

Лекция 1. СРЕДА ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ VISUAL STUDIO.NET

Введение в дисциплину

В предыдущем семестре мы уже отмечали, что данные, методы, да и сама программа, написанная на языке C#, должна размещаться в классах.

Введение в программирование новой структуры данных это не только количественное увеличение новых типов данных. Появление классов изменило технологию программирования. Если раньше в структурированном программировании основной единицей программы являлись функции и процедуры, то появление классов позволило создавать функционально законченные модули программы, которые объединяли не только данные, но и множество методов их обработки. Основной единицей таких программ стали классы (объекты), а технологии программирования с помощью классов получила название объектно-ориентированного программирования.

Дисциплина «Технологии программирования» рассматривает вопросы, связанные с использованием технологий объектно-ориентированного программирования при проектировании Windows-приложений (сложных программных систем). Использование классов в технологиях объектно-ориентированного программирования (ООП) показывает, что класс может выполнять две функции – выступать в роли модуля программы или типа данных.

Модульность построения – основное свойство Windows-приложений. Разработка больших программных систем, без разделения системы на модули, потребует значительно больше времени, чем системы создаваемые коллективами разработчиков, использующих модульную технологию программирования. В ООП Windows-приложения, разрабатываются по модульному принципу, состоят из классов, являющихся основным видом модуля. Модульность построения – основное средство по ускорению процесса разработки сложных программных систем.

С другой стороны класс это тип данных. Объектно-ориентированная разработка Windows-приложений основана на стиле, называемом проектированием от данных. Проектирование системы сводится к поиску абстракций данных, подходящих для конкретной задачи. Каждая из таких абстракций реализуется в виде класса, который и становится модулем – архитектурной единицей построения программной системы.

В большинстве разрабатываемых Windows-приложений классы выполняют обе функции, так что каждый модуль программной системы имеет вполне определенную смысловую нагрузку. Язык C# допускает как классы, являющиеся типами данных, так и классы, играющие единственную роль модуля. К классам модулям относятся, например, такие классы, как Console, Convert, Math. Классы, играющие единственную роль модуля, объектов создавать не могут. Точнее, существует единственный объект этого

класса, представляющий модуль. Поля и методы этого модуля обычно доступны методам других классов.

При разработке больших программных систем определяющим является среда разработки. Технологии программирования, предоставляемые средой программирования, значительно сокращают время разработки больших программных систем.

Поэтому изучение дисциплины «Технологии программирования» мы начинаем со знакомства со средой программирования Visual Studio.NET при создании приложений для Windows (Windows Forms Application).

1.1 Понятие о событийном управлении Windows

Если рассматривать работу программ, написанных в консольном приложении, то следует отметить, что после их запуска начинают выполняться операторы метода Main().

Особенность поведения программ, написанных для **Windows**, является то, что программы после их запуска переходят в бесконечный цикл ожидания сообщений поступающих от **Windows**.

Сообщения – это реакция операционной системы **Windows** на происходящих в системе событиях – нажатии клавиши на клавиатуре, перемещении курсора мыши, деления на ноль и т.д.

События обрабатывается специальными программами **Windows** – драйверами. Например, драйверы периферийных устройств компьютера (клавиатуры, мыши или таймер). Драйвера создают сообщения, которые пересылаются **Windows**.

В основе работы операционной системы **Windows** лежит принцип событийного управления. Это означает, что и сама система и все приложения, написанные для **Windows**, после запуска ожидают действий пользователя или сообщений операционной системы и реагируют на них определенным образом.

Сообщение **Windows** является записью, которая содержит информацию о том, что произошло и дополнительную информацию (параметры) о произошедшем событии. Например, структура некоторого сообщения может включать дескриптор окна программы, код (идентификатор) сообщения, уточняющие параметры (например, координаты x и y курсора мыши) и время создания сообщения.

Все сообщения, получаемые **Windows**, помещаются в системную очередь сообщений, которая существует в единственном варианте. Далее из системной очереди сообщения распределяются в очереди сообщений отдельных Windows-приложений. При этом для каждого приложения создается своя очередь сообщений. Очередь сообщения приложений может пополняться не только из системной очереди. Любое приложение может послать сообщение любому другому сообщению, в том числе и само себе.

Каждое Windows-приложений имеет непрерывный цикл обработки собственной очереди сообщений поступающих от **Windows**. С помощью этого цикла приложение извлекает сообщения и передает их соответствующим обработчикам сообщений окна приложения. Каждое окно приложения имеет свой цикл обработки сообщений, а также свою функцию окна, которой передаются сообщения, извлеченные из очереди приложения.

Обычно Windows-приложений имеет главное окно, в котором располагаются основные элементы управления – меню, кнопки, полосы прокрутки, флажки и т. д. Работая с приложением, пользователь выбирает строки меню, нажимает кнопки или используете другие элементы управления.

Каждый элемент управления (кнопка или строка меню) имеет свой идентификатор. Когда Вы нажимаете на кнопку или выбираете строку меню, в очередь сообщений приложения **Windows** заносит сообщение, содержащее идентификатор использованного элемента управления. Таким образом операционная система **Windows** направляет сообщение от использованного элемента управления в очередь того приложения, к которому принадлежит данный элемент управления.

В предложенной структуре есть очевидная часть работы программиста – написать обработчики сообщений на некоторые события, например, клик мышкой по кнопке окна вашего приложения.

В приложениях, создаваемых для **Windows**, (File -> New -> Project -> Windows Forms Application), всегда используется два основных типа (классы пространства имен) – Form и Application.

Класс Application управляет поведением приложения – запускает метод Main(), в котором находится цикл обработки сообщений, выполняет необходимые действия при выборке сообщений и корректно завершает работу приложения (файл Program.cs).

Класс Form определяет пользовательский интерфейс приложения – он инициализирует окно формы и готовит приложение к работе (файл Form1.cs).

Более подробно работу этих классов мы будем изучать по мере необходимости при изучении материала дисциплины.

Рассмотрим последовательность действий при создании простого приложения для **Windows**.

1.2 Основные окна среды Visual Studio.Net

Условно процесс создания приложения для **Windows** включает два основных этапа.

Первый этап – этап визуального программирования предназначен для проектирования пользовательского интерфейса, т.е. задания всех необходимых элементов управления. На этом этапе очень важны дизайнерские умения программиста.

На втором этапе необходимо разработать коды обработчиков сообщений приложения, т.е. определить поведение программы при появлении того или иного сообщения от **Windows**.

Чтобы выполнить первый этап разработки приложения для **Windows** необходимо открыть среду разработки **Visual Studio .Net** (File -> New -> Project -> Windows Forms Application) – смотри рисунок 1.1. При выборе проекта вам необходимо указать название папки на рабочем столе, в которой будут размещаться создаваемые средой файлы.

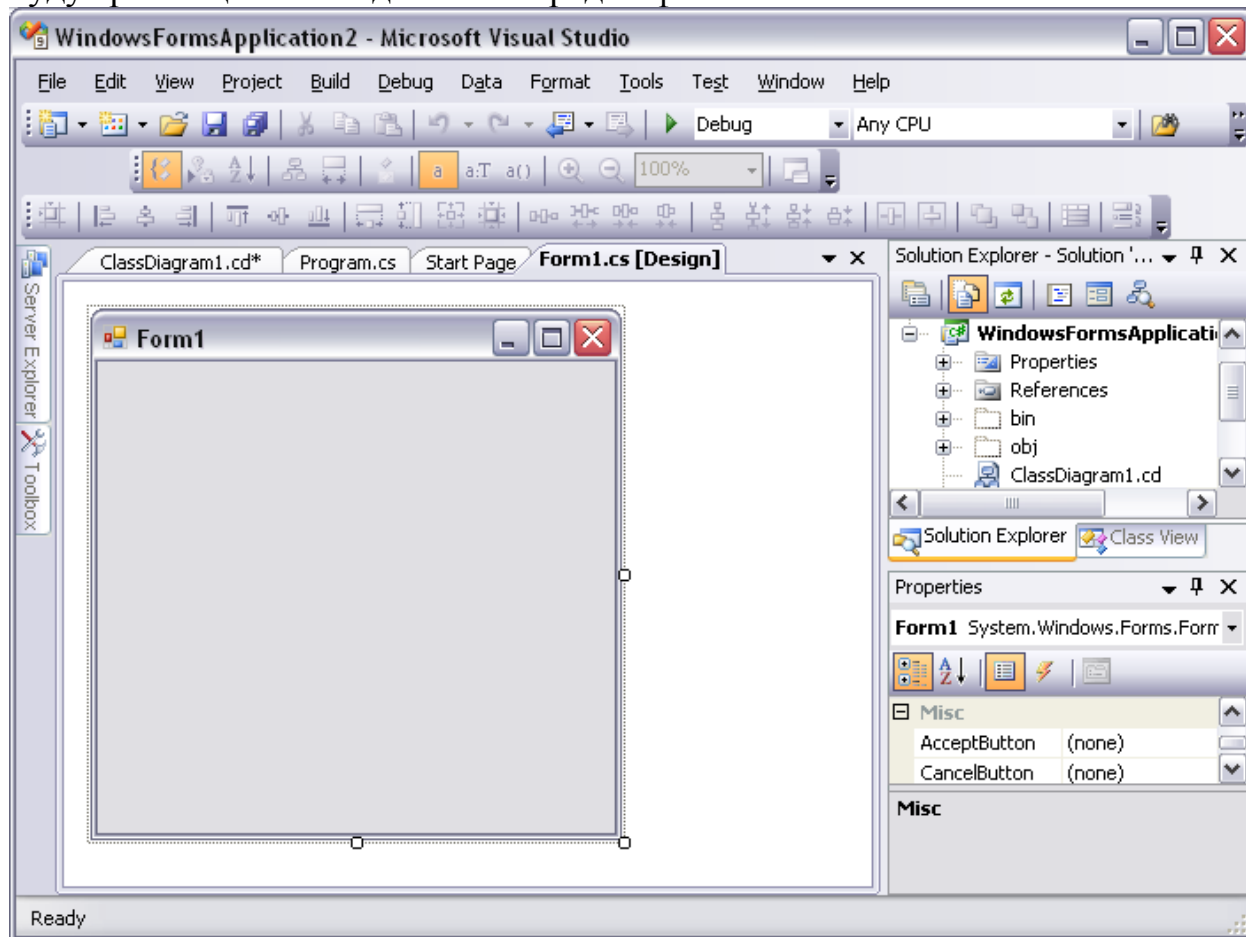


Рисунок 1.1 – Среда разработки **Visual Studio .Net**

В центре рисунка 1.1 расположено окно Form 1, которое относится к пространству имен **System.Windows.Forms**. Пока окно этой формы пустое, но разрабатываемый проект уже можно запустить на выполнение. Для этого необходимо нажать клавишу F5 или выбрать режим работы среды Debug и в нем команду Start. Окно запущенного приложения смотрите на рисунке 1.2.

Пока закроем наш проект и рассмотрим подробнее назначение отдельных окон среды разработки **Visual Studio .Net**.

Окно Form1 имеет 4 страницы, которые используются в различных режимах редактирования проекта, например, Form1.cs[Design] используется для размещения элементов управления на форме, а окно Program.cs для редактирования кода программы.

Окно Properties – свойства элементов управления (в нашем примере это форма) или файлов, предварительно установленных в окне Server Explorer. Страницы этого окна сгруппированы либо по категориям, либо в алфавитном порядке, либо по свойствам и событиям.

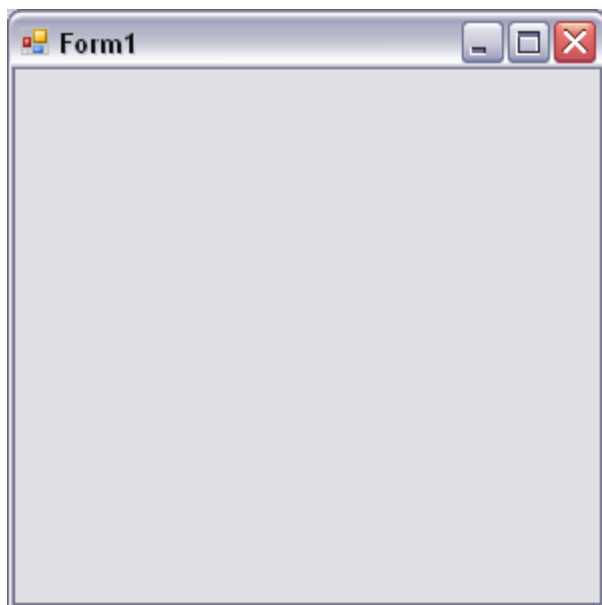


Рисунок 1.2 – Рабочее окно программы

Окно Server Explorer (на рисунке 1.1 слева в свернутом состоянии) служит для доступа к источникам данных на вашем компьютере и на удаленном сервере (информация из журнала событий и др. служб системы).

Окно Toolbox (инструменты – на рисунке 1.1 слева в свернутом состоянии) содержит различные элементы управления, которые мы будем использовать для размещения на форме.

Окно Solution Explorer – на рисунке 1.1 справа, позволяет просматривать и редактировать файлы проекта. Просмотр может осуществляться по файлам или по классам.

Обычно после запуска программы появляется окно ошибок, в котором перечислены номера строк кода программы, в которых допущены ошибки.

Подробное описание работы со средой разработки Visual Studio .Net приведено в книге А.В. Фролов, Г.В. Фролов Визуальное проектирование приложений C#.

1.3 Основные структурные элементы разработки проекта C#

Язык C# является объектно-ориентированным языком программирования и основным понятием является понятие класса.

Определив класс, разработчик получает возможность динамически создавать объекты (переменные или экземпляры) класса.

Для программистов, начинающих работать в объектном стиле, типичной ошибкой является путаница понятий объекта и класса. Нужно с самого начала уяснить разницу. Класс, создаваемый разработчиком, представляет тип – шаблон описание множества объектов. Объект – это динамическое понятие, он создается в ходе выполнения программной системы, реально существует в памяти компьютера и обычно удаляется из памяти компьютера по завершении выполнения проекта. Программист может создать проект, включающий два – три класса, но в ходе работы такого проекта могут динамически появляться сотни объектов, взаимодействующих друг с другом достаточно сложным образом.

Путаница понятий класса и объекта характерна и определений базового класса языка C#. Базовый класс в библиотеке FCL, являющийся прародителем всех классов как библиотечных, так и создаваемых разработчиком, назван именем Object.

Понятие пространство имен.

Пространство имен – это оболочка, которая содержит множество классов, объединенных, как правило, общей тематикой или группой разработчиков. Собственные имена классов внутри пространства имен должны быть уникальны. В разных пространствах могут существовать классы с одинаковыми именами. Полное или уточненное имя класса состоит из уникального имени пространства имен и собственного имени класса. В пространстве имен могут находиться как классы, так и пространства имен.

Пространства имен позволяют задать древовидную структуру на множестве классов большого проекта. Они облегчают независимую разработку проекта большим коллективом разработчиков, каждая группа которого работает в своем пространстве имен.

Пространства имен систематизирует структуру библиотеки FCL, которая содержит большое число различных пространств имен, объединяющих классы определенной тематики. Центральным пространством имен библиотеки FCL является пространство System – оно содержит другие пространства и классы, имеющие широкое употребление в различных проектах. Если рассматривать все пространства имен как некоторое иерархическое дерево классов и пространств имен, то пространство System, а в нем класс Object будут являться корнем такого дерева.

Проект – это единица компиляции. Результатом компиляции проекта является сборка. Каждый проект содержит одно или несколько пространств имен. Как уже говорилось, на начальном этапе создания проекта по заданному типу проекта автоматически строится каркас проекта, состоящий из классов, которые являются наследниками классов, входящих в состав библиотеки FCL. Так, если разработчик указывает, что он хочет построить проект типа "**Windows Forms Application**", то в состав каркаса проекта по умолчанию войдет класс Form1 – наследник библиотечного класса **Form**. Разработчик проекта наполнит созданную форму элементами управления – объектами соответствующих классов, тем самым расширив возможности класса, построенного по умолчанию.

Каждый проект содержит всю информацию, необходимую для построения сборки. В проект входят все файлы с классами, построенные автоматически в момент создания проекта, и файлы с классами, созданные разработчиком проекта. Помимо этого, проект содержит ссылки на пространства имен из библиотеки **FCL**, которые включают классы, используемые в ходе работы программы. Проект содержит ссылки на все подключаемые к проекту **DLL** и другие проекты. В проект входят установки и ресурсы, требуемые для работы. Частью проекта является файл, содержащий описание сборки.

В зависимости от выбранного типа проект может быть выполняемым или невыполняемым. К выполняемым проектам относятся, например, проекты типа **Console** или **Windows**. При построении каркаса выполняемого проекта в него включается класс, содержащий статический метод с именем **Main**. В результате компиляции такого проекта создается **PE-файл (Portable Executable file)** – выполняемый переносимый файл с уточнением **exe**. Напомним, что **PE-файл** может выполняться только на компьютерах, где установлен **Framework .Net**, поскольку это файл с управляемым кодом.

К невыполняемым проектам относятся, например, проекты типа **DLL**. В результате компиляции такого проекта в сборку войдет файл с уточнением **DLL**. Такие проекты (сборки) непосредственно не могут быть выполнены на компьютере. Они присоединяются к выполняемым сборкам, откуда и вызываются методы классов, размещенных в невыполняемом проекте (**DLL**).

Сборка – результат компиляции проекта. Сборка представляет собой коллекцию из одного или нескольких файлов, помеченных номером версии. Каждая сборка разворачивается на компьютере как единое целое. Программист работает с проектами, **CLR** работает со сборками. Сборка позволяет решать вопросы безопасности, так как содержит описание требуемых ей ресурсов и права доступа к элементам сборки. Каждая сборка содержит манифест, включающий полное описание сборки, ее элементов, требуемые ресурсы, ссылки на другие сборки, исполняемые файлы. Благодаря этому описанию **CLR** не требуется никакой дополнительной информации для развертывания сборки, трансляции промежуточного кода и его выполнения. Манифест идентифицирует сборку, специфицирует файлы, требуемые для реализации сборки, специфицирует типы и ресурсы, составляющие сборку, задает зависимости, необходимые в период компиляции для связи с другими сборками, специфицирует множество разрешений, необходимых, чтобы сборка могла выполняться на данном компьютере.

Каждый проект, создаваемый в **Visual Studio.NET 2008**, помещается в некоторую оболочку, называемую Решением – **Solution**. Решение может содержать несколько проектов, как правило, связанных общей темой. Например, в одно Решение можно поместить три проекта: **DLL** с разработанными классами, определяющими содержательную сторону приложения, консольный проект решения задачи и проект под управлением **Windows**. Когда создается новый проект, он может быть помещен в уже

существующее Решение или может быть создано новое Решение, содержащее проект.

1.4 Пример первой учебной программы

Рассмотреть пример использования формы для создания приложения, работающего в системе **Windows**. В качестве первого примера выбрана известная задача вычисления периметра треугольника.

Задача 1.1 В режиме диалога необходимо задать стороны треугольника и вычислить его периметр. После ввода значений сторон треугольника выполнять следующие проверки: все стороны треугольника должны быть больше нуля и сумма любых двух сторон больше третьей. Работу программы сопровождать необходимыми комментариями.

На этапе визуального программирования мы будем использовать три стандартных элемента управления из окна **Toolbox**: статический текст или метка (**Label**), поле ввода или вывода текста – окно редактирования (**TextBox**) и командную кнопку (**Button**).

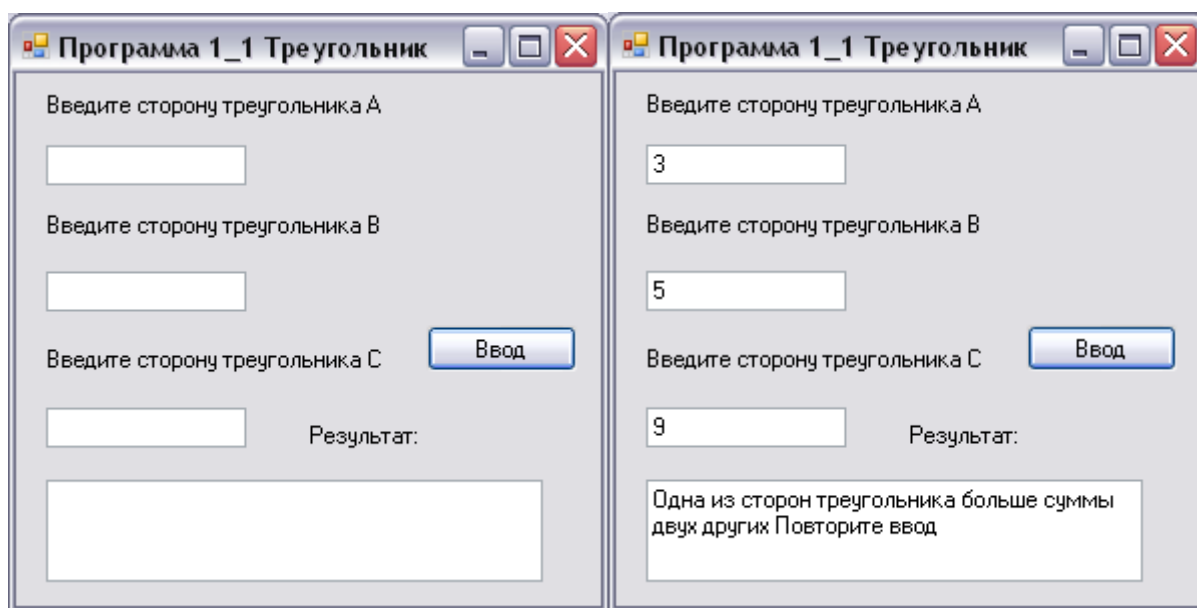
 – статический текст;

 – поле ввода или вывода текста;

 – командная кнопка;

Метки нужны для размещения поясняющих надписей – четыре меток.

Три поля ввода и одно поле вывода результата и одна кнопка управления.



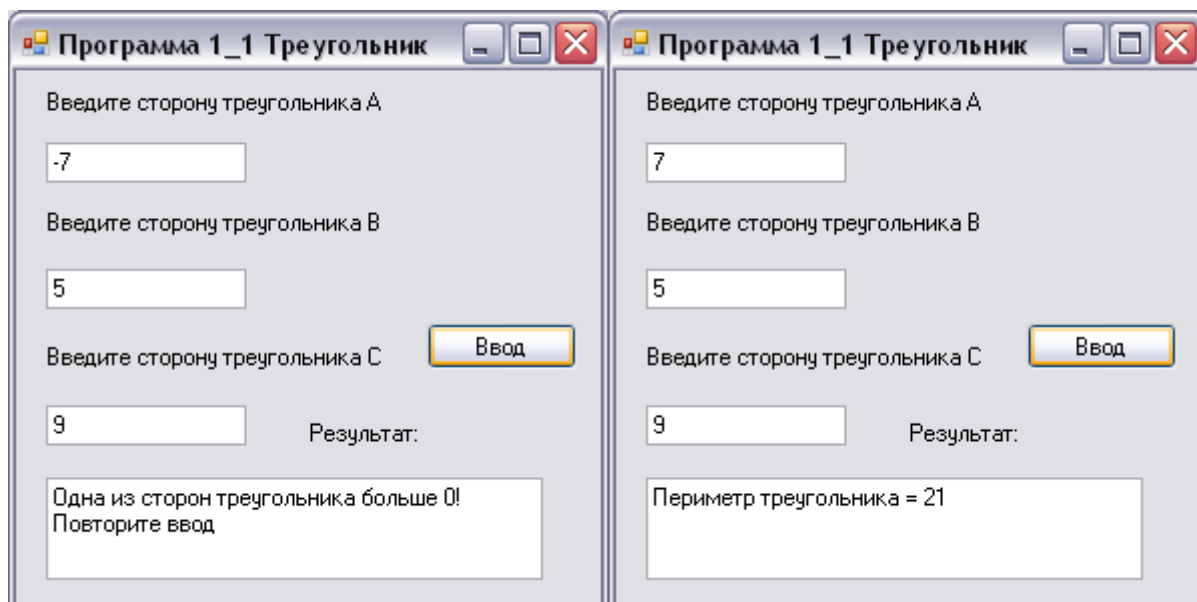


Рисунок 1.3 Окна программы «Треугольник».

Весь процесс визуального программирования заключался в перетаскивании необходимые элементы управления из окна **Toolbox** на форму, и размещении их в определённом порядке. Если это требуется, то с помощью мыши можно отрегулировать размеры и расположение элементов управления, добавленных на форму, а также размеры самой формы.

Обычно окно **Toolbox** находится в «свернутом» состоянии. Чтобы его «развернуть» необходимо левой клавишей мыши кликнуть на панель **Toolbox**, раскрыв ее, и закрепить в определённом месте экрана с помощью элемента  (кликнуть на него). По окончании работы с окном **Toolbox** его можно «свернуть» кликнув на элемента .

В процессе визуального программирования необходимо изменять некоторые свойства элементов управления, например, у меток и кнопки было изменено свойство **Text** в соответствии с рисунком 1.3. Для этого необходимо пользоваться окном Properties, которое изображено на рисунке 1.4.

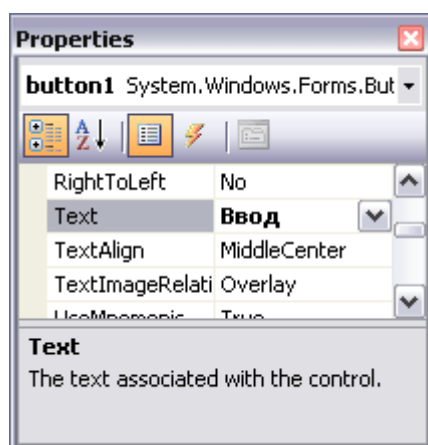


Рисунок 1.4 – Окно Properties для элемента button1

Отличие поля результата от полей ввода в том, что в нем установлено свойство **Multiline = true**.

Во всех элементах управления использовалось поле **Text**.

Для получения «пустого» метода – обработчика сообщения о нажатии на кнопку «Ввод» достаточно на этапе визуального программирования дважды кликнуть на эту кнопку. В пустую заготовку обработчика сообщения

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

включим код ввода в режиме диалога значения сторон треугольника и необходимые проверки на их соответствие треугольнику.

Исходный код файла Program.cs:

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

namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Исходный код файла Form1.cs:

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

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

```

private void button1_Click(object sender, EventArgs e)
{
    int a,b,c,p;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = Convert.ToInt32(textBox3.Text);
    p = a + b + c;
    if (a > 0 && b > 0 && c > 0)
        if (a + b > c && a + c > b && b + c > a)
            textBox4.Text = "Периметр треугольника = " + p.ToString();
        else
        {
            textBox4.Text = "Одна из сторон треугольника больше суммы
двух других Повторите ввод ";
        }
    else
    {
        textBox4.Text = "Одна из сторон треугольника больше 0!
Повторите ввод ";
    }
}
}
}
}

```

Среда автоматически формирует богатый список пространств имен. Рассмотрим некоторые из них.

Пространство имен **System** содержит определение фундаментальных и базовых классов, определяющих типы данных, события, обработчики событий и другие, необходимые в каждом приложении компоненты.

В пространстве имен **System.Collections** определены классы, реализующие функциональность таких контейнеров, как массивы, списки, словари, хэши и т.п.

Классы пространства **System.ComponentModel** используются для реализации необходимого поведения компонентов и элементов управления приложения на этапе его разработки и выполнения.

Класс **System.Data** необходим приложениям, работающим с базами данных посредством интерфейса **ADO.NET**. Этот интерфейс вы будете рассматривать при изучении баз данных.

Пространство имен **System.Drawing** необходимо для доступа к интерфейсу графических устройств (**Graphics Device Interface, GDI**), а точнее, к его расширенной версии **GDI+**. Классы, определенные в этом пространстве имен, необходимы для рисования в окнах приложений текста, линий, двумерных фигур, изображений и других графических объектов.

Пространство **System.Linq** содержит классы, задающие типы, интерфейсы, стандартные операторы запроса.

Пространство имен **System.Windows.Forms** — в нем определены классы, реализующие поведение оконных форм, составляющих базу оконных приложений **Microsoft windows** на платформе **Microsoft .NET Frameworks**.

Реально программе нужны два пространства имен – **System** и **System.Windows.Forms**, все остальные пространства имен формируются «на вырост».

После отладки программы все файлы необходимо записать (в меню выбрать действие **File->Save All**).

Визуальная среда программирования даже для небольшой программы создает более 10 файлов и вложенных папок, пояснение которых уводит читателя в такие дебри работы среды, что ни один из известных авторов книг по программированию на языке **C#** не отважился дать пояснений, какие файлы и зачем создаются в папках разрабатываемого проекта.

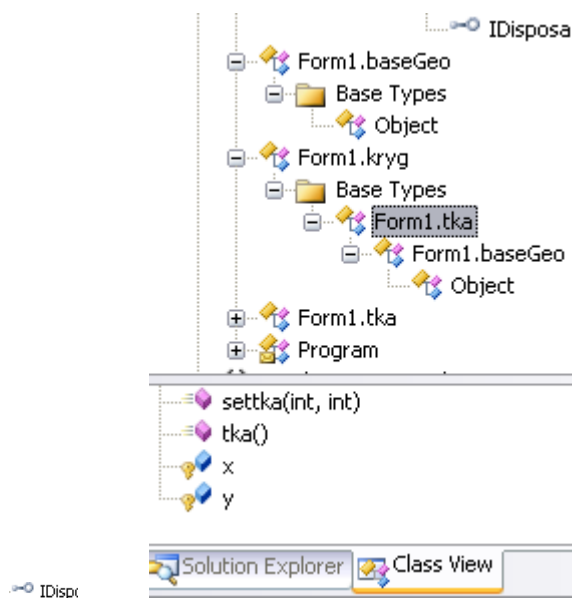
Мы не будем отходить от традиций, отметив только что в папке **1_1_treygolnik**, созданной на рабочем столе для первой программы, находится папка **WindowsFormsApplication1**, в которой еще одна папка с именем **WindowsFormsApplication1** и файл проекта программы, вызываемый для редактирования **WindowsFormsApplication1.csproj**. В очередной папке **WindowsFormsApplication1** (как в матрешке) находятся еще три папки **bin**, **obj**, **Properties** и несколько файлов, в том числе файлы с кодом программы – **Program.cs** и кодом формы – **Form1.cs**. Здесь же находится ресурсный файл формы **Form1**, в котором сохраняется внешний вид формы, и файл **Form1.Designer.cs**, в котором запоминаются значения «свойств» формы и всех элементов управления, размещенных на форме.

Реально для работы со средой разработки **Visual Studio .Net** нам пока нужен файл кода формы **Form1.cs**.

Рекомендую, на первых этапах освоения программирования, не «вламываться» в свойства и названия остальных файлов, а ограничиться работой с текстом только файла **Form1.cs**.

Некоторые условные обозначения, принятые в среде

-  **WindowsFormsApplica** — Проект
-  — Пространство имен
-  — Класс
-  **IDispi** — Интерфейс
-  **settk(int, int)** — Метод
-  **x** — Поле



- |\$1 — Проект
 - {} — Пространство имен
 - III — Класс
 - *o — Интерфейс
 - Q: /;ёф — Метод
 - Иг* — Свойство
 - # — Поле
 - *III — Структура
 - Ц| — Перечислимый тип
 - Q: ,JSJI — Элемент перечислимого типа
 - 9 — Событие
-
- |\$1 — Проект