



ЧАСТЬ II

Автор курса:

доцент каф. ИСТ
Кислицын Дмитрий Игоревич

Содержание 1

- * Работа с формами и элементами управления
- * Обработка событий
- * Создание второй формы
- * Передача данных между формами
- * Динамическое добавление/удаление контрола
- * Создание массива контролов
- * Обработка однотипный событий массива контролов
- * Класс Graphics. Построение графика функции
- * Работа с текстовым файлом
- * Выборка данных из XML-файла
- * Взаимодействие с MS Access
- * Взаимодействие с MS SQL Server
- * Основы LINQ

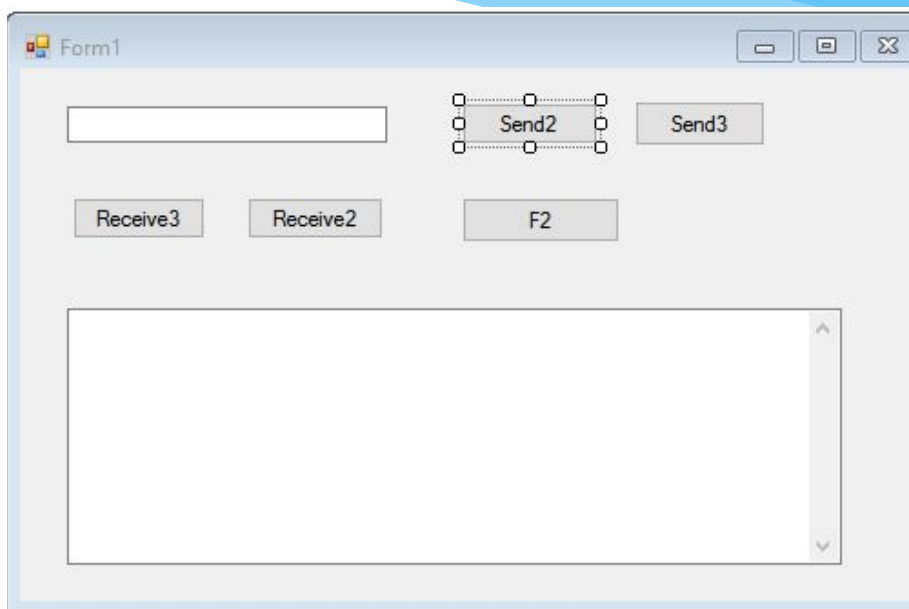
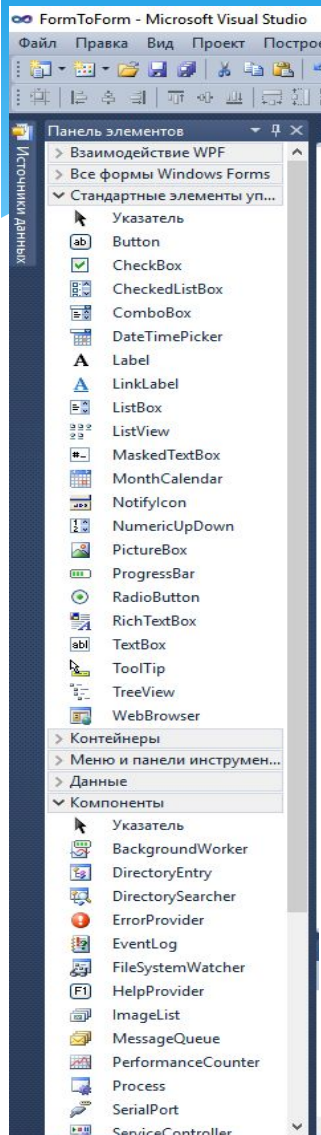
Содержание 2

- * **Основы LINQ**
- * Общая форма запроса
- * Отбор запрашиваемых значений с помощью оператора where
- * Сортировка результатов запроса с помощью оператора orderby
- * Операторы select и from
- * Группирование результатов с помощью оператора group
- * Продолжение запроса с помощью оператора into
- * Применение оператора let для создания временной переменной в запросе
- * Объединение двух последовательностей с помощью оператора join
- * Методы запроса
- * Режимы выполнения запросов: отложенный и немедленный
- * Использование LINQ для работы с XML-файлами
- * Использование LINQ для работы с базами данных

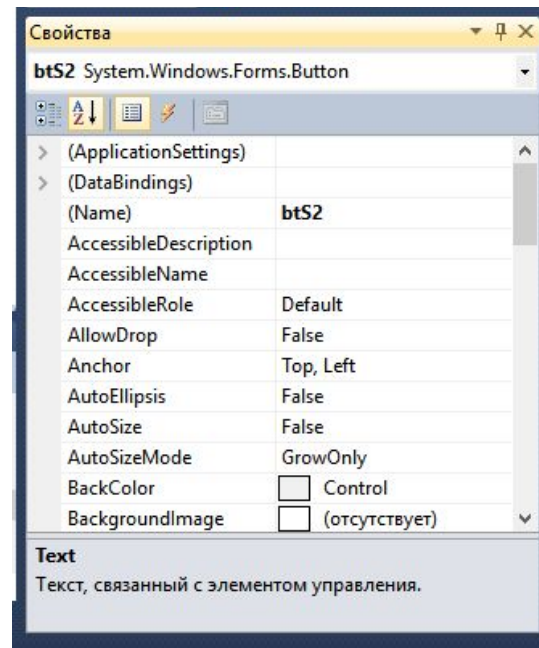
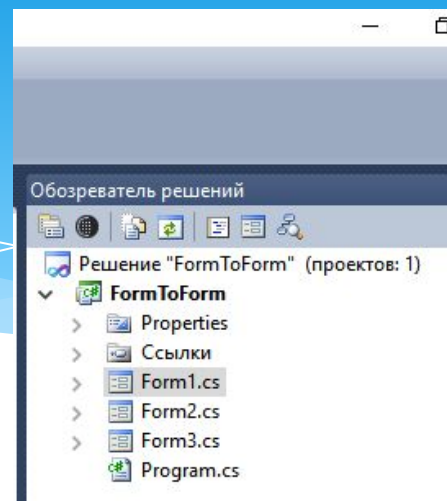
Что почитать

- * Герберт Шилдт - C# 4.0. Полное руководство, 2011
- * <https://docs.microsoft.com/ru-ru/dotnet/articles/csharp/programming-guide/index> - MSDN. Руководство по программированию на C#
- * Казанский А. А. Объектно-ориентированное программирование на языке Microsoft Visual C# в среде разработки Microsoft Visual Studio 2008 и .NET Framework 4.3 : Учебное пособие и практикум, Москва : Московский государственный строительный университет, ЭБС АСВ, 2013
- * Культин Н. Б. Microsoft Visual C# в задачах и примерах. – СПб.: БХВ-Петербург, 2009. – 320 с.
- * <https://metanit.com/sharp/> - программирование на C# и платформе .NET

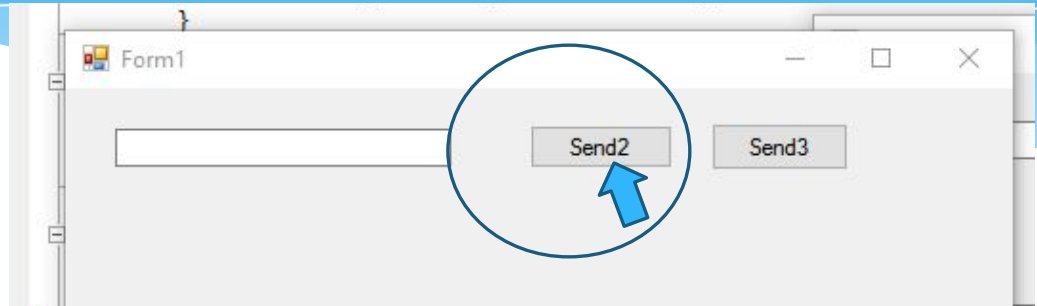
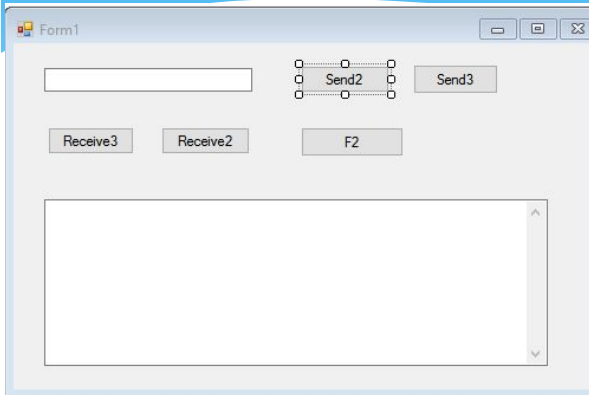
Работа с формами и элементами управления



```
private void btS2_Click(object sender, EventArgs e)
{
    f2.setText(textBox1.Text, this.Name );
}
```

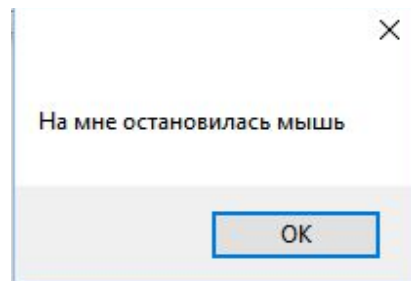


Обработка событий



```
private void btS2_Click(object sender, EventArgs e)
{
    f2.setText(textBox1.Text, this.Name );
}
```

```
private void btS2_Hover(object sender, EventArgs e)
{
    MessageBox.Show("На мне остановилась мышь");
}
```



Property	Value
KeyUp	
PreviewKeyDown	
Layout	
MarginChanged	
Move	
PaddingChanged	
Resize	
MouseDown	
MouseEnter	
MouseHover	btS2_Hover
MouseLeave	

Click
Возникает при щелчке элемента управления.

Создание второй формы

Любая форма представляет из себя класс, унаследованный от `Form`. Экземпляр главной формы создается в файле `Program.cs` по умолчанию.

Чтобы отобразить вторую форму, надо создать экземпляр этого класса (`Form2`), например в обработчике события главной.

