Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КАЛИНИНГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

А. А. Бабаева

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ ПРИЛОЖЕНИЙ ДЛЯ ОТКРЫТЫХ СИСТЕМ

Учебно-методическое пособие по выполнению
лабораторных работ для студентов специальности
10.05.03 Информационная безопасность автоматизированных систем

Калининград
Издательство ФГБОУ ВО «КГТУ»
2022

Рецензент
доцент кафедры информационной безопасности ФГБОУ ВО
«Калининградский государственный технический университет»
А.Г. Жестовский

Бабаева, А. А.
Технология построения защищенных приложений для открытых систем: учеб-методич. пособие по выполнению лабораторных работ для студентов специальности 10.05.03 Информационная безопасность автоматизированных систем / А. А. Бабаева. - Калининград: Изд-во ФГБОУ ВО «КГТУ», 2022. – 120 с.

Учебно-методическое пособие является руководством по выполнению лабораторных работ по дисциплине «Технология построения защищенных приложений для открытых систем» студентами, обучающимися по специальности 10.05.03 Информационная безопасность автоматизированных систем. Лабораторные работы предназначены для закрепления теоретического материала и приобретения практических навыков проектирования, создания, эксплуатации приложений для открытых систем и обеспечения их безопасности на всех этапах жизненного цикла.
Список лит. – 5 наименований

Учебно-методическое пособие рассмотрено и одобрено к использованию в качестве электронного методического материала кафедрой информационной безопасности 19 мая 2022 г., протокол № 7

Учебно-методическое пособие рекомендовано к использованию в учебном процессе в качестве электронного методического материала методической комиссией института цифровых технологий ФГБОУ ВО «Калининградский государственный технический университет» 28 июня 2022 г., протокол № 4

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

Данное учебно-методическое пособие предназначено для студентов специальности 10.05.03 Информационная безопасность автоматизированных систем, изучающих дисциплину «Технология построения защищенных приложений для открытых систем».

Цель настоящего лабораторного практикума: в результате освоения дисциплины ожидается, что студенты получат целостное представление о способах разработки распределенных приложений для открытых систем с использованием технологии Microsoft .NET, сформируют понятия о современных подходах к проектированию и построению, эксплуатации и модернизации защищенного программного обеспечения.

Лабораторный практикум содержит 11 лабораторных работ.

В результате выполнения лабораторных работ ожидается, что студенты сформируют навыки работы в среде Visual Studio с целью создания защищенных служб на языке C#, научатся создавать распределенные приложения для открытых систем и настраивать параметры безопасности в них для функциональных особенностей каждой службы, приобретут навыки использования транзакций в службах, публикации метаданных и различных способов размещения служб.

2. ЛАБОРАТОРНАЯ РАБОТА № 1. ОСНОВЫ КЛИЕНТ-СЕРВЕРНОГО ВЗАИМОДЕЙСТВИЯ

1. Общие сведения

Цель: укрепить знания в области распределенных вычислений и подготовиться к практике по созданию распределенного приложения в среде Visual Studio.

Материалы, оборудование, программное обеспечение:

Для выполнения данной лабораторной работы потребуется персональный компьютер с установленным на нем офисным пакетом Microsoft Office и доступ в сеть Интернет.

Условия допуска к выполнению:

Подготовить конспект по теоретической части задания.

Критерии положительной оценки: предоставить преподавателю отчет (конспект с ответами на вопросы) в электронном виде и устно ответить на два любых вопроса из списка – пройти защиту.

Планируемое время выполнения:

Аудиторное время выполнения (под руководством преподавателя): 2 ч.

Время самостоятельной подготовки: 1 ч.

2. Теоретическое введение

Контрольные вопросы для самопроверки: не предусмотрены.

3. Задание к лабораторной работе

Ответить на список вопросов, приведенный ниже. Для этого самостоятельно найти нужную литературу (при необходимости) или проанализировать информацию, изученную ранее на других дисциплинах.

Это поможет студенту освежить знания в области сетевых технологий и технологий построения распределенных приложений и подготовиться к созданию таких приложений в защищенном исполнении.

4. Методические указания и порядок выполнения работы

Ответить в электронном документе на следующие вопросы:

1. Какова роль сервера в клиент-серверном взаимодействии (виды архитектур, сравнение)?
2. Типы серверов (их виды и особенности использования).
3. Толщина клиент-серверных технологий (определение конфигурации, какое оборудование нужно использовать). Какое программное обеспечение определяет толщину клиента?
4. Виды сетевых протоколов и виды пакетов в них.
5. Принципы формирования структуры пакета (обмен информации по сетям).
6. Роль драйвера при сетевом подключении, роль драйверов сетевой карты.

5. Индивидуальное задание.

При защите лабораторной: вопросы 2 и 4 будут у каждого по вариантам. Необходимо изучить все протоколы, но при ответе выбрать один любой и рассказать про него.

6. Требования к отчету и защите

Подготовить отчет, разместить в ЭИОС и защитить у преподавателя.

Содержание отчета: титульный лист с указанием названия дисциплины, названия работы и фамилии студента, ответы на все вопросы из пункта 2.4.

Порядок защиты: устный ответ на 2 любых (выбирает преподаватель) вопроса из отчета + вопросы по вариантам. Защита проводится во время занятий.

3. ЛАБОРАТОРНАЯ РАБОТА № 2. ПРИМЕР ПРОГРАММИРОВАНИЯ СОБЫТИЯ КЛИКА НА КНОПКЕ В C# . РАЗРАБОТКА ПРОГРАММЫ ОПРЕДЕЛЕНИЯ ПЛОЩАДИ ПОВЕРХНОСТИ ШАРА

7. 3.1 Общие сведения

Цель: научиться программировать события в среде Visual Studio на языке C#, создавать событие клика на кнопке.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной работы №1.

Критерии положительной оценки: разработанная программа определения площади поверхности шара и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 1 ч.
Время самостоятельной подготовки: 1 ч.

8. 3.2 Теоретическое введение

Площадь поверхности шара вычисляется по формуле:

$$S = 4 \cdot \pi \cdot R^2 ,$$

где ***R*** – радиус шара, ***π*** – константа равная 3.1415.
В программе используются следующие элементы управления:

- Label – метка для вывода сообщений;
- Button – кнопка для выполнения расчета;
- TextBox – поле ввода, предназначенное для ввода значения *R*.

Контрольные вопросы для самопроверки: Каким образом мы создаем события клика? Какие это могут быть события? Что такое метод и как его реализовать?

9. 3.3 Задание к лабораторной работе

Составить программу, которая введенному радиусу R находит площадь поверхности шара. Программу реализовать как Windows Form Application.

10. 3.4 Методические указания и порядок выполнения работы

1. Запустить MS Visual Studio. Создать проект по шаблону Windows Forms Application. Сохранить проект под любым именем.

