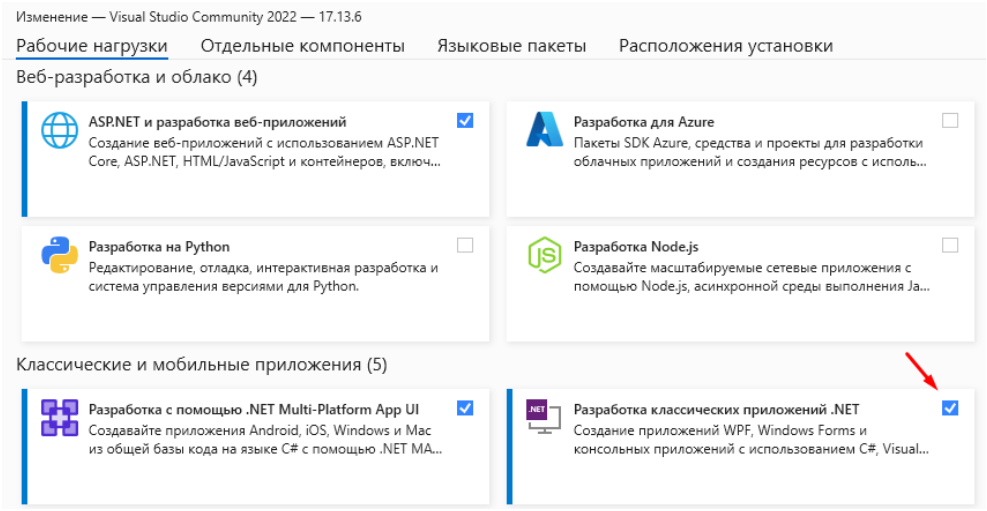
Лабораторная работа №11. Введение и разработка приложений в Windows Forms

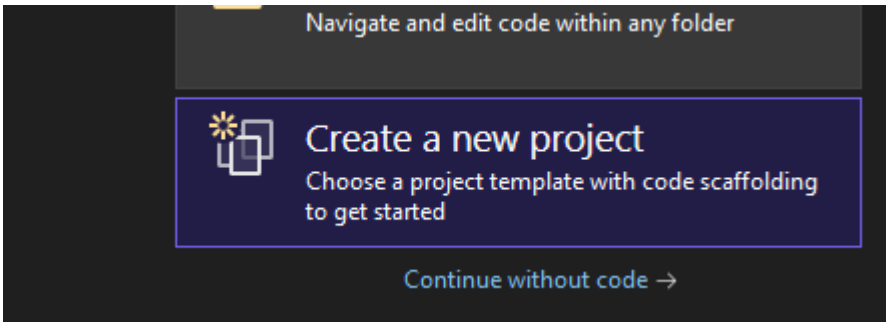
Для создания графических интерфейсов с помощью платформы **.NET** применяются разные технологии – **Windows Forms**, **WPF**, **UWP**. Однако наиболее простой и удобной платформой до сих пор остается **Windows Forms** или сокращенно **WinForms**. В данной лабораторной работе даются основы создания графических интерфейсов с помощью технологии **WinForms** и создания простых графических приложений.

Шаг 1. Создаём графическое приложение Window Forms в Visual Studio 2022.

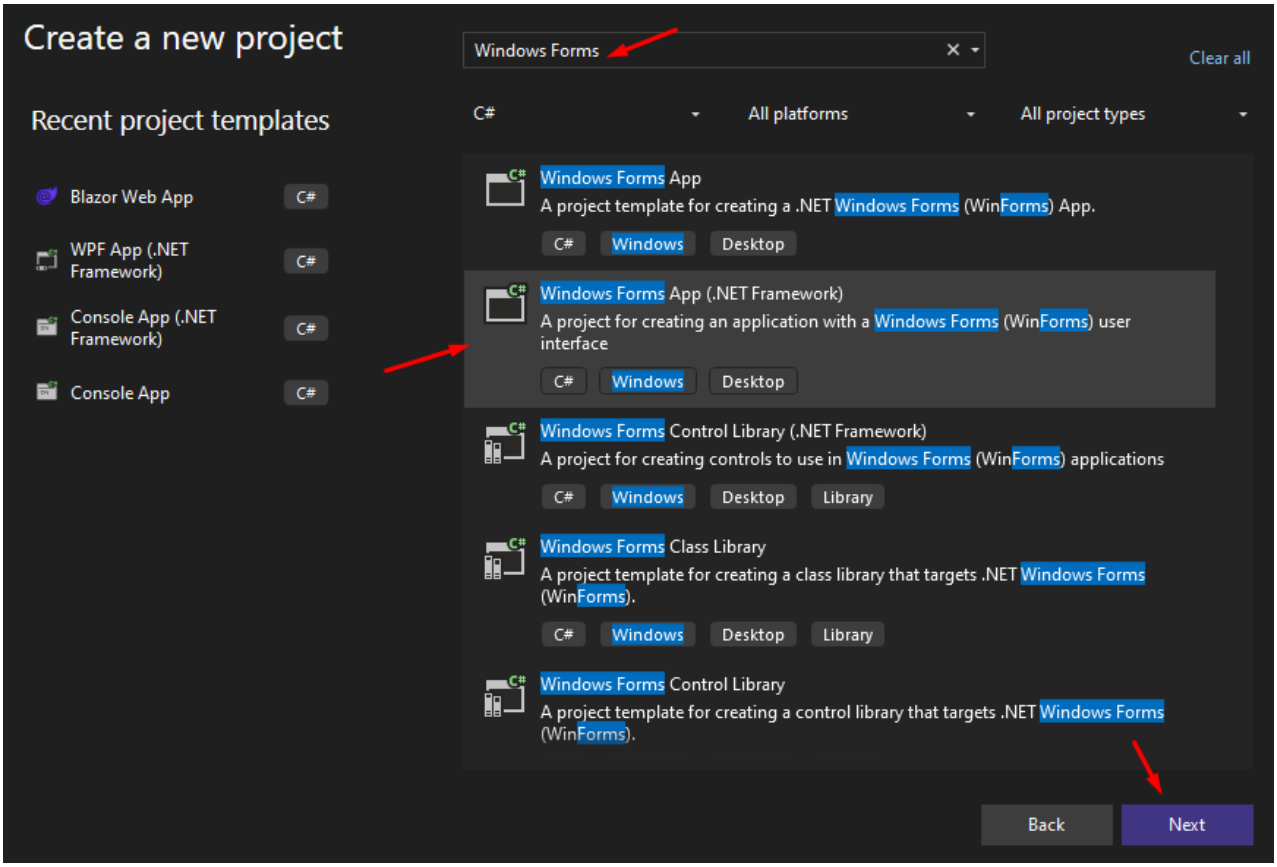
Для работы с **Windows Forms** в **Visual Studio 2022** необходимо наличие компонента ***"Разработка классических приложений .NET"*** в установщике.



Откройте среду разработки **Visual Studio 2022**. На стартовом окне выберите **Create a new project**:

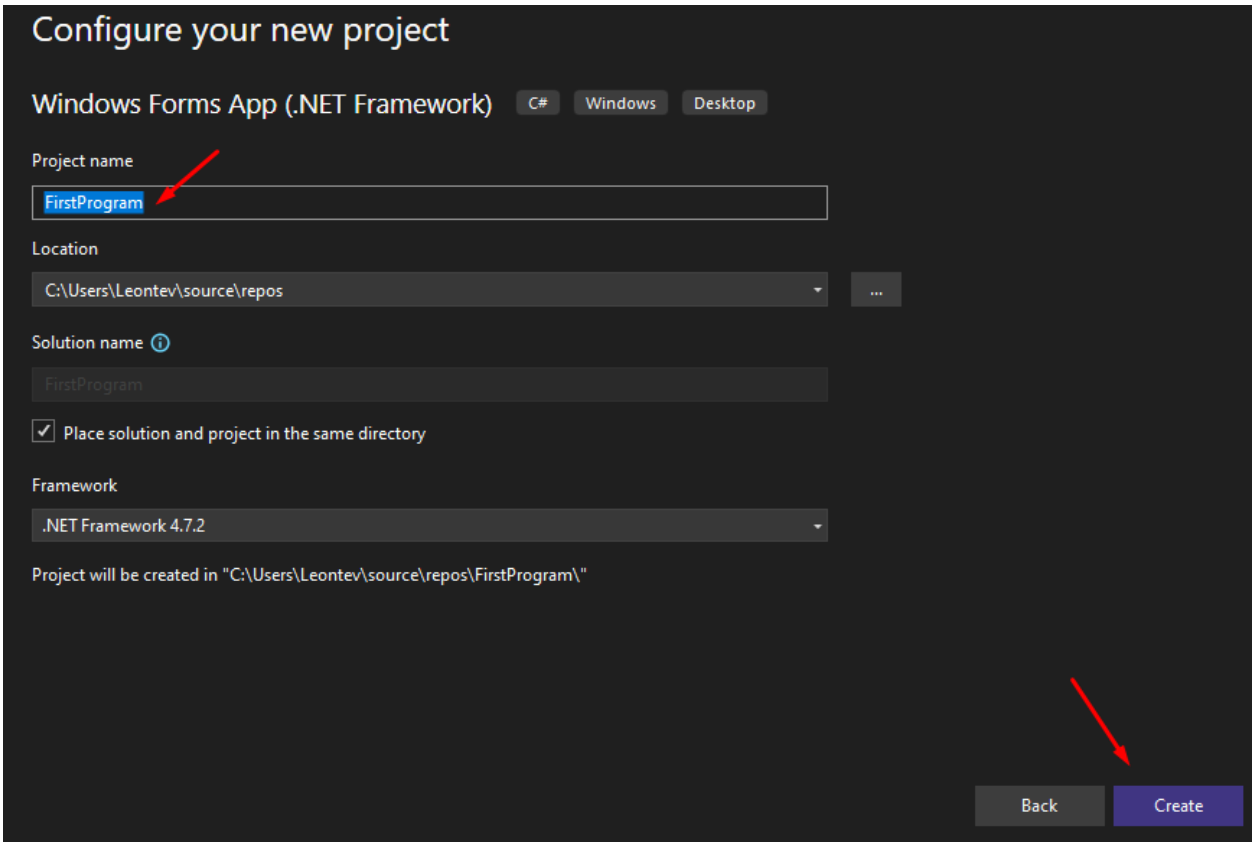


В строке поиска введите "**Windows Forms**". Выберите шаблон **Windows Forms App (.NET Framework)** и нажмите **Next**:

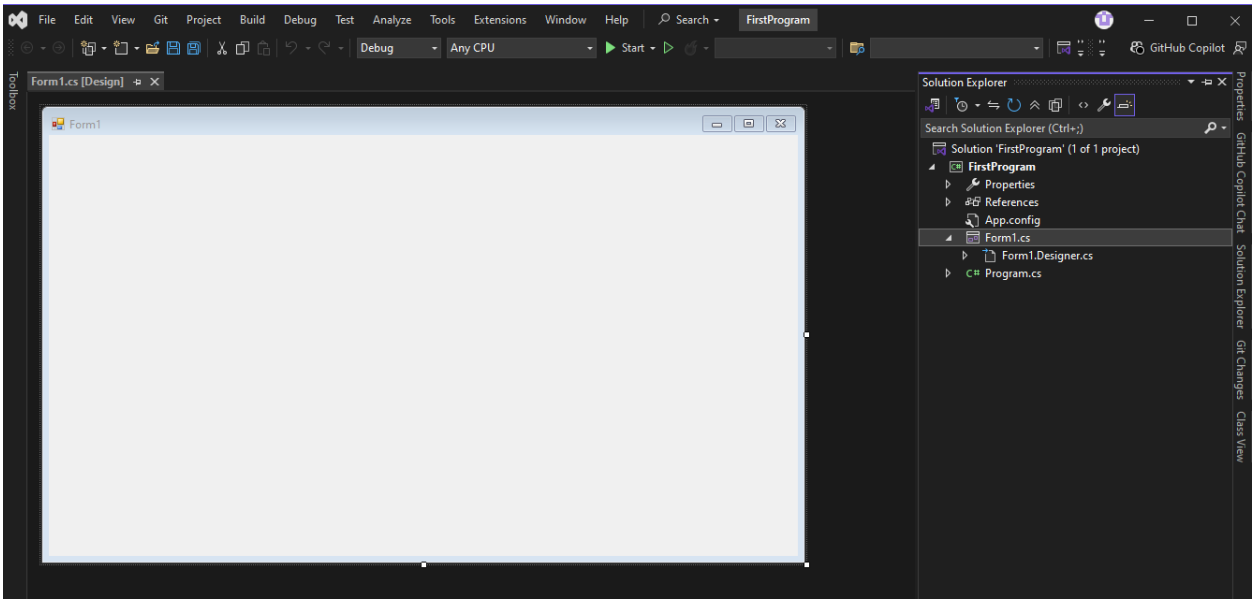


Примечание: Несмотря на использование устаревшего фреймворка, этот вариант обеспечивает совместимость со старыми версиями ОС.

В поле **Project name** укажите название проекта **FirstProgram** и нажмите **Create**:



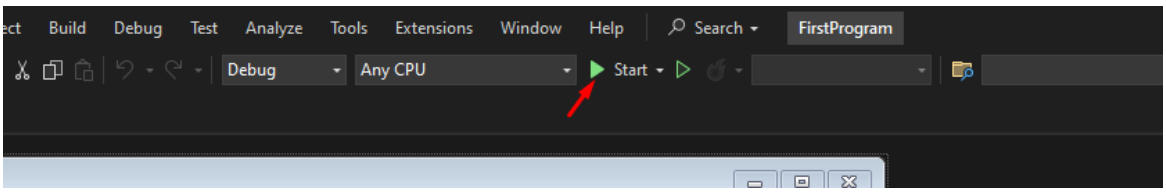
После создания откроется рабочая среда с файлами проекта:



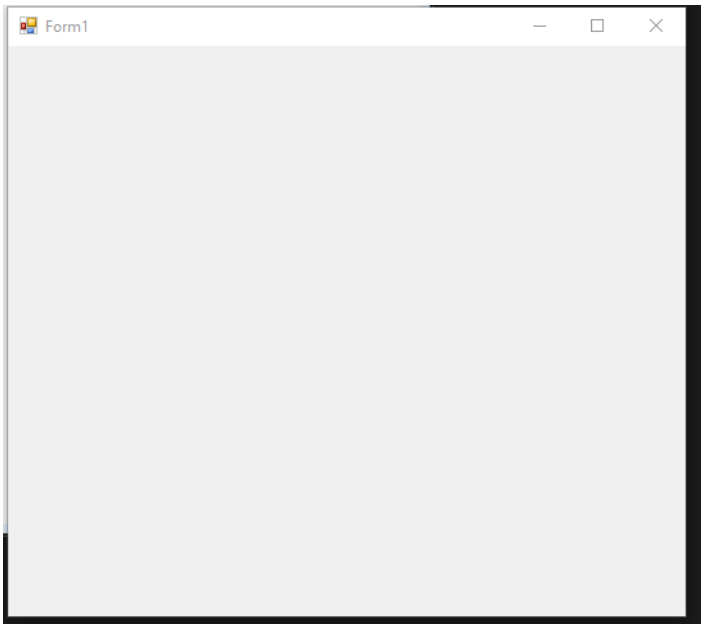
Справа в окне **Solution Explorer** можно увидеть структуру нашего проекта:

- ***Dependencies*** — сборки .NET, используемые в проекте.
- ***Form1.Designer.cs*** — автоматически генерируемый код формы.
- ***Form1.cs*** — основная форма (открыта по умолчанию).
- ***Program.cs*** — точка входа в приложение.

5. Чтобы запустить приложение в режиме отладки, нажмем на клавишу **F5** или на зеленую стрелочку на панели **Visual Studio**:



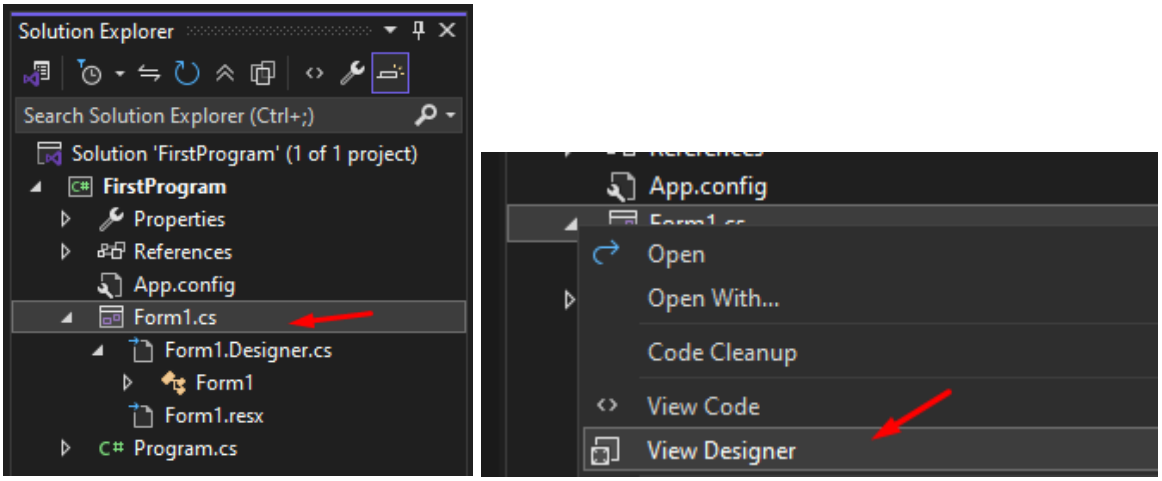
Откроется пустая форма **Form1**:



Шаг 2. Разработка простого приложения

Одним из преимуществ разработки в **Visual Studio** приложений **Windows Forms** является наличие графического редактора, который позволяет в графическом виде представить создаваемую форму и в принципе упрощает работу с графическими компонентами.

