Практическое занятие № 26

Изучение интегрированной среды разработчика.

Цель: познакомиться со средой Visual Studio.NET и получить навыки работы в ней.

Норма времени: 2 часа.

Оборудование: Компьютер, среда программирования Visual Studio.

Порядок выполнения работы

Задания:

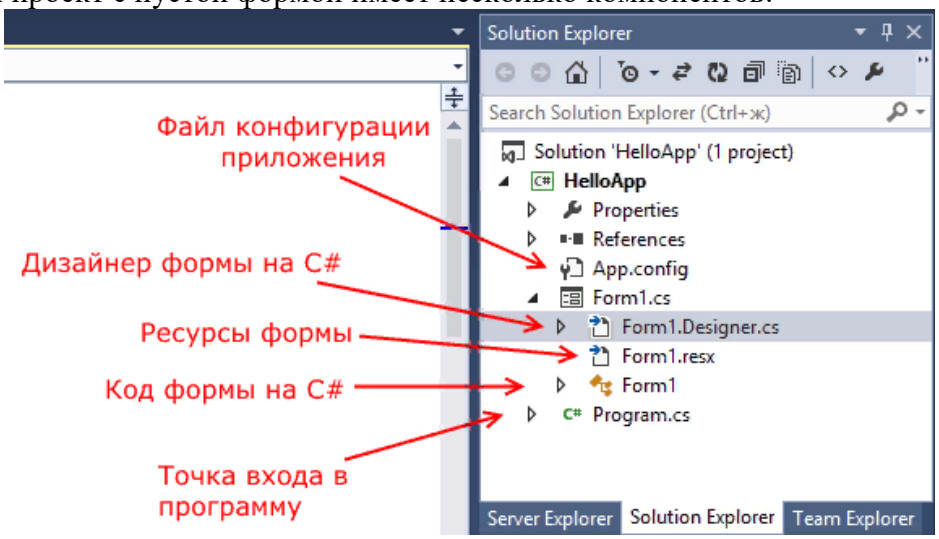
1. Познакомиться с интерфейсом среды Visual Studio.NET.
2. Выполнить задания из методических рекомендаций, расположенные далее.
3. Выполните действия, описанные в каждом разделе.
4. Познакомиться со структурой проекта.
5. Познакомиться со свойствами проекта.

Основы форм

Внешний вид приложения является нам преимущественно через формы. Формы являются основными строительными блоками. Они предоставляют контейнер для различных элементов управления. А механизм событий позволяет элементам формы отзываться на ввод пользователя, и, таким образом, взаимодействовать с пользователем.

При открытии проекта в Visual Studio в графическом редакторе мы можем увидеть визуальную часть формы - ту часть, которую мы видим после запуска приложения и куда мы переносим элементы с панели управления. Но на самом деле форма скрывает мощный функционал, состоящий из методов, свойств, событий и прочее. Рассмотрим основные свойства форм.

Если мы запустим приложение, то нам отобразится одна пустая форма. Однако даже такой простой проект с пустой формой имеет несколько компонентов:



Несмотря на то, что мы видим только форму, но стартовой точкой входа в графическое приложение является класс Program, расположенный в файле Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    static class Program
```

```
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Сначала программой запускается данный класс, затем с помощью выражения Application.Run(new Form1()) он запускает форму Form1. Если вдруг мы захотим изменить стартовую форму в приложении на какую-нибудь другую, то нам надо изменить в этом выражении Form1 на соответствующий класс формы.

Сама форма сложна по содержанию. Она делится на ряд компонентов. Так, в структуре проекта есть файл Form1.Designer.cs, который выглядит примерно так:

```
namespace HelloApp
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #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(284, 261);
            this.Name = "Form1";
            this.Text = "Привет мир!";
            this.ResumeLayout(false);
        }
        #endregion
    }
}
```

Здесь объявляется частичный класс формы Form1, которая имеет два метода: Dispose(), который выполняет роль деструктора объекта, и InitializeComponent(), который устанавливает начальные значения свойств формы.

При добавлении элементов управления, например, кнопок, их описание также добавляется в этот файл.

Но на практике мы редко будем сталкиваться с этим классом, так как они выполняет в основном дизайнерские функции - установка свойств объектов, установка переменных.

Еще один файл - Form1.resx - хранит ресурсы формы. Как правило, ресурсы используются для создания однообразных форм сразу для нескольких языковых культур.

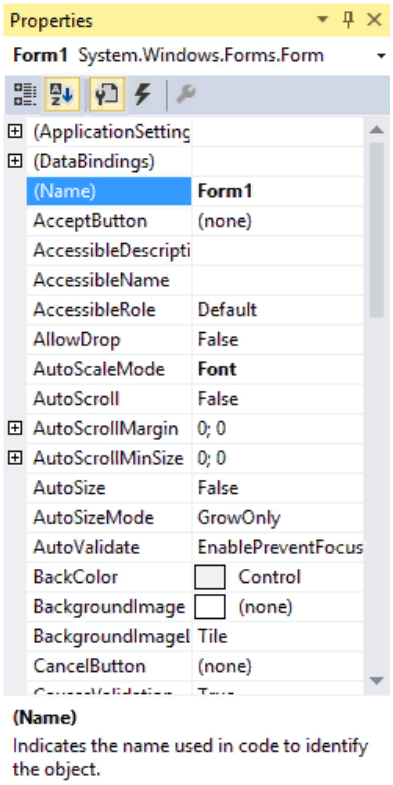
И более важный файл - Form1.cs, который в структуре проекта называется просто Form1, содержит код или программную логику формы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

По умолчанию здесь есть только конструктор формы, в котором просто вызывается метод InitializeComponent(), объявленный в файле дизайнера Form1.Designer.cs. Именно с этим файлом мы и будем больше работать.

Основные свойства форм

С помощью специального окна Properties (Свойства) справа Visual Studio предоставляет нам удобный интерфейс для управления свойствами элемента:



Большинство этих свойств оказывает влияние на визуальное отображение формы. Пробежимся по основным свойствам:

- Name: устанавливает имя формы - точнее имя класса, который наследуется от класса Form
- BackColor: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры
- BackgroundImage: указывает на фоновое изображение формы
- BackgroundImageLayout: определяет, как изображение, заданное в свойстве BackgroundImage, будет располагаться на форме.
- ControlBox: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение false, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма
- Cursor: определяет тип курсора, который используется на форме
- Enabled: если данное свойство имеет значение false, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.
- Font: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его
- ForeColor: цвет шрифта на форме
- FormBorderStyle: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в None можно создавать внешний вид приложения произвольной формы
- HelpButton: указывает, отображается ли кнопка справки формы
- Icon: задает иконку формы
- Location: определяет положение по отношению к верхнему левому углу экрана, если для свойства StartPosition установлено значение Manual
- MaximizeBox: указывает, будет ли доступна кнопка максимизации окна в заголовке формы
- MinimizeBox: указывает, будет ли доступна кнопка минимизации окна
- MaximumSize: задает максимальный размер формы
- MinimumSize: задает минимальный размер формы
- Opacity: задает прозрачность формы
- Size: определяет начальный размер формы
- StartPosition: указывает на начальную позицию, с которой форма появляется на экране
- Text: определяет заголовок формы
- TopMost: если данное свойство имеет значение true, то форма всегда будет находиться поверх других окон
- Visible: видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение false
- WindowState: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном

Программная настройка свойств

С помощью значений свойств в окне Свойства мы можем изменить по своему усмотрению внешний вид формы, но все, то, же самое мы можем сделать динамически в коде. Перейдем к коду, для этого нажмем правой кнопкой мыши на форме и выберем в

появившемся контекстном меню View Code (Просмотр кода). Перед нами открывается файл кода Form1.cs. Изменим При использовании конструктора формы надо учитывать, что весь остальной код должен идти после вызова метода InitializeComponent(), поэтому все установки свойств здесь расположены после этого метода.

Начальное расположение формы

Начальное расположение формы устанавливается с помощью свойства StartPosition, которое может принимать одно из следующих значений:

- Manual: Положение формы определяется свойством Location
- CenterScreen: Положение формы в центре экрана
- WindowsDefaultLocation: Позиция формы на экране задается системой Windows, а размер определяется свойством Size
- WindowsDefaultBounds: Начальная позиция и размер формы на экране задается системой Windows
- CenterParent: Положение формы устанавливается в центре родительского окна

Все эти значения содержатся в перечислении FormStartPosition, поэтому, чтобы, например, установить форму в центре экрана, нам надо прописать так:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

Фон и цвета формы

Чтобы установить цвет, как фона формы, так и шрифта, нам надо использовать цветовое значение, хранящееся в структуре Color:

```
this.BackColor = Color.Aquamarine;
this.ForeColor = Color.Red;
```

Кроме того, мы можем в качестве фона задать изображение в свойстве BackgroundImage, выбрав его в окне свойств или в коде, указав путь к изображению:

```
this.BackgroundImage = Image.FromFile("C:\\Users\\Eugene\\Pictures\\3332.jpg");
```

Чтобы должным образом настроить нужное нам отображение фоновой картинки, надо использовать свойство **BackgroundImageLayout**, которое может принимать одно из следующих значений:

- None: Изображение помещается в верхнем левом углу формы и сохраняет свои первоначальные значения
- Tile: Изображение располагается на форме в виде мозаики
- Center: Изображение располагается по центру формы
- Stretch: Изображение растягивается до размеров формы без сохранения пропорций
- Zoom: Изображение растягивается до размеров формы с сохранением пропорций

Например, расположим форму по центру экрана:

```
this.StartPosition = FormStartPosition.CenterScreen;
```

Добавление форм. Взаимодействие между формами

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Solution Explorer (Обозреватель решений) правой кнопкой мыши и выберем **Add(Добавить)->Windows Form...**

Дадим новой форме какое-нибудь имя, например, Form2.cs:

Итак, у нас в проект была добавлена вторая форма. Теперь попробуем осуществить взаимодействие между двумя формами. Допустим, первая форма по нажатию на кнопку

будет вызывать вторую форму. Во-первых, добавим на первую форму Form1 кнопку и двойным щелчком по кнопке перейдем в файл кода. Итак, мы попадем в обработчик события нажатия кнопки, который создается по умолчанию после двойного щелчка по кнопке:

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

Теперь добавим в него код вызова второй формы. У нас вторая форма называется Form2, поэтому, сначала мы создаем объект данного класса, а потом для его отображения на экране вызываем метод Show:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 newForm = new Form2();
    newForm.Show();
}
```

Теперь сделаем наоборот - чтобы вторая форма воздействовала на первую. Пока вторая форма не знает о существовании первой. Чтобы это исправить, надо второй форме как-то передать сведения о первой форме. Для этого воспользуемся передачей ссылки на форму в конструкторе.

Итак, перейдем ко второй форме и перейдем к ее коду - нажмем правой кнопкой мыши на форму и выберем **View Code** (Просмотр кода). Пока он пустой и содержит только конструктор. Поскольку C# поддерживает перегрузку методов, то мы можем создать несколько методов и конструкторов с разными параметрами и в зависимости от ситуации вызывать один из них. Итак, изменим файл кода второй формы на следующий:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
        public Form2(Form1 f)
        {
            InitializeComponent();
            f.BackColor = Color.Yellow;
        }
    }
}
```

Фактически мы только добавили здесь новый конструктор public Form2(Form1 f), в котором мы получаем первую форму и устанавливаем ее фон в желтый цвет. Теперь перейдем к коду первой формы, где мы вызывали вторую форму и изменим его на следующий:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 newForm = new Form2(this);
    newForm.Show();
}
```

Поскольку в данном случае ключевое слово this представляет ссылку на текущий объект - объект Form1, то при создании второй формы она будет получать ее (ссылку) и через нее управлять первой формой.

Теперь после нажатия на кнопку у нас будет создана вторая форма, которая сразу изменит цвет первой формы.

Мы можем также создавать объекты и текущей формы:

```
private void button1_Click(object sender, EventArgs e)
{
    Form1 newForm1 = new Form1();
    newForm1.Show();
    Form2 newForm2 = new Form2(newForm1);
    newForm2.Show();
}
```

При работе с несколькими формами надо учитывать, что одна из них является главной - которая запускается первой в файле Program.cs. Если у нас одновременно открыта куча форм, то при закрытии главной закрывается все приложение и вместе с ним все остальные формы.

События в Windows Forms. События формы

Для взаимодействия с пользователем в Windows Forms используется механизм событий. События в Windows Forms представляют стандартные события на C#, только применяемые к визуальным компонентам и подчиняются тем же правилам, что события в C#. Но создание обработчиков событий в Windows Forms все же имеет некоторые особенности.

Прежде всего, в WinForms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов. Отдельные элементы добавляют свои события, но принципы работы с ними будут похожие. Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм.

Чтобы добавить обработчик, можно просто два раза нажать по пустому полю рядом с названием события, и после этого Visual Studio автоматически сгенерирует обработчик события.

И в этом поле отобразится название метода обработчика события Load. По умолчанию он называется Form1_Load.

Если мы перейдем в файл кода формы Form1.cs, то увидим автосгенерированный метод Form1_Load:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e)
    {
    }
}
```

И при каждой загрузке формы будет срабатывать код в обработчике Form1_Load.

Как правило, большинство обработчиков различных визуальных компонентов имеют два параметра: sender - объект, инициировавший событие, и аргумент, хранящий информацию о событии (в данном случае EventArgs e).

Но это только обработчик. Добавление же обработчика, созданного таким образом, производится в файле Form1.Designer.cs:

```
namespace HelloApp
{
    partial class Form1
    {
```

```
            private System.ComponentModel.IContainer components = null;
            protected override void Dispose(bool disposing)
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }
            private void InitializeComponent()
            {
                this.SuspendLayout();
                this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
                this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
                this.ClientSize = new System.Drawing.Size(284, 261);
                this.Name = "Form1";
                // добавление обработчика
                this.Load += new System.EventHandler(this.Form1_Load);
                this.ResumeLayout(false);
            }
        }
    }
```

Для добавления обработчика используется стандартный синтаксис C#: **this.Load += new System.EventHandler(this.Form1_Load)**

Поэтому если мы захотим удалить созданный подобным образом обработчик, то нам надо не только удалить метод из кода формы в Form1.cs, но и удалить добавление обработчика в этом файле.

Однако мы можем добавлять обработчики событий и програмно, например, в конструкторе формы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.Load += LoadEvent;
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }
        private void LoadEvent(object sender, EventArgs e)
        {
            this.BackColor = Color.Yellow;
        }
    }
}
```

Кроме ранее созданного обработчика Form1_Load здесь также добавлен другой обработчик загрузки формы: **this.Load += LoadEvent;**, который устанавливает в качестве фона желтый цвет.

Создание непрямоугольных форм. Закрытие формы

По умолчанию все формы в Windows Forms являются прямоугольными. Однако мы можем создавать и непрямоугольные произвольные формы. Для этого используется свойство **Region**. В качестве значения оно принимает объект одноименного класса **Region**.

При создании непрямоугольных форм, как правило, не используются границы формы, так как границы задаются этим объектом **Region**. Чтобы убрать границы формы, надо присвоить у формы свойству **FormBorderStyle** значение **None**.

И еще один аспект, который надо учитывать, заключается в перемещении, закрытии, максимизации и минимизации форм. То есть в данном случае , как в обычной форме, мы не сможем нажать на крестик, чтобы закрыть форму, не сможем ее переместить на новое место. Поэтому нам надо дополнительно определять для этого программную логику.

Итак, перейдем к коду формы и изменим его следующим образом:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form1 : Form
    {
        Point moveStart; // точка для перемещения
        public Form1()
        {
            InitializeComponent();
            this.FormBorderStyle = FormBorderStyle.None;
            this.BackColor = Color.Yellow;
            Button button1 = new Button
            {
                Location = new Point
                {
                    X = this.Width / 3,
                    Y = this.Height / 3
                }
            };
            button1.Text = "Закрыть";
            button1.Click += button1_Click;
            this.Controls.Add(button1); // добавляем кнопку на форму
            this.Load += Form1_Load;
            this.MouseDown += Form1_MouseDown;
            this.MouseMove += Form1_MouseMove;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            System.Drawing.Drawing2D.GraphicsPath myPath = new
System.Drawing.Drawing2D.GraphicsPath();
            // создаем эллипс с высотой и шириной формы
            myPath.AddEllipse(0, 0, this.Width, this.Height);
            // создаем с помощью эллипса ту область формы, которую мы
хотим видеть
            Region myRegion = new Region(myPath);
            // устанавливаем видимую область
            this.Region = myRegion;
        }
```

```
        private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            // если нажата левая кнопка мыши
            if (e.Button == MouseButtons.Left)
            {
                moveStart = new Point(e.X, e.Y);
            }
        }
       private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            // если нажата левая кнопка мыши
            if ((e.Button & MouseButtons.Left) != 0)
            {
                // получаем новую точку положения формы
                Point deltaPos = new Point(e.X - moveStart.X, e.Y -
moveStart.Y);
               // устанавливаем положение формы
                this.Location = new Point(this.Location.X + deltaPos.X,
                  this.Location.Y + deltaPos.Y);
            }
        }
    }
```

Создание области формы происходит в обработчике события **Form1_Load**. Для создания области используется графический путь - объект класса **System.Drawing.Drawing2D.GraphicsPath**, в который добавляется эллипс. Графический путь позволяет создать фигуру любой формы, поэтому, если мы захотим форму в виде морской звезды, то нам просто надо должным образом настроить используемый графический путь.

Для закрытия формы в обработчике события нажатия кнопки **button1_Click** форма закрывается программным образом: **this.Close()**

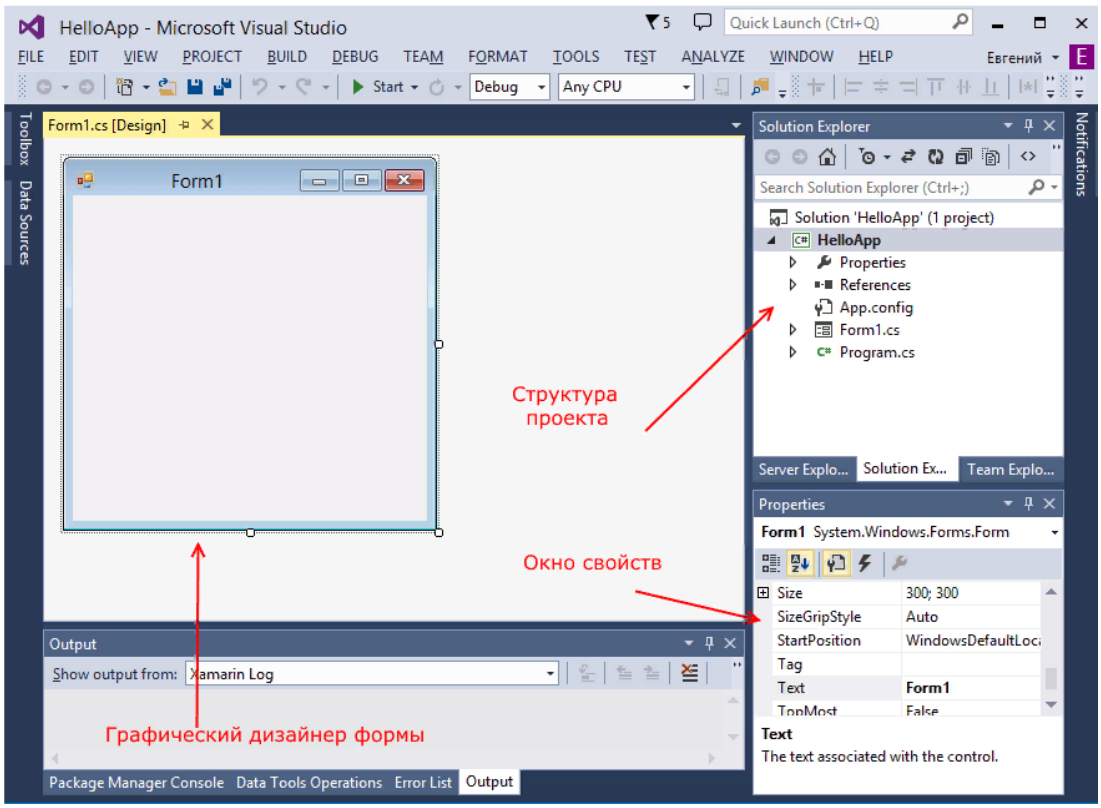
Для перемещения формы обрабатываются два события формы - событие нажатия кнопки мыши и событие перемещения указателя мыши.

Создание графического приложения

После установки среды и всех ее компонентов, запустим Visual Studio и создадим проект графического приложения. Для этого в меню выберем пункт **File** (Файл) и в подменю выберем **New - > Project** (Создать - > Проект).

В левой колонке выберем **Windows Desktop**, а в центральной части среди типов проектов - тип Windows Forms Application и дадим ему какое-нибудь имя в поле внизу. Например, назовем его HelloApp. После этого нажимаем OK.

После этого Visual Studio откроет наш проект с созданными по умолчанию файлами:

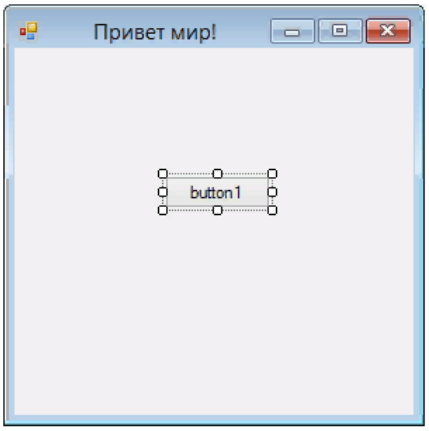


Большую часть пространства Visual Studio занимает графический дизайнер, который содержит форму будущего приложения. Пока она пуста и имеет только заголовок **Form1**. Справа находится окно файлов решения/проекта - **Solution Explorer** (Обозреватель решений). Там и находятся все связанные с нашим приложением файлы, в том числе файлы формы **Form1.cs**.

Внизу справа находится окно свойств - **Properties**. Так как у меня в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

Теперь перенесем на поле какой-нибудь элемент управления, например, кнопку. Для этого найдем в левой части Visual Studio вкладку **Toolbox** (Панель инструментов). Нажмем на эту вкладку, и у нас откроется панель с элементами, откуда мы можем с помощью мыши перенести на форму любой элемент:

Найдем среди элементов кнопку и, захватив ее указателем мыши, перенесем на форму:



Это визуальная часть. Теперь приступим к самому программированию. Добавим простейший код на языке C#, который бы выводил сообщение по нажатию кнопки. Для

этого мы должны перейти в файл кода, который связан с этой формой. Если у нас не открыт файл кода, мы можем нажать на форму правой кнопкой мыши и в появившемся меню выбрать **View Code** (Посмотреть файл кода).

Однако воспользуемся другим способом, чтобы не писать много лишнего кода. Наведем указатель мыши на кнопку и щелкнем по ней двойным щелчком. Мы автоматически попадаем в файл кода **Form1.cs**, который выглядит так:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Добавим вывод сообщения по нажатию кнопки, изменив код следующим образом:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace HelloApp
{
    public partial class Form1 : Form
    {
        public Form1()
       {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Привет");
        }
    }
}
```

Запуск приложения

Чтобы запустить приложение в режиме отладки, нажмем на клавишу **F5** или на зеленую стрелочку на панели Visual Studio. После этого запустится наша форма с одинокой кнопкой. И если мы нажмем на кнопку на форме, то нам будет отображено сообщение с приветствием.

После запуска приложения студия компилирует его в файл с расширением exe. Найти данный файл можно, зайдя в папку проекта и далее в каталог **bin/Debug** или **bin/Release**

Контрольные вопросы

1. Перечислите основные свойства форм
2. Как выполняется программная настройка свойств?
3. Как задается фон и цвета формы?
4. Опишите алгоритм добавления форм и взаимодействия между формами
5. Как выполняется создание непрямоугольных форм, закрытие формы?