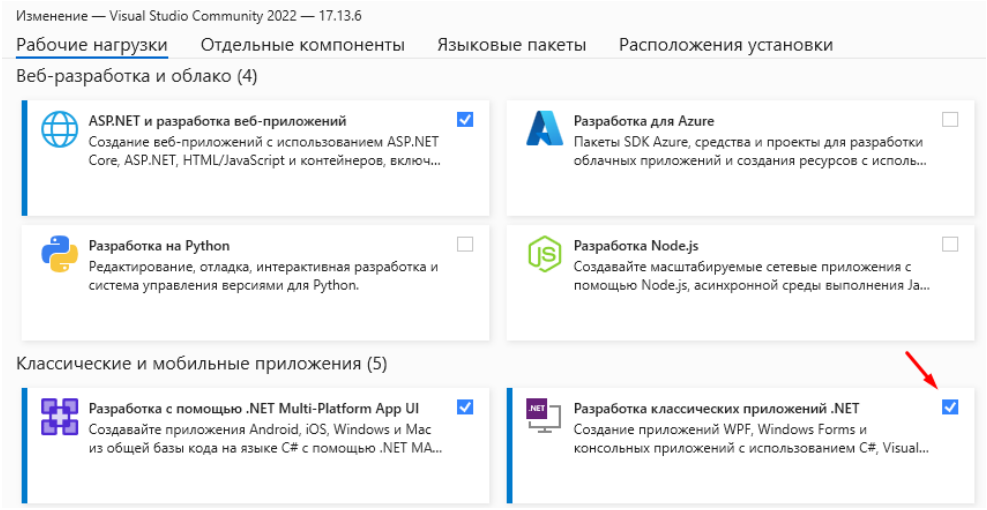
Лабораторная работа №11. Введение и разработка приложений в Windows Forms

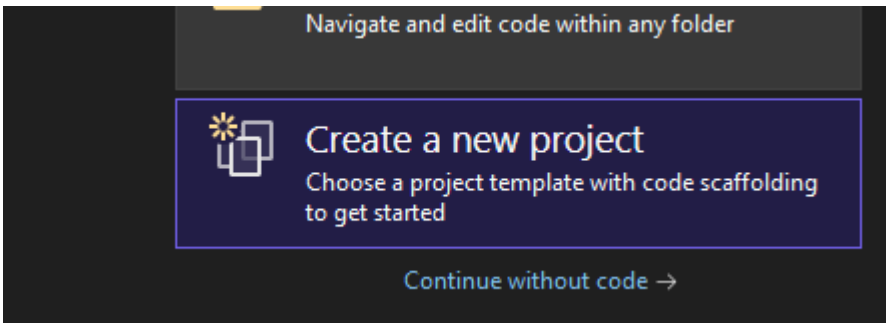
Для создания графических интерфейсов с помощью платформы **.NET** применяются разные технологии – **Windows Forms**, **WPF**, **UWP**. Однако наиболее простой и удобной платформой до сих пор остается **Windows Forms** или сокращенно **WinForms**. В данной лабораторной работе даются основы создания графических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

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

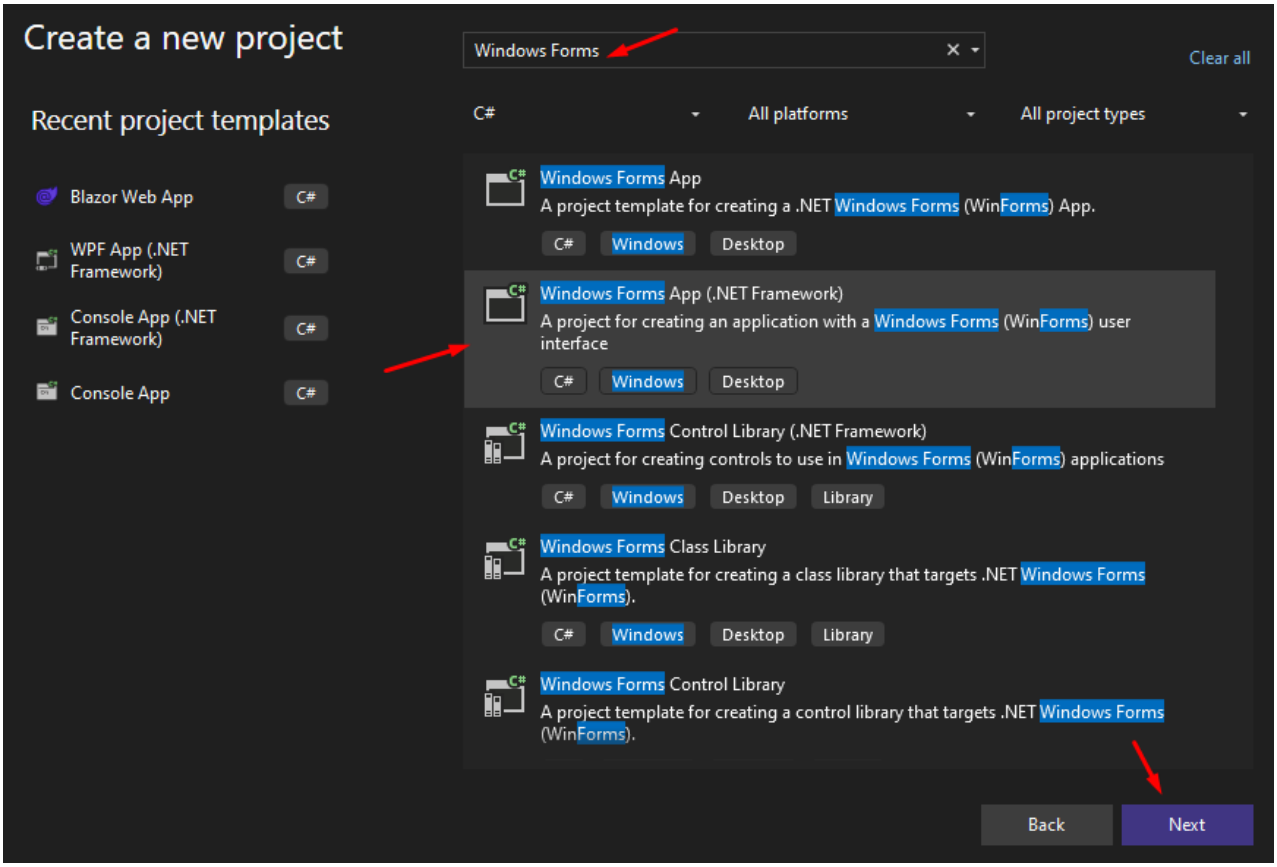
Для работы с **Windows Forms** в **Visual Studio 2022** необходимо наличие компонента ***"Разработка классических приложений .NET"*** в установщике.



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

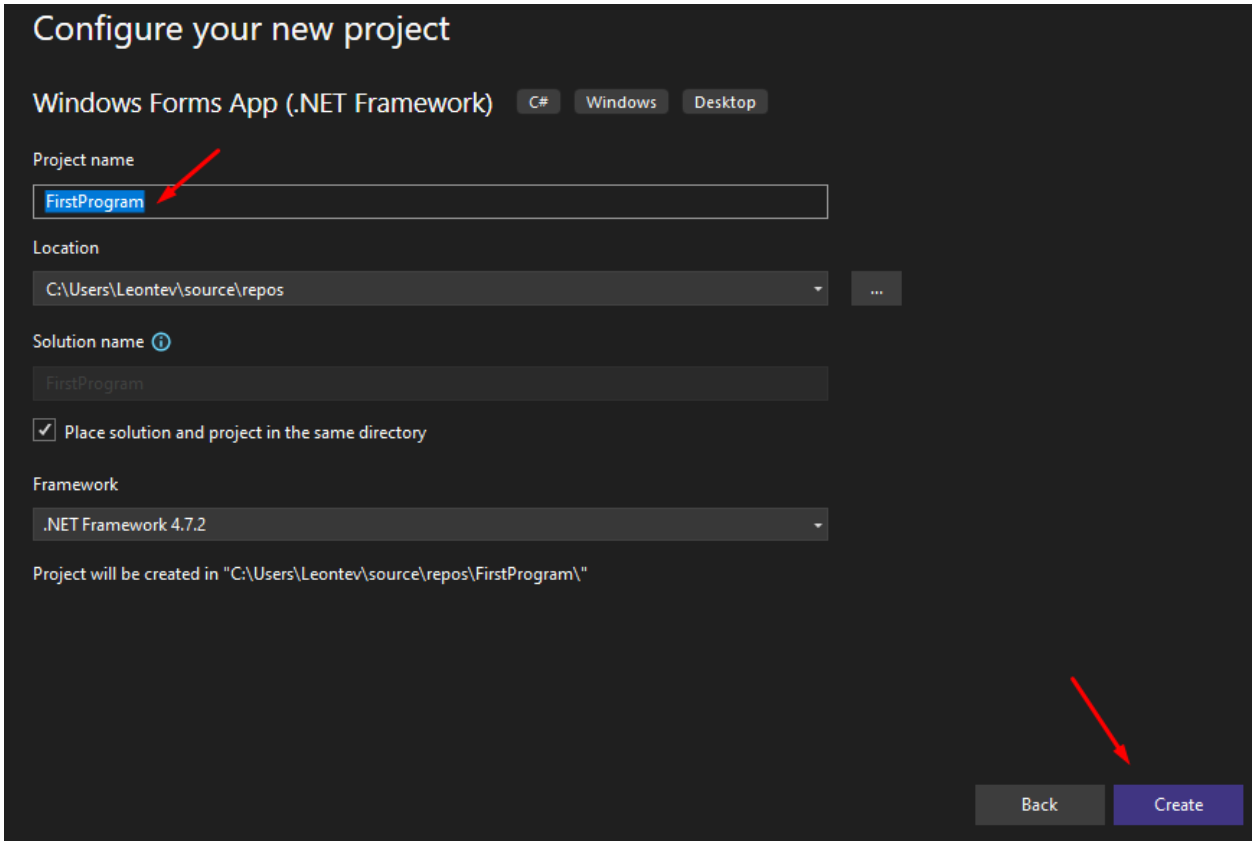


В строке поиска введите "**Windows Forms**". Выберите шаблон **Windows Forms App (.NET Framework)** и нажмите **Next**:

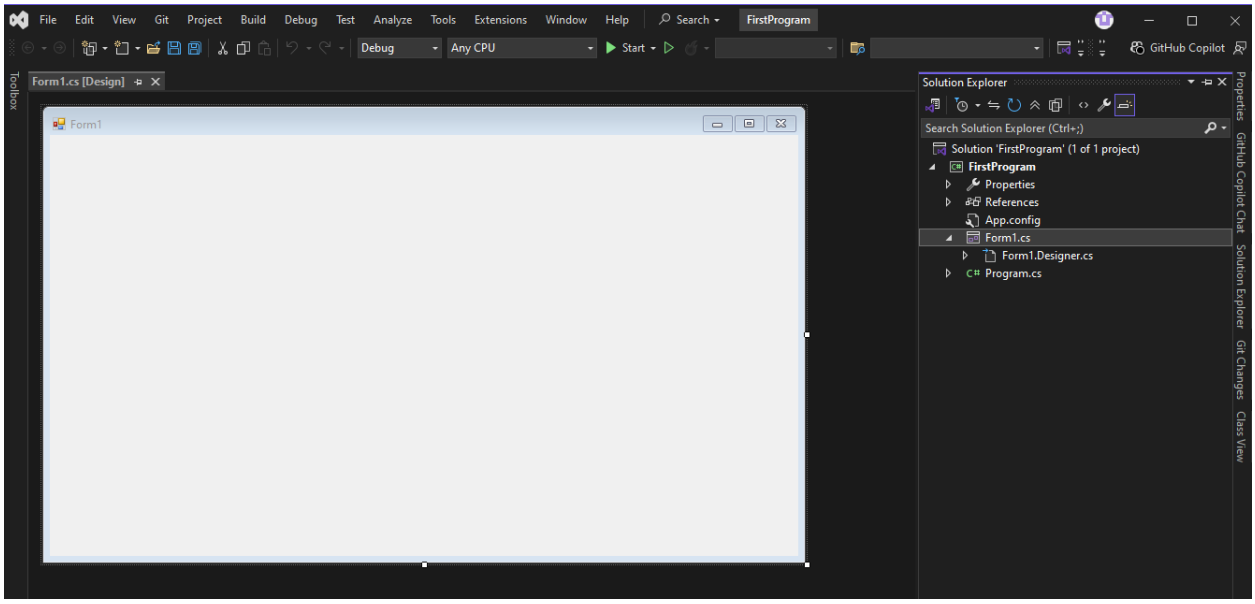


Примечание: Несмотря на использование устаревшего фреймворка, этот вариант обеспечивает совместимость со старыми версиями ОС.

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



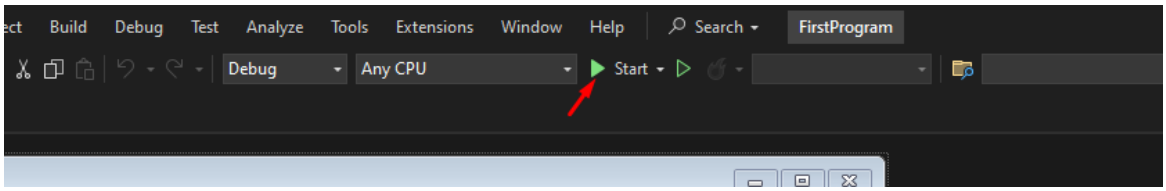
После создания откроется рабочая среда с файлами проекта:



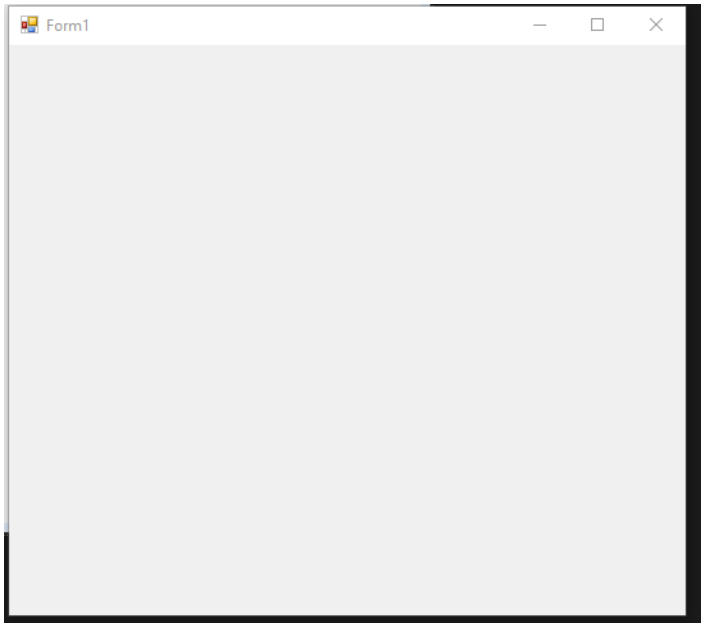
Справа в окне **Solution Explorer** можно увидеть структуру нашего проекта:

- ***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**:



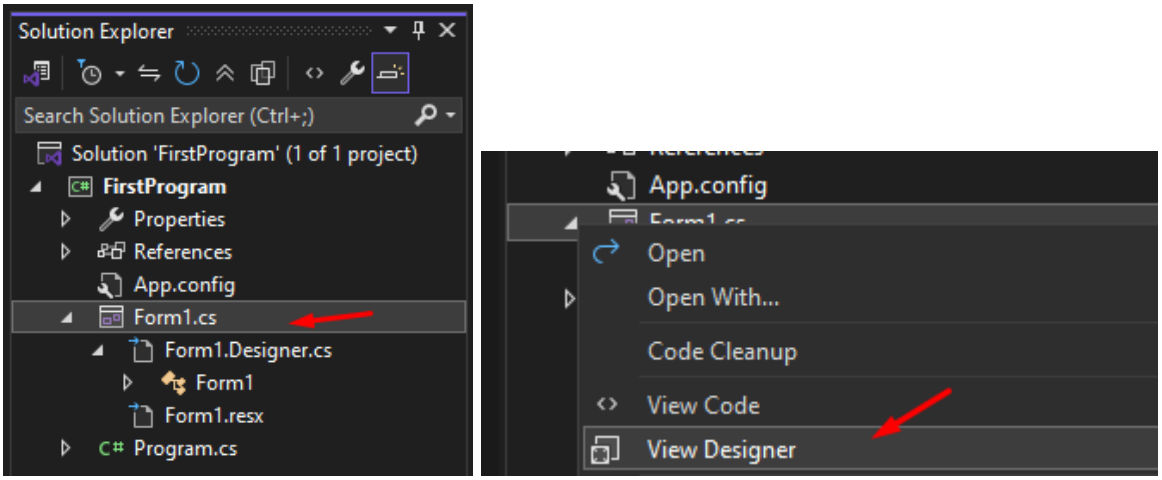
Откроется пустая форма **Form1**:



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.

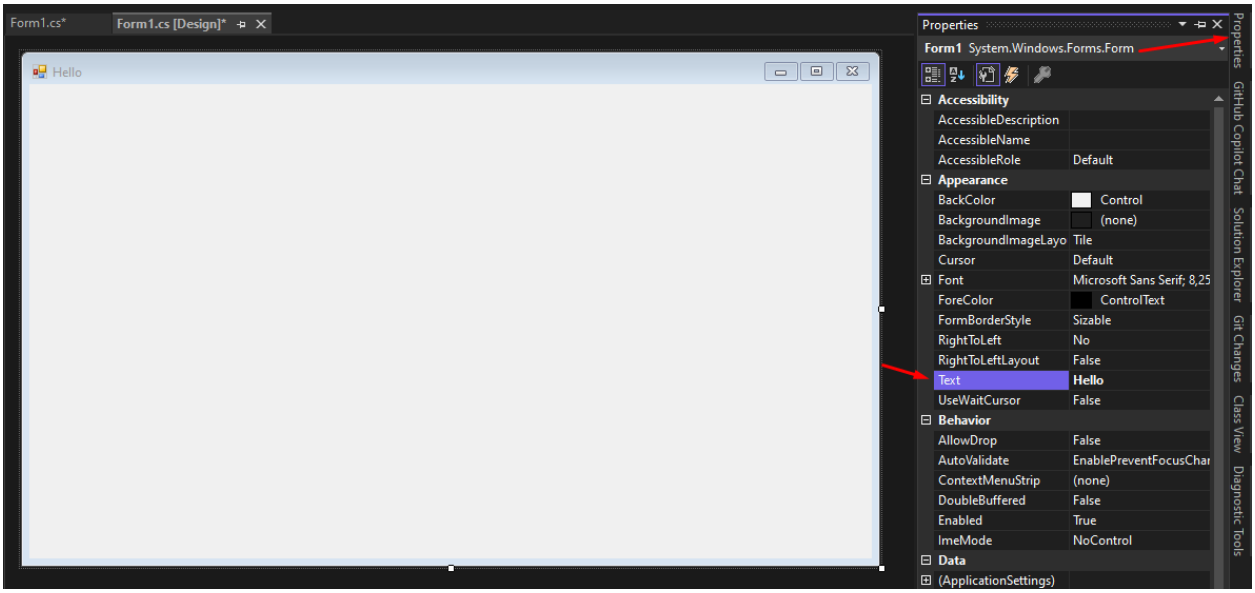
Для открытия формы в режиме графического дизайнера нажмем в структуре проекта на файл **Form1.cs** либо левой кнопкой мыши двойным кликом, либо правой кнопкой мыши и в появившемся контекстном меню выберем **View Designer** (также можно использовать комбинацию клавиш **Shift+F7**):



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

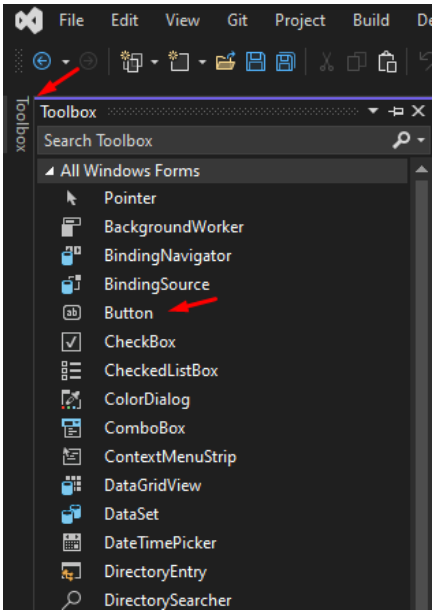
При выборе формы в окне дизайнера внизу справа под структурой проекта мы сможем найти окно **Properties**. Так как у меня в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

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

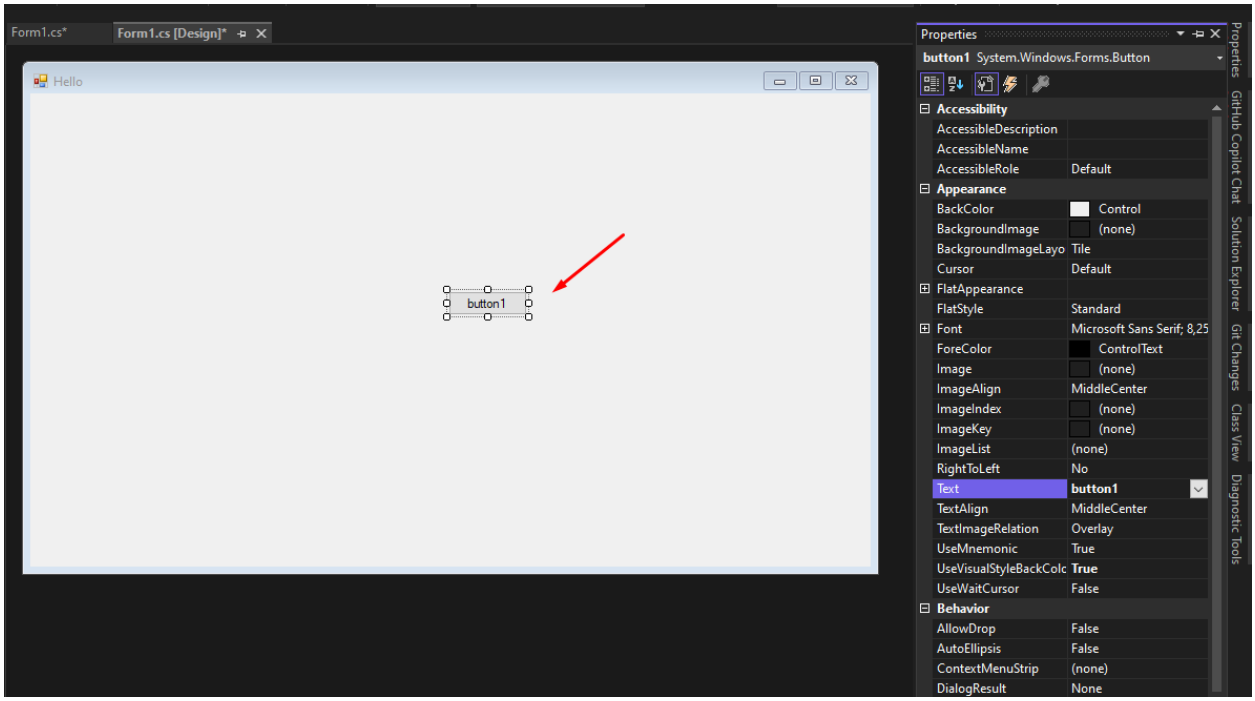


Таким образом мы поменяли заголовок формы. И подобным образом мы можем поменять другие свойства формы, которые доступны в окне свойств.

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:

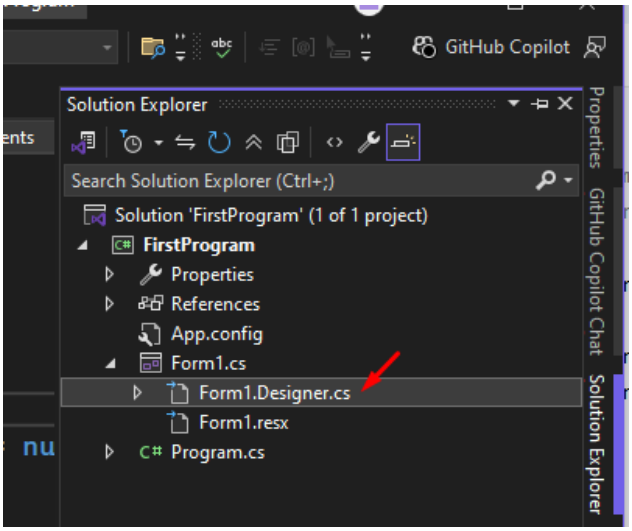


Найдем среди элементов **кнопку** и, захватив ее указателем мыши, перенесем на форму:



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:**



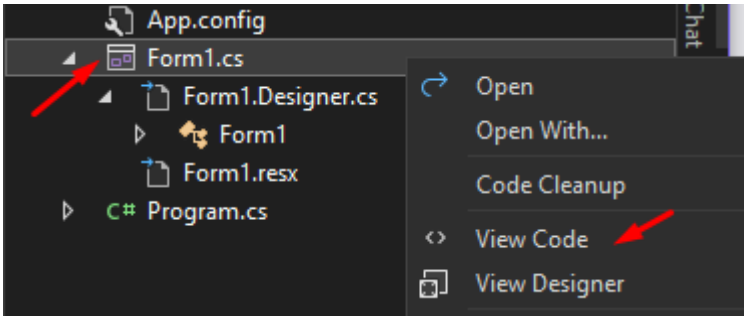
Мы увидим, что в класс **Form1** была добавлена переменная **button1** типа **Button** и для этой переменной, как и для объекта формы, задан ряд свойств:

```
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
```

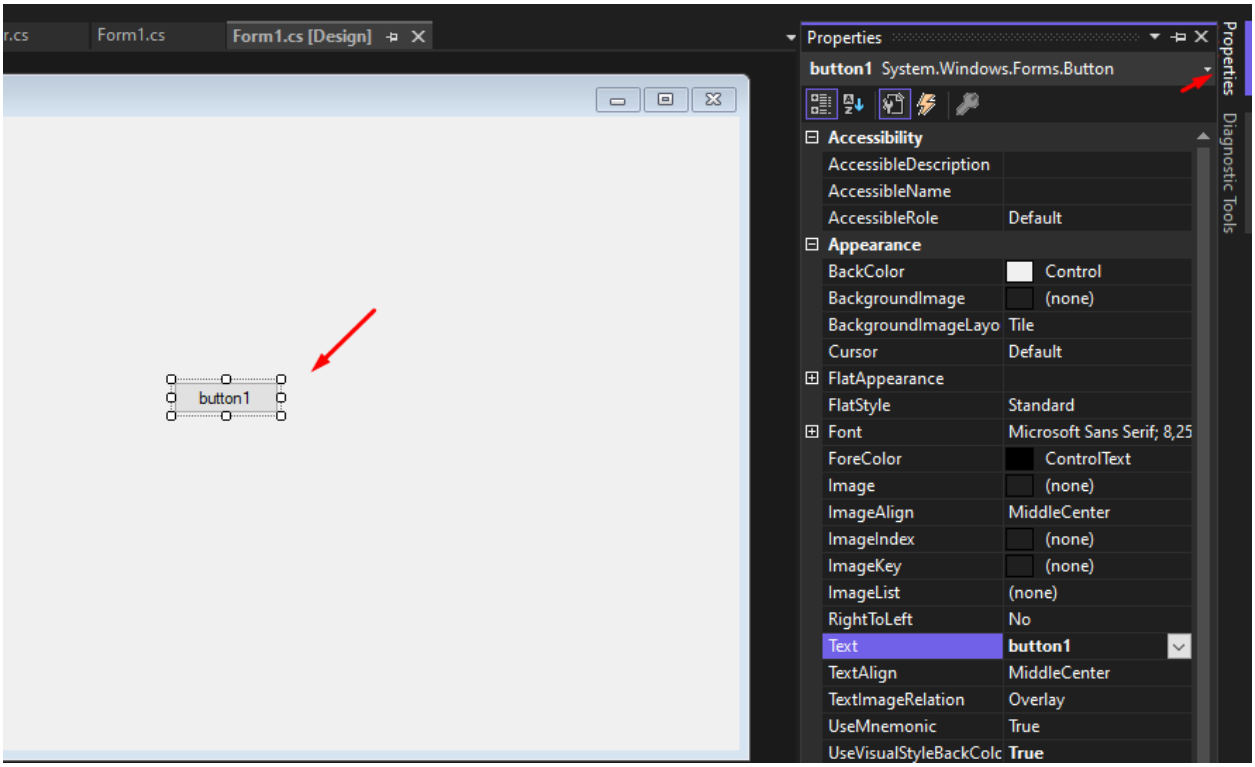
Это визуальная часть. Теперь приступим к самому программированию. Добавим простейший код на языке C#, который бы выводил сообщение по нажатию кнопки. Для этого перейдем в файл кода **Form1.cs**, который связан с этой формой:



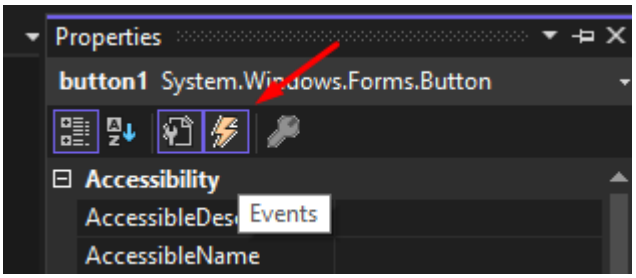
По умолчанию после создания проекта он имеет код:

```
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
```

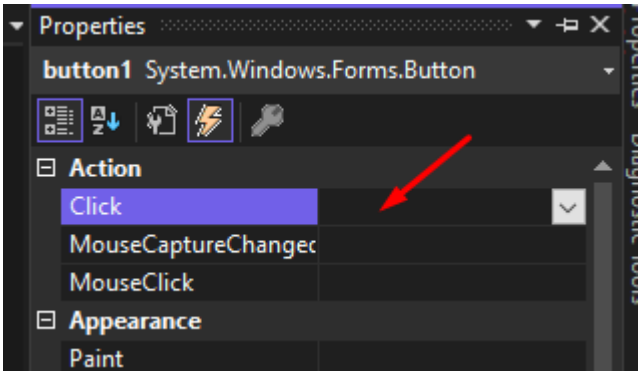
Снова вернёмся в окно **Designer**, выделим кнопку и перейдём в **Propertis**:



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



И щёлкнем два раза по поле **Click**:

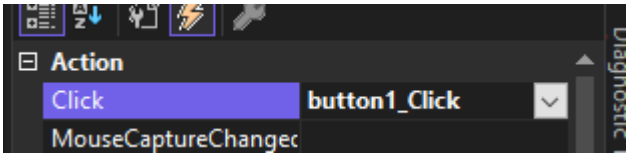


У вас сразу откроется код, куда будет вставлен обработчик событий:

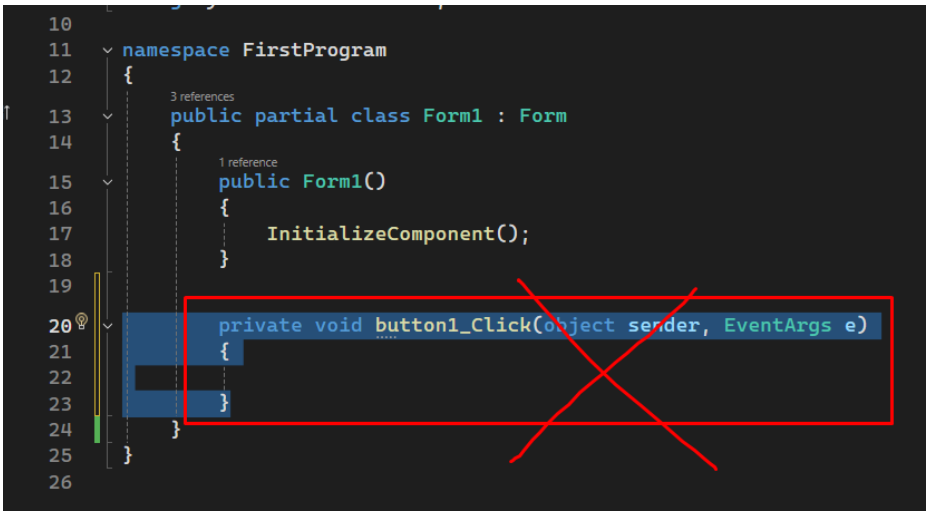
```
1 reference
private void button1_Click(object sender, EventArgs e)
{

}
```

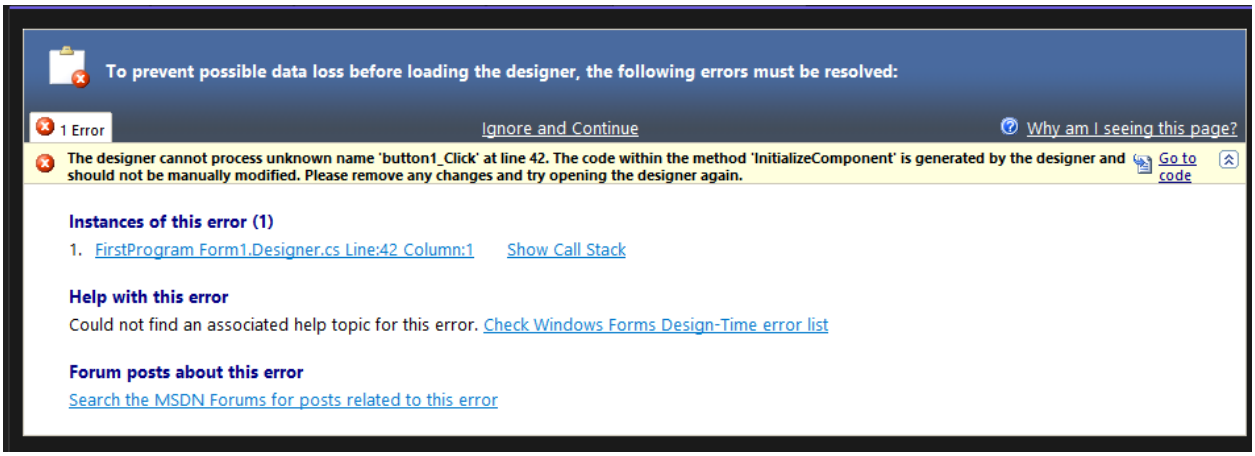
Но если мы вернёмся в **Designer**, то увидим, что на клик повесилось событие:



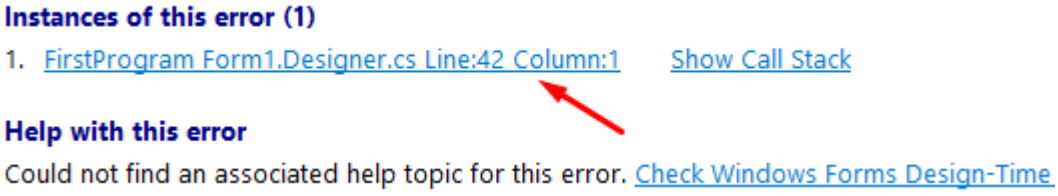
Если мы в коде удалим наше событие:



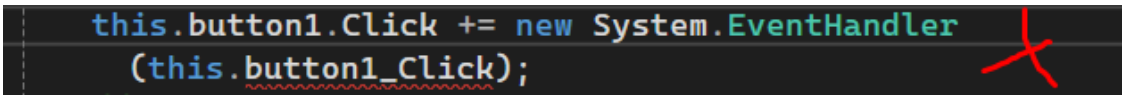
И вернёмся снова в **Designer**, то увидим ошибку:



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



И удалить красную подчеркиваемую строку:

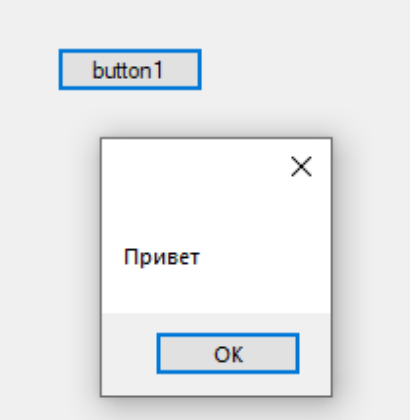


Теперь повторите предыдущие шаги, и самостоятельно добавьте клик на кнопку.

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

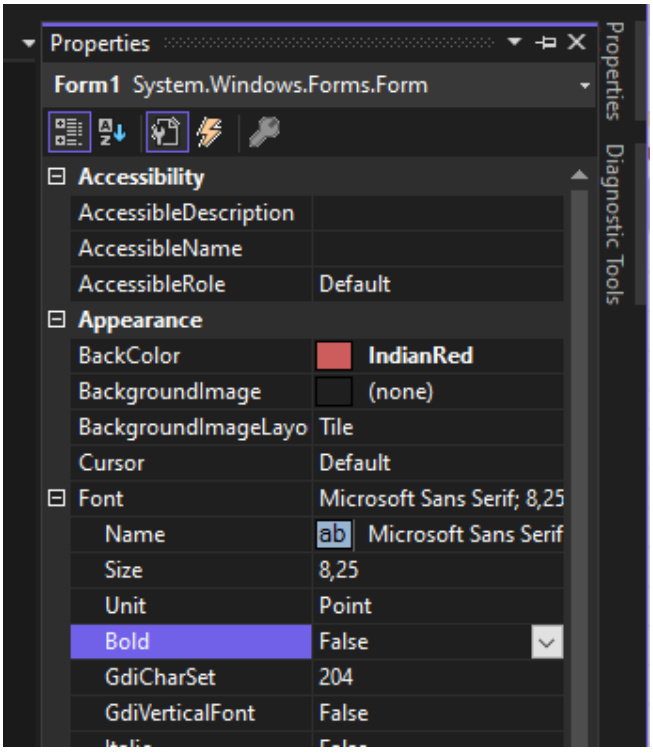
```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привет");
}
```

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

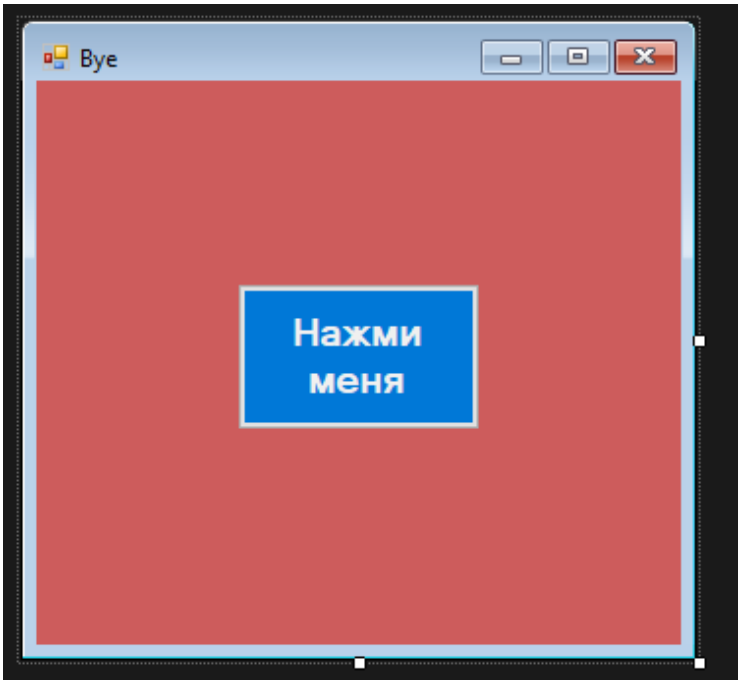


Стоит отметить, что графический дизайнер позволяет автоматически сгенерировать обработчик нажатия кнопки. Для этого надо в окне дизайнера нажать на кнопку на форме двойным щелчком мыши.

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:



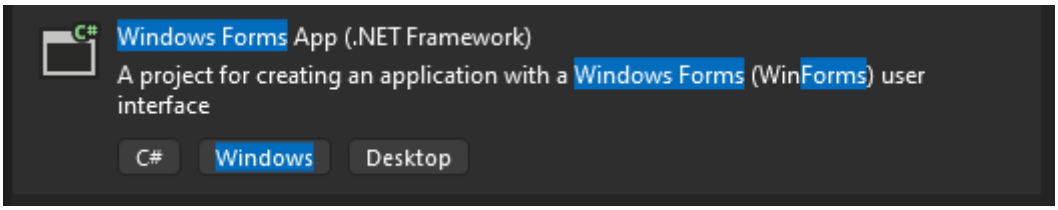
Приложение 1. Секундомер

Это простое приложение-секундомер, созданное на платформе .NET Framework с использованием Windows Forms. Оно позволяет:

- Запускать отсчет времени (Start)
- Останавливать отсчет (Stop)
- Сбрасывать показания (Reset)

1. Создание проекта

- Создаем новый проект "**Windows Forms App (.NET Framework)**"

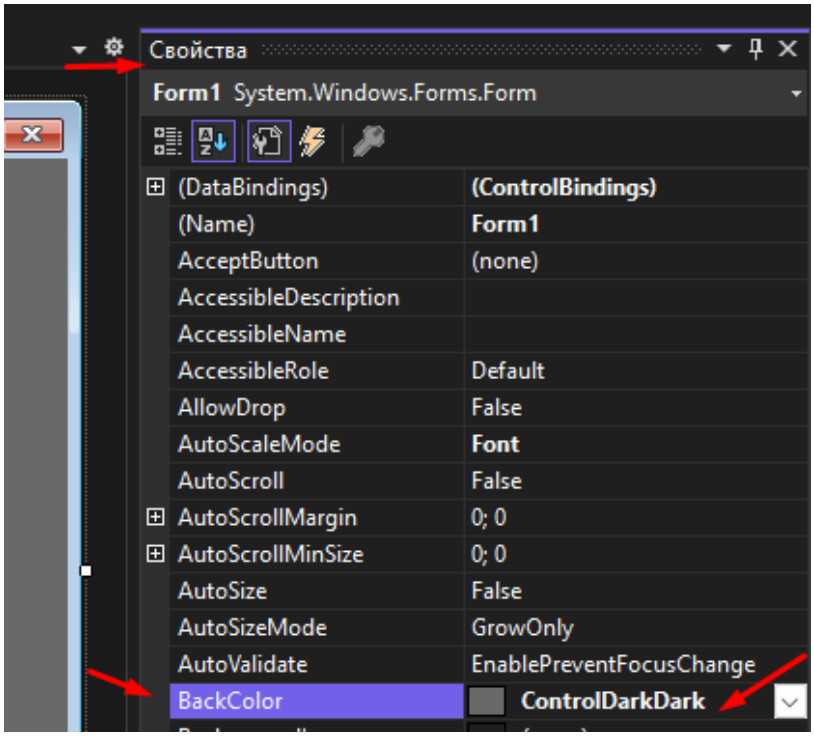


- Называем проект "**Stopwatch**"

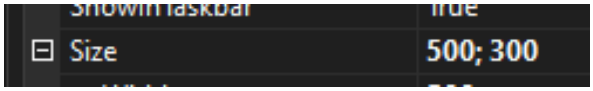
2. Настройка главной формы

Устанавливаем следующие свойства формы:

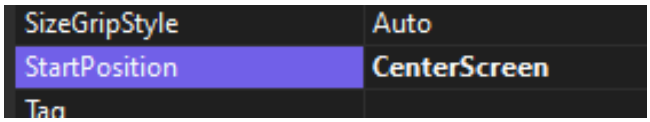
- **BackColor: ControlDarkDark (темный фон)**



- **Size: 500, 300 (размер окна)**

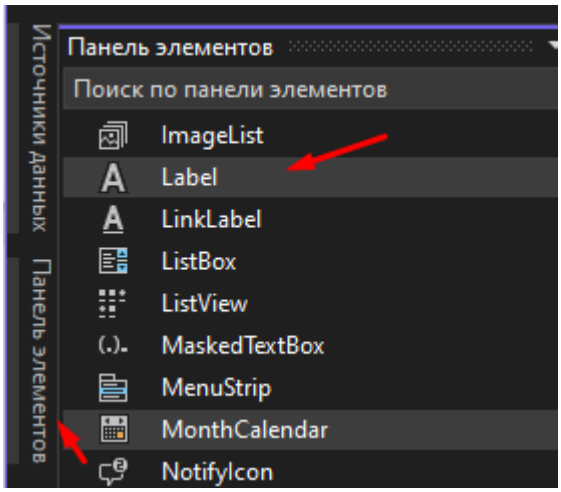


- **StartPosition: CenterScreen (окно появится по центру экрана)**



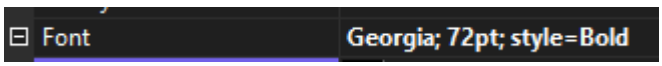
3. Добавление и настройка Label

Добавляем элемент **Label** для отображения времени:



- **ForeColor: White (белый текст)**

- **Font: Georgia, 48pt, Bold (или LCD Mono шрифт)**

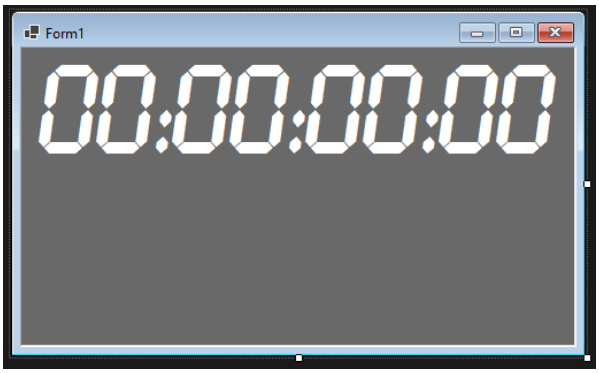


- **Text: "00:00:00:00" (начальное значение)**

- Также вы можете скачать и установить более подходящий шрифт LCD Mono: https://fonts-online.ru/fonts/lcdmono/download

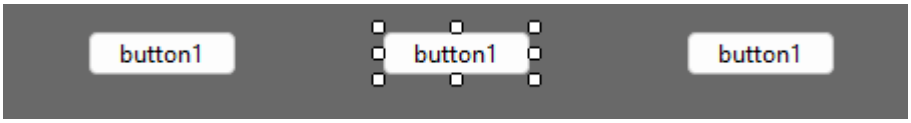
  После его установки нужно будет перезапустить Visual Studio.

- Размещаем по центру в верхней части формы



4. Добавление и настройка кнопок

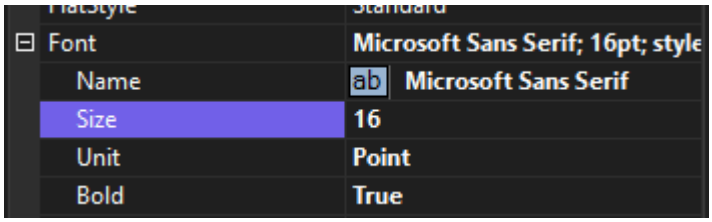
Создаем три кнопки с общими настройками:



- **Размер: 150x60**

- **ForeColor: White**

- **Font: Microsoft Sans Serif, 16pt, Bold**

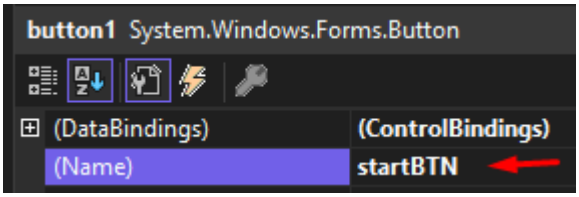


- **FlatStyle: Popup**

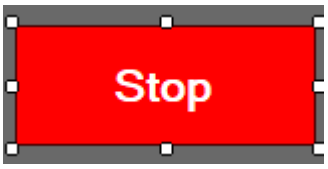
Индивидуальные настройки кнопок:

1. **Кнопка Start:**
   - **BackColor: YellowGreen**

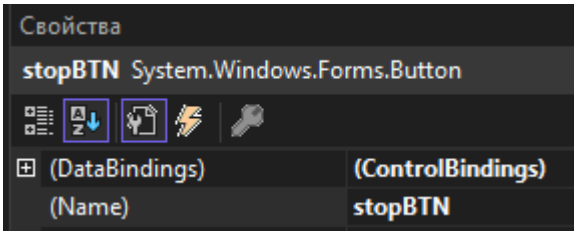
- **Text: "Start"**
- **Имя: startBTN**



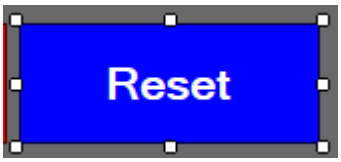
2. **Кнопка Stop:**
   - **BackColor: Red**



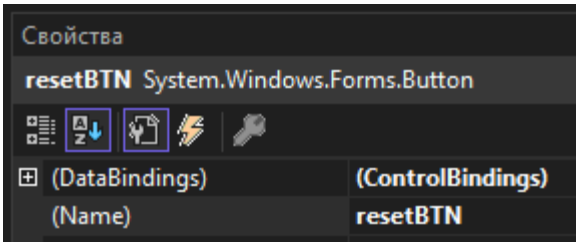
   - **Text: "Stop"**
   - **Имя: stopBTN**



3. **Кнопка Reset:**
   - **BackColor: Blue**

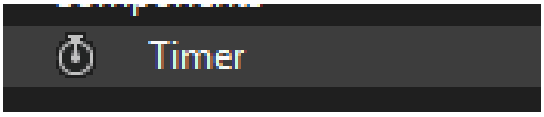


   - **Text: "Reset"**
   - **Имя: resetBTN**



5. Добавление таймера

Добавляем элемент **Timer** из панели элементов.



6. Далее добавим **обработчики событий** для **кнопок** и **таймера**, самый простой способ — это дважды нажать по ним:

```
private void startBTN_Click(object sender, EventArgs e) {

}

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

}

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

}

1 reference
private void timer1_Tick_1(object sender, EventArgs e) {

}
```

7. Теперь перейдём к написанию кода.

7.1. Пространства имён

Удаляем ненужные пространства имён, чтобы код был чище и не загружал проект лишними зависимостями.

Оставляем только:

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

- **System** — базовое пространство имён, без него нельзя работать с основными типами данных (int, string, DateTime и т. д.).
- **System.Windows.Forms** — нужно для работы с элементами WinForms (формы, кнопки, метки, таймеры и т. д.).

7.2. Локальные переменные

Далее создадим локальные переменные:

```
namespace Windows_Forms_Lab_11 {
    3 references
    public partial class Form1 : Form {
        private int hours = 0;
        private int minutes = 0;
        private int seconds = 0;
        private int milliseconds = 0;
```

Эти переменные нужны, чтобы хранить **состояние секундомера**:

- **hours** — количество прошедших часов;
- **minutes** — количество минут;
- **seconds** — количество секунд;
- **milliseconds** — доли секунды (условные миллисекунды, чтобы выводить красивый формат).

7.3. Настройка таймера в конструкторе

В главной форме запустим обновление таймера на 10

```
public Form1() {
    InitializeComponent();
    timer1.Interval = 10;
}
```

- timer1.Interval = 10; — указывает, что таймер будет "тикать" каждые **10 миллисекунд**.

Это значит, что событие timer1_Tick_1 будет вызываться **100 раз в секунду**. Сделано так для большей точности секундомера.

7.4. Кнопки управления

Теперь реализуем самые простые кнопки – старта и остановки:

```
private void startBTN_Click(object sender, EventArgs e) {
    timer1.Start();
}
```

- Запускает таймер → начинается отсчёт времени.

```
private void stopBTN_Click(object sender, EventArgs e) {
    timer1.Stop();
}
```

- Останавливает таймер → время перестаёт идти, но текущее значение остаётся.

Теперь реализуем кнопку сброса:

```
private void resetBTN_Click(object sender, EventArgs e) {
    timer1.Stop();
    hours = minutes = seconds = milliseconds = 0;
    label1.Text = "00:00:00:00";
}
```

- Останавливает таймер,
- обнуляет все переменные,
- сразу обновляет текст метки (label1), чтобы пользователь видел нули.

7.5. Событие таймера

Теперь перейдём к реализации таймера:

```
private void timer1_Tick(object sender, EventArgs e) {
    milliseconds += 15;
    if (milliseconds >= 1000) {
        milliseconds = 0;
        seconds++;
    }
    if (seconds >= 60) {
        seconds = 0;
        minutes++;
    }
    if (minutes >= 60) {
        minutes = 0;
        hours++;
    }

    label1.Text = string.Format("{0:00}:{1:00}:{2:00}:{3:00}",
hours, minutes, seconds, milliseconds);
}
```

Что тут происходит:

1. Каждый тик (раз в 10 мс) увеличиваем milliseconds на **15**. Это небольшая "поправка", потому что WinForms таймер не идеально точный, и таким образом компенсируется задержка (он реально срабатывает чуть реже).
2. Проверяем переполнение:
    - если миллисекунд ≥ 1000 → сбрасываем и прибавляем секунду,
    - если секунд ≥ 60 → сбрасываем и прибавляем минуту,
    - если минут ≥ 60 → сбрасываем и прибавляем час.

3. Обновляем текст метки. Это форматированная строка вида:

часы: минуты:секунды:миллисекунды

8. Запускам и проверяем работоспособность кода.

Рефакторинг

Можно чуть упростить и сделать код аккуратнее, без изменения логики.

1) Компактные методы:

```
private void startBTN_Click(object sender, EventArgs e) => timer1.Start();
1 reference
private void stopBTN_Click(object sender, EventArgs e) => timer1.Stop();
```

- startBTN_Click и stopBTN_Click стали однострочными.

2) **Инициализация переменных в одной строке**

```
private int hours, minutes, seconds, milliseconds;
```

- Вместо объявления всех по отдельности.

3) Выделяем метод UpdateLabel() и изменим вывод миллисекунд:

```
private void UpdateLabel() {
    label1.Text = $"{hours:00}:{minutes:00}:{seconds:00}:{milliseconds:000}";
}
```

- Теперь выводится ровно 3 цифры (например, 035 вместо 35), что привычнее для секундомера.

4) Вызываем **UpdateLabel()** в методах **resetBTN_Click** и **timer1_Tick**:

```
private void resetBTN_Click(
    timer1.Stop();
    hours = minutes = second
    UpdateLabel();
}
```

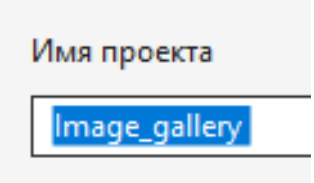
```
private void timer1_
    // код
    UpdateLabel();
}
```

- Вместо дублирования строки форматирования — отдельный метод, отвечающий только за обновление текста.

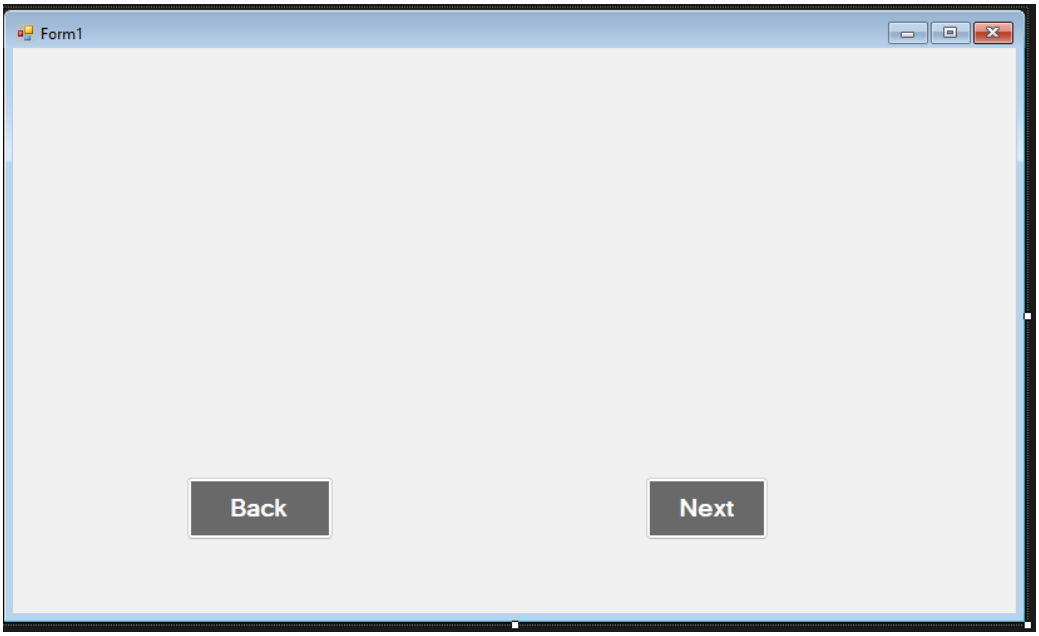
Приложение 2. Галерея изображений

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.

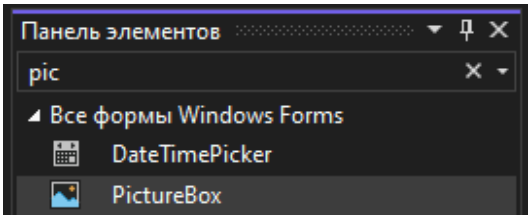
2. Создаём новое приложение WF и называем его **Imagy_gallery**



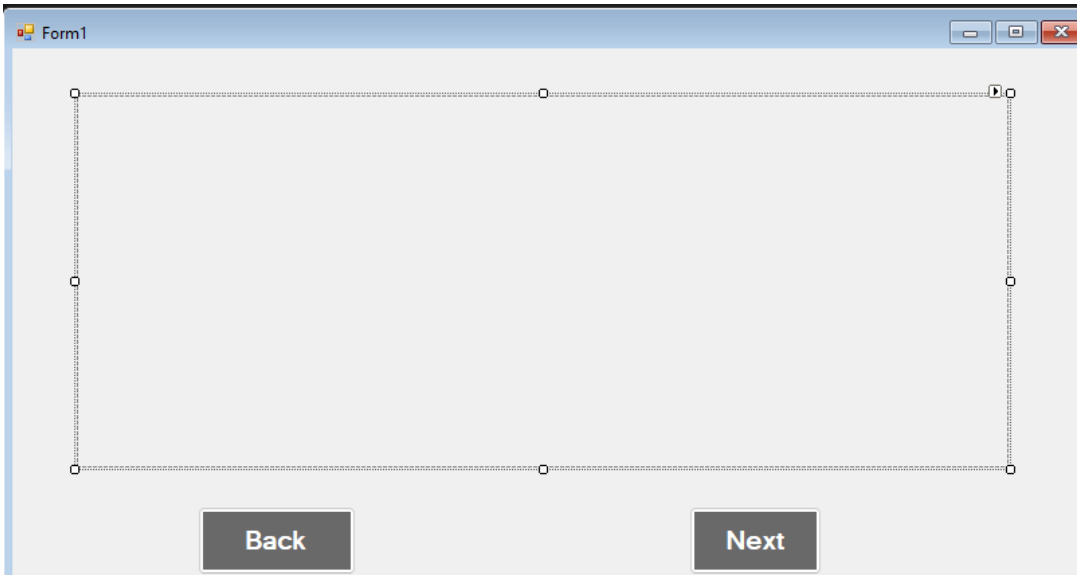
Добавляем две кнопки, и настраиваем внешний вид:



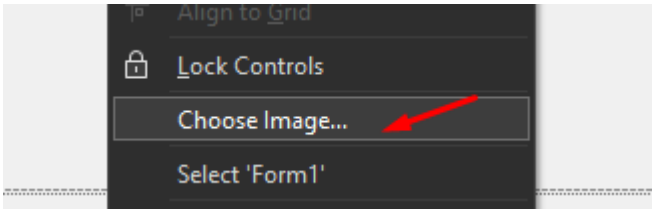
3. Добавляем **PictureBox**:



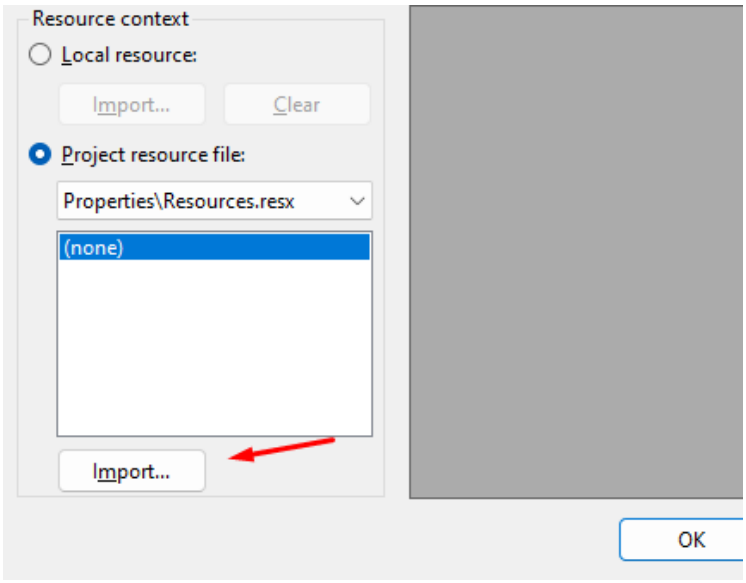
4. Увеличьте его размер:



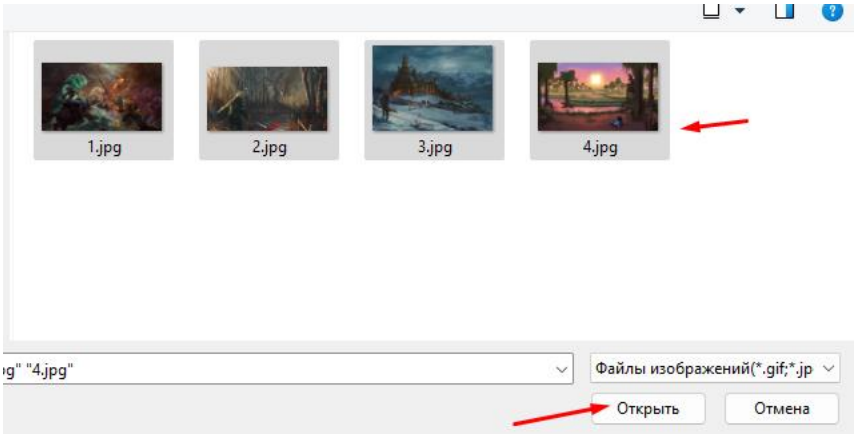
5. Щёлкните правой кнопкой мыши по **PictureBox** и выберите пункт «**Chose Image**»:



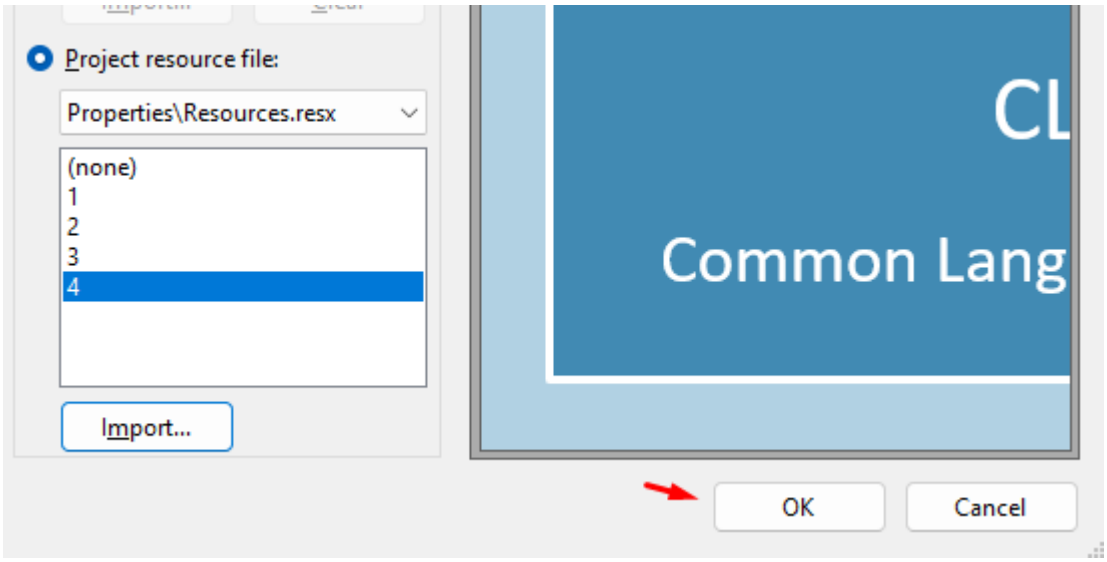
Далее нажимает Import:



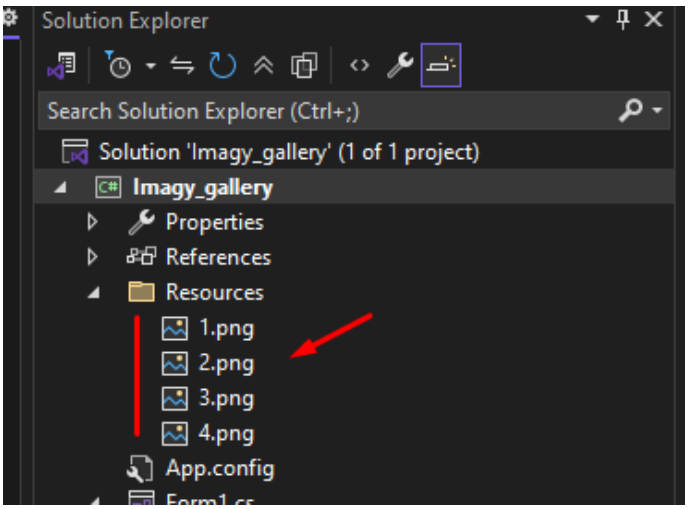
Выделяем все наши изображения и нажимаем **открыть**:



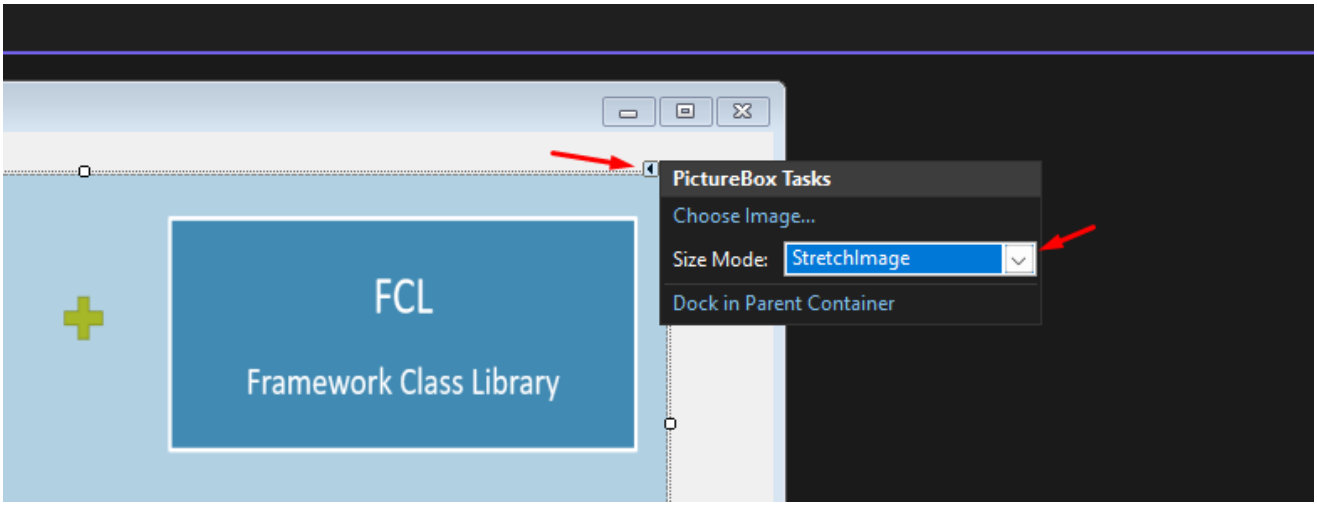
Видим, что наши изображения появились, и жмём **ОК**:



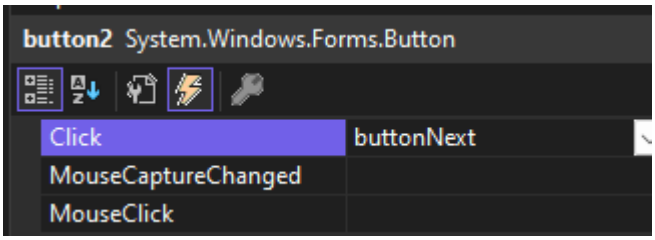
Они также отобразились в **Solution Explorer:**



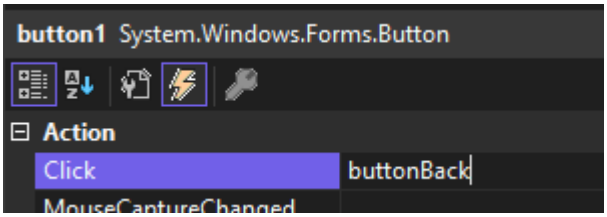
Чтобы растянуть изображение на наш холст, нужно нажать на стрелочку и выбрать один из режимов (например **StretchImage** или **Zoom**):



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:

1. Удаляем ненужные пространства имён, и оставляем только базовые:

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

2. Создадим начальную переменную **i** для отслеживания текущего изображения. Изначально установим значение равное 1:

```
public partial class Form1 : Form {
    int i = 1;
```

- Переменная i хранит номер текущего изображения.
- Начинаем с **1** → значит, при запуске будет первое изображение.

3. Создадим метод для смены изображения **changeImage**():

```
private void changeImage(int number) {
    switch (number) {
        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;
    }
}
```

- В switch проверяется, какое число передано в метод (number).
- В зависимости от этого числа выбирается нужная картинка из ресурсов проекта (Properties.Resources).
- Например, если number == 2, то в PictureBox отобразится картинка _2.

4. Кнопка **Next (вперёд)**

```
private void buttonNext(object sender, EventArgs e) {
    i++;
    if (i > 5) i = 1;
    changeImage(i);
}
```

- Увеличиваем номер картинки i++.
- Если номер стал больше 5 → возвращаемся к первому (i = 1).
- Вызываем метод changeImage(i) → меняем картинку.

То есть можно **листать картинки по кругу**: после последней снова идёт первая.

5. Кнопка **Back (назад)**

```
private void buttonBack(object sender, EventArgs e) {
    i--;
    if (i < 1) i = 5;
    changeImage(i);
}
```

- Уменьшаем номер картинки i--.
- Если номер стал меньше 1 → переходим на последнюю (i = 5).
- Вызываем метод changeImage(i) → показываем нужную картинку.

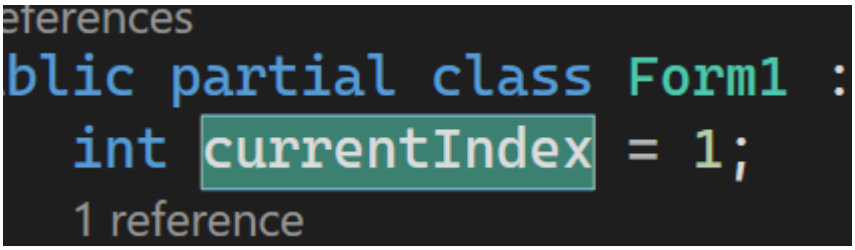
Таким образом, можно **листать изображения в обе стороны**.

Запустите и протестируйте приложение.

Рефакторинг

1) **Говорящее имя переменной**

Переменной i лучше дать более **говорящее название**, чтобы сразу было понятно, что она хранит текущую позицию картинки. Переименовываем на **currentIndex** (индекс текущего изображения):



Для быстрого переименования по всему коду вспомните сочетание клавиш **Ctrl+R+R**

2) **Массив изображений**

```
public partial class Form1 : Form {
    int currentIndex = 1;

    private readonly object[] images = {
        Properties.Resources._1,
        Properties.Resources._2,
        Properties.Resources._3,
        Properties.Resources._4
    };
```

- Все картинки собраны в массив и не нужно писать switch и case.

3) **Создаём метод ShowImage()**

```
private void ShowImage() {
    pictureBox1.Image = (System.Drawing.Image)images[currentIndex];
}
```

- Вынесена логика обновления PictureBox в отдельный метод и код стал чище.

4) Вызываем метод **ShowImage() в главной форме:**

```
public Form1() {
    InitializeComponent();
    ShowImage();
}
```

5) Обновляем код в кнопках:

```
private void buttonNext(object sender, EventArgs e) {
    currentIndex++;
    if (currentIndex >= images.Length)
        currentIndex = 0;
    ShowImage();
}
```

```
private void buttonBack(object sender, EventArgs e) {
    currentIndex--;
    if (currentIndex < 0) currentIndex = images.Length - 1;
    ShowImage();
}
```

- Используется images.Length, а не захардкоженные 4.
- Если в будущем будет больше картинок, то код сразу будет работать без изменений.

6) Метод changeImage() нам больше не нужен, поэтому удаляем его:

```
private void changeImage(int number) {
    switch (number) {
        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;
    }
}
```

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: