Министерство науки и высшего образования Российской Федерации
Рубцовский индустриальный институт (филиал)
ФГБОУ ВО «Алтайский государственный технический
университет им. И.И. Ползунова»

Кафедра прикладной математики

Л.А. ПОПОВА

ПРОГРАММИРОВАНИЕ ПРИЛОЖЕНИЙ (ЧАСТЬ 2)

Учебно-методические указания для студентов направления
«Информатика и вычислительная техника»

Рубцовск 2021

ББК 32.973.2

Попова Л.А. Программирование приложений (часть 2): Учебно-методические указания для студентов направления «Информатика и вычислительная техника» / Л.А. Попова. – Рубцовск: РИИ, 2021. – 51 с. [ЭР].

Учебно-методические указания предназначены для проведения практических и лабораторных работ по курсу «Программирование приложений» у студентов направления 09.03.01 «Информатика и вычислительная техника» очной и заочной форм обучения.

Приведены задания к практическим и лабораторным работам, а также вопросы для самостоятельной подготовки к текущему и промежуточному тестированию.

Рассмотрены и одобрены на заседании кафедры прикладной математики Рубцовского индустриального института.
Протокол № 9 от 18.03.2021 г.

Содержание

Введение

Учебно-методические указания написаны в соответствии с программой дисциплины «Программирование приложений» для студентов направления «Информатика и вычислительная техника» очной и заочной форм обучения, предназначены для аудиторной и самостоятельной работы по данному курсу.

Указания содержат краткий теоретический материал по основам программирования на языке C#, примеры решения задач, задания к лабораторным работам и практическим занятиям, а также вопросы для самостоятельной работы и подготовки к текущему контролю успеваемости и промежуточной аттестации.

Перечень планируемых результатов обучения по дисциплине, соотнесенных с индикаторами достижения компетенций

Компетенция	Содержание компетенции	Индикатор	Содержание индикатора
ПК-5	Способен разрабатывать требования и проектировать программное обеспечение	ПК-5.1	Применяет выбранные языки программирования для написания программ
ПК-3	Способен проектировать пользовательские интерфейсы по готовому образцу или концепции интерфейса	ПК-3.1	Проектирует интерфейс по концепции или по образцу уже спроектированной части интерфейса

Общий объем дисциплины в 3 семестре: 4 з.е. / 144 час.
Форма промежуточной аттестации: Зачет

1 Технология Windows Forms

1.1 Макет окна приложения

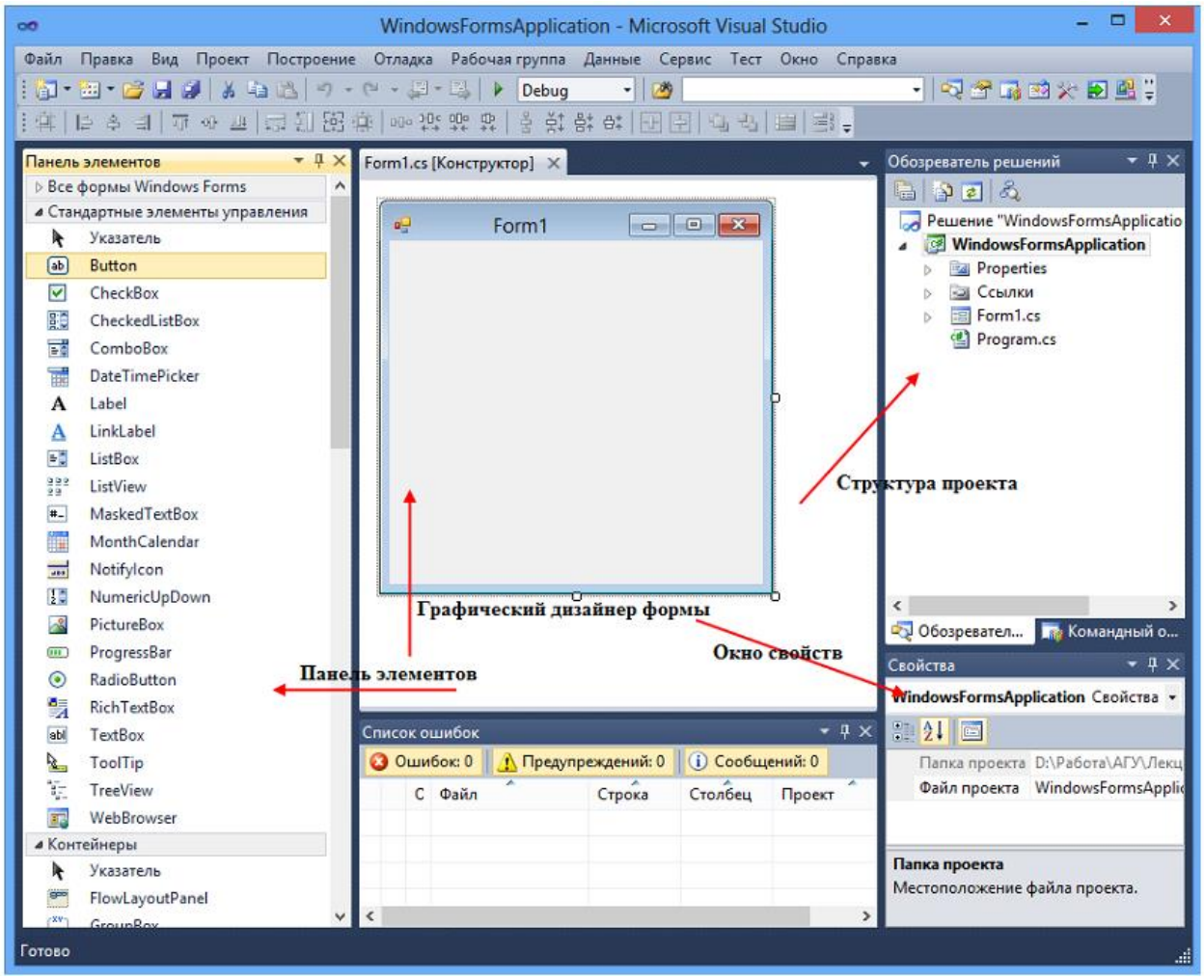


Рисунок 1 – Макет окна приложения

Точный макет окна зависит от версии Visual Studio, настроек и других факторов. Однако необходимо убедиться, что все три окна присутствуют на экране.

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 – Обозреватель решений (Solution Explorer). Там и находятся все связанные с данным приложением файлы, в том числе файлы формы Form1.cs. Внизу справа находится Окно свойств (Properties) – основной инструмент настройки файлов проекта, формы и ее компонентов. Содержимое этого окна представляет собой весь список свойств выбранного в данный момент компонента или формы.

Вызывается это окно несколькими способами:

- в меню **Вид** (**View**) выбрать пункт **Окно свойств** (**Properties Window**) (или клавиша **F4**);
- на выбранном объекте щелкнуть правой кнопкой мыши и в контекстном меню найти пункт **Свойства** (**Properties**);
- выделить объект и нажать клавишу **F4**.

Окно свойств позволяет управлять размером, внешним видом и поведением создаваемых элементов управления.

1.2 Работа с формами

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.

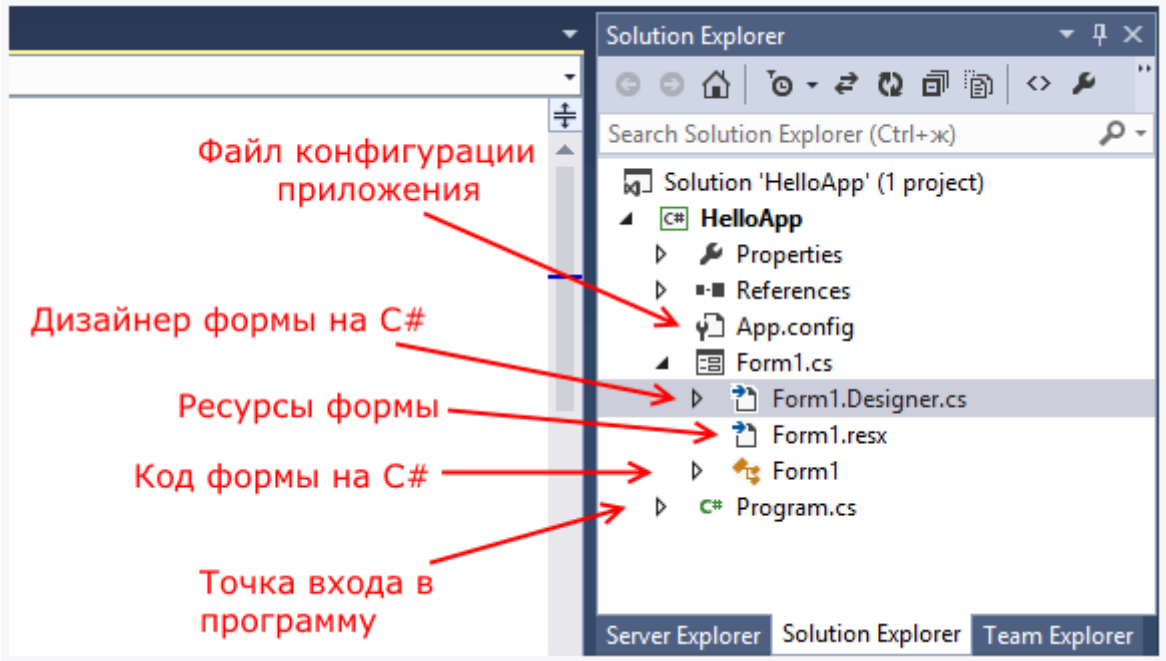


Рисунок 2 – Структура оконного приложения

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

Сама форма сложна по содержанию. Она делится на ряд компонентов. Так, в структуре проекта есть файл Form1.Designer.cs. В нем объявляется частичный класс формы Form1, который имеет два метода: Dispose() – выполняет роль деструктора объекта, и InitializeComponent() – устанавливает начальные значения свойств формы. При добавлении элементов управления, например, кнопок, их описание также добавляется в этот файл.

Файл Form1.resx хранит ресурсы формы. Как правило, ресурсы используются для создания однообразных форм сразу для нескольких языковых культур.

Файл Form1.cs содержит код или программную логику формы.

С помощью специального окна Properties (Свойства) Visual Studio предоставляет удобный интерфейс для управления свойствами элемента. Большинство этих свойств оказывает влияние на визуальное отображение формы.

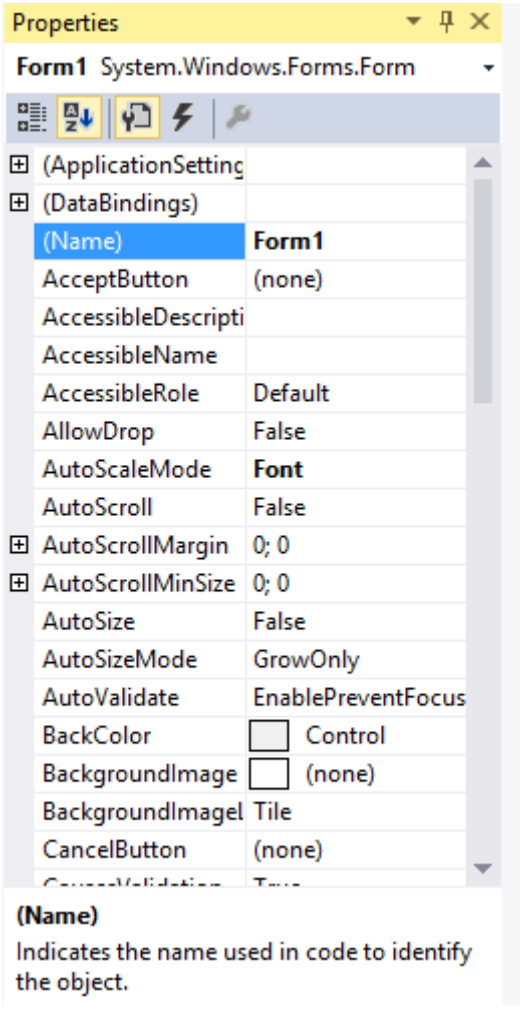


Рисунок 3 – Окно для настройки свойств объекта

Для настройки нужно выделить форму (щелкнуть на любом ее свободном месте). В окне Свойства отображается имя редактируемого объекта. Значения свойств объектов, установленные по умолчанию, имеют обычное начертание, а измененные значения – полужирное начертание.

У формы есть различные свойства. Например, можно установить цвет переднего плана (ForeColor) и фона (BackColor), текст заголовка (Text), который отображается в верхней части формы, размер формы (Size) и другие свойства.

Для некоторых свойств, таких как BackColor и ForeColor, значения можно определить средствами специального редактора. Поскольку выбор цвета важен для дизайна пользовательского интерфейса, следует отнестись к нему очень внимательно. Чтобы облегчить чтение форм, следует применять высококонтрастные цветовые схемы.

Свойство BackGroundImage позволяет использовать изображение вместо однотонного фона. Если задано фоновое изображение, то изменение свойства BackColor формы по умолчанию отразится на значении одноименного свойства всех размещенных на ней элементов управления.

Свойство ShowInTaskBar получает или задает значение, указывающее, отображается ли форма на панели задач Windows. Свойство FormBorderstyle устанавливает тип границы, которая появляется вокруг формы. Значения могут

быть следующие: Sizable, SizableToolWindow, Fixed3D, FixedDialog, Fixedsingle, FixedToolWindow, None. Большинство значений этих свойств очевидно, за исключением двух инструментальных окон со стилями SizableToolWindow и FixedToolWindow – окна не будут появляться на панели задач, независимо от того, как установлено свойство ShowInTaskBar. Также инструментальные окна не будут показываться в списке окон, когда пользователь нажмет комбинацию клавиш Alt + Tab.

Для позиционирования окна формы на экране при ее первом отображении используется свойство StartPosition, которое принимает значение из перечисления FormStartPosition: CenterParent – форма центрируется в клиентской области родительской формы; CenterScreen – форма центрируется на текущем экране; Manual – местоположение формы основано на свойстве Location; WindowsDefaultBounds – форма располагается в позиции по умолчанию, определяемой Windows, и имеет размеры по умолчанию; WindowsDefaultLocation – форма располагается в позиции по умолчанию, определяемой операционной системой.

С помощью значений свойств можно изменить внешний вид формы, но все то же самое можно сделать динамически в коде. Для перехода к коду формы можно вызвать контекстное меню формы и выбрать команду View Code (Просмотр кода) или дважды щелкнуть на компоненте формы, что не только открывает код, но и создает (или открывает для редактирования) соответствующий обработчик события.

1.3 Добавление форм. Взаимодействие между формами

Чтобы добавить еще одну форму в проект, можно нажать правой кнопкой мыши на имя проекта в окне Solution Explorer (Обозреватель решений) и выбрать Add (Добавить) ->Windows Form... Затем в диалоговом окне ввести имя формы.

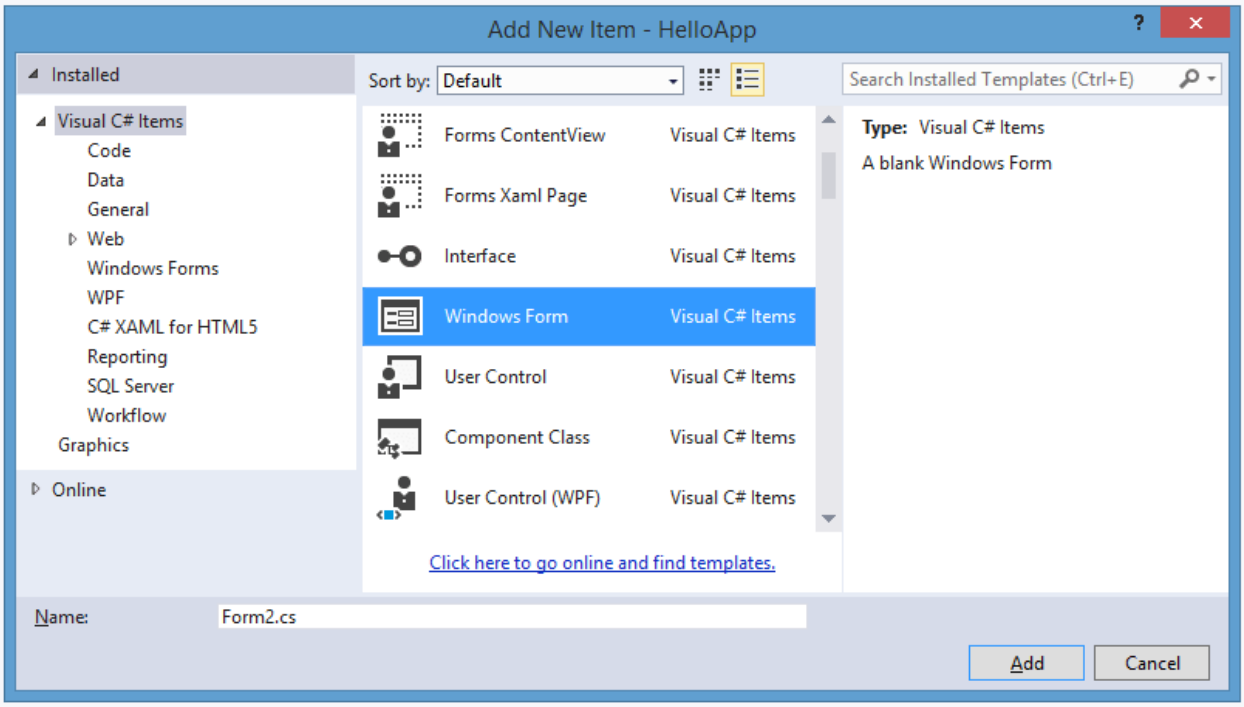


Рисунок 4 – Диалоговое окно создания новой формы

Теперь в проекте две формы. Осуществим 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 ней перейдем в файл кода. Попадем в обработчик события нажатия кнопки:

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

}
```

Добавим в него код вызова второй формы Form2:

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

Пока вторая форма не знает о существовании первой. Сделаем так, чтобы вторая форма воздействовала на первую. Для этого воспользуемся передачей ссылки на форму в конструкторе.

Перейдем к коду второй формы. Пока он пустой и содержит только конструктор. Изменим файл кода второй формы:

```
namespace HelloApp
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        public Form2(Form1 f)
        {
            InitializeComponent();
            f.BackColor = Color.Yellow;
        }
    }
}
```

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

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

Поскольку слово this представляет ссылку на текущий объект Form1, то при создании второй формы она будет получать ссылку и через нее управлять первой формой.

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

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

Лабораторная работа 1

Технология Windows Forms

1 Запустите **Microsoft Visual Studio**.

2 Создайте новый проект **Windows Forms** (.NET Framework) в своей папке.

3 Изучите элементы управления, используемые для редактирования.

4 Изучите структуру проекта в **Обозревателе решений**.

5 Откройте файл с кодом формы. Для этого вызвать контекстное меню формы и выбрать команду **Перейти к коду** или нажать клавишу **F7**.

6 Переименуйте все файлы, связанные с формой в **MainForm**.

Для этого вызовите контекстное меню файла **Form1.cs** и выберите команду **Переименовать** (рисунок 2), введите имя MainForm.cs и подтвердите переименование всех связанных ссылок (рисунок 3).

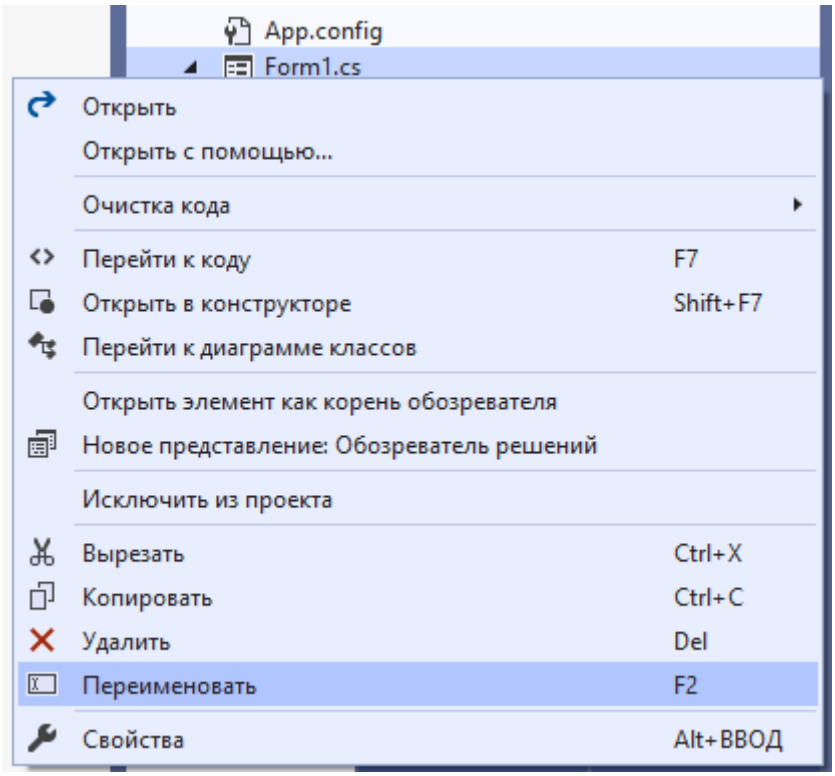


Рисунок 5 – Диалоговое окно переименования файлов проекта

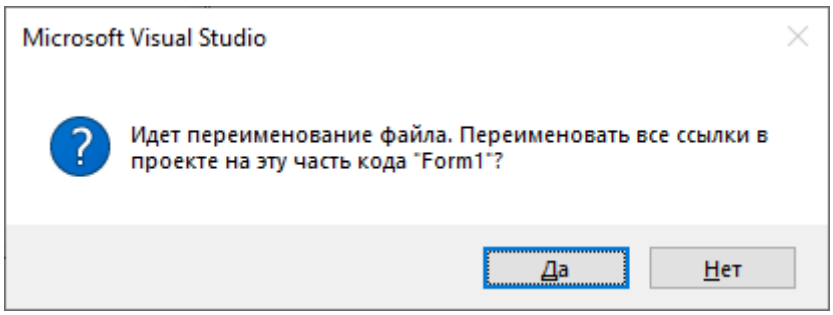


Рисунок 6 – Диалоговое окно для подтверждения переименования файла

7 Ознакомьтесь со свойствами формы.

8 Настройте свойства формы:

- для свойства **Text** ввести значение **Технология Windows Forms**, затем нажать клавишу **Enter**. Теперь форма в заголовке окна должна содержать соответствующий текст;
- определить расположение формы после запуска программы на выполнение – в центре экрана;
- изменить цвет формы, используя кодировку RGB (запустить графический редактор, например, Paint, и с его помощью определить значения каждого канала для выбранного цвета; установить эти значения (ввести через точку с запятой) для свойства BackColor формы);
- задать прозрачность (**Opacity**) 95%;
- добавить рисунок в центр формы.

9 Добавьте на форму справа от рисунка объекты:

- контейнер для группировки (GroupBox) с заголовком **Направление движения**; внутри него разместить три переключателя (RadioButton): **по горизонтали** (для свойства Checked – значение True), **по вертикали** и **по контуру**;
- две кнопки с текстом **Движение** и **Выход** так, чтобы они были расположены вблизи от центра формы. Определить их свойства самостоятельно.

10 Расположите метку (Label) в верхнем левом углу. Введите текст, например, **Hello World**, размер которого **24**, **полужирный** (свойство **Font**).

11 Изучите изменения, внесенные в файл MainForm.Designer.cs.

12 Вызовите обработчик события нажатия на левую кнопку (Click) дважды щелкнув по кнопке с текстом **Выход**.

13 Переименуйте метод (вызовите соответствующую команду из контекстного меню или нажмите «горячие клавиши»), введите новое имя **Exit**.

Этот метод будет отвечать за вывод диалогового окна (рисунок 7) и завершения работы программы.

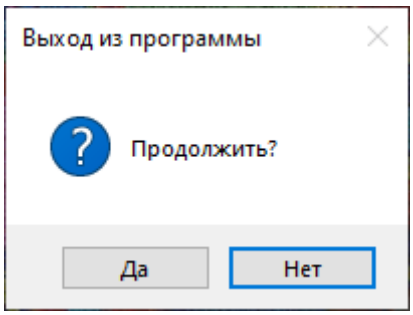


Рисунок 7 – Диалоговое окно для завершения работы

За формирование и вывод диалогового окна будет отвечать команда:

```
DialogResult result = MessageBox.Show("Продолжить?", "Выход из программы",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2);
```

За завершение работы команда:

```
if (result == DialogResult.Yes) Close();
```

Так как выход из программы организован с помощью диалогового окна, то можно убрать кнопки управления окном (свойству **ControlBox** присвоить значение **False**).

14 Аналогично создайте обработчик события с именем **Motion** для кнопки **Движение**.

Данный метод будет отвечать за движение метки по форме в одном из направлений, определенном с помощью переключателей.

Создайте три дополнительных метода для организации движения:

```
void Horizontally() // Движение по горизонтали.
{

}
void Vertically() // Движение по вертикали.
{

}
void AlongTheContour() // Движение по контуру.
{

}
```

15 В методе **Motion** определите условия для вызова этих методов:

```
if (radioButton1.Checked) Horizontally();
else if (radioButton2.Checked) Vertically();
else AlongTheContour();
```

16 Введите в методы команды, задающие движение метки по форме, с учетом их размеров (так, чтобы при изменении размеров формы или метки не пришлось редактировать код методов).

Лабораторная работа 2

Меню, формы и пользовательские элементы управления

1 Создайте новый проект **Windows Forms** (.NET Framework) в своей папке.
2 Переименуйте все файлы, связанные с формой в **MainForm**.
3 Настройте свойства формы.
4 Добавьте на форму объект меню (**MenuStrip**).

Элемент управления **MenuStrip** представляет контейнер для структуры меню формы. Можно добавить объекты **ToolStripMenuItem** в объект MenuStrip, который представляет отдельные команды в структуре меню. Каждый объект ToolStripMenuItem может быть командой для приложения или родительским меню для других элементов вложенного меню. Элемент управления **MenuStrip** дает возможность визуально конструировать систему главного меню формы (рисунок 8). Перетаскивание этого элемента управления из панели Toolbox на пустую форму автоматически прикрепит меню к верхнему краю формы. После того как поместили на форму этот элемент управления, выбор элемента MenuStrip активирует смарт-тег. Смарт-тег позволяет автоматически вставить в меню стандартные пункты и выставить свойства **RenderMode**, **Dock** и **GripStyle**.

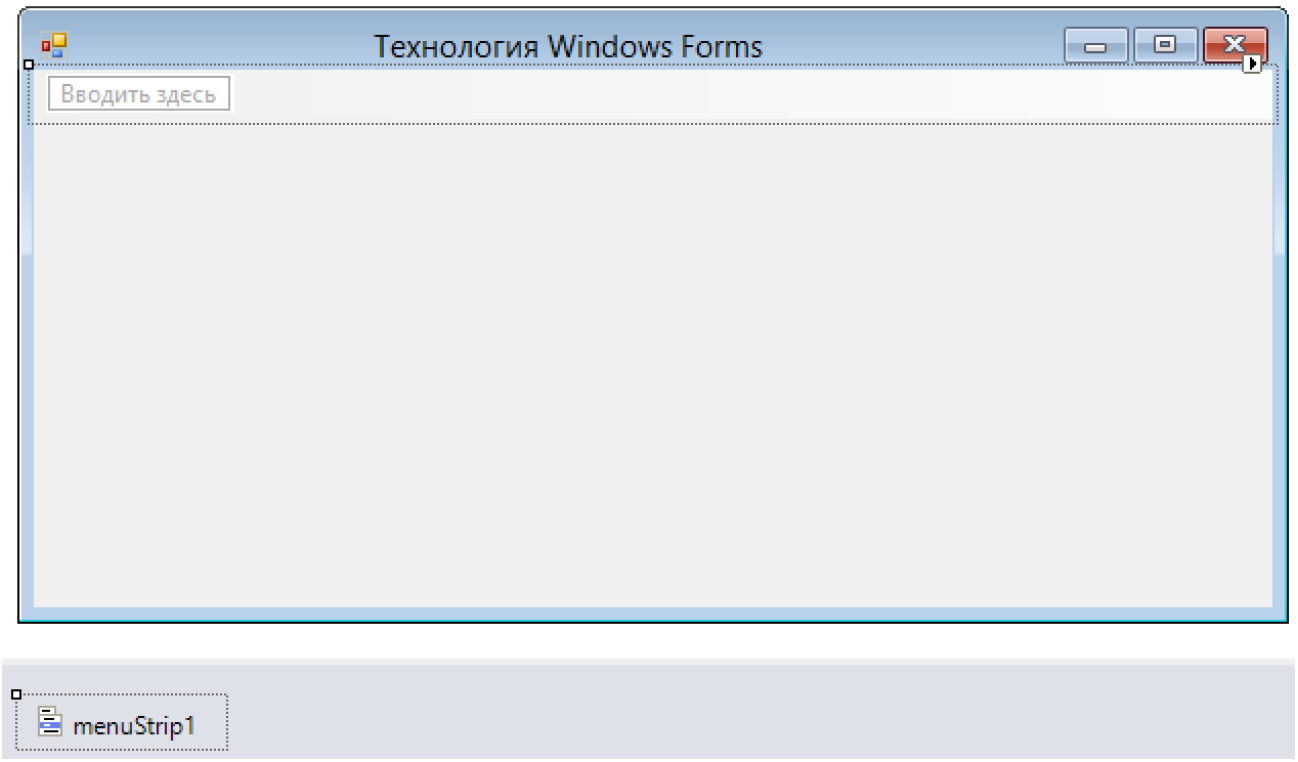


Рисунок 8 – Форма, содержащая элемент управления «Главное меню»

5 Введите название нового пункта меню, например Файл (рисунок 9).

Введенный текст появляется и в редакторе «**Меню**», и в поле **Text** в окне **Свойства** (рисунок 10). Изменить свойства нового пункта меню можно в любом из этих мест.

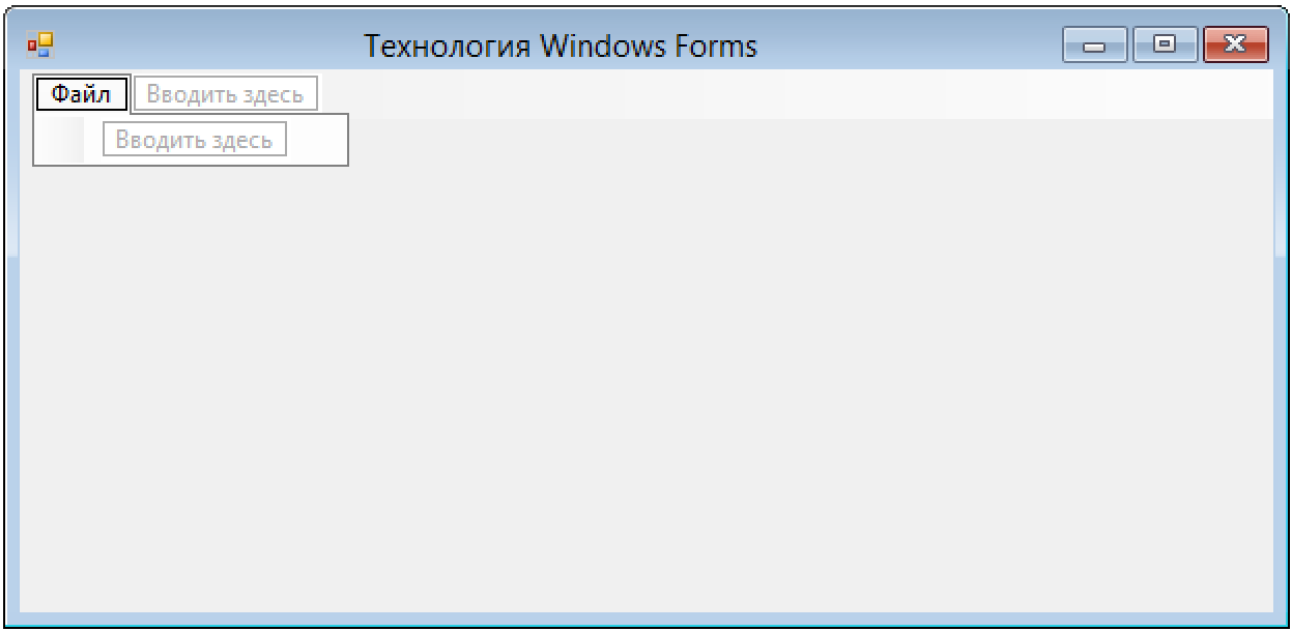


Рисунок 9 – Ввод пунктов меню

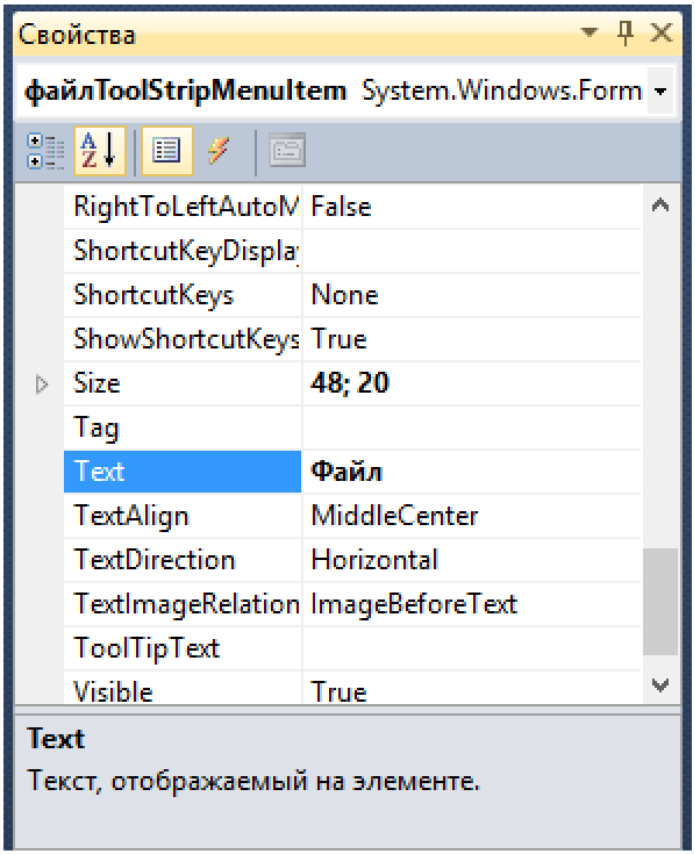


Рисунок 10 – Окно свойств меню

Как только будет введено название пункта меню, справа появится поле для ввода нового элемента, а также появится область для ввода подпункта.

6 Создайте следующее меню (рисунок 11).

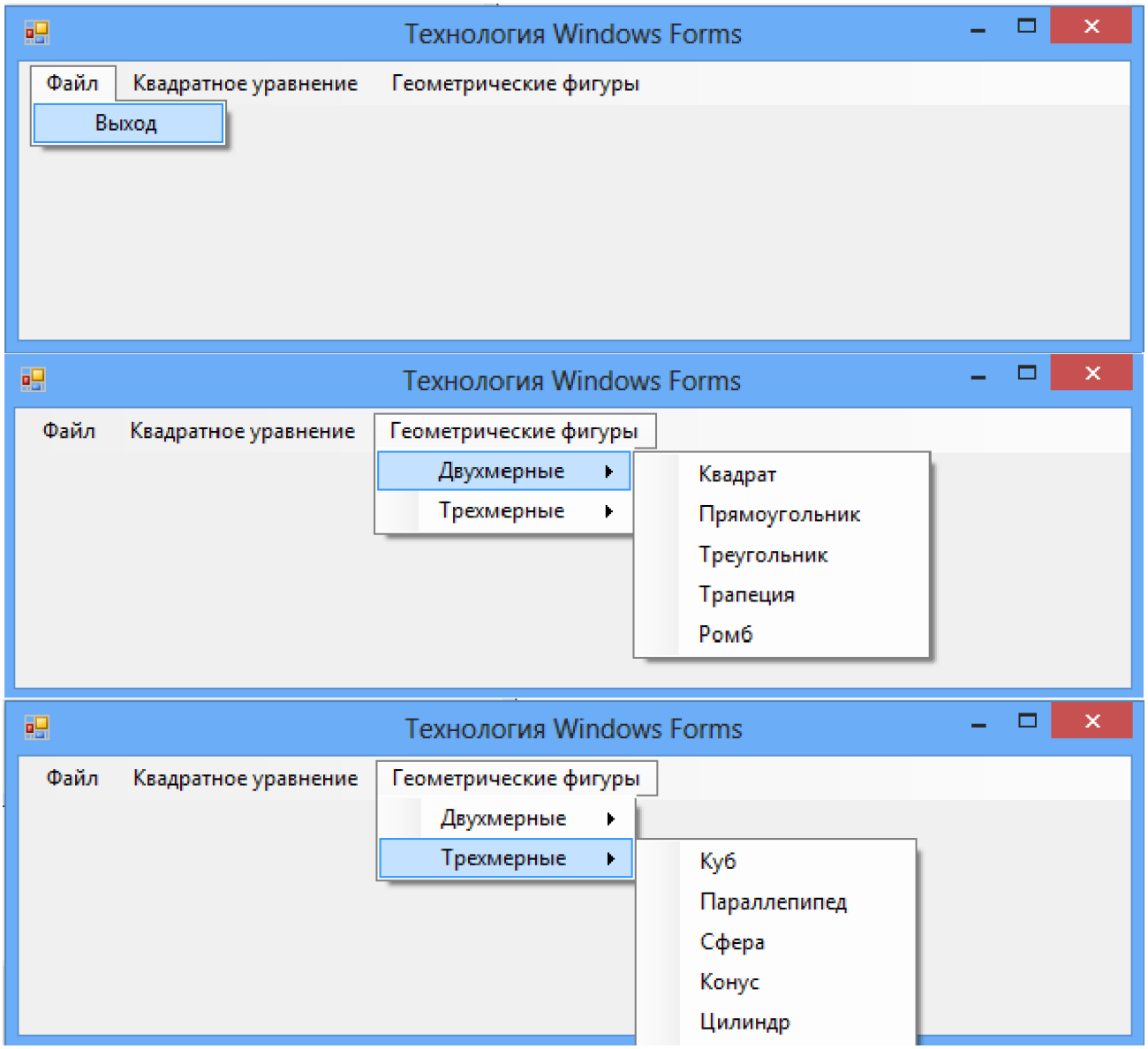


Рисунок 11 – Содержимое главного меню окна

7 Введите обработчик события для подпункта «**Выход**» в пункте «**Файл**» (см. лабораторную работу №1).

8 Создайте дополнительную форму, которая будет вызываться при выборе пункта меню «**Квадратное уравнение**».

Для добавления в проект новой формы, можно щелкнуть правой кнопкой мыши на **имени проекта** в окне **Обозреватель решений** и в контекстном меню выбрать пункт **Добавить** – **Форма Windows** (рисунок 12).

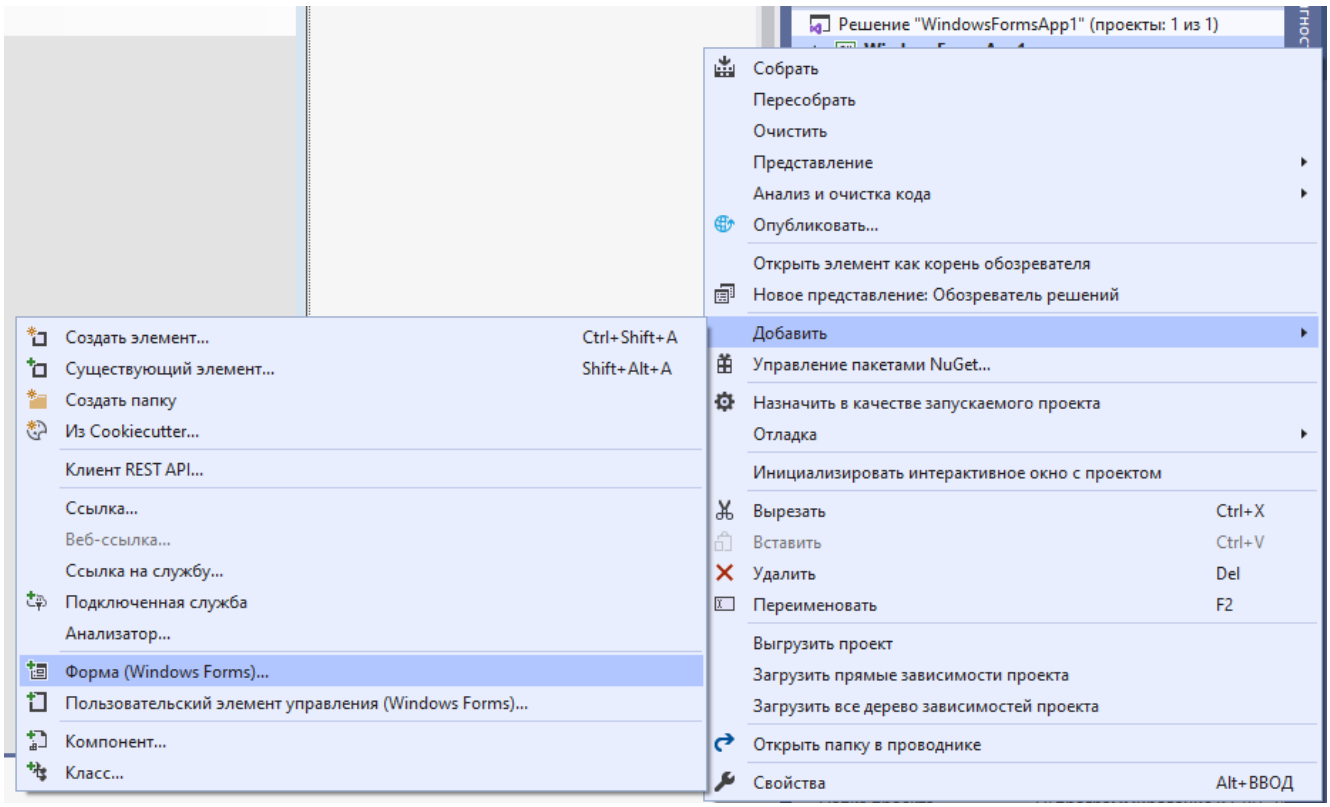


Рисунок 12 – Добавление новой формы в проект

9 Введите новое имя кодового файла, в соответствии с решаемой в нем задачей (решение квадратного уравнения, рисунок 13).

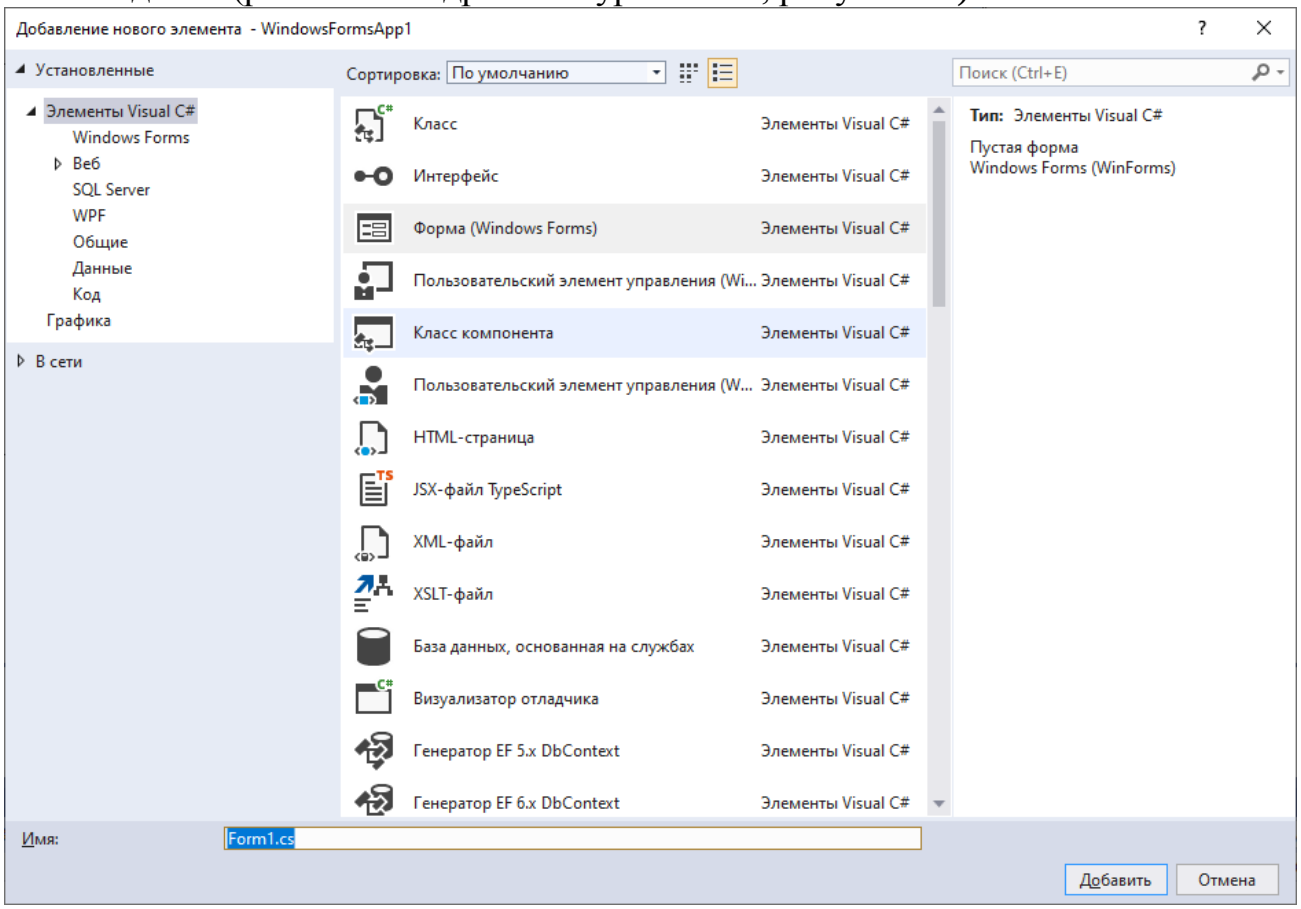


Рисунок 13 – Диалоговое окно создания формы

10 Введите заголовок формы **Решение квадратного уравнения**.

11 Создайте обработчик события для щелчка мышью по пункту **Квадратное уравнение**, который будет открывать вторую форму.

12 Расположите на форме объекты в соответствии с рисунком 14.

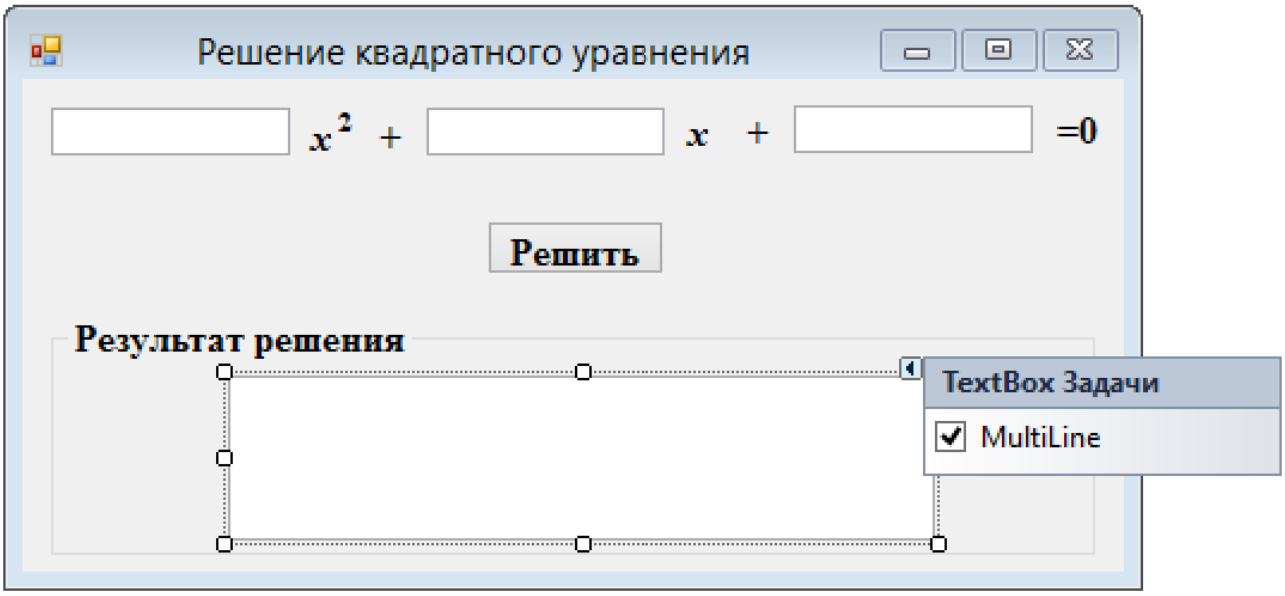


Рисунок 14 – Окно формы «Решение квадратного уравнения»

13 Создайте в проекте две новые **папки**: **Classes** и **UserControls** (рисунок 15).

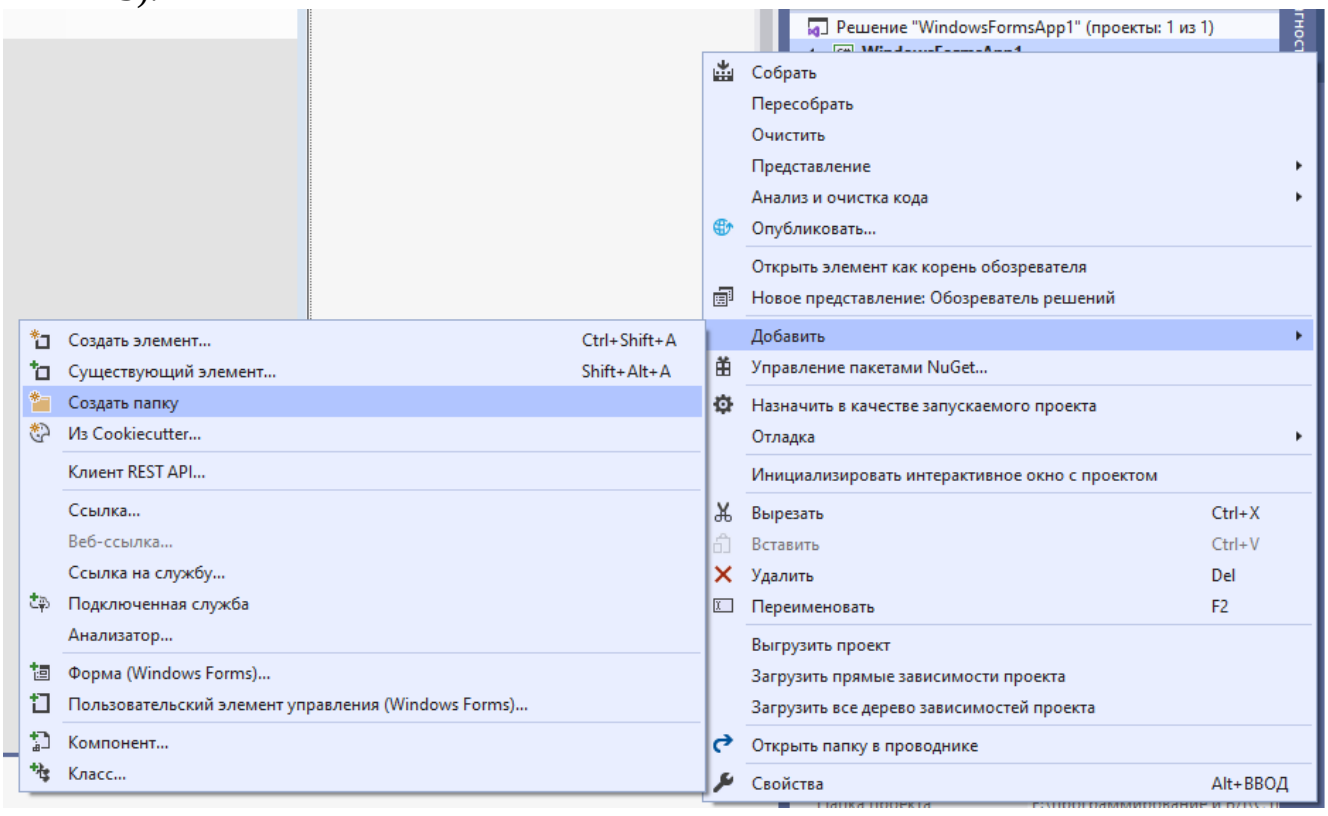


Рисунок 15 – Создание в проекте новой папки

14 Создайте в папке **Classes** класс для решения квадратного уравнения.

15 Измените для класса пространство имен на то, которое используется в проекте (удалить **.Classes**). Введите код.

16 Создайте обработчик события для кнопки «**Решить**». *Примечание*: для ввода данных используйте метод **double.TryParse**().

Лабораторная работа 3

Создание GUI с использованием текстовых полей, меток и кнопок

1 Откройте проект, созданный в лабораторной работе № 2.

2 Создайте в папке **UserControls** новый элемент (рисунок 16).

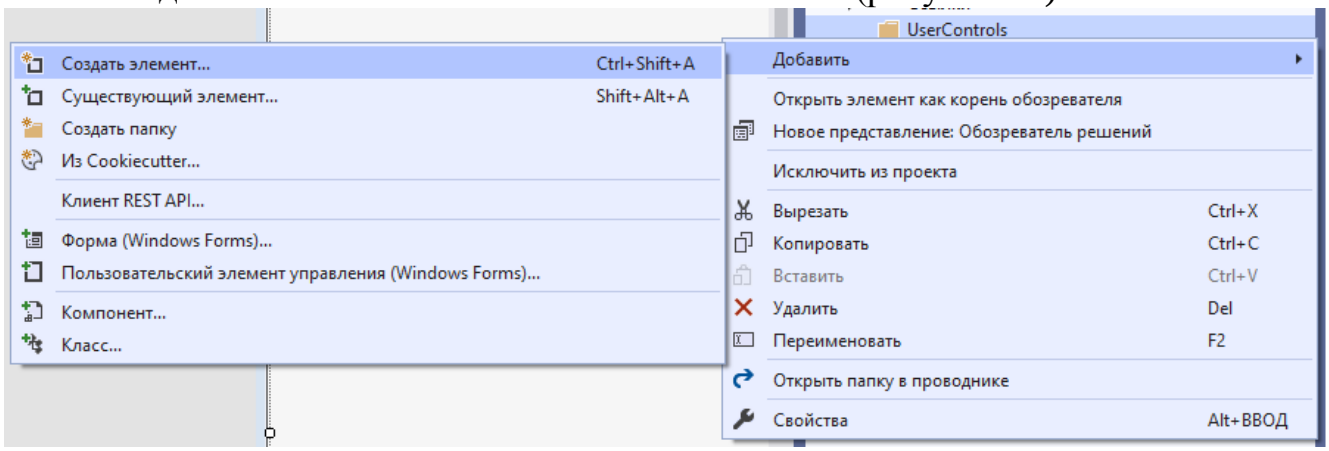
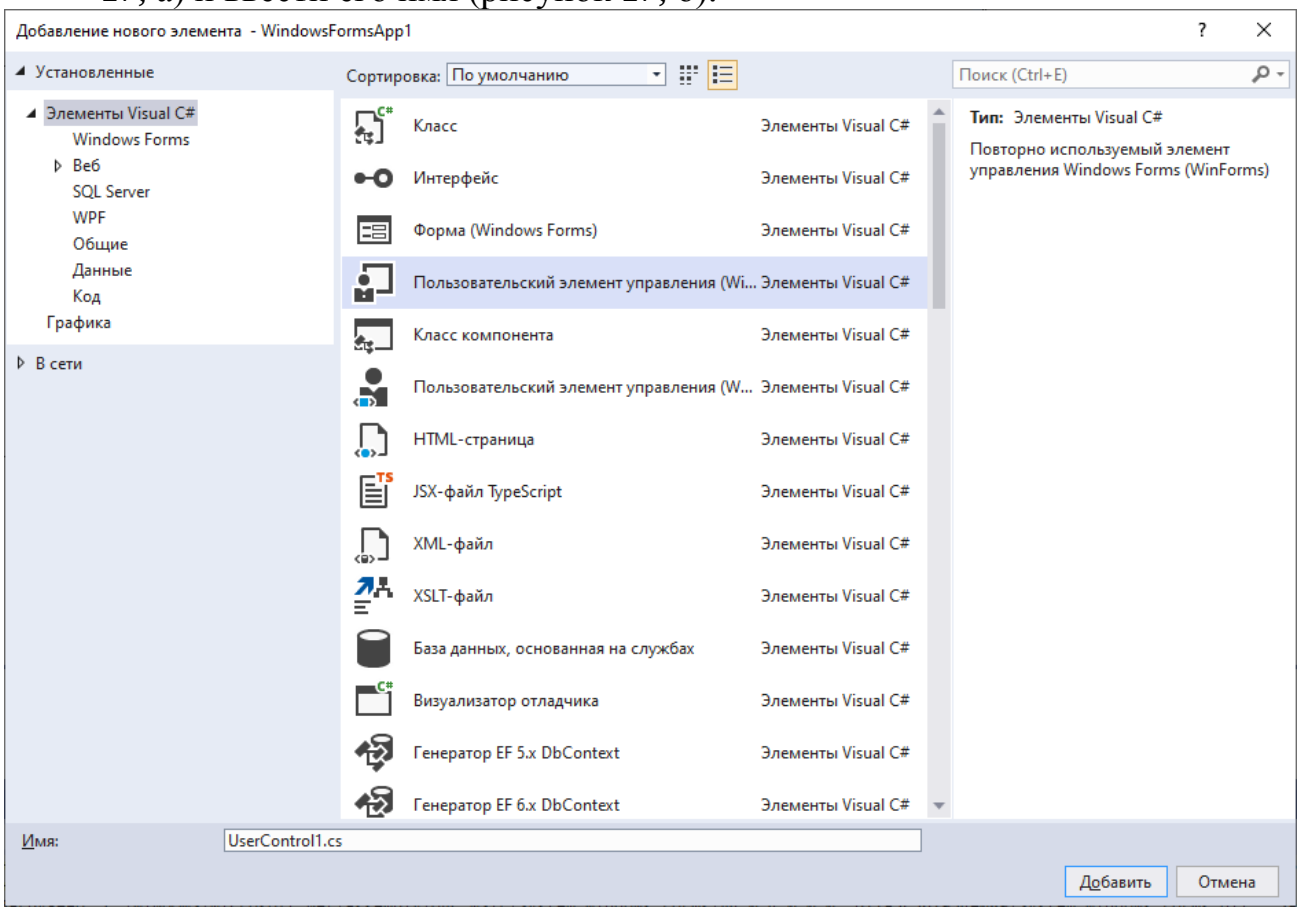
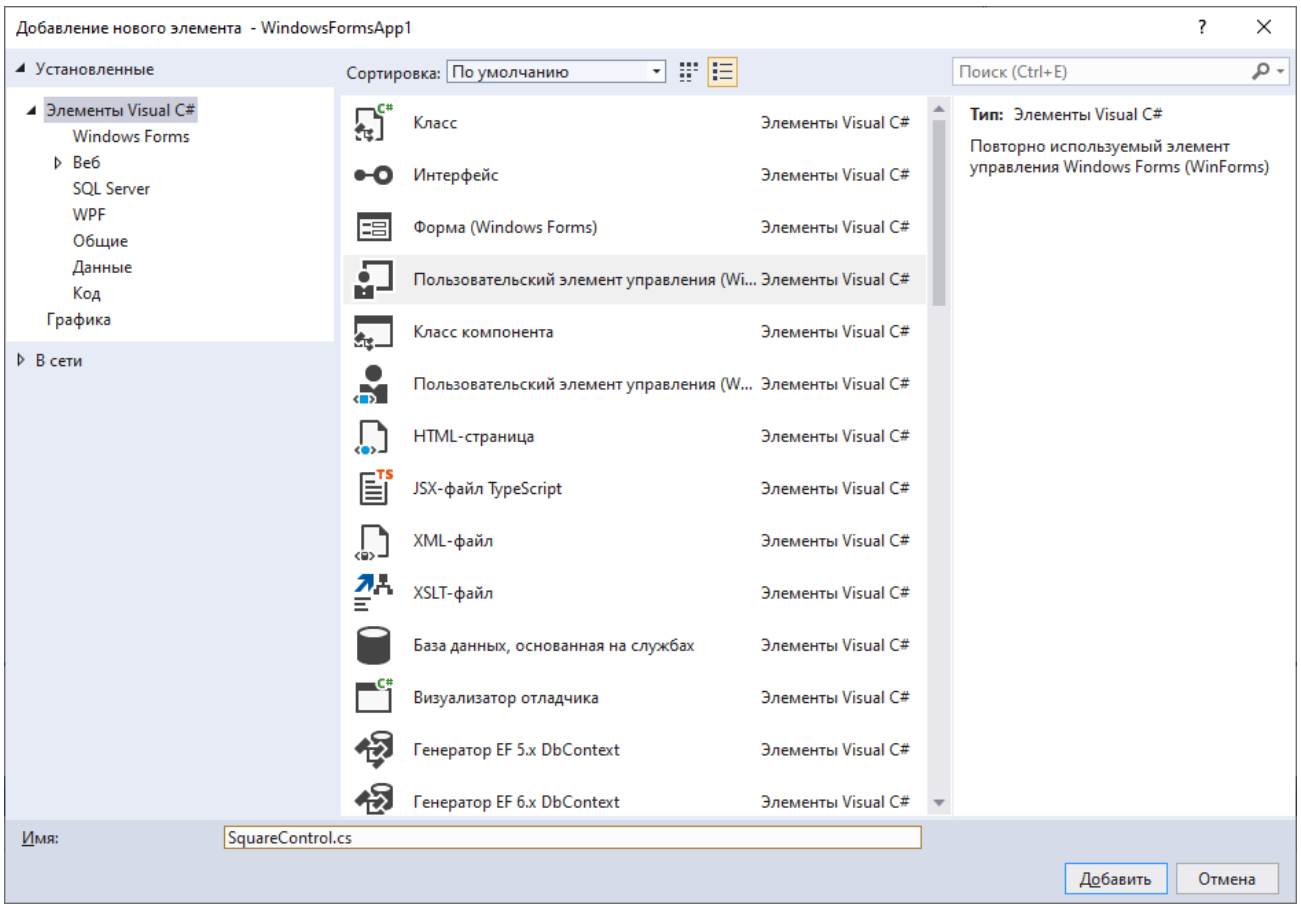


Рисунок 16 – Добавление в папку проекта нового элемента

3 Выберите тип элемента – Пользовательский элемент управления (рисунок 17, а) и ввести его имя (рисунок 17, б).



(а)



(б)

Рисунок 17 – Выбор пользовательского элемента управления в диалоговом окне

4 Задать размеры созданного контрола: 400; 250

5 Расположить на нем элементы управления в соответствии с рисунком 18.

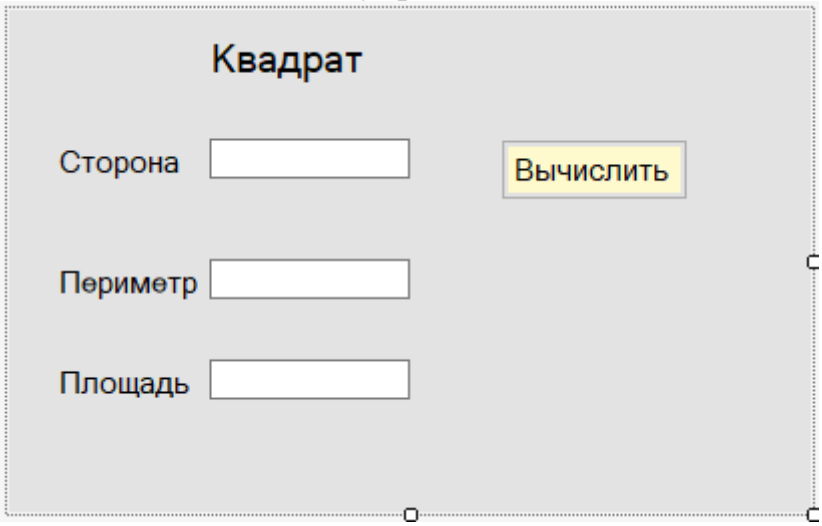


Рисунок 18 – Расположение элементов управления на SquareControl

6 Для полей, в которых будут выводиться значения периметра (**textBox2**) и площади (**textBox3**) задайте свойство **Enabled**: **False**.

7 В поле для ввода стороны квадрата (**textBox1**) разрешите ввод только вещественных чисел. Для этого в списке событий выберите **KeyPress**, создайте обработчик события и введите имя **SideKeyPress**.

```
private void SideKeyPress(object sender, KeyPressEventArgs e)
{
    char charCode = e.KeyChar;
    AllowDoubleNumbers(e, charCode);
}

/// <summary>
/// Разрешить ввод только цифр, запятой и символа backspase.
/// </summary>
/// <param name="e">Событие</param>
/// <param name="charCode">Введенный символ</param>
private void AllowDoubleNumbers(KeyPressEventArgs e, char charCode)
{
    if (!char.IsDigit(charCode) && charCode != ',' && charCode != 8)
        e.Handled = true;
}
```

8 Измените пространство имен для класса. *Примечание*: так как описание класса разделено на две части, то изменения должны быть сделаны в каждой из них.

9 Создайте в папке **Classes** класс для нахождения периметра и площади квадрата, прямоугольника и треугольника.

10 Измените пространство имен.

11 Введите код, добавив к нему строки документации (/// <summary> … /// </summary>) для конструкторов и свойств класса.

12 Создайте обработчик события для кнопки «Вычислить»:

```
private void CalcClick(object sender, EventArgs e)
{
    if (double.TryParse(textBox1.Text, out double a))
    {
        GeometricFig1 square = new GeometricFig1(a);
        textBox2.Text = square.Perimeter.ToString("#.##");
        textBox3.Text = square.Area.ToString("#.##");
    }
    else
    {
        textBox2.Text = "";
        textBox3.Text = "";
        MessageBox.Show("Неправильный формат данных");
    }
}
```

13 Добавьте в проект классы и пользовательские элементы управления для вычисления периметров и площадей геометрических фигур на плоскости (прямоугольника, треугольника, равнобедренной трапеции и ромба); площадей поверхности и объемов фигур в пространстве (куба, прямоугольного параллелепипеда, сферы, конуса и цилиндра).

14 Поместите на форму **MainForm** контейнер **Panel**, размеры которой должны быть не меньше, чем у **SquareControl**.

15 Добавьте обработчик события для выбора команды меню **Геометрические фигуры – Двумерные – Квадрат**.

16 Введите для него код:

```
var userControl = new SquareControl();
if (panel1.Controls.Count > 0)
    panel1.Controls.RemoveAt(0);
panel1.Controls.Add(userControl);
```

17 Аналогичным образом добавьте обработчики событий для всех остальных команд меню.

2 Технология WPF (Windows Presentation Foundation)

2.1 Введение в компоновку

Компоновка (layout) представляет собой процесс размещения элементов внутри контейнера. Возможно, вы обращали внимание, что одни программы и веб-сайты на разных экранах с разным разрешением выглядят по-разному: где-то лучше, где-то хуже. В большинстве своем такие программы используют жестко закодированные в коде размеры элементов управления. WPF уходит от такого подхода в пользу так называемого "резинового дизайна", где весь процесс позиционирования элементов осуществляется с помощью компоновки.

Благодаря компоновке мы можно настроить элементы интерфейса, позиционировать их определенным образом. Например, элементы компоновки в WPF позволяют при сжатии или растяжении масштабировать элементы, что визуально не создает незаполненных пустот на форме.

В WPF компоновка осуществляется при помощи специальных контейнеров. Фреймворк предоставляет нам следующие контейнеры: Grid, UniformGrid, StackPanel, WrapPanel, DockPanel и Canvas.

В WPF при компоновке и расположении элементов внутри окна нам надо придерживаться следующих принципов:

- Нежелательно указывать явные размеры элементов (за исключением минимальных и максимальных размеров). Размеры должны определяться контейнерами.
- Нежелательно указывать явные позицию и координаты элементов внутри окна. Позиционирование элементов всецело должно быть прерогативой контейнеров. И контейнер сам должен определять, как элемент будет располагаться. Если нам надо создать сложную систему компоновки, то мы можем вкладывать один контейнер в другой, чтобы добиться максимально удобного расположения элементов управления.

Процесс компоновки проходит два этапа: измерение (measure) и расстановка (arrange). На этапе измерения контейнер производит измерение предпочтительного для дочерних элементов места. Однако не всегда контейнер имеет достаточно места, чтобы расставить все элементы по их предпочтительным размером, поэтому их размеры приходится усекать. Затем

происходит этап непосредственной расстановки дочерних элементов внутри контейнера.

2.2 Компоновка с помощью Grid

Это наиболее мощный и часто используемый контейнер, напоминающий обычную таблицу. Он содержит столбцы и строки, количество которых задает разработчик. Для определения строк используется свойство *RowDefinitions*, а для определения столбцов – свойство *ColumnDefinitions*:

```
<Grid.RowDefinitions>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
    <RowDefinition></RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
    <ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
```

Каждая строка задается с помощью вложенного элемента *RowDefinition*, который имеет открывающий и закрывающий тег. При этом задавать дополнительную информацию необязательно. То есть в данном случае у нас определено в гриде 3 строки.

Каждая столбец задается с помощью вложенного элемента *ColumnDefinition*. Таким образом, определили 3 столбца, то есть получится таблица 3×3.

Чтобы задать позицию элемента управления с привязкой к определенной ячейке *Grid*, в разметке элемента нужно прописать значения свойств *Grid.Column* и *Grid.Row*, тем самым указывая, в каком столбце и строке будет находиться элемент.

Можно задавать пропорциональные размеры ячеек таблицы.Например, ниже задаются два столбца, второй из которых имеет ширину в четверть от ширины первого:

```
ColumnDefinition Width="*" />
<ColumnDefinition Width="0.25*" />
```

Если строка или столбец имеет размер, указанный как *, то данная строка или столбец будет занимать все оставшееся место. Если есть несколько строк или столбцов, высота которых равна *, то все доступное место делится поровну между ними. При этом если есть коэффициенты, то они складываются и затем все пространство делится на сумму коэффициентов.

Например, если у нас три столбца:

```
<ColumnDefinition Width="*" />
<ColumnDefinition Width="0.5*" />
<ColumnDefinition Width="1.5*" />
```

В этом случае сумма коэффициентов равна 1* + 0.5* + 1.5* = 3*. Если грид имеет ширину 300 единиц, то для коэффициент 1* будет соответствовать пространству 300 / 3 = 100 единиц. Поэтому первый столбец будет иметь

ширину в 100 единиц, второй – 100*0.5=50 единиц, а третий – 100 * 1.5 = 150 единиц.

Можно комбинировать все типы размеров. В этом случае от ширины/высоты грида отнимается ширина/высота столбцов/строк с абсолютными или автоматическими размерами, и затем оставшееся место распределяется между столбцами/строками с пропорциональными размерами:

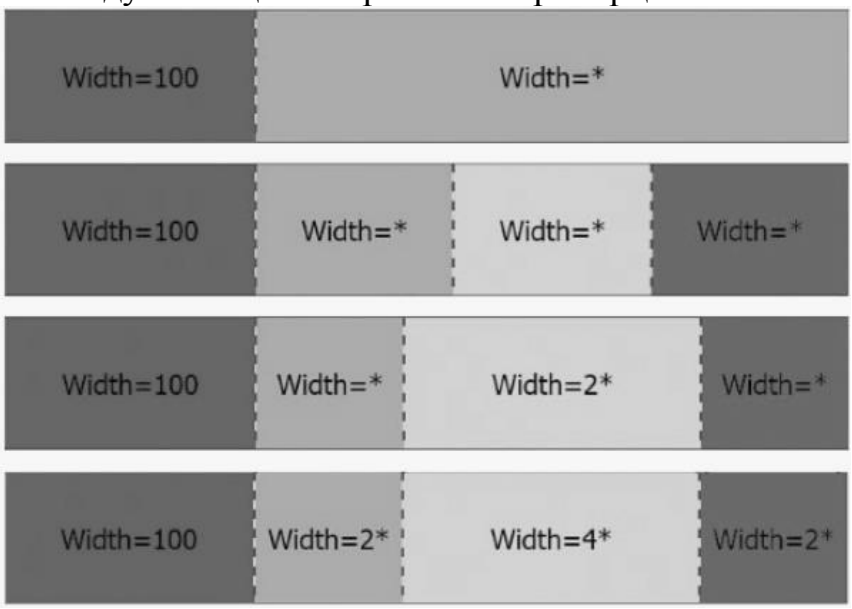


Рисунок 19 – Пример разметки на форме

2.3 Модель событий

WPF в отличие от других технологий, например, от Windows Forms, предлагает новую концепцию событий – *маршрутизированные события* (routed events).

Для элементов управления в WPF определено большое количество событий, которые условно можно разделить на несколько групп событий:

- клавиатуры;
- мыши;
- стилуса;
- сенсорного экрана/мультитач;
- жизненного цикла.

Подключить обработчики событий можно декларативно в файле xaml-кода, а можно стандартным способом в файле отделенного кода.

Декларативное подключение:

```
<Button x:Name="Button1" Content="Click" Click="Button_Click" />
```

Подключим еще один обработчик в коде, чтобы при нажатии на кнопку срабатывали сразу два обработчика:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        Button1.Click += Button1_Click;
    }
```

```
    // обработчик, подключаемый в XAML
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Hi from Button_Click");
    }
    // обработчик, подключаемый в конструкторе
    private void Button1_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Hi from Button1_Click");
    }
}
```

Большинство событий клавиатуры (KeyUp/PreviewKeyUp, KeyDown/PreviewKeyDown) принимает в качестве аргумента объект KeyEventArgs, у которого можно отметить следующие свойства:

- *Key* позволяет получить нажатую или отпущенную клавишу;
- *SystemKey* позволяет узнать, нажата ли системная клавиша, например, Alt;
- *KeyboardDevice* получает объект *KeyboardDevice*, представляющее устройство клавиатуры;
- *IsRepeat* указывает, что клавиша удерживается в нажатом положении;
- *IsUp* и *IsDown* указывает, была ли клавиша нажата или отпущена;
- *IsToggled* указывает, была ли клавиша включена – относится только к включаемым клавишам Caps Lock, Scroll Lock, Num Lock.

2.4 Команды

В WPF кроме обработки событий приложение может взаимодействовать с пользователем с помощью команд. Команды представляют механизм выполнения какой-нибудь задачи, например, копирования текста – когда нажимаем Ctrl+C, то мы копируем текст в буфер. В процессе копирования выполняется ряд действий, и все вместе эти действия объединяются в одну команду. Использование команд помогает сократить объем кода и использовать одну и ту же команду для нескольких элементов управления в различных местах программы. Таким образом, команды позволяют абстрагировать набор действий от конкретных событий конкретных элементов.

Модель команд в WPF состоит из четырех аспектов:

- сама *команда*, которая представляем выполняемую задачу;
- *привязка* команд, которая связывает команду с определенной логикой приложения;
- *источник* команды – элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда);
- **цель** команды – элемент интерфейса, на котором выполняется команда.

Все команды реализуют интерфейс System.Window.Input.ICommand:

```
public interface ICommand
{
    event EventHandler CanExecuteChanged;
```

```
    void Execute (object parameter);
    bool CanExecute (object parameter);
}
```

Метод *Execute* предназначен для хранения логики команды. Функция *CanExecute* возвращает true, если команда включена и доступна для использования, и false, если команда отключена. Событие *CanExecuteChanged* вызывается при изменении состояния команды.

В WPF этот интерфейс реализован встроенным классом *System.Windows.Input.RoutedCommand*, который является базовым для всех встроенных команд. Поэтому, если нам потребуется создать свой класс команды, мы можем либо реализовать *ICommand*, либо унаследовать свой класс команды от *RoutedCommand*.

WPF уже обладает большим набором встроенных команд. Все они представляют объекты класса *RoutedUICommand*, который является производным от *RoutedCommand*.

Лабораторная работа 4

Технология WPF. Создание Windows-приложения

1 Запустите **Microsoft Visual Studio**.

2 Создайте новый проект **WPF** в своей папке.

3 Изучить структуру проекта и вид окна редактора.

По умолчанию студия создает и открывает два файла: файл декларативной разметки интерфейса MainWindow.xaml и файл связанного с разметкой кода MainWindow.xaml.cs. Файл MainWindow.xaml имеет два представления: визуальное – в режиме WYSIWIG отображает весь графический интерфейс данного окна приложения, и под ним декларативное объявление интерфейса в XAML.

XAML (eXtensible Application Markup Language) – язык разметки, используемый для инициализации объектов в технологиях на платформе .NET. Применительно к WPF данный язык используется, прежде всего, для создания пользовательского интерфейса декларативным путем.

Файл App.xaml и связанный с ним файл кода App.xaml.cs – это глобальные файлы для всего приложения. App.xaml задает файл окна программы, которое будет открываться при запуске приложения. Если вы откроете этот файл, то можете найти в нем строку StartupUri="MainWindow.xaml" – то есть при запуске приложения, будет создаваться интерфейс из файла MainWindow.xaml.

Примечание: подробную информацию о технологии WPF можно найти на сайте: https://metanit.com/sharp/wpf/

4 Добавить на форму, полученную по умолчанию при создании проекта, кнопку. Задать внешний вид формы и кнопки.

5 Создать Windows-приложение «Текстовый редактор» в котором предусмотреть:

6 Главное меню, позволяющее выбрать команды «Создать новый текст», «Открыть текст», «Сохранить», «Выход».

7 Многооконный интерфейс, дочернее окно которого, использует объект Memo в качестве текстового редактора.

8 Запустите программу на выполнение и протестируйте полученное приложение.

3 Делегаты и анонимные методы

3.1 Определение и пример использования делегата

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты – это указатели на методы, с помощью которых можно вызвать данные методы.

Общий шаблон объявления делегата:

```
delegate тип имя (аргументы);
```

Например:

```
delegate void Message();  // Объявляем делегат.
Message mes; // Создаем переменную делегата.

void print() { Console.WriteLine("Hello world"); }
mes = new Message(print); // Присваиваем этой переменной адрес метода.
```

или так:

```
mes = print; // Присваиваем этой переменной адрес метода.
```

Затем через делегат вызываем метод, на который ссылается данный делегат:

```
mes(); // Вызываем метод.
```

Делегаты могут указывать на методы, которые определены в том же классе, где определена переменная делегата, а также на методы из других классов и структур. Для нестатического метода обращение выполняется через объект, а для статического метода – через имя класса.

Пример 1. Вычислить значение функции $y = f(x) = \begin{cases} 2x + \cos(2x), & x < 0, \\ 3x + \sin(3x), & x \geq 0; \end{cases}$ При заданном значении аргумента.

```
delegate double Calculate(double argument);

private void CalculateClick(object sender, EventArgs e)
{
    if (double.TryParse(textBox1.Text, out double x))
    {
        if (x < 0) textBox2.Text = GetResult(Math.Cos, 2 * x);
        else textBox2.Text = GetResult(Math.Sin, 3 * x);
    }
    else
    {
        textBox2.Text = "";
        MessageBox.Show("Данные введены не верно", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

string GetResult(Calculate calculate, double arg)
{
    return (arg + calculate(arg)).ToString("1.##");
}
```

3.2 Добавление методов в делегат и удаление их из делегата

В примере выше переменная делегата указывала на один метод. В реальности делегат может указывать на множество методов, которые имеют ту же сигнатуру и возвращаемые тип. Все методы в делегате попадают в специальный список вызова или invocation list. И при вызове делегата все методы из этого списка последовательно вызываются.

Для добавления делегатов применяется операция +=. Однако стоит отметить, что в реальности будет происходить создание нового объекта делегата, который получит методы старой копии делегата и новый метод, и новый созданный объект делегата будет присвоен переменной mes1.

При добавлении делегатов следует учитывать, что можно добавить ссылку на один и тот же метод несколько раз, и в списке вызова делегата тогда будет несколько ссылок на один и то же метод. Соответственно при вызове делегата добавленный метод будет вызываться столько раз, сколько он был добавлен:

```
Message mes1 = Hello;
mes1 += HowAreYou;
mes1 += Hello;
mes1 += Hello;
mes1();
```

При удалении методов из делегата фактически будет создаваться новый делегат, который в списке вызова методов будет содержать на один метод меньше.

Лабораторная работа 5

Решение задач с использованием делегатов

1 Запустите **Microsoft Visual Studio**.
2 Создайте новый проект **WPF** в своей папке.
3 Добавьте на форму **splitContainer** – контейнер, содержащий две панели для решения двух задач. На каждой панели разместить:
- **groupBox** с переключателями, определяющими методы, вызываемые с помощью делегата;
- **текстовые поля** для ввода и вывода данных;
- **метки** для подписей;
- **кнопку** для вызова обработчика события.

4 Добавьте в проект классы для решения задач.
5 Опишите методы для проверки корректности вводимых данных.
6 В кодовом файле формы опишите два делегата для вызова соответствующих методов.
7 Опишите обработчики событий для вызова методов через делегаты, в зависимости от выбранного с помощью переключателей критерия.
8 Задайте очистку предыдущих результатов при выборе нового переключателя.
9 Запустите программу на выполнение и протестируйте полученное приложение.

Задача 1. Выполнить вычисления с использованием одной из функций:
- $f(x) = \sqrt[3]{2x-1} + 5$;
- $f(x) = \ln(2x^3 + 3) + \sqrt{|1 - 2x^2|}$;
- $f(x) = \sin\left(\frac{3\pi}{2x} + x\right)$.

Примечание: методы для вычисления должны быть описаны в статическом классе.

Задача 2. Вычислять различные характеристики шара в зависимости от выбранного критерия.

Примечания:
- Создайте класс **Sphere** (шар). В классе должны быть описаны конструктор и четыре метода для вычисления длины окружности и площади круга, полученные в центральном сечении, площади поверхности и объема шара.
- Класс и методы должны быть не статические.

4 Основы LINQ

LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных. В качестве источника данных может выступать объект, реализующий интерфейс IEnumerable (например, стандартные

коллекции, массивы), набор данных DataSet, документ XML. Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.

Существует несколько разновидностей LINQ:

- **LINQ to Objects**: применяется для работы с массивами и коллекциями;
- **LINQ to Entities**: используется при обращении к базам данных через технологию Entity Framework;
- **LINQ to SQL**: технология доступа к данным в MS SQL Server;
- **LINQ to XML**: применяется при работе с файлами XML;
- **LINQ to DataSet**: применяется при работе с объектом DataSet;
- **Parallel LINQ (PLINQ)**: используется для выполнения параллельных запросов.

4.1 LINQ to Objects

Чтобы использовать функциональность LINQ, нужно в кодовом файле подключить пространство имен *System.LINQ*.

Простейшее определение запроса LINQ выглядит следующим образом:

```
from переменная in набор_объектов
select переменная;
```

Выражение `from t in teams` проходит по всем элементам массива `teams` и определяет каждый элемент как `t`. Используя переменную `t`, можно проводить над ней разные операции.

Несмотря на то, что мы не указываем тип переменной `t`, выражения `LINQ` являются строго типизированными. То есть среда автоматически распознает, что набор `teams` состоит из объектов `string`, поэтому переменная `t` будет рассматриваться в качестве строки.

Далее с помощью оператора `where` проводится фильтрация объектов, и если объект соответствует критерию (в данном случае начальная буква должна быть "Б"), то этот объект передается дальше.

Оператор `orderby` упорядочивает по возрастанию, то есть сортирует выбранные объекты.

Оператор `select` передает выбранные значения в результирующую выборку, которая возвращается `LINQ`-выражением.

В данном случае результатом выражения `LINQ` является объект `IEnumerable<T>`. Нередко результирующая выборка определяется с помощью ключевого слова `var`, тогда компилятор на этапе компиляции сам выводит тип.

Преимуществом подобных запросов также является и то, что они интуитивно похожи на запросы языка `SQL`, хотя и имеют некоторые отличия.

4.2 Методы расширения LINQ

Кроме стандартного синтаксиса `from .. in .. select` для создания запроса `LINQ` мы можем применять специальные методы расширения, которые

определены для интерфейса IEnumerable. Как правило, эти методы реализуют ту же функциональность, что и операторы LINQ типа where или orderby.

Например:

```
string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

var selectedTeams = teams.Where(t=>t.ToUpper().StartsWith("Б")).OrderBy(t => t);

foreach (string s in selectedTeams)
    Console.WriteLine(s);
```

Запрос `teams.Where(t=>t.ToUpper().StartsWith("Б")).OrderBy(t => t)` будет аналогичен предыдущему. Он состоит из цепочки методов Where и OrderBy. В качестве аргумента эти методы принимают делегат или лямбда-выражение.

Не каждый метод расширения имеет аналог среди операторов LINQ, но в этом случае можно сочетать оба подхода. Например, используем стандартный синтаксис linq и метод расширения Count(), возвращающий количество элементов в выборке:

```
int number = (from t in teams where t.ToUpper().StartsWith("Б") select t).Count();
```

4.3 Фильтрация

Для выбора элементов из некоторого набора по условию используется метод Where. Например, выберем все четные элементы, которые больше 10.

Фильтрация с помощью операторов LINQ:

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };

IEnumerable<int> evens = from i in numbers
                         where i%2==0 && i>10
                         select i;
foreach (int i in evens)
    Console.WriteLine(i);
```

Здесь используется конструкция from: `from i in numbers`

Тот же запрос с помощью метода расширения:

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
IEnumerable<int> evens = numbers.Where(i => i % 2 == 0 && i > 10);
```

Если выражение в методе Where для определенного элемента будет равно true (в данном случае выражение i % 2 == 0 && i > 10), то данный элемент попадает в результирующую выборку.

Лабораторная работа 6

Обработка данных с использованием LINQ

Задание. Выполнить отбор (фильтрацию) в зависимости от заданных критериев и обработку данных, содержащихся в тестовом файле. Результаты фильтрации записать в новые текстовые файлы или вывести на форму.

Используемые технологии и инструменты: WPF (Windows Presentation Foundation), LINQ (Language-Integrated Query), библиотеки для работы с текстовыми файлами StreamReader и StreamWriter.

1 В редакторе создайте текстовый файл, содержащий данные о студентах: номер студенческого билета, фамилию, имя, отчество студента, дату рождения, домашний адрес, номер группы, оценки за четыре экзамена последней сессии. Данные разделить «точкой с запятой».
2 Создайте новый проект **WPF** в своей папке.
3 Создайте класс **Student** для описания данных из текстового файла.
4 Создайте обработчик события при нажатии на кнопку, в котором реализовать:
   – создание списка объектов из данных, содержащихся в исходном файле;
   – отбор данных о должниках и запись в новый файл: ФИО студента, номер группы и оценки за экзамены (в качестве разделителя в файле использовать пробелы так, чтобы данные выводились в столбики);
   – вычисление возраста студентов и запись в новый файл личных данных студентов, которые младше 20 лет, в отсортированном по возрастанию виде.
5 Добавьте на форму компоненты и в код обработчик события для решения следующей задачи: по введенному номеру группы вывести на форму данные о студентах (номер студенческого билета и ФИО).
6 Создайте шаблоны документов Word для вывода в них результатов отбора данных, выполненных в п.4.
7 Запустите программу на выполнение и протестируйте полученное приложение.

5 Работа с документами Word и Excel

5.1 Введение в технологию взаимодействия с COM-объектами

Компонентная модель объектов или технология COM (Component Object Model) предоставляет возможность одной программе (клиенту) работать с объектом другой программы (сервером). Технология COM упрощает создание программ, обеспечивая их совместимость в разных версиях платформы Windows и относительную независимость от языка программирования (компоненты COM могут создаваться на разных языках и далее внедряться в приложении путем использование стандартного интерфейса).

COM-приложение может содержать от одного до нескольких внедренных объектов, а каждый объект может содержать один или несколько интерфейсов, каждый из которых содержит методы объекта. Внешняя программа, используя

методы объекта, выполняет необходимые действия по обмену данными с сервером, не беспокоясь об особенностях внутренней реализации. Сервером может быть либо исполняемый файл, либо библиотека.

Основным недостатком использования COM-объектов являлась необходимость хранения и регистрации в системном реестре всей информация обо всех COM компонентах.

Сборка (assembly) – это совокупность файлов, устанавливаемых на компьютер и необходимых для выполнения программного кода приложения (среда выполнения – NET Runtime Framework не входит в понятие сборки). В большинстве случаев создаются однофайловые сборки, состоящие из одного EXE или DLL-файла.

"Раннее связывание" – когда COM объекты добавляются в решение, и в целевой сборке присутствуют их dll файлы.

Вначале добавим файлы библиотек для Word и Excel. Для этого выбираем мышкой в дереве структуры решения узел Ссылки. Далее, правым кликом мышки входим в контекстное меню и выбираем пункт Добавить ссылку... При его выборе появляется диалоговое окно с тремя закладками.

Сборки – содержит все доступные .NET компоненты.

COM – содержит все доступные COM компоненты.

Проекты – содержит все компоненты многократного использования, созданные разработчиком (совокупность проектов решения).

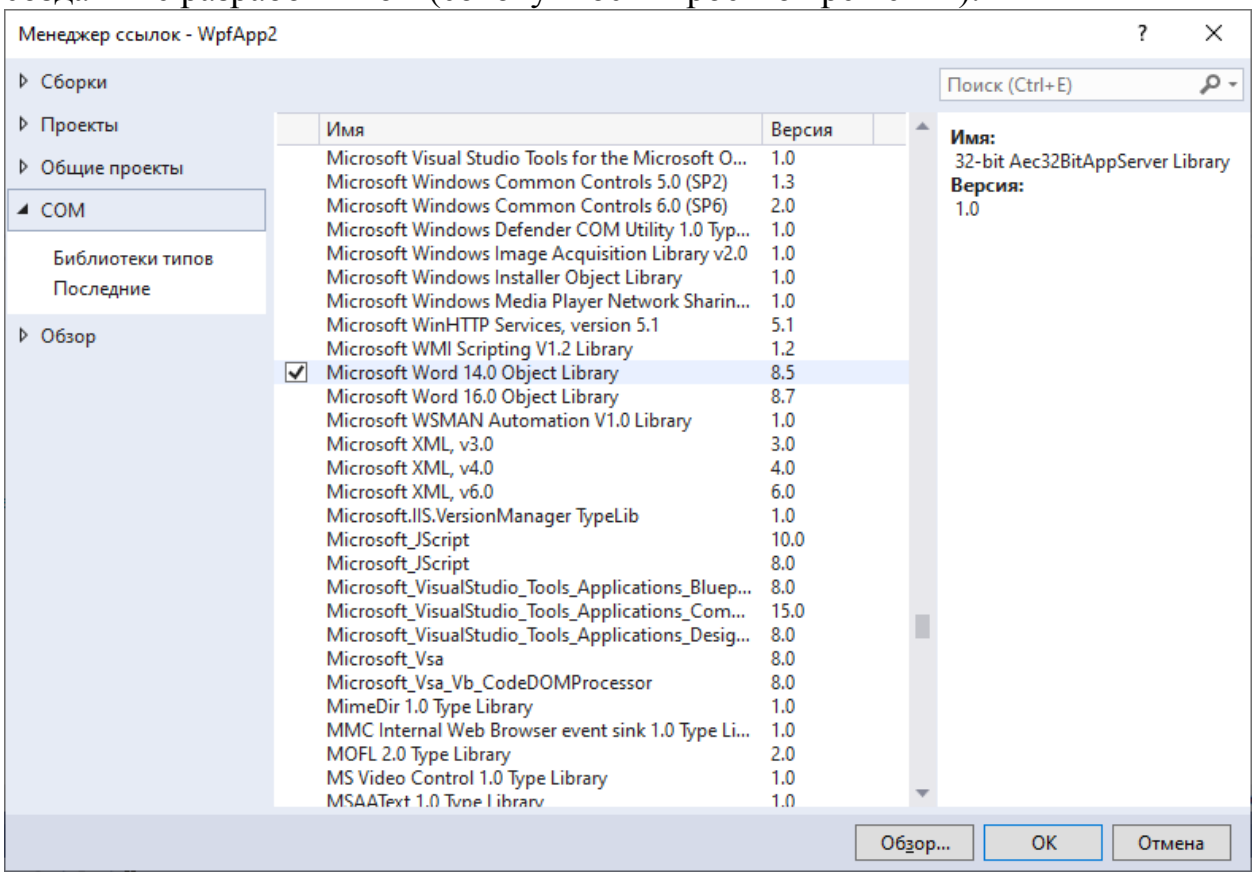


Рисунок 20 – Дерево структуры решения

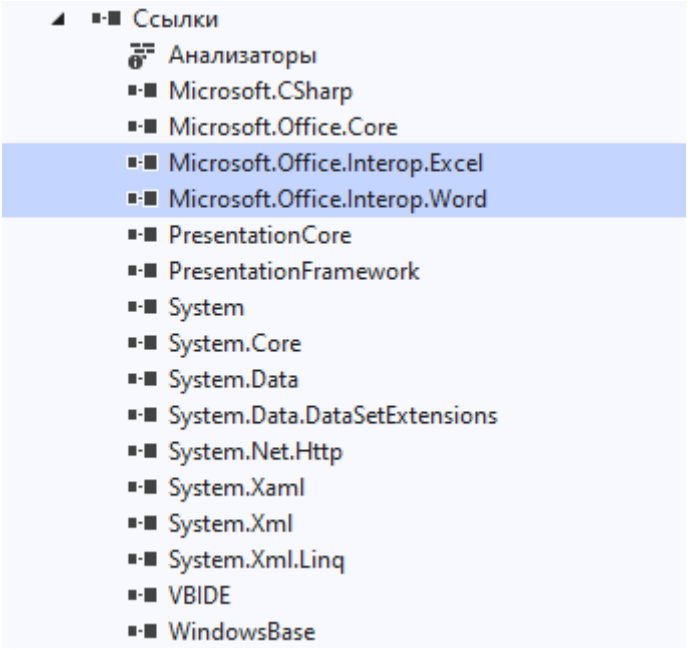


Рисунок 21 – Добавление в ссылки библиотек

NuGet Manager – это инструмент, который помогает разработчику управлять сторонними библиотеками (пакетами) в своем проекте.

Под управлением подразумевается поиск, скачивание, установка, настройка, обновление и удаление файлов сторонних разработчиков у себя в приложении.

С помощью **NuGet** можно добавить в проект ресурсы для работы с библиотеками Word и Excel.

Для удобства обращения к свойствам и методам нужно в код проекта добавить:

```
using Excel = Microsoft.Office.Interop.Excel;
using Word  = Microsoft.Office.Interop.Word;
```

Для запуска приложений Word и Excel используются команды:

```
Word.Application wordapp = new Word.Application();
 wordapp.Visible=true;
 Excel.Application excelapp = new Excel.Application();
 excelapp.Visible = true;
```

5.2 Работа с Word-документами

Создание класса, содержащего методы для открытия документа, его редактирования и сохранения.

```
class WordDocument
{
    Word.Application wordApp;

    public WordDocument()
    {
        try
        {
            wordApp = new Word.Application(); // Доступ к ресурсу.
            wordApp.Visible = false;  // Сделать окна невидимыми.
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
        }
    }

    /// <summary>
    /// Открытие документа для редактирования.
    /// </summary>
    /// <param name="name">Имя документа.</param>
    public void OpenDoc(string name)
    {
        try
        {
            //Word.Document wordDocument;
            object documentTemplate = name;// Путь доступа к шаблону документа.
            wordApp.Documents.Open(documentTemplate);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
        }
    }

   /// <summary>
   /// Создание нового документа на основе стандартного шаблона и его сохранение.
   /// </summary>
   /// <param name="newName">Новое имя документа.</param>
    public void AddDoc(string newName)
    {
        try
        {
            Word.Document wordDocument = wordApp.Documents.Add();
            wordDocument.SaveAs(newName);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
        }
    }
```

```
/// <summary>
/// Создание нового документа на основе существующего документа и его сохранение.
/// </summary>
/// <param name="name">Имя открываемого документа-шаблона.</param>
/// <param name="newName">Новое имя документа.</param>
public void AddDoc(string name, string newName)
{
    try
    {
        object documentTemplate = name;// Путь доступа к документу.
        Word.Document wordDocument = wordApp.Documents.Add(documentTemplate);
        wordDocument.SaveAs(newName);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
    }
}
```

```
/// <summary>
/// Закрытие документов.
/// </summary>
public void CloseDoc()
{
    try
    {
        wordApp.Visible = true;  // Сделать окна видимыми.
        Object saveChanges = WdSaveOptions.wdPromptToSaveChanges;
        wordApp.Quit(saveChanges);
        wordApp = null;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
    }
}
```

```
    /// <summary>
    /// Замена текстовых шаблонов в документе.
    /// </summary>
    /// <param name="stubToReplace">Текст, который заменяем.</param>
    /// <param name="text">Текст для замены.</param>
    /// <param name="nameDoc">Имя документа.</param>
    public void ReplaceWordStub(string stubToReplace, string text, string nameDoc)
    {
        try
        {
            bool fl;
            Word.Document wordDocument = (Word.Document)wordApp.Documents.get_Item(nameDoc);
            do
            {
                // Выполнить все замены.
                var range = wordDocument.Content;

                fl = range.Find.Execute(FindText: stubToReplace, ReplaceWith: text);
            } while (fl);
            wordDocument.Save();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
        }
    }

    /// <summary>
    /// Заполнение таблицы.
    /// </summary>
    /// <param name="number">Номер таблицы.</param>
    /// <param name="student">Объект класса.</param>
    /// <param name="nameDoc">Имя файла.</param>
    public void FillTable(int number, IEnumerable<Student> students, string nameDoc)
    {
        try
        {
            Word.Document wordDocument = (Word.Document)wordApp.Documents.get_Item(nameDoc);
            Word.Table table = wordDocument.Tables[number]; // Получить доступ к таблице.
            int i = 2;
            foreach (var student in students)
            {
                if (i > 2) table.Rows.Add();
                table.Rows[i].Cells[1].Range.Text = student.Fam;
                table.Rows[i].Cells[2].Range.Text = student.Name;
                table.Rows[i].Cells[3].Range.Text = student.Age.ToString();
                i++;
            }
            wordDocument.Save();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message); // Нужно будет заменить на делегат.
        }
    }
}
```

5.3 Работа с Excel-документами

Создание псевдонима для работы с Excel:

```
using Excel = Microsoft.Office.Interop.Excel;
 //Объявляем приложение
 Excel.Application ex = new Microsoft.Office.Interop.Excel.Application();

 //Отобразить Excel
 ex.Visible = true;

 //Количество листов в рабочей книге
 ex.SheetsInNewWorkbook = 2;

 //Добавить рабочую книгу
 Excel.Workbook workBook = ex.Workbooks.Add(Type.Missing);

 //Отключить отображение окон с сообщениями
 ex.DisplayAlerts = false;

 //Получаем первый лист документа (счет начинается с 1)
 Excel.Worksheet sheet = (Excel.Worksheet)ex.Worksheets.get_Item(1);

 //Название листа (вкладки снизу)
 sheet.Name = "Отчет за 13.12.2017";

 //Пример заполнения ячеек
 for (int i = 1; i <= 9; i++)
 {
 for (int j = 1; j < 9; j++)
 sheet.Cells[i, j] = String.Format("Boom {0} {1}", i, j);
 }

 //Захватываем диапазон ячеек
 Excel.Range range1 = sheet.get_Range(sheet.Cells[1, 1], sheet.Cells[9,
 9]);

 //Шрифт для диапазона
 range1.Cells.Font.Name = "Tahoma";
 //Размер шрифта для диапазона
 range1.Cells.Font.Size = 10;

 //Захватываем другой диапазон ячеек
 Excel.Range range2 = sheet.get_Range(sheet.Cells[1, 1], sheet.Cells[9,
 2]);
 range2.Cells.Font.Name = "Times New Roman";

 //Задаем цвет этого диапазона. Необходимо подключить System.Drawing
 range2.Cells.Font.Color = ColorTranslator.ToOle(Color.Green);
 //Фоновый цвет
 range2.Interior.Color = ColorTranslator.ToOle(Color.FromArgb(0xFF, 0xFF,
 0xCC));
```

```
//Расставляем рамки со всех сторон:
range2.Borders.get_Item(Excel.XlBordersIndex.xlEdgeBottom).LineStyle =
Excel.XlLineStyle.xlContinuous;
range2.Borders.get_Item(Excel.XlBordersIndex.xlEdgeRight).LineStyle =
Excel.XlLineStyle.xlContinuous;
range2.Borders.get_Item(Excel.XlBordersIndex.xlInsideHorizontal).LineSty
le = Excel.XlLineStyle.xlContinuous;
range2.Borders.get_Item(Excel.XlBordersIndex.xlInsideVertical).LineStyle
= Excel.XlLineStyle.xlContinuous;
range2.Borders.get_Item(Excel.XlBordersIndex.xlEdgeTop).LineStyle =
Excel.XlLineStyle.xlContinuous;
//Цвет рамки можно установить так:
range2.Borders.Color = ColorTranslator.ToOle(Color.Red);
//Выравнивания в диапазоне задаются так:
rangeDate.VerticalAlignment = Excel.XlVAlign.xlVAlignCenter;
rangeDate.HorizontalAlignment = Excel.XlHAlign.xlHAlignLeft;
//Для начала снова получим диапазон ячеек:
Excel.Range formulaRange = sheet.get_Range(sheet.Cells[4, 1],
sheet.Cells[9, 1]);
//Далее получим диапазон вида A4:A10 по адресу ячейки ( [4,1]; [9;1] )
описанному выше:
string adder = formulaRange.get_Address(1, 1,
Excel.XlReferenceStyle.xlA1, Type.Missing, Type.Missing);
//Теперь в переменной adder у нас хранится строковое значение диапазона
( [4,1]; [9;1] ),
//то есть A4:A10.
//Вычисляем формулу:
//Одна ячейка как диапазон
Excel.Range r = sheet.Cells[10, 1] as Excel.Range;
//Оформления
r.Font.Name = "Times New Roman";
r.Font.Bold = true;
r.Font.Color = ColorTranslator.ToOle(Color.Blue);
//Задаем формулу суммы
r.Formula = String.Format("=СУММ({0}", adder);
//Выделение ячейки или диапазона ячеек
//Так же можно выделить ячейку или диапазон, как если бы мы выделили их
мышкой:
sheet.get_Range("J3", "J8").Activate();
//или
sheet.get_Range("J3", "J8").Select();
//Можно вписать одну и ту же ячейку, тогда будет выделена одна ячейка.
sheet.get_Range("J3", "J3").Activate();
sheet.get_Range("J3", "J3").Select();
Чтобы настроить авто ширину и высоту для диапазона, используем такие
команды:
```

```
range.EntireColumn.AutoFit();
range.EntireRow.AutoFit();
//Чтобы получить значение из ячейки, используем такой код:
//Получение одной ячейки как ранга
Excel.Range forYach = sheet.Cells[ob + 1, 1] as Excel.Range;
//Получаем значение из ячейки и преобразуем в строку
string yach = forYach.Value2.ToString();
//Добавляем лист в рабочую книгу
//Чтобы добавить лист и дать ему заголовок, используем следующее:
var sh = workBook.Sheets;
Excel.Worksheet sheetPivot = (Excel.Worksheet)sh.Add(Type.Missing,
sh[1], Type.Missing, Type.Missing);
sheetPivot.Name = "Сводная таблица";
//Добавление разрыва страницы
//Ячейка, с которой будет разрыв
Excel.Range razr = sheet.Cells[n, m] as Excel.Range;
//Добавить горизонтальный разрыв (sheet - текущий лист)
sheet.HPageBreaks.Add(razr);
//VPageBreaks - Добавить вертикальный разрыв
//Сохраняем документ
ex.Application.ActiveWorkbook.SaveAs("doc.xlsx", Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Excel.XlSaveAsAccessMode.xlNoChange,
Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing);
//Как открыть существующий документ Excel
ex.Workbooks.Open(@"C:\Users\Myuser\Documents\Excel.xlsx",
Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing);
```

Лабораторная работа 7

Создание и редактирование документов формата .docx

1. Откройте проект, созданный в лабораторной работе 6.
2. Добавьте в него класс **WordDocument** для работы с текстовыми документами.
3. Создайте шаблоны документов в формате .docx, в которые должны быть выведены данные, полученные в результатов выборок, полученных в лабораторной работе 6.
4. Спроектируйте пользовательский интерфейс.
5. Добавьте элементы управления на форму и дополнить программный код обработчиками событий для создания экземпляра класса **WordDocument** и заполнения документов.

6 Запустите программу на выполнение и протестируйте полученное приложение.

Лабораторная работа 8

Работа с таблицами и табличными документами

1 Откройте проект, созданный в лабораторных работах 6-7.
2 Создать класс **ExcelDocument** для работы с табличными документами.
3 Спроектируйте пользовательский интерфейс.
4 Добавить элементы управления на форму и дополнить программный код обработчиками событий для создания экземпляра класса **WordDocument** и заполнения документов.
5 Создайте документы в формате .docx на основе результатов выборок, полученных в лабораторной работе 6.
6 Запустите программу на выполнение и протестируйте полученное приложение.

Лабораторная работа 9

Создание Windows приложения с использованием библиотек классов

При помощи Visual Studio на языке C# можно создавать библиотеки DLL, которые могут быть вызваны другими управляемыми приложениями и неуправляемым кодом. Заключение часто используемой функции DLL в оболочку управляемого класса представляет собой эффективный способ инкапсуляции функциональных возможностей платформы. Хотя применение оболочки класса необязательно, оно удобно, так как процесс определения функций DLL может быть громоздким и представлять собой источник ошибок. При программировании на C# функции DLL должны объявляться в пределах класса или модуля.

В классе для каждой вызываемой функции DLL должен быть определен статический метод. Определение может включать дополнительные сведения, например, кодировку или соглашение о вызовах, используемое при передаче аргументов метода. Если эти сведения отсутствуют, используются параметры по умолчанию. Полный список параметров объявлений и их значения по умолчанию.

После упаковки в оболочку методы для функции можно вызывать точно так же, как и методы для любой другой статической функции. Вызов неуправляемого кода обрабатывает базовую экспортируемую функцию автоматически. При разработке управляемого класса для вызова неуправляемого кода следует учитывать связи между классами и функциями DLL. Например, разработчик может выполнять следующие действия:

- объявлять функции DLL в существующем классе;
- создавать для каждой функции DLL отдельный класс в целях изоляции и упрощения поиска функций;

– создавать один класс для набора связанных функций DLL, формируя логические группирования и сокращая дополнительные издержки.

Разработчик может назвать класс и его методы по своему усмотрению.

Библиотека динамической компоновки (DLL) связывается с программой во время выполнения. Построение и использование библиотеки DLL рассматривается в следующем сценарии:

MathLibrary.DLL: Файл библиотеки с методами, вызываемыми во время выполнения. В этом примере библиотека DLL содержит два метода: Add и Multiply.

Add.cs: Исходный файл с методом Add(long i, long j). Он возвращает сумму своих параметров. Класс AddClass с методом Add является членом пространства имен UtilityMethods.

Mult.cs: Исходный код, содержащий метод Multiply(long x, long y). Он воз-вращает результат своих параметров. Класс MultiplyClass с методом Multiply также является членом пространства имен UtilityMethods.

TestCode.cs: Файл с методом Main. Он использует методы в DLL-файле для вычисления суммы и результата аргументов времени выполнения.

```
// File: Add.cs
namespace UtilityMethods
{
   public class AddClass
   {
      public static long Add(long i, long j) => i + j;
   }
}
```

Задание: создать оконное приложение для проведения опроса с использованием библиотек классов.

1 Опрос должен осуществляться на форме библиотеки DLL, все данные сохраняются в текстовый файл.

2 Опрос должен содержать следующие поля: «Имя», «Фамилия» – «обязательные», реализованные с помощью TextBox; «Пол» – «обязательное» реализованное с помощью RadioButton; «Образование» – «обязательное» реализованное с помощью ComboBox; «Возраст», «Социальный статус», «Увлечения» – «необязательные» реализованные с помощью Edit. Для «необязательных» пунктов использовать флажок CheckBox, указывающий, хочет вводить пользователь этот пункт или нет.

3 Просмотр данных всех опросов осуществляется из главного приложения с помощью Memo.

4 Спроектируйте пользовательский интерфейс в соответствии с требованиями п. 2-3.

5 Введите программный код.

6 Запустите программу на выполнение и протестируйте полученное приложение.

6 Использование ORM Entity Framework для работы с базами данных

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- **Code first**: разработчик создает класс модели данных, которые будут храниться в БД, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы
- **Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных
- **Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.

Подход Code First очень прост и удобен. Но он также и очень гибкий. Так, вполне часто распространена ситуация, когда база данных уже имеется. И здесь опять же поможет Code First. Иногда программисты называют данный подход **Code Second**.

6.1 Создание модели

Для создания БД нужно разработать структуру классов, содержащих свойства, аналогичные полям будущей БД и добавить их в проект. Например, создадим следующие два класса.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public virtual Blog Blog { get; set; }
}
```

В них используются два свойства навигации (Blog.Posts и Post.Blog), которые являются виртуальными. Это включает функцию отложенной загрузки Entity Framework. Отложенная загрузка означает, что содержимое этих свойств будет автоматически загружаться из базы данных при попытке доступа к ним.

6.2 Создание контекста

```
using System.Data.Entity;
```

Добавьте класс контекста в проекта.

```
public class BloggingContext : DbContext
```

```
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

6.3 Метод Load

В примере показано использование Load для загрузки фильтрованной коллекции связанных сущностей:

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Load the posts with the 'entity-framework' tag related to a given
blog
    context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();
}
```

Лабораторная работа 10

Создание модели и базы данных

1 Создайте проект оконного приложения.
2 Используя подход Code First, добавьте два класса Команды и Игроки, а также файл контекста приложения.
3 Настройте строку подключения к БД в файле App.config.
4 Добавьте форму для вывода данных.
5 Спроектируйте пользовательский интерфейс.
6 Запустите проект для создания базы данных на основе разработанных классов.
7 Откройте базу данных в программе Microsoft SQL Management Studio.
8 Введите данные в таблицы БД и выведите их на формы программы.
9 Запустите программу на выполнение и протестируйте полученное приложение.

Лабораторная работа 11

Ввод и редактирование данных в таблицах БД

1 Откройте проект, созданный в лабораторной работе 10.
2 Добавьте на форму компоненты, а в кодовый файл обработчики событий для добавления, изменения и удаления данных.
3 Создайте диалоговые окна для редактирования данных.
4 Запустите программу на выполнение и протестируйте полученное приложение.

Лабораторная работа 12

Сортировка и отбор данных по различным критериям

1 Откройте проект, созданный в лабораторных работах 10-11.
2 Добавьте в класс программируемые поля.
3 Задать два критерия для отбора данных.
4 Задайте критерии для сортировки данных.
5 Запустите программу на выполнение и протестируйте полученное приложение.

7 Динамические структуры данных

7.1 Стек

Стек представляет собой динамическую линейную структуру данных, которая работает по принципу LIFO (Last In First Out – "последний пришел – первый вышел"). Графически стек можно представить в виде столбика или стопки объектов:

Стек имеет вершину, который образует последний добавленный элемент. При добавлении новый элемент помещается поверх вершины стека и образует новую вершину. При удалении удаляется элемент из вершины стека, а предыдущий элемент образует новую вершину.

В библиотеке классов .NET есть свой класс, который выполняет роль стека – System.Collections.Generic.Stack<T> представляет коллекцию, которая использует алгоритм LIFO.

7.2 Бинарные деревья

Основными операциями с бинарными деревьями являются добавление элемента в дерево, удаление элемента и поиск элемента в дереве. Сложность каждой из этих операций зависит от сбалансированности дерева. Основным преимуществом двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки.

Данные (data) обладают ключом (key), на котором определена операция сравнения «меньше». В конкретных реализациях это может быть пара (key, value) – (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.

Двоичное дерево поиска применяется для построения более абстрактных структур, таких, как множества, мультимножества, ассоциативные массивы.

7.3 Балансирование дерева (АВЛ-дерево)

АВЛ-дерево – структура данных, изобретенная в 1968 году двумя советскими математиками: Евгением Михайловичем Ландисом и Георгием Максимовичем Адельсон-Вельским. Прежде чем дать конструктивное определение АВЛ-дереву, сделаем это для сбалансированного двоичного дерева поиска.

Сбалансированным называется такое двоичное дерево поиска, в котором высота каждого из поддеревьев, имеющих общий корень, отличается не более чем на некоторую константу k, и при этом выполняются условия характерные для двоичного дерева поиска.

АВЛ-дерево – сбалансированное двоичное дерево поиска с k=1. Для его узлов определен коэффициент сбалансированности (balance factor). Balance factor – это разность высот правого и левого поддеревьев, принимающая одно значение из множества {-1, 0, 1}. Ниже изображен пример АВЛ-дерева, каждому узлу которого поставлен в соответствие его реальный коэффициент сбалансированности.

Лабораторная работа 13

Структура данных стек. Реализация калькулятора на основе польской записи

1 Создайте новый проект.

Задача 1. Скобочная последовательность

Проверить корректность скобочного выражение при помощи стека.

Есть выражение, которое состоит из разного типа скобок. Правильным считается скобочное выражение, в котором каждая открывающаяся скобка закрывается скобкой такого же типа.

Эта задача используется в компиляторах языков программирования. Разбирают код программы с помощью алгоритма, который называется «автомат с магазинной памятью» (базируется на стеке).

Входные данные

Строка из символов **( ) [ ] { }** < >.

Выходные данные

Вывести, если скобочная последовательность корректная, иначе вывести **False**.

Примеры:

Входные данные
(()[{}()])
Выходные данные
True
Входные данные
(([])])
Выходные данные
False

Задача 2. Калькулятор с обратной польской записью

Найти значение выражения (если оно корректно), введенного в обратной польской записи (ОПЗ), содержащего знаки операций +, -, *, /.

Обратная польская запись – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как обратная бесскобочная запись, постфиксная нотация,

бесскобочная символика Лукасевича, польская инверсная запись, ПОЛИЗ. Выражения, преобразованные в ОПЗ, можно вычислять последовательно, слева направо.

Стековой машиной называется алгоритм, проводящий вычисления по обратной польской записи.

Входные данные

Строка из вещественных чисел и символов +, -, *, /.

Выходные данные

Число или сообщение о том, что данные не корректные.

Пример:

Входные данные

2 5 + 6,5 * 4 1 + – 8,2 /

Выходные данные

4,94

Лабораторная работа 14

Структуры данных: списки и деревья

1 Создать проект для построения оконного приложения.
2 Добавьте классы для работы с бинарным деревом поиска и с сбалансированным АВЛ-деревом.
3 Спроектируйте пользовательский интерфейс.
4 Добавьте на главную форму компоненты для создания объектов, проверки их работы и вывода результатов.
5 В кодовый файл введите соответствующие компонентам обработчики событий.
6 Запустите программу на выполнение и протестируйте полученное приложение.

8 Работа с графикой

8.1 Класс Graphics

Концепция графического интерфейса GDI+ несколько отличается от концепции «классического» графического интерфейса GDI, с которым привыкли иметь дело разработчики приложений Microsoft Windows. Прежде всего, это касается класса Graphics, реализующего в себе как свойства контекста отображения, так и инструменты, предназначенные для рисования в этом контексте.

Для того чтобы приложение могло что-нибудь нарисовать в окне, оно должно, прежде всего, получить или создать для этого окна объект класса Graphics. Далее, пользуясь свойствами и методами этого объекта, приложение может рисовать в окне различные фигуры или текстовые строки.

Имена большого количества методов, определенных в классе Graphics, начинается с префикса Draw* и Fill*. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур.

Например, метод DrawLine рисует линию, соединяющую две точки с заданными координатами:

```
public void DrawLine(Pen, Point, Point);
public void DrawLine(Pen, PointF PointF;
public void DrawLine(Pen, int, int, int, int);
public void DrawLine(Pen, float, float, float, float);
```

Первый параметр задает инструмент для рисования линии – перо. Перья создаются как объекты класса Pen, например:

```
Pen p = new Pen(Brushes.Black,2);
```

8.2 Рисование с помощью мыши

Текущее состояние мыши будет храниться в поле doDraw типа bool. Нужно добавить это поле в класс Form1, проинициализировав его следующим образом:

```
bool doDraw = false;
```

Создадим два обработчика событий MouseDown и MouseUp для формы Form1:

```
private void Form1_MouseDown(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  doDraw = true;
}

private void Form1_MouseUp(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  doDraw = false;
}
```

Чтобы создать эти обработчики событий, откройте окно визуального проектирования формы, а затем на вкладке событий щелкните дважды события MouseDown и MouseUp.

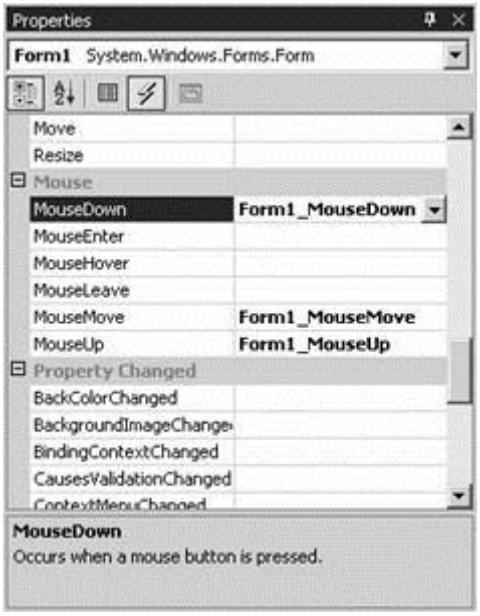


Рисунок 22 – Диалоговое окно для создания событий формы

Когда пользователь нажимает левую клавишу мыши, управление передается обработчику событий Form1_MouseDown. Этот обработчик записывает в поле doDraw значение true, отмечая таким способом тот факт, что пользователь приступил к процедуре рисования.

Нарисовав линию, пользователь отпускает левую клавишу мыши. При этом управление передается обработчику событий Form1_MouseUp, записывающему в поле doDraw значение false. Это означает завершение процедуры рисования.

Для отслеживание перемещения курсора мыши, нужно добавить обработчик события Form1_MouseMove:

```
private void Form1_MouseMove(object sender,
  System.Windows.Forms.MouseEventArgs e)
{
  if(doDraw)
  {
     Graphics g = Graphics.FromHwnd(this.Handle);
     SolidBrush redBrush = new SolidBrush(Color.Red);
     g.FillRectangle(redBrush, e.X, e.Y, 1, 1);
  }
}
```

Этот обработчик будет получать управление при всяком перемещении курсора мыши, причем свойства e.X и e.Y будут содержать новые координаты курсора.

Мы воспользуемся этим обстоятельством, нарисовав в месте нового расположения курсора квадрат, с шириной стороны в один пиксел. На экране такой квадрат будет выглядеть как точка. Рисование должно выполняться только в том случае, если в поле doDraw хранится значение true.

Лабораторная работа 15

Работа с графикой

1 Создать оконное приложение.

2 Расположить на форме компоненты: панель для построения изображения; элементы для определения параметров изображения; кнопку для очистки панели. Образец программы можно найти на сайте: Графическая «доска» на C# за 10 минут. | Примеры программ на языке C# (programming-csharp.ru)

3 Добавить обработчики событий в код программы.

4 Добавить на форму элементы управления (меню, переключатели или кнопки) для выбора типа объекта (например, прямоугольника, эллипса т.п.) для рисования.

5 Запустите программу на выполнение и протестируйте полученное приложение.

Список использованной литературы

1 Горелов, С.В. Современные технологии программирования: разработка Windows-приложений на языке C#: учебник для студентов, обучающихся по дисциплине «Современные технологии программирования», направление «Прикладная информатика» (09.03.03 — для бакалавров, 09.04.03 — для магистров) : в 2 томах : [16+] / С.В. Горелов ; под науч. ред. П.Б. Лукьянова ; Финансовый университет при Правительстве Российской Федерации. – Москва : Прометей, 2019. – Том 1. – 363 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=576037 (дата обращения: 24.01.2021). – Библиогр. в кн. – ISBN 978-5-907100-09-1. – Текст : электронный.

2 Горелов, С.В. Современные технологии программирования: разработка Windows-приложений на языке C#: учебник для студентов, обучающихся по дисциплине «Современные технологии программирования», направление «Прикладная информатика» (09.03.03 — для бакалавров, 09.04.03 — для магистров) : в 2 томах : [16+] / С.В. Горелов ; под науч. ред. П.Б. Лукьянова ; Финансовый университет при Правительстве Российской Федерации. – Москва : Прометей, 2019. – Том 2. – 379 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=576036 (дата обращения: 24.01.2021). – Библиогр. в кн. – ISBN 978-5-907100-18-3. – Текст : электронный.

3 Абрамян, А.В. Разработка пользовательского интерфейса на основе технологии Windows Presentation Foundation: учебник по курсу «Основы разработки пользовательского интерфейса» для студентов направления 02.03.02 «Фундаментальная информатика и информационные технологии» (бакалавриат) / А.В. Абрамян, М.Э. Абрамян ; Южный федеральный университет. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2018. – 302 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=499453 (дата обращения: 17.12.2020). – Библиогр.: с. 294. – ISBN 978-5-9275-2375-7. – Текст : электронный.

4 Рояк, М.Э. Программирование под Windows графических интерфейсов пользователя : учебное пособие : [16+] / М.Э. Рояк, И.М. Ступаков ; Новосибирский государственный технический университет. – Новосибирск : Новосибирский государственный технический университет, 2018. – 72 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=575018 (дата обращения: 24.01.2021). – Библиогр. в кн. – ISBN 978-5-7782-3754-4. – Текст : электронный.

5 Абрамян, М. Э. Технология LINQ на примерах. Практикум с использованием электронного задачника Programming Taskbook for LINQ : учебное пособие / М. Э. Абрамян. — Москва : ДМК Пресс, 2014. — 326

с. — ISBN 978-5-94074-981-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/66478 (дата обращения: 09.01.2021). — Режим доступа: для авториз. пользователей.
6 Подбельский, В. В. Язык декларативного программирования XAML / В. В. Подбельский. — Москва : ДМК Пресс, 2018. — 336 с. — ISBN 978-5-97060-573-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/111428 (дата обращения: 24.01.2021). — Режим доступа: для авториз. пользователей.
7 Дэвис, А. Асинхронное программирование в C# 5.0 / А. Дэвис. — Москва : ДМК Пресс, 2013. — 120 с. — ISBN 978-5-94074-886-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/9132 (дата обращения: 24.01.2021). — Режим доступа: для авториз. пользователей.
8 Документация по C# [режим доступа] https://docs.microsoft.com/ru-ru/dotnet/csharp/
9 Программирование на C# и .NET [режим доступа] https://metanit.com/sharp/