



---

# Урок №1

## Содержание

1. Что такое Windows Forms?
2. Отличие Windows Forms от других GUI библиотек.
3. Анализ типичного Windows Forms приложения
4. Окна сообщений
5. Форма
  - 5.1. Понятие формы.
  - 5.2. Свойства формы.
  - 5.3. Модальные и немодальные формы
6. Принципы обработки сообщений мыши
7. Использование таймера.
8. Принципы работы со временем и датой.



## 1. Что такое Windows Forms?

Итак, подошел важный момент в вашем изучении платформы **.Net Framework**. Сегодня вы начнете разбор важнейшей части иерархии классов **BCL** под кодовым названием **Windows Forms**.

В дальнейшем тексте может также употребляться общепринятое сокращение **WinForms**. Что же такое **WinForms**?

**Windows Forms** - это набор классов иерархии BCL, отвечающих в платформе **.Net Framework** за создание графического интерфейса пользователя (**Graphical User Interface**).

Очень важно понять, что с помощью **WinForms** вы сможете создавать только клиентские приложения, запускаемые на компьютере конечного пользователя. Это значит, что для построения веб-приложений вам придется использовать другую часть **.Net Framework** под названием **WebForms**. Однако вернемся к тематике нашего урока. **Windows Forms** представлена в **BCL** пространством **System.Windows.Forms**.

Классы данного пространства можно поделить на следующие разделы (данная таблица взята из MSDN):

Категория классов	Подробное описание
Элементы управления, пользовательские элементы управления и формы	Большинство классов в пространстве имен <b>System.Windows.Forms</b> созданы на основе класса <a href="#">Control</a> . Класс <a href="#">Control</a> предоставляет основные функциональные возможности для всех элементов управления, отображаемых в <a href="#">Form</a> . Класс <a href="#">Form</a> представляет окно в приложении. Оно включает диалоговые окна, немодальные окна, а также клиентские и родительские окна интерфейса MDI. На основе класса <a href="#">UserControl</a> можно создавать собственные классы элементов управления.



Категория классов	Подробное описание
Меню и панели инструментов	<p>Windows Forms включает широкий набор классов, которые позволяют создавать пользовательские панели инструментов и меню, отличающиеся современным обликом и поведением. <a href="#">ToolStrip</a>, <a href="#">MenuStrip</a>, <a href="#">ContextMenuStrip</a> и <a href="#">StatusStrip</a> позволяют создавать панели инструментов, меню, контекстные меню и строки состояния, соответственно.</p>
Элементы управления	<p>Пространство имен <b>System.Windows.Forms</b> предоставляет большое количество классов элементов управления, которые позволяют создавать пользовательские интерфейсы с расширенными возможностями. Некоторые элементы управления предназначены для ввода данных в приложении, например <a href="#">TextBox</a> и <a href="#">ComboBox</a>. Другие элементы управления отображают данные приложений, например <a href="#">Label</a> и <a href="#">ListView</a>. Это пространство имен также предоставляет элементы управления для вызова команд в приложении, например <a href="#">Button</a>.</p> <p>Элемент управления <a href="#">WebBrowser</a> и такие классы управляемых HTML-страниц, как <a href="#">HtmlDocument</a>, позволяют отображать HTML-страницы и выполнять с ними определенные действия в области управляемого приложения Windows Forms. Элемент управления <a href="#">MaskedTextBox</a> представляет собой улучшенный элемент управления вводом данных, который позволяет создавать маску для принятия или отклонения введенных пользователем данных в автоматическом режиме.</p> <p>Кроме того, с помощью элемента управления <a href="#">PropertyGrid</a> можно создать собственный конструктор Windows Forms, отображающий видимые конструктором свойства элементов управления.</p>
Макет	<p>Несколько ключевых классов в Windows Forms предназначены для управления расположением элементов управления на экра-</p>



Категория классов	Подробное описание
	не, то есть на форме или в элементе управления. <a href="#">FlowLayoutPanel</a> позволяет разместить все элементы управления один за другим, а <a href="#">TableLayoutPanel</a> позволяет определить строки и ячейки для размещения элементов управления по заданной сетке. <a href="#">SplitContainer</a> позволяет разделить интерфейс на несколько частей с изменяемыми размерами.
Данные и привязка данных	Windows Forms обеспечивает расширенную архитектуру для привязывания к таким источникам данных, как базы данных и XML-файлы. Элемент управления <a href="#">DataGridView</a> предоставляет настраиваемую таблицу для отображения данных и позволяет настраивать формат ячеек, строк, столбцов и границ. Элемент управления <a href="#">BindingNavigator</a> представляет стандартный способ навигации и работы с данными в форме; <a href="#">BindingNavigator</a> часто используется в сочетании с элементом управления <a href="#">BindingSource</a> для перемещения от одной записи к другой в форме, а также для выполнения операций с записями.
Компоненты	Помимо элементов управления пространство имен <b>System.Windows.Forms</b> предоставляет другие классы, которые не являются производными от класса <a href="#">Control</a> , но также обеспечивают визуальные функции для приложений Windows. Такие классы, как <a href="#">ToolTip</a> и <a href="#">ErrorProvider</a> , расширяют возможности или предоставляют сведения пользователям. Классы <a href="#">Help</a> и <a href="#">HelpProvider</a> позволяют отображать текст справки для пользователя, который работает с приложениями.
Общие диалоговые окна	Операционная система Windows предоставляет ряд основных диалоговых окон, позволяющих обеспечить единообразие пользовательского интерфейса в приложениях Windows при выполнении таких операций



Категория классов	Подробное описание
	<p>как открытие и сохранение файлов, задание цвета шрифта или текста и печать. Классы <a href="#">OpenFileDialog</a> и <a href="#">SaveFileDialog</a> предоставляют возможность отображения диалогового окна, в котором пользователь может выполнить поиск файла, а также ввести имя файла, который необходимо открыть или сохранить. Класс <a href="#">FontDialog</a> отображает диалоговое окно для изменения элементов <a href="#">Font</a>, используемого приложением. Классы <a href="#">PageSetupDialog</a>, <a href="#">PrintPreviewDialog</a> и <a href="#">PrintDialog</a> отображают диалоговые окна, позволяющие пользователю управлять параметрами печати документов. Дополнительные сведения о печати с помощью приложений Windows см. в разделе, посвященном пространству имен <a href="#">System.Drawing.Printing</a>. Помимо основных диалоговых окон пространство имен <b>System.Windows.Forms</b> предоставляет класс <a href="#">MessageBox</a> для отображения окна сообщения, в котором могут отображаться и извлекаться данные пользователя.</p>

Это был всего лишь краткий обзор пространства, впереди у вас детальное знакомство с аспектами использования данной технологии.



---

## 2. Отличие Windows Forms от других GUI библиотек.

В рамках курса **Visual C++** вы уже познакомились с принципами создания пользовательских интерфейсов с помощью **WinAPI**. Механизмы использования **Windows Forms** базируются на принципах ООП, в отличие от **WinAPI**, что является немаловажным фактором. Это означает, что вы будете использовать predetermined classes для доступа к той или иной функциональности библиотеки, что, безусловно, удобнее прямого общения с функциями **WinAPI**. Субъективно можно отметить, что количество усилий затрачиваемых на изучение и применение библиотеки **Windows Forms** в разы меньше чем аналогичная деятельность для **WinAPI**.



### 3. Анализ типичного Windows Forms приложения

Рассмотрим два способа создания **Windows Forms** приложения. Первый способ – создание приложения без использования **Visual Studio**, второй вариант с помощью среды разработки **Visual Studio**.

Приступим ☺

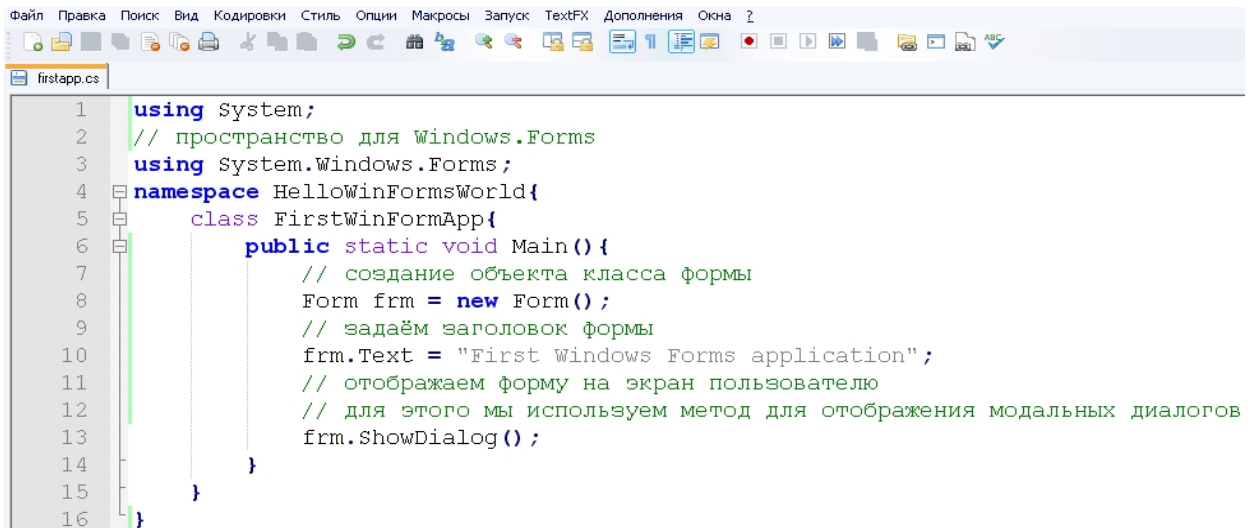
Берем простейший текстовый редактор (например, Блокнот), не вставляющий символы специального форматирования и пишем следующий код:

```
using System;
// пространство для Windows.Forms
using System.Windows.Forms;
namespace HelloWinFormsWorld{
    class FirstWinFormApp{
        public static void Main() {
            // создание объекта класса формы
            Form frm = new Form();
            // задаём заголовок формы
            frm.Text = "First Windows Forms application";
            // отображаем форму на экран пользователю
            // для этого мы используем метод ShowDialog для отображения
            frm.ShowDialog();
        }
    }
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **First Template**.



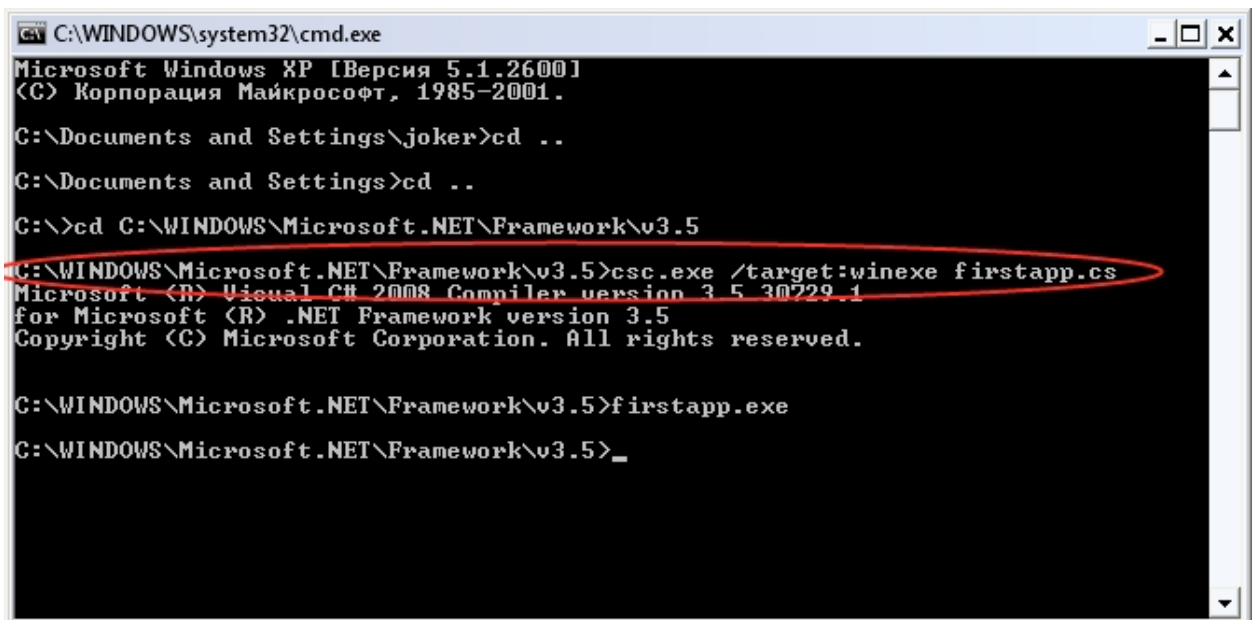
Результат написания кода в текстовом редакторе может выглядеть примерно так:



```
1 using System;
2 // пространство для Windows.Forms
3 using System.Windows.Forms;
4 namespace HelloWinFormsWorld{
5     class FirstWinFormApp{
6     public static void Main(){
7         // создание объекта класса формы
8         Form frm = new Form();
9         // задаём заголовок формы
10        frm.Text = "First Windows Forms application";
11        // отображаем форму на экран пользователю
12        // для этого мы используем метод для отображения модальных диалогов
13        frm.ShowDialog();
14    }
15 }
16 }
```

Не забудьте сохранить файл, указав в качестве расширения .cs.

Следующий этап – нам необходимо скомпилировать программу. Для этого используем уже известный вам компилятор csc.exe.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\Documents and Settings\joker>cd ..
C:\Documents and Settings>cd ..
C:\>cd C:\WINDOWS\Microsoft.NET\Framework\v3.5
C:\WINDOWS\Microsoft.NET\Framework\v3.5>csc.exe /target:winexe firstapp.cs
Microsoft (R) Visual C# 2008 Compiler version 3.5.30729.1
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.

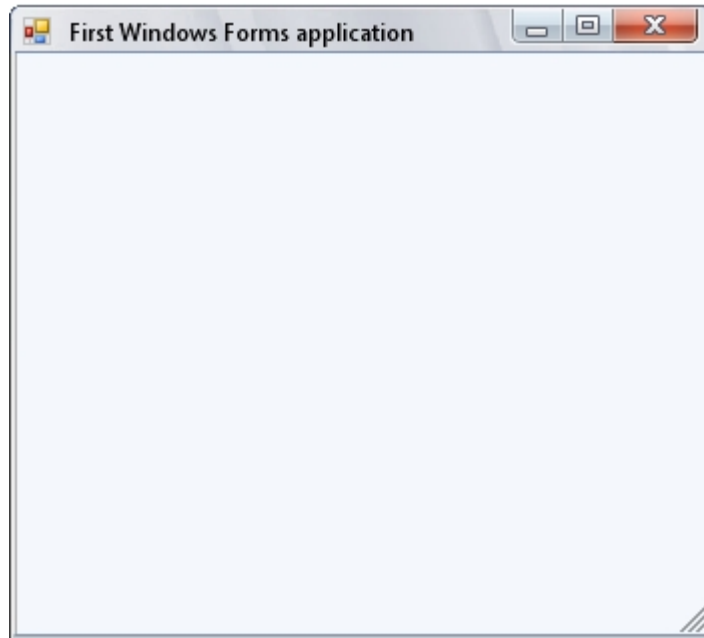
C:\WINDOWS\Microsoft.NET\Framework\v3.5>firstapp.exe
C:\WINDOWS\Microsoft.NET\Framework\v3.5>_
```

Обратите внимание на ключ командной строки /target:winexe. Он указывает на то, что необходимо собрать Windows приложение. После





отработки csc.exe будет создан файл firstapp.exe. Запустим его и увидим:



Несколько дополнительных комментариев по коду программы. В тексте программы мы создаём объект класса **Form**. Класс **Form** отвечает за всевозможные операции с формой. Форма – это аналог диалога, уже известного вам по курсу WinAPI. **Text** – это свойство с помощью, которого мы задаём заголовок форме. Для отображения формы мы используем метод **ShowDialog**. Обратите внимание, что программа остановится на строке с **ShowDialog** и не пойдёт дальше пока окно не будет закрыто. Немного видоизменим код нашей программы, добавив класс, который наследуется от класса **Form**.

```
using System;
// пространство для Windows.Forms
using System.Windows.Forms;
namespace HelloWinFormsWorld
{
    // пользовательский класс
    class MyForm : Form
    {
        // конструктор класса
        public MyForm(string caption)
        {
            // задаём заголовок формы
        }
    }
}
```



```
        Text = caption;
    }
}
class FirstWinFormApp
{
    public static void Main()
    {
        // создание объекта пользовательского класса формы
        MyForm frm = new MyForm("Hello, world!!!");
        // отображаем форму на экран пользователю
        // для этого мы используем метод для отображения модальных
        // диалогов
        frm.ShowDialog();
    }
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **Second Template**.

Скомпилируем этот пример по указанному выше алгоритму, получим опять же форму с измененным заголовком окна. Отличие данного примера от предыдущего состоит в том, что теперь у нас появился пользовательский класс формы в рамках, которого мы можем размещать переменные-члены, методы-члены, обработчики событий и так далее. Обратите внимание на доступ к свойство **Text** без указания какого-либо объекта или ссылки. Такой вид обращения обусловлен тем что, наш класс наследуется от класса формы, то есть получает доступ к устройству родителя, к его доступным полям и методам.

Произведем ещё некоторую модификацию примера для того чтобы получить код схожий с тем шаблоном, который генерируется **Visual Studio**.

```
using System;
// пространство для Windows.Forms
using System.Windows.Forms;
namespace HelloWinFormsWorld{
    class MyForm : Form{
        public MyForm(string caption){
            Text = caption;
        }
    }
    class FirstWinFormApp{
        public static void Main(){
```



```
// создание объекта пользовательского класса формы
// запуск обработки очереди сообщений и отображение формы
Application.Run(new MyForm("Hello, world!!!"));
    }
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **Third Template**.

В этом примере мы также используем пользовательский класс, основное отличие состоит в коде метода **Main**. Разберем единственную строку метода. Класс **Application** предназначен для выполнения операций с приложением, например, таких как старт приложения, остановка приложения, запуск очереди сообщений приложения, получения информации о приложении и так далее.

Мы обращаемся к методу **Run** этого класса. Мы используем следующий вариант перегрузки метода **Run**.

```
public static void Run(Form mainForm)
```

Метод **Run** запускает стандартный цикл обработки сообщений приложения в текущем потоке и делает указанную форму видимой.

Ссылка на форму передаётся в качестве аргумента данному методу.

Обратите внимание, что приложение отображает форму, после чего, продолжает исполнение метода **Main**, не дожидаясь закрытия формы.

Теперь дополним наш пример обработкой события. В качестве примера обрабатываем событие **Click**, возникающее при клике кнопки мышки.

Обратимся в MSDN для получения информации о данном событии.

```
public event EventHandler Click
```



Из объявления события получаем информацию о том, что сигнатура обработчика следующая:

```
public delegate void EventHandler(  
    Object sender,  
    EventArgs e  
)
```

Аргумент **sender** - источник события, **e** - объект **EventArgs**, не содержащий данных события.

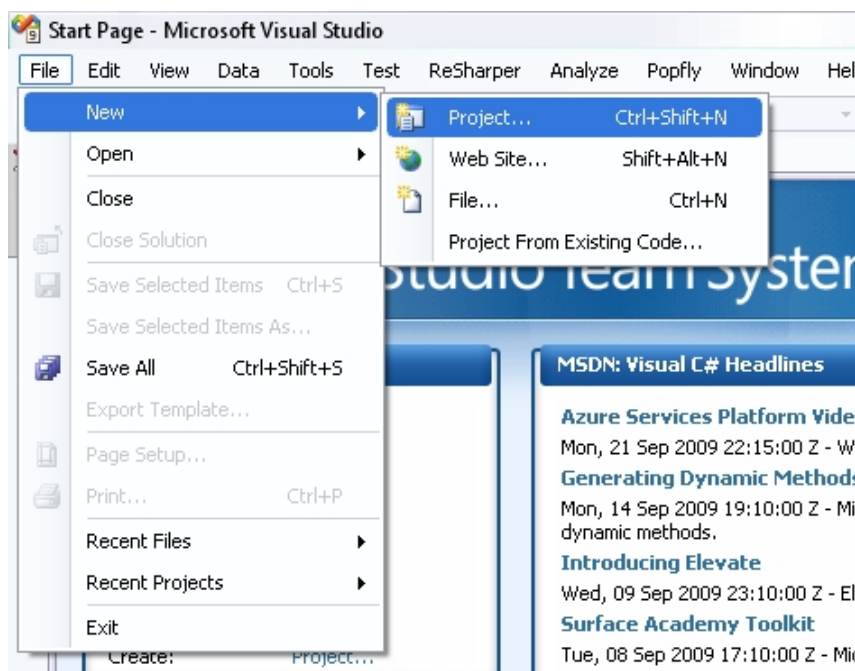
Перейдем непосредственно к коду:

```
using System;  
// пространство для Windows.Forms  
using System.Windows.Forms;  
namespace HelloWinFormsWorld{  
    class MyForm : Form{  
        public MyForm(string caption){  
            // установка заголовка окна  
            Text = caption;  
            // закрепляем обработчик события  
            Click+=new EventHandler(ClickHandler);  
        }  
        public void ClickHandler(Object sender,EventArgs e){  
            MessageBox.Show("Click");  
        }  
    }  
    class FirstWinFormApp{  
        public static void Main(){  
            // создание объекта пользовательского класса формы  
            // запуск обработки очереди сообщений и отображение формы  
            Application.Run(new MyForm("Hello, world!!!"));  
        }  
    }  
}
```

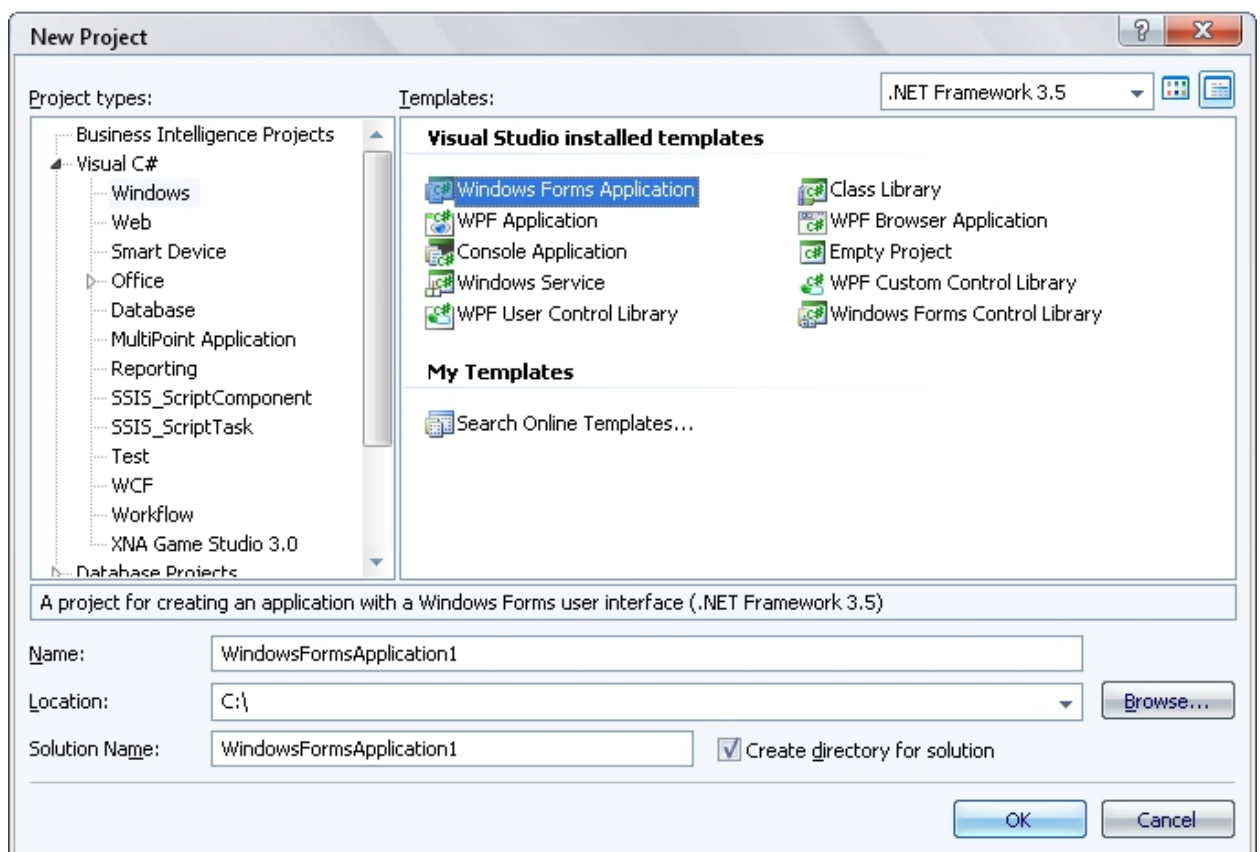
Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **Fourth Template**.

Обработчик события, закрепляется уже известным для вас образом с помощью синтаксиса событий и делегатов.

Теперь настало время создать проект с помощью **Visual Studio**. Запускаем **Visual Studio**. Выбираем пункт меню **File->New->Project**

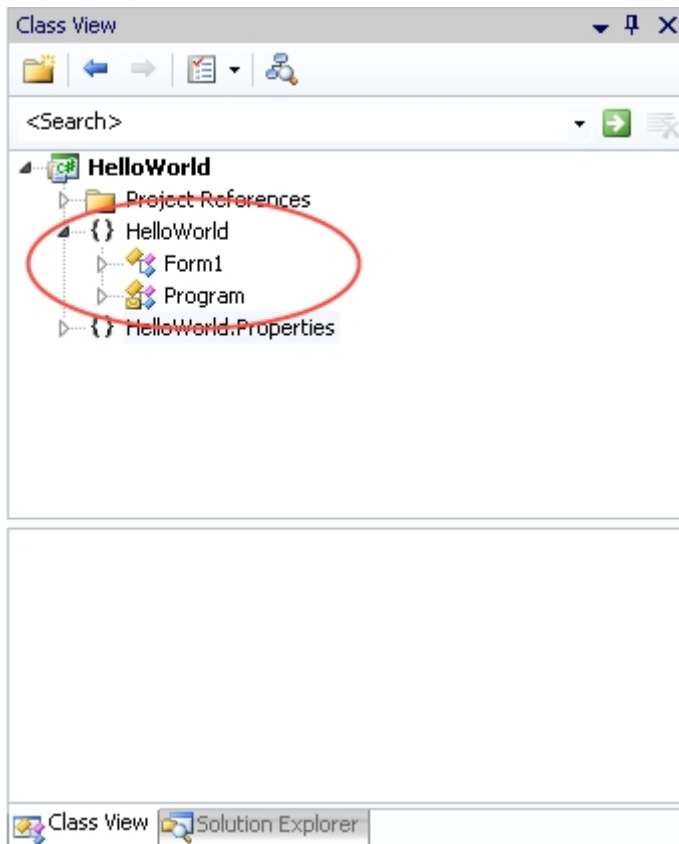


В появившемся окне выбираем **Visual C#->Windows->Windows Forms Application**. Выбираем путь, вводим название проекта и нажимаем на кнопку **OK**. После чего у нас генерируется каркас приложения.





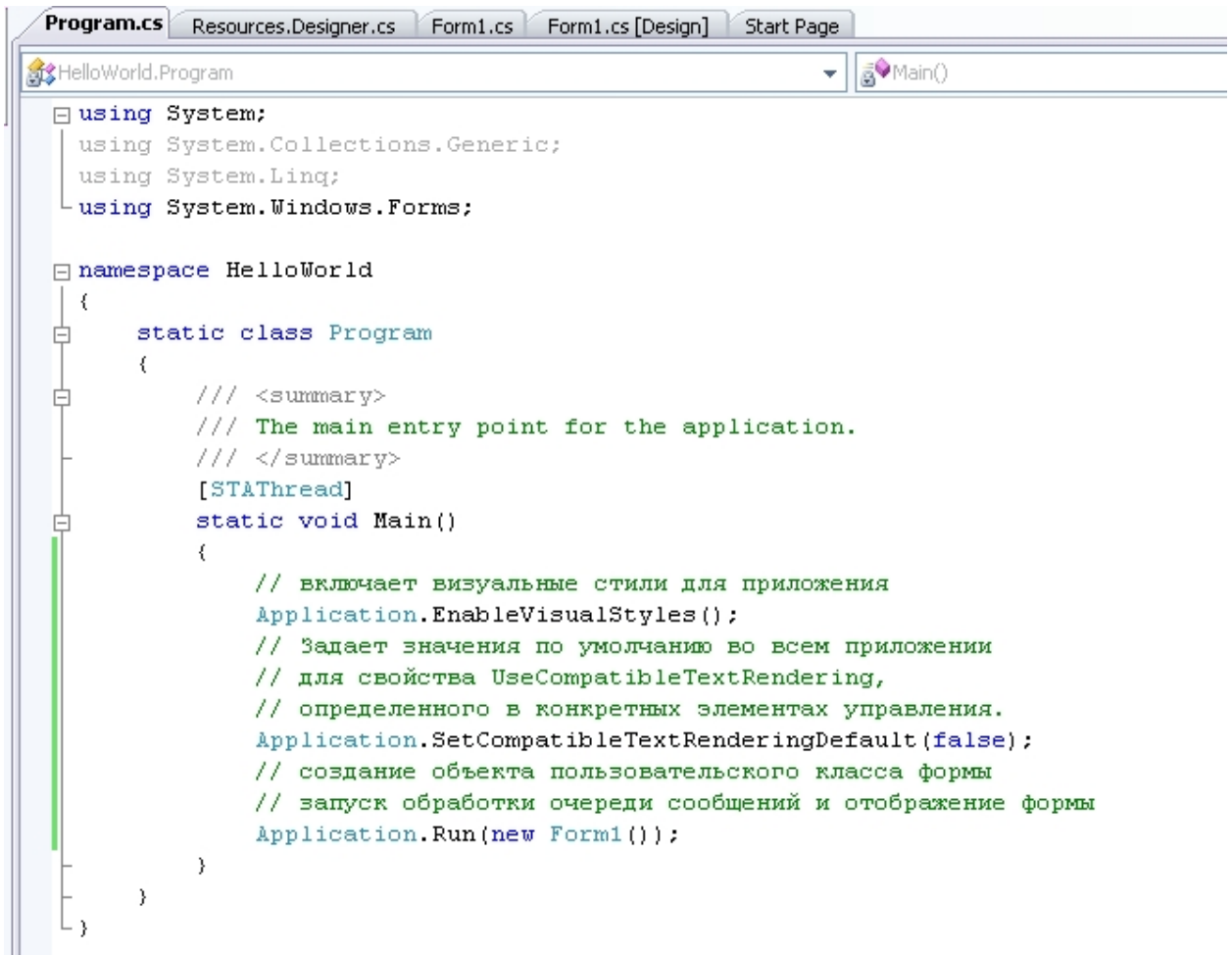
В результате генерации каркаса будет создан проект с набором классов. Нас волнуют два класса **Form1** и **Program**.



Класс **Form1** - пользовательский класс, наследуемый от класса **Form**, отвечает за операции с формой. Класс **Program** содержит метод **Main**.



Рассмотрим код метода **Main**.



Фактически тело Main, как и в нашем примере выше.

Класс **Form1** содержит 3 метода:

1. **Dispose** – о данном методе вы уже знаете из материалов курса C#
2. **Form1** – конструктор класса
3. **InitializeComponent** – специальный метод, используемый так называемым **дизайнером**. **Дизайнер** – это утилита Visual Studio, которая автоматически генерирует код в то время, когда вы настраиваете форму в редакторе. Программисту категорически не рекомендуется писать свой код в данный метод, так как он пересоздается дизайнером при изменении формы. Метод **InitializeComponent** вызывается конструктором класса. Ниже приведен пример тела метода, сразу после генерации приложения.



```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.SuspendLayout();
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(292, 271);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
}

#endregion
```

Как вы видите, в коде задаётся размер клиентской области окна, заголовок окна и так далее. Ещё раз напомним, что данный код генерируется автоматически. Если у вас есть желание как-либо программно инициализировать форму, можно воспользоваться, например конструктором.





## 4. Окна сообщений

Рассмотрим принцип отображения сообщений в **Windows Forms**. Вспомним, в **WinAPI** мы использовали **API** вызов **MessageBox**. Хорошая новость состоит в том, что в **WinForms** мы также будем использовать **MessageBox**, но не функцию, а класс.

Основная цель класса **MessageBox** состоит в том, чтобы отобразить окно сообщения, содержащее текст, кнопки, символы. Класс **MessageBox** находится в пространстве **System.Windows.Forms**. Наследуется от класса **Object**. Это означает, что мы получаем в наследство члены класса **Object**.

```
System.Object  
    System.Windows.Forms.MessageBox
```

Сам **MessageBox** добавляет фактически только один метод **Show**, основная задача, которого отображение информационного окна. У метода **Show** есть много перегруженных версий. Рассмотрим ту версию, которую будем использовать в примерах.

```
public static DialogResult Show(  
    string text,  
    string caption,  
    MessageBoxButtons buttons,  
    MessageBoxIcon icon  
)
```

Данный вариант **Show** отображает окно сообщения с заданным текстом, заголовком, кнопками и иконкой. Параметры:

1. **text** - текст, отображаемый в окне сообщения.
2. **caption** – текст для отображения в строке заголовка окна сообщения.



3. **buttons** - одно из значений **MessageBoxButtons**, указывающее, какие кнопки отображаются в окне сообщения.

4. **icon** - одно из значений **MessageBoxIcon**, указывающее, какой значок отображается в окне сообщения.

**MessageBoxButtons** – перечисление, содержащее все виды кнопок, которые возможно отобразить в **MessageBox**.

```
public enum MessageBoxButtons
```

Возможные варианты: **OK**, **OKCancel**, **AbortRetryIgnore**, **YesNoCancel**, **YesNo**, **RetryCancel**. Например, так:

**MessageBoxButtons.AbortRetryIgnore** (показать три кнопки "Прервать", "Повторить" и "Пропустить".)

**MessageBoxIcon** - перечисление, содержащее все виды иконок, которые возможно отобразить в **MessageBox**.

```
public enum MessageBoxIcon
```

Возможные варианты: **None** (данное окно сообщения не содержит иконок), **Hand** (данное окно сообщения содержит иконку, состоящую из белого значка X, заключенного в красный кружок), **Exclamation** (данное окно сообщения содержит символ, состоящий из восклицательного знака в желтом треугольнике) и так далее. Более подробную информацию смотрите в **MSDN**. Пример использования:

**MessageBoxIcon.Exclamation**

Метод **Show** возвращает код нажатой пользователем кнопки. Код записывается в переменную типа **DialogResult**. Данный тип также является перечислением.

```
public enum DialogResult
```



Примеры возможных значений **DialogResult**: **OK**, **Cancel**, **Abort**, **Retry**, **Ignore**, **Yes**, **No**.

То есть, например, если вернулось значение **DialogResult.OK** это означает что была нажата кнопка **OK**.

Рассмотрим пример вызова данного метода. Отобразим окно с информационным сообщением, тремя кнопками **Abort, Retry, Ignore**, а также иконкой **Error**.

```
DialogResult result = MessageBox.Show("Произошла ошибка при доступе к  
диску!", "Ошибка", MessageBoxButtons.AbortRetryIgnore, MessageBoxIcon.Error);  
if (result == DialogResult.Abort)  
{  
    MessageBox.Show("Вы нажали кнопку Прервать");  
}  
else if (result == DialogResult.Retry)  
{  
    MessageBox.Show("Вы нажали кнопку Повтор");  
}  
else if (result == DialogResult.Ignore)  
{  
    MessageBox.Show("Вы нажали кнопку Пропустить");  
}
```

Рассмотрим ещё пример приложения, отображающего различные варианты вызова метода **Show**.

```
static class Program  
{  
    static DialogResult ShowMessageBoxes()  
    {  
        //показ окна, отображающего текстовое сообщение и кнопку OK  
        String message = "Окно, отображающее текстовое сообщение";  
        MessageBox.Show(message);  
  
        //показ окна с текстом и двумя кнопками OK и CANCEL  
        message = "Окно с текстом и двумя кнопками OK и CANCEL";  
        String caption = "Окно с двумя кнопками";  
        DialogResult result = MessageBox.Show(message, caption,  
        MessageBoxButtons.OKCancel);  
        String button = result.ToString();  
  
        //показ Окна с тремя кнопками и какой-то иконкой  
        result = MessageBox.Show("Вы нажали кнопку " + button + ".  
Повторить?", button, MessageBoxButtons.AbortRetryIgnore,
```



```
        MessageBoxIcon.Asterisk);  
        return result;  
    }  
  
    [STAThread]  
    static void Main()  
    {  
        DialogResult result;  
        do  
        {  
            result = ShowMessageBoxes();  
        } while (result == DialogResult.Retry);  
    }  
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **FirstSample**.

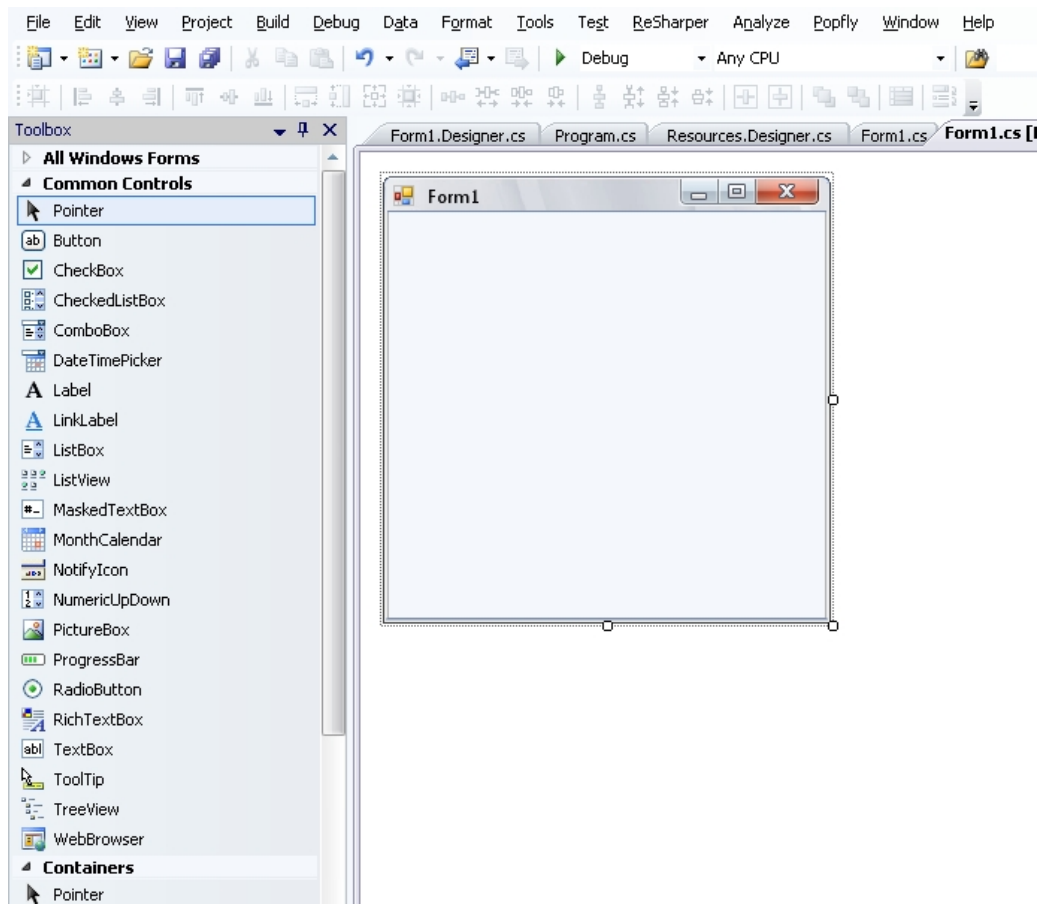


## 5. Форма

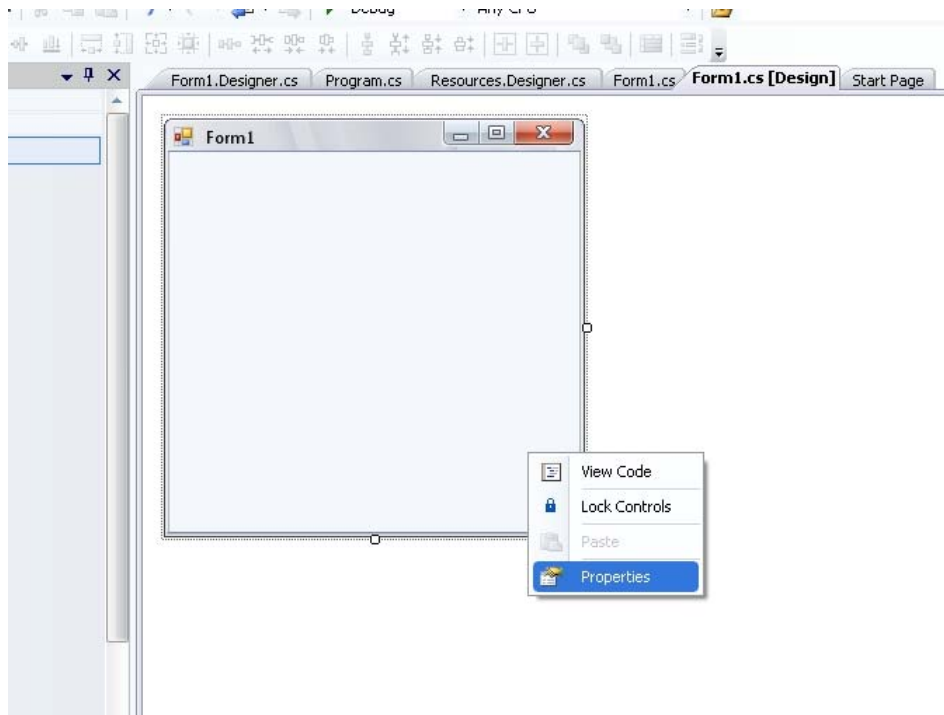
Как мы уже говорили форма – это аналог диалога, то есть фактически диалоговое окно, на котором вы можете размещать элементы управления. С его помощью происходит взаимодействие с пользователем. Класс **Form** в **.Net Framework** отвечает за операции с формой. Ниже приведена иерархия наследования для него:

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ScrollableControl
          System.Windows.Forms.ContainerControl
            System.Windows.Forms.Form
```

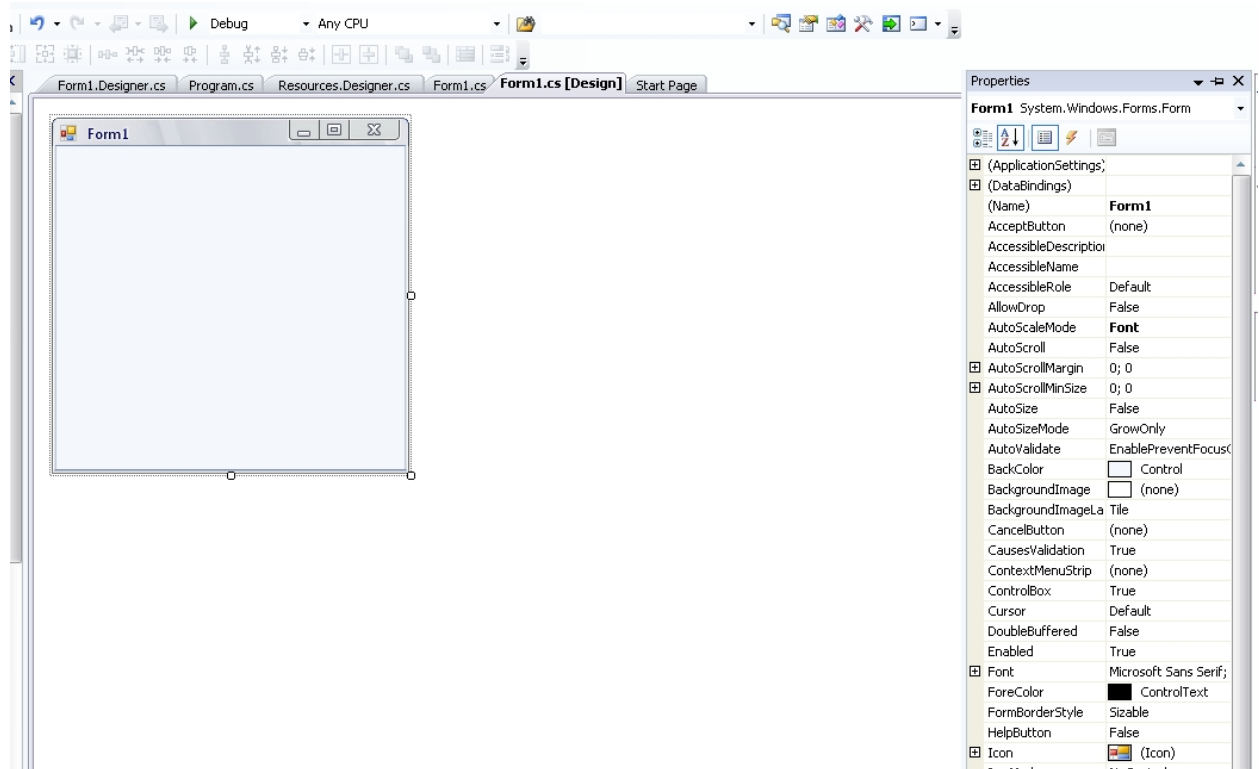
Как видно из указанного выше дерева класс **Form** находится в пространстве **System.Windows.Forms** и наследуется от большого количества классов. В режиме дизайна форма отображается так:



Для того чтобы настроить свойства формы необходимо нажать правую кнопку мышки и выбрать пункт **Properties**.



После активизации этого пункта отобразится окно свойство, в котором можно настроить свойства формы.



Не забывайте о том, что настройка свойств в этом окне ведет к генерации кода внутри метода **InitializeComponent**. А теперь давайте рассмотрим некоторые полезные свойства и методы класса **Form**. Важно отметить, что многие важные поля и методы наследуются от базового класса **Control**.

Свойство **Text** возвращает или задает заголовок окна. Наследуется из класса **Control**.

```
public virtual string Text { get; set; }
```

Свойство **Size** возвращает или задает размер формы.

```
public Size Size { get; set; }
```

Тип **Size** находится в пространстве **System.Drawing**. С помощью **Size** можно задать ширину и высоту формы.



Свойство **StartPosition** возвращает или задает начальное положение формы в режиме выполнения.

```
public FormStartPosition StartPosition { get; set; }
```

**FormStartPosition** – перечисление, задающее позицию. Возможные варианты значений:

1. **Manual** - Положение формы определяется свойством **Location** класса **Form**.
2. **CenterScreen** - Форма с заданными размерами располагается в центре текущего отображения.
3. **WindowsDefaultLocation** - Форма с заданными размерами размещается в расположении, определенном по умолчанию в Windows.
4. **WindowsDefaultBounds** - Положение формы и ее границы определены в Windows по умолчанию.
5. **CenterParent** - Форма располагается в центре родительской формы.

Свойство **Location** возвращает или задает объект **Point**, который представляет собой верхний левый угол формы в экранных координатах.

```
public Point Location { get; set; }
```

**Point** – структура, находящаяся в пространстве **System.Drawing**, представляет упорядоченную пару целых чисел — координат **X** и **Y**, определяющую точку на двумерной плоскости.

Свойство **BackColor** возвращает или задает цвет фона для формы. Наследуется от класса **Control**.





```
public virtual Color BackColor { get; set; }
```

**Color** – структура из пространства **System.Drawing**, используется для представления цвета в терминах каналов альфа, красного, зеленого и синего (ARGB).

Свойство **ForeColor** возвращает или задает цвет надписей на форме. Наследуется из класса **Control**.

```
public virtual Color ForeColor { get; set; }
```

Давайте вспомним о понятиях модальности и немодальности. Модальное окно приложения не даёт переключиться на другие окна этого приложения до тех пор, пока оно не будет закрыто. Немодальное окно в отличие от модального позволяет это сделать.

Для отображения модальной формы используется метод **ShowDialog**.

```
public DialogResult ShowDialog()
```

Возвращает тип **DialogResult** уже известный вам.

Для отображения немодальной формы используется метод **Show** унаследованный от класса **Control**

```
public void Show()
```

Информацию по остальным свойствам, событиям, методам класса **Form** можно посмотреть в **MSDN**.



## 6. Принципы обработки мышинных сообщений

Настало время познакомиться с принципами обработки событий в **WinForms**. Начнем с уже известных вам по курсу **WinAPI** мышинных сообщений. Сразу вспоминаются такие события как **MouseMove** (движение мышкой), **MouseClick** (клик мышкой), **MouseDown** (нажата кнопка мышки), **MouseUp** (отпущена кнопка мышки), и так далее. Важно отметить, что все мышинные события определены в классе **Control**. Класс **Control** является базовым классом для элементов управления и для некоторых оконных классов. Данный класс предназначен для аккумуляции общих механизмов для окон и элементов управления. Рассмотрим в качестве примера мышино событие **MouseMove**.

```
public event MouseEventHandler MouseMove
```

**MouseMove** происходит при перемещении указателя мыши по элементу управления (окну). Тип обработчика события определяется делегатом **MouseEventHandler**.

```
public delegate void MouseEventHandler(  
    Object sender,  
    MouseEventArgs e)
```

Параметры:

1. **sender** – источник события
2. **e** – ссылка на объект класса **MouseEventArgs**, содержащий информацию о событии. Например, в полях **X** и **Y** данного класса будут содержаться координаты по осям **x** и **y** в момент генерации события, в поле **Button** будет содержаться информация о нажатых кнопках мыши в момент возникновения события.

**Использование библиотеки Windows Forms. Урок 1**



Принцип обработки остальных мышиных событий схож с **MouseMove**. Для практического закрепления материала рассмотрим пример приложения, отображающего информацию о текущем положении мышки, а также реагирующего на мышиные клики.

```
private String CoordinatesToString(MouseEventArgs e)
{
    return "Координаты мыши: x=" + e.X.ToString() + "; y=" +
e.Y.ToString();
}
// обработчик MouseMove
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    //отображение текущих координат мыши в заголовке окна
    Text = CoordinatesToString(e);
}

// обработчик события MouseClick
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    //определим какую кнопку мыши нажал пользователь
    String message = "";
    if (e.Button == MouseButton.Right)
    {
        message = "Вы нажали правую кнопку мыши.";
    }
    if (e.Button == MouseButton.Left)
    {
        message = "Вы нажали левую кнопку мыши.";
    }
    message += "\n" + CoordinatesToString(e);

    //выведем сообщение в диалоговое окно
    String caption = "Клик мыши";
    MessageBox.Show(message, caption, MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **SecondSample**.



## 7. Использование таймера

Понятие таймера вам уже известно из курса **WinAPI**. В **.Net Framework** существует несколько видов таймера. Сегодня мы разберем таймер, находящийся в пространстве **System.Windows.Forms**, предназначенный для использования в **WinForms** приложениях. Рассмотрим иерархию наследования

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Timer
```

Из диаграммы можно сделать вывод, что класс не является потомком класса **Control** и это вполне естественно так как таймер не является окном или элементом управления.

Для запуска таймера необходимо:

1. Создать объект класса **Timer**. Конструктор класса **Timer** не получает параметров.
2. Установить временной интервал для срабатывания таймера. Для этого используется свойство **Interval** класса **Timer**. Значение интервала указывается в миллисекундах.
3. Закрепить обработчик на событие **Tick** класса **Timer**.
4. Запустить таймер с помощью вызова метода **Start** класса **Timer** или свойства **Enabled** того же класса
5. Для установки таймера необходимо использовать метод **Stop** класса **Timer**.

Рассмотрим практический пример реализации таймера. В данном проекте пользователь имеет возможность установить таймер на указанное количество секунд, стартовать таймер, остановить таймер.

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



```
{
    Timer vTimer = new Timer();

    public Form1()
    {
        InitializeComponent();
        button2.Enabled = false;

        //определяем обработчик события для таймера
        vTimer.Tick += new EventHandler>ShowTimer;
    }

    private void ShowTimer(object vObject, EventArgs e)
    {
        //останавливаем таймер
        vTimer.Stop();

        button2.Enabled = false;

        MessageBox.Show("Таймер отработал!", "Таймер");
    }

    private void button1_Click(object sender, EventArgs e)
    {
        //проверяем введенное количество секунд для таймера
        if (numSeconds.Value <= 0)
        {
            MessageBox.Show("Количество секунд должно быть больше 0!");
            return;
        }
        //разрешаем прервать таймер
        button2.Enabled = true;

        //интервал задается в миллисекундах, поэтому секунды умножаем на
1000
        //и задаем интервал таймера
        vTimer.Interval = Decimal.ToInt32(numSeconds.Value) * 1000;

        //запуск таймера
        vTimer.Start();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        //останавливаем таймер
        vTimer.Stop();

        MessageBox.Show("Таймер не успел отработать!", "Таймер");

        button2.Enabled = false;
    }
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **ThirdSample**.



## 8. Принципы работы со временем и датой

Для работы со временем и датой в **.Net Framework** используются структуры **DateTime** и **TimeSpan**. **DateTime** используется для представления даты и времени, а **TimeSpan** для представления временного промежутка.

Рассмотрим некоторые поля и методы структуры **DateTime**.

Конструктор класса существует в большом количестве перегрузок. Рассмотрим несколько из них.

```
public DateTime(  
    int year,  
    int month,  
    int day  
)  
  
public DateTime(  
    int year,  
    int month,  
    int day,  
    int hour,  
    int minute,  
    int second,  
    int millisecond  
)
```

Рассмотрим параметры: **year** - год (от 1 до 9999), **month** - месяц (от 1 до 12), **day** - день (от 1 до количества дней в month), **hour** - часы (от 0 до 23), **minute** - минуты (от 0 до 59), **second** - секунды (от 0 до 59), **millisecond** - миллисекунды (от 0 до 999).

Для получения текущей даты и времени данного компьютера, выраженного как местное время, используется статическое свойство **Now**.

```
public static DateTime Now { get; }
```



Для получения только текущей даты используется свойство **Today**.  
Временная составляющая будет равняться 00:00:00.

```
public static DateTime Today { get; }
```

Для преобразования **DateTime** в строку используется группа методов **ТипПреобразованияString** (**ToLongDateString**, **ToLongTimeString**, **ToShortDateString**, **ToShortTimeString**), а также уже известный вам метод **ToString**.

Рассмотрим в качестве примера метод **ToLongTimeString**. Он преобразует значение текущего объекта **DateTime** в эквивалентное ему длинное строковое представление времени.

```
public string ToLongTimeString()
```

По отношению к объекту типа **DateTime** можно применять перегруженные операторы, такие как **+, -, ==, <, >, <=, >=, !=**

Рассмотрим перегрузку **+** и **-**

```
public static DateTime operator +(
    DateTime d,
    TimeSpan t
)

public static TimeSpan operator -(
    DateTime d1,
    DateTime d2
)

public static DateTime operator -(
    DateTime d,
    TimeSpan t
)
```

В перегруженный оператор **+** передаётся два аргумента: **d** - объект, содержащий время и дату, **t** - объект, содержащий временной про-



межуток (например 3 часа 12 минут). Из оператора возвращается объект **DateTime** содержащий результат сложения.

В перегруженный оператор - передаётся два аргумента: **d1** - объект, содержащий время и дату, **d2** – второй объект, содержащий время и дату. Из оператора возвращается объект **TimeSpan**, содержащий временной промежуток - результат вычитания двух дат. Второй вариант перегрузки получает два аргумента: **d** - объект, содержащий время и дату, **t** – объект, содержащий временной промежуток (например 3 часа 12 минут).

Из оператора возвращается объект **DateTime** содержащий результат вычитания.

Перейдем к рассмотрению структуры **TimeSpan**. У **TimeSpan** существует несколько перегруженных конструкторов. Рассмотрим несколько из них.

```
public TimeSpan(  
    int hours,  
    int minutes,  
    int seconds  
)  
public TimeSpan(  
    int days,  
    int hours,  
    int minutes,  
    int seconds  
)  
public TimeSpan(  
    int days,  
    int hours,  
    int minutes,  
    int seconds,  
    int milliseconds  
)
```

Ничего особенно сложного в данных перегруженных конструкторах нет. Передаются часы, минуты, секунды и миллисекунды, то есть самые обычные характеристики для задания временного промежутка.





Свойства **Days**, **Hours**, **Minutes**, **Seconds**, **Milliseconds** используются для получения соответственно дневной, часовой, минутной, секундной и миллисекундой составляющей **TimeSpan**.

Для **TimeSpan** существует набор перегруженных операторов, как и для **DateTime**. Возьмем например оператор **+**.

```
public static TimeSpan operator +(
    TimeSpan t1,
    TimeSpan t2
)
```

В данный оператор передаются два аргумента, представляющих временные промежутки. Из оператора возвращается результат сложения двух промежутков.

Более подробную информацию о **DateTime** и **TimeSpan** смотрите в **MSDN**.

Рассмотрим практический пример, показывающий принципы использования механизмов **DateTime** и таймера. Цель примера - отображать текущее время.

```
public partial class Form1 : Form
{
    // создаём таймер
    private static Timer vTimer = new Timer();
    // Обработчик тика для таймера
    private void ShowTime(object vObj, EventArgs e)
    {
        // преобразование к строке
        labelTime.Text = DateTime.Now.ToLongTimeString();
    }

    public Form1()
    {
        InitializeComponent();
        // преобразование к строке
        labelTime.Text = DateTime.Now.ToLongTimeString();
        // закрепление обработчика
        vTimer.Tick += new EventHandler(ShowTime);
        // установка интервала времени
        vTimer.Interval = 500;
        // стартуем таймер
        vTimer.Start();
    }
}
```



```
}  
}
```

Напомним вам, что проекты, используемые в данном уроке, как обычно располагаются в папке **Sources**. Данный проект называется **FourthSample**.



## Домашнее задание

1. Вывести на экран свое (краткое !!!) резюме с помощью последовательности MessageBox'ов (числом не менее трех). Причем на заголовке последнего должно отобразиться среднее число символов на странице (общее количество символов в резюме / количество MessageBox'ов).
2. Написать функцию, которая "угадывает" задуманное пользователем число от 1 до 2000. Для запроса к пользователю использовать MessageBox. После того, как число отгадано, необходимо вывести количество запросов, потребовавшихся для этого, и предоставить пользователю возможность сыграть еще раз, не выходя из программы. (MessageBox'ы оформляются кнопками и значками соответственно ситуации).
3. Представьте, что у вас на форме есть прямоугольник, границы которого на 10 пикселей отстоят от границ рабочей области формы. Необходимо создать следующие обработчики:
  - a. Обработчик нажатия левой кнопки мыши, который выводит сообщение о том, где находится текущая точка: внутри прямоугольника, снаружи, на границе прямоугольника. Если при нажатии левой кнопки мыши была нажата кнопка Control (Ctrl), то приложение должно закрываться.
  - b. Обработчик нажатия правой кнопки мыши, который выводит в заголовок окна информацию о размере клиентской (рабочей) области окна в виде: Ширина =  $x$ , Высота =  $y$ , где  $x$  и  $y$  - соответствующие параметры высшего окна.
  - c. Обработчик перемещения указателя мыши в пределах рабочей области, который должен выводить в заголовок окна текущие координаты мыши  $x$  и  $y$ .