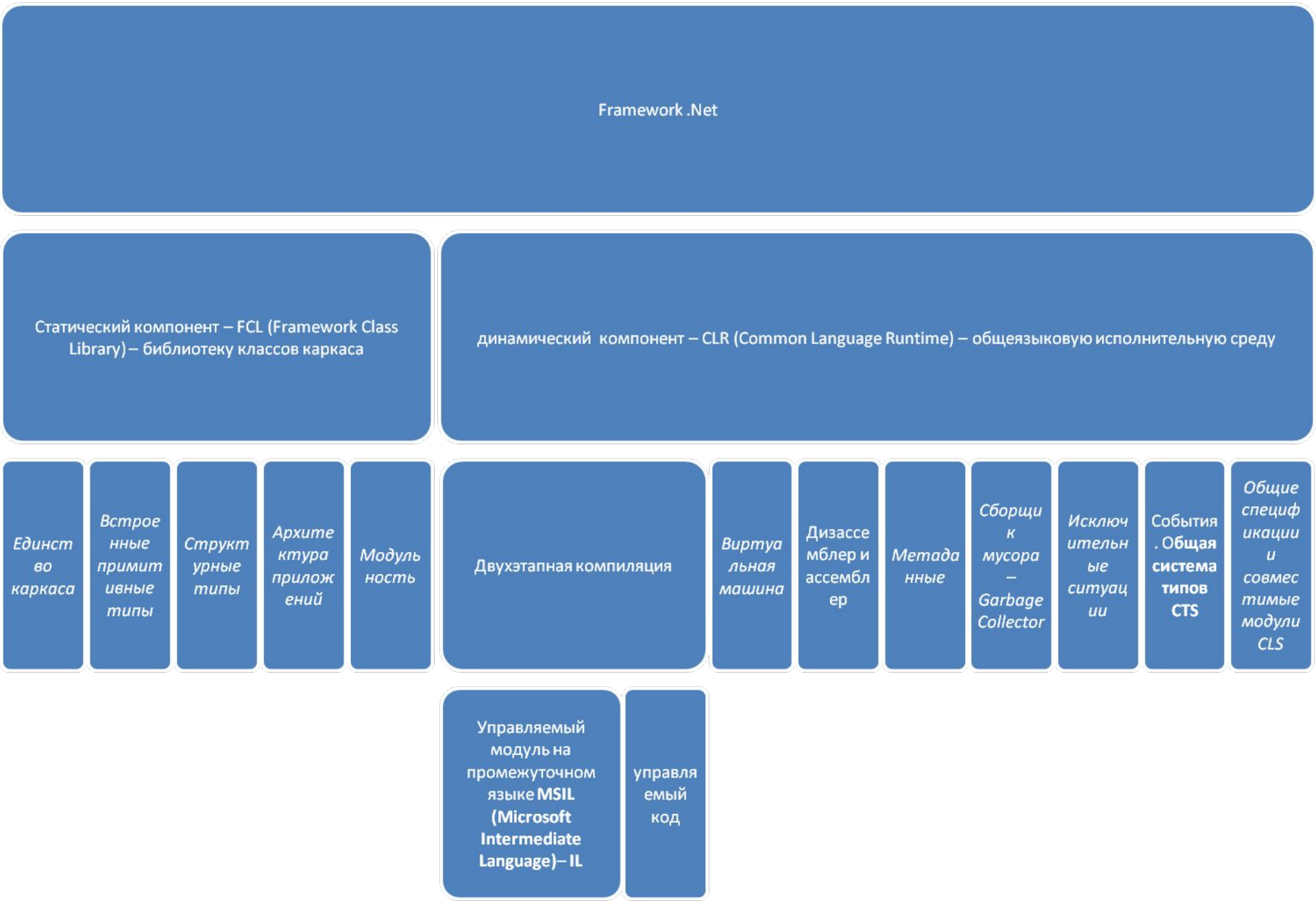
C#. Основные понятия и определения

Visual Studio .Net

Среда разработки Visual Studio .Net – это уже проверенный временем программный продукт, являющийся седьмой версией студии. Но новинки этой версии, связанные с идей .Net, позволяют считать ее принципиально новой разработкой, определяющей новый этап в создании программных продуктов. Выделю две важнейшие, на мой взгляд, идеи:

- **открытость** для **языков** программирования;
- принципиально новый подход к построению **каркаса среды** – **Framework .Net**.

Framework .Net – единый каркас среды разработки



Преимущества Visual Studio и C#

Язык C#, оптимизированный для программирования в Windows, вместе с Visual Studio позволяет сфокусироваться на непосредственных задачах.

Основная операция

Основной операцией, инициирующей вычисления в объектно-ориентированных приложениях, является вызов метода **F** некоторого класса **x**, имеющий вид:

x.F(arg1, arg2, …, argN);

В этом вызове **x** называется целью вызова, и здесь возможны три ситуации:

x – имя класса. В этом случае метод **F** должен быть статическим методом класса, объявленным с атрибутом static, как это имеет место, например, для точки вызова – процедуры **Main**.

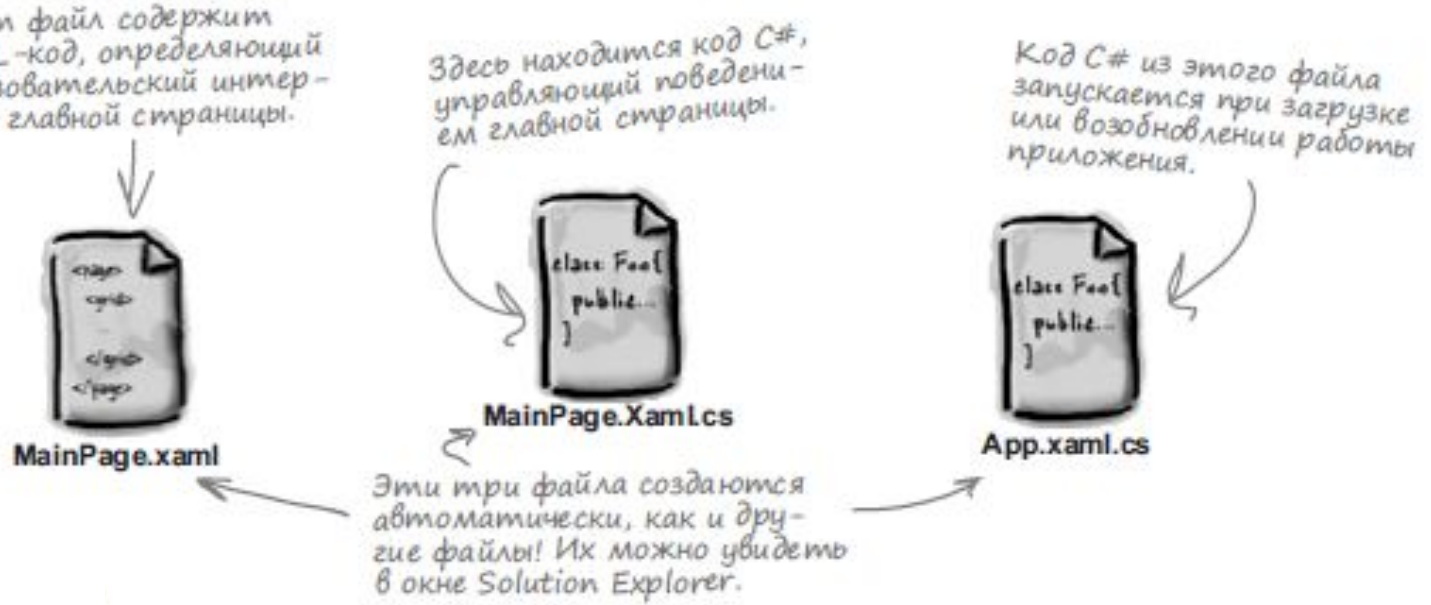
x – имя объекта или объектное выражение. В этом случае **F** должен быть обычным, не статическим методом. Иногда такой метод называют **экземплярным**, подчеркивая тот факт, что метод вызывается экземпляром класса – некоторым объектом.

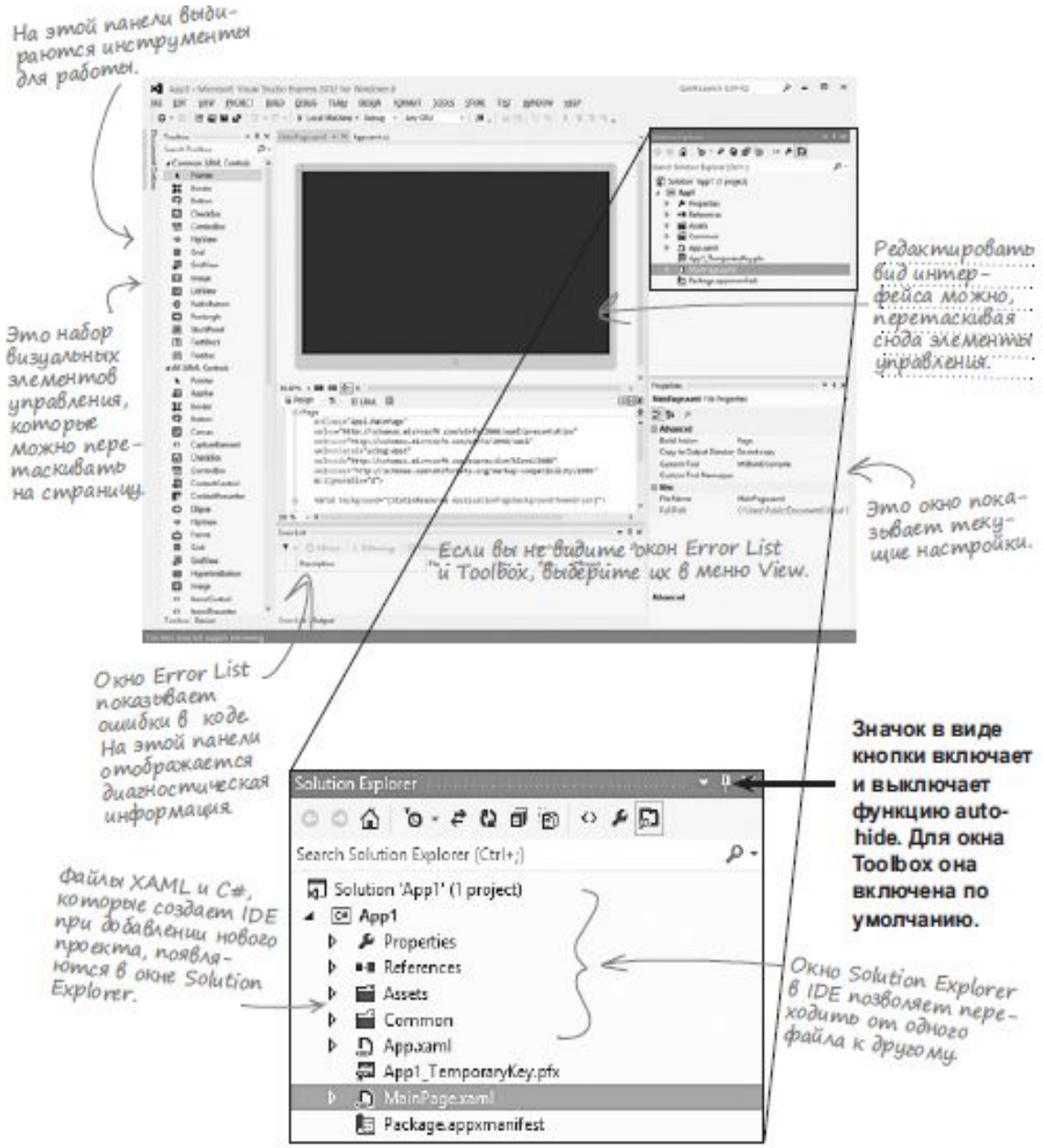
x – не указывается при вызове. Такой вызов называется неквалифицированным, в отличие от двух первых случаев. Заметьте, неквалифицированный вызов вовсе не означает, что цель вызова отсутствует, – она просто задана по умолчанию. Целью является текущий объект (текущий класс для статических методов). Текущий объект имеет зарезервированное имя this. Применяя это имя, любой неквалифицированный вызов можно превратить в квалифицированный вызов. Иногда без этого имени просто не обойтись.

А это Visual Studio делает за Вас

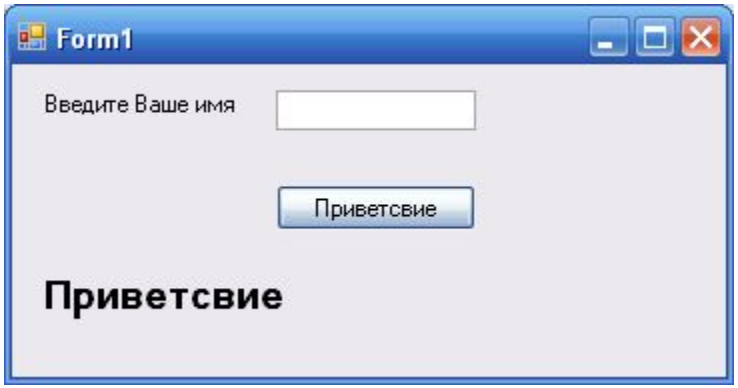
Команда Save All из меню File сохраняет все открытые файлы, в то время как команда Save — только файл, активный в данный момент.

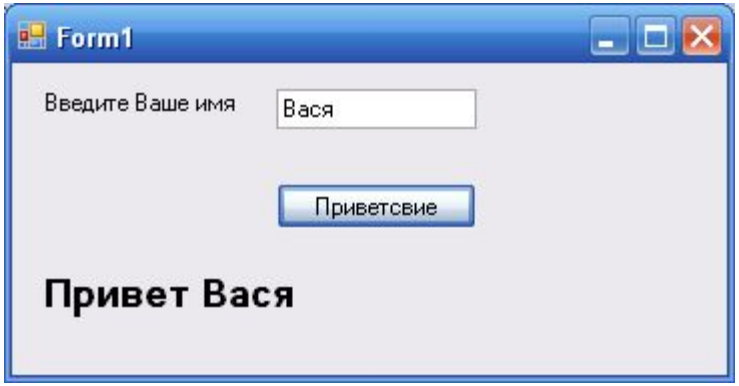
В момент сохранения проекта IDE создает набор файлов, в том числе файлы *MainPage.xaml*, *MainPage.Xaml.cs* и *App.xaml.cs* для новых проектов. Они добавляются в окно Solution Explorer и по умолчанию попадают в папку *Projects\App1\App1*.

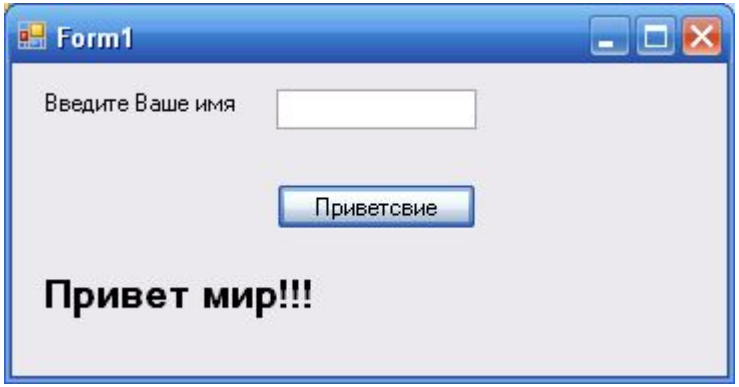



На этой панели выдаются инструменты для работы.
Это набор визуальных элементов управления, которые можно перетаскивать на страницу.
Редактировать вид интерфейса можно, перетаскивая сюда элементы управления.
Это окно показывает текущие настройки.
Если вы не видите окон Error List и Toolbox, выберите их в меню View.
Окно Error List показывает ошибки в коде. На этой панели отображается диагностическая информация.
Значок в виде кнопки включает и выключает функцию auto-hide. Для окна Toolbox она включена по умолчанию.
Файлы XAML и C#, которые создает IDE при добавлении нового проекта, появляются в окне Solution Explorer.
Окно Solution Explorer в IDE позволяет переходить от одного файла к другому.
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'App1' (1 project)
App1
Properties
References
Assets
Common
App.xaml
App1_TemporaryKey.pfx
MainPage.xaml
Package.appxmanifest

Пример. Создадим приложение для приветствия пользователя








```
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 prim1_lek1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         string s;
         s = textBox1.Text;
         if (s != "")
            label2.Text = "Привет " + s;
         else
            label2.Text = "Привет мир!!!";
      }   } }
```

Progam.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace prim1_lek1
{
   static class Program
   {
      /// <summary>
      /// Главная точка входа для приложения.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }   } }
```

Form1.Designer.cs

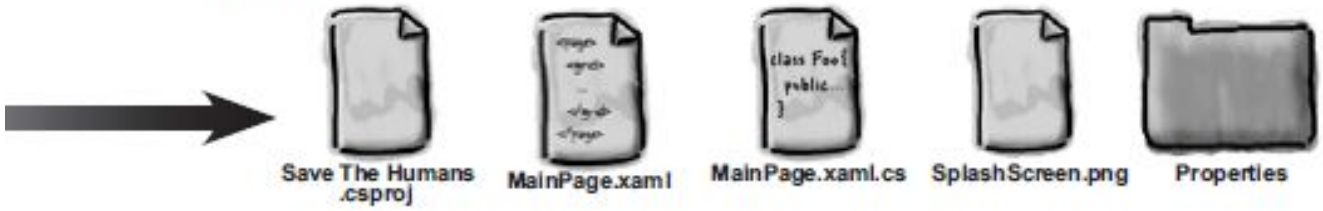
```
namespace prim1_lek1
{
    partial class Form1
    {
        /// <summary>
        /// Требуется переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Код, автоматически созданный конструктором форм Windows
        /// <summary>
        /// Обязательный метод для поддержки конструктора - не изменяйте
        /// содержимое данного метода при помощи редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(13, 13);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(102, 13);
            this.label1.TabIndex = 0;
            this.label1.Text = "Введите Ваше имя";
            //
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(132, 13);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(100, 20);
            this.textBox1.TabIndex = 1;
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif", 14F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.label2.Location = new System.Drawing.Point(12, 103);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(130, 24);
            this.label2.TabIndex = 2;
            this.label2.Text = "Приветсвие";
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(132, 60);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(100, 23);
            this.button1.TabIndex = 3;
            this.button1.Text = "Приветсвие";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(357, 156);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.label2);
            this.Controls.Add(this.textBox1);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);
            this.PerformLayout();
        }
        #endregion
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button button1;
    }
}
```

...IDE делает это.

Любые ваши действия приводят к изменениям кода, а значит, и файлов, которые содержат этот код. Иногда редактируются всего несколько строчек, а иногда система создает новые файлы.

Эти файлы были созданы из предустановленного шаблона, содержащего базовый код создания и отображения страницы.

1 **...IDE создает для проекта файлы и папки.**



2 **... IDE добавляет к файлу MainPage.xaml код кнопки и метод MainPage.xaml.cs, который запускается при каждом щелчке на этой кнопке.**

```
private void startButton_Click(object sender, RoutedEventArgs e)
{

}
```

IDE умеет добавлять пустой метод обработки щелчков на кнопке. Но IDE не знает, что должно находиться внутри этого метода. Это должны написать вы.

3 **...IDE открывает файл MainPage.xaml и обновляет строку кода XAML.**

IDE входит в этот файл...

```
<Button x:Name="startButton"
                Content="Start!"
                HorizontalAlignment="Center"
                VerticalAlignment="Center" Click="startButton_Click"/>
```

Структура программы

Структура программы

Код всех программ на C# структурирован одинаково. Везде для простоты управления кодом применяются **пространства имен**, **классы** и **методы**.

Внимательно рассмотрим код

1 **Файл кода начинается с перечисления инструментов .NET Framework.**
В верхней части любого файла программы находится набор строк с оператором `using`. Они указывают, с какой частью .NET Framework или Windows Store API будет работать программа. Чтобы воспользоваться классами из других пространств имен, нужно указать их с помощью оператора `using`. Приложения часто задействуют инструменты .NET Framework и Windows Store API, поэтому в верхней части файла страницы можно увидеть много операторов `using`.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
```

Подобные строки находятся в верхней части любого файла с кодом. Они указывают C#, какие классы из .NET Framework следует использовать. Каждый такой оператор указывает программе, что классы в определенном файле .cs будут пользоваться классами определенного пространства имен .NET Framework (System) или Windows Store API.

В принципе, без оператора `using` можно обойтись, если пользоваться полными именами. В приложении Save the Humans есть строка:

```
using Windows.UI.Xaml.Media.Animation;
```

Превратите ее в комментарий, поставив в начале `//`, и изучите ошибки. Одну из них можно исправить. Найдите слово `Storyboard`, которое IDE подчеркивает как ошибку, и вместо него напишите `Windows.UI.Xaml.Media.Animation.Storyboard` (но чтобы программа заработала, следует убрать добавленный комментарий).

2 **Программы C# используют классы**

Любая программа на C# использует классы. Класс может делать что угодно, но большинство классов заточено на определенное действие. При создании программы добавляется отображающий страницу класс `MainPage`.

При присвоении программе имени Save the Humans IDE создала пространство имен Save_the_Humans (пробелы при этом автоматически заменяются нижним подчеркиванием), добавив ключевое слово namespace. Все находящееся в скобках является частью пространства имен Save_the_Humans.

```
namespace Save_the_Humans
{
    public sealed partial class MainPage : Page
    {
```

Это класс MainPage, который содержит код, заставляющий вашу страницу работать. IDE формирует его, получая команду создать проект Windows Store.

3 **Классы содержат методы**

Для выполнения различных действий классы используют методы. Методы производят некое действие на основе входных данных. Данные в метод передаются при помощи параметров. Именно параметры влияют на поведение метода. Некоторые методы возвращают значение. Ключевое слово `void` перед именем метода означает, что он не возвращает никаких данных.

Обращайте внимание на пары скобок. Среди них могут попадаться вложенные.

```
        void startButton_Click(object sender, object e)
        {
            StartGame();
        }
```

*Эта строка **вызывает метод** StartGame(), в создании которого вам помогла IDE, когда вы попросили добавить заглушку метода.*

*У этого метода два **параметра**: sender и e.*

4 **Каждый оператор выполняет всего одно действие**

При заполнении метода `StartGame()` вы добавили набор **операторов**. Именно операторы составляют тело любого метода. При вызове метода сначала выполняется первый оператор, потом следующий и т. д. При достижении конца списка или оператора `return` метод завершает работу, и программа продолжает работу после вызвавшего метод оператора.

*Это **метод** StartGame(), вызываемый при нажатии игроком кнопки Start.*

```
        private void StartGame()
        {
            human.IsHitTestVisible = true;
            humanCaptured = false;
            progressBar.Value = 0;
            startButton.Visibility =
                        Visibility.Collapsed;
            playArea.Children.Clear();
            playArea.Children.Add(target);
            playArea.Children.Add(human);
            enemyTimer.Start();
            targetTimer.Start();
        }
    }
}
```

*Метод StartGame() содержит девять **операторов**. После каждого оператора ставится точка с запятой.*

Для улучшения читабельности строки можно разрывать. При построении программы это игнорируется.

Это закрывающаяся фигурная скобка в самом низу вашего файла MainPage.xaml.cs.

Классификация типов

Стандарт языка C++ включает следующий набор **фундаментальных** типов:

- **Логический** тип (bool).
- **Символьный** тип (char).
- **Целые** типы. Целые типы могут быть одного из трех размеров – short, int, long, сопровождаемые описателем signed или unsigned, указывающим, на то, как интерпретируется значение, – со знаком или без оного.
- Типы с **плавающей точкой**. Эти типы также могут быть одного из трех размеров – float, double, long double.
- Кроме того, в языке есть Тип void, используемый для указания на отсутствие информации.
- Язык позволяет конструировать типы:
- **Указатели** (например, int* – типизированный указатель на переменную типа int).
- **Ссылки** (например, double& – типизированная ссылка на переменную типа double).
- **Массивы** (например, char[] – массив элементов типа char).

Язык позволяет конструировать пользовательские типы:

- **Перечислимые** типы (enum) для представления значений из конкретного множества.
- **Структуры** (struct)
- **Классы**.

Эта схема типов сохранена и в языке C#. Однако здесь на верхнем уровне используется и другая классификация, носящая для C# принципиальный характер. Согласно этой классификации все типы можно разделить на четыре категории:

- типы-**значения** (**value**) или **значимые** типы.
- **ссылочные** (**reference**).
- **указатели** (**pointer**).
- тип **void.**

Рассмотрим классификацию, согласно которой все типы делятся на **встроенные** и **определенные пользователем**.

Систем типов

Логический тип			
Имя типа	**Системный тип**	**Значения**	**Размер**
bool	**System.Boolean**	true, false	8 бит
Арифметические целочисленные типы			
Имя типа	**Системный тип**	**Диапазон**	**Размер**
sbyte	**System.SByte**	-128 – 128	Знаковое, 8-бит
byte	**System.Byte**	0 – 255	Беззнаковое, 8-бит
short	**System.Short**	-32768 –32767	Знаковое, 16-бит
ushort	**System.UShort**	0 – 65535	Беззнаковое, 16-бит
int	**System.Int32**	≈(-2*109 – 2*109)	Знаковое, 32-бит
uint	**System.UInt32**	≈(0 – 4*10)	Беззнаковое, 32-бит
long	**System.Int64**	≈(-9*1018 – 9*1018)	Знаковое, 64-бит
ulong	**System.UInt64**	≈(0– 18*1018)	Беззнаковое, 64-бит
Арифметический тип с плавающей точкой			
Имя типа	**Системный тип**	**Диапазон**	**Точность**
float	**System.Single**	±1.5 × 10-45 ±3.4 × 1038	7 цифр
double	**System.Double**	±5.0 × 10-324 ±1.7 × 10308	15-16 цифр
Арифметический тип с фиксированной точкой			
Имя типа	**Системный тип**	**Диапазон**	**Точность**
decimal	**System.Decimal**	±1.0 × 10-28 ±7.9 × 1028	28-29 значащих цифр
Символьные типы			
Имя типа	**Системный тип**	**Диапазон**	**Точность**
char	**System.Char**	U+0000 – U+ffff	16-бит Unicode символ
string	**System.String**	Строка из символов Unicode	
Объектный тип			
Имя типа	**Системный тип**	**Примечание**	
object	System.Object	Прародитель всех встроенных и пользовательских типов	

Типы и классы

Родительским, **базовым классом** является класс **object**. Все же остальные типы или, точнее, классы являются его **потомками**, наследуя методы этого класса. У класса **object** есть четыре наследуемых **метода**:

- bool **Equals(object obj)** – проверяет эквивалентность текущего объекта и объекта, переданного в качестве аргумента;
- **System.Type GetType()** – возвращает системный тип текущего объекта;
- string **ToString()** – возвращает строку, связанную с объектом. Для арифметических типов возвращается значение, преобразованное в строку;
- int **GetHashCode()** – служит, как хэш-функция в соответствующих алгоритмах поиска по ключу при хранении данных в хэш-таблицах.

Пример:

```
int x=11;
int v = new Int32();
v = 007;
string s1 = "Agent";
s1 = s1 + v.ToString() +x.ToString();
```

Семантика присваивания.

Рассмотрим присваивание:

x = **e**;

Чтобы присваивание было допустимым, типы переменной **x** и выражения **e** должны быть согласованными. Пусть сущность **x** согласно объявлению принадлежит классу **T**. Будем говорить, что тип **T** основан на классе **T** и является базовым типом **x**, так что базовый тип определяется классом объявления. Пусть теперь в рассматриваемом нами присваивании выражение **e** связано с объектом типа **T1**.

Например, пусть задан некоторый класс Parent, а класс Child – его потомок, объявленный следующим образом:

class Child:Parent {…}

Пусть теперь в некотором классе, являющемся клиентом классов Parent и Child объявлены переменные этих классов и созданы связанные с ними объекты:

Parent p1 = new Parent(), p2 = new Parent();

Child ch1 = new Child(), ch2 = new Child();

Тогда допустимы присваивания:

p1 = p2; p2= p1; ch1=ch2; ch2 = ch1; p1 = ch1; p1 = ch2;

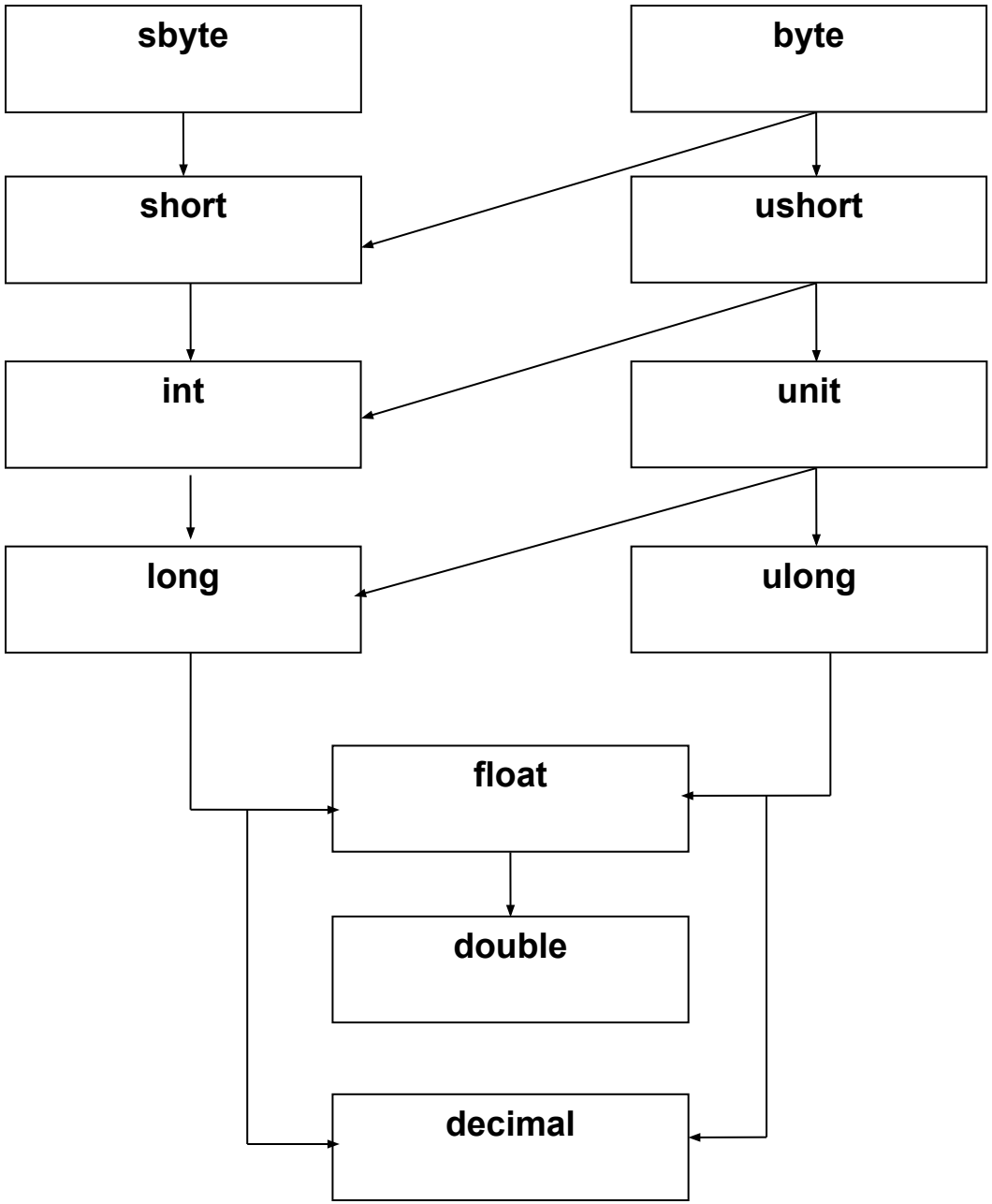
Но недопустимы присваивания:

ch1 = p1; ch2 = p1; ch2 = p2;

следующие присваивания допустимы:

p1 = ch1; … ch1 = (Child)p1;

Преобразования типов



Объявление переменных

//Объявления локальных переменных

[<атрибуты>] [<модификаторы>] <тип> <объявители>;

```
int x, s; //без инициализации
int y =0, u = 77; //обычный способ инициализации
//допустимая инициализация
float w1=0f, w2 = 5.5f, w3 =w1+ w2 + 125.25f;
//допустимая инициализация в объектном стиле
int z= new int();
// Недопустимая инициализация.
//Конструктор с параметрами не определен
int v = new int(77);
```

Всюду, где можно объявить переменную, можно объявить и именованную константу. Синтаксис объявления схож. В объявление добавляется модификатор const, инициализация констант обязательна и не может быть отложена. Инициализирующее выражение может быть сложным, важно, чтобы оно было вычислимым в момент его определения. Вот пример объявления констант:

```
const int SmallSize = 38, LargeSize =58;
const int MidSize = (SmallSize + LargeSize)/2;
const double pi = 3.141593;
LargeSize = 60; //Значение константы нельзя изменить.
```