`ShowDialog()` - блокирует главную форму, т.е. управление вернется в нее, только по закрытию второй формы,

`Show()` - просто отображает вторую форму (будут доступны обе формы)

```
Form2 f2;  
private void button1_Click(object sender, EventArgs e)  
{  
    if (f2 == null || f2.IsDisposed) // f2 не создана или была закрыта  
    {  
        f2 = new Form2();  
        f2.Show();  
    }  
}
```

Для передачи данных между формами необходимо передать в конструктор `f2` текущую форму

```
f2 = new Form2(this);
```

Передача данных из Form1 в Form2

Изменение в конструкторе *Form2*

```
Form1 f1;  
public Form2 (Form1 f)  
    {  
        f1 = f;  
        InitializeComponent ();  
    }
```

Передача из textBox1 Form1 в textBox1 Form2

```
try  
{  
    f2.data = this.textBox1.Text; // data - public string data, т.к.  
} // TextBox по умолчанию private  
catch  
{  
    f2 = new Form2 (this);  
    f2.data = this.textBox1.Text;  
    f3.Show ();  
}
```

```
public string data  
{  
    get  
    {  
        return this.textBox1.Text;  
    }  
    set  
    {  
        this.textBox1.Text = value;  
    }  
}
```


Передача данных из Form1 в Form2

The screenshot displays three overlapping Windows Forms windows:

- Form1:** Contains a text box with the text "Из формы 1 в форму 3". Below it are two buttons labeled "Send2" and "Send3". A list box at the bottom shows the following messages:
 - 09.02.2017 0:21:36 Отправитель Form2
Из формы 2 в форму 1
 - 09.02.2017 0:21:44 Отправитель Form3
Из формы 3 в форму 1
- Form2:** Contains a text box with the text "Из формы 2 в форму 1". Below it are two buttons labeled "Send1" and "Send3". A list box at the bottom shows the following messages:
 - 09.02.2017 0:21:18 Отправитель Form1
Из формы 1 в форму 2
 - 09.02.2017 0:21:48 Отправитель Form3
Из формы 3 в форму 2
- Form3:** Contains a text box with the text "Из формы 3 в форму 2". Below it are two buttons labeled "Send1" and "Send2". A list box at the bottom shows the following messages:
 - 09.02.2017 0:21:23 Отправитель Form1
Из формы 1 в форму 3
 - 09.02.2017 0:21:33 Отправитель Form2
Из формы 2 в форму 3

Динамическое добавление/удаление контрола

(Добавление)

Пример динамического создания кнопки:

```
System.Windows.Forms.Button button1 = new System.Windows.Forms.Button();  
// создаем контрол  
button1.Location = new System.Drawing.Point(101, 50);  
// устанавливаем необходимые свойства  
button1.Name = "button1";  
button1.Size = new System.Drawing.Size(75, 23);  
button1.TabIndex = 0; button1.Text = "button1";  
button1.UseVisualStyleBackColor = true;  
button1.Click += new System.EventHandler(button1_Click);  
// button1_Click - функция обработчик события нажатия на кнопку  
Controls.Add(button1);  
// добавляем на форму
```

Динамическое добавление/удаление контроля

(Удаление)

```
Controls.Remove(button1);  
button1.Dispose(); // освобождает ресурсы
```

Создание массива контролов

```
TextBox[] tb = new TextBox[10];  
for (int i = 0; i < tb.Length; i++) {  
    tb[i] = new System.Windows.Forms.TextBox();  
    tb[i].Location = new System.Drawing.Point(101, 50 + i * 30);  
    tb[i].Name = "textBox" + i.ToString();  
    tb[i].Size = new System.Drawing.Size(75, 23);  
    tb[i].TabIndex = i;  
    tb[i].Text = "textBox" + i.ToString();  
    Controls.Add(tb[i]);  
}
```

Обработка однотипный событий массива

КОНТРОЛОВ

```
Label [,] lbs;  
private void button1_Click(object sender, EventArgs e)  
{  
    lbs = new Label [n,n];  
    for (i = 0; i < n; i++)  
        for (j = 0; j < n; j++)  
        {  
            Label lb=new Label();  
            lb.Location = new System.Drawing.Point(50 + i * 25, 50 + j * 25);  
            lb.Name = "lb" + i.ToString() + j.ToString();  
            lb.Size = new System.Drawing.Size(25, 25);  
            lb.Text = "[ ]";  
            lb.Click += new System.EventHandler(lb_Click);  
            Controls.Add(lb);  
            lbs[i,j]=lb;  
        }  
    }  
void lb_Click(object sender, EventArgs e)  
{  
    Label lb = (Label )sender;  
    lb.text="X";  
}
```

Класс Graphics

Класс **Graphics** предоставляет методы для рисования объектов на устройстве отображения. А **Graphics** связан с конкретным контекстом устройства

```
e.Graphics.DrawString(text, sfont, Brushes.BlueViolet, x, y);
```

▲ 3 из 6 ▼ void Graphics.DrawString(**string** s, System.Drawing.Font font, Brush brush, float x, float y)

Создает указываемую текстовую строку в заданном месте с помощью определяемых объектов System.Drawing.Brush и System.Drawing.Font.
s: Строка для рисования.

```
public partial class Form1 : Form
```

```
{
```

```
    string text = "TEXT";
```

```
    public Form1()
```

```
    {
```

```
        Initialize Component();
```

```
    }
```

```
    private void Form_resize(object sender, EventArgs e)
```

```
    {
```

```
        this.Refresh();
```

```
    }
```

```
    private void Form_Paint(object sender, PaintEventArgs e)
```

```
    {
```

```
        Font sfont = new Font("Tahoma", 16, FontStyle.Italic);
```

```
        //sfont=this.Font;
```

```
        int w = (int)e.Graphics.MeasureString(text, sfont).Width;
```

```
        int h = (int)e.Graphics.MeasureString(text, this.Font).Height;
```

```
        int x = (this.ClientSize.Width - w) / 2;
```

```
        int y = (this.ClientSize.Height - h) / 2;
```

```
        e.Graphics.DrawString(text, sfont, Brushes.BlueViolet, x, y);
```

```
    }
```

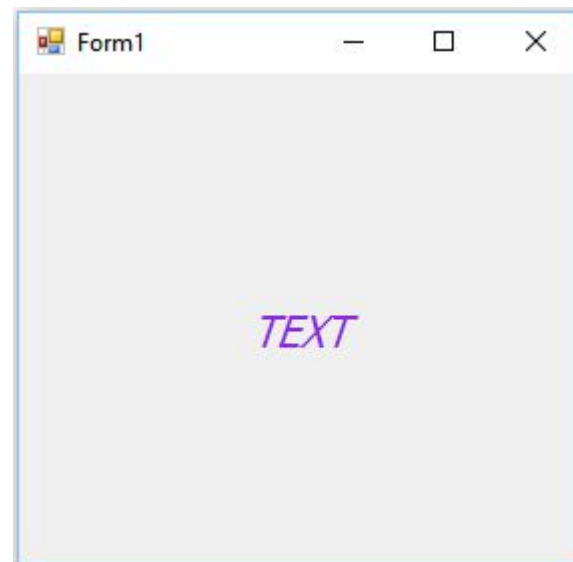
```
}
```

SizeF Graphics.MeasureString(string text, Font font, int width, StringFormat format) (+ перепрыжок: 6)

Измеряет указанную строку при ее отображении с заданным шрифтом System.Drawing.Font и отформатированную с помощью заданного формата System.Drawing.StringFormat.

Исключения:

System.ArgumentException



Класс Graphics

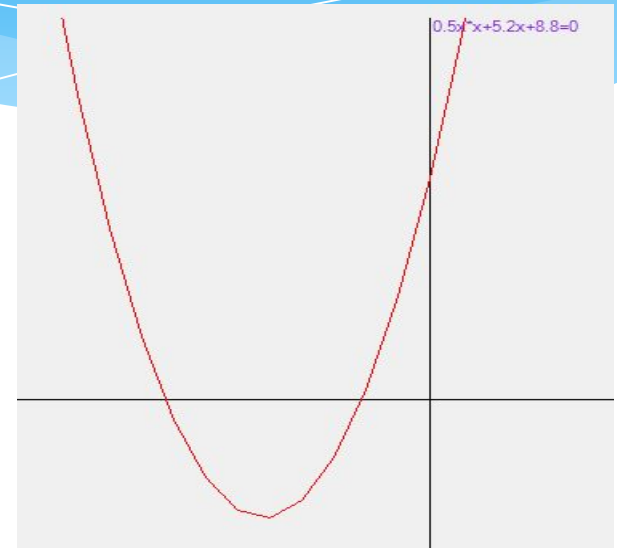
▲ 3 из 4 ▼ void Graphics.DrawLine(**Pen pen**, float x1, float y1, float x2, float y2)

Проводит линию, соединяющую две точки, задаваемые парами координат.

pen: Структура *System.Drawing.Pen*, определяющая цвет, ширину и стиль линии.

```
private void panel1_Paint(object sender, PaintEventArgs e)
{
    int W = pictureBox1.Width, H = pictureBox1.Height;
    int halfW = W / 2, halfH = H / 2;
    e.Graphics.DrawLine(Pens.Black, halfW, 0, halfW, H);
    e.Graphics.DrawLine(Pens.Black, 0, halfH, W, halfH);
}
```

```
private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
{
    a = hScrollBarA.Value / 10.0;
    tbA.Text = a.ToString();
    pictureBox1.Refresh();
}
```



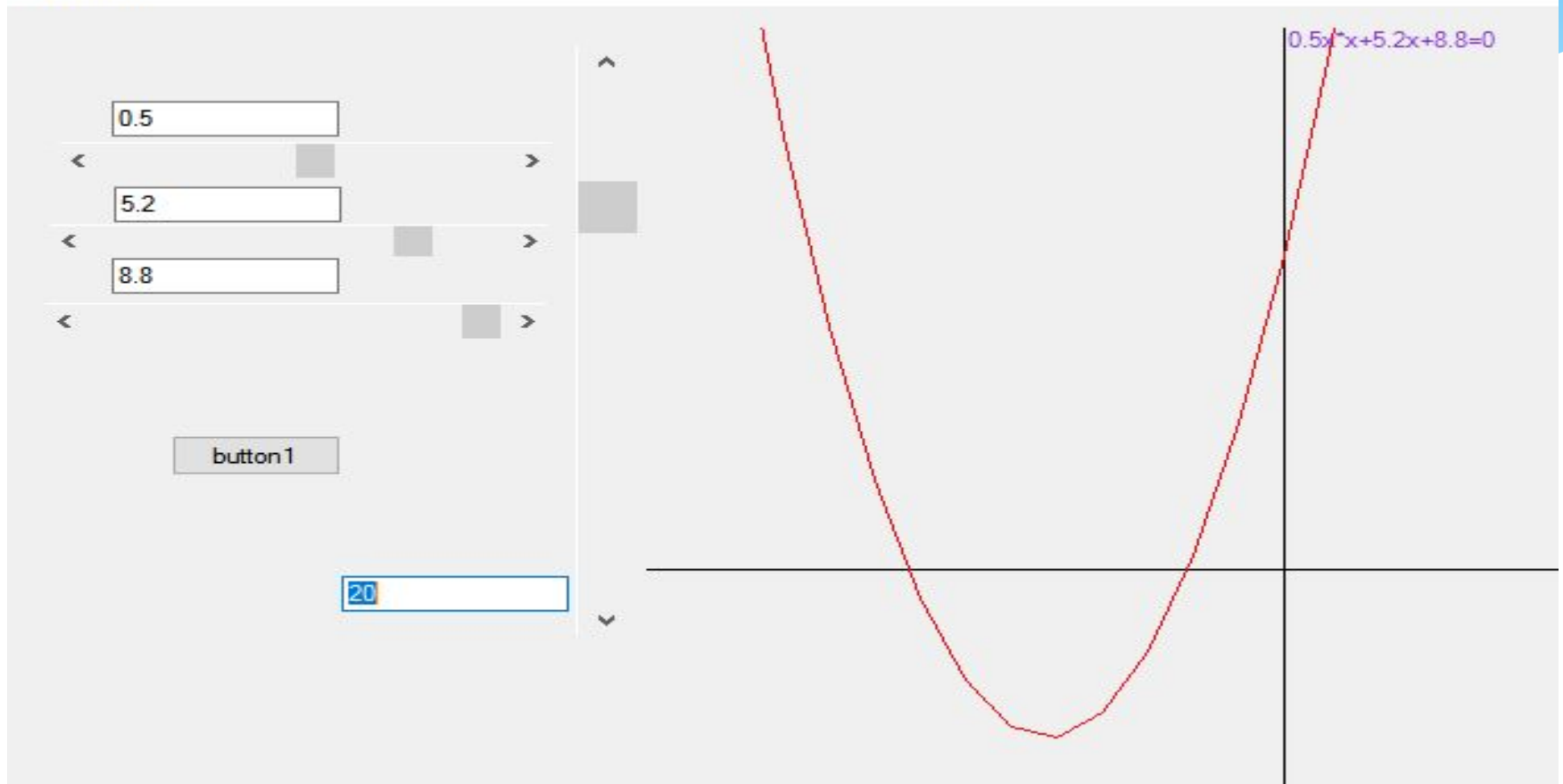
Класс Graphics

▲ 3 из 4 ▼ `void Graphics.DrawLine(Pen pen, float x1, float y1, float x2, float y2)`

Проводит линию, соединяющую две точки, задаваемые парами координат.

pen: Структура `System.Drawing.Pen`, определяющая цвет, ширину и стиль линии.

Form1



Класс Graphics

Построение круговой диаграммы

Построение границы сектора

```
▲ 4 из 4 ▼ void Graphics.DrawPie(Pen pen, int x, int y, int width, int height, int startAngle, int sweepAngle)
```

Заливка сектора

```
▲ 3 из 3 ▼ void Graphics.FillPie(Brush brush, int x, int y, int width, int height, int startAngle, int sweepAngle)
```

// длина дуги

sweepAngle = (int)(360 * p[i]); доля

```
▲ 3 из 4 ▼ void Graphics.FillRectangle(Brush brush, float x, float y, float width, float height)
```



// задать цвет сектора

switch (i)

{

case 0:

brush = Brushes.YellowGreen;

break;

case 1:

brush = Brushes.Gold;

break;

case 2:

brush = Brushes.Pink;

break;

}

Класс Graphics

Построение круговой диаграммы

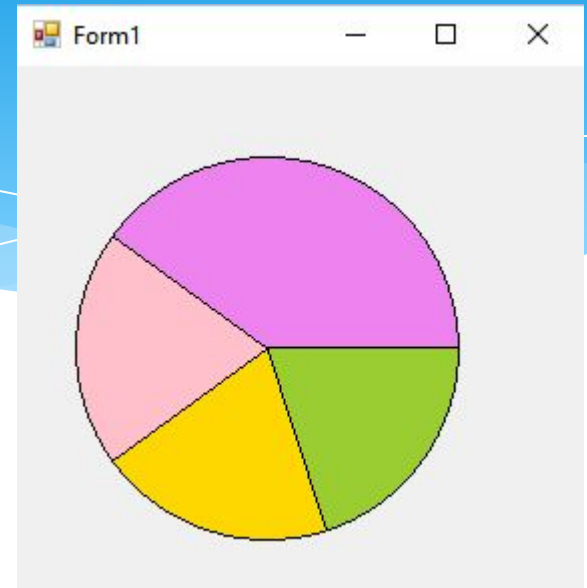
```
private void Form1_Paint(object sender, PaintEventArgs e)
```

```
{  
    // графическая поверхность  
    Graphics g = e.Graphics;  
    // диаметр диаграммы  
    int d = ClientSize.Height - 70;  
    int xo = 30;  
    int yo = (ClientSize.Height - d) / 2 + 10;  
    // длина дуги сектора  
    int swe;  
    // кисть для заливки сектора диаграммы  
    Brush fBrush = Brushes.White;  
    // начальная точка дуги сектора  
    int sta = 0;  
    // рисуем диаграмму  
    for (int i = 0; i < 4; i++)  
    {
```

```
        // длина дуги  
        swe = (int)(360 * 0.2);  
        // задать цвет сектора  
        switch (i)  
        {
```

```
            case 0:  
                fBrush = Brushes.YellowGreen;  
                break;  
            case 1:  
                fBrush = Brushes.Gold;  
                break;  
            case 2:  
                fBrush = Brushes.Pink;  
                break;
```

```
            case 3:  
                fBrush = Brushes.Violet;  
                break;  
        }
```



```
        if (i == 4 - 1)  
        {  
            // последний сектор  
            swe = 360 - sta;  
        }  
        // рисуем сектор  
        g.FillPie(fBrush, xo, yo, d, d, sta, swe);  
        // рисуем границу сектора  
        g.DrawPie(System.Drawing.Pens.Black, xo, yo, d, d, sta, swe);  
        // начальная точка дуги для следующего сектора  
        sta = sta + swe;  
    }  
}
```

Работа с текстовым файлом

Для работы с текстовыми файлами проще всего воспользоваться потоками классов **StreamWriter** и **StreamReader**. Для работы с потоками необходимо подключить библиотеку **System.IO**.

Using System.IO;

Режимы работы с файлом:

1. Запись в файл
 - 1) Создание нового файла и запись в него строк
 - 2) Дописывание строк в существующий файл
2. Чтение существующего файла

Работа с текстовым файлом

Запись в новый текстовый файл

1. Создать файл методом **File.CreateText()** и открыть поток для записи в файл через класс **StreamWriter**

```
StreamWriter sw = File.CreateText("test.txt");
```

2. Записать строку(и) используя методы **Write()** и **WriteLine()**

```
sw.WriteLine("Мой первый файл, созданный в C#");
```

3. Закрывать поток используя метод **Close()**

```
sw.Close();
```

Работа с текстовым файлом

Дописывание в существующий файл

1. Создать поток (через класс **StreamWriter**) и указать режим добавления в файл методом **AppendText()**

```
StreamWriter sw = File.AppendText("test.txt");
```

2. Записать строку(и)

```
sw.WriteLine("Добавили вторую строчку в файл");
```

3. Закрыть поток

```
sw.Close();
```

Работа с текстовым файлом

Чтение из текстового файла

1. Открыть поток класса **StreamReader**, привязав его к файлу методом **File.OpenText**(‘имя файла’).

```
StreamReader sr = File.OpenText("test.txt");
```

2. Считать (например, в консоль):

- одну строку из файла методом **ReadLine()**

```
System.Console.WriteLine(sr.ReadLine());
```

- все строки из файла

```
while (true)
```

```
{
```

```
string st = sr.ReadLine();
```

```
if (st==null)
```

```
break;
```

```
System.Console.WriteLine(st);
```

```
}
```

3. Закрыть поток используя метод **Close()**

```
sr.Close();
```

Работа с текстовым файлом

Пример

```
StreamWriter sw = File.CreateText("test.txt");  
sw.WriteLine("Первая строка");  
sw.WriteLine("Вторая строка");  
sw.Close();
```

```
StreamReader sr = File.OpenText("test.txt");  
while (true)  
{  
    string st = sr.ReadLine();  
    if (st==null)  
        break;  
    System.Console.WriteLine(st);  
}  
sw.Close();  
System.Console.ReadLine();
```

Работа с текстовым файлом



```
openFileDialog1.InitialDirectory = "c:\\";
```

```
openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
```

```
openFileDialog1.FilterIndex = 1;
```

```
openFileDialog1.ShowDialog(); // выбор файла
```

```
openFileDialog1.FileName; // содержит путь к выбранному файлу
```


Работа с текстовым файлом

Задание:

- 1) написать класс по работе с текстовым файлом, позволяющий создавать файл, дописывать в файл, читать из файла, удалять файл, копировать файл, шифровать файл, определять число строк в файле;
- 2) на основе разработанного класса написать простой текстовый редактор.

Работа с XML

```
class User
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Company { get; set; }
}
```

```
User user1 = new User { Name = "Bill Gates", Age = 48, Company = "Microsoft" };
User user2 = new User { Name = "Larry Page", Age = 42, Company = "Google" };
List<User> users = new List<User> { user1, user2 };
```

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user name="Bill Gates">
    <company>Microsoft</company>
    <age>48</age>
  </user>
  <user name="Larry Page">
    <company>Google</company>
    <age>48</age>
  </user>
</users>
```

Объявляется XML-документ версии 1.0 с кодировкой utf-8
Корневой элемент **users**
Элемент **user** с атрибутом **name**
и вложенными элементами **company**
и **age**

Атрибут

name

Вложенный элемент

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user>
    <name>Bill Gates</name>
    <company>Microsoft</company>
    <age>48</age>
  </user>
  <user>
    <name>Larry Page</name>
    <company>Google</company>
    <age>48</age>
  </user>
</users>
```

Работа с XML с помощью классов System.Xml

Классы для работы с XML

XmlNode - представляет узел xml. В качестве узла может использоваться весь документ, так и отдельный элемент

XmlDocument - представляет весь xml-документ

XmlElement - представляет отдельный элемент. Наследуется от класса XmlNode

XmlAttribute - представляет атрибут элемента

XmlText - представляет значение элемента в виде текста, то есть тот текст, который находится в элементе между его открывающим и закрывающим тегами

XmlComment - представляет комментарий в xml

XmlNodeList - используется для работы со списком узлов

Свойства класса XmlNode:

- **Attributes** - возвращает объект XmlAttributeCollection, который представляет коллекцию атрибутов
- **ChildNodes** - возвращает коллекцию дочерних узлов для данного узла
- **HasChildNodes** - возвращает true, если текущий узел имеет дочерние узлы
- **FirstChild** - возвращает первый дочерний узел
- **LastChild** - возвращает последний дочерний узел
- **InnerText** - возвращает текстовое значение узла
- **InnerXml** - возвращает всю внутреннюю разметку xml-узла
- **Name** - возвращает название узла (значение свойства Name равно "user")
- **ParentNode** - возвращает родительский узел у текущего узла

Работа с XML с помощью классов System.Xml

Чтение XML-документа

```
using System.Xml;  
class Program
```

```
{  
    static void Main(string[] args)  
    {  
        XmlDocument xDoc = new XmlDocument();  
        xDoc.Load("users.xml");  
        // получим корневой элемент  
        XmlElement xRoot = xDoc.DocumentElement;  
        // обход всех узлов в корневом элементе  
        foreach(XmlNode xnode in xRoot)  
        {  
            // получаем атрибут name  
            if(xnode.ChildNodes.Count>0)  
            {  
                XmlNode attr = xnode.Attributes.GetNamedItem("name");  
                if (attr!=null)  
                    Console.WriteLine(attr.Value);  
            }  
            // обходим все дочерние узлы элемента user  
            foreach(XmlNode childnode in xnode.ChildNodes)  
            {  
                // если узел - company  
                if(childnode.Name=="company")  
                {  
                    Console.WriteLine("Компания: {0}", childnode.InnerText);  
                }  
                // если узел age  
                if (childnode.Name == "age")  
                {  
                    Console.WriteLine("Возраст: {0}", childnode.InnerText);  
                }  
            }  
            Console.WriteLine();  
        }  
    }  
}
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<users>  
  <user name="Bill Gates">  
    <company>Microsoft</company>  
    <age>48</age>  
  </user>  
  <user name="Larry Page">  
    <company>Google</company>  
    <age>42</age>  
  </user>  
</users>
```

```
Bill Gates  
Компания: Microsoft  
Возраст: 48  
  
Larry Page  
Компания: Google  
Возраст: 42
```

Работа с XML с помощью классов System.Xml

Чтение XML-документа

```
List<User> users = new List<User>();

XmlDocument xDoc = new XmlDocument();
xDoc.Load("users.xml");
XmlElement xRoot = xDoc.DocumentElement;
foreach(XmlElement xnode in xRoot)
{
    User user = new User();
    if (xnode.ChildNodes.Count > 0)
    {
        XmlNode attr = xnode.Attributes.GetNamedItem("name");
        if (attr!=null)
            user.Name=attr.Value;
    }
    foreach (XmlNode childnode in xnode.ChildNodes)
    {
        if (childnode.Name == "company")
            user.Company=childnode.InnerText;

        if (childnode.Name == "age")
            user.Age = Int32.Parse(childnode.InnerText);
    }
    users.Add(user);
}
foreach (User u in users)
    Console.WriteLine("{0} {1} - {2}", u.Name, u.Company, u.Age);
```

Users.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user name="Bill Gates">
    <company>Microsoft</company>
    <age>48</age>
  </user>
  <user name="Larry Page">
    <company>Google</company>
    <age>42</age>
  </user>
</users>
```

```
class User
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Company { get; set; }
}
```

Работа с XML с помощью классов System.Xml

Редактирование XML-документа

(добавление, изменение, удаление элементов)

Методы класса **XmlNode**:

- **AppendChild** - добавляет в конец текущего узла новый дочерний узел
- **InsertAfter** - добавляет новый узел после определенного узла
- **InsertBefore** - добавляет новый узел до определенного узла
- **RemoveAll** - удаляет все дочерние узлы текущего узла
- **RemoveChild** - удаляет у текущего узла один дочерний узел и возвращает его

Методы класса **XmlElement**

- **CreateNode** - создает узел любого типа
- **CreateElement** - создает узел типа XmlDocument
- **CreateAttribute** - создает узел типа XmlAttribute
- **CreateTextNode** - создает узел типа XmlTextNode
- **CreateComment** - создает комментарий

Работа с XML с помощью классов System.Xml

Редактирование XML-документа

(добавление элементов)

```
XmlDocument xDoc = new XmlDocument();
xDoc.Load("Users.xml");
XmlElement xRoot = xDoc.DocumentElement;
// создаем новый элемент user
XmlElement userElem = xDoc.CreateElement("user");
// создаем атрибут name
XmlAttribute nameAttr = xDoc.CreateAttribute("name");
// создаем элементы company и age
XmlElement companyElem = xDoc.CreateElement("company");
XmlElement ageElem = xDoc.CreateElement("age");
// создаем текстовые значения для элементов и атрибута
XmlText nameText = xDoc.CreateTextNode("Mark Zuckerberg");
XmlText companyText = xDoc.CreateTextNode("Facebook");
XmlText ageText = xDoc.CreateTextNode("30");

//добавляем узлы
nameAttr.AppendChild(nameText);
companyElem.AppendChild(companyText);
ageElem.AppendChild(ageText);
userElem.Attributes.Append(nameAttr);
userElem.AppendChild(companyElem);
userElem.AppendChild(ageElem);
xRoot.AppendChild(userElem);
xDoc.Save("Users1.xml");
```

Users.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user name="Bill Gates">
    <company>Microsoft</company>
    <age>48</age>
  </user>
  <user name="Larry Page">
    <company>Google</company>
    <age>42</age>
  </user>
</users>
```

Users1.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<users>
  <user name="Bill Gates">
    <company>Microsoft</company>
    <age>48</age>
  </user>
  <user name="Larry Page">
    <company>Google</company>
    <age>42</age>
  </user>
  <user name="Mark Zuckerberg">
    <company>Facebook</company>
    <age>30</age>
  </user>
</users>
```

Работа с XML с помощью классов System.Xml

Редактирование XML-документа

(удаление элементов)

```
XmlDocument xDoc = new XmlDocument();  
xDoc.Load("Users.xml");  
XmlElement xRoot = xDoc.DocumentElement;
```

```
XmlNode firstNode = xRoot.FirstChild;  
xRoot.RemoveChild(firstNode);  
xDoc.Save("Users2.xml");
```

Users.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<users>  
  <user name="Bill Gates">  
    <company>Microsoft</company>  
    <age>48</age>  
  </user>  
  <user name="Larry Page">  
    <company>Google</company>  
    <age>42</age>  
  </user>  
</users>
```

Users2.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<users>  
  <user name="Larry Page">  
    <company>Google</company>  
    <age>42</age>  
  </user>  
</users>
```

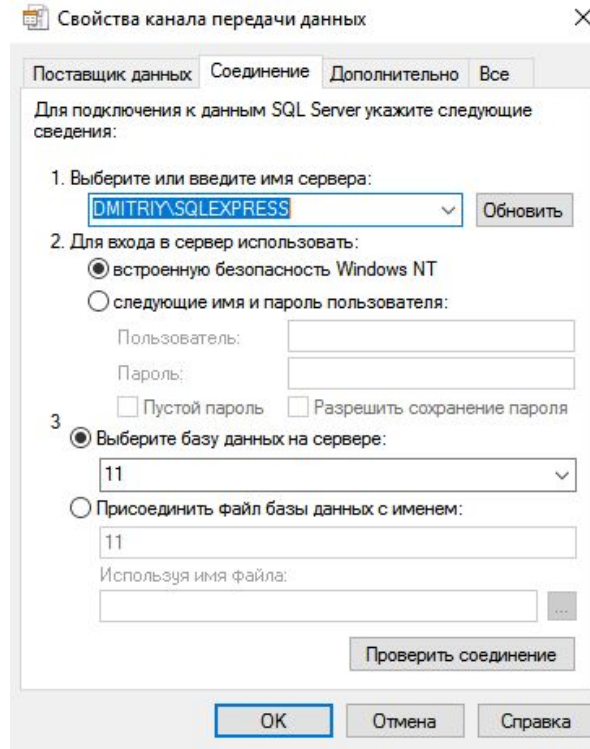
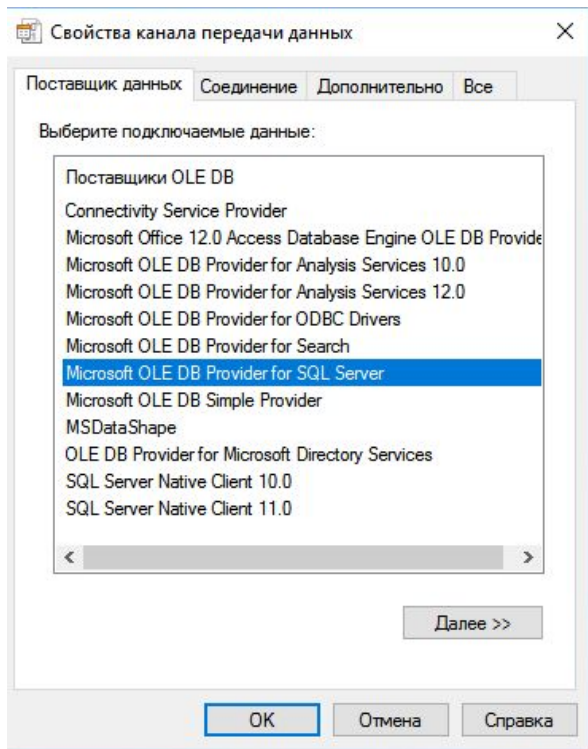

Взаимодействие с БД

Строка подключения

<https://www.connectionstrings.com/>

Создание строки подключения через OLE DB

Создать текстовый файл и изменить расширение с txt на udl.



Может

понадобиться

<https://www.microsoft.com/en-us/download/details.aspx?id=13255>

Microsoft Access
Database Engine 2010
Redistributable

[oledb]

Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=11;
Data Source=DMITRIY\SQLEXPRESS

Взаимодействие с БД

Подключение и выполнение запроса через OLEDB

```
// Подключение через OLEDB
```

```
using System.Data.OleDb;
```

```
string connect="Provider=Microsoft.ACE.OLEDB.12.0; // строка подключения к MS Access  
                Data Source=C:\\Users\\base.accdb; Jet OLEDB:Database Password=0";
```

```
// Создание строки подключения к БД через OleDb
```

```
OleDbConnection cn = new OleDbConnection(connect);
```

```
//Подготовка SQL-запроса
```

```
string query = "SELECT * FROM Table1";
```

```
//Загрузка в DataSet и DataTable
```

```
OleDbDataAdapter adapterOleDb = new OleDbDataAdapter(query, cn);
```

```
DataSet ds = new DataSet();
```

```
adapterOleDb.Fill(ds);
```

```
DataTable dt = ds.Tables[0];
```

```
dataGridView1.DataSource = dt;
```

```
dt.Dispose();
```

```
ds.Dispose();
```

```
cn.Close();
```

Взаимодействие с БД

Подключение и выполнение запроса через SqlConnection

// Подключение через SqlConnection

```
using System.Data.SqlClient;
```

```
string connect= "Data Source= Server ;Initial Catalog= Database ;Integrated Security=True";
```

//Подготовка SQL-запроса

```
string query = "SELECT * FROM Table1";
```

//Загрузка в DataSet и DataTable

```
SqlDataAdapter adapterSql = new SqlDataAdapter(query, connect);
```

```
DataSet ds = new DataSet();
```

```
adapterSql.Fill(ds);
```

```
DataTable dt = ds.Tables[0];
```

//Загрузка в dataGridView

```
dataGridView1.DataSource = dt;
```

```
dt.Dispose();
```

```
ds.Dispose();
```

Взаимодействие с БД

Обновление данных из DataGridView в MS SQL Server через SqlCommandBuilder

```
SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapterSql) ;
```

```
adapterSql.Update(ds);
```

```
// adapter.Update(dt); //обновление только одной таблицы
```

```
ds.Clear(); // очищаем полностью DataSet
```

```
adapterSql.Fill(ds); // перезагружаем данные
```

Взаимодействие с БД

Доступ к ячейке DataTable и DataGridView

Доступ к ячейке DataTable

```
dt.Rows[0][0].ToString();
```

Доступ к ячейке DataGridView

```
dataGridView1[0, 0].Value.ToString();
```

Делегаты

Делегат представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, то в итоге получается объект, содержащий ссылку на метод, т.е. делегат позволяет вызывать метод, на который он ссылается.

Объект делегата поддерживает три важных фрагмента информации:

- * адрес метода, на котором он вызывается;
- * аргументы (если есть) этого метода;
- * возвращаемое значение (если есть) этого метода.

Определение типа делегата в C#:

delegate возвращаемый_тип имя (список_параметров);

Делегат может служить для вызова любого метода с соответствующей сигнатурой и возвращаемым типом.

Делегаты

```
using System;
namespace ConsoleApplication1 {
delegate int IntOperation (int i, int j); // Создадим делегат
class Program {
static int Sum(int x, int y) {
return x + y;
}
static int Prz(int x, int y) {
return x * y;
}
static int Del(int x, int y) {
return x / y;
}
static void Main() {
// Сконструируем делегат
IntOperation op1 = new IntOperation(Sum);
int result = op1(5, 10);
Console.WriteLine("Сумма: " + result);
// Изменим ссылку на метод
op1 = new IntOperation(Prz);
result = op1(5, 10);
Console.WriteLine("Произведение: " + result);
}
}
}
```

ОСНОВЫ LINQ

- * Общая форма запроса
- * Методы расширения LINQ
- * Лямбда-выражения
- * Фильтрация (Where)
- * Сложные фильтры
- * Проекция
- * Переменные в запросах и оператор let
- * Выборка из нескольких источников
- * Сортировка результатов запроса с помощью оператора orderby
- * Группирование результатов с помощью оператора group
- * Продолжение запроса с помощью оператора into
- * Объединение двух последовательностей с помощью оператора join
- * Методы запроса
- * Режимы выполнения запросов: отложенный и немедленный
- * Использование LINQ для работы с XML-файлами
- * Использование LINQ для работы с базами дан

ОСНОВЫ LINQ

LINQ - технология Microsoft, предназначенная для поддержки запросов к данным всех типов на уровне языка. Эти типы включают массивы и коллекции в памяти, базы данных, документы XML и многое другое

ОСНОВЫ LINQ

```
string[] teams = {"Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона"};
```

```
var selectedTeams = new List<string>();  
foreach(string s in teams)  
{  
    if (s.ToUpper().StartsWith("Б"))  
        selectedTeams.Add(s);  
}  
selectedTeams.Sort();
```

без LINQ

```
foreach (string s in selectedTeams)  
    Console.WriteLine(s);
```

```
using System.Linq
```

```
string[] teams = {"Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ",  
"Барселона"};
```

```
var selectedTeams = from t in teams // определяем каждый объект из teams как t  
                    where t.ToUpper().StartsWith("Б") //фильтрация по критерию  
                    orderby t // упорядочиваем по возрастанию  
                    select t; // выбираем объект
```

с LINQ

```
foreach (string s in selectedTeams)  
    Console.WriteLine(s);
```

ОСНОВЫ LINQ

Простейшее определение запроса LINQ :

from переменная **in** набор_объектов **select** переменная;

Методы расширения LINQ

Select: определяет проекцию выбранных значений

Where: определяет фильтр выборки

OrderBy: упорядочивает элементы по возрастанию

OrderByDescending: упорядочивает элементы по убыванию

ThenBy: задает дополнительные критерии для упорядочивания элементов возрастанию

ThenByDescending: задает дополнительные критерии для упорядочивания элементов по убыванию

Join: соединяет две коллекции по определенному признаку

GroupBy: группирует элементы по ключу

ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь

GroupJoin: выполняет одновременно соединение коллекций и группировку элементов по ключу

Reverse: располагает элементы в обратном порядке

All: определяет, все ли элементы коллекции удовлетворяют определенному условию

Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию

ОСНОВЫ LINQ

Методы расширения LINQ

Contains: определяет, содержит ли коллекция определенный элемент

Distinct: удаляет дублирующиеся элементы из коллекции

Except: возвращает разность двух коллекций, то есть те элементы, которые содержатся только в одной коллекции

Union: объединяет две однородные коллекции

Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обеих коллекциях

Count: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию

Sum: подсчитывает сумму числовых значений в коллекции

Average: подсчитывает среднее значение числовых значений в коллекции

Min: находит минимальное значение

Max: находит максимальное значение

Take: выбирает определенное количество элементов

Skip: пропускает определенное количество элементов

TakeWhile: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно

SkipWhile: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы

Concat: объединяет две коллекции

Zip: объединяет две коллекции в соответствии с определенным условием

ОСНОВЫ LINQ

Методы расширения LINQ

First: выбирает первый элемент коллекции

FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

Single: выбирает единственный элемент коллекции, если коллекция содержит больше или меньше одного элемента, то генерируется исключение

SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

ElementAt: выбирает элемент последовательности по определенному индексу

ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона

Last: выбирает последний элемент коллекции

LastOrDefault: выбирает последний элемент коллекции или возвращает значение по умолчанию

ОСНОВЫ LINQ

```
string[] teams = {"Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ",  
"Барселона"};
```

```
var selectedTeams = from t in teams // определяем каждый объект из teams как t  
                    where t.ToUpper().StartsWith("Б") //фильтрация по критерию  
                    orderby t // упорядочиваем по возрастанию  
                    select t; // выбираем объект
```

```
foreach (string s in selectedTeams)  
    Console.WriteLine(s);
```

```
string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ",  
"Барселона" };
```

```
var selectedTeams = teams.Where(t=>t.ToUpper().StartsWith("Б")).OrderBy(t => t);
```

```
foreach (string s in selectedTeams)  
    Console.WriteLine(s);
```

с использованием лямбда-выражений

ОСНОВЫ LINQ

Лямбда-выражения

Лямбда-выражения определяются как разделенный запятыми список параметров, за которым следует лямбда-операция, а за ней — выражение или блок операторов. Если параметров более одного, входные параметры помещаются в скобки. В C# лямбда-операция записывается как `=>`. Таким образом, лямбда-выражение в C# выглядит подобно следующему:

(параметр1, параметр2, параметр3) => выражение

`x => x` // ввод x возвращает x

`(x, y) => x == y` // ввод x, y возвращает результат логического выражения `x == y`

Или, когда требуется более высокая сложность, может применяться блок операторов:

```
(параметр1, параметр2, параметр3) =>
{
    оператор1;
    оператор2;
    оператор3;
    return(тип_возврата_лямбда_выражения);
}
```

```
(x, y) =>
{
    if(x > y)
        return(x);
    else
        return(y);
}
```

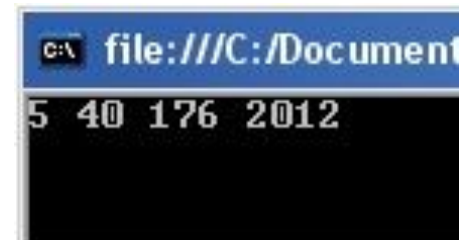
ОСНОВЫ LINQ

Лямбда-выражения

```
string[] numbers = { "40", "2012", "176", "5" };
```

```
// Преобразуем массив строк в массив типа int и сортируем по возрастанию  
// используя LINQ
```

```
int[] nums = numbers.Select(s => Int32.Parse(s)).OrderBy(s => s).ToArray();
```



```
C:\ file:///C:/Document  
5 40 176 2012
```


ОСНОВЫ LINQ

Фильтрация (Where)

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
IEnumerable<int> evens = from i in numbers
                        where i%2==0 && i>10
                        select i;
foreach (int i in evens)
    Console.WriteLine(i);
```

все четные
элементы, которые
больше 10

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
IEnumerable<int> evens = numbers.Where(i => i % 2 == 0 && i > 10);
```

ОСНОВЫ LINQ

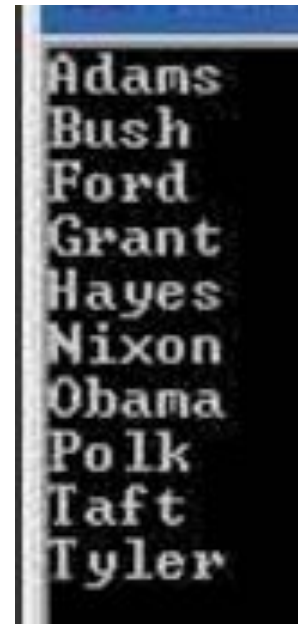
Фильтрация (Where)

```
string[] names = { "Adams", "Arthur", "Buchanan", "Bush", "Carter", "Cleveland",  
"Clinton", "Coolidge", "Eisenhower", "Fillmore", "Ford", "Garfield", "Grant", "Harding",  
"Harrison", "Hayes", "Hoover", "Jackson", "Jefferson", "Johnson", "Kennedy",  
"Lincoln", "Madison", "McKinley", "Monroe", "Nixon", "Obama", "Pierce", "Polk",  
"Reagan", "Roosevelt", "Taft", "Taylor", "Truman", "Tyler", "Van Buren", "Washington",  
"Wilson"};
```

```
// Использование точечной нотации  
IEnumerable<string> sequence = names  
    .Where(n => n.Length < 6)  
    .Select(n => n);
```

```
// Использование синтаксиса выражения запроса  
IEnumerable<string> sequence = from n in names  
    where n.Length < 6  
    select n;
```

```
foreach (string name in sequence) {  
    Console.WriteLine("{0}", name);  
}
```



```
Adams  
Bush  
Ford  
Grant  
Hayes  
Nixon  
Obama  
Polk  
Taft  
Tyler
```

ОСНОВЫ LINQ

Сложные фильтры

```
class User {  
    public string Name { get; set; }  
    public int Age { get; set; }  
    public List<string> Languages { get; set; }  
    public User() {  
        Languages = new List<string>();  
    }  
}
```

```
List<User> users = new List<User>  
{  
    new User {Name="Том", Age=23, Languages = new List<string> {"английский", "немецкий"}},  
    new User {Name="Боб", Age=27, Languages = new List<string> {"английский", "французский"}},  
    new User {Name="Джон", Age=29, Languages = new List<string> {"английский", "испанский"}},  
    new User {Name="Элис", Age=24, Languages = new List<string> {"испанский", "немецкий"}}  
};
```

```
var selectedUsers = from user in users  
                    from lang in user.Languages  
                    where user.Age < 28  
                    where lang == "английский"  
                    select user;
```

```
var selectedUsers = users.SelectMany  
    (u => u.Languages,  
    (u, l) => new { User = u, Lang = l })  
    .Where(u => u.Lang ==  
    "английский" && u.User.Age < 28)  
    .Select(u=>u.User);
```

ОСНОВЫ LINQ

Проекция

Проекция позволяет спроектировать из текущего типа выборки какой-то другой тип. Для проекции используется оператор **select**.

```
class User
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
List<User> users = new List<User>();
users.Add(new User { Name = "Sam", Age = 43 });
users.Add(new User { Name = "Tom", Age = 33 });
```

```
var names = from u in users select u.Name;
```

```
foreach (string n in names)
    Console.WriteLine(n);
```

ОСНОВЫ LINQ

➤ Переменные в запросах и оператор let

```
List<User> users = new List<User>()
{
    new User { Name = "Sam", Age = 43 },
    new User { Name = "Tom", Age = 33 }
};

var people = from u in users
              let name = "Mr. " + u.Name
              select new
              {
                  Name = name,
                  Age = u.Age
              };

foreach (var n in people)
    Console.WriteLine("{0} - {1}", n.Name, n.Age );
```

ОСНОВЫ LINQ

Выборка из нескольких источников

```
class Phone
{
    public string Name { get; set; }
    public string Company { get; set; }
}
class User
{
    public string Name { get; set; }
    public int Age { get; set; }
}

List<User> users = new List<User>()
{
    new User { Name = "Sam", Age = 43 },
    new User { Name = "Tom", Age = 33 }
};
List<Phone> phones = new List<Phone>()
{
    new Phone {Name="Lumia 630", Company="Microsoft" },
    new Phone {Name="iPhone 6", Company="Apple"},
};

var people = from user in users
              from phone in phones
              select new { Name = user.Name, Phone = phone.Name };
foreach (var p in people)
    Console.WriteLine("{0} - {1}", p.Name, p.Phone);
```

```
Sam - Lumia 630
Sam - iPhone 6
Tom - Lumia 630
Tom - iPhone 6
```

ОСНОВЫ LINQ

Использование LINQ для работы с XML-файлами (LINQ to XML)

Пространство имен System.Xml.Linq

Основные классы System.Xml.Linq:

- **XDocument** - представляет весь xml-документ
- **XElement** - представляет отдельный xml-элемент
- **XAttribute** - представляет атрибут xml-элемента
- **XComment** - представляет комментарий

Ключевым классом является **XElement**, который позволяет получать вложенные элементы и управлять ими

Методы класса XElement:

- **Add()** - добавляет новый атрибут или элемент
- **Attributes()** - возвращает коллекцию атрибутов для данного элемента
- **Elements()** - возвращает все дочерние элементы данного элемента
- **Remove()** - удаляет данный элемент из родительского объекта
- **RemoveAll()** - удаляет все дочерние элементы и атрибуты у данного элемента

ОСНОВЫ LINQ

LINQ to XML (Создание XML-файла)

```
XDocument xdoc = new XDocument();  
XElement iphone = new XElement("phone"); // создаем первый элемент  
XAttribute iphoneNameAttr = new XAttribute("name", "iPhone"); // создаем атрибут  
XElement iphoneCompanyElem = new XElement("company", "Apple");  
XElement iphonePriceElem = new XElement("price", "40000");  
iphone.Add(iphoneNameAttr); // добавляем атрибут и элементы в первый элемент  
iphone.Add(iphoneCompanyElem);  
iphone.Add(iphonePriceElem);
```

```
// создаем второй элемент  
XElement galaxy = new XElement("phone");  
XAttribute galaxyNameAttr = new XAttribute("name", "Samsung Galaxy");  
XElement galaxyCompanyElem = new XElement("company", "Samsung");  
XElement galaxyPriceElem = new XElement("price", "33000");  
galaxy.Add(galaxyNameAttr);  
galaxy.Add(galaxyCompanyElem);  
galaxy.Add(galaxyPriceElem);
```

```
XElement phones = new XElement("phones"); // создаем корневой элемент  
phones.Add(iphone); // добавляем в корневой элемент  
phones.Add(galaxy);  
xdoc.Add(phones); // добавляем корневой элемент в документ  
xdoc.Save("phones.xml"); // сохраняем документ
```

```
<?xml version="1.0" encoding="utf-8"?>  
<phones>  
  <phone name="iPhone">  
    <company>Apple</company>  
    <price>40000</price>  
  </phone>  
  <phone name="Samsung Galaxy">  
    <company>Samsung</company>  
    <price>33000</price>  
  </phone>  
</phones>
```


ОСНОВЫ LINQ

• LINQ to XML (Создание XML-файла)

```
XDocument xdoc = new XDocument(new XElement("phones",
    new XElement("phone",
        new XAttribute("name", "iPhone"),
        new XElement("company", "Apple"),
        new XElement("price", "40000")),
    new XElement("phone",
        new XAttribute("name", "Samsung Galaxy"),
        new XElement("company", "Samsung"),
        new XElement("price", "33000"))));
xdoc.Save("phones.xml");
```

```
<?xml version="1.0" encoding="utf-8"?>
<phones>
  <phone name="iPhone">
    <company>Apple</company>
    <price>40000</price>
  </phone>
  <phone name="Samsung Galaxy">
    <company>Samsung</company>
    <price>33000</price>
  </phone>
</phones>
```

ОСНОВЫ LINQ

LINQ to XML (Чтение XML-файла)

```
XDocument xdoc = XDocument.Load("phones.xml");
foreach (XElement phoneElement in xdoc.Element("phones").Elements("phone"))
{
    XAttribute nameAttribute = phoneElement.Attribute("name");
    XElement companyElement = phoneElement.Element("company");
    XElement priceElement = phoneElement.Element("price");

    if (nameAttribute != null && companyElement != null && priceElement != null)
    {
        Console.WriteLine("Смартфон: {0}", nameAttribute.Value);
        Console.WriteLine("Компания: {0}", companyElement.Value);
        Console.WriteLine("Цена: {0}", priceElement.Value);
    }
    Console.WriteLine();
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<phones>
  <phone name="iPhone">
    <company>Apple</company>
    <price>40000</price>
  </phone>
  <phone name="Samsung Galaxy">
    <company>Samsung</company>
    <price>33000</price>
  </phone>
</phones>
```

ОСНОВЫ LINQ

LINQ to XML (Чтение XML-файла)

```
class Phone
{
    public string Name { get; set; }
    public string Price { get; set; }
}
```

```
XDocument xdoc = XDocument.Load("phones.xml");
var items = from xe in xdoc.Element("phones").Elements("phone")
            where xe.Element("company").Value=="Samsung"
            select new Phone
            {
                Name = xe.Attribute("name").Value,
                Price = xe.Element("price").Value
            };

foreach (var item in items)
    Console.WriteLine("{0} - {1}", item.Name, item.Price);
```

```
<?xml version="1.0" encoding="utf-8"?>
<phones>
  <phone name="iPhone">
    <company>Apple</company>
    <price>40000</price>
  </phone>
  <phone name="Samsung Galaxy">
    <company>Samsung</company>
    <price>33000</price>
  </phone>
</phones>
```

```
Galaxy - 33000
```

ОСНОВЫ LINQ

LINQ to XML (Изменение XML-документа)

```
<?xml version="1.0" encoding="utf-8"?>
<phones>
  <phone name="iPhone">
    <company>Apple</company>
    <price>40000</price>
  </phone>
  <phone name="Samsung Galaxy">
    <company>Samsung</company>
    <price>33000</price>
  </phone>
</phones>
```

```
XDocument xdoc = XDocument.Load("phones.xml");
XElement root = xdoc.Element("phones");
```

```
foreach (XElement xe in root.Elements("phone").ToList())
{ // изменяем название и цену
  if (xe.Attribute("name").Value == "Samsung Galaxy")
  {
    xe.Attribute("name").Value = "Samsung Galaxy Note 4";
    xe.Element("price").Value = "31000";
  }
  //если iphone - удаляем его
  else if (xe.Attribute("name").Value == "iPhone")
  {
    xe.Remove();
  }
}
// добавляем новый элемент
root.Add(new XElement("phone",
  new XAttribute("name", "Nokia Lumia 930"),
  new XElement("company", "Nokia"),
  new XElement("price", "19500")));
xdoc.Save("phones1.xml");
// выводим xml-документ на консоль
Console.WriteLine(xdoc);
```

ОСНОВЫ LINQ

LINQtoSQL (Определение контекста данных и моделей)

LINQtoSQL представляет технологию доступа и управления реляционными данными. Данная технология позволяет составлять запросы к БД в удобной форме с помощью операторов LINQ, которые затем трансформируются в sql-выражения. Ключевыми объектами здесь являются:

- сущности, которые хранятся в БД;
- контекст данных;
- запрос LINQ.

Для взаимодействия с БД в LINQtoSQL используются:

- модели (классы, которые сопоставляются с одной из таблиц в БД);
- контекст данных (представленный объектом DataContext (строка подключения), через который идёт работа с БД).

Для использования LINQtoSQL в проекте необходимо добавить библиотеку

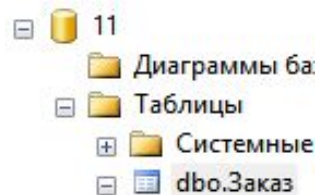
System.Data.Linq.dll

и пространство имён

```
using System.Data.Linq.Mapping;
```

ОСНОВЫ LINQ

★ LINQ to SQL (Определение контекста данных и моделей)



Имя столбца	Тип данных
Код	nchar(10)
Клиент	nchar(10)
Количество	int
Товар	nchar(10)
Доставка	bit
Цена	decimal(18, 0)

	Код	Клиент	Количество	Товар	Доставка	Цена
1	10	Егор	1	Лук	1	20
2	12	Петр	10	Хлеб	0	15
3	123	Иван	112	Масло	1	20
4	2	Дмитрий	5	икра	1	50

```
using System.Data.Linq.Mapping;
namespace LINQtoSQL
{
```

```
[Table(Name = "Заказ")]
public class Class1
{
    [Column(Name = "Код", IsPrimaryKey = true,
            IsDbGenerated = false)]
    public string Id { get; set; }
    [Column(Name = "Клиент")]
    public string Client { get; set; }
    [Column(Name = "Количество")]
    public int Kol_vo { get; set; }
    [Column(Name = "Товар")]
    public string Tovar { get; set; }
    [Column(Name = "Доставка")]
    public bool Dostavka { get; set; }
    [Column(Name = "Цена")]
    public decimal Price { get; set; }
}
}
```

ОСНОВЫ LINQ

LINQ to SQL (Определение контекста данных и моделей)

Атрибут **[Column]** имеет ряд свойств, с помощью которого можно настроить сопоставление столбца:

- **AutoSync**: указывает, как надо извлекать значение столбца после вставки или обновления
- **CanBeNull**: указывает, может ли столбец принимать значение null
- **DbType**: определяет тип столбца. Указывается, если надо создать новую базу данных
- **Expression**: хранит выражение, которое будет использоваться для вычисления значения свойства
- **IsPrimaryKey**: хранит логическое значение и указывает, выполняет ли столбец роль первичного ключа (как в данном случае Id)
- **IsDbGenerated**: хранит логическое значение, которое указывает, будет ли значение столбца генерироваться самой БД
- **IsDiscriminator**: указывает, будет ли столбец разграничителем в системе наследования классов
- **IsVersion**: указывает, будет ли столбец хранить номер версии строки или значение timestamp, которое указывает на время последнего изменения строки
- **Name**: задает имя столбца, с которым будет сопоставляться данное свойство
- **Storage**: указывает на имя приватной переменной, которая будет хранить значение данного столбца
- **UpdateCheck**: определяет, как LINQ to SQL будет решать проблему параллелизма. Если в модели нет свойств со значением `IsVersion=true`, то операциях с данными БД будет сравнивать значения строк из таблицы со новыми значениями

ОСНОВЫ LINQ

★ LINQ to SQL (Определение контекста данных и моделей)

```
class Program
{
    static string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=11;Integrated Security=True";

    private void button1_Click(object sender, EventArgs e)
    {
        DataContext db = new DataContext(connectionString);

        // Получаем таблицу пользователей
        Table<Class1> users = db.GetTable<Class1>();

        foreach (var user in users)
        {
            textBox1.Text += user.Id+"\t"+ user.Client+"\t"+ user.Kol_vo+"\t"+ user.Tovar+"\t"+
                user.Dostavka+"\t"+ user.Price+"\r\n";
        }
    }
}
```

10	Егор	1	Лук	True	20	
12	Петр	10	Хлеб	False	15	
123	Иван	112	Масло	True	20	
2	Дмитрий		5	икра	True	50

ОСНОВЫ LINQ

LINQtoSQL (Фильтрация и сортировка)

```
class Program
{
    static string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=11;Integrated
Security=True";
    private void button2_Click(object sender, EventArgs e)
    {
        DataContext db = new DataContext(connectionString);
        var query = db.GetTable<Class1>().Where(u => u.Price > 18).OrderBy(u => u.Tovar );
// или
// var query = from u in db.GetTable<Class1 >()
// where u.Price > 18
// orderby u.Tovar
// select u;
        textBox1.Text = "";
        foreach (var user in query)
        {
            textBox1.Text += user.Id + "\t" + user.Client + "\t" + user.Kol_vo + "\t" + user.Tovar + "\t" +
            user.Dostavka + "\t" + user.Price + "\r\n";
        }
    }
}
```

2	Дмитрий	5	икра	True	50
10	Егор	1	Лук	True	20
123	Иван	112	Масло	True	20

```
SELECT *
FROM [Заказ] WHERE [Цена] > 18
ORDER BY [Товар]
```

ОСНОВЫ LINQ

LINQtoSQL (Группировка)

```
private void button3_Click(object sender, EventArgs e)
{
    DataContext db = new DataContext(connectionString);
    var query = db.GetTable<Class1>().GroupBy(u => u.Price );
```

// или так

```
// var query = from u in db.GetTable<Class1>()
// group u by u.Price // группировка по возрасту
// into grouped
// select grouped;
textBox1.Text = "";
foreach (var group in query)
{
    textBox1.Text += "\r\nЦена: " + group.Key + "\r\n" ;
    foreach (var user in group)
        textBox1.Text += "\t" + user.Client ;
}
}
```

```
Цена: 15
      Петр
Цена: 20
      Егор   Иван
Цена: 50
      Дмитрий
```

decimal IGrouping<decimal, Class1>.Key
Возвращает ключ объекта System.Linq.IGrouping<TKey, TElement>.

ОСНОВЫ LINQ

LINQtoSQL (Изменение объектов)

При получении объектов из базы данных в контекст `DataContext` эти объекты кэшируются, и у них устанавливается состояние `Unchanged`. Если мы изменим значения свойств какого-либо объекта из полученного набора, то `DataContext` для этого объекта создает копию с измененными значениями и устанавливает у нее статус `ToBeUpdated`.

При вызове метода `SubmitChanges()` контекст данных сверяет значения оригинального объекта и его измененной копии. И если два объекта отличаются, то создается sql-выражение `UPDATE`, с помощью которого происходит обновление объекта в базе данных.

ОСНОВЫ LINQ

LINQtoSQL (Изменение объектов)

```
private void button4_Click(object sender, EventArgs e)
{
    DataContext db = new DataContext(connectionString);
    textBox1.Text = "";
    textBox1.Text += "До обновления\r\n";

    foreach (var user in db.GetTable<Class1>().Take(3))
    {
        textBox1.Text += "\r\n" + user.Client + "\t" + user.Tovar + "\t" + user.Price;
    }

    Class1 user1 = db.GetTable<Class1>().FirstOrDefault();
    user1.Price = 50;
    db.SubmitChanges(); // сохраним изменения

    textBox1.Text += "\r\n После обновления\r\n";
    foreach (var user in db.GetTable<Class1>().Take(5))
    {
        textBox1.Text += "\r\n" + user.Client + "\t" + user.Tovar + "\t" + user.Price;
    }
}
```

До обновления

Егор	Лук	20	
Петр	Хлеб	15	
Иван	Масло		20

После обновления

Егор	Лук	50	
Петр	Хлеб	15	
Иван	Масло		20
Дмитрий		икра	50

ОСНОВЫ LINQ

LINQtoSQL (Добавление объектов)

Чтобы добавить новый объект в базу данных, необходимо вызвать у таблицы в контексте данных метод **InsertOnSubmit()** или **InsertAllOnSubmit()** (если нужно добавить список объектов)

Если в базе данных значение какого-либо столбца, например, столбца для **Id**, должно генерироваться автоматически, то в классе модели атрибут **Column** над соответствующим свойством должен иметь значение **IsDbGenerated = true**

ОСНОВЫ LINQ

LINQtoSQL (Добавление объектов)

```
private void button5_Click(object sender, EventArgs e)
{
    DataContext db = new DataContext(connectionString);
    textBox1.Text = "До добавления";
    foreach (var user in db.GetTable<Class1 >().OrderByDescending(u => u.Id))
    {
        textBox1.Text += "\r\n" + user.Client + "\t" + user.Tovar + "\t" + user.Price;
    }
    Class1 user1 = new Class1 {Id = "10101", Client = "Ronald", Kol_vo= 10, Dostavka = true , Tovar =
    "Кирпич", Price = 34 };

    db.GetTable<Class1 >().InsertOnSubmit(user1);
    db.SubmitChanges();

    textBox1.Text += "\r\nПосле добавления";
    foreach (var user in db.GetTable<Class1 >().OrderByDescending(u => u.Id))
    {
        textBox1.Text += "\r\n" + user.Client + "\t" + user.Tovar + "\t" + user.Price;
    }
}
```

ОСНОВЫ LINQ

★ LINQtoSQL (Удаление объектов)

Для удаления объекта из БД применяется метод **DeleteOnSubmit()** или **DeleteAllOnSubmit()**, если удаляется список объектов

ОСНОВЫ LINQ

LINQtoSQL (Удаление объектов)

```
private void button6_Click(object sender, EventArgs e)
{
    DataContext db = new DataContext(connectionString);

    // получим последний объект для удаления
    var user = db.GetTable<Class1 >().OrderByDescending(u => u.Id).FirstOrDefault();
    textBox1.Text = "";
    if (user != null)
    {
        textBox1.Text += "Удаляемый объект:";
        textBox1.Text += "Id: " + user.Id;
        textBox1.Text += "Client: " + user.Client ;
        textBox1.Text += "Tovar: " + user.Tovar ;

        db.GetTable<Class1>().DeleteOnSubmit(user);
        db.SubmitChanges();
        textBox1.Text += "Объект удален";
    }
}
```


ОСНОВЫ LINQ

LINQtoSQL

(Непосредственное выполнение кода sql в классе DataContext)

Метод **ExecuteCommand()** принимает выполняемое sql-выражение и список параметров

```
DataContext db = new DataContext(connectionString);  
int modifiedRowsNumber = db.ExecuteCommand("DELETE FROM Заказ WHERE Товар={0}", 'Курнич');  
textBox1.Text = "Обработано " + modifiedRowsNumber + " строк";
```

Метод **ExecuteQuery()** принимает выполняемое sql-выражение и список параметров и возвращает результат выполнения запроса SELECT:

```
DataContext db = new DataContext(connectionString);  
IEnumerable<Class1> users = db.ExecuteQuery<Class1>("SELECT * FROM Заказ WHERE Цена>{0}", 15);  
textBox1.Text = "";  
foreach (var user in users)  
    textBox1.Text += user.Id + "|t" + user.Client + "|t" + user.Kol_vo + "|t" + user.Tovar + "|t" +  
        user.Dostavka + "|t" + user.Price + "|r|n";
```

	Код	Клиент	Количество	Товар	Доставка	Цена
1	10	Егор	1	Лук	1	20
3	123	Иван	112	Масло	1	20
4	2	Дмитрий	5	икра	1	50

Решение квадратного уравнения

Техническое задание

1. Написать класс для решения квадратного уравнения вида $ax^2+bx+c=0$.

Входные данные: a, b, c (тип double).

Выходные данные:

- решение уравнения (1 или 2 корня),
- проверка решения (максимальная из двух решений величина ошибки),
- уведомление, если уравнение не имеет действительных корней или не является квадратным.

2. Реализовать проверку входных данных и в случае ввода данных в неверном формате (не число) сообщить об ошибке и запросить данные повторно.

3. После вывода результатов решения уравнения запросить пользователя ввод нового уравнения или выход из программы.

Решение квадратного уравнения

Примерная структура класса

KvadratUr
Class

Поля

a

b

c

er

err

коэффициенты при неизвестных

0, 1, 2 (количество корней); 3 – «неквadrатное уравнение»
«Действительных корней нет» / «Неквadrатное уравнение»

Свойства

x1

x2

содержат решения или сообщения об ошибках

Методы

D

KvadratUr

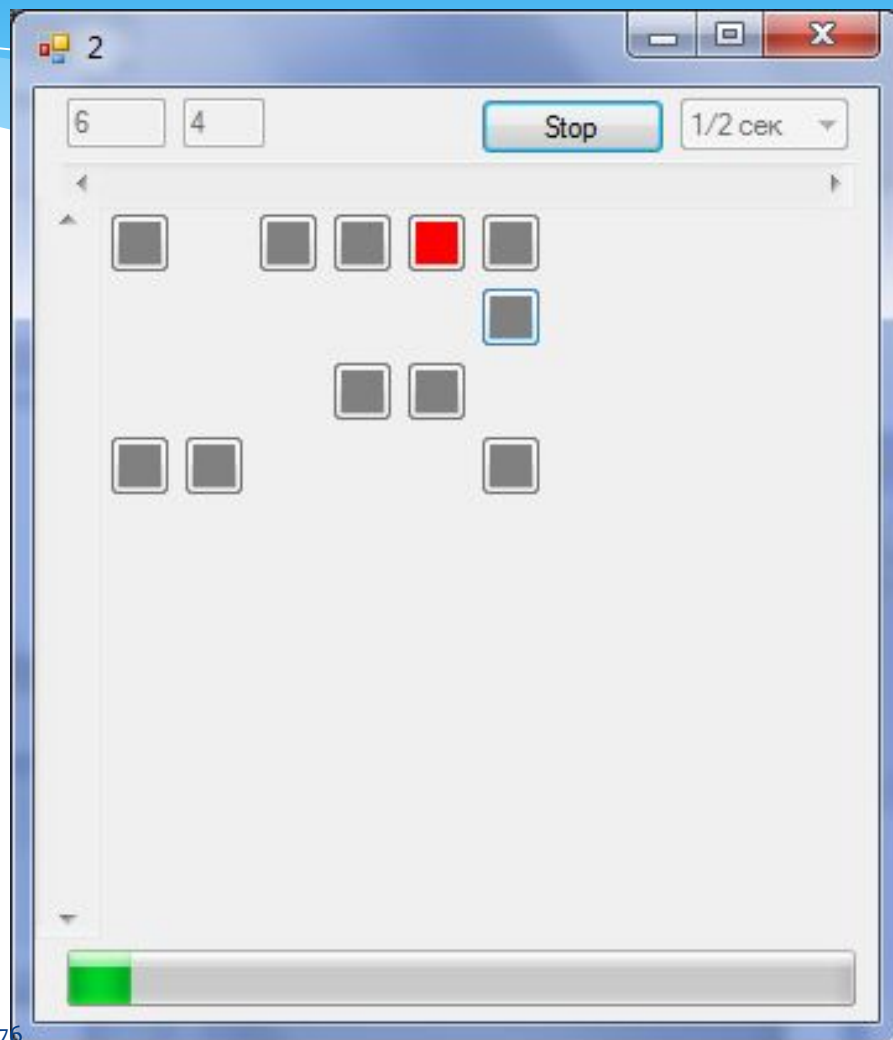
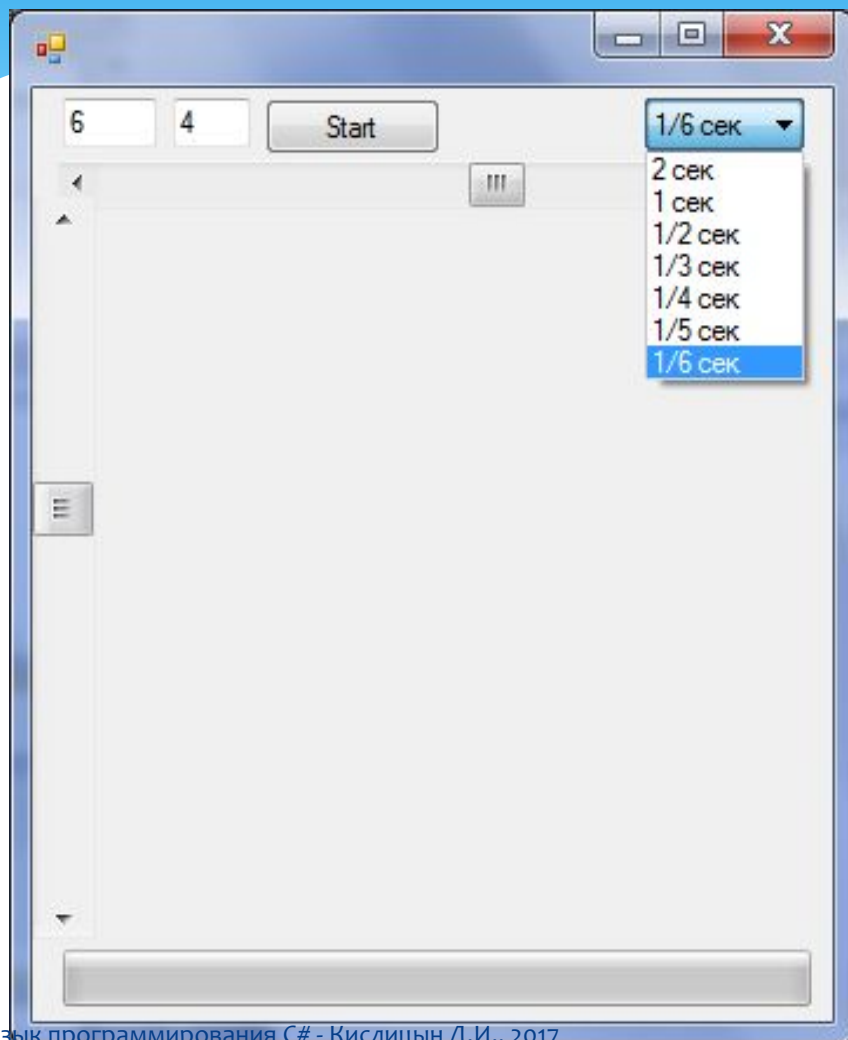
Proverka

вычисление дискриминанта

конструктор, определяющий er и err

вычисление максимального отклонения в решении

Игра «Тир»



Крестики-нолики