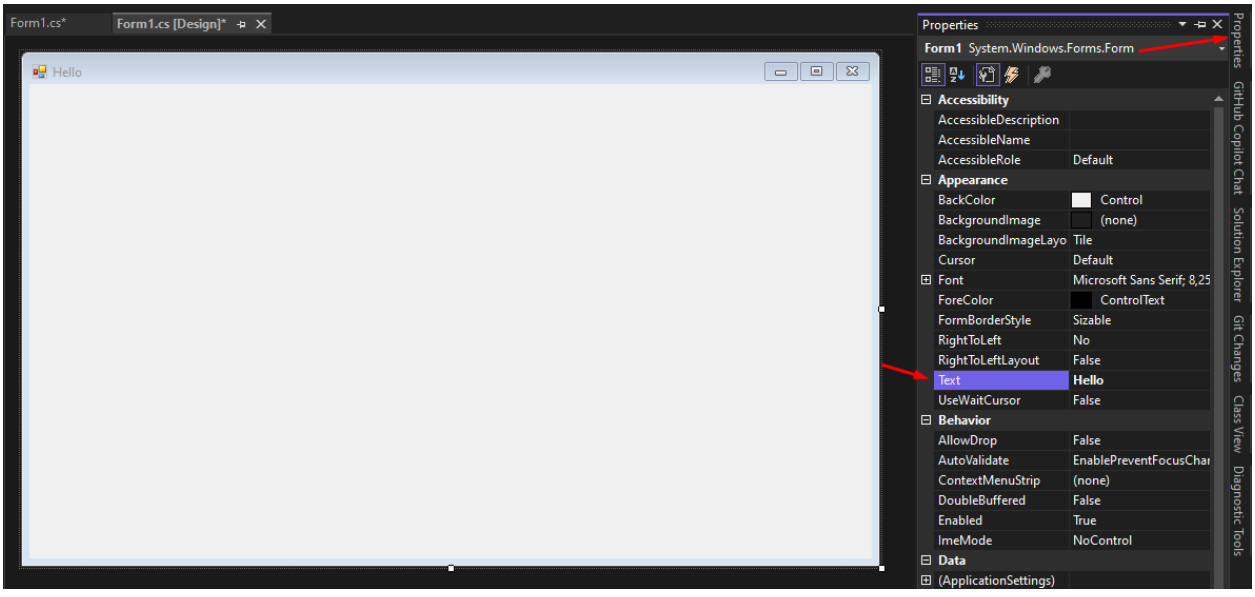
Для открытия формы в режиме графического дизайнера нажмем в структуре проекта на файл **Form1.cs** либо левой кнопкой мыши двойным кликом, либо правой кнопкой мыши и в появившемся контекстном меню выберем **View Designer** (также можно использовать комбинацию клавиш **Shift+F7**):



После этого в **Visual Studio** откроется выбранная форма в графическом виде.

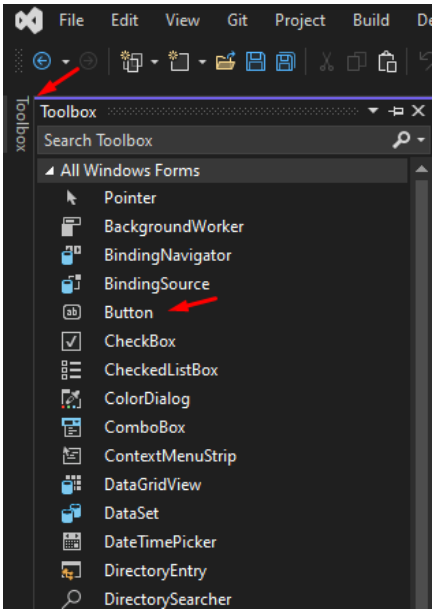
При выборе формы в окне дизайнера внизу справа под структурой проекта мы сможем найти окно **Properties**. Так как у меня в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

Теперь найдем в этом окне свойство формы **Text** и изменим его значение на **Hello**:

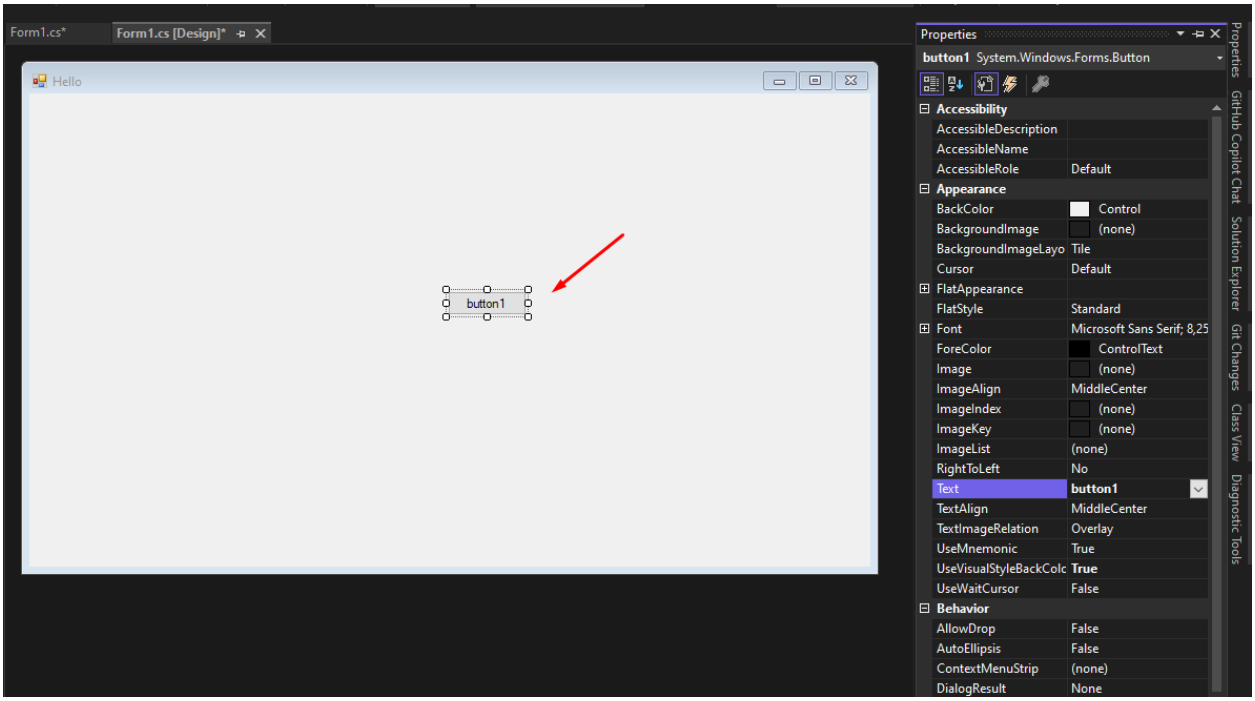


Таким образом мы поменяли заголовок формы. И подобным образом мы можем поменять другие свойства формы, которые доступны в окне свойств.

Но Visual Studio имеет еще одну связанную функциональность. Она обладает панелью графических инструментов. И мы можем, вместо создания элементов управления в коде C#, просто переносить их на форму с панели инструментов с помощь мыши. Так, перенесем на форму какой-нибудь элемент управления, например, кнопку. Для этого найдем в левой части **Visual Studio** вкладку **Toolbox**. Нажмем на эту вкладку, и у нас откроется панель с элементами, откуда мы можем с помощью мыши перенести на форму любой элемент:

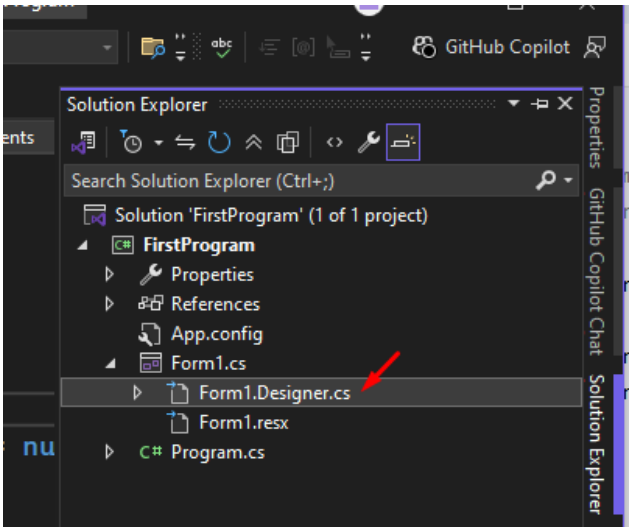


Найдем среди элементов **кнопку** и, захватив ее указателем мыши, перенесем на форму:



Причем при выборе кнопки она открывается в окне свойств и, как и для всей формы, для кнопки в окне свойств мы можем изменить значения различных свойств.

Кроме того, если после переноса кнопки на форму мы откроем файл **Form1.Designer.cs:**



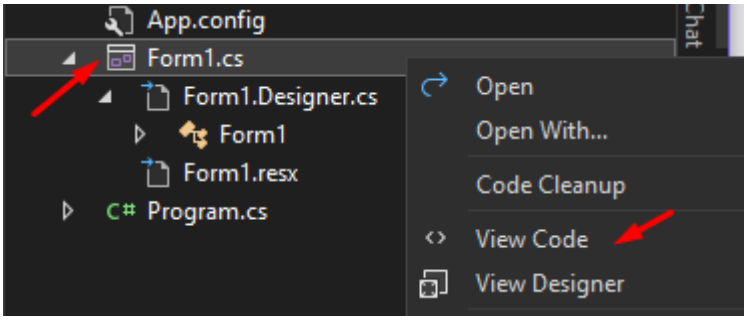
Мы увидим, что в класс **Form1** была добавлена переменная **button1** типа **Button** и для этой переменной, как и для объекта формы, задан ряд свойств:

```
33          //
34          // button1
35          //
36          this.button1.Location = new System.Drawing.Point(398, 188);
37          this.button1.Name = "button1";
38          this.button1.Size = new System.Drawing.Size(75, 23);
39          this.button1.TabIndex = 0;
40          this.button1.Text = "button1";
41          this.button1.UseVisualStyleBackColor = true;
42          //
43          // Form1
44          //
45          this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
46          this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
47          this.ClientSize = new System.Drawing.Size(800, 450);
48          this.Controls.Add(this.button1);
49          this.Name = "Form1";
50          this.Text = "Hello";
51          this.ResumeLayout(false);
52
53      }
54
55      #endregion
56
57      private System.Windows.Forms.Button button1;
58  }
```

И если в окне свойств мы поменяем значения этих свойств, то в этом файле также изменяться их значения. Поменяйте текст на **Bye**:

```
43          // Form1
44          //
45          this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
46          this.AutoScaleMode = System.Windows.
47          this.ClientSize = new System.Drawing.
48          this.Controls.Add(this.button1);
49          this.Name = "Form1";
50          this.Text = "Bye";
51          this.ResumeLayout(false);
52
```

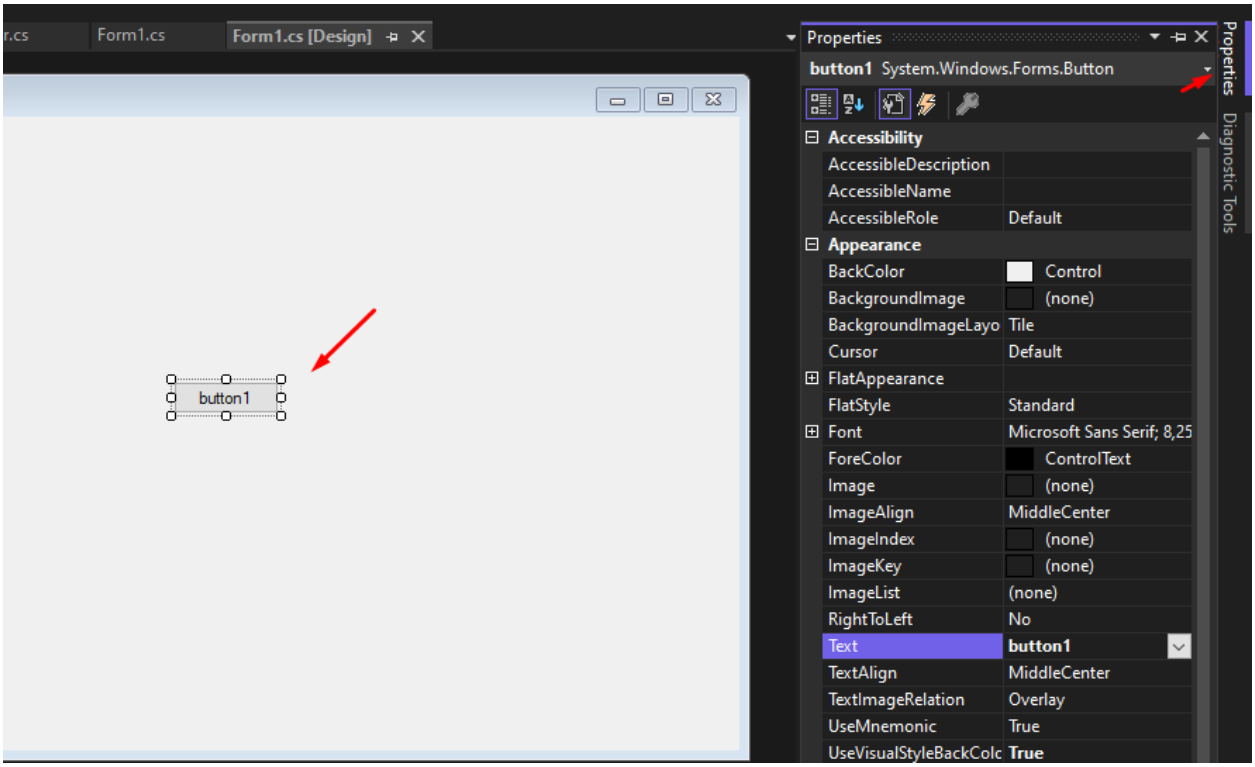
Это визуальная часть. Теперь приступим к самому программированию. Добавим простейший код на языке C#, который бы выводил сообщение по нажатию кнопки. Для этого перейдем в файл кода **Form1.cs**, который связан с этой формой:



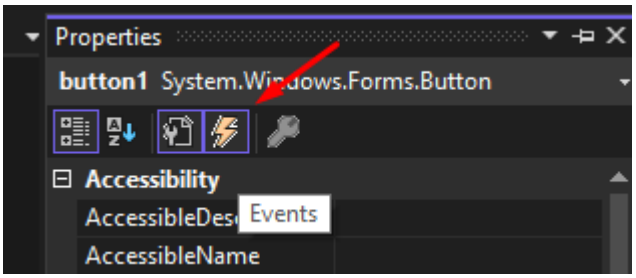
По умолчанию после создания проекта он имеет код:

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10
11  namespace FirstProgram
12  {
        3 references
13      public partial class Form1 : Form
14      {
            1 reference
15          public Form1()
16          {
17              InitializeComponent();
18          }
19      }
20  }
21
```

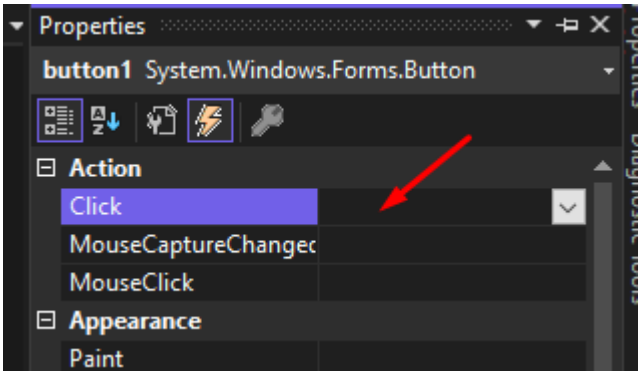
Снова вернёмся в окно **Designer**, выделим кнопку и перейдём в **Propertis**:



Теперь мы хотим добавить вывод сообщения при нажатии на кнопку. Нажмём на **Events**:



И щёлкнем два раза по поле **Click**:

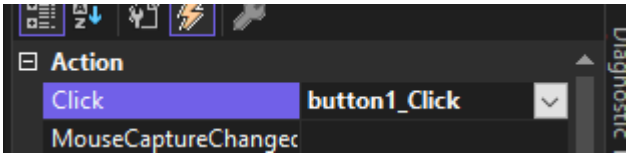


У вас сразу откроется код, куда будет вставлен обработчик событий:

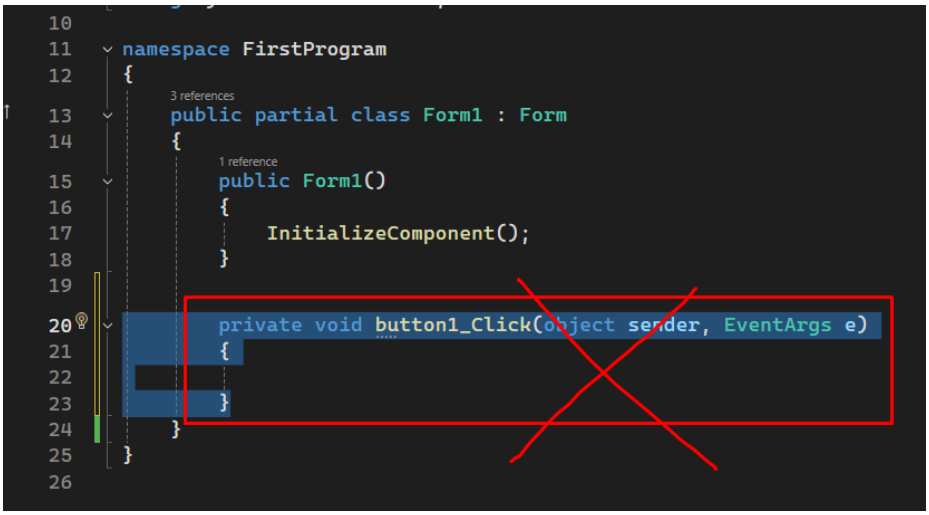
```
1 reference
private void button1_Click(object sender, EventArgs e)
{

}
```

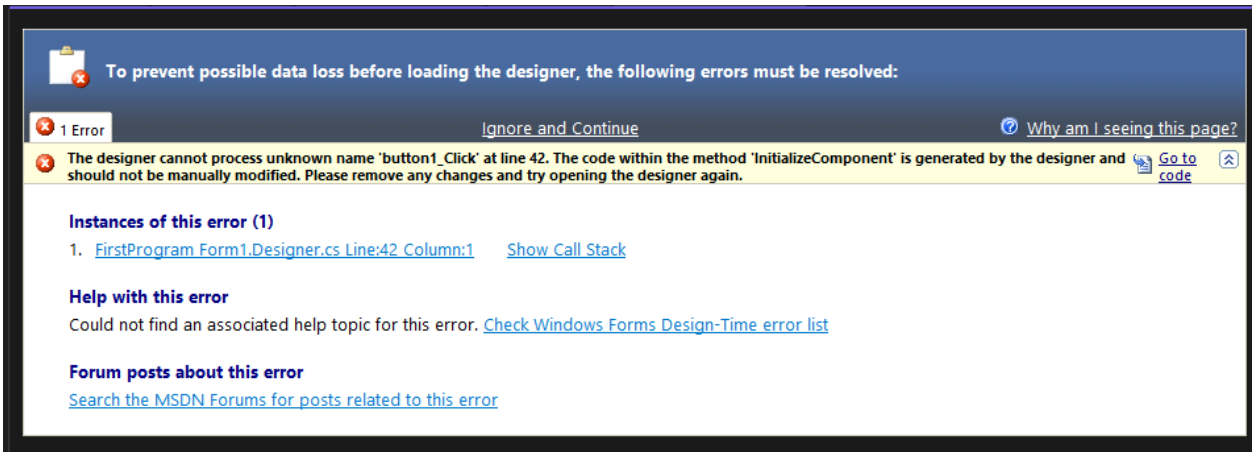
Но если мы вернёмся в **Designer**, то увидим, что на клик повесилось событие:



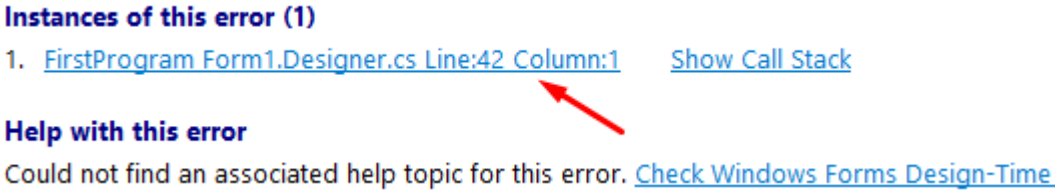
Если мы в коде удалим наше событие:



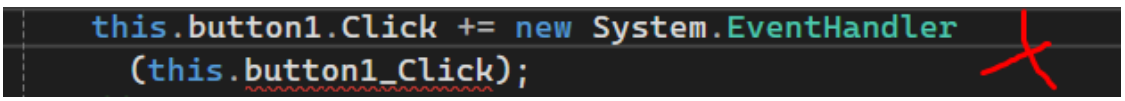
И вернёмся снова в **Designer**, то увидим ошибку:



Для того чтобы её исправить, нужно нажать на линию, куда указывает компилятор:



И удалить красную подчеркиваемую строку:

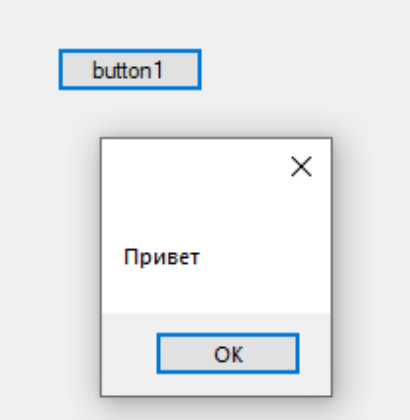


Теперь повторите предыдущие шаги, и самостоятельно добавьте клик на кнопку.

В методе **button1_Click** давайте добавим метод **MessageBox.Show**, которые будет выводить приветствие пользователя:

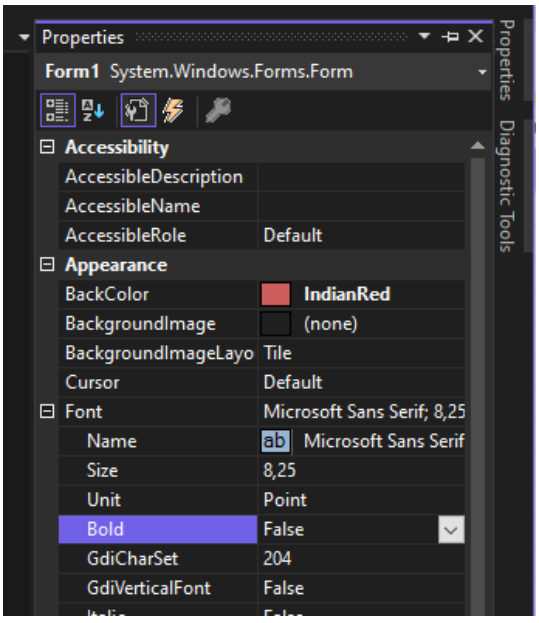
```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привет");
}
```

Теперь запустим проект, и мы увидим форму с кнопкой, на которую мы можем нажать и получить сообщение:

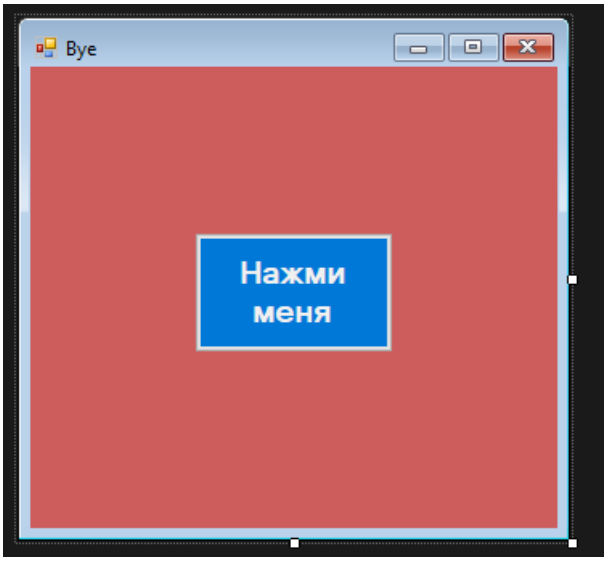


Стоит отметить, что графический дизайнер позволяет автоматически сгенерировать обработчик нажатия кнопки. Для этого надо в окне дизайнера нажать на кнопку на форме двойным щелчком мыши.

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

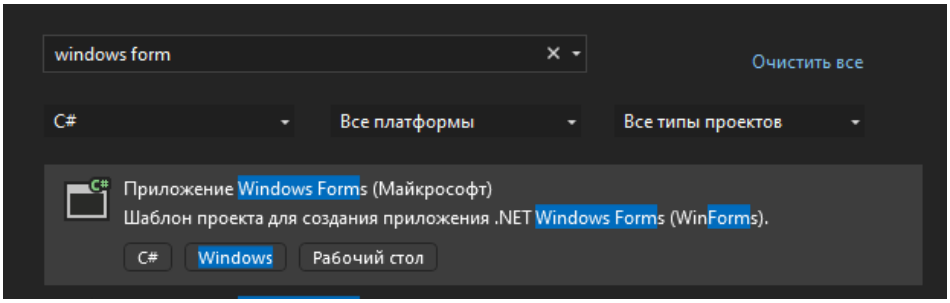


Ваша задача изменить фон и размер приложения, увеличить размер кнопки и поменять внутри неё текст следующим образом:



Приложение 1. Секундомер

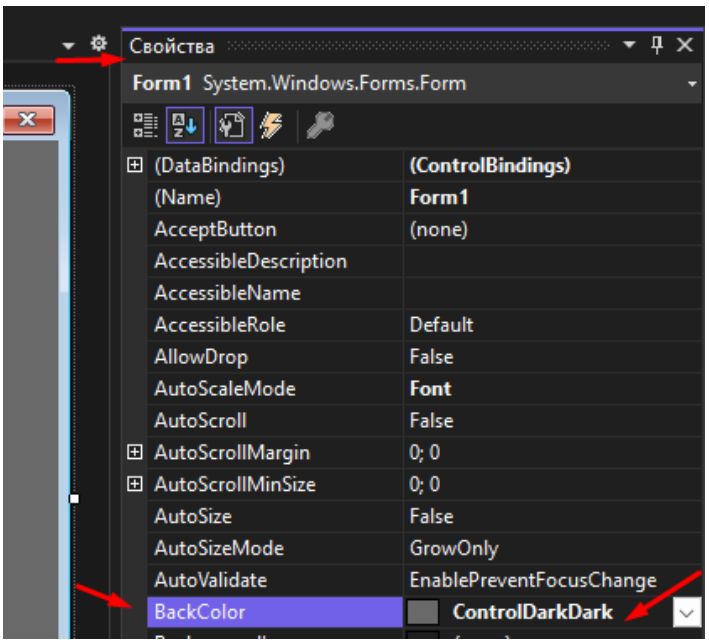
1. Создаём новый проект Приложение **Windows Forms** (.NET Framework).



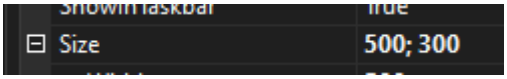
В качестве имени проекта указываем **Stopwatch**, местоположение проекта выбираем на свой вкус.

2. Выставим некоторые свойства для главной формы (если у вас нет этой вкладки нажмите **Alt+Enter** предварительно выбрав **Form1**):

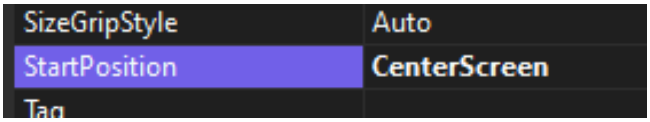
- Меняем **BackColor** на **ControlDarkDark**



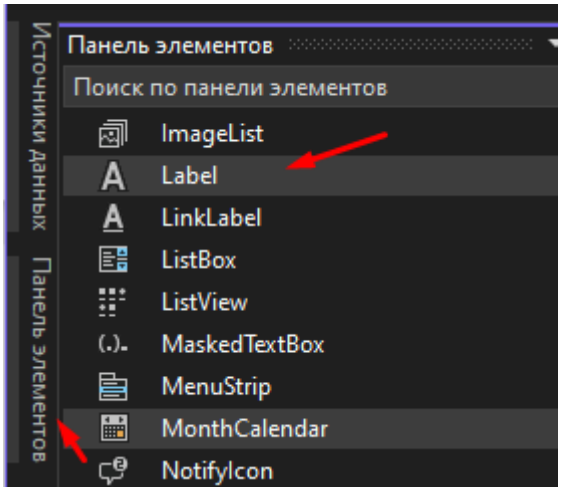
- Size на **500, 300**



- **StartPosition** на **CenterScreen**

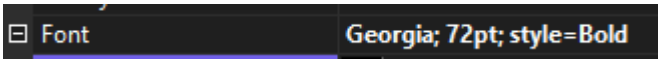


3. Добавьте **Label** из панели элементов на вашу форму



- Поменяйте у неё цвет переднего плана на **белый**

- Шрифт поменяйте на **Georgia, 48pt, Bold:**



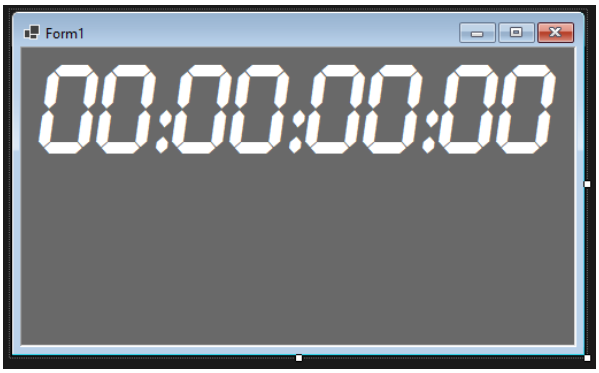
Также вы можете скачать и установить более подходящий шрифт LCD Mono:

https://fonts-online.ru/fonts/lcdmono/download

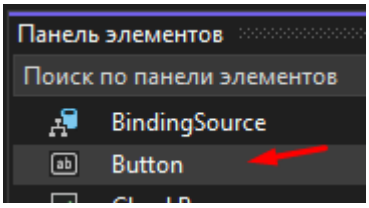
После его установки нужно будет перезапустить Visual Studio. В данном задании я буду использовать его.

- **Text** поменяйте на **00:00:00:00**

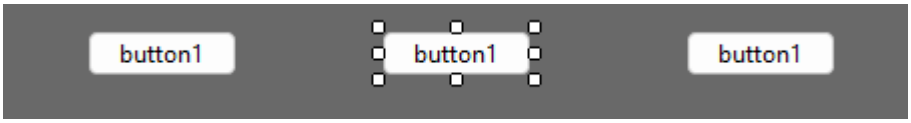
И расположите его сверху по центру



4. Добавьте кнопку **Button**:



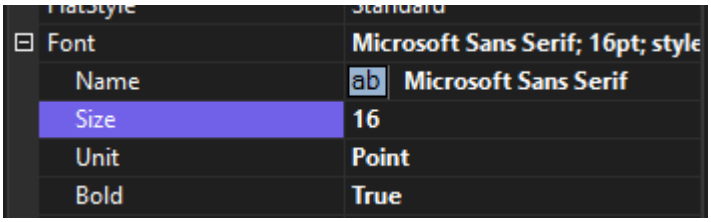
- Создайте её 3 копии:



Выделите все **3** кнопки через **Shift** и поменяйте размер на **150; 60**:

- Поменяйте у них цвет переднего плана на **белый**

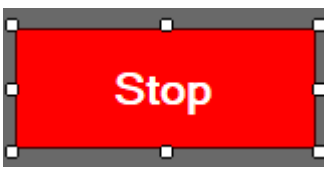
- Шрифт поменяйте на **Microsoft Sans Serif, 16pt, Bold:**



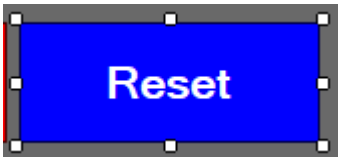
- **FlatStyle** на **Popup**

- У первой кнопки меняем **BackColor** на зелёный **YellowGreen** (не забудьте снять выделение с других кнопок) в **Text** пишем **Start**:

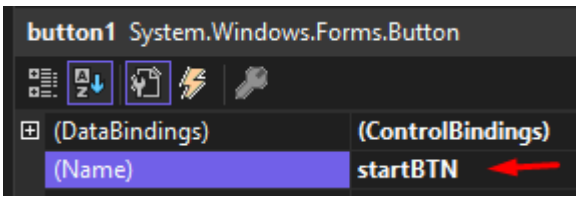
- У второй кнопки меняем **BackColor** на зелёный **Red** (не забудьте снять выделение с других кнопок) в **Text** пишем **Stop**:

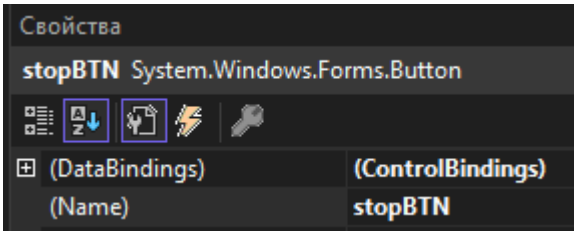


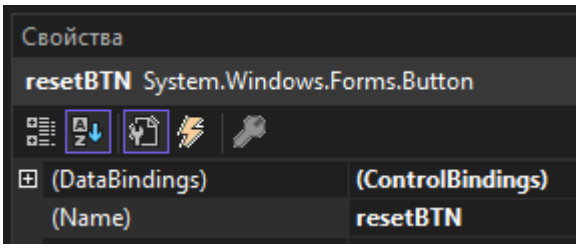
- У третьей кнопки меняем **BackColor** на зелёный **Blue** (не забудьте снять выделение с других кнопок) в **Text** пишем **Reset**:



Меняем у кнопок имена соответственно на **startBTN, stopBTN, resetBTN**:







5. Переходим к написанию кода. Щёлкаем два раза по нашей форме.

```
using System;
using System.Windows.Forms;

namespace Stopwatch
{
    // Объявляем частичный класс Stopwatch, который наследуется от
класса Form. Это означает, что наш класс будет представлять форму
Windows Forms.
    public partial class  Form1 : Form
    {
        //Объявляем переменные, которые будут использоваться в
нашем классе. Timer будет управлять временем, а h, m, s, ms будут
хранить часы, минуты, секунды и миллисекунды соответственно.
        System.Timers.Timer timer;
        int h, m, s, ms;
        // Конструктор класса. Вызывается при создании экземпляра
класса Stopwatch.
        public Form1()
        {
            // Инициализирует компоненты формы. Этот метод
автоматически создается дизайнером формы Visual Studio.
            InitializeComponent();
        }
```

```
        //Обработчик события, который вызывается при закрытии
формы.

        private void Form1_Load(object sender, EventArgs e)
        {
            timer = new System.Timers.Timer(); //Создает новый
экземпляр таймера.
            timer.Interval = 10; //Устанавливает интервал таймера в
10 миллисекунд.
            timer.Elapsed += OnTimeEvent; // : Подписываемся на
событие Tick таймера. Это означает, что метод OnTimerTick будет
вызываться каждый раз, когда таймер "тикает" (каждые 10
миллисекунд).
        }
        private void startBTN_Click(object sender, EventArgs e)
        {
            //Запускает таймер при нажатии кнопки "Start".
            timer.Start();
        }

        private void stopBTN_Click(object sender, EventArgs e)
        {
            // Останавливает таймер при нажатии кнопки "Stop".
            timer.Stop();
        }

        private void resetBTN_Click(object sender, EventArgs e)
        {
            //Сбрасывает таймер и значения времени при нажатии
кнопки "Reset".
            timer.Stop();
            h = 0;
            m = 0;
            s = 0;
            ms = 0;
            label1.Text = "00:00:00:00";
        }

        private void OnTimeEvent(object sender, EventArgs e)
        {
            ms += 15;
            if (ms >= 1000)
            {
                ms = 0;
                s += 1;
            }
```

```
            if (s >= 60)
            {
                s = 0;
                m += 1;
            }
            if (m >= 60)
            {
                m = 0;
                h += 1;
            }
            // Используется форматирование строк для обеспечения
отображения времени с двумя цифрами в каждом сегменте (например,
01:05:09:09).
            label1.Text = $"{h:D2}:{m:D2}:{s:D2}:{ms / 10:D2}"; //
:D2 - это форматный спецификатор. Он означает, что число будет
преобразовано в строку и дополнено нулями до длины 2 символа. Если
h меньше 10, то результатом будет строка "08", "09" и т.д.
        }
    }
}
```

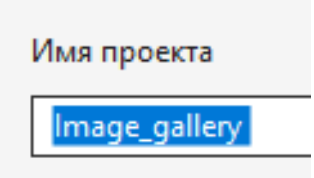
6. Запускам и проверяем работоспособность кода.

7. Сохраняем работу.

Приложение 2. Галерея изображений

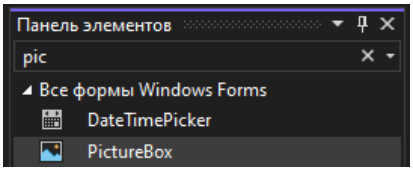
1. Скачаем в интернете несколько изображений, и поместим их в папку. Поменяйте их названия на более короткие, чтобы потом легче было работать с ними в коде.

2. Создаём новое приложение WF.

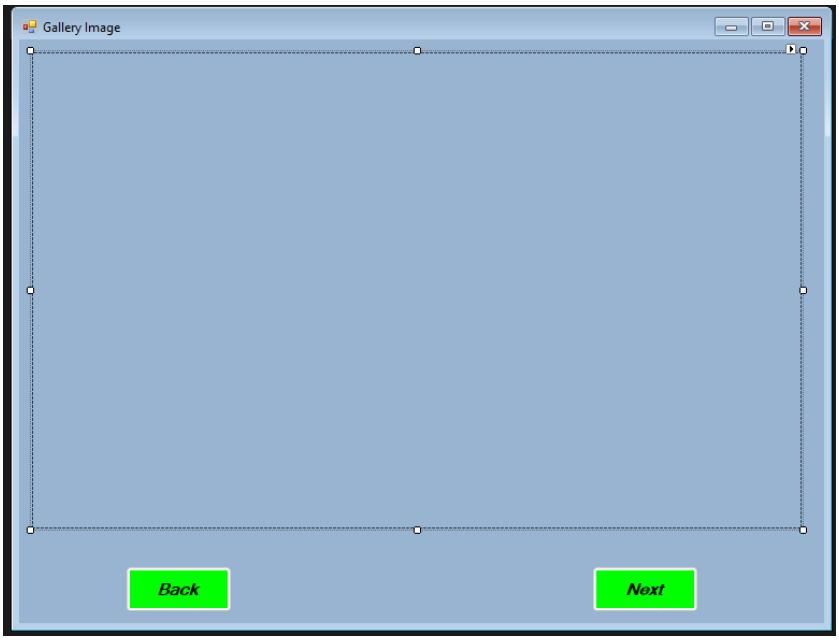


Добавляем две кнопки, и настраиваем внешний вид:

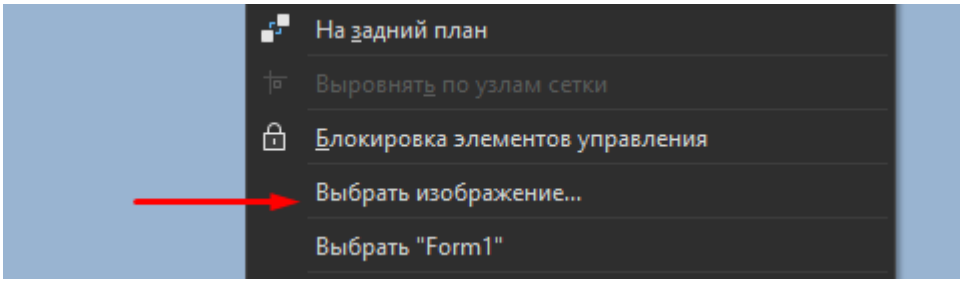
3. Добавляем **PictureBox**:



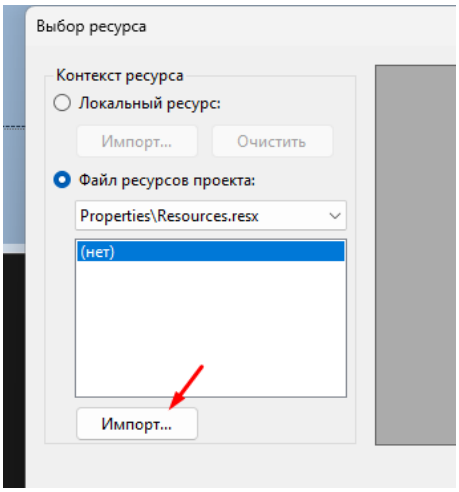
4. Увеличьте его размер:



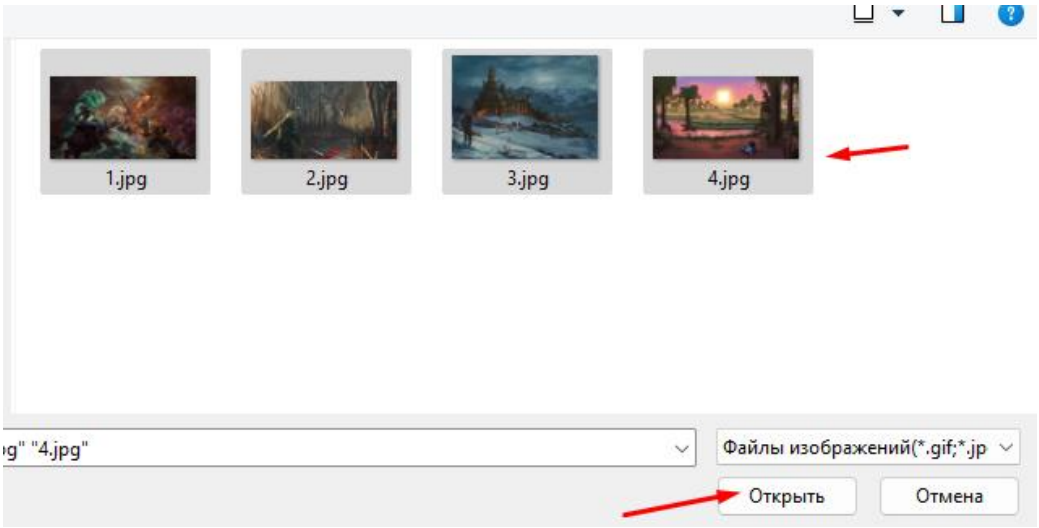
5. Щёлкните правой кнопкой мыши по **PictureBox** и выберите пункт «**Выбрать изображение**»:



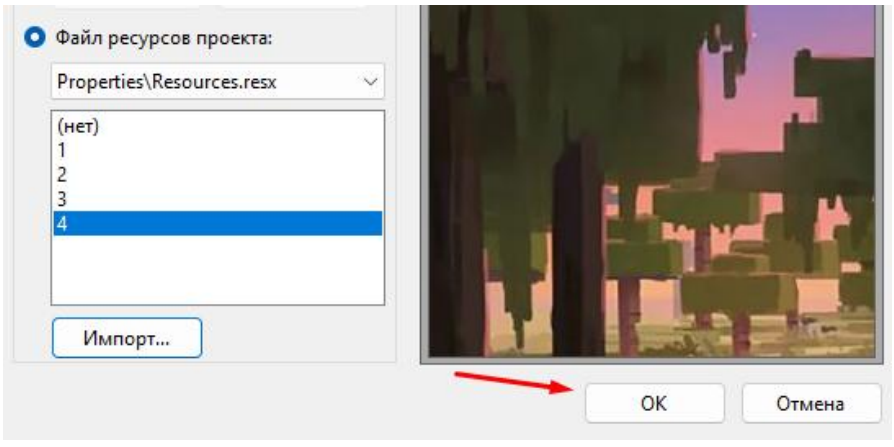
Импорт:



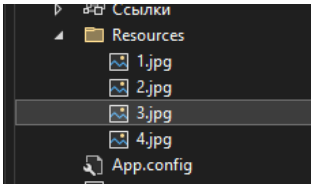
Выделяем все наши изображения и нажимаем **открыть**:



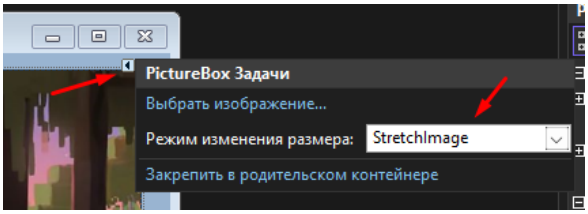
Видим, что наши изображения появились, и жмём **ОК**:



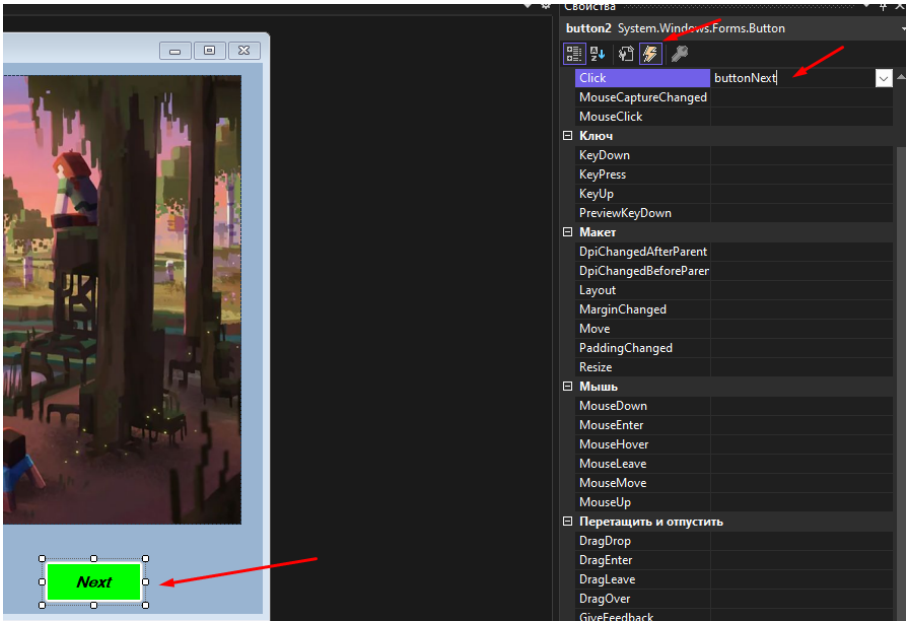
Они также отобразились в **Обозревателе решений:**



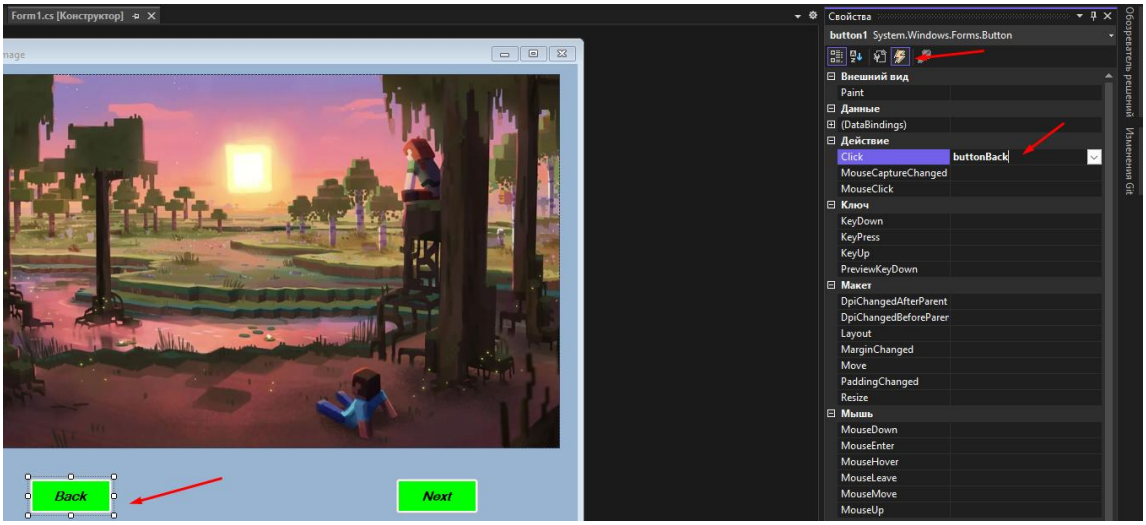
Чтобы растянуть изображение на наш холст, нужно нажать на стрелочку и выбрать один из режимов (например **StretchImage** или **Zoom**):



6. Нажимаем на кнопку, которая будет отвечать за переход к следующему изображению. Во вкладке события для клика назначим имя **buttonNext**:



Для кнопки назад сделаем то же самое, но название **buttonBack:**



Пропишем код:

```
namespace Image_gallery // Определяет пространство имен для
организации кода
{
    public partial class Form1 : Form // Объявление частичного
класса Form1, который наследует от Form. Это форма Windows Forms.
    {
        int i = 1; // Переменная i используется для отслеживания
текущего изображения. Изначально установлена в 1.
        public Form1() // Конструктор класса. Вызывает метод
InitializeComponent, который инициализирует компоненты формы.
        {
            InitializeComponent();
        }

        private void buttonNext(object sender, EventArgs e) //
Обработчик события, вызываемый при нажатии кнопки "Next".
        {
            i++; // Увеличивает значение i на 1.

            if (i > 5) // Если значение i превышает 5, оно
сбрасывается до 1.
                i = 1;

            changeImage(i); // Вызывает метод changeImage для смены
изображения.
        }

        private void buttonBack(object sender, EventArgs e) //
Обработчик события, вызываемый при нажатии кнопки "Back".
        {
            i--; // Уменьшает значение i на 1.

            if (i < 1) // Если значение i меньше 1, оно
устанавливается в 5.
                i = 5;

            changeImage(i); // Вызывает метод changeImage для смены
изображения.
        }

        private void changeImage(int number) // Метод для смены
изображения.
        {
            switch(number)  // Переключатель, который в зависимости
от значения number устанавливает соответствующее изображение из
ресурсов в pictureBox1.
```

```
            {
                case 1:
                    pictureBox1.Image = Properties.Resources._1;
                    break;
                case 2:
                    pictureBox1.Image = Properties.Resources._2;
                    break;
                case 3:
                    pictureBox1.Image = Properties.Resources._3;
                    break;
                case 4:
                    pictureBox1.Image = Properties.Resources._4;
                    break;
            }
        }
    }
}
```

7. Произведём рефакторинг кода.

7.1. Используем массив изображений, чтобы упростить управление изображениями.

7.2. Избежим дублирования кода для методов **buttonNext** и **buttonBack**.

```
namespace Image_gallery
{
    public partial class Form1 : Form
    {
        int i = 0;
        private readonly Image[] images; // Используем массив
images, чтобы хранить все изображения. Это позволяет легко
добавлять и управлять изображениями.

/*readonly - Поля для чтения представляют такие поля класса или
структуры, значение которых нельзя изменить. Таким полям можно
присвоить значение либо при непосредственно при их объявлении, либо
в конструкторе. В других местах программы присваивать значение
таким полям нельзя, можно только считывать их значение.*/

        public Form1()
        {
            InitializeComponent();
            images = new Image[]
            {
                Properties.Resources._1,
                Properties.Resources._2,
                Properties.Resources._3,
                Properties.Resources._4
```

```
            };

            changeImage(i);
        }

        private void buttonNext(object sender, EventArgs e) //
Упрощены, чтобы использовать единый метод ChangeImageIndex для
изменения индекса изображения.
        {
            ChangeImageIndex(1); // Когда вызывается метод
ChangeImageIndex с параметром 1, это указывает на то, что мы хотим
перейти к следующему изображению. Действие: Прибавление 1 к
текущему индексу i, чтобы перейти к следующему изображению в
массиве.
        }

        private void buttonBack(object sender, EventArgs e) //
Упрощены, чтобы использовать единый метод ChangeImageIndex для
изменения индекса изображения.
        {
            ChangeImageIndex(-1); // Когда вызывается метод
ChangeImageIndex с параметром -1, это указывает на то, что мы хотим
вернуться к предыдущему изображению. Вычитание 1 из текущего
индекса i, чтобы перейти к предыдущему изображению в массиве.
        }

        private void ChangeImageIndex(int delta) // Этот метод
отвечает за изменение текущего индекса изображения и обновление
изображения в PictureBox.
        //delta: Это смещение, которое указывает, насколько нужно
изменить текущий индекс i. Может быть положительным (для перехода к
следующему изображению) или отрицательным (для перехода к
предыдущему изображению).
        {
            i = (i + delta + images.Length) % images.Length;
            // i + delta: Сначала мы добавляем смещение delta к
текущему индексу i.
            // i + delta + images.Length: Затем добавляем длину
массива images.Length. Это делается для того, чтобы избежать
отрицательных значений индекса, когда delta отрицательное.
Например, если i равно 0 и delta равно -1, то без этого добавления
результат был бы -1, что вне допустимого диапазона индексов
массива.
            // (i + delta + images.Length) % images.Length:
Используем операцию модуля (%), чтобы результат всегда оставался в
пределах допустимых значений индекса массива. То есть, если
```

```
результат больше длины массива, он будет корректно обрезан до
допустимого значения. Это также позволяет "заворачивать" индексы,
когда они выходят за пределы допустимых значений. Например, если
текущий индекс был последним и мы прибавили 1, то результат
обрежется до 0, что вернёт нас к первому изображению в массиве.
            changeImage(i); // После вычисления нового индекса,
вызывается метод changeImage с обновленным значением i, чтобы
обновить изображение в PictureBox.
        }

        private void changeImage(int index) // Теперь использует
индекс массива для установки изображения.
        {
            pictureBox1.Image = images[index];
        }
    }
}
```

Самостоятельные задания

Напишете простую игру для набора слов, где пользователю предоставляется случайное слово для набора. При нажатии клавиши **Enter** ввод проверяется, и обновляется количество правильных и неправильных ответов, после чего пользователю предоставляется новое слово.

Пример реализации: