false;
    }
    else {
        if (timer1.Enabled != true)
            timer1.Enabled = true;
    }
}
```

Полет в облаках

Программа **Полет в облаках** демонстрирует формирование изображения из нескольких битовых образов (самолет движется на фоне неба). Изображения объекта и фона загружаются из файлов. Программа демонстрирует работу с битовыми образами, действие метода `Invalidate`. Окно программы приведено на рис. 1.27. Фоновый рисунок и битовый образ представлены соответственно на рис. 1.28, *а* и *б*.

Рис. 1.27. Окно программы **Полет в облаках**

Рис. 1.28, *а*. Фоновый рисунок программы **Полет в облаках**

Рис. 1.28, *б*. Битовый образ для программы **Полет в облаках**

```
// переменные содержат изображения неба и самолета
private System.Drawing.Bitmap sky, plane;

// рабочая поверхность
private Graphics g;

// приращение координаты по X определяет скорость полета
private int dx;

// область, в которой находится самолет
private Rectangle rct;

// true - самолет скрывается в облаках
private Boolean demo = true;

// генератор случайных чисел
private System.Random rnd;

// таймер
private System.Windows.Forms.Timer timer1;

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    try {
        // загрузить битовые образы
        sky   = new Bitmap("sky.bmp");     // небо
        plane = new Bitmap("plane.bmp");  // самолет

        plane.MakeTransparent();
    }
    catch(Exception exc) {
        MessageBox.Show("Ошибка загрузки битовых образов\n" +
            "(sky.bmp или plane.bmp)." + exc.ToString(),
            "Полет в облаках",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);

        this.Close();
        return;
    }
```

```
    // фоновый рисунок формы
    BackgroundImage = new Bitmap("sky.bmp");

    // задаем размер формы, соответствующий
    // фоновому рисунку
    this.ClientSize =
        new System.Drawing.Size(
            new Point(BackgroundImage.Width,
            BackgroundImage.Height));

    // тип границы окна приложения
    this.FormBorderStyle =
        System.Windows.Forms.FormBorderStyle.FixedSingle;
    this.MaximizeBox = false;

    // g - графическая поверхность, на которой
    // будем формировать рисунок
    g = Graphics.FromImage(BackgroundImage);

    // инициализация генератора случ. чисел
    rnd = new System.Random();

    // начальное положение самолета
    rct.X = -40;
    rct.Y =  20 + rnd.Next(20);

    rct.Width  = plane.Width;
    rct.Height = plane.Height;

    /*
    скорость полета определяется периодом следования
    сигналов от таймера (значение свойства Timer1.Interval)
    и величиной приращения координаты по X
    */

    dx = 2;         // скорость полета - 2 пиксела/тик_таймера

    timer1.Interval = 20;
    timer1.Enabled  = true;
}
```

```
// сигнал от таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    // стираем изображение самолета путем копирования
    // области фона (из буфера) на рабочую поверхность
    g.DrawImage(sky, new Point(0,0));

    // изменяем положение самолета
    if (rct.X < this.ClientRectangle.Width)
        rct.X += dx;
    else {
        // если достигли границы, задаем заново
        // положение самолета
        rct.X = -40;
        rct.Y = 20 +
            rnd.Next(this.ClientSize.Height -
                     40 - plane.Height);

        // скорость полета от 2 до 5 пикселов/тик_таймера
        dx = 2 + rnd.Next(4);
    }

    // рисуем самолет на рабочей поверхности
    g.DrawImage(plane, rct.X, rct.Y);

    /*
    Метод Refresh перерисовывает всю форму. Метод Invalidate
    без параметра - тоже. Метод Invalidate с параметорм
    обеспечивает перерисовку только той области формы, которая
    указана в параметрах вызова метода. Ниже приведен фрагмент
    кода, который демонстрирует работу метода Invalidate.
    Если в качестве параметра методу Invalidate передать
    объект rct, то самолет долетит до правой границы окна,
    если reg - то исчезнет, как только значение reg.X
    будет больше значения sky.Width-40.
    */

    if (!demo)
        // обновить (перерисовать) область,
        // в которой сейчас должен быть самолет
        this.Invalidate(rct);
```

```
    else {
        // используется для демонстрации работы
        // метода Invalidate
        Rectangle reg = new Rectangle(20, 20,
            sky.Width - 40, sky.Height - 40);

        g.DrawRectangle(Pens.Black,
            reg.X , reg.Y, reg.Width-1, reg.Height-1);

        // обновить область
        this.Invalidate(reg);
    }
}
```

БАЗЫ ДАННЫХ

В состав Borland C#Builder включены компоненты, поддерживающие различные технологии доступа к данным. В этом разделе приведены примеры использования компонентов Borland Data Provider.

Общие замечания

- Соединение с базой данных (InterBase, MS SQL, MSAccess и др.) осуществляется при помощи компонента `BdpConnection`.
- Взаимодействие с базой данных, после того как соединение установлено, осуществляется при помощи компонента `BdpAdapter`.
- Хранение таблиц базы данных может осуществляться компонентом `DataSet`.
- Таблица базы данных записывается в `DataSet` при помощи вызова метода `Fill()` компонента `BdpDataAdapter`.
- Простой вывод содержимого компонента `DataSet` может быть осуществлен при помощи компонента `DataGrid`.

Записная книжка

Программа **Записная книжка** является простым примером программы работы с базой данных Microsoft Access. Демонстрирует использование компонентов `BdpConnection`, `BdpDataAdapter`, `DataSet` и `DataGrid`. База данных **Записная книжка** (phones.mdb) состоит из одной-единственной таблицы `Phones` (табл. 1.1). Форма и окно программы приведены на рис. 1.29 и 1.30, значения свойств компонентов `dataSet1`, `dataGrid1`, `bdpConnection1` и `bdpDataAdapter1` приведены в табл. 1.2—1.5.

Таблица 1.1. *Поля таблицы* `Phones` *базы данных* `phones.mdb`

Поле	Тип	Размер	Информация
`Title`	Текстовый	60	Имя
`Phone`	Текстовый	20	Телефон

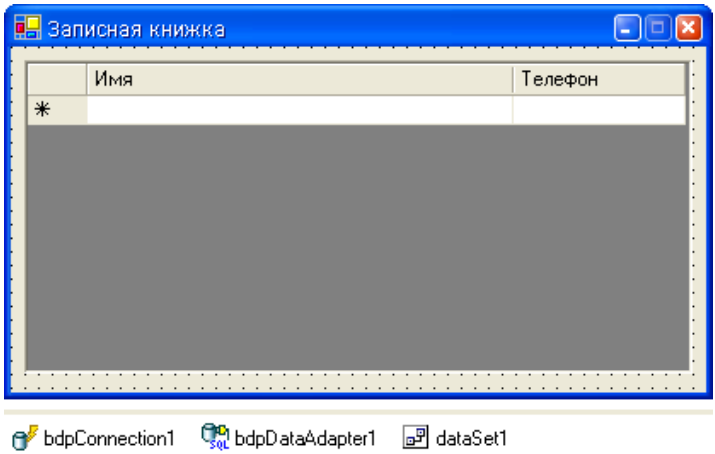

Рис. 1.29. Форма программы **Записная книжка**

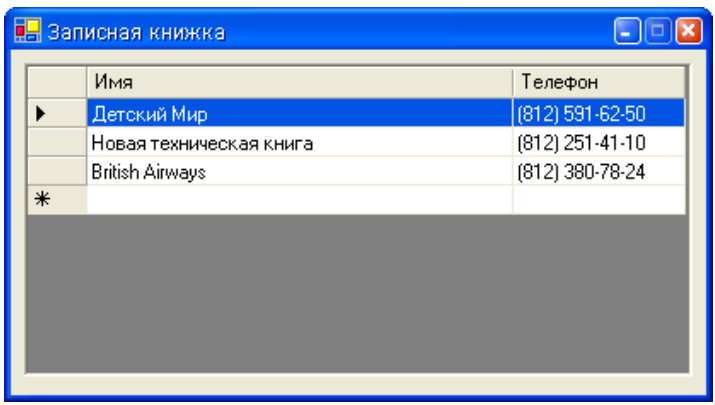

Рис. 1.30. Окно программы **Записная книжка**

Таблица 1.2. *Значения свойств компонента* `dataSet1`

Свойство	**Значение**
`Locale`	`Russian(Russia)`
`Tables.dataTable1.TableName`	`Phones`
`Tables.dataTable1.Columns. dataColumn1.Caption`	`Title`

Таблица 1.2 *(окончание)*

Свойство	Значение
Tables.dataTable1.Columns. dataColumn1.ColumnName	Title
Tables.dataTable1.Columns. dataColumn2.Caption	Phone
Tables.dataTable1.Columns. dataColumn2.ColumnName	Phone

Таблица 1.3. *Значения свойств компонента* `dataGrid1`

Свойство	Значение
DataSource	dataSet1
DataMember	Phones
CaptionVisible	False
ColumnHeaderVisible	True
TableStyles.dataGridTableStyle1. MappingName	Phones
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.HeaderText	Имя
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.NullText	пустая строка
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.Width	250
TableStyles.dataGridTableStyle1. GridColumnStyles. dataGridTextBoxColumn1.MappingName	Title
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.HeaderText	Телефон
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.NullText	пустая строка
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.Width	100
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.MappingName	Phone

Таблица 1.4. *Значения свойств компонента* `bdpConnection1`

Свойство	Значение
ConnectionString.Database	phones.mdb
ConnectionString.UserName	admin
ConnectionString.UserPassword	пустая строка

Таблица 1.5. *Значения свойств компонента* `bdpDataAdapter1`

Свойство	Значение
TableMapping.SourceTable	Table
TableMapping.DatasetTable	Phones
TableMapping.ColumnMappings	SourceColumn: Title DatasetColumn: Title
TableMapping.ColumnMappings	SourceColumn: Phone DatasetColumn: Phone
SelectCommand.CommandText	SELECT Title, Phone FROM Phones
SelectCommand.Connection	bdpConnection1
SelectCommand.ParametrCount	0
DeleteCommand.CommandText	DELETE FROM Phones WHERE Title = ? AND Phone = ?
DeleteCommand.Connection	bdpConnection1
DeleteCommand.ParametrCount	2
DeleteCommand.Parameters. TitleOriginal.BdpType	String
DeleteCommand.Parameters. TitleOriginal.MaxPrecision	60
DeleteCommand.Parameters. TitleOriginal.SourceColumn	Title
DeleteCommand.Parameters. TitleOriginal.SourceVersion	Original
DeleteCommand.Parameters. PhoneOriginal.BdpType	String

Таблица 1.5 *(продолжение)*

Свойство	Значение
DeleteCommand.Parameters. PhoneOriginal.MaxPrecision	20
DeleteCommand.Parameters. PhoneOriginal.SourceColumn	Phone
DeleteCommand.Parameters. PhoneOriginal.SourceVersion	Original
InsertCommand.CommandText	INSERT INTO Phones (Title, Phone) VALUES (?, ?)
InsertCommand.Connection	bdpConnection1
InsertCommand.ParametrCount	2
InsertCommand.Parameters. Title.BdpType	String
InsertCommand.Parameters. Title.MaxPrecision	60
InsertCommand.Parameters. Title.SourceColumn	Title
InsertCommand.Parameters. Title.SourceVersion	Current
InsertCommand.Parameters. Phone.BdpType	String
InsertCommand.Parameters. Phone.MaxPrecision	20
InsertCommand.Parameters. Phone.SourceColumn	Phone
InsertCommand.Parameters. Phone.SourceVersion	Current
UpdateCommand.CommandText	UPDATE Phones SET Title = ?, Phone = ? WHERE Title = ? AND Phone = ?
UpdateCommand.Connection	bdpConnection1
UpdateCommand.ParametrCount	4
UpdateCommand.Parameters. Title.BdpType	String

Таблица 1.5 *(окончание)*

Свойство	Значение
UpdateCommand.Parameters. Title.MaxPrecision	60
UpdateCommand.Parameters. Title.SourceColumn	Title
UpdateCommand.Parameters. Title.SourceVersion	Current
UpdateCommand.Parameters. Phone.BdpType	String
UpdateCommand.Parameters. Phone.MaxPrecision	20
UpdateCommand.Parameters. Phone.SourceColumn	Phone
UpdateCommand.Parameters. Phone.SourceVersion	Current
UpdateCommand.Parameters. TitleOriginal.BdpType	String
UpdateCommand.Parameters. TitleOriginal.MaxPrecision	60
UpdateCommand.Parameters. TitleOriginal.SourceColumn	Title
UpdateCommand.Parameters. TitleOriginal.SourceVersion	Original
UpdateCommand.Parameters. PhoneOriginal.BdpType	String
UpdateCommand.Parameters. PhoneOriginal.MaxPrecision	20
UpdateCommand.Parameters. PhoneOriginal.SourceColumn	Phone
UpdateCommand.Parameters. PhoneOriginal.SourceVersion	Original

```
// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    try {
        bdpDataAdapter1.Fill(dataSet1.Tables["Phones"]);
    }
    catch(Exception exc) {
        MessageBox.Show(
            "Ошибка доступа к базе данных.\n" +
            exc.Message, "Записная книжка",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        this.Close();
    }
}

// завершение работы программы
private void WinForm_Closing(object sender,
   System.ComponentModel.CancelEventArgs e)
{
    try {
        // сохранить изменения БД
        bdpDataAdapter1.Update(dataSet1);
    }
    catch(Exception exc) {
        MessageBox.Show(
            "Ошибка доступа к базе данных.\n" +
            exc.Message, "Записная книжка",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}
```

Контакты

Программа **Контакты** является еще одним простым примером программы работы с базой данных Microsoft Access. Демонстрирует использование компонентов `BdpConnection`, `BdpDataAdapter`, `DataSet`, `ImageList`, `ToolTip`. Компонент `ImageList` используется

для хранения изображения стрелок операционных кнопок, `ToolTip` обеспечивает вывод всплывающих подсказок при наведении указателя мыши на операционные кнопки. Форма и окно программы приведены на рис. 1.31 и 1.32. Характеристики полей таблицы `Contacts` базы данных contacts.mdb приведены в табл. 1.6, значения свойств компонентов приведены в табл. 1.7—1.10.

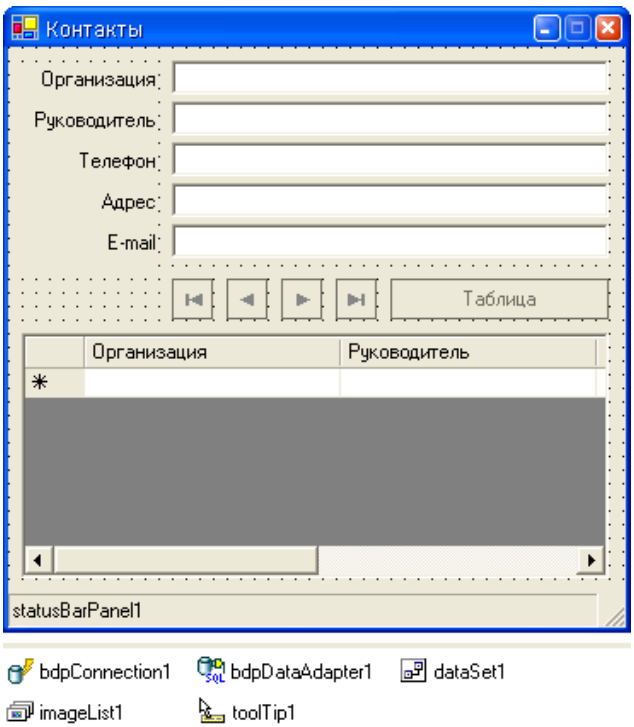

Рис. 1.31. Форма программы **Контакты**

Таблица 1.6. *Поля таблицы* `Contacts` *базы данных* `contacts.mdb`

Поле	Тип	Размер	Информация
`Title`	Текстовый	60	Организация
`Phone`	Текстовый	20	Телефон

Таблица 1.6 *(окончание)*

Поле	Тип	Размер	Информация
`Manager`	Текстовый	50	Руководитель
`Address`	Текстовый	50	Адрес
`Email`	Текстовый	50	E-mail

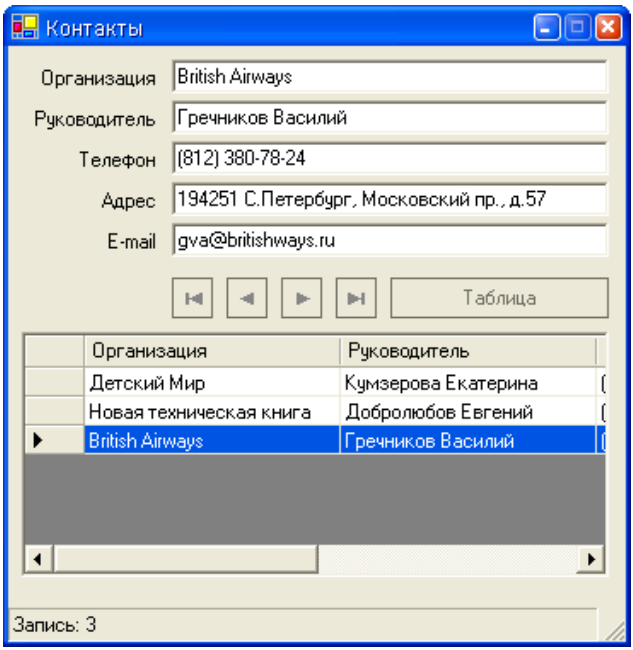

Рис. 1.32. Окно программы **Контакты**

Таблица 1.7. *Значения свойств компонента* `dataSet1`

Свойство	Значение
`Locale`	`Russian(Russia)`
`Tables.dataTable1.TableName`	`Contacts`
`Tables.dataTable1.Columns.dataColumn1.ColumnName`	`Title`

Таблица 1.7 *(окончание)*

Свойство	Значение
Tables.dataTable1.Columns.dataColumn2.ColumnName	Phone
Tables.dataTable1.Columns.dataColumn3.ColumnName	Manager
Tables.dataTable1.Columns.dataColumn4.ColumnName	Address
Tables.dataTable1.Columns.dataColumn5.ColumnName	Email

Таблица 1.8. *Значения свойств компонентов* `textBox1` *—* `textBox5`

Свойство	Значение
textBox1.(DataBindings).Text	dataTable1 - Title
textBox2.(DataBindings).Text	dataTable1 - Manager
textBox3.(DataBindings).Text	dataTable1 - Phone
textBox4.(DataBindings).Text	dataTable1 - Address
textBox5.(DataBindings).Text	dataTable1 - Email

Таблица 1.9. *Значения свойств компонента* `dataGrid1`

Свойство	Значение
DataSource	dataSet1
DataMember	Contacts
CaptionVisible	False
ColumnHeaderVisible	True

Таблица 1.10. *Значения свойств кнопок* `button1` *—* `button5`

Свойство	Значение
button1-button5.ForeColor	SystemColors.ControlDarkDark
button1-button5.FlatStyle	Flat
button1.ImageList	imageList1
button1.ImageIndex	0

Таблица 1.10 *(окончание)*

Свойство	Значение
button1.ToolTip	К первой
button1.ImageList	imageList1
button2.ImageIndex	3
button2.ToolTip	К предыдущей
button1.ImageList	imageList1
button3.ImageIndex	2
button3.ToolTip	К следующей
button1.ImageList	imageList1
button4.ImageIndex	1
button4.ToolTip	К последней
button5.ToolTip	Отобразить таблицу

```
// объект myBindingManagerBase используется для связи
// компонента dataSet1 с таблицей базы данных Contacts
// (для индексирования выбранной записи БД)
private
    System.Windows.Forms.BindingManagerBase
    myBindingManagerBase;

// изменился индекс выбранной записи БД
private void PositionChanged(object sender,
System.EventArgs e){
    if (myBindingManagerBase.Position != -1) {
        // записи в БД нумеруются с нуля
        statusBar1.Panels[0].Text =
            "Запись: " +
            (myBindingManagerBase.Position + 1).ToString();

        if (dataGrid1.Visible)
            dataGrid1.CurrentRowIndex =
                myBindingManagerBase.Position;
    }
}
```

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    try{
        // загружаем данные в dataSet1
        bdpDataAdapter1.Fill(dataSet1.Tables["Contacts"]);
    }
    catch(Exception exc) {
        MessageBox.Show(
            "Ошибка доступа к базе данных.\n" +
            exc.Message, "Контакты",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }

    // создаем объект типа BindingContext и определяем
    // процедуру обработки события PositionChanged
    myBindingManagerBase =
        this.BindingContext[dataSet1.Tables["Contacts"]];

    myBindingManagerBase.PositionChanged +=
        new System.EventHandler(this.PositionChanged);

    // скрываем dataDrid1 и уменьшаем размер формы
    dataGrid1.Visible = false;

    this.ClientSize =
        new Size(this.ClientSize.Width,
        dataGrid1.Top + statusBar1.Height);

    // настраиваем поля вывода информации
    textBox1.ReadOnly = true;
    textBox2.ReadOnly = true;
    textBox3.ReadOnly = true;
    textBox4.ReadOnly = true;
    textBox5.ReadOnly = true;
```

```
    textBox1.BackColor = Color.White;
    textBox2.BackColor = Color.White;
    textBox3.BackColor = Color.White;
    textBox4.BackColor = Color.White;
    textBox5.BackColor = Color.White;

    dataGrid1.ReadOnly = true;

    if (myBindingManagerBase.Count == 0)
        statusBar1.Panels[0].Text =
            "База данных не содержит записей.";
    else
        statusBar1.Panels[0].Text =
            "Запись: " +
            (myBindingManagerBase.Position + 1).ToString();
}

// щелчок на кнопке К первой записи
private void button1_Click(object sender, System.EventArgs e)
{
    if (myBindingManagerBase.Count > 0)
        myBindingManagerBase.Position = 0;
}

// щелчок на кнопке К предыдущей записи
private void button2_Click(object sender, System.EventArgs e)
{
    if (myBindingManagerBase.Position > 0)
        myBindingManagerBase.Position -= 1;
}

// щелчок на кнопке К следующей записи
private void button3_Click(object sender, System.EventArgs e)
{
    if ((myBindingManagerBase.Position <
         myBindingManagerBase.Count-1) &&
        (myBindingManagerBase.Count > 0))
        myBindingManagerBase.Position += 1;
}
```

```
// щелчок на кнопке К последней записи
private void button4_Click(object sender, System.EventArgs e)
{
    if (myBindingManagerBase.Count > 0)
        myBindingManagerBase.Position =
            myBindingManagerBase.Count-1;
}

// щелчок на кнопке Таблица
private void button5_Click(object sender, System.EventArgs e)
{
    if (dataGrid1.Visible) {
        // скрыть таблицу
        dataGrid1.Visible = false;

        this.ClientSize =
        new Size(this.ClientSize.Width,
            dataGrid1.Top + statusBar1.Height);

        toolTip1.SetToolTip(button5, "Отобразить таблицу");
    }
    else {
        // отобразить таблицу
        dataGrid1.Visible = true;

        dataGrid1.CurrentRowIndex =
            myBindingManagerBase.Position;

        this.ClientSize =
            new Size(this.ClientSize.Width,
            dataGrid1.Bottom + 16 + statusBar1.Height);

        toolTip1.SetToolTip(button5, "Скрыть таблицу");
    }
}

// изменился индекс выбранной ячейки компонента dataGrid1
private void dataGrid1_CurrentCellChanged(object sender,
   System.EventArgs e)
```

```
{
    if (myBindingManagerBase.Position !=
            dataGrid1.CurrentRowIndex)
        myBindingManagerBase.Position =
            dataGrid1.CurrentRowIndex;
}
```

Ежедневник

Программа **Ежедневник** демонстрирует работу с базой данных Microsoft Access (journal.mdb). База данных содержит таблицу journal, в которой содержатся записи запланированных дел. Программа позволяет просмотреть список запланированных дел на сегодня, завтра, эту неделю и следующую. Добавление нового события осуществляется при помощи формы **Новое событие** (рис. 1.33). Выборка данных при просмотре ежедневника осуществляется из таблицы компонента `dataSet1`, а не непосредственно из базы данных. Характеристики полей таблицы journal базы данных journal.mdb приведены в табл. 1.11, значения свойств компонентов `dataSet1`, `bdpConnection1`, `bdpDataAdapter1` и

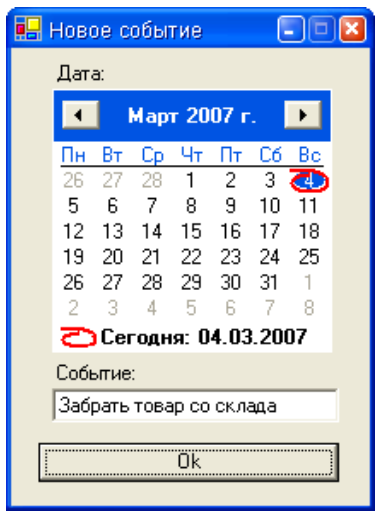

Рис. 1.33. Форма **Новое событие**

dataGrid1 приведены в табл. 1.12—1.15, форма и окно программы приведены на рис. 1.34 и 1.35.

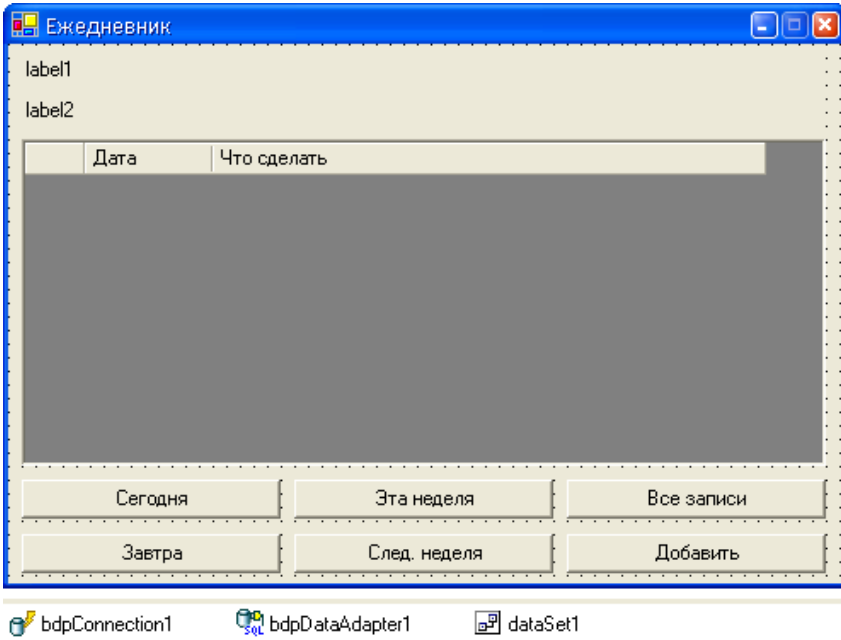

Рис. 1.34. Форма программы **Ежедневник**

Таблица 1.11. *Поля таблицы* `Contacts` *базы данных* `journal.mdb`

Поле	Тип	Размер	Информация
jdate	Дата/время	-	Дата
jwhat	Текстовый	60	Что сделать

Таблица 1.12. *Значения свойств компонента* `dataSet1`

Свойство	Значение
Locale	Russian(Russia)
Tables.dataTable1.TableName	journal

Таблица 1.12 *(окончание)*

Свойство	Значение
Tables.dataTable1.Columns.dataColumn1.ColumnName	jdate
Tables.dataTable1.Columns.dataColumn1.DataType	System.DataType
Tables.dataTable1.Columns.dataColumn2.ColumnName	jwhat
Tables.dataTable1.Columns.dataColumn2.DataType	System.String
Tables.dataTable2.TableName	journalSelect
Tables.dataTable2.Columns.dataColumn1.ColumnName	jdate
Tables.dataTable2.Columns.dataColumn1.DataType	System.DataType
Tables.dataTable2.Columns.dataColumn2.ColumnName	jwhat
Tables.dataTable2.Columns.dataColumn2.DataType	System.String

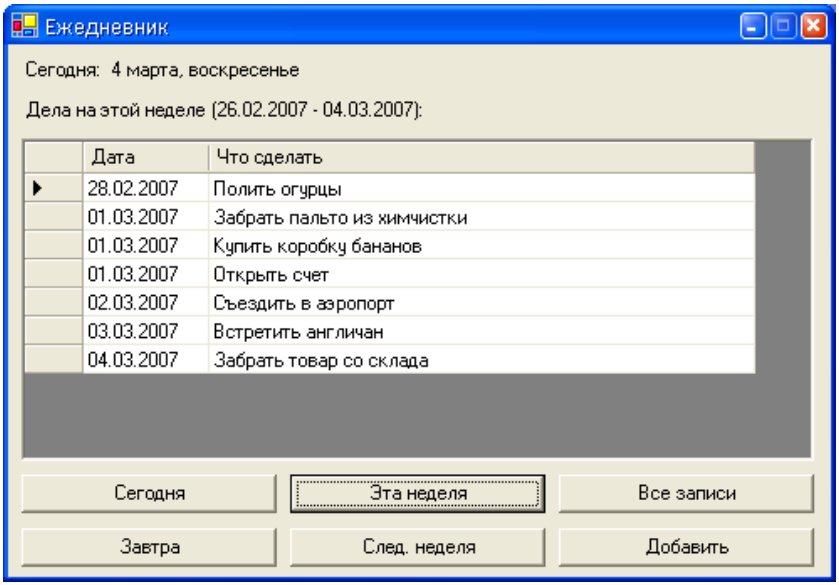

Рис. 1.35. Окно программы **Ежедневник**

Таблица 1.13. *Значения свойств компонента* `bdpConnection1`

Свойство	Значение
ConnectionString.Database	journal.mdb
ConnectionString.UserName	admin
ConnectionString.UserPassword	пустая строка

Таблица 1.14. *Значения свойств компонента* `bdpDataAdapter1`

Свойство	Значение
TableMapping.SourceTable	Table
TableMapping.DatasetTable	journal
TableMapping.ColumnMappings	SourceColumn: jdate DatasetColumn: jdate
TableMapping.ColumnMappings	SourceColumn: jwhat DatasetColumn: jwhat
SelectCommand.CommandText	SELECT jdate, jwhat FROM journal
InsertCommand.CommandText	INSERT INTO journal (jdate, jwhat) VALUES (?, ?)
UpdateCommand.CommandText	UPDATE journal SET jdate = ?, jwhat = ? WHERE jdate = ? AND jwhat = ?

Таблица 1.15. *Значения свойств компонента* `dataGrid1`

Свойство	Значение
DataSource	dataSet1
DataMember	journalSelect
CaptionVisible	False

Таблица 1.15 *(окончание)*

Свойство	Значение
ColumnHeaderVisible	True
TableStyles.dataGridTableStyle1.MappingName	journalSelect
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn1.HeaderText	Дата
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn1.Width	75
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn1.Format	dd/MM/yyyy
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn1.MappingName	jdate
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn2.HeaderText	Что сделать
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn2.Width	325
TableStyles.dataGridTableStyle1.GridColumnStyles.dataGridTextBoxColumn2.MappingName	jwhat
ReadOnly	True

```
// главная форма программы (WinForm):
// конструктор формы
public WinForm()
{
    InitializeComponent();

    label1.Text = "Сегодня:  " +
        DateTime.Today.ToString("d MMMM, dddd");

    label2.Text = "Дела на сегодня:";

    try{
        // загружаем данные из journal.mdb в dataSet1
        bdpDataAdapter1.Fill(dataSet1.Tables["journal"]);
    }
```

```
    catch(Exception exc) {
        MessageBox.Show(
            "Ошибка доступа к базе данных.\n" +
            exc.Message, "Ежедневник",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);

        button1.Enabled = false;
        button2.Enabled = false;
        button3.Enabled = false;
        button4.Enabled = false;
        button5.Enabled = false;
        button6.Enabled = false;

        return;
    }

    // формируем выборку дел на сегодня
    System.Data.DataRow[] dr =
        dataSet1.Tables["journal"].Select("jdate = #" +
            DateTime.Today.Month.ToString() + "/" +
            DateTime.Today.Day.ToString() + "/" +
            DateTime.Today.Year.ToString() + "#");

    // добавляем полученную выборку в таблицу
    if (dr.Length > 0)
        for(int i = 0; i < dr.Length; i++)
            dataSet1.Tables["journalSelect"].Rows.
                Add(dr[i].ItemArray);

    // сортировка выводимой в dataGrid1 информации
    dataSet1.Tables["journalSelect"].
        DefaultView.Sort = "jdate, jwhat ASC";

    dataGrid1.DataSource =
        dataSet1.Tables["journalSelect"].DefaultView;
}
```

```
// щелчок на кнопке Сегодня
private void button1_Click(object sender, System.EventArgs e)
{
    label2.Text = "Дела на сегодня:";

    // формируем выборку дел на сегодня
    System.Data.DataRow[] dr =
        dataSet1.Tables["journal"].Select("jdate = #" +
            DateTime.Today.Month.ToString() + "/" +
            DateTime.Today.Day.ToString() + "/" +
            DateTime.Today.Year.ToString() + "#");

    // добавляем полученную выборку в таблицу
    dataSet1.Tables["journalSelect"].Clear();
    if (dr.Length > 0)
        for(int i = 0; i < dr.Length; i++)
            dataSet1.Tables["journalSelect"].Rows.
                Add(dr[i].ItemArray);
    else
        MessageBox.Show(
            "На сегодня никаких дел не запланировано.",
            "Ежедневник", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
}

// щелчок на кнопке Завтра
private void button2_Click(object sender, System.EventArgs e)
{
    label2.Text = "Дела на завтра:";

    // завтрашняя дата
    DateTime tm = DateTime.Today.AddDays(1);

    // формируем выборку дел на завтра
    System.Data.DataRow[] dr =
        dataSet1.Tables["journal"].Select("jdate = #" +
            tm.Month.ToString() + "/" +
            tm.Day.ToString() + "/" +
            tm.Year.ToString() + "#");
```

```
    // добавляем полученную выборку в таблицу
    dataSet1.Tables["journalSelect"].Clear();
    if (dr.Length > 0)
        for(int i = 0; i < dr.Length; i++)
            dataSet1.Tables["journalSelect"].Rows.
                Add(dr[i].ItemArray);
    else
        MessageBox.Show(
            "На завтра никаких дел не запланировано.",
            "Ежедневник", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
}

// щелчок на кнопке Эта неделя
private void button3_Click(object sender, System.EventArgs e)
{
    // определяем дату начала и конца этой недели
    DateTime wf = DateTime.Today;

    while (wf.DayOfWeek != System.DayOfWeek.Monday)
        wf = wf.AddDays(-1);

    DateTime wl = wf.AddDays(6);

    label2.Text = "Дела на этой неделе " +
        wf.ToString("(dd/MM/yyyy -") +
        wl.ToString(" dd/MM/yyyy):");

    // формируем выборку
    System.Data.DataRow[] dr =
        dataSet1.Tables["journal"].Select("jdate >= #" +
        wf.Month.ToString() + "/" +
        wf.Day.ToString() + "/" +
        wf.Year.ToString() + "# and " +
        "jdate <= #" +
        wl.Month.ToString() + "/" +
        wl.Day.ToString() + "/" +
        wl.Year.ToString() + "#");
```

```
    // добавляем полученную выборку в таблицу
    dataSet1.Tables["journalSelect"].Clear();
    if (dr.Length > 0)
        for(int i = 0; i < dr.Length; i++)
            dataSet1.Tables["journalSelect"].Rows.
                Add(dr[i].ItemArray);
    else
        MessageBox.Show(
            "На этой неделе никаких дел не запланировано.",
            "Ежедневник", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
}

// щелчок на кнопке След. неделя
private void button4_Click(object sender, System.EventArgs e)
{
    // определяем дату начала и конца этой недели
    DateTime wf = DateTime.Today;

    // если сегодня понедельник
    if (wf.DayOfWeek == System.DayOfWeek.Monday)
        wf = wf.AddDays(7);
    else
        while (wf.DayOfWeek != System.DayOfWeek.Monday)
            wf = wf.AddDays(1);

    DateTime wl = wf.AddDays(6);

    label2.Text = "Дела на следующей неделе " +
        wf.ToString("(dd/MM/yyyy -") +
        wl.ToString(" dd/MM/yyyy):");

    // формируем выборку
    System.Data.DataRow[] dr =
        dataSet1.Tables["journal"].Select("jdate >= #" +
        wf.Month.ToString() + "/" +
        wf.Day.ToString() + "/" +
        wf.Year.ToString() + "# and " +
```

```
        "jdate <= #" +
        wl.Month.ToString() + "/" +
        wl.Day.ToString() + "/" +
        wl.Year.ToString() + "#");

    // добавляем полученную выборку в таблицу
    dataSet1.Tables["journalSelect"].Clear();
    if (dr.Length > 0)
        for(int i = 0; i < dr.Length; i++)
            dataSet1.Tables["journalSelect"].Rows.
                Add(dr[i].ItemArray);
    else
        MessageBox.Show(
            "На следующей неделе никаких дел
             не запланировано.",
            "Ежедневник", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
}

// щелчок на кнопке Все записи
private void button5_Click(object sender, System.EventArgs e)
{
    label2.Text = "Все записи ежедневника:";

    dataSet1.Tables["journalSelect"].Clear();

    for(int i = 0;
        i < dataSet1.Tables["journal"].Rows.Count; i++)
        dataSet1.Tables["journalSelect"].Rows.
            Add(dataSet1.Tables["journal"].Rows[i].ItemArray);
}

// щелчок на кнопке Добавить
private void button6_Click(object sender, System.EventArgs e)
{
    // выводим форму Добавить событие
    WinFormAdd af = new WinFormAdd();
    af.ShowDialog();
```

```
    // введено новое событие
    if (af.IsDtEventSet()) {
        // добавляем событие в dataSet1
        dataSet1.Tables["journal"].Rows.
        Add(new string[2]
            {af.dtime, af.dwhat});
    }
}

// завершение работы программы
private void WinForm_Closing(object sender,
   System.ComponentModel.CancelEventArgs e)
{
    // обновляем базу данных
    bdpDataAdapter1.Update(dataSet1.Tables["journal"]);
}


// форма Новое событие (WinFormAdd):
// параметры события
public string dtime = string.Empty;
public string dwhat = string.Empty;

// возвращает true, если заполнены поля dtime и dwhat
public Boolean IsDtEventSet() {
    if (dtime != string.Empty &&
        dwhat != string.Empty) return true;
    else return false;
}

// конструктор формы
public WinFormAdd()
{
    InitializeComponent();

    monthCalendar1.MaxSelectionCount = 1;
    textBox1.MaxLength = 60;
}
```

```
// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    // параметры события
    this.dtime =
        monthCalendar1.SelectionStart.ToShortDateString();
    this.dwhat = textBox1.Text;

    this.Close();
}
```

Чтение данных из xml-файла

Программа демонстрирует чтение данных из xml-файла. Форма программы приведена на рис. 1.36.

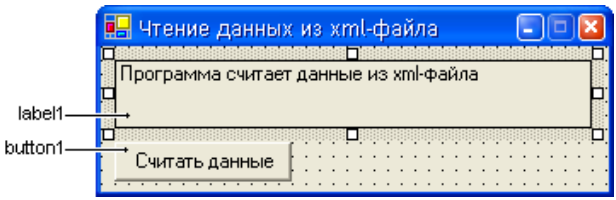

Рис. 1.36. Форма программы **Чтение данных из xml-файла**

```
// имя xml-файла или путь к нему
private string fname;

// XmlReader обеспечивает чтение данных из xml-файла
private XmlReader xmlReader;

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // если указывается только имя фала, предполагается,
    // что он находится в той же папке, что и программы.
    // Если это не так, указывается полный путь к файлу
    fname = "phones.xml";
```

```
    try{
        xmlReader = new XmlTextReader(fname);
        xmlReader.Read();
    }
    catch(Exception exc) {
        MessageBox.Show("Ошибка доступа к файлу.\n" +
            exc.ToString(), "Ошибка доступа к файлу",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
                this.Close();
    }
}

// щелчок на кнопке Считать данные
private void button1_Click(object sender, System.EventArgs e)
{
    label1.Text = string.Empty;

    if (!xmlReader.EOF) {
        // ищем узел "person"
        while (xmlReader.Name != "person") {
            xmlReader.Read();
            if (xmlReader.EOF) {
                label1.Text = "Достигнут конец файла";
                button1.Enabled = false;
                return;
            }
        }

        // входим в узел "person"
        xmlReader.Read();

        // считываем данные узла
        while (xmlReader.Name != "person") {
            if (xmlReader.NodeType == XmlNodeType.Text)
                label1.Text +=  xmlReader.Value + " ";
            xmlReader.Read();
        }
```

```
            // выходим из узла "person"
            xmlReader.Read();
        }
}
```

Конвертер базы данных Microsoft Access

Программа позволяет конвертировать базу данных Microsoft Access в xml-файл. Демонстрирует работу компонентов `DataSet`, `OpenFileDialog`, `SaveFileDialog`, работу с потоками. Форма и окно программы приведены на рис. 1.37 и 1.38.

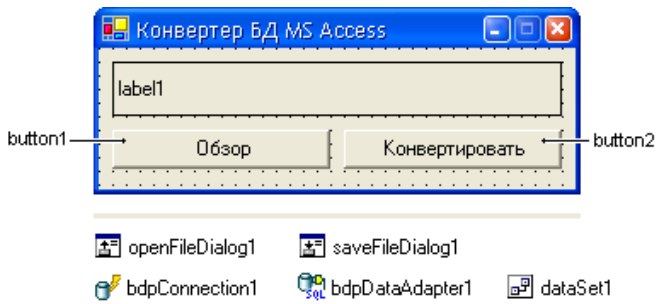

Рис. 1.37. Форма программы **Конвертер БД MS Access**

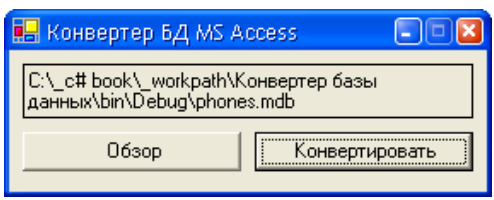

Рис. 1.38. Окно программы **Конвертер БД MS Access**

```
// строка соединения с БД
private string cString;

// опции соединения с БД
private string cOptions;
```

```
// конструктор формы
public WinForm()
{
    InitializeComponent();

    // настройка свойств компонента openFileDialog1
    openFileDialog1.DefaultExt = "mdb";
    openFileDialog1.Filter = "MSAccess|*.mdb";
    openFileDialog1.Title  = "Открыть файл БД";

    // настройка свойств компонента saveFileDialog1
    saveFileDialog1.DefaultExt = "xml";
    saveFileDialog1.Filter = "xml-файл|*.xml";
    saveFileDialog1.Title  = "Сохранить xml-файл";

    label1.Text = "Программа конвертирует
        базу данных Microsoft Access в xml-файл";

    cString  =
        "assembly=Borland.Data.Msacc, Version=2.5.0.0,
         Culture=neutral, PublicKeyToken=91d62ebb5b0d1b1b;
         vendorclient=msjet40.dll;pooling=True;
         grow on demand=True;username=admin;
         max pool size=100;password=;provider=MSAccess";
    cOptions = "transaction isolation=ReadCommitted;
         blobsize=1024";

    bdpDataAdapter1.SelectCommand.Connection =
        bdpConnection1;

    button2.Enabled = false;

    // создаем таблицу в dataSet1 для хранения базы данных
    dataSet1.Tables.Add("Table");
    dataSet1.DataSetName = "converted";
}

// щелчок на кнопке Обзор
private void button1_Click(object sender, System.EventArgs e)
```

```
{
    // отобразить диалог Открыть
    if (openFileDialog1.ShowDialog() == DialogResult.OK) {
        // отобразить имя файла БД (путь к нему)
        label1.Text = openFileDialog1.FileName;

        // закрываем соединение с предыдущей БД
        if (bdpConnection1.State == ConnectionState.Open)
            bdpConnection1.Close();

        // задаем строку соединения с базой данных
        bdpConnection1.ConnectionString =
            "database=" + openFileDialog1.FileName +
            ";" + cString;
        bdpConnection1.ConnectionOptions = cOptions;

        // имя базы данных
        string dbName = openFileDialog1.FileName.Substring(
            openFileDialog1.FileName.LastIndexOf("\\") + 1,
            openFileDialog1.FileName.Length - 5 -
            openFileDialog1.FileName.LastIndexOf("\\"));

        // запрос чтения данных
        bdpDataAdapter1.SelectCommand.CommandText =
            "select * from " + dbName;

        // обращаемся к базе
        try {
            // соединяемся с БД
            bdpConnection1.Open();

            dataSet1.Tables[0].Clear();
            dataSet1.Tables[0].TableName = dbName;

            bdpDataAdapter1.Fill(dataSet1.Tables[dbName]);

            if (!button2.Enabled) button2.Enabled = true;
        }
```

```
        catch(Exception exc){
            MessageBox.Show(
                "Ошибка доступа к базе данных.\n" +
                exc.ToString(),
                "Ошибка доступа к базе данных");
            button2.Enabled = false;
        }
    }
}

// щелчок на кнопке Конвертировать
private void button2_Click(object sender, System.EventArgs e)
{
    // отобразить диалог Сохранить
    if (saveFileDialog1.ShowDialog() ==
        DialogResult.OK) {

        // поток для записи
        System.IO.StreamWriter sw;

        // информация о файле
        System.IO.FileInfo fi =
            new System.IO.FileInfo(
                saveFileDialog1.FileName);

        try {
            // создать файл и связать с ним поток
            sw = fi.CreateText();

            // записать xml-файл
            sw.Write(dataSet1.GetXml());

            // закрыть поток
            sw.Close();

            MessageBox.Show(
                "База данных\n" +
                label1.Text +
                "\n успешно сконвертирована в\n" +
                saveFileDialog1.FileName + ".\n",
```

```
                "Конвертер базы данных",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }
        catch(Exception exc) {
            MessageBox.Show(exc.Message,
                "Ошибка конвертации БД.\n",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}
```

Каталог

Программа **Каталог** работает с базой данных catalogue, представляющей собой xml-файл. Подузлы узла `unit` (единица товара в БД catalogue) содержат такие поля, как: код товара `uCode`, наименование `uTitle`, цена `uPrice`, краткое описание `uMemo`, имя файла, иллюстрации `uImage`. Работа с xml-файлом БД осуществляется следующим образом: его содержимое записывается в таблицу `catalogue` компонента `dataSet1`, которая содержит коллекцию столбцов, имена которых соответствуют полям БД xml-файла, после чего все дальнейшие операции проводятся с таблицей `catalogue` компонента `dataSet1`. Добавление и настройка таблиц компонента `dataSet1` осуществляется при помощи свойства `Tables`. Форма и окно программы приведены на рис. 1.39 и 1.40. Значения свойств используемых компонентов `dataGrid1` и `textBox1`—`textBox5` приведены в табл. 1.16 и 1.17.

Таблица 1.16. *Значения свойств компонента* `dataGrid1`

Свойство	Значение
DataMember	catalogue
DataSource	dataSet1
CaptionVisible	False
ColumnHeaderVisible	True

Таблица 1.16 *(окончание)*

Свойство	Значение
TableStyles.dataGridTableStyle1.MappingName	catalogue
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.HeaderText	Код
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.Width	70
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn1.MappingName	uCode
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.HeaderText	Наименование
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.Width	140
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn2.MappingName	uTitle
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn3.HeaderText	Цена
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn3.Width	70
TableStyles.dataGridTableStyle1.GridColumnStyles. dataGridTextBoxColumn3.MappingName	uPrice

Таблица 1.17. *Значения свойства* `DataBindings` *полей* `textBox1` *—* `textBox5`

Свойство	Значение
textBox1.(DataBindings).Text	dataSet1 – catalogue.uTitle
textBox2.(DataBindings).Text	dataSet1 – catalogue.uMemo
textBox3.(DataBindings).Text	dataSet1 – catalogue.uPrice
textBox4.(DataBindings).Text	dataSet1 – catalogue.uCode
textBox5.(DataBindings).Text	dataSet1 – Catalogue.uImage

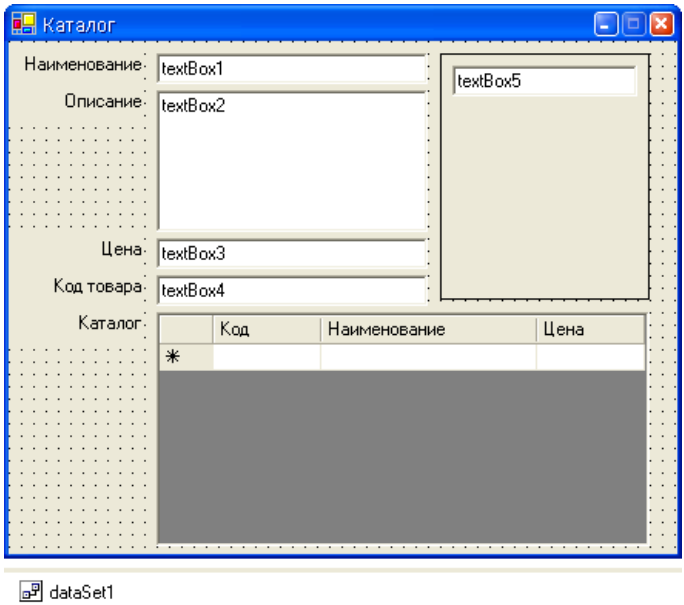

Рис. 1.39. Форма программы **Каталог**

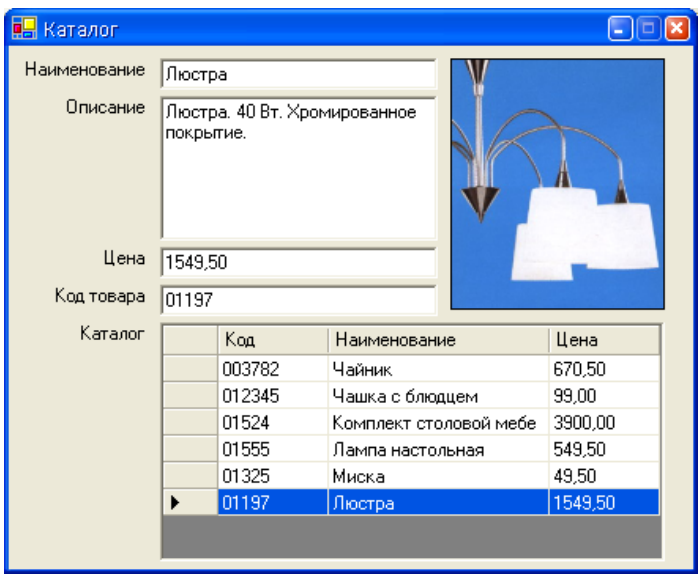

Рис. 1.40. Окно программы **Каталог**

```
// путь к каталогу xml-файла
private string fpath;

// имя xml-файла
private string fname;

// XmlReader обеспечивает чтение данных из xml-файла
private XmlReader xmlReader;

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    // если указывается только имя фала, предполагается,
    // что он находится в той же папке, что и программы.
    // Если это не так, указывается полный путь к файлу
    fpath = Application.StartupPath + "\\db_source\\";
    fname = "catalogue.xml";

    // поля формы доступны только для чтения
    textBox1.ReadOnly = true;
    textBox2.ReadOnly = true;
    textBox3.ReadOnly = true;
    textBox4.ReadOnly = true;

    textBox1.BackColor = System.Drawing.Color.White;
    textBox2.BackColor = System.Drawing.Color.White;
    textBox3.BackColor = System.Drawing.Color.White;
    textBox4.BackColor = System.Drawing.Color.White;

    dataGrid1.ReadOnly = true;

    // поле textBox5 содержит имя файла изображения
    textBox5.Visible = false;

    // задаем режим отображения иллюстрации
    // в компоненте pictureBox1
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
try{
    // получаем доступ к xml-документу
    xmlReader = new XmlTextReader(fpath + fname);
    xmlReader.Read();
}
catch(Exception exc) {
    MessageBox.Show("Ошибка доступа к файлу.\n" +
        exc.ToString(), "Ошибка доступа к файлу",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    this.Close();
}

// массив dRow содержит считанные данные
string[] dRow = new string[5];

// загружаем данные из xml-файла
while (!xmlReader.EOF) {
    // xmlReader.EOF показывает, достигнут ли конец
    // файла, true - достигнут, false - не достигнут

    // ищем узел "unit"
    while (xmlReader.Name != "unit") {
        xmlReader.Read();
        if (xmlReader.EOF) return;
    }

    // очищаем массив dRow
    for(int i=0; i< 5; i++) dRow[i] = string.Empty;

    // входим в узел "unit"
    xmlReader.Read();

    // считываем данные узла "unit"
    while (xmlReader.Name != "unit") {
        xmlReader.Read();

        // наименование uTitle
        if (xmlReader.Name == "uTitle") {
            xmlReader.Read();
```

```
                dRow[0] = xmlReader.Value;
                xmlReader.Read();
            }

            // описание uMemo
            if (xmlReader.Name == "uMemo") {
                xmlReader.Read();
                dRow[1] = xmlReader.Value;
                xmlReader.Read();
            }

            // цена uPrice
            if (xmlReader.Name == "uPrice") {
                xmlReader.Read();
                dRow[2] = xmlReader.Value;
                xmlReader.Read();
            }

            // код товара uCode
            if (xmlReader.Name == "uCode") {
                xmlReader.Read();
                dRow[3] = xmlReader.Value;
                xmlReader.Read();
            }

            // изображение uImage
            if (xmlReader.Name == "uImage") {
                xmlReader.Read();
                dRow[4] = xmlReader.Value;
                xmlReader.Read();
            }
        }

        // выходим из узла "unit"
        xmlReader.Read();

        // добавляем считанные данные в dataSet1
        dataSet1.Tables[0].Rows.Add(dRow);
    }
}
```

```
// изменился код товара
private void textBox4_TextChanged(object sender,
   System.EventArgs e)
{
    // загружаем соответствующее изображение
    try {
        pictureBox1.Image =
            new Bitmap(fpath + textBox5.Text);
    }
    catch {
        pictureBox1.Image = null;
    }
}
```

ИГРЫ И ДРУГИЕ ПОЛЕЗНЫЕ ПРОГРАММЫ

Парные картинки

Правила игры **Парные картинки** следующие: игровое поле разделено на клетки, за каждой из которых скрыта картинка. Каждая картинка имеет пару — точно такую же картинку. В начале игры все клетки закрыты. Щелчок мыши в клетке открывает картинку. Щелчок в другой клетке открывает вторую картинку. Если картинки в открытых клетках совпали, клетки исчезают, если нет, то они остаются открытыми. Очередной щелчок открывает новую картинку и закрывает уже открытые. Игра заканчивается, когда игрок найдет все пары картинок. Программа демонстрирует работу с графическими областями `Graphics`, битовыми образами `Bitmap`, использование кистей `Brush` и карандашей `Pen`. Рисунки, используемые программой, загружаются из файла. Окно программы приведено на рис. 1.41. Файл-источник с картинками приведен на рис. 1.42.

```
// объявление констант
private const int
    nw = 4,         // кол-во клеток по горизонтали
    nh = 4,         // кол-во клеток по вертикали
                            // произведение nw и nh
                            // должно быть кратно 2-м;
    np = (int)(nw*nh / 2);  // кол-во пар парных картинок

// рабочая графическая поверхность
private System.Drawing.Graphics g;

// картинки, загруженные из файла
private Bitmap pics;

// ширина и высота клетки
private int cw, ch;
```

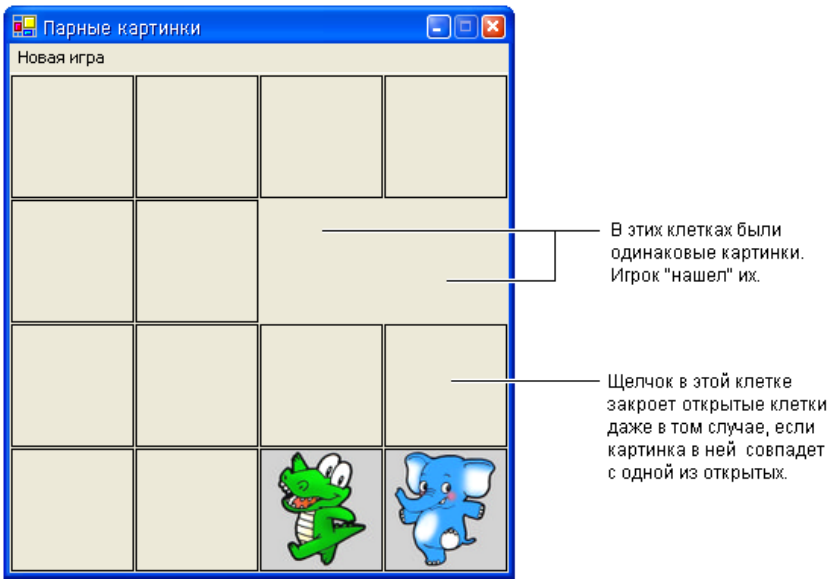

Рис. 1.41. Окно программы **Парные картинки**

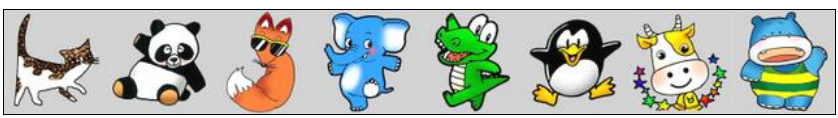

Рис. 1.42. Файл, содержащий изображения

```
// игровое поле
private int[,] field = new int[nw,nh];
// field[i,j] <  100 - код картинки, клетка закрыта;
// field[i,j] > 100 и < 200 - клетка открыта,
//                            игрок видит картинку;
// field[i,j] > 200 - игрок нашел пару для этой картинки

// кол-во открытых пар картинок
private int n;

// количество открытых в данный момент клеток
private int count;
```

```
// координаты 1-й открытой клетки
private int[] open1 = new int[2];

// координаты 2-й открытой клетки
private int[] open2 = new int[2];

// таймер
private System.Windows.Forms.Timer timer1;

// рисует клетку поля field[i,j]
private void cell(int i, int j) {
    int        // левый верхний угол клетки (координаты)
    x = i*(cw+2),
    y = j*(ch+2);

    if (field[i,j] > 200)
        // для этой клетки найдена пара,
        // ее нужно убрать с поля
        g.FillRectangle(SystemBrushes.Control,
            x, y, cw+2, ch+2);

    if ((field[i,j] > 100) && (field[i,j] < 200)) {
        // клетка открыта - вывести картинку
        g.DrawImage(pics,
            new Rectangle(x+1, y+1, cw, ch),
            new Rectangle((field[i,j]-101)*cw, 0, cw, ch),
            GraphicsUnit.Pixel);
        g.DrawRectangle(Pens.Black,
            x+1, y+1, cw, ch);
    }

    if ((field[i,j] > 0) && (field[i,j] < 100)) {
        // клетка закрыта
        g.FillRectangle(SystemBrushes.Control,
            x+1, y+1, cw, ch);
        g.DrawRectangle(Pens.Black,
            x+1, y+1, cw, ch);
    }
}
```

```
// отрисовывает игровое поле field
private void drawField() {
    for(int i = 0; i<nw; i++)
        for(int j = 0; j<nh ; j++)
            this.cell(i,j);
}

// новая игра
private void newGame() {
    // генератор случайных чисел
    Random rnd = new Random();
    int rndN;

    int[] buf = new int[np];
    // np - кол-во пар парных картинок;
    // в buf[i] записываем, сколько чисел i
    // (индентификаторы картинок) записали в массив field

    // сгенерируем игровое поле:
    // запишем в массив field случайные числа от 1 до k,
    // каждое число должно быть записано два раза
    for(int i = 0; i<nw; i++)
        for(int j = 0; j<nh ; j++){
            do {
                rndN = rnd.Next(np) + 1;
            } while(buf[rndN-1] == 2);

            field[i,j] = rndN;
            buf[rndN-1]++;
        }

    n = 0;
    count = 0;

    this.drawField();
}

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
```

```
{
    try {
        // загружаем файл с картинками
        pics = new Bitmap("psource.bmp");
    }
    catch(Exception exc) {
        MessageBox.Show("Файл 'psource.bmp' не найден.\n" +
            exc.ToString(), "Парные картинки",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        this.Close();
        return;
    }

    // определяем высоту и ширину клетки игрового поля
    cw = (int)(pics.Width / np);
    ch = pics.Height;

    // ширина и высота рабочей области формы
    this.ClientSize =
        new System.Drawing.Size(nw*(cw+2) + 1, nh*(ch+2) + 1);

    // рабочая графическая поверхность
    g = this.CreateGraphics();

    timer1.Enabled  = false;
    timer1.Interval = 200;

    this.newGame();
}

// щелчок мыши на игровом поле
private void WinForm_MouseDown(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    int
        i = (int)(e.X / cw),  // индекс клетки по горизонтали
        j = (int)(e.Y / ch);  // и по вертикали (с нуля)
```

```
// таймер запущен, т. е. только что была открыта
// пара одинаковых картинок, но они еще не "стерты";
// щелчок мыши производится на одной из этих картинок
if ((timer1.Enabled) && (field[i,j] > 200)) return;

// щелчок на месте одной из двух уже найденных
// парных картинок
if (field[i,j] > 200) return;

// открытых клеток нет
if (count == 0) {
    count++;

    // записываем координаты 1-й открытой клетки
    open1[0] = i; open1[1] = j;

    // клетка помечается как открытая
    field[i,j] += 100;

    // отрисовать клетку
    this.cell(i,j);

    return;
}

// открыта одна клетка, надо открыть вторую
if (count == 1) {
    // записываем координаты 2-й открытой клетки
    open2[0] = i; open2[1] = j;

    // если открыта одна клетка, и щелчок сделан
    // в той же клетке, ничего не происходит
    if ((open1[0] == open2[0]) && (open1[1] == open2[1]))
        return;
    else {
        // теперь открыты две клетки
        count++;

        // клетка помечается как открытая
        field[i,j] += 100;
```

```
            // отрисовать клетку
            this.cell(i,j);

            // открыты 2 одинаковые картинки
            if (field[open1[0],open1[1]] ==
                field[open2[0],open2[1]]) {
                n++;

                // в массиве field клетки помечаются
                // как совпавшие
                field[open1[0],open1[1]] += 100;
                field[open2[0],open2[1]] += 100;

                count = 0;

                // запускаем таймер - процедура обработки
                // события Timer "сотрет" одинаковые картинки
                timer1.Enabled = true;
            }
        }
        return;
    }

    // открыты 2 клетки с разными картинками, закроем
    // их и откроем новую, в которой сделан щелчок
    if (count == 2) {
        // закрываем открытые клетки
        field[open1[0],open1[1]] -= 100;
        field[open2[0],open2[1]] -= 100;

        this.cell(open1[0],open1[1]);
        this.cell(open2[0],open2[1]);

        // записываем в open1 номер текущей клетки
        open1[0] = i; open1[1] = j;

        count = 1;          // счетчик открытых клеток

        // открыть текущую клетку
        field[i,j] += 100;
```

```
        this.cell(i,j);
    }
}

// обработка события таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    // отрисовать клетки
    this.cell(open1[0],open1[1]);
    this.cell(open2[0],open2[1]);

    // остановка таймера
    timer1.Enabled = false;

    if (n == np)
        // открыты все пары
        g.DrawString("Game over! Thanks for playing.",
            new Font("Tahoma", 10, FontStyle.Bold),
            Brushes.Black, 5, 5);
}

// обработка события Paint
private void WinForm_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
    // отрисовать игровое поле
    this.drawField();
}

// щелчок на пункте меню Новая игра
private void menuItem1_Click(object sender, System.EventArgs
   e)
{
    g.FillRectangle(SystemBrushes.Control,
        0, 0, nw*(cw+2), nh*(ch+2));
    this.newGame();
}
```

Собери картинку

Программа **Собери картинку** — графический вариант известной игры "15". Ее цель — расположить фрагменты картинки в правильном порядке. Окно программы приведено на рис. 1.43.

Рис. 1.43. Окно программы **Собери картинку**

```
// размер игрового поля
private const int nw = 4, nh = 4;

// рабочая графическая поверхность
private System.Drawing.Graphics g;

// картинка, загружаемая из файла
private Bitmap pics;

// ширина и высота клетки
private int cw, ch;

// игровое поле
private int[,] field = new int[nw,nh];
```

```
// координаты пустой клетки
private int ex, ey;

// показывает, отображаются ли номера фишек
private Boolean showNumbers = false;

// новая игра
private void newGame() {
    // располагаем фишки в правильном порядке
    for(int j = 0; j < nh; j++)
        for(int i = 0; i < nw; i++)
            field[i,j] = j*nw + i + 1;

    // последняя фишка - пустая
    field[nw-1, nh-1] = 0;
    ex = nw-1; ey = nh-1;

    this.mixer();        // перемешиваем фишки
    this.drawField();    // выводим игровое поле
}

// процедура перемешивает фишки
private void mixer() {
    int x, y,  // эту клетку переместить в пустую
           d;  // направление перемещения относительно пустой

    // генератор случайных чисел
    Random rnd = new Random();

    for(int i = 0; i < nw * nh * 10; i++)
    // nw * nh * 10 - кол-во перестановок
    {
        x = ex;
        y = ey;

        d = rnd.Next(4);
        switch (d) {
            case 0: if (x > 0)    x--; break;
            case 1: if (x < nw-1) x++; break;
```

```
            case 2: if (y > 0)    y--; break;
            case 3: if (y < nh-1) y++; break;
        }

        // здесь определили фишку, которую
        // нужно переместить в пустую клетку
        field[ex,ey] = field[x,y];
        field[x,y] = 0;

        // запоминаем координаты пустой фишки
        ex = x; ey = y;
    }
}

// процедура отрисовывает поле field
private void drawField() {
    // содержимое клеток
    for(int i = 0; i < nw; i++)
        for(int j = 0; j < nh; j++) {
            if (field[i,j] != 0)
                // выводим фишку с картинкой:
                // ( ((field[i,j] - 1) % nw) * cw,
                //   (int)((field[i,j] - 1) / nw) * ch ) -
                // координаты левого верхнего угла
                // области файла-источника картинки
                g.DrawImage(pics,
                    new Rectangle(i*cw, j*ch, cw, ch),
                    new Rectangle(
                        ((field[i,j] - 1) % nw) * cw,
                        (int)((field[i,j] - 1) / nw) * ch,
                        cw, ch),
                    GraphicsUnit.Pixel);
            else
                // выводим пустую фишку
                g.FillRectangle(SystemBrushes.Control,
                    i*cw, j*ch, cw, ch);

            // рисуем границу
            g.DrawRectangle(Pens.Black,
                i*cw, j*ch, cw, ch);
```

```
            // номер фишки
            if ((showNumbers) && field[i,j] != 0)
                g.DrawString(Convert.ToString(field[i,j]),
                    new Font("Tahoma",10,FontStyle.Bold),
                    Brushes.Black, i*cw + 5, j*ch + 5);
        }
}

// проверяет, расположены ли фишки в правильном порядке
private Boolean finish() {
    int i = 0, j = 0,
        c;    // счетчик фишек

    for(c = 1; c < nw*nh; c++) {
        if (field[i,j] != c) return false;

        // к следующей клетке
        if (i < nw - 1) i++;
        else { i = 0; j++; }
    }
    return true;
}

// "перемещает" фишку в соседнюю пустую клетку,
// если она есть, конечно;
// (cx, cy) - клетка, в которой сделан щелчок,
// (ex, ey) - пустая клетка
private void move(int cx, int cy) {
    // проверим, возможен ли обмен
    if (!(( Math.Abs(cx - ex) == 1) && (cy - ey == 0)) ||
        (( Math.Abs(cy - ey) == 1) && (cx - ex == 0))))
        return;

    // обмен. переместим фишку из (x, y) в (ex, ey)
    field[ex,ey] = field[cx,cy];
    field[cx,cy] = 0;

    ex = cx; ey = cy;
```

```
    // отрисовать поле
    this.drawField();

    if (this.finish()) {
        field[nw-1, nh-1] = nh*nw;
        this.drawField();

        // игра закончена. сыграть еще раз?
        // No  - завершить работу программы,
        // Yes - новая игра
        if (MessageBox.Show("Congratulations!\n" +
                  "Еще раз?", "Собери картинку",
                   MessageBoxButtons.YesNo,
                   MessageBoxIcon.Question)
                   == System.Windows.Forms.DialogResult.No)
            this.Close();
        else this.newGame();
    }
}

// загрузка формы
private void WinForm_Load(object sender, System.EventArgs e)
{
    try {
        // загружаем файл картинок
        pics = new Bitmap("psource.bmp");
    }
    catch(Exception exc) {
        MessageBox.Show("Файл 'psource.bmp' не найден.\n" +
            exc.ToString(), "Файл не найден",
            MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        this.Close();
        return;
    }

    // определяем высоту и ширину клетки (фишки)
    cw = (int)(pics.Width  / nw);
    ch = (int)(pics.Height / nh);
```

```
    // ширина и высота рабочей области формы
    this.ClientSize =
        new System.Drawing.Size(cw * nw + 1, ch * nh + 1);

    // рабочая графическая поверхность
    g = this.CreateGraphics();

    this.newGame();
}

// щелчок мыши на игровом поле
private void WinForm_MouseDown(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    // преобразуем координаты мыши в координаты клетки
    this.move((int)(e.X / cw), (int)(e.Y / ch));
}

// обработка события Paint
private void WinForm_Paint(object sender,
   System.Windows.Forms.PaintEventArgs e)
{
    // отрисовать игровое поле
    this.drawField();
}

// щелчок на пункте меню Новая игра
private void menuItem1_Click(object sender, System.EventArgs
   e)
{
    g.FillRectangle(SystemBrushes.Control,
        0, 0, cw * nw + 1, ch * nh + 1);
    this.newGame();
}

// щелчок на пункте меню Показать/Скрыть номера фишек
private void menuItem2_Click(object sender,
System.EventArgs e)
{
    if (showNumbers) {
        showNumbers = false;
```

```
        menuItem2.Text = "Показать номера фишек";
        this.drawField();
    }
    else {
        showNumbers = true;
        menuItem2.Text = "Скрыть номера фишек";
        this.drawField();
    }
}
```

Сапер

Игра **Сапер**, окно которой приведено на рис. 1.44, — аналог одноименной игры, хорошо знакомой пользователям Windows. Программа демонстрирует работу с графикой, массивами, вывод справочной информации. Главная форма и форма **О программе** приведены на рис. 1.45 и 1.46 соответственно. Окно **О программе** появляется на экране в результате выбора в меню **Справка** команды **О программе**.

```
private const int
    MR = 10,    // кол-во клеток по вертикали
    MC = 10,    // кол-во клеток по горизонтали
    NM = 10,    // кол-во мин
    W  = 40,    // ширина и
    H  = 40;    // высота клетки поля

// минное поле
private int[,] Pole = new int[MR+2, MC+2];
// значение элемента массива:
// 0..8 - количество мин в соседних клетках
// 9 - в клетке мина
// 100..109 - клетка открыта
// 200..209 - в клетку поставлен флаг

private int nMin,    // кол-во найденных мин
            nFlag;   // кол-во поставленных флагов
```

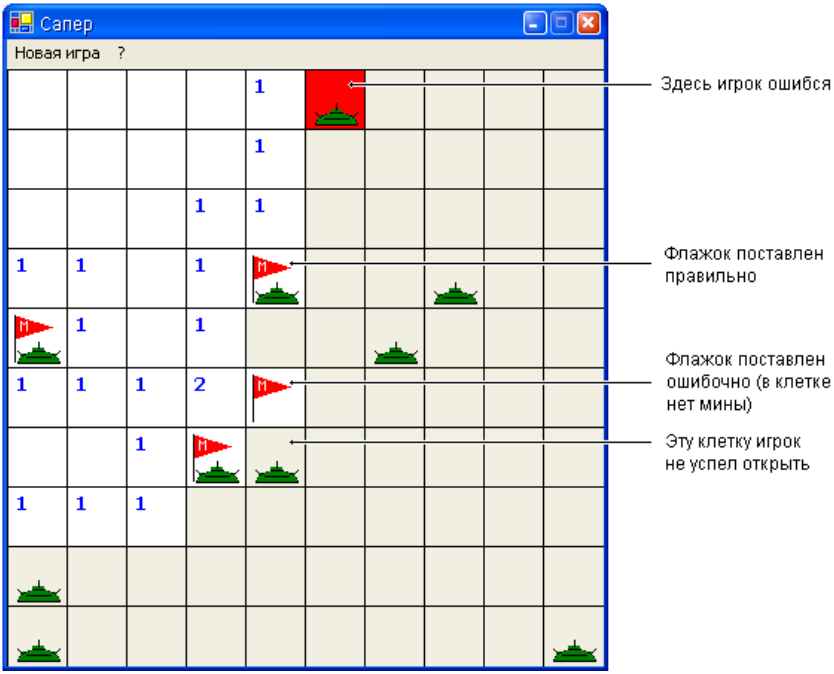

Рис. 1.44. Окно программы **Сапер**

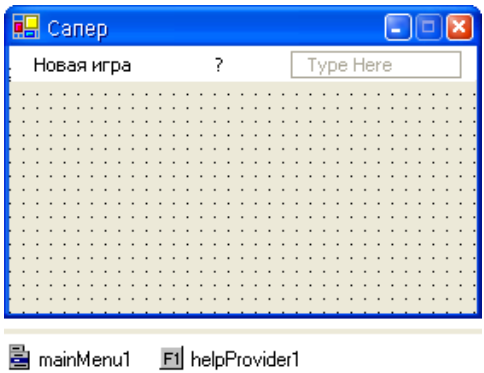

Рис. 1.45. Главная форма программы **Сапер**

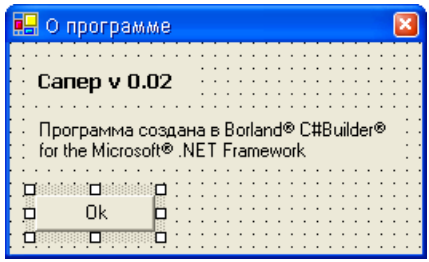

Рис. 1.46. Форма окна **О программе**

```
// статус игры
private int status;
// 0 - начало игры,
// 1 - игра,
// 2 - результат

// новая игра
private void newGame() {
    int row, col;     // индексы клетки
    int n = 0;        // количество поставленных мин
    int k;            // кол-во мин в соседних клетках

    // обнулим массив минного поля Pole
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            Pole[row,col] = 0;

    // генератор случайных чисел
    Random rnd = new Random();

    // расставим мины
    do{
        row = rnd.Next(MR) + 1;
        col = rnd.Next(MC) + 1;

        if (Pole[row,col] != 9) {
            Pole[row,col] = 9;
            n++;
        }
    }
    while (n != NM);
```

```
    // для каждой клетки вычислим кол-во мин
    // в соседних клетках
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            if (Pole[row,col] != 9) {
                k = 0;

                if (Pole[row-1,col-1] == 9) k++;
                if (Pole[row-1,col]   == 9) k++;
                if (Pole[row-1,col+1] == 9) k++;
                if (Pole[row,col-1]   == 9) k++;
                if (Pole[row,col+1]   == 9) k++;
                if (Pole[row+1,col-1] == 9) k++;
                if (Pole[row+1,col]   == 9) k++;
                if (Pole[row+1,col+1] == 9) k++;

                Pole[row,col] = k;
            }

    status = 0;       // начало игры
    nMin   = 0;       // нет обнаруженных мин
    nFlag  = 0;       // нет поставленных флагов
}

// отрисовывает поле
private void showPole(Graphics g, int status) {
    for(int row = 1; row <= MR; row++)
        for(int col = 1; col <= MC; col++)
            this.kletka(g, row, col, status);
}

// выводит содержимое клетки
private void kletka(Graphics g,
    int row, int col, int status) {

    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;
```

```
    // неоткрытые клетки - серые
    if (Pole[row,col] < 100)
        g.FillRectangle(SystemBrushes.ControlLight,
            x-1, y-1, x+W, y+H);

    // открытые или помеченные клетки
    if (Pole[row,col] >= 100) {

        // открываем клетку, открытые - белые
        if (Pole[row,col] != 109)
            g.FillRectangle(Brushes.White,
                x-1, y-1, x+W, y+H);
        else
            // на этой мине подорвались!
            g.FillRectangle(Brushes.Red,
                x-1, y-1, x+W, y+H);

        // если рядом с клеткой есть мины,
        // подписываем значение (их количество)
        if ((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
            g.DrawString((Pole[row,col]-100).ToString(),
                new Font("Tahoma", 10,
                    System.Drawing.FontStyle.Bold),
                Brushes.Blue, x+3, y+2);
    }

    // в клетке поставлен флаг
    if (Pole[row,col] >= 200)
        this.flag(g, x, y);

    // рисуем границу клетки
    g.DrawRectangle(Pens.Black,
        x-1, y-1, x+W, y+H);

    // если игра завершена (status = 2),
    // показываем мины
    if ((status == 2) && ((Pole[row,col] % 10) == 9))
        this.mina(g, x, y);
}
```

```
// открывает текущую и все соседние клетки,
// в которых нет мин
private void open(int row, int col) {
    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    if (Pole[row,col] == 0) {
        Pole[row,col] = 100;

        // отобразить содержимое клетки
        this.Invalidate(new Rectangle(x, y, W, H));

        // открыть примыкающие клетки
        // слева, справа, сверху, снизу
        this.open(row, col-1);
        this.open(row-1, col);
        this.open(row, col+1);
        this.open(row+1, col);

        //примыкающие диагонально
        this.open(row-1,col-1);
        this.open(row-1,col+1);
        this.open(row+1,col-1);
        this.open(row+1,col+1);
    }
else
    if ((Pole[row,col] < 100) &&
        (Pole[row,col] != -3)) {
        Pole[row,col] += 100;

        // отобразить содержимое клетки
        this.Invalidate(new Rectangle(x, y, W, H));
    }
}

// рисует мину
private void mina(Graphics g, int x, int y) {
```

```
    // корпус
    g.FillRectangle(Brushes.Green,
        x+16, y+26, 8, 4);
    g.FillRectangle(Brushes.Green,
        x+8, y+30, 24, 4);
    g.DrawPie(Pens.Black,
        x+6, y+28, 28, 16, 0, -180);
    g.FillPie(Brushes.Green,
        x+6, y+28, 28, 16, 0, -180);

    // полоса на корпусе
    g.DrawLine(Pens.Black,
        x+12, y+32, x+28, y+32);

    // вертикальный "ус"
    g.DrawLine(Pens.Black,
        x+20, y+22, x+20, y+26);

    // боковые "усы"
    g.DrawLine(Pens.Black,
        x+8, y+30, x+6, y+28);
    g.DrawLine(Pens.Black,
        x+32, y+30, x+34, y+28);
}

// рисует флаг
private void flag(Graphics g, int x, int y) {
    Point[] p = new Point[3];
    Point[] m = new Point[5];

    // флажок
    p[0].X = x+4;   p[0].Y = y+4;
    p[1].X = x+30;  p[1].Y = y+12;
    p[2].X = x+4;   p[2].Y = y+20;
    g.FillPolygon(Brushes.Red, p);

    // древко
    g.DrawLine(Pens.Black,
        x+4, y+4, x+4, y+35);
```

```
    // буква M на флажке
    m[0].X = x+8;   m[0].Y = y+14;
    m[1].X = x+8;   m[1].Y = y+8;
    m[2].X = x+10;  m[2].Y = y+10;
    m[3].X = x+12;  m[3].Y = y+8;
    m[4].X = x+12;  m[4].Y = y+14;
    g.DrawLines(Pens.White, m);
}

// конструктор формы
public WinForm()
{
    InitializeComponent();

    // назначаем файл справки
    helpProvider1.HelpNamespace = "sapper.chm";

    // В неотображаемые эл-ты массива, соответствующие
    // клеткам на границе игрового поля, запишем число -3.
    // Это значение используется процедурой open()
    // для завершения рекурсивного процесса открытия
    // соседних пустых клеток
    for(int row = 0; row <= MR+1; row++) {
        Pole[row,0] = -3;
        Pole[row,MC+1] = -3;
    }

    for(int col = 0; col <= MC+1; col++) {
        Pole[0,col] = -3;
        Pole[MR+1,col] = -3;
    }

    // новая игра
    this.newGame();

    // устанавливаем размер формы в соответствии
    // с размером игрового поля
    this.ClientSize = new Size(W*MC + 1, H*MR + 1);
}
```

```
// нажатие кнопки мыши на игровом поле
private void WinForm_MouseDown(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    // игра завершена
    if (status == 2) return;

    // первый щелчок
    if (status == 0) status = 1;

    // преобразуем координаты мыши в индексы
    // клетки поля, в которой был сделан щелчок;
    // (e.X, e.Y) - координаты точки формы,
    // в которой была нажата кнопка мыши;
    int row = (int)(e.Y/H) + 1,
        col = (int)(e.X/W) + 1;

    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    // щелчок левой кнопки мыши
    if (e.Button == MouseButtons.Left) {

        // открыта клетка, в которой есть мина
        if (Pole[row,col] == 9) {
            Pole[row,col] += 100;

            // игра закончена
            status = 2;

            // перерисовать форму
            this.Invalidate();
        }
        else
            if (Pole[row,col] < 9)
                this.open(row,col);
    }
```

```
    // щелчок правой кнопки мыши
    if (e.Button == MouseButtons.Right) {

        // в клетке не было флага, ставим его
        if (Pole[row,col] <= 9) {
            nFlag += 1;

            if (Pole[row,col] == 9)
                nMin += 1;

            Pole[row,col] += 200;

            if ((nMin == NM) && (nFlag == NM)) {
                // игра закончена
                status = 2;

                // перерисовываем все игровое поле
                this.Invalidate();
            }
            else
                // перерисовываем только клетку
                this.Invalidate(new Rectangle(x, y, W, H));
        }

        else
            // в клетке был поставлен флаг,
            // повторный щелчок правой кнопки мыши
            // убирает его и закрывает клетку
            if (Pole[row,col] >= 200) {
                nFlag -= 1;
                Pole[row,col] -= 200;

                // перерисовываем клетку
                this.Invalidate(new Rectangle(x, y, W, H));
            }
    }
}
```

```
// щелчок на пункте меню Новая игра
private void menuItem1_Click(object sender,
System.EventArgs e)
{
    this.newGame();
    this.Invalidate();
}

// щелчок на пункте меню Справка
private void menuItem3_Click(object sender,
System.EventArgs e)
{
    Help.ShowHelp(this,
        this.helpProvider1.HelpNamespace);
}

// щелчок на пункте меню О программе
private void menuItem4_Click(object sender,
System.EventArgs e)
{
    WinFormAbout af = new WinFormAbout();
    af.ShowDialog();
}

// обработка события Paint
private void WinForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    this.showPole(e.Graphics, status);
}
```

Master Mind

Программа **Master Mind** (цветовая криптограмма) является аналогом игры **Угадай число**, только вместо числа угадывается цветовая комбинация. Игрок видит на экране набор фишек. Установить цвет фишки можно при помощи палитры или щелчком на фишке. После того как цветовая последовательность установлена,

нужно нажать кнопку **Ok**. Если введенная последовательность совпала с загаданной, игра заканчивается. Если нет, последовательность добавляется на панель истории введенных последовательностей, где рядом с ней указывается количество угаданных цветов и количество фишек, находящихся на своих местах. Игрок может устанавливать уровень (длину последовательности) и сложность (количество цветов) игры. Окно программы приведено на рис. 1.47.

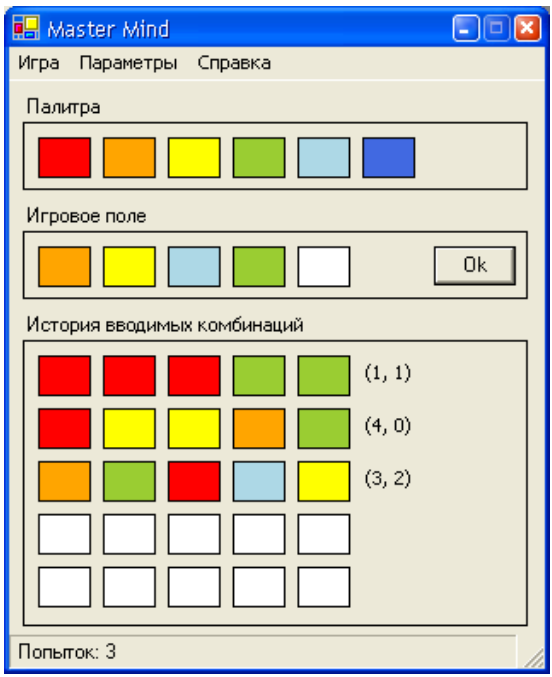

Рис. 1.47. Окно программы **Master Mind**

```
// объявление констант и переменных
private const int
    nN = 6,      // максимальная длина последовательности
    cN = 7,      // максимальное число цветов палитры
    fh = 23,     // высота и
    fw = 30;     // ширина фишки
```

```
private int
    conL,       // уровень
                // (длина последовательности)
    conC,       // сложность (количество цветов
                // последовательности)
    t,          // количество сделанных попыток
    ind;        // индекс выбранной фишки

// загаданная последовательность
private int[] n = new int[nN];

// последовательность игрока
private int[] p = new int[nN];

// история (5 последних) сделанных ходов:
// h[0]..h[nN-1] - последовательность,
// h[nN] - количество угаданных цветов фишек,
// h[nN+1] - количество фишек на своих местах
private int[,] h = new int[5, nN+2];

// палитра цветов фишек
private Brush[] c = new Brush[cN+1];

private Graphics
    g1,         // рабочая поверхность палитры цветов
    g2,         // рабочая поверхность игрового поля
    g3;         // рабочая поверхность истории сделанных ходов

// создание новой игры
private void newGame() {
    // генератор случайных чисел
    Random rnd = new Random();

    // количество сделанных попыток
    t = 0;

    for(int i = 0; i < conL; i++) {
        // генерируем цветовую последовательность
        n[i] = rnd.Next(conC) + 1;
```

```
        // обнуляем последовательность игрока
        p[i] = 0;
    }

    // обнуляем историю от предыдущей игры
    for(int i = 0; i < 5; i++) {
        for(int j = 0; j < conL; j++)
            h[i,j] = 0;

        h[i,conL]   = -1;
        h[i,conL+1] = -1;
    }

    // индекс выбранной фишки
    ind = 0;

    // обновляем палитру
    g1.Clear(System.Drawing.SystemColors.Control);
    this.drawPalette();

    // выводим игровое поле
    g2.Clear(System.Drawing.SystemColors.Control);
    this.drawInField();

    // история сделанных ходов
    g3.Clear(System.Drawing.SystemColors.Control);
    this.drawPrevious();

    statusBar1.Panels[0].Text = " Попыток: 0";
}

// выводит палитру цветов фишек
private void drawPalette() {
    for(int i = 1; i <= conC; i++) {
        g1.FillRectangle(c[i],
            8+(i-1)*(fw+8), 8, fw, fh);
        g1.DrawRectangle(Pens.Black,
            8+(i-1)*(fw+8), 8, fw, fh);
    }
}
```

```
// выводит игровое поле
private void drawInField() {
    for(int i = 0; i < conL; i++) {
        g2.FillRectangle(c[p[i]],
            8+i*(fw+8), 8, fw, fh);
        g2.DrawRectangle(Pens.Black,
            8+i*(fw+8), 8, fw, fh);
    }
}

// выводит историю предыдущих ходов
private void drawPrevious() {
    for(int i = 0; i < 5; i++) {
        for(int j = 0; j < conL; j++) {
            g3.FillRectangle(c[h[i,j]],
                8 + j*(fw+8), (fh+8)*4+8 - (fh+8)*i, fw, fh);
            g3.DrawRectangle(Pens.Black,
                8 + j*(fw+8), (fh+8)*4+8 - (fh+8)*i, fw, fh);
        }

        if ((h[i,conL] != -1) && (h[i,conL+1] != -1)) {
            g3.FillRectangle(SystemBrushes.Control,
                8 + (conL)*(fw+8), (fh+8)*4+8 - (fh+8)*i,
                60, fh);
            g3.DrawString(
                "(" + h[i,conL].ToString() + ", " +
                h[i,conL+1].ToString() + ")",
                new Font("Tahoma", 8),
                Brushes.Black,
                8 + (conL)*(fw+8), (fh+8)*4+11 - (fh+8)*i);
        }
    }
}

// конструктор формы
public WinForm()
{
    InitializeComponent();
```

```
// параметры окна приложения
this.FormBorderStyle =
    FormBorderStyle.FixedSingle;
this.MaximizeBox = false;

// палитра цветов
c[0] = Brushes.White;
c[1] = Brushes.Red;
c[2] = Brushes.Orange;
c[3] = Brushes.Yellow ;
c[4] = Brushes.YellowGreen;
c[5] = Brushes.LightBlue;
c[6] = Brushes.RoyalBlue;
c[7] = Brushes.Purple;

// настройка меню Уровень (3 фишки)
menuItem8.Checked = true;

// настройка меню Сложность (4 цвета)
menuItem12.Checked = true;

conL = 3;    // длина последовательности
conC = 4;    // количество используемых цветов

// графические поверхности
g1 = this.panel1.CreateGraphics();
g2 = this.panel2.CreateGraphics();
g3 = this.panel3.CreateGraphics();

// назначение функций обработки события
// щелчка мыши Click на подпунктах меню Уровень
this.menuItem8.Click  += new
    System.EventHandler(this.MenuItemSetL_Click);
this.menuItem9.Click  += new
    System.EventHandler(this.MenuItemSetL_Click);
this.menuItem10.Click += new
    System.EventHandler(this.MenuItemSetL_Click);
this.menuItem11.Click += new
    System.EventHandler(this.MenuItemSetL_Click);
```

```
    // назначение функций обработки события
    // щелчка мыши Click на пунктах подменю Сложность
    this.menuItem12.Click += new
        System.EventHandler(this.MenuItemSetC_Click);
    this.menuItem13.Click += new
        System.EventHandler(this.MenuItemSetC_Click);
    this.menuItem14.Click += new
        System.EventHandler(this.MenuItemSetC_Click);
    this.menuItem15.Click += new
        System.EventHandler(this.MenuItemSetC_Click);

    // назначаем файл справки
    helpProvider1.HelpNamespace = "master_mind.chm";

    // новая игра
    this.newGame();
}

// щелчок на пункте подменю Уровень
private void MenuItemSetL_Click(object sender,
   System.EventArgs e)
{
    // выбранный пункт меню
    MenuItem menuItem = (MenuItem)sender;

    // пользователь изменил уровень
    if (menuItem.Checked == false) {
        // снимаем галочку с предыдущего значения
        switch (conL) {
            case 3:
                menuItem8.Checked = false;
                break;
            case 4:
                menuItem9.Checked = false;
                break;
            case 5:
                menuItem10.Checked = false;
                break;
```

```
            case 6:
                menuItem11.Checked = false;
                break;
        }

        conL = Convert.ToInt16(
            menuItem.Text.Substring(0,1));

        // устанавливаем галочку на новое значение
        switch (conL) {
            case 3:
                menuItem8.Checked = true;
                break;
            case 4:
                menuItem9.Checked = true;
                break;
            case 5:
                menuItem10.Checked = true;
                break;
            case 6:
                menuItem11.Checked = true;
                break;
        }

        // новая игра
        this.newGame();
    }
}

// щелчок на пункте подменю Сложность
private void MenuItemSetC_Click(object sender,
   System.EventArgs e)
{
    // выбранный пункт меню
    MenuItem menuItem = (MenuItem)sender;

    // пользователь изменил сложность
    if (menuItem.Checked == false) {
        // снимаем галочку с предыдущего значения
        switch (conC) {
```

```
            case 4:
                menuItem12.Checked = false;
                break;
            case 5:
                menuItem13.Checked = false;
                break;
            case 6:
                menuItem14.Checked = false;
                break;
            case 7:
                menuItem15.Checked = false;
                break;
        }

        conC = Convert.ToInt16(
            menuItem.Text.Substring(0,1));

        // устанавливаем галочку на новое значение
        switch (conC) {
            case 4:
                menuItem12.Checked = true;
                break;
            case 5:
                menuItem13.Checked = true;
                break;
            case 6:
                menuItem14.Checked = true;
                break;
            case 7:
                menuItem15.Checked = true;
                break;
        }

        // новая игра
        this.newGame();
    }
}
```

```
// нажата кнопка мыши в области палитры цветов
// (устанавливает цвет выбранной фишки)
private void panel1_MouseDown(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    int ci = -1;    // выбранный цвет палитры

    if ((e.Y > 8) && (e.Y < 8+fh))
        if ((e.X % (fw+8)) > 8)
            ci = (int)(e.X/(fw+8)) + 1;

    if (ci != -1) {
        // изменяем цвет фишки
        p[ind] = ci;

        // перерисовываем игровое поле
        this.drawInField();

        // увеличиваем индекс выбранной фишки
        ind = (ind+1) % conL;
    }
}

// нажата кнопка мыши в области игрового поля
// (при щелчке на фишке ее цвет меняется
// на следующий по списку)
private void panel2_MouseDown(object sender,
   System.Windows.Forms.MouseEventArgs e)
{
    int pi = -1;    // индекс нажимаемой фишки

    if ((e.Y > 8) && (e.Y < 8+fh))
        if ((e.X % (fw+8)) > 8)
            pi = (int)(e.X/(fw+8));

    if (pi != -1) {
        // изменяем цвет фишки
        p[pi] = (p[pi] % conC) + 1;

        // перерисовываем игровое поле
        this.drawInField();
```

```
        // увеличиваем индекс выбранной фишки
        ind = (pi+1) % conL;
    }
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    int rn = 0,         // угаданные цифры
        pn = 0;         // цифры на правильных позициях

    // учет угаданных цифр
    Boolean[] r = new Boolean[nN];

    for(int i = 0; i < conL; i++)
        if (p[i] == 0) {
            MessageBox.Show(
                "Неполная последовательность.\n" +
                "Некоторые фишки пустые.",
                "Mind Master");

            return;
        }

    t++;
    statusBar1.Panels[0].Text =
        " Попыток: " + t.ToString();

    ind = 0;

    // считаем угаданные цвета и позиции
    for(int i = 0; i <  conL; i++) {
        for(int j = 0; j < conL; j++)
            if ((p[i] == n[j]) && (!r[j])) {
                rn++;
                r[j] = true;
                break;
            }
```

```
        if (p[i] == n[i])
            pn++;
    }

    // последовательность угадана
    if (pn == conL) {
        if (MessageBox.Show("Congratulations!\n" +
            "Последовательность угадана!\n" +
            "Совершено попыток: " + t.ToString() +
            "\nAnother game?", "Mind Master",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question)
            == System.Windows.Forms.DialogResult.No)
            this.Close();
        else
            this.newGame();
    }
    else {
        // добавляем последовательность в массив истории
        for(int i = 0; i < 4; i++)
            for(int j = 0; j < conL + 2; j++)
                h[i,j] = h[i+1,j];

        for(int j = 0; j < conL; j++)
            h[4,j] = p[j];

        h[4,conL]   = rn;
        h[4,conL+1] = pn;

        // обновляем историю вводимых комбинаций
        this.drawPrevious();

        // обнуляем последовательность игрока
        for(int i = 0; i < conL; i++)
            p[i] = 0;

        // обновляем игровое поле
        this.drawInField();
    }
}
```

```
// обработка события Paint
private void WinForm_Paint(object sender,
System.Windows.Forms.PaintEventArgs e)
{
    this.drawInField();
    this.drawPalette();
    this.drawPrevious();
}

// щелчок на пункте меню Новая игра
private void menuItem2_Click(object sender,
System.EventArgs e)
{
    this.newGame();
}

// щелчок на пункте меню Выход
private void menuItem4_Click(object sender,
System.EventArgs e)
{
    this.Close();
}

// щелчок на пункте меню Справка
private void menuItem16_Click(object sender,
System.EventArgs e)
{
    Help.ShowHelp(this,this.helpProvider1.HelpNamespace);
}
```

Будильник

Программа **Будильник** демонстрирует использование компонентов `Timer`, `NotifyIcon` и `ContextMenu`. Форма программы приведена на рис. 1.48. После установки будильника, окно программы можно свернуть (значок программы отображается в системной части панели задач). Контекстное меню свернутого окна содержит пункты **Hide/Show a Clock** (Отобразить/Скрыть), **About** (О программе) и **Exit** (Выход) (рис. 1.49).

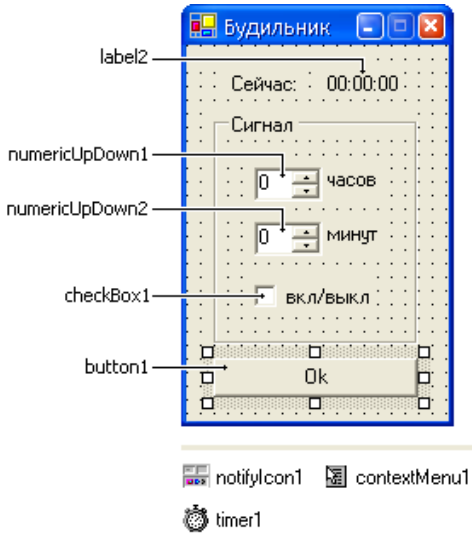

Рис. 1.48. Форма программы **Будильник**

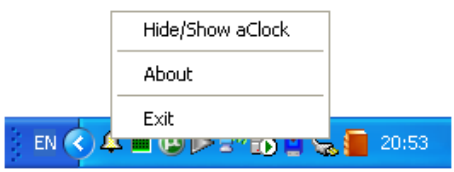

Рис. 1.49. Контекстное меню программы **Будильник**

```
// импортируем API метод PlaySound библиотеки winmm.dll,
// позволяющий проигрывать звуковые файлы
[System.Runtime.InteropServices.DllImport("winmm.dll")]
private static extern bool
    PlaySound(string lpszName, int hModule, int dwFlags);

// время срабатывания будильника
private DateTime alarm;

// конструктор формы
public WinForm()
{
    InitializeComponent();
```

```
    // параметры компонентов numericUpDown
    numericUpDown1.Maximum = 23;
    numericUpDown1.Minimum = 0;

    numericUpDown2.Maximum = 59;
    numericUpDown2.Minimum = 0;

    numericUpDown1.Value = DateTime.Now.Hour;
    numericUpDown2.Value = DateTime.Now.Minute;

    // контекстное меню значка панели задач
    notifyIcon1.ContextMenu = contextMenu1;

    notifyIcon1.Text = "Будильник не установлен.";
    notifyIcon1.Visible = true;

    // период обработки события сигнала от таймера
    timer1.Interval = 1000;
    timer1.Enabled = true;

    label2.Text = DateTime.Now.ToLongTimeString();
}

// вкл/выкл будильника
private void checkBox1_CheckedChanged(object sender,
   System.EventArgs e)
{
    if (checkBox1.Checked) {
        numericUpDown1.Enabled = false;
        numericUpDown2.Enabled = false;

        // установить время сигнала
        alarm = new DateTime(
            DateTime.Now.Year,
            DateTime.Now.Month,
            DateTime.Now.Day,
            Convert.ToInt16(numericUpDown1.Value),
            Convert.ToInt16(numericUpDown2.Value),
            0, 0);
```

```
        // если установленное время будильника меньше
        // текущего, нужно увеличить дату срабатывания
        // на единицу (+1 день)
        if (DateTime.Compare(DateTime.Now,alarm) > 0)
            alarm = alarm.AddDays(1);

        notifyIcon1.Text = "Будильник: " +
            alarm.ToShortTimeString();
    }
    else {
        numericUpDown1.Enabled = true;
        numericUpDown2.Enabled = true;

        notifyIcon1.Text =
            "Будильник не установлен.";
    }
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    // скрыть окно программы
    this.Hide();
}

// сигнал от таймера
private void timer1_Tick(object sender, System.EventArgs e)
{
    label2.Text = DateTime.Now.ToLongTimeString();

    // будильник установлен
    if (checkBox1.Checked) {
        // время срабатывания будильника
        if (DateTime.Compare(DateTime.Now,alarm) > 0) {
            checkBox1.Checked = false;

            PlaySound(Application.StartupPath +
                "\\ring.wav", 0, 1 );
```

```
            this.Show();
        }
    }
}

// выбор из контекстного меню contextMenu1
// команды Show/Hide (menuItem1)
private void menuItem1_Click(object sender,
System.EventArgs e)
{
    if (this.Visible) this.Hide();
    else this.Show();
}

// выбор из контекстного меню contextMenu1
// команды About (menuItem3)
private void menuItem3_Click(object sender,
System.EventArgs e)
{
    MessageBox.Show("Программа aClock.\n\n" +
        "Простой будильник.\n\n" +
        "Сделано в Borland® C#Builder®\n" +
        "for the Microsoft® .NET Framework\n",
        "aClock 0.02",
        MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

// выбор из контекстного меню contextMenu1
// команды Exit (menuItem5)
private void menuItem5_Click(object sender,
System.EventArgs e)
{
    this.Close();
}
```

Экзаменатор

Программа **Экзаменатор** позволяет автоматизировать процесс тестирования. В окне программы (рис. 1.50) отображается тест — последовательность вопросов, на которые пользователь должен ответить путем выбора правильного варианта ответа из предложенных. В рассматриваемой программе вопросы загружаются из файла, который представляет собой xml-документ (пример файла теста приведен на рис. 1.51). Имя файла теста передается программе при ее запуске (указывается в качестве параметра командной строки). Форма программы приведена на рис. 1.52.

```
// путь к каталогу файла теста
private string fpath;

// имя файла теста
private string fname;

// XmlReader обеспечивает чтение данных xml-файла
private System.Xml.XmlReader xmlReader;

private string
    qw,     // вопрос
    pic;    // путь к файлу иллюстрации

// варианты ответа
private string[] answ = new string[3];

private int
    right,    // правильный ответ (номер)
    otv,      // выбранный ответ (номер)
    n,        // количество правильных ответов
    nv,       // общее количество вопросов
    mode;     // состояние программы:
              // 0 - вывести первый вопрос (начало работы);
              // 1 - вывести следующий вопрос;
              // 2 - завершение работы программы

// выводит заголовок теста
private void showHead() {
```

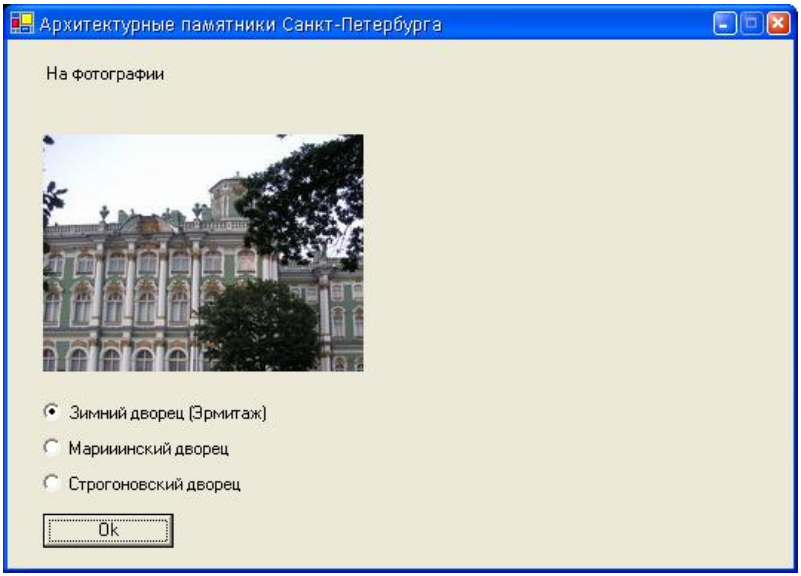

Рис. 1.50. Окно программы **Экзаменатор**

```
<?xml version="1.0" encoding="Windows-1251"?>
<test>
 <head>Архитектурные памятники Санкт-Петербурга</head>
 <description>Сейчас Вам будут предложены вопросы по архитектуре Петербурга.
     Вы должны из предложенных нескольких вариантов ответа выбрать
     правильный.</description>
 <qw>
  <q text="Архитектор Исаакиевского собора" src ="is.jpg" >
   <a right="no">Доменико Трезини</a>
   <a right="yes">Огюст Монферран</a>
   <a right="no">Карл Росси</a>
  </q>
  <q text="На фотографии" src ="marks.jpg">
   <a right="yes">Зимний дворец(Эрмитаж)</a>
   <a right="no">Марииинский дворец</a>
   <a right="no">Строгоновский дворец</a>
   </q>
 </qw>
 <levels>
  <level score="2" text = "На все вопросы вы ответили не правильно.
     Оценка - ОТЛИЧНО."/>
  <level score="1" text = "На некоторые вопросы вы ответили не правильно.
     Оценка - ХОРОШО."/>
  <level score="0" text = "Вы плохо подготовились к испытанию. Оценка - ПЛОХО!"/>
 </levels>
</test>
```

Рис. 1.51. Файл теста "Архитектурные памятники Санкт-Петербурга"

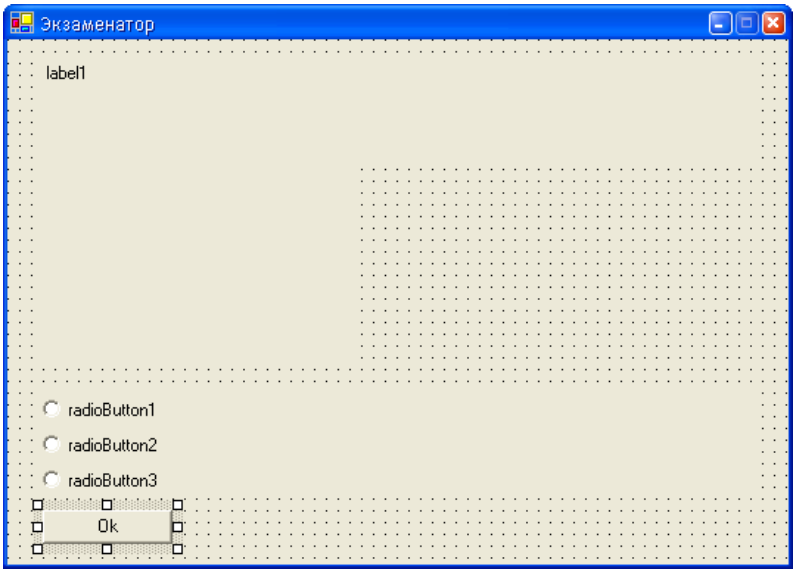

Рис. 1.52. Форма программы **Экзаменатор**

```
    // ищем узел <head>
    do xmlReader.Read();
    while(xmlReader.Name != "head");

    // считываем заголовок
    xmlReader.Read();
    this.Text = xmlReader.Value;

    // выходим из узла <head>
    xmlReader.Read();
}

// выводит описание теста
private void showDescription() {
    // ищем узел <description>
    do xmlReader.Read();
    while(xmlReader.Name != "description");

    // считываем описание теста
    xmlReader.Read();
    label1.Text = xmlReader.Value;
```

```
    // выходим из узла <description>
    xmlReader.Read();

    // ищем узел вопросов <qw>
    do xmlReader.Read();
    while(xmlReader.Name != "qw");

    // входим внутрь узла
    xmlReader.Read();
}

// читает вопрос из файла теста
private Boolean getQw() {
    // считываем тег <q>
    xmlReader.Read();

    if (xmlReader.Name == "q") {
        // здесь прочитан тег <q>,
        // атрибут text которого содержит вопрос, а
        // атрибут src содержит имя файла иллюстрации.

        // извлекаем значение атрибутов:
        qw  = xmlReader.GetAttribute("text");
        pic = xmlReader.GetAttribute("src");
        if (!pic.Equals(string.Empty)) pic = fpath + pic;

        // входим внутрь узла
        xmlReader.Read();
        int i = 0;

        // считываем данные узла вопроса <q>
        while (xmlReader.Name != "q") {
            xmlReader.Read();

            // варианты ответа
            if (xmlReader.Name == "a") {
                // запоминаем правильный ответ
                if (xmlReader.GetAttribute("right") == "yes")
                    right = i;
```

```
                // считываем вариант ответа
                xmlReader.Read();
                if (i < 3) answ[i] = xmlReader.Value;

                // выходим из узла <a>
                xmlReader.Read();

                i++;
            }
        }

        // выходим из узла вопроса <q>
        xmlReader.Read();

        return true;
    }
    // если считанный тег не является
    // тегом вопроса <q>
    else
        return false;
}

// выводит вопрос и варианты ответа
private void showQw() {
    // выводим вопрос
    label1.Text = qw;

    // иллюстрация
    if (pic.Length != 0) {
        try {
            pictureBox1.Image =
                new Bitmap(pic);

            pictureBox1.Visible = true;

            radioButton1.Top = pictureBox1.Bottom + 16;
        }
        catch {
            if (pictureBox1.Visible)
                pictureBox1.Visible = false;
```

```
            label1.Text +=
                "\n\n\nОшибка доступа к файлу " + pic + ".";

            radioButton1.Top = label1.Bottom + 8;
        }
    }
    else {
        if (pictureBox1.Visible)
            pictureBox1.Visible = false;

        radioButton1.Top = label1.Bottom;
    }

    // варианты ответа
    radioButton1.Text = answ[0];
    radioButton2.Top  = radioButton1.Top + 24;;
    radioButton2.Text = answ[1];
    radioButton3.Top  = radioButton2.Top + 24;;
    radioButton3.Text = answ[2];

    if (radioButton1.Checked) radioButton1.Checked = false;
    if (radioButton2.Checked) radioButton2.Checked = false;
    if (radioButton3.Checked) radioButton3.Checked = false;

    button1.Enabled = false;
}

// выводит оценку пройденного теста
private void showLevel() {

    // ищем узел оценок <levels>
    do xmlReader.Read();
    while(xmlReader.Name != "levels");

    // входим внутрь узла
    xmlReader.Read();

    // считываем данные узла оценок
    while (xmlReader.Name != "levels") {
        xmlReader.Read();
```

```
        if (xmlReader.Name == "level")
            // n - кол-во правильных ответов,
            // проверяем, попадаем ли в категорию
            if (n >= System.Convert.ToInt32(
                xmlReader.GetAttribute("score")))
                break;
    }

    // выводим оценку
    label1.Text =
        "Тестирование завершено.\n" +
        "Всего вопросов: " + nv.ToString() + ". " +
        "Правильных ответов: " + n.ToString() + ".\n" +
         xmlReader.GetAttribute("text");
}

// конструктор формы, имеет параметры запуска args
public WinForm(string[] args)
{
    InitializeComponent();

    radioButton1.Visible = false;
    radioButton2.Visible = false;
    radioButton3.Visible = false;

    // имя теста (или путь к нему) указано
    // в параметрах командной строки запуска программы
    if (args.Length > 0) {
        // указано имя файла теста
        if (args[0].IndexOf(":") == -1) {
            fpath = Application.StartupPath + "\\";
            fname = args[0];
        }
        // указан полный путь
        else {
            fpath = args[0].Substring(0,
                        args[0].LastIndexOf("\\")+1);
            fname = args[0].Substring(
                        args[0].LastIndexOf("\\")+1);
        }
```

```
        try{
            // получаем доступ к xml-документу
            xmlReader = new System.Xml.XmlTextReader(
                            fpath + fname);
            xmlReader.Read();

            mode = 0;
            n    = 0;

            // загрузить заголовок теста
            this.showHead();

            // загрузить описание теста
            this.showDescription();
        }
        catch(Exception exc) {
            label1.Text = "Ошибка доступа к файлу " +
                fpath + fname;

            MessageBox.Show("Ошибка доступа к файлу.\n" +
                fpath + fname + "\n" +
                exc.ToString(), "Ошибка доступа к файлу",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);

            mode = 2;
        }
    }
    // не задан файл теста
    else {
        label1.Text =
            "Не задан файл теста.\n" +
            "Файл теста необходимо указать " +
            "в командной строке запуска программы.\n" +
            "Например: 'exam economics.xml' " +
            "или 'exam c:\\spb.xml'.";
        mode = 2;
    }
}
```

```
// щелчок на кнопке выбора ответа
private void radioButton_Click(object sender,
   System.EventArgs e)
{
    if ((RadioButton)sender == radioButton1) otv = 0;
    if ((RadioButton)sender == radioButton2) otv = 1;
    if ((RadioButton)sender == radioButton3) otv = 2;

    button1.Enabled = true;
}

// щелчок на кнопке Ok
private void button1_Click(object sender, System.EventArgs e)
{
    switch (mode) {
        case 0:        // начало работы программы
            radioButton1.Visible = true;
            radioButton2.Visible = true;
            radioButton3.Visible = true;

            this.getQw();
            this.showQw();

            mode = 1;
            break;

        case 1:
            nv++;

            // правильный ли ответ выбран
            if (otv == right) n++;

            if (this.getQw()) this.showQw();
            else {
                // больше вопросов нет
                radioButton1.Visible = false;
                radioButton2.Visible = false;
                radioButton3.Visible = false;

                pictureBox1.Visible  = false;
```

```
                // обработка и вывод результата
                this.showLevel();

                // след. щелчок на кнопке Ok
                // закроет окно программы
                mode = 2;
            }
            break;
        case 2:         // завершение работы программы
            this.Close();
            break;
    }
}
```

ASP.NET

В этом разделе приведены примеры, демонстрирующие возможности Borland C#Builder по созданию ASP.NET-приложений.

Общие замечания

- Работу ASP.NET-приложения обеспечивает Web-сервер Internet Information Services (IIS) и Microsoft .NET Framework.
- Для того чтобы пользователь мог запустить ASP.NET-приложение, сервер должен быть соответствующим образом настроен. Настройка сервера выполняется при помощи консоли IIS и заключается в создании виртуального каталога приложения.
- Чтобы запустить ASP.NET-приложение, нужно в окне браузера (например, Internet Explorer) набрать адрес стартовой ASP.NET Web-страницы, который в общем виде выглядит следующим образом: **http://Сервер/Приложение/Стартовая_Страница.aspx**, где: *Сервер* — имя сервера (если браузер запущен на сервере, то вместо имени сервера можно набрать `localhost`); Приложение — имя приложения (виртуального каталога); Стартовая_Страница.aspx — имя стартовой ASP.NET-страницы приложения.
- ASP.NET-приложение можно запустить из C#Builder (в меню **Run** выбрать команду **Run whithout debug**). При этом настройка IIS будет выполнена автоматически.
- Настройка ASP.NET через командную строку осуществляется при помощи команды aspnet_regiis.
- При разработке ASP.NET-форм (страниц) следует использовать WebControls-компоненты.

Добро пожаловать

Приложение **Добро пожаловать** демонстрирует механизм формирования Web-страницы. Вид страницы в окне конструктора приведен на рис. 1.53, в окне браузера — на рис. 1.54.

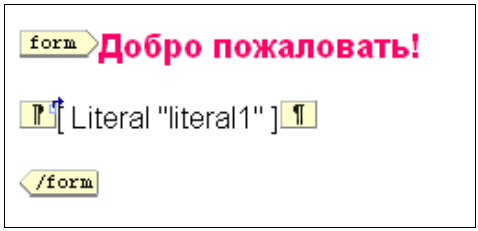

Рис. 1.53. Страница в окне конструктора

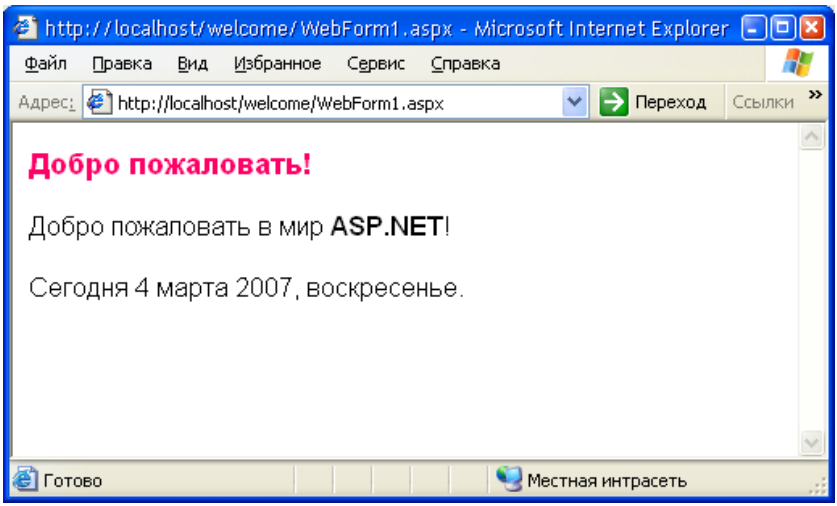

Рис. 1.54. Страница в окне браузера

```
// загрузка страницы
private void Page_Load(object sender, System.EventArgs e)
{
    // загрузка страницы в первый раз
    if (!IsPostBack) {
        literal1.Text = "<p>Добро пожаловать в мир " +
            "<b>ASP.NET</b>!</p>" +
            "<p>Сегодня " +
            DateTime.Now.ToString("d MMMM yyyy, dddd.") +
            "</p>";
    }
}
```

Хорошие окна

Программа **Хорошие окна** позволяет посетителю Web-сайта самостоятельно подсчитать цену металлопластикового окна. Вид страницы в окне конструктора приведен на рис. 1.55, в окне браузера — на рис. 1.56.

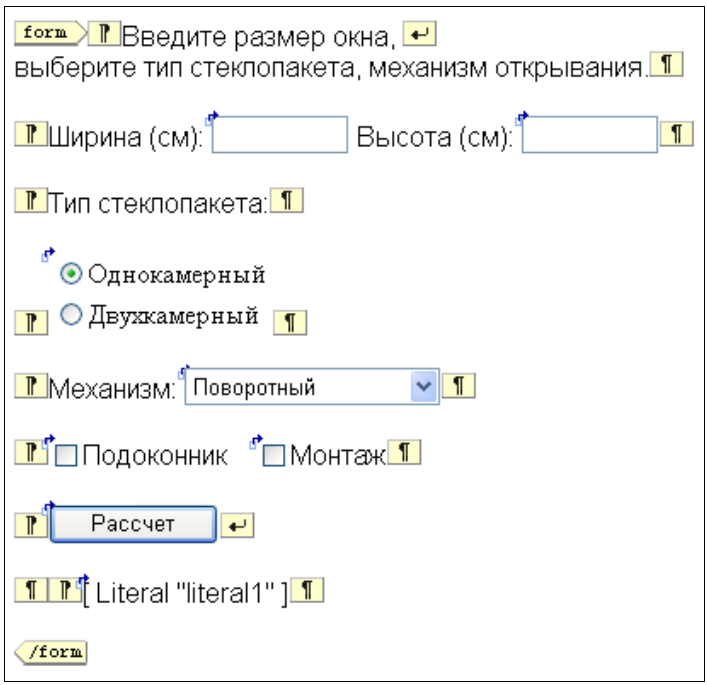

Рис. 1.55. Страница в окне конструктора

```
// щелчок на кнопке Расчет
private void button1_Click(object sender, System.EventArgs e)
{
    Single w,h,s,     // ширина, высота и площадь
           c,         // цена за 1 кв.м.
           cst;       // стоимость

    if (textBox1.Text.Length > 0 &&
        textBox2.Text.Length > 0) {
```

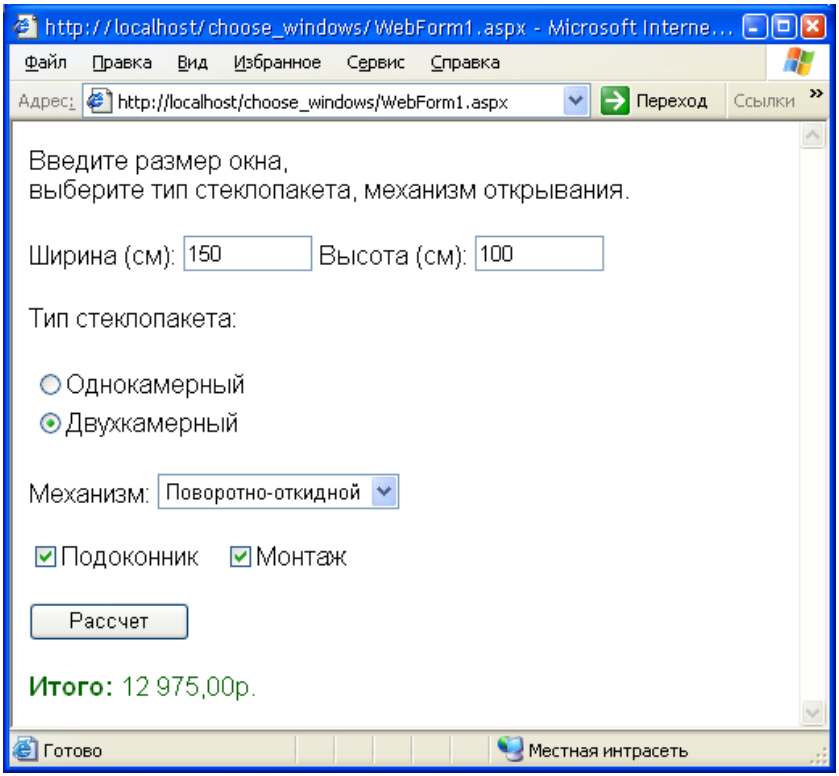

Рис. 1.56. Страница в окне браузера

```
try {
    w = Convert.ToSingle(textBox1.Text);
    h = Convert.ToSingle(textBox2.Text);
    s = w * h / 10000;
}
catch {
    literal1.Text =
        "<font color=DarkRed>Ошибка
         исходных данных.<br>" +
        "Проверьте правильность данных " +
        "в полях <b>Ширина</b> и
         <b>Высота</b>.</font>";
    return;
}
```

```
        if (radioButtonList1.Items[0].Selected)
            c = 119;    // Однокамерный стеклопакет
        else
            c = 130;    // Двухкамерный стеклопакет

        cst = s * c;

        // поворотный механизм
        switch (dropDownList1.SelectedIndex) {
            case 0: cst += 20; break;
            case 1: cst += 40; break;
        }

        // если установлен флажок Подоконник
        if (checkBox1.Checked) cst += (float)0.4*w;

        // если установлен флажок Монтаж
        if (checkBox2.Checked) cst += 80;

        // пересчет цены из у.е. в рубли
        cst = cst*(float)34.6;

        literal1.Text = "<font color=DarkGreen>" +
            "<b>Итого:</b> " + cst.ToString("C") +
            "</font>";
    }
    else
        literal1.Text = "<font color=DarkRed>" +
            "Заполните, пожалуйста, оба поля " +
            "<b>Ширина</b> и <b>Высота</b>.</font>";
}
```

Авторизация

Программа демонстрирует реализацию механизма ограничения доступа к странице. Чтобы получить доступ к главной странице информационной системы, пользователь должен ввести имя и пароль в соответствующие поля страницы авторизации. Если имя и пароль введены правильно, становится доступной главная страница информационной системы. При попытке загрузить главную страницу путем набора ее адреса в окне браузера или

если имя и пароль указаны неправильно, система "отправляет" пользователя на страницу авторизации. Страница авторизации (WebForm1.aspx) и главная страница системы (WebForm2.aspx) приведены на рис. 1.57—1.60 (в окне дизайнера формы и в окне браузера).

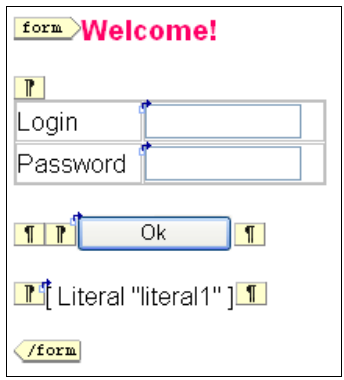

Рис. 1.57. Страница авторизации в окне конструктора (WebForm1.aspx)

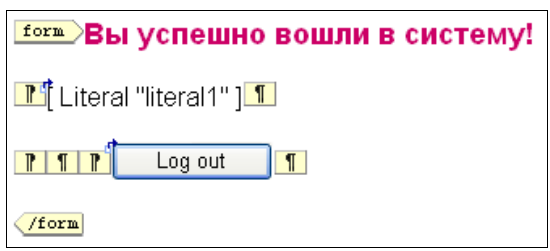

Рис. 1.58. Главная страница (WebFor2.aspx)

```
// страница авторизации (WebForm1.aspx):
// загрузка страницы
private void Page_Load(object sender, System.EventArgs e)
{
    /*
    Загрузка страницы выполняется в результате набора
    ее адреса в браузере или перехода на страницу с другой
    страницы, в том числе программным способом, например,
    в результате выполнения команды Response.Redirect( )
    */
```

```
    if (!this.IsPostBack) {
        // если задан параметр pl, то переход на WebForm1.aspx
        // был выполнен с WebForm2.aspx
        if (Request.QueryString.Get("pl") == "PassLogError")
            literal1.Text = "<font color=DarkRed>" +
                "Неверное сочетание имени и пароля.</font>";
    }
}

// щелчок на кнопке OK
private void button1_Click(object sender, System.EventArgs e)
{
    if (textBox1.Text.Length > 0 &&
        textBox2.Text.Length > 0)
        Response.Redirect(
            "WebForm2.aspx?uid=" + textBox1.Text +
            "&pwd=" + textBox2.Text);
    else
        literal1.Text = "<font color=DarkRed>" +
            "Необходимо ввести имя и пароль!</font>";
}


// главная страница (WebForm2.aspx):
// загрузка страницы
private void Page_Load(object sender, System.EventArgs e)
{
    if (!this.IsPostBack) {
        // доступ к странице WebForm2.aspx открыт
        // только с главной страницы  (WebForm1.aspx)

        /*
        проверим, кто загружает эту страницу - программа
        или пользователь. Если программа, то свойство
        Request.UrlReferrer содержит адрес страницы,
        с которой выполнен переход на эту. Если пользователь
        (путем ввода имени страницы в окне браузера),
        то значение свойства - пустая строка.
        */
```

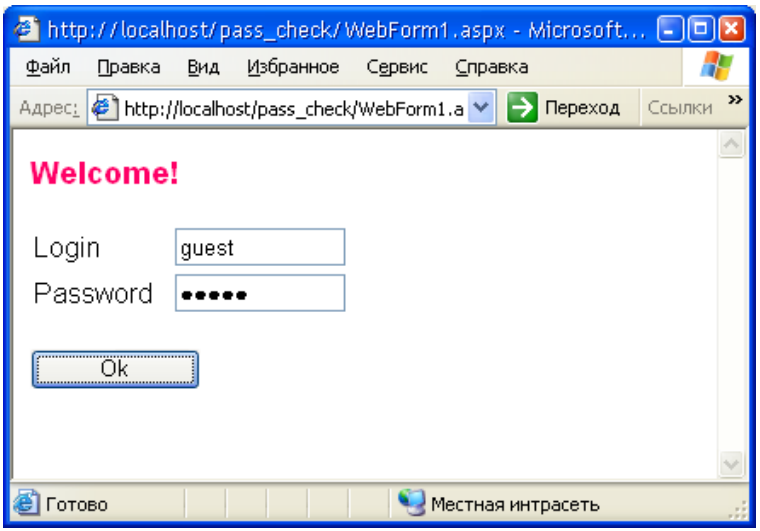

Рис. 1.59. Страница авторизации в окне браузера

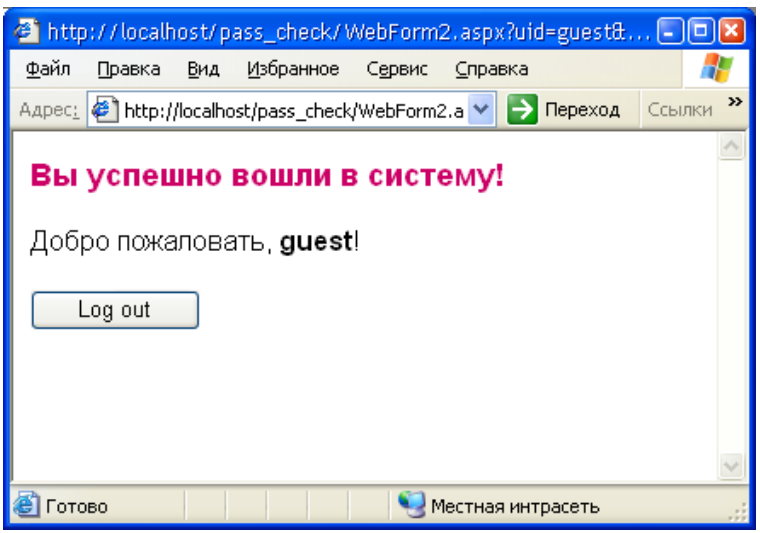

Рис. 1.60. Главная страница в окне браузера

```
if (Request.UrlReferrer == null) {
    // попытка загрузить страницу WebForm2.aspx
    // путем ввода ее адреса в браузере,
    // "отправляем" пользователя на главную страницу
    Response.Redirect("WebForm1.aspx");
}
else {
    /*
    свойство Request.UrlReferrer содержит
    полный адрес страницы, включая имя сервера и
    виртуального каталога. Выделяем из свойства
    Request.UrlReferrer имя страницы
    */
    string urlRefForm =
        Request.UrlReferrer.ToString().Substring(
            Request.UrlReferrer.ToString().
            LastIndexOf("/") + 1);

    if (urlRefForm != "WebForm1.aspx" &&
        urlRefForm !=
            "WebForm1.aspx?pl=PassLogError"){
        // попытка перейти на WebForm2.aspx с другой,
        // а не с WebForm1.aspx страницы
        Response.Redirect("WebForm1.aspx");
    }
    else {
        // переход выполнен с WebForm1.aspx,
        // проверяем имя и пароль;
        // имя пользователя и пароль указаны
        // в качестве параметров загрузки страницы
        if ((Request.QueryString.Get("uid") ==
                "guest") &&
            (Request.QueryString.Get("pwd") ==
                "guest"))
            literal1.Text = "Добро пожаловать, <b>" +
                Request.QueryString.Get("uid") +
                "</b>!";
        else
            Response.Redirect(
                "WebForm1.aspx?pl=PassLogError");
```

```
            }
        }
    }
}

// щелчок на кнопке Log out
private void button1_Click(object sender, System.EventArgs e)
{
    Response.Redirect("WebForm1.aspx");
}
```

Контакты

Программа демонстрирует возможность использования ASP.NET-приложения в качестве клиента, обеспечивающего доступ к базе данных. База данных **Контакты** представляет собой файл contacts_asp.xml. Форма программы приведена на рис. 1.61, окно — на рис. 1.62. Значения свойств компонентов, обеспечивающих доступ к данным и их отображение, представлены в табл. 1.18 и 1.19.

Таблица 1.18. *Значения свойств компонента dataSet1*

Свойство	Значение
dataSet1.dataTable1.TableName	Contacts
dataSet1.dataTable1.Columns.dataColumn1.ColumnName	Title
dataSet1.dataTable1.Columns.dataColumn2.ColumnName	Phone

Таблица 1.19. *Значения свойств компонента dataGrid1*

Свойство	Значение
dataGrid1.AutoGenerateColumns	False
dataGrid1.DataSource	dataSet1
dataGrid1.DataMember	Contacts
dataGrid1.PageSize	10

Таблица 1.19 *(окончание)*

Свойство	Значение
dataGrid1.AllowPaging	True
dataGrid1.Columns.DataFields.Title.HeaderText	Контакт
dataGrid1.Columns.DataFields.Phone.HeaderText	Телефон
dataGrid1.Columns.Paging.Mode	PageNumbers

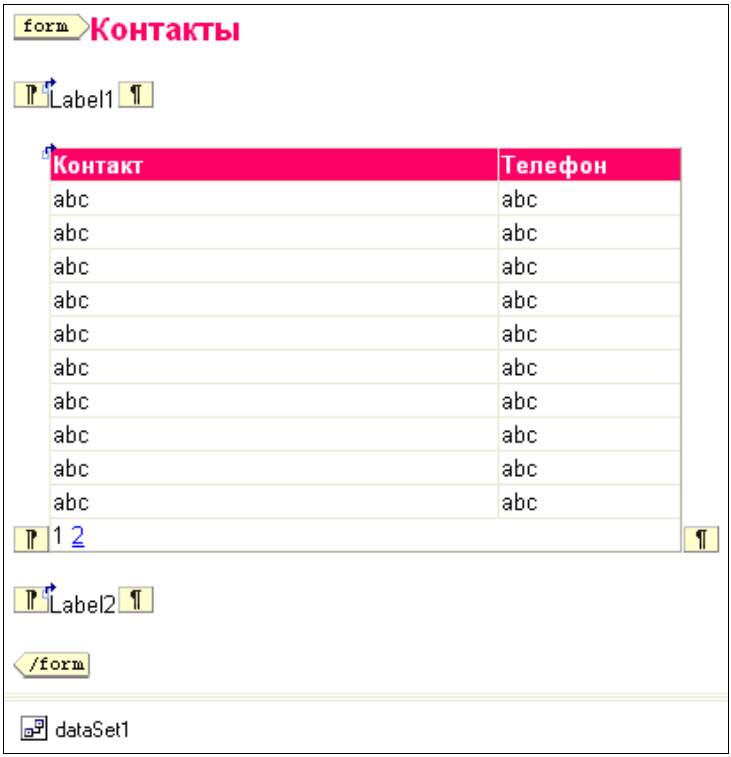

Рис. 1.61. Главная страница

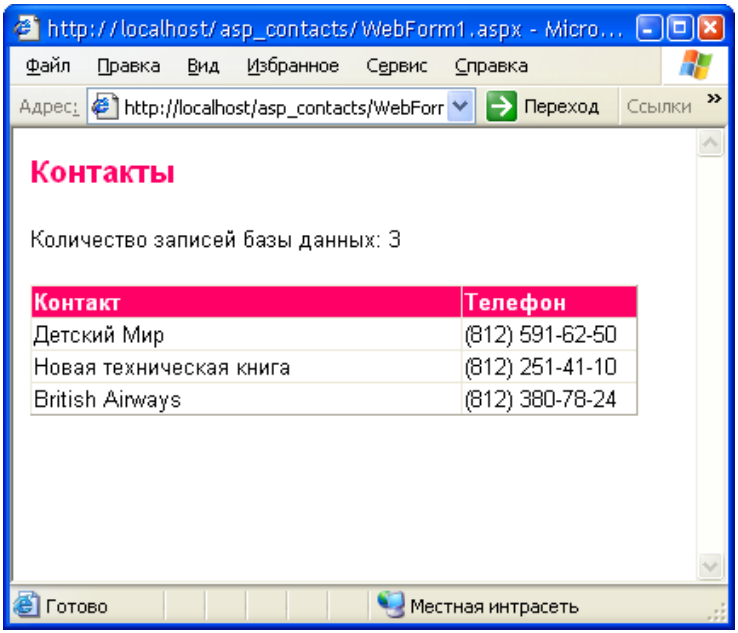

Рис. 1.62. Главная страница в окне браузера

```
// загрузка страницы
private void Page_Load(object sender, System.EventArgs e)
{
    if (!this.IsPostBack) {

        // проверяем, существует ли файл БД
        string dbpath =
            Server.MapPath("contacts_asp.xml");

        if (System.IO.File.Exists(dbpath)) {
            // считываем данные БД
            dataSet1.ReadXml(dbpath);

            label1.Text =
                "Количество записей базы данных:  " +
                dataSet1.Tables["Contacts"].
                Rows.Count.ToString();
```

```
            if (dataSet1.Tables["Contacts"].Rows.Count <
                dataGrid1.PageSize) {
                dataGrid1.AllowPaging = false;
            }

            // связать данные с компонентом,
            // обеспечивающим их отображение
            dataGrid1.DataBind();

            // выбранная страница
            if (dataGrid1.AllowPaging)
                label2.Text = "Страница: " +
                (dataGrid1.CurrentPageIndex + 1).ToString();
            else
                label2.Text = string.Empty;
        }
    }
}

// переход к другой странице
private void dataGrid1_PageIndexChanged(object source,
   System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
{
    dataGrid1.CurrentPageIndex = e.NewPageIndex;
    dataSet1.ReadXml(Server.MapPath("contacts_asp.xml"));
    dataGrid1.DataBind();

    label2.Text = "Страница: " +
        (dataGrid1.CurrentPageIndex + 1).ToString();
}
```

ЧАСТЬ 2

Краткий справочник

Форма

Свойства формы (объекта `WinForm`) приведены в табл. 2.1.

Таблица 2.1. *Свойства* `формы (объекта WinForm)`

Свойство	Описание
`Name`	Имя формы
`Text`	Текст заголовка
`Location`	Положение компонента на поверхности формы
`Size.Width`	Ширина формы (рабочая область вместе с правой и левой границей)
`Size.Height`	Высота формы (рабочая область вместе с верхней, нижней границей и заголовком)
`StartPosition`	Положение формы в момент первого появления на экране (после запуска программы). Форма может находиться в центре экрана (`CenterScreen`), в центре родительской формы (`CenterParent`). Положение формы может также определяться значением свойства `Location`. В последнем случае свойство `StartPosition` должно иметь значение `Manual`
`Location.X`	Расстояние от верхней границы формы до верхней границы экрана (или родительской формы)
`Location.Y`	Расстояние от левой границы формы до левой границы экрана (или родительской формы)

Таблица 2.1 *(продолжение)*

Свойство	Описание
`FormBorderStyle`	Вид границы формы. Граница может быть масштабируемой (`Sizable`), тонкой не масштабируемой (`FixedSingle`), отсутствовать (`None`). При отсутствии границы (`None`) окно отображается без заголовка, вследствие чего изменить размер и положение такого окна не удастся
`ControlBox`	Если значение равно `False`, системное меню программы (контекстное меню, отображаемое при правом щелчке мыши на заголовке окна программы или на панели задач), а также кнопки управления окном не отображаются (команды **Свернуть**, **Развернуть**, **Закрыть** в заголовке окна программы)
`MaximazeBox`	Признак доступности кнопки **Развернуть**
`MinimazeBox`	Признак доступности кнопки **Свернуть**
`Icon`	Значок в заголовке окна
`Font`	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства компонента не было задано (изменено) вручную)
`ForeColor`	Цвет, используемый по умолчанию компонентами, находящимися на поверхности формы, для отображения текста, а также в качестве основного цвета. Изменение свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства компонента не было задано (изменено) вручную)
`BackColor`	Цвет фона. Можно указать название или же привязаться к цветовой схеме операционной системы. Привязка к цветовой схеме задается путем указания элемента интерфейса (например, `Control`)
`BackGroundImage`	Фоновое изображение. Если изображение меньше размера формы, фоновое изображение будет сформировано путем дублирования его по вертикали и горизонтали

Таблица 2.1 *(окончание)*

Свойство	Описание
`Opacity`	Степень прозрачности формы. Форма может быть абсолютно непрозрачной (100 %) или абсолютно прозрачной (0 %). Если значение находится в диапазоне 0—100%, то сквозь форму будет видна поверхность, на которой она находится

Компоненты

В этом разделе приведено краткое описание базовых компонентов. Подробное описание этих и других компонентов можно найти в справочной системе.

Button

Компонент `Button` представляет собой командную кнопку. Свойства компонента приведены в табл. 2.2.

Таблица. 2.2. *Свойства компонента* `Button`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к свойствам компонента
`Text`	Текст (надпись) на кнопке
`TextAlign`	Положение текста (надписи) на кнопке. Надпись может располагаться в центре (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`FlatStyle`	Кнопка может быть стандартной (`Standard`), плоской (`Flat`) или "всплывающей" (`Popup`)
`Location`	Положение кнопки на поверхности формы. Уточняющее свойство `X` определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство `Y` — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)

Таблица. 2.2 *(окончание)*

Свойство	Описание
`Size`	Размер кнопки
`Font`	Шрифт, используемый для отображения текста на кнопке
`ForeColor`	Цвет текста, отображаемого на кнопке
`Enabled`	Признак доступности кнопки. Кнопка доступна, если значение свойства равно `True`, и недоступна (например, событие `Click` в результате щелчка на кнопке не возникает), если значение свойства равно `False`
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Cursor`	Вид указателя мыши при позиционировании указателя на кнопке
`Image`	Картинка на поверхности формы. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
`ImageAlign`	Положение картинки на кнопке. Картинка может располагаться в центе (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`ImageList`	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения нужно добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) изображения из набора `ImageList`, которое отображается на кнопке
`ToolTip`	Подсказка, появляющаяся рядом с указателем мыши при позиционировании его на кнопке. Чтобы свойство было доступно, в форму приложения надо добавить компонент `ToolTip`

ComboBox

Компонент ComboBox представляет собой комбинацию поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 2.3.

Таблица 2.3. *Свойства компонента* ComboBox

Свойство	Описание
DropDownStyle	Вид компонента: DropDown — поле ввода и раскрывающийся список; Simple — поле ввода со списком; DropDownList — раскрывающийся список
Text	Текст, находящийся в поле ввода/редактирования (для компонентов типа DropDown и Simple)
Items	Элементы списка — коллекция строк
Items.Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
MaxDropDownItems	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем MaxDropDownItems, то появляется вертикальная полоса прокрутки
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDown и DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
DropDownWidth	Ширина области списка
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

ContextMenu

Компонент ContextMenu представляет собой контекстное меню — список команд, который отображается в результате щелчка правой кнопкой мыши. Элементы контекстного меню — объекты MenuItem. Свойства объекта MenuItem приведены в табл. 2.4.

Таблица 2.4. *Свойства объекта* MenuItem

Свойство	Описание
Text	Команда контекстного меню
Enabled	Признак доступности команды. Если значение свойства равно False, то команда недоступна (в результате щелчка на команде событие Click не происходит, название элемента меню отображается инверсным цветом по отношению к доступному пункту меню)
Checked	Признак того, что элемент меню выбран. Если значение свойства равно True, то элемент помечается галочкой или (если значение свойства RadioCheck равно True) точкой. Свойство Checked обычно используется для тех элементов меню, которые применяются для отображения параметров
RadioCheck	Признак того, что для индикации состояния свойства Checked используется точка (True), а не галочка (False)

CheckBox

Компонент CheckBox представляет переключатель, который может находиться в одном из двух состояний: выбранном или невыбранном. Часто вместо "выбранный" говорят "установленный", а вместо "невыбранный" — "сброшенный" или "выключенный". Рядом с переключателем обычно находится поясняющий текст. Свойства компонента CheckBox приведены в табл. 2.5.

Таблица 2.5. *Свойства компонента* CheckBox

Свойство	Описание
Text	Текст, который находится справа от кнопки

Таблица 2.5 *(продолжение)*

Свойство	Описание
Checked	Состояние, внешний вид переключателя. Если переключатель выбран, то значение свойства равно True. Если переключатель сброшен, то значение свойства равно False
TextAllign	Положение текста в поле отображения текста. Текст может располагаться в центре поля (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения текста надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
CheckAllign	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (TopLeft), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (MiddleLeft). Есть и другие варианты размещения кнопки в поле компонента
Enabled	Свойство позволяет сделать переключатель недоступным (False)
Visible	Свойство позволяет скрыть (False) переключатель
AutoCheck	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно True
FlatStyle	Стиль (вид) переключателя. Переключатель может быть обычным (Standard), плоским (Flat) или "всплывающим" (Popup). Стиль определяет поведение переключателя при позиционировании указателя мыши на его изображении
Appearance	Определяет вид переключателя. Переключатель может выглядеть обычным образом (Normal) или как кнопка (Button)
Image	Картинка, которая отображается в поле компонента
ImageAlign	Положение картинки в поле компонента. Картинка может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)

Таблица 2.5 *(окончание)*

Свойство	Описание
`ImageList`	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения следует добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) картинки из набора `ImageList`, которая отображается в поле компонента

CheckedListBox

Компонент `CheckedListBox` представляет собой список, перед каждым элементом которого находится переключатель `CheckBox`. Свойства компонента `CheckedListBox` приведены в табл. 2.6.

Таблица 2.6. *Свойства компонента* `CheckedListBox`

Свойство	Описание
`Items`	Элементы списка — коллекция строк
`Items.Count`	Количество элементов списка
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`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 свойства равно `True`, то щелчок на элементе списка выделяет элемент и устанавливает во включенное состояние переключатель
`CheckedItems`	Свойство `CheckedItems` представляет собой коллекцию, элементы которой содержат выбранные элементы списка
`CheckedItems.Count`	Количество выбранных элементов списка, переключатели которых установлены в выбранное состояние

Таблица 2.6 *(окончание)*

Свойство	Описание
CheckedIndices	Свойство CheckedIndices представляет собой коллекцию, элементы которой содержат номера выбранных (помеченных) элементов списка
MultiColumn	Признак необходимости отображать список в несколько колонок. Количество отображаемых колонок зависит от количества элементов и размера области отображения списка
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDown и DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

GroupBox

Компонент GroupBox представляет собой контейнер для других компонентов. Обычно он используется для объединения в группы компонентов RadioButton по функциональному признаку. Свойства компонента GroupBox приведены в табл. 2.7.

Таблица 2.7. *Свойства компонента* GroupBox

Свойство	Описание
Text	Заголовок — текст, поясняющий назначение компонентов, которые находятся в поле компонента GroupBox
Enabled	Позволяет управлять доступом к компонентам, находящимся в поле (на поверхности) компонента GroupBox. Если значение свойства равно False, то все находящиеся в поле GroupBox компоненты недоступны
Visible	Позволяет скрыть (сделать невидимым) компонент GroupBox и все компоненты, которые находятся на его поверхности

ImageList

Компонент ImageList представляет собой коллекцию изображений и может использоваться другими компонентами (например, Button или ToolBar) как источник иллюстраций. Компонент является невизуальным, т. е. он не отображается в окне программы во время ее работы. Во время создания формы компонент отображается в нижней части окна редактора формы. Свойства компонента ImageList приведены в табл. 2.8.

Таблица 2.8. *Свойства компонента* ImageList

Свойство	Описание
Images	Коллекция изображений (объектов Bitmap)
ImageSize	Размер изображений коллекции. Уточняющее свойство Width определяет ширину изображений, Height — высоту
TransparentColor	Прозрачный цвет. Участки изображения, окрашенные этим цветом, не отображаются
ColorDepht	Глубина цвета — количество байтов, используемых для кодирования цвета точки (пиксела)

Label

Компонент Label предназначен для отображения текстовой информации. Задать текст, отображаемый в поле компонента, можно как во время разработки формы, так и во время работы программы, присвоив нужное значение свойству Text. Свойства компонента приведены в табл. 2.9.

Таблица 2.9. *Свойства компонента* Label

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
Text	Отображаемый текст
Location	Положение компонента на поверхности формы
Size	Размер компонента (поля отображения текста)

Таблица 2.9 *(окончание)*

Свойство	Описание
`Font`	Шрифт, используемый для отображения текста
`ForeColor`	Цвет текста, отображаемого в поле компонента
`BackColor`	Цвет закраски области вывода текста
`TextAlign`	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (`TopLeft`), посередине (`TopCentre`) и по центру (`MiddleCenter`)
`BorderStyle`	Вид рамки (границы) компонента. По умолчанию граница вокруг поля `Label` отсутствует (значение свойства равно `None`). Граница компонента может быть обычной (`Fixed3D`) или тонкой (`FixedSingle`)

ListBox

Компонент `ListBox` представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 2.10.

Таблица 2.10. *Свойства компонента* `ListBox`

Свойство	Описание
`Items`	Элементы списка — коллекция строк
`Items.Count`	Количество элементов списка
`SelectedIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`SelectionMode`	Определяет режим выбора элементов списка: `One` — только один элемент; `MultiSimple` — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка; `MultiExtended` — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка при нажатой клавише <Ctrl>, или выделить диапазон, щелкнув при нажатой клавише <Shift> на первом и последнем элементе диапазона

Таблица 2.10 *(окончание)*

Свойство	Описание
`MultiColumn`	Признак необходимости отображать список в несколько колонок. Количество отображаемых колонок зависит от количества элементов и размера области отображения списка
`Location`	Положение компонента на поверхности формы
`Size`	Размер компонента без (для компонентов типа `DropDown` и `DropDownList`) или с учетом (для компонента типа `Simple`) размера области списка или области ввода
`Font`	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

MainMenu

Компонент `MainMenu` представляет собой главное меню программы. Элементы меню — объекты `MenuItem`. Свойства объекта `MenuItem` приведены в табл. 2.11.

Таблица 2.11. *Свойства объекта* `MenuItem`

Свойство	Описание
`Text`	Название элемента меню
`Enabled`	Признак доступности элемента меню. Если значение свойства равно `False`, то элемент меню недоступен (в результате щелчка на элементе меню событие `Click` не происходит, название элемента меню отображается инверсным цветом по отношению к доступному пункту меню)
`Checked`	Признак того, что элемент меню выбран. Если значение свойства равно `True`, то элемент помечается галочкой. Свойство `Checked` обычно используется для тех элементов меню, которые применяются для отображения параметров
`Shortcut`	Свойство определяет функциональную клавишу (или комбинацию клавиш), нажатие которой активизирует выполнение команды

Таблица 2.11 *(окончание)*

Свойство	Описание
ShowShortcut	Если значение свойства равно True и задано значение свойства Shortcut, то после названия команды отображается название функциональной клавиши, нажатие которой активизирует команду

NotifyIcon

Компонент NotifyIcon представляет собой значок, который изображает на панели задач программу, работающую в фоновом режиме. Обычно при позиционировании указателя мыши на значке появляется подсказка, а в результате щелчка правой кнопки — контекстное меню, команды которого позволяют получить доступ к программе или завершить ее работу. Компонент NotifyIcon является невизуальным. Свойства компонента приведены в табл. 2.12.

Таблица 2.12. *Свойства компонента* NotifyIcon

Свойство	Описание
Icon	Значок, который отображается на панели задач
Text	Подсказка (обычно название программы), которая отображается рядом с указателем мыши при позиционировании указателя на находящемся на панели задач значке
ContextMenu	Ссылка на компонент ContextMenu, который обеспечивает отображение контекстного меню как результат щелчка правой кнопкой мыши на значке, находящемся на панели задач
Visible	Свойство позволяет скрыть (False) значок с панели задач или сделать его видимым

NumericUpDown

Компонент NumericUpDown предназначен для ввода числовых данных. Данные в поле редактирования можно ввести путем набора на клавиатуре или при помощи командных кнопок **Увеличить** и

Уменьшить, которые находятся справа от поля редактирования. Свойства компонента `NumericUpDown` приведены в табл. 2.13.

Таблица 2.13. *Свойства компонента* `NumericUpDown`

Свойство	Описание
`Value`	Значение, соответствующее содержимому поля редактирования
`Maximum`	Максимально возможное значение, которое можно ввести в поле компонента
`Minimum`	Минимально возможное значение, которое можно ввести в поле компонента
`Increment`	Величина, на которую увеличивается или уменьшается значение свойства `Value` при каждом щелчке мышью на кнопках **Увеличить** или **Уменьшить**
`DecimalPlaces`	Количество цифр дробной части числа

OpenFileDialog

Компонент `OpenFileDialog` представляет собой стандартное диалоговое окно **Открыть**. Свойства компонента приведены в табл. 2.14.

Таблица 2.14. *Свойства компонента* `OpenFileDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Открыть**
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке файлов отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов надо отобразить только те файлы, расширение которых txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

Таблица 2.14 *(окончание)*

Свойство	Описание
FileNane	Имя выбранного пользователем файла
InitialDirectory	Каталог, содержимое которого отображается при появлении диалога на экране
RestoreDirectory	Признак необходимости отображать содержимое каталога, указанного в свойстве InitialDirectory, при каждом появлении окна. Если значение свойства равно False, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз

Panel

Компонент Panel представляет собой контейнер для других компонентов и позволяет легко управлять компонентами, которые находятся на панели. Например, для того чтобы сделать недоступными компоненты, которые находятся на панели, достаточно присвоить значение False свойству Enabled панели. Свойства компонента Panel приведены в табл. 2.15.

Таблица 2.15. *Свойства компонента* Panel

Свойство	Описание
Dock	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (Left), правой (Right), верхней (Top) или нижней (Bottom) границе
BorderStyle	Вид границы панели: FixedSingle — рамка; Fixed3D — объемная граница; None — граница не отображается
Enabled	Свойство позволяет сделать недоступными (False) все компоненты, которые находятся на панели
Visible	Позволяет скрыть (False) панель
AutoScroll	Признак необходимости отображать полосы прокрутки, если компоненты, которые находятся на панели, не могут быть выведены полностью

PictureBox

Компонент PictureBox обеспечивает отображение иллюстрации (рисунка, фотографии и т. п.). Свойства компонента приведены в табл. 2.16.

Таблица 2.16. *Свойства компонента* PictureBox

Свойство	Описание
Image	Иллюстрация, которая отображается в поле компонента
SizeMode	Режим отображения иллюстрации (способ масштабирования), если размер иллюстрации не соответствует размеру компонента: Normal — масштабирование не выполняется (если размер компонента меньше размера иллюстрации, то отображается только часть иллюстрации); StretchImage — выполняется масштабирование иллюстрации таким образом, что она занимает всю область отображения (если размер компонента не пропорционален размеру иллюстрации, она будет искажена); AutoSize — размер компонента автоматически изменяется и соответствует размеру иллюстрации; CenterImage — центрирование иллюстрации в поле компонента, если размер иллюстрации меньше размера области отображения
Image.PhysicalDimension	Свойство содержит информацию о размере картинки (иллюстрации), загруженной в поле компонента
Location	Положение компонента (области отображения иллюстрации) на поверхности формы. Уточняющее свойство X определяет расстояние от левой границы области до левой границы формы, уточняющее свойство Y — от верхней границы области до верхней границы клиентской области формы (нижней границы заголовка)
Size	Размер компонента (области отображения иллюстрации). Width определяет ширину области, Height — высоту

Таблица 2.16 *(окончание)*

Свойство	Описание
`Visible`	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
`BorderStyle`	Вид границы компонента: `None` — граница не отображается; `FixedSingle` — тонкая; `Fixed3D` — объемная
`Graphics`	Поверхность, на которую можно выводить графику (доступ к графической поверхности компонента есть у процедуры обработки события `Paint`)

RadioButton

Компонент `RadioButton` представляет собой кнопку (переключатель), состояние которой зависит от состояния других кнопок (компонентов `RadioButton`). Обычно компоненты `RadioButton` объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента `GroupBox`). В каждый момент времени только одна из кнопок группы может находиться в выбранном состоянии (возможна ситуация, когда ни одна из кнопок не выбрана). Состояние компонентов, принадлежащих одной группе, не зависит от состояния компонентов, принадлежащих другой группе. Свойства компонента приведены в табл. 2.17.

Таблица 2.17. *Свойства компонента* `RadioButton`

Свойство	Описание
`Text`	Текст, который находится справа от кнопки
`Checked`	Состояние, внешний вид кнопки. Если кнопка выбрана, то значение свойства `Checked` равно `True`; если кнопка не выбрана, то значение свойства `Checked` равно `False`

Таблица 2.17 *(окончание)*

Свойство	Описание
TextAllign	Положение текста в поле отображения. Текст может располагаться в центре поля (MiddleCenter), прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения текста надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
CheckAllign	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (TopLeft), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (MiddleLeft). Есть и другие варианты размещения кнопки в поле компонента
Enabled	Свойство позволяет сделать кнопку недоступной (False)
Visible	Свойство позволяет скрыть (False) кнопку
AutoCheck	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно True
FlatStyle	Стиль кнопки. Кнопка может быть обычной (Standard), плоской (Flat) или "всплывающей" (Popup). Стиль кнопки определяет ее поведение при позиционировании указателя мыши на изображении кнопки
Appearance	Определяет вид переключателя. Переключатель может выглядеть обычным образом (Normal) или как кнопка (Button)
Image	Картинка, которая отображается в поле компонента
ImageAlign	Положение картинки в поле компонента. Картинка может располагаться в центре (MiddleCenter), прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения надо добавить компонент ImageList
ImageIndex	Номер (индекс) картинки из набора ImageList, которая отображается в поле компонента

ProgressBar

Компонент ProgressBar представляет собой индикатор, который обычно используется для наглядного представления протекания процесса (например, обработки (копирования) файлов, загрузки информации из сети и т. п.) Свойства компонента ProgressBar приведены в табл. 2.18. Следует обратить внимание, что выход значения свойства Value за границы диапазона, заданного свойствами Minimum и Maximum, вызывает исключение.

Таблица 2.18. *Свойства компонента* ProgressBar

Свойство	Описание
Value	Значение, которое отображается в поле компонента в виде последовательности прямоугольников, количество которых пропорционально значению свойства Value
Minimum	Минимально допустимое значение свойства Value
Maximum	Максимально допустимое значение свойства Value
Step	Приращение (шаг) изменения значения свойства Value при использовании для изменения значения свойства Value метода PerformStep

SaveFileDialog

Компонент SaveFileDialog представляет собой стандартное диалоговое окно **Сохранить**. Свойства компонента приведены в табл. 2.19.

Таблица 2.19. *Свойства компонента* SaveFileDialog

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**
FileName	Полное имя файла, которое задал пользователь. В общем случае оно образуется из имени каталога, содержимое которого отображается в диалоговом окне, имени файла, которое пользователь ввел в поле **Имя файла**, и расширения, заданного значением свойства DefaultExt

Таблица 2.19 *(окончание)*

Свойство	Описание
`DefaultExt`	Расширение файла по умолчанию. Если пользователь в поле **Имя файла** не укажет расширение, то к имени файла будет добавлено расширение (при условии, что значение свойства `AddExtension` равно `True`), заданное значением этого свойства
`InitialDirectory`	Каталог, содержимое которого отображается при появлении диалога на экране
`RestoreDirectory`	Признак необходимости отображать содержимое каталога, указанного в свойстве `InitialDirectory`, при каждом появлении окна. Если значение свойства равно `False`, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз
`CheckPathExists`	Признак необходимости проверки существования каталога, в котором следует сохранить файл. Если указанного каталога нет, то выводится информационное сообщение
`CheckFileExists`	Признак необходимости проверки существования файла с заданным именем. Если значение свойства равно `True` и файл с указанным именем уже существует, то появляется окно запроса, в котором пользователь может подтвердить необходимость замены (перезаписи) существующего файла
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке файлов отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов надо отобразить только файлы с расширением txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

StatusBar

Компонент `StatusBar` представляет собой строку состояния, которая обычно отображается в нижней части окна. Свойства компонента `StatusBar` приведены в табл. 2.20.

Для того чтобы панель состояния была разделена на области, в коллекцию `Panels` надо добавить соответствующее количество элементов. Свойства объекта `StatusBarPanel` приведены в табл. 2.21.

Таблица 2.20. *Свойства компонента* `StatusBar`

Свойство	Описание
`Panels`	Коллекция объектов типа `StatusBarPanel`, каждый из которых представляет собой отдельную область панели `StatusBar`
`SizingGrip`	Признак отображения на панели `StatusBar` кнопки изменения размера панели (три наклонные линии в правом нижнем углу панели)
`Text`	Текст, который отображается на панели состояния, если панель не разделена на области. Если панель разделена на области, то текст, находящийся в области, определяется свойством `Text` соответствующего элемента коллекции `Panels`
`ShowPanels`	Признак необходимости отображать области

Таблица 2.21. *Свойства объекта* `StatusBarPanel`

Свойство	Описание
`Text`	Текст, отображаемый на панели
`Icon`	Картинка, отображаемая на панели
`AutoSize`	Признак автоматического изменения размера панели. Если значение свойства равно `Contents`, то ширина панели зависит от ее содержания (длины текста). Если значение свойства равно `None`, то ширину панели определяет свойство `Width`. Если значение свойства равно `Spring`, то ширина панели устанавливается такой, чтобы находящаяся справа другая панель была прижата к правой границе окна. Если справа панели нет, то ширина устанавливается такой, чтобы правая граница панели была прижата к правой границе формы
`BorderStyle`	Вид границы панели. Панель может быть приподнята (`Raised`), утоплена (`Sunken`) или не иметь границы (`None`)

TextBox

Компонент TextBox предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 2.22.

Таблица 2.22. *Свойства компонента* TextBox

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в т. ч. к тексту, который находится в поле редактирования
Text	Текст, который находится в поле редактирования
Location	Положение компонента на поверхности формы
Size	Размер компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ForeColor	Цвет текста, находящегося в поле компонента
BackColor	Цвет фона поля компонента
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle) или отсутствовать (None)
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находиться по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
PasswordChar	Символ, который используется для отображения вводимых пользователем символов (введенная пользователем строка находится в свойстве Text)
Multiline	Разрешает (True) или запрещает (False) ввод нескольких строк текста
ReadOnly	Разрешает (True) или запрещает (False) редактирование отображаемого текста
Dock	Способ привязки положения и размера компонента к размеру формы. По умолчанию привязка отсутствует (None). Если значение свойства равно Top или Bottom,

Таблица 2.22 *(окончание)*

Свойство	Описание
	ширина компонента устанавливается равной ширине формы и компонент прижимается соответственно к верхней или нижней границе формы. Если значение свойства `Dock` равно `Fill`, а свойства `Multiline` — `True`, то размер компонента устанавливается максимально возможным
`Lines`	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме `MultiLine`. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
`ScrollBars`	Задает отображаемые полосы прокрутки: `Horizontal` — горизонтальная; `Vertical` — вертикальная; `Both` — горизонтальная и вертикальная; `None` — не отображать

ToolBar

Компонент `ToolBar` представляет собой панель инструментов, на которой находятся командные кнопки. Свойства компонента `ToolBar` приведены в табл. 2.23.

Таблица 2.23. *Свойства компонента* `ToolBar`

Свойство	Описание
`Buttons`	Кнопки панели инструментов — коллекция объектов типа `ToolBarButton`
`ImageList`	Ссылка на компонент `ImageList`, который содержит картинки для кнопок
`ButtonSize`	Размер кнопок, находящихся на панели инструментов
`Appearance`	Вид кнопок. Кнопка может быть обычной (выпуклой) — `Normal` или плоской — `Flat`
`TextAlign`	Положение поясняющего текста, если кнопки панели инструментов плоские. Текст может быть расположен ниже (`Underneath`) или справа (`Right`) от картинки
`BorderStyle`	Вид границы панели инструментов. Граница может отсутствовать (`None`), быть объемной (`Fixed3D`) или тонкой (`FixedSingle`)

Таблица 2.23 *(окончание)*

Свойство	Описание
`Visible`	Свойство позволяет скрыть (`False`) или сделать видимой (`True`) панель инструментов
`Enabled`	Свойство управляет доступом к кнопкам панели инструментов. Если значение свойства равно `False`, то кнопки недоступны
`Dock`	Свойство определяет границу окна, к которой "прикреплена" панель инструментов: к верхней (`Top`), левой (`Left`), нижней (`Bottom`) или правой (`Right`)

ToolTip

Компонент `ToolTip` является вспомогательным компонентом, который используется всеми другими компонентами формы для вывода подсказок при позиционировании указателя мыши на компоненте. Свойства компонента приведены в табл. 2.24.

Таблица 2.24. *Свойства компонента* `ToolTip`

Свойство	Описание
`Active`	Разрешает (`True`) или запрещает (`False`) отображение подсказок
`AutoPopDelay`	Время отображения подсказки
`InitialDelay`	Время, в течение которого указатель мыши должен быть неподвижным, чтобы появилась подсказка
`ReshowDelay`	Время задержки отображения подсказки после перемещения указателя мыши с одного компонента на другой

Timer

Компонент `Timer` представляет собой компонент, который генерирует последовательность событий `Tick`. Компонент является невизуальным. Свойства компонента приведены в табл. 2.25.

Таблица 2.25. *Свойства компонента* `Timer`

Свойство	Описание
`Interval`	Период генерации события `Tick`. Задается в миллисекундах
`Enabled`	Разрешение работы. Разрешает (значение `True`) или запрещает (значение `False`) генерацию события `Tick`

Графика

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть два набора карандашей: стандартный и системный. Также программист может создать свой собственный карандаш, объект типа `Pen`, свойства которого (табл. 2.26) определяют вид линии, которую чертит карандаш.

Таблица 2.26. *Свойства объекта* `Pen`

Свойство	Описание
`Color`	Цвет линии
`Width`	Толщина линии (задается в пикселах)
`DashStyle`	Стиль линии: `DashStyle.Solid` — сплошная; `DashStyle.Dash` — пунктирная, длинные штрихи; `DashStyle.Dot` — пунктирная, короткие штрихи; `DashStyle.DashDot` — пунктирная, чередование длинного и короткого штрихов; `DashStyle.DashDotDot` — пунктирная, чередование одного длинного и двух коротких штрихов; `DashStyle.Custom` — пунктирная линия, вид которой определяет значение свойства `DashPattern`
`DashPattern`	Длина штрихов и промежутков пунктирной линии `DashStyle.Custom`

Стандартный набор карандашей — это цветные карандаши (всего их 141), которые рисуют непрерывную линию толщиной в

один пиксел. Некоторые карандаши из стандартного набора приведены в табл. 2.27.

Таблица 2.27. *Некоторые карандаши из стандартного набора*

Карандаш	Цвет
`Pens.Red`	Красный
`Pens.Orange`	Оранжевый
`Pens.Yellow`	Желтый
`Pens.Green`	Зеленый
`Pens.LightBlue`	Голубой
`Pens.Blue`	Синий
`Pens.Purple`	Фиолетовый
`Pens.Black`	Черный
`Pens.LightGray`	Серый
`Pens.White`	Белый
`Pens.Transparent`	Прозрачный

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. В распоряжении программиста есть четыре типа кистей: стандартные (`Brush`), штриховые (`HatchBrush`), градиентные (`LinearGradientBrush`) и текстурные (`TextureBrush`).

Стандартная кисть закрашивает область одним цветом (сплошная закраска). В стандартном наборе более 100 кистей, некоторые из которых приведены в табл. 2.28.

Таблица 2.28. *Некоторые кисти из стандартного набора*

Кисть	Цвет
`Brushes.Red`	Красный
`Brushes.Orange`	Оранжевый
`Brushes.Yellow`	Желтый

Таблица 2.28 *(окончание)*

Кисть	Цвет
`Brushes.Green`	Зеленый
`Brushes.LightBlue`	Голубой
`Brushes.Blue`	Синий
`Brushes.Purple`	Фиолетовый
`Brushes.Black`	Черный
`Brushes.LightGray`	Серый
`Brushes.White`	Белый
`Brushes.Transparent`	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 2.29 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Таблица 2.29. *Некоторые стили штриховки областей*

Стиль	Штриховка
`HatchStyle.LightHorizontal`	Редкая горизонтальная
`HatchStyle.Horizontal`	Средняя горизонтальная
`HatchStyle.NarrowHorizontal`	Частая горизонтальная
`HatchStyle.LightVertical`	Редкая вертикальная
`HatchStyle.Vertical`	Средняя вертикальная
`HatchStyle.NarrowVertical`	Частая вертикальная
`HatchStyle.LageGrid`	Крупная сетка из горизонтальных и вертикальных линий
`HatchStyle.SmallGrid`	Мелкая сетка из горизонтальных и вертикальных линий
`HatchStyle.DottedGrid`	Сетка из горизонтальных и вертикальных линий, составленных из точек

Таблица 2.29 *(окончание)*

Стиль	Штриховка
`HatchStyle.ForwardDiagonal`	Диагональная штриховка "вперед"
`HatchStyle.BackwardDiagonal`	Диагональная штриховка "назад"
`HatchStyle.Percent05` — `HatchStyle.Percent90`	Точки (степень заполнения 5 %, 10 %, ..., 90 %)
`HatchStyle.HorizontalBrick`	"Кирпичная стена"
`HatchStyle.LargeCheckerBoard`	"Шахматная доска"
`HatchStyle.SolidDiamond`	"Бриллиант" ("Шахматная доска", повернутая на 45°)
`HatchStyle.Sphere`	"Пузырьки"
`HatchStyle.ZigZag`	"Зигзаг"

Градиентная кисть (`LinearGradientBrush`) представляет собой прямоугольную область, цвет точек которой зависит от расстояния до границы. Обычно градиентные кисти двухцветные, т. е. цвет точек по мере удаления от границы постепенно меняется с одного на другой. Цвет может меняться вдоль горизонтальной или вертикальной границы области. Возможно также изменение цвета вдоль линии, образующей угол с горизонтальной границей.

Текстурная кисть (`TextureBrush`) представляет собой рисунок, который обычно загружается во время работы программы из файла (bmp, jpg или gif) или из ресурса. Закраска области текстурной кистью выполняется путем дублирования рисунка внутри области.

Графические примитивы

Любая картинка, чертеж, схема представляют собой совокупность графических *примитивов*: точек, линий, окружностей, дуг, текста и др.

Вычерчивание графических примитивов на графической поверхности (`Graphics`) выполняют соответствующие методы (табл. 2.30).

Таблица 2.30. *Некоторые методы вычерчивания графических примитивов*

Метод	Действие
`DrawLine(Pen, x1, y1, x2, y2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `x1`, `y1`, `x2`, `y2` — координаты точек начала и конца линии
`DrawRectangle(Pen, x, y, w, h)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника
`FillRectangle(Brush, x, y, w, h)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника
`DrawEllipse(Pen, x, y, w, h)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметры `x`, `y`, `w`, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`FillEllipse(Brush, x, y, w, h)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметры `x`, `y`, `w`, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`DrawPolygon(Pen, P)`	Рисует контур многоугольника. Параметр `Pen` определяет цвет, толщину и стиль линии границы многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника
`FillPolygon(Brush, P)`	Рисует закрашенный многоугольник. Параметр `Brush` определяет цвет и стиль закраски внутренней области многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника

Таблица 2.30 *(окончание)*

Метод	Действие
`DrawString(str, Font, Brush, x, y)`	Выводит на графическую поверхность строку текста. Параметр `Font` определяет шрифт; `Brush` — цвет символов; `x` и `y` — точку, от которой будет выведен текст
`DrawImage(Image, x, y)`	Выводит на графическую поверхность иллюстрацию. Параметр `Image` определяет иллюстрацию; `x` и `y` — координату левого верхнего угла области вывода иллюстрации

Типы данных

Целый тип

Целые типы приведены в табл. 2.31.

Таблица. 2.31. *Основные целые типы*

Тип	Диапазон	Формат
`System.SByte`	-128..127	8 бит, со знаком
`System.Int16`	-32768..32767	16 бит, со знаком
`System.Int32`	-2147483648..2147483647	32 бита, со знаком
`System.Int64`	$-2^{63}..2^{63}$	64 бит, со знаком
`System.Byte`	0..255	8 бит, без знака
`System.UInt16`	0..65535	16 бит, без знака
`System.UInt32`	0..4294967295	32 бит, без знака

Вещественный тип

Вещественные типы приведены в табл. 2.32.

Таблица 2.32. *Основные вещественные типы*

Тип	Диапазон	Значащих цифр	Байт
System.Single	$-1.5 \cdot 10^{45} .. 3.4 \cdot 1038$	7—8	4
System.Double	$-5.0 \cdot 10^{324} .. 1.7 \cdot 10^{308}$	15—16	8

Символьный тип

Основным символьным типом является тип System.Char.

Строковый тип

Основным строковым типом является тип System.String.

Функции

В этом разделе приведено описание некоторых наиболее часто используемых функций.

Функции преобразования

В табл. 2.33 приведены функции, обеспечивающие преобразование строки символов в число, изображением которого является строка. Если строка не может быть преобразована в число (например, из-за того что содержит недопустимый символ), то возникает исключение типа FormatException. Функции преобразования принадлежат пространству имен System.Convert.

Таблица 2.33. *Функции преобразования строки в число*

Функция	Значение
ToSingle(s) ToDouble	Дробное типа Single, Double
ToByte(s) ToInt16(s) ToInt32(s) ToInt64(s)	Целое типа Byte, Int16, Int32, Int64

Таблица 2.33 *(окончание)*

Функция	Значение

`ToUInt16(s)`
`ToUInt32(s)`
`ToUInt64(s)`
	Целое типа `UInt16`, `UInt32`, `UInt64`

Преобразование числа в строку символов обеспечивает метод (функция) `ToString`. В качестве параметра функции `ToString` можно указать символьную константу (табл. 2.34), которая задает формат строки-результата.

Таблица 2.34. *Форматы отображения чисел*

Константа	Формат	Пример
`c`, `C`	`Currency` — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28р.
`e`, `E`	`Scientific` (`exponential`) — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004
`f`, `F`	`Fixed` — число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55055,28
`g`, `G`	`General` — универсальный формат. Похож на `Number`, но разряды не разделены на группы	55055,275
`n`, `N`	`Number` — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28

Таблица 2.34 *(окончание)*

Константа	Формат	Пример
r, R	Roundtrip — без округления. В отличие от формата N, этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55055,2775

Функции манипулирования строками

Некоторые функции (методы) манипулирования строками приведены в табл. 2.35.

Таблица 2.35. *Функции манипулирования строками*

Функция (метод)	Значение
s.Length	Длина (количество) символов строки. Строго говоря, Length — это не метод, а свойство объекта String
s.IndexOf(c)	Положение (номер) символа c в строке s (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
s.LastIndexOf(c)	Положение (от конца) символа c в строке s (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
s.IndexOf(st)	Положение подстроки st в строке s (символы строки s нумеруются с нуля). Если указанной подстроки в строке нет, то значение функции равно минус один
s.Trim	Строка, полученная из строки s путем "отбрасывания" пробелов, находящихся в начале и в конце строки
s.Substring(n)	Подстрока, выделенная из строки s, начиная с символа с номером n (символы строки s нумеруются с нуля). Если значение n больше, чем количество символов в строке, то возникает исключение ArgumentOutOfRangeException

Таблица 2.35 *(окончание)*

Функция (метод)	Значение
`s.Substring(n,k)`	Подстрока длиной `k` символов, выделенная из строки `s`, начиная с символа с номером `n` (символы строки `s` нумеруются с нуля). Если значение `n` больше, чем количество символов в строке или если `k` больше, чем `len – n` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`s.Insert(pos,st)`	Строка, полученная путем вставки в строку s строки st. Параметр pos задает номер символа строки s, после которого вставляется строка st.
`s.Remove(pos,n)`	Строка, полученная путем удаления из строки `s` цепочки символов (подстроки). Параметр `pos` задает положение подстроки, параметр `n` — количество символов, которое надо удалить. Если значение `pos` больше, чем количество символов в строке или если `n` больше, чем `len – pos` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`s.Replace(old,new)`	Строка, полученная из строки `s` путем замены всех символов `old` на символ `new`
`s.ToUpper()`	Строка, полученная из строки `s` путем замены строчных символов на прописные
`s.ToLower()`	Строка, полученная из строки `s` путем замены прописных символов на строчные
`s.Split(sep)`	Массив строк, полученный разбиением строки `s` на подстроки. Параметр `sep` (массив типа `Char`) задает символы, которые используются методом `Split` для обнаружения границ подстрок

Функции манипулирования датами и временем

Некоторые функции манипулирования строками приведены в табл. 2.36 (d – некоторый объект типа `DataTime`).

Таблица 2.36. *Функции манипулирования датами и временем*

Функция (метод)	Значение
`DateTime.Now`	Структура типа `DateTime`. Текущие дата и время
`DateTime.Today`	Структура типа `DateTime`. Текущая дата
`d.ToString()`	Строка вида dd.mm.yyyy mm:hh:ss (например, 05.06.2006 10:00)
`d.ToString(f)`	Строка — дата и/или время. Вид строки определяет параметр `f`. Полному формату даты и времени соответствует строка `dd.MM.yyyy hh:mm:ss`
`d.Day`	День (номер дня месяца)
`d.Month`	Номер месяца (1 — январь, 2 — февраль и т. д.)
`d.Year`	Год
`d.Hour`	Час
`d.Minute`	Минуты
`d.DayOfYear`	Номер дня года (отсчет от 1 января)
`d.DayOfWeek`	День недели
`d.ToLongDateString`	"Длинная" дата. Например: 5 июня 2006
`d.ToShortDateString`	"Короткая" дата. Например: 05.06.2006
`d.ToLongTimeString`	Время в формате hh:mm:ss
`d.ToShotrTimeString`	Время в формате hh:mm
`d.AddDays(n)`	Дата, отстоящая от `d` на `n` дней. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`d.AddMonths(n)`	Дата, отстоящая от `d` на `n` месяцев. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`d.AddYears(n)`	Дата, отстоящая от `d` на `n` лет. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`d.AddHours(n)`	Дата (время), отстоящая от `d` на `n` часов. Положительное `n` "сдвигает" дату вперед, отрицательное — назад

Таблица 2.36 *(окончание)*

Функция (метод)	Значение
d.Minutes(т)	Дата (время), отстоящая от d на n минут. Положительное n "сдвигает" дату вперед, отрицательное — назад

Функции манипулирования каталогами и файлами

Функции (методы) манипулирования каталогами и файлами (табл. 2.37) принадлежат пространству имен System.IO. При выполнении операций с каталогами и файлами возможны исключения. (di —объект типа DirectoryInfo, fi —объект типа FileInfo, sr — объект типа StreamReader, sw — объект типа StreamWriter)

Таблица. 2.37. *Функции (методы) манипулирования каталогами и файлами*

Функция (метод)	Результат (значение)
DirectoryInfo (Path)	Создает объект типа DirectoryInfo, соответствующий каталогу, заданному параметром Path
di.GetFiles(fn)	Формирует список файлов каталога — массив объектов типа FileInfo. Каждый элемент массива соответствует файлу каталога, заданного объектом di (тип DirectoryInfo). Параметр fn задает критерий отбора файлов
di.Exists	Проверяет, существует ли в системе каталог. Если каталог существует, то значение функции равно True, в противном случае — False
di.Create (dn)	Создает каталог. Если путь к новому каталогу указан неправильно, возникает исключение
fi.CopyTo(Path)	Копирует файл, заданный объектом fi (тип FileInfo), в каталог и под именем, заданным параметром Path. Значение метода — объект типа FileInfo, соответствующий файлу, созданному в результате копирования

Таблица. 2.37 *(продолжение)*

Функция (метод)	Результат (значение)
`fi.MoveTo(Path)`	Перемещает файл, заданный объектом `fi` (тип `FileInfo`), в каталог и под именем, заданным параметром `Path`
`fi.Delete()`	Уничтожает файл, соответствующий объекту `fi` (тип `FileInfo`)
`StreamReader (fn)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`. Поток открывается для чтения в формате UTF-8
`StreamReader (fn,encd)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`. Поток открывается для чтения в кодировке, заданной параметром `encd` (объект типа `System.Text.Encoding`). Для чтения текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System.Text.Encoding.GetEncoding(1251)`
`sr.Read()`	Читает символ из потока `sr` (объект типа `StremReader`). Значение метода — код символа
`sr.Read(buf, p, n)`	Читает из потока `sr` (объект типа `StremReader`) символы и записывает их в массив символов `buf` (тип `Char`). Параметр `p` задает номер элемента массива, в который будет помещен первый прочитанный символ, параметр `n` — количество символов, которое надо прочитать
`sr.ReadToEnd()`	Читает весь текст из потока `sr` (объект типа `StremReader`). Значение метода — прочитанный текст
`sr.ReadLine()`	Читает строку из потока `sr` (объект типа `StremReader`). Значение метода — прочитанная строка

Таблица. 2.37 *(окончание)*

Функция (метод)	Результат (значение)
`StreamWriter (fn)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`. Поток открывается для записи в формате (кодировке) UTF-8
`StreamWriter (fn,a,encd)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`. Поток открывается для записи в кодировке, заданной параметром `encd` (объект типа `System.Text.Encoding`). Для записи текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System.Text.Encoding.GetEncoding(1251)`. Параметр `a` задает режим записи: `True` — добавление в файл, `False` — перезапись
`sw.Write(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`. В качестве `v` можно использовать выражение строкового или числового типа
`sw.WriteLine(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`, и символ "новая строка". В качестве `v` можно использовать выражение строкового или числового типа
`sw.WriteLine`	Записывает в поток `sw` (объект типа `StreamWriter`) символ "новая строка"
`s.Close`	Закрывает поток `s`

Математические функции

Некоторые математические функции, принадлежащие пространству имен `Math`, приведены в табл. 2.38.

Таблица. 2.38. *Математические функции*

Функция (метод)	Значение
`Abs(n)`	Абсолютное значение `n`
`Sign(n)`	1 (если `n` — положительное) минус 1 (если `n` — отрицательное)
`Round(n,m)`	Дробное, полученное путем округления `n` по известным правилам. Параметр `m` задает количество цифр дробной части
`Ceiling(n)`	Дробное, полученное путем округления `n` "с избытком"
`Floor(n)`	Дробное, полученное путем округления `n` "с недостатком " (отбрасыванием дробной части)
`Log(n)`	Натуральный логарифм `n` (логарифм `n` по основанию `E`, где `E` — постоянная, значение которой равно 2.7182818284590452354)
`Log(n,m)`	Логарифм `n` по основанию `m`
`Log10(n)`	Десятичный логарифм `n`
`Exp(n)`	Экспонента `n` (число "`E`" в степени `n`)
`Cos(α)`	Косинус угла α, заданного в радианах
`Sin(α)`	Синус угла α, заданного в радианах
`Tan(α)`	Тангенс угла α, заданного в радианах
`ASin(n)`	Арксинус `n` — угол (в радианах), синус которого равен `n`
`ACos(n)`	Арккосинус `n` — угол (в радианах), косинус которого равен `n`
`ATan(n)`	Арктангенс `n` — угол (в радианах), тангенс которого равен `n`
`Sqrt(n)`	Квадратный корень из `n`
`r.Next(hb)`	Случайное число из диапазона 0 .. (`hb-1`). `r` — объект типа `System.Random`

Величина угла тригонометрических функций должна быть задана в радианах. Чтобы преобразовать величину угла из градусов в радианы, надо значение, выраженное в градусах, умножить на π/180, где π – число "Пи". Вместо константы 3.1415926 (значения числа "Пи") можно использовать именованную константу `PI`.

События

В табл. 2.39 приведено краткое описание некоторых событий.

Таблица 2.39. *События*

Событие	Происходит
`Click`	При щелчке кнопкой мыши
`Closed`	Возникает сразу за событием `Closing`
`Closing`	Возникает как результат щелчка на системной кнопке **Закрыть окно**
`DblClick`	При двойном щелчке кнопкой мыши
`Enter`	При получении элементом управления фокуса
`KeyDown`	При нажатии клавиши клавиатуры. Сразу за событием `KeyDown` возникает событие `KeyPress`. Если нажатая клавиша удерживается, то событие `KeyDown` возникает еще раз. Таким образом, пара событий `KeyDown` — `KeyPress` генерируется до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `KeyUp`)
`KeyPress`	При нажатии клавиши клавиатуры. Событие `KeyPress` возникает сразу после события `KeyDown`
`KeyUp`	При отпускании нажатой клавиши клавиатуры
`Leave`	При потере элементом управления фокуса
`Load`	В момент загрузки формы. Используется для инициализации программы
`MouseDown`	При нажатии кнопки мыши
`MouseMove`	При перемещении мыши
`MouseUp`	При отпускании кнопки мыши
`Paint`	При появлении окна (элемента управления) на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном, и в других случаях. Только процедура обработки события `Paint` имеет доступ к свойству `Graphics`, методы которого обеспечивают отображение графики
`Resize`	При изменении размера окна. Если размер формы увеличивается, то сначала происходит событие `Paint`, затем — `Resize`. Если размер формы уменьшается, то происходит только событие `Resize`

Исключения

В табл. 2.40 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 2.40. *Типичные исключения*

Исключение	Возникает
`FormatException` — ошибка преобразования	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
`IndexOutOfRangeException` — выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
`ArgumentOutOfRangeException` — выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками
`OverflowException` – переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю
`FileNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частой причиной является отсутствие требуемого файла (ошибка в имени файла или обращение к несуществующему или недоступному устройству)
`DirectoryNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частой причиной является отсутствие требуемого каталога (ошибка в имени каталога или обращение к несуществующему или недоступному устройству)

ПРИЛОЖЕНИЕ

Описание прилагаемого компакт-диска

Прилагаемый к книге компакт-диск содержит проекты, приведенные в книге, а также дистрибутив Turbo C# Explorer (дистрибутив помещен с разрешения CodeGear — подразделения Borland Software Corporation).

Проекты находятся в каталоге Projects (каждый проект в отдельном подкаталоге). Помимо файлов, образующих проект, в каталоге проекта есть readme-файл, который содержит информацию о проекте, и exe-файл (в подкаталоге bin\debug), что позволяет запустить программу с компакт-диска (при условии, что на компьютере установлена Microsoft .NET Framework for Windows), не загружая проект в Turbo C#. Следует обратить внимание на то, что приложения работы с базами данных и ASP.NET-приложения требуют предварительной настройки системы (процесс настройки описан в соответстствующем readme-файле).

Для активной работы (чтобы иметь возможность вносить изменения в программы) скопируйте каталоги проектов на жесткий диск компьютера.

Дистрибутив Turbo C# Explorer находится в каталоге TurboCsharp. Там же находится файл Установка.doc, в котором подробно описан процесс установки Turbo C# Explorer.

Предметный указатель

S

System.Byte 226
System.Int16 226
System.Int32 226
System.Int64 226
System.SByte 226
System.Single 227
System.UInt16 226
System.UInt32 226

Б

Битовый образ 86

Г

Графический примитив 224

И

Исключение:
◊ FileNotFound 237
◊ FormatException 237
◊ OverflowException 237

К

Карандаш 221
◊ стандартный набор 221
Кисть:
◊ градиентная 224
◊ стандартная 222
◊ текстурная 224
◊ штриховая 223
Компонент:
◊ BdpAdapter 95
◊ BdpConnection 95
◊ BdpDataAdapter 95
◊ Button 25, 199
◊ CheckBox 16, 202
◊ CheckedListBox 204
◊ ComboBox 14, 201
◊ ContextMenu 169, 202
◊ DataGrid 95
◊ DataSet 95, 122
◊ FolderBrowserDialog 30
◊ FontDialog 56
◊ GroupBox 205
◊ HelpProvider 44
◊ ImageList 101, 206
◊ Label 206
◊ ListBox 30, 207
◊ MainMenu 56
◊ NotifyIcon 169, 209
◊ NumericUpDown 37, 209
◊ OpenDialog 56
◊ OpenFileDialog 122, 210
◊ Panel 211
◊ PictureBox 30, 212
◊ PrintDocument 56
◊ ProgressBar 215
◊ RadioButton 19, 213
◊ SaveDialog 56
◊ SaveFileDialog 122, 215
◊ StatusBar 40
◊ TextBox 8, 22, 218

Компонент *(прод.)*:
◊ Timer 34, 169, 220
◊ ToolBar 219
◊ ToolTip 102, 220

Л

Линия:
◊ стиль 221
◊ толщина 221
◊ цвет 221

М

Метод:
◊ DrawLine 66
◊ DrawPie 66
◊ DrawRectangle 66
◊ DrawString 66, 68
◊ FillPie 66
◊ FillRectangle 66
◊ Invalidate 90

С

Событие:
◊ Click 236
◊ DblClick 236
◊ Enter 236
◊ KeyDown 236
◊ KeyPress 236
◊ KeyUp 11, 236
◊ MouseDown 236
◊ MouseMove 236
◊ MouseUp 236
◊ Paint 236

Т

Тип:
◊ System.Byte 226
◊ System.Double 227
◊ System.Int16 226
◊ System.Int32 226
◊ System.Int64 226
◊ System.SByte 226
◊ System.Single 227
◊ System.UInt16 226
◊ System.UInt32 226

Ф

Формат:
◊ Currency 228
◊ Fixed 228
◊ General 228
◊ Number 228
◊ Roundtrip 229
◊ Scientific 228
◊ без округления 229
◊ научный 228
◊ универсальный 228
◊ фиксированная точка 228
◊ финансовый 228

Функция:
◊ ToByte 227
◊ ToDouble 227
◊ ToInt16 227
◊ ToInt32 227
◊ ToInt64 227
◊ ToSingle 227
◊ ToUInt16 228
◊ ToUInt32 228
◊ ToUInt64 228

Ц

Цвет линии 221

Ш

Штриховка 223