2. Размещение элементов управления на форме.

Из вкладки «Common Controls» выносим на форму четыре элемента управления (рисунок 1).

- два элемента управления типа Label (метка);
- элемент управления типа Button (кнопка);
- элемент управления типа TextBox (строка ввода).

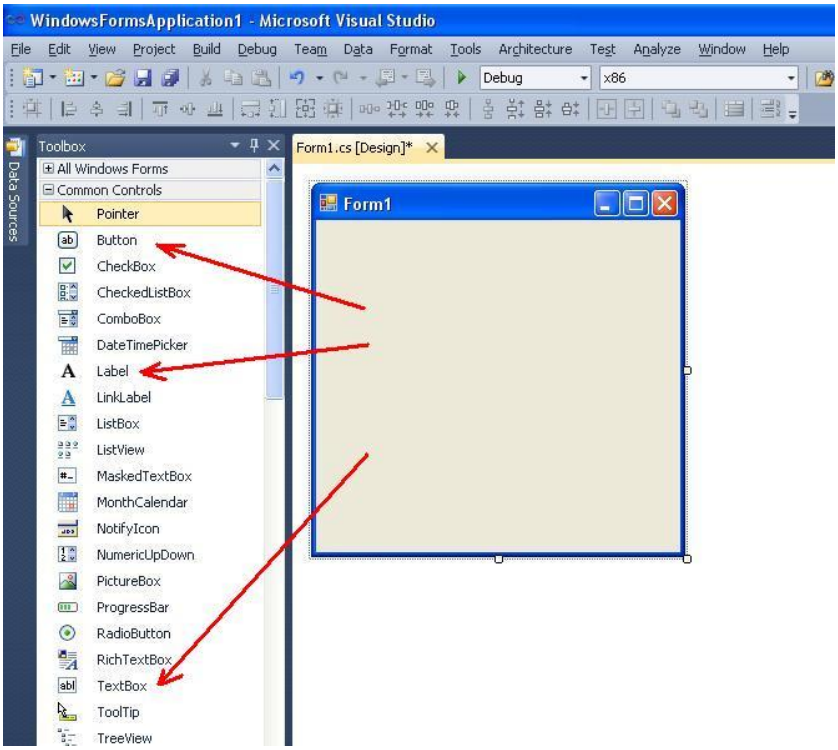


Рисунок 1 - Элементы управления Label, Button, TextBox

Автоматически создаются четыре объекта с именами label1, label2, button1 и textBox1. По этим именам можно будет иметь доступ к свойствам этих объектов.

Форма приложения будет иметь вид, как показано на рисунке 2.

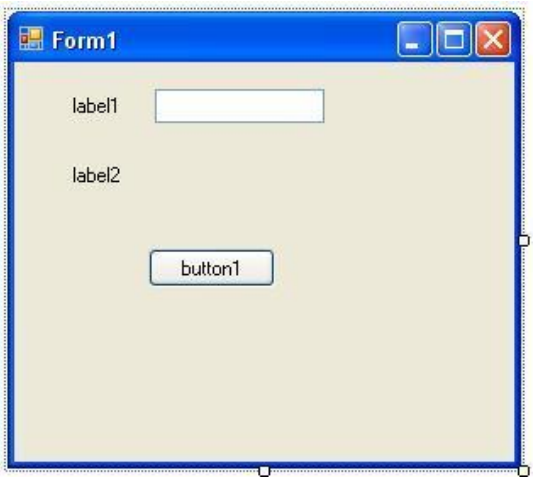


Рисунок 2 - Форма приложения после размещения label1, label2, button1 и textBox1

3. Настройка элементов управления типа Label.

Выделяем элемент управления (объект) label1. В палитре Toolbox изменяем свойство Text, набираем «R = «.

Точно так же изменяется свойство Text для элемента управления label2 (рисунок 3). Вводим текст «Площадь поверхности шара = «.

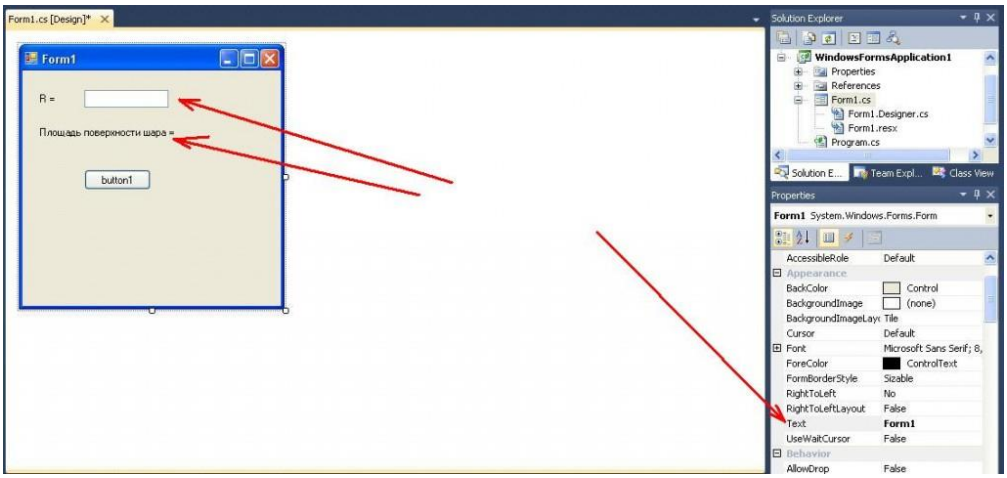


Рисунок 3 - Изменение свойства Text в label1

4. Элемент управления Button.

Аналогично к label1 выделяем элемент управления (объект) button1. В свойстве Text вводим строку «Вычислить«.

5. Корректировка вида формы.

Изменяем название формы. Для этого выделяем форму. В свойстве Text формы вводим текст «Площадь поверхности шара«.
После корректировки форма будет иметь вид, как показано на рисунке 4.

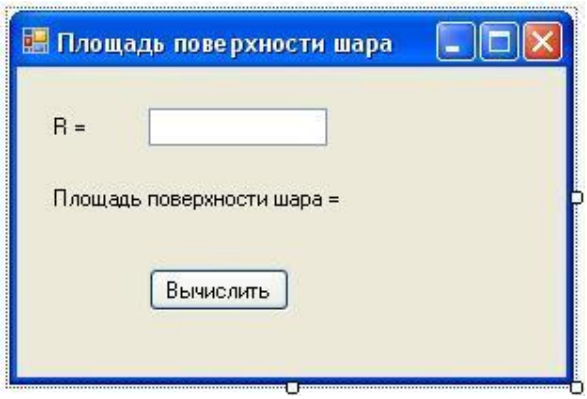


Рисунок 4 - Вид формы после корректировки

6. Программирование события клика на кнопке «Вычислить«.

6.1. Вызов программного кода.

В программе нужно обработать событие, которое генерируется, когда пользователь делает клик «мышкой» на кнопке «Вычислить«.

После нажатия на кнопке «Вычислить» формируется фрагмент кода, который будет обрабатываться нашим приложением. Во время обработки сначала определяется значение введенного радиуса R, затем делается расчет по формуле и вывод результата.

Чтобы вызвать фрагмент кода обработки события на кнопке button1 нужно выполнить такие действия:

- выделить кнопку button1 (рис. 5);
- перейти к вкладке Events (события) в окне свойств;
- сделать двойной щелчок «мышкой» напротив события «Click» (рисунки 5 — 7).

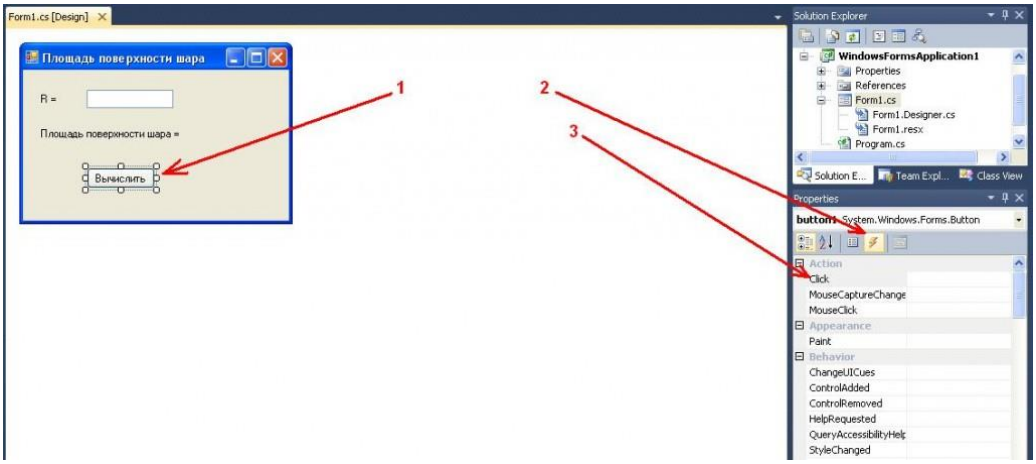


Рисунок 5 - Последовательность вызова фрагмента кода обработки события Click

В итоге, откроется вкладка программного кода, который размещен в файле «Form1.cs» (рисунок 6).

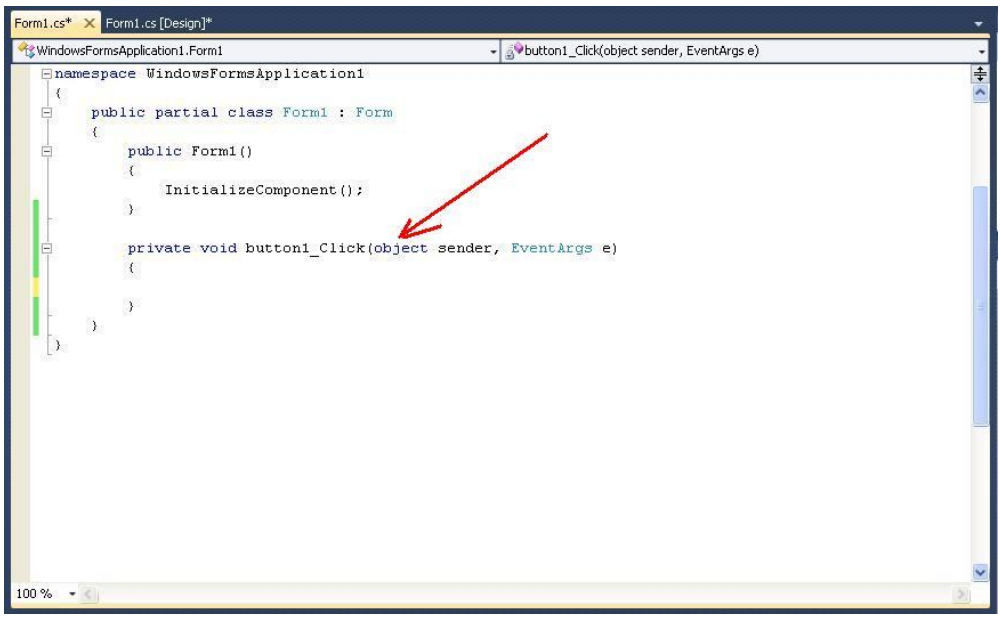


Рисунок 6 - Метод обработки события клика на кнопке button1

Листинг 1 - Программный код

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {

    }
  }
}
```

6.2. Ввод программного кода обработки события.

В методе button1_Click вписываем код обработки события.

Листинг 2 – Реализация метода

...

```
private void button1_Click(object sender, EventArgs e)
{
   const double Pi = 3.1415;
   double R,S;
   R = Double.Parse(textBox1.Text);
   S = 4 * Pi * R * R;
   label2.Text = "Площадь поверхности шара = " + S.ToString();
}
```

В коде описываются две переменные *R* и *S* типа double. Также описывается константа *Pi*.

Для преобразования из текстового типа string в тип double используется метод Parse. Таким образом заполняется значение переменной *R*.

```
R = Double.Parse(textBox1.Text.ToString());
```

Подобным образом можно преобразовывать данные и других типов. Например, для типа int можно написать:

```
int d;
d = Int32.Parse("100"); // d = 100
```

Результат вычисления площади поверхности шара выводится в label2.Text. Преобразование в тип stringосуществляется с помощью метода ToString().

6.3. Корректировка программного кода.

В коде, описанном в пункте 6.2, нет защиты от некорректного ввода значения *R*. Поэтому метод button1_Click нужно переписать следующим образом:

Листинг 3 – Реализация метода button1_Click

```
...
private void button1_Click(object sender, EventArgs e)
{
   try
   {
      const double Pi = 3.1415;
      double R, S;
      R = Double.Parse(textBox1.Text);
      S = 4 * Pi * R * R;
      label2.Text = "Площадь поверхности шара = " + S.ToString();
   }
   catch (FormatException ex)
   {
      label2.Text = "Ошибка - " + ex.Message;
   }
}
...
```

Блок

```
try
{
  ...
}
catch (...)
{
  ...
}
```

позволяет осуществить программный перехват критической ситуации, которая может возникнуть в результате ввода некорректного значения в поле textBox1 (например «абракадабра»).

В этом случае в свойстве ex.Message будет выведено сообщение:

Ошибка – Input string was not in a correct format.

7. Запуск программы на выполнение.

После этого можно запустить нашу программу на выполнение и протестировать ее работу при любых значениях *R*.

11. 3.5 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

4. ЛАБОРАТОРНАЯ РАБОТА № 3. ПРИМЕР СОЗДАНИЯ ДВУМЕРНОЙ МАТРИЦЫ НА ФОРМЕ

12. 4.1 Общие сведения

Цель: научиться вносить данные в двумерный массив (матрицу) и иметь возможность их обрабатывать.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 2.

Критерии положительной оценки: разработанная программа создания двумерной матрицы и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:

Аудиторное время выполнения (под руководством преподавателя): 1 ч.

Время самостоятельной подготовки: 1 ч.

13. 4.2 Теоретическое введение

Объясним некоторые значения переменных:

- Max – максимально-допустимая размерность матрицы;
- n – размерность матрицы, введенная пользователем из клавиатуры в элементе управления TextBox1;
- MatrText – двумерная матрица элементов управления типа TextBox. В эту матрицу будут вводиться элементы матрицы в виде строк. Ввод данных будет формироваться в форме Form2;
- Matr1, Matr2 – матрицы элементов типа double, в которые будут копироваться данные из матрицы MatrText;
- Matr3 – результирующая матрица, которая равная произведению матриц Matr1 и Matr2;
- f1, f2 – переменные, определяющие были ли введенные данные соответственно в матрицы Matr1 и Matr2;
- dx, dy – габариты одной ячейки типа TextBox в матрице MatrText;
- form2 – объект класса формы Form2, по которыму будет получен доступ к этой форме.

Контрольные вопросы для самопроверки: что такое матрица? Каким образом мы задаем ее размерность? Какие типы данных можно вносить в матрицы?

14. 4.3 Задание к лабораторной работе

15. Составить программу, которая осуществляет произведение двух матриц размерностью **n**. Матрицы вводятся из клавиатуры в отдельной форме и заносятся во внутренние структуры данных. Пользователь имеет возможность просмотреть результирующую матрицу. Также есть возможность сохранения результирующей матрицы в текстовом файле “Res_Matrix.txt”.

16. 4.4 Методические указания и порядок выполнения работы

1. Запуск Microsoft Visual Studio. Создание проекта

2. Создание главной формы Form1

Создать форму, как показано на рисунке 7.

Разместить на форме элементы управления следующих типов:

- четыре элемента управления типа Button. Автоматически будут созданы четыре объекта (переменные) с именами button1, button2, button3, button4;
- три элемента управления типа Label с именами label1, label2, label3;
- один элемент управления типа TextBox, доступ к которому можно получить по имени textBox1.

Сформировать свойства элементов управления типа Button и Label:

- в объекте button1 свойство Text = “Ввод матрицы 1 …«;
- в объекте button2 свойство Text = “Ввод матрицы 2 …«;
- в объекте button3 свойство Text = “Результат …»;
- в объекте button4 свойство Text = “Сохранить в файле “Res_Matr.txt” ”;
- в элементе управления label1 свойство Text = “n = ”.

Для настройки вида и поведения формы нужно выполнить следующие действия:

- установить название формы. Для этого свойство Text = “Произведение матриц”;
- свойство StartPosition = “CenterScreen” (форма размещается по центру экрана);
- свойство MaximizeBox = “false” (убрать кнопку развертывания на весь экран).

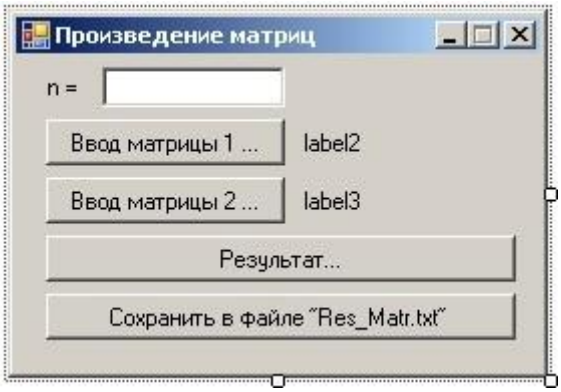


Рисунок 7 - Форма приложения

3. Создание второстепенной формы Form2

Во второстепенной форме Form2 будут вводиться данные в матрицы и выводиться исходный результат.

Добавить новую форму к приложению, вызвав команду:

Project -> Add Windows Form …

В открывшемся окне выбрать «Windows Form». Имя файла оставить без изменений «Form2.cs».

Разместить на форме в любом месте элемент управления типа Button (рисунок 8). В результате будет получен объект с именем button1.

В элементе управления button1 нужно установить следующие свойства:

- свойство Text = “OK”;
- свойство DialogResult = “OK” (рисунок 9). Это означает, что при нажатии (клике «мышкой») на button1, окно закроется с кодом возвращения равным “OK”;
- свойство Modifiers = “Public”. Это означает, что кнопка button1 будет видимой из других модулей (из формы Form1).

Настроить свойства формы Form2:

- свойство Text = “Ввод матрицы”;
- свойство StartPosition = “CenterScreen” (форма размещается по центру экрана);
- свойство MaximizeBox = “false” (убрать кнопку развертывания на весь экран).

Рисунок 8 - Форма Form2 после настройки

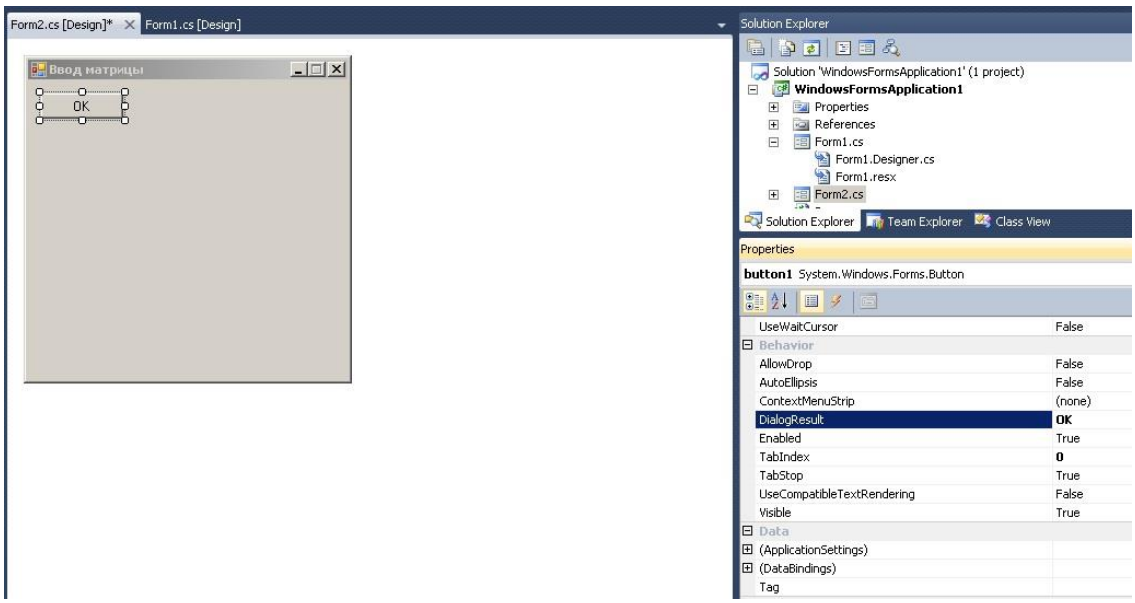


Рисунок 9- Свойство DialogResult элемента управления button1 формы Form2

4. Ввод внутренних переменных

Следующий шаг – введение внутренних переменных в текст модуля “Form1.cs”. Для этого сначала нужно активировать модуль “Form1.cs”.
В тексте модуля “Form1.cs” добавляем следующий код (листинг 1):

Листинг 1 – Код модуля Form1.cs

```
...

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      const int MaxN = 10; // максимально допустимая размерность матрицы
      int n = 3; // текущая размерность матрицы
      TextBox[,] MatrText = null; // матрица элементов типа TextBox
      double[,] Matr1 = new double[MaxN, MaxN]; // матрица 1 чисел с плавающей точкой
      double[,] Matr2 = new double[MaxN, MaxN]; // матрица 2 чисел с плавающей точкой
      double[,] Matr3 = new double[MaxN, MaxN]; // матрица результатов
      bool f1; // флажок, который указывает о вводе данных в матрицу Matr1
      bool f2; // флажок, который указывает о вводе данных в матрицу Matr2
      int dx = 40, dy = 20; // ширина и высота ячейки в MatrText[,]
      Form2 form2 = null;   // экземпляр (объект) класса формы Form2

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

5. Программирование события Load формы Form1

Листинг обработчика события Load формы Form1 следующий (листинг 2):

Листинг 2 – Код обработчика событий

```
private void Form1_Load(object sender, EventArgs e)
{
  // I. Инициализация элементов управления и внутренних переменных
  textBox1.Text = "";
  f1 = f2 = false; // матрицы еще не заполнены
  label2.Text = "false";
  label3.Text = "false";

  // II. Выделение памяти и настройка MatrText
  int i, j;

  // 1. Выделение памяти для формы Form2
  form2 = new Form2();

  // 2. Выделение памяти под самую матрицу
  MatrText = new TextBox[MaxN, MaxN];

  // 3. Выделение памяти для каждой ячейки матрицы и ее настройка
  for (i = 0; i < MaxN; i++)
    for (j = 0; j < MaxN; j++)
    {
      // 3.1. Выделить память
      MatrText[i, j] = new TextBox();

      // 3.2. Обнулить эту ячейку
      MatrText[i, j].Text = "0";

      // 3.3. Установить позицию ячейки в форме Form2
      MatrText[i, j].Location = new System.Drawing.Point(10 + i * dx, 10 + j * dy);

      // 3.4. Установить размер ячейки
      MatrText[i, j].Size = new System.Drawing.Size(dx, dy);

      // 3.5. Пока что спрятать ячейку
      MatrText[i, j].Visible = false;

      // 3.6. Добавить MatrText[i,j] в форму form2
      form2.Controls.Add(MatrText[i, j]);
    }
}
```

Объясним некоторые фрагменты кода в методе Form1_Load().

Событие Load генерируется (вызывается) в момент загрузки любой формы. Поскольку форма Form1 есть главной формой приложения, то событие Load формы Form1 будет вызываться сразу после запуска приложения на выполнение. Поэтому, здесь целесообразно ввести начальную инициализацию глобальных элементов управления и внутренних переменных программы. Эти элементы управления могут быть вызваны из других методов класса.

В обработчике события Form1_Load() выделяется память для двумерной матрицы строк MatrText один лишь раз. При завершении приложения эта память будет автоматически освобождена.

Память выделяется в два этапа:

- для самой матрицы MatrText – как двумерного массива;
- для каждого элемента матрицы, который есть сложным объектом типа TextBox.

После выделения памяти для любого объекта осуществляется настройка основных внутренних свойств (позиция, размер, текст, видимость в некоторой форме).

Также каждая созданная ячейка добавляется (размещается) на форму Form2 с помощью метода Add() из класса Controls. Каждая новая ячейка может быть добавлена в любую другую форму приложения.

6. Разработка дополнительного метода обнуления данных в матрице MatrText

В будущем, чтобы многократно не использовать код обнуления строк матрицы MatrText, нужно создать собственный метод (например, Clear_MatrText()) реализующий этот код.
Листинг метода Clear_MatText() следующий (листинг 3):

Листинг 3 – Код метода Clear_MatText()

```
private void Clear_MatrText()
{
  // Обнуление ячеек MatrText
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      MatrText[i, j].Text = "0";
}
```

7. Программирование события клика на кнопке button1 («Ввод матрицы 1 …»)

При нажатии (клике) на button1 должно вызываться окно ввода новой матрицы. Размер матрицы зависит от значения n.
Листинг обработчика события клика на кнопке button1 следующий (листинг 4):

Листинг 4 – Код обработчика событий клика

```
private void button1_Click(object sender, EventArgs e)
{
  // 1. Чтение размерности матрицы
  if (textBox1.Text == "") return;
  n = int.Parse(textBox1.Text);

  // 2. Обнуление ячейки MatrText
  Clear_MatrText();

  // 3. Настройка свойств ячеек матрицы MatrText
  //    с привязкой к значению n и форме Form2
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
      // 3.1. Порядок табуляции
      MatrText[i, j].TabIndex = i * n + j + 1;
```

```
    // 3.2. Сделать ячейку видимой
    MatrText[i, j].Visible = true;
   }

  // 4. Корректировка размеров формы
  form2.Width = 10 + n * dx + 20;
  form2.Height = 10 + n * dy + form2.button1.Height + 50 ;

  // 5. Корректировка позиции и размеров кнопки на форме Form2
  form2.button1.Left = 10;
  form2.button1.Top = 10 + n * dy + 10;
  form2.button1.Width = form2.Width - 30;

  // 6. Вызов формы Form2
  if (form2.ShowDialog() == DialogResult.OK)
  {
   // 7. Перенос строк из формы Form2 в матрицу Matr1
   for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
     if (MatrText[i, j].Text != "")
      Matr1[i, j] = Double.Parse(MatrText[i, j].Text);
     else
      Matr1[i, j] = 0;
     // 8. Данные в матрицу Matr1 внесены
     f1 = true;
     label2.Text = "true";
  }
}
```

В вышеприведенном листинге читается значение n. После этого осуществляется настройка ячеек матрицы строк MatrText.
На основе введенного значения n формируются размеры формы form2 и позиция кнопки button1.

Если в форме Form2 пользователь нажал на кнопке OK (button2), то строки с MatrText переносятся в двумерную матрицу вещественных чисел Matr1. Преобразование из строки в соответствующее вещественное число выполняется методом Parse() из класса Double.

Также формируется переменная f1, которая указывает что данные в матрицу Matr1 внесены.

8. Программирование события клика на кнопке button2 (“Ввод матрицы 2…«)

Листинг обработчика события клика на кнопке button2 подобен листингу обработчика события клика на кнопке button1. Только он отличается шагами 7-8. На этом участке формируются матрица Matr2 и переменная f2.

```
private void button2_Click(object sender, EventArgs e)
{
  // 1. Чтение размерности матрицы
  if (textBox1.Text == "") return;
  n = int.Parse(textBox1.Text);

  // 2. Обнулить ячейки MatrText
```

```
  Clear_MatrText();

  // 3. Настройка свойств ячеек матрицы MatrText
  //    с привязкой к значению n и форме Form2
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
      // 3.1. Порядок табуляции
      MatrText[i, j].TabIndex = i * n + j + 1;

      // 3.2. Сделать ячейку видимой
      MatrText[i, j].Visible = true;
    }

  // 4. Корректировка размеров формы
  form2.Width = 10 + n * dx + 20;
  form2.Height = 10 + n * dy + form2.button1.Height + 50;

  // 5. Корректировка позиции и размеров кнопки на форме Form2
  form2.button1.Left = 10;
  form2.button1.Top = 10 + n * dy + 10;
  form2.button1.Width = form2.Width - 30;

  // 6. Вызов формы Form2
  if (form2.ShowDialog() == DialogResult.OK)
  {
    // 7. Перенос строк из формы Form2 в матрицу Matr2
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        Matr2[i, j] = Double.Parse(MatrText[i, j].Text);

    // 8. Матрица Matr2 сформирована
    f2 = true;
    label3.Text = "true";
  }
}
```

9. Программирование события Leave потери фокуса ввода элементом управления textBox1

В приложении может возникнуть ситуация, когда пользователь изменяет значение n на новое. В этом случае должны заново формироваться флажки f1 и f2. Также изменяется размер матрицы MatrText, которая выводится в форме Form2.

Изменение значения n можно проконтролировать с помощью события Leave элемента управление textBox1.Событие Leave генерируется в момент потери фокуса ввода элементом

управления textBox1 (рисунок 4).

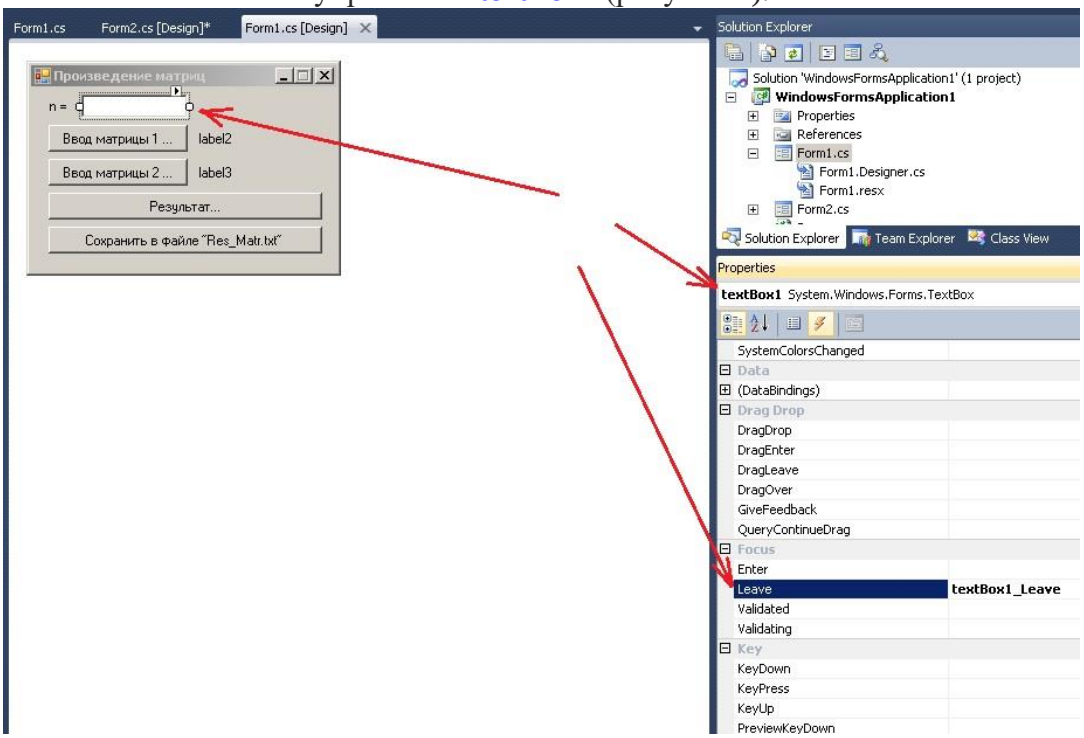


Рисунок 10 - Событие Leave элемента управления textBox1

Листинг обработчика события Leave следующий (листинг 5):

Листинг 5 – Код обработчика событий Leave

```
private void textBox1_Leave(object sender, EventArgs e)
{
  int nn;
  nn = Int16.Parse(textBox1.Text);
  if (nn != n)
  {
    f1 = f2 = false;
    label2.Text = "false";
    label3.Text = "false";
  }
}
```

10. Программирование события клика на кнопке button3 («Результат»)

Вывод результата будет осуществляться в ту же форму, в которой вводились матрицы Matr1 и Matr2.Сначала произведение этих матриц будет сформировано в матрице Matr3. Потом значение с Matr3переносится в MatrText и отображается на форме Form2.

Листинг обработчика события клика на кнопке button3 (листинг 6).

Листинг 6 – Код обработчика событий

```
private void button3_Click(object sender, EventArgs e)
{
  // 1. Проверка, введены ли данные в обеих матрицах
  if (!((f1 == true) && (f2 == true))) return;

  // 2. Вычисление произведения матриц. Результат в Matr3
```

```
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
      Matr3[j,i] = 0;
      for (int k = 0; k < n; k++)
        Matr3[j, i] = Matr3[j, i] + Matr1[k, i] * Matr2[j, k];
    }

  // 3. Внесение данных в MatrText
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
      // 3.1. Порядок табуляции
      MatrText[i, j].TabIndex = i * n + j + 1;

      // 3.2. Перевести число в строку
      MatrText[i, j].Text = Matr3[i, j].ToString();
    }

  // 4. Вывод формы
  form2.ShowDialog();
}
```

11. Программирование события клика на кнопке button4 («Сохранить в файле «Res_Matr.txt””)

Для сохранения результирующей матрицы Matr3 можно использовать возможности класса FileStream.

Класс FileStream описан в модуле System.IO. Поэтому в начале приложения нужно добавить следующий код:

```
using System.IO;
```

Листинг обработчика события клика на кнопке button4 следующий (листинг 7):

Листинг 7 – Код обработчика события клика на кнопке button4

```
private void button4_Click(object sender, EventArgs e)
{
  FileStream fw = null;
  string msg;
  byte[] msgByte = null; // байтовый массив

  // 1. Открыть файл для записи
  fw = new FileStream("Res_Matr.txt", FileMode.Create);

  // 2. Запись матрицы результата в файл
  // 2.1. Сначала записать число элементов матрицы Matr3
  msg = n.ToString() + "\r\n";
  // перевод строки msg в байтовый массив msgByte
  msgByte = Encoding.Default.GetBytes(msg);

  // запись массива msgByte в файл
  fw.Write(msgByte, 0, msgByte.Length);

  // 2.2. Теперь записать саму матрицу
```

```
 msg = "";
 for (int i = 0; i < n; i++)
 {
  // формируем строку msg из элементов матрицы
  for (int j = 0; j < n; j++)
    msg = msg + Matr3[i, j].ToString() + "  ";
  msg = msg + "\r\n"; // добавить перевод строки
 }

 // 3. Перевод строки msg в байтовый массив msgByte
 msgByte = Encoding.Default.GetBytes(msg);

 // 4. запись строк матрицы в файл
 fw.Write(msgByte, 0, msgByte.Length);

 // 5. Закрыть файл
 if (fw != null) fw.Close();
}
```

12. Запуск приложения на выполнение

После этого можно запускать приложение на выполнение и тестировать его работу.

1. 4.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

5. ЛАБОРАТОРНАЯ РАБОТА № 4. СОЗДАНИЕ ПРОСТОГО ПРИМЕРА WCF.

2. 5.1 Общие сведения

Цель: научиться создавать проекты WCF в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 3.

Критерии положительной оценки: разработанная программа консольного приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:

Аудиторное время выполнения (под руководством преподавателя): 1 ч.

Время самостоятельной подготовки: 1 ч.

3. 5.2 Задание к лабораторной работе

Написать программу простого консольного приложения WCF, которая при нажатии кнопки выводи сообщение.

5.3 Методические указания и порядок выполнения работы

1. Входим в среду программирования Microsoft Visual Studio 2010. Создаем проект через пункт меню "File" -> "New Project"* (рисунок 11).

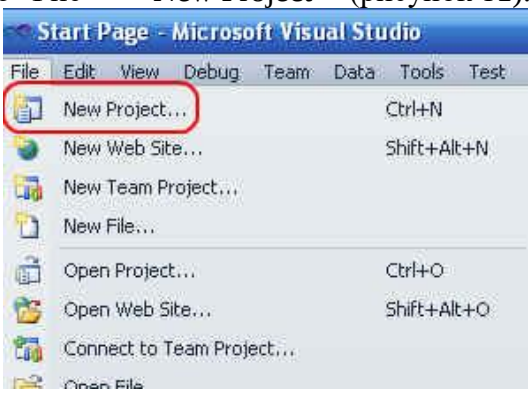


Рисунок 11 – Создание проекта меню

Если ваша Visual Studio не настроена по умолчанию на C#, то вам придется войти в ветку Other Languages, затем в Visaul C#, а потом выбрать Windows* (рисунок 12).

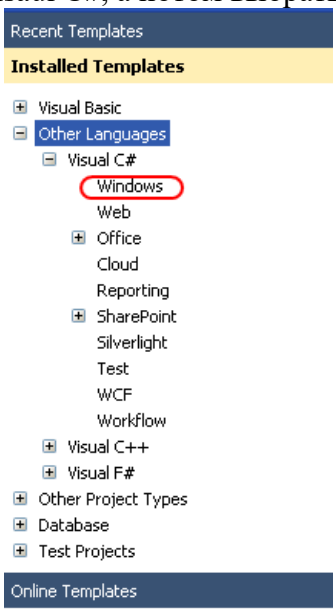


Рисунок 12 – Выбор ветки

Теперь нам надо выбрать тип создаваемого приложения (Windows form application)* (рисунок 13).

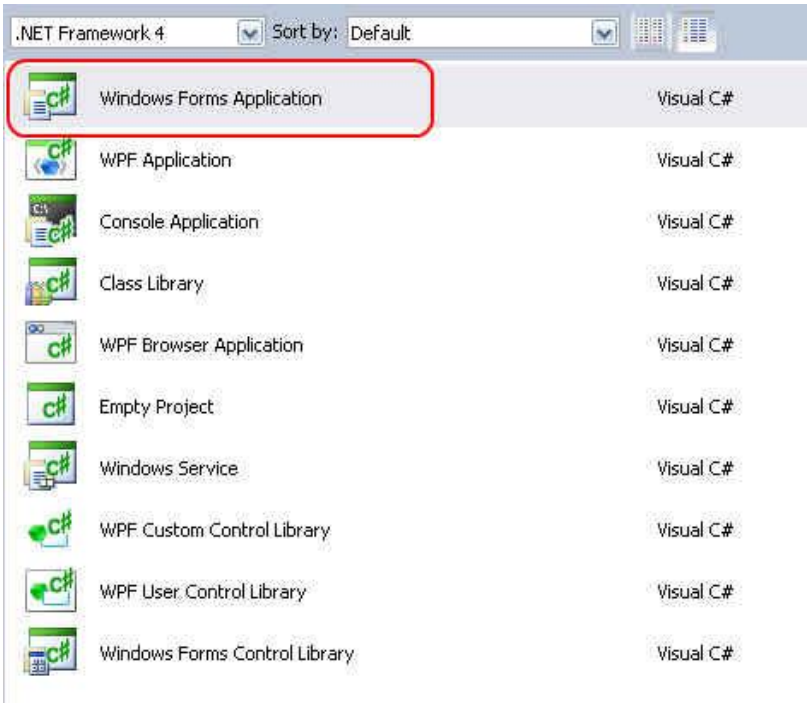


Рисунок 13 – Тип приложения

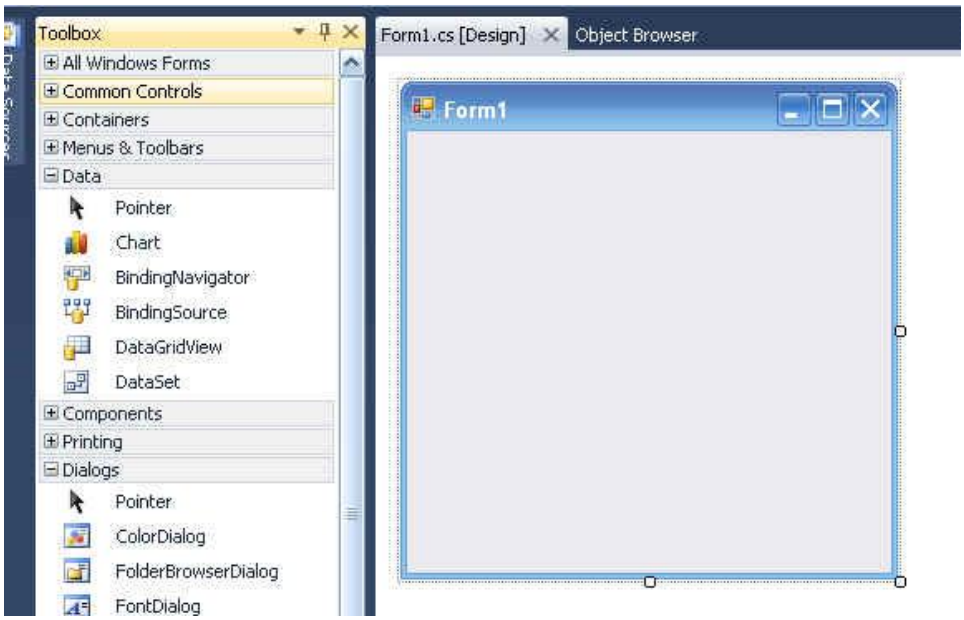


Рисунок 14 – Палитра компонентов

Давайте для начала раскроем вкладку "Common Controls", ткнем мышкой в кнопочку "Button" и той же мышкой ткнем по форме (там, где у нас заголовок Form1).

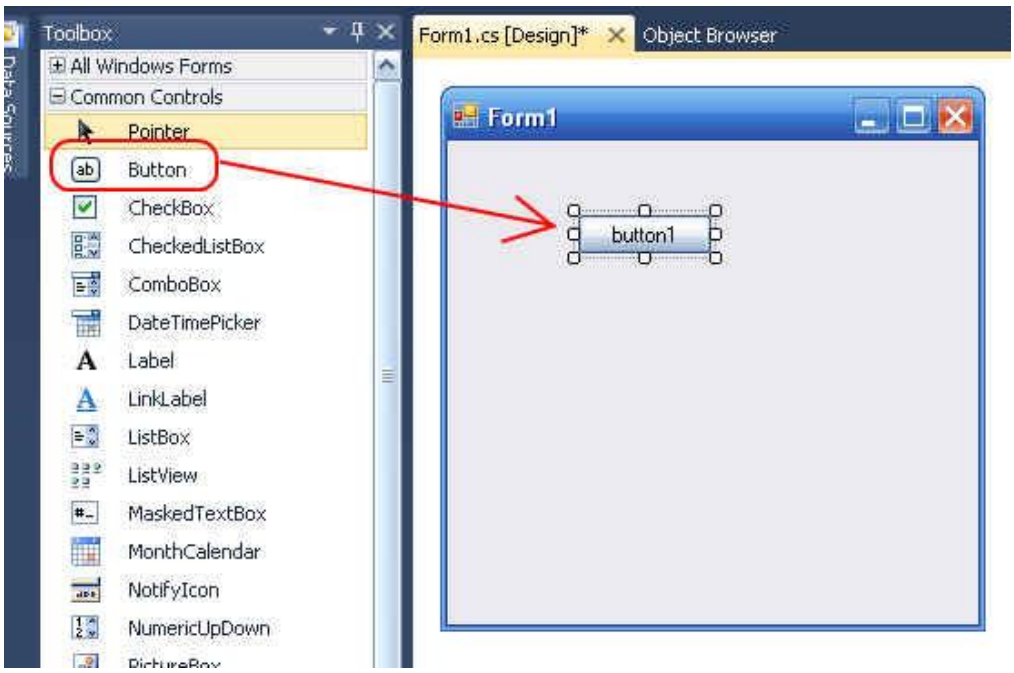


Рисунок 15 – Выбор элемента управления

«Button» («кнопка») - это *компонент (component)* «Button» («кнопка»). Так же можно положить на форму другие компоенты, например «ListBox» (Визуальный список), «TextBox» (поле редактирования текста).

Компонент – это такой *объект*, который можно положить на форму. Он уже заранее запрограммирован разработчиком и включен в систему. В C# этих компонентов великое множество. Но мы пока рассмотрим толькоButton и Label. Итак, кладем на форму еще и Label* (рисунок 15) :

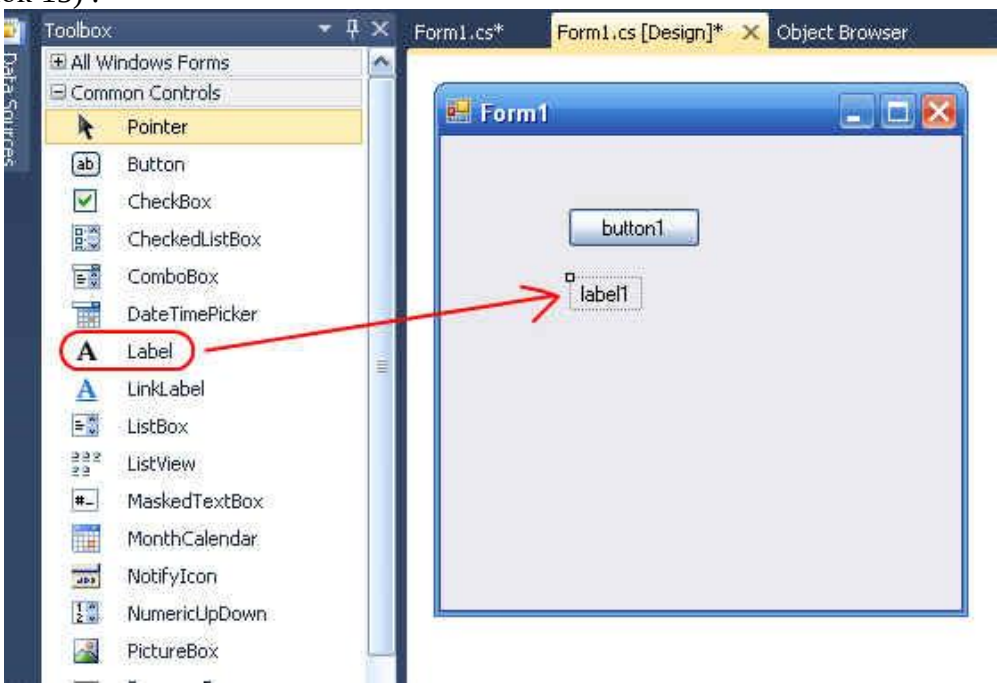


Рисунок 16 - Выбор элемента управления

Теперь щелкнем на положенную нами на форму кнопку. Откроется окно редактирования кода, при этом будет еще и автоматически создан шаблон обработчика нажатия кнопочки (участка кода, который запускается, когда юзер мышкой жмет на кнопку)*:

Пока не будет заворачиваться с непонятными словами языке C#, а в предложенное системой место (после private void button1_Click(object sender, EventArgs e) {) введем вот такой код:

```
label1.Text = "Hellow, world!";
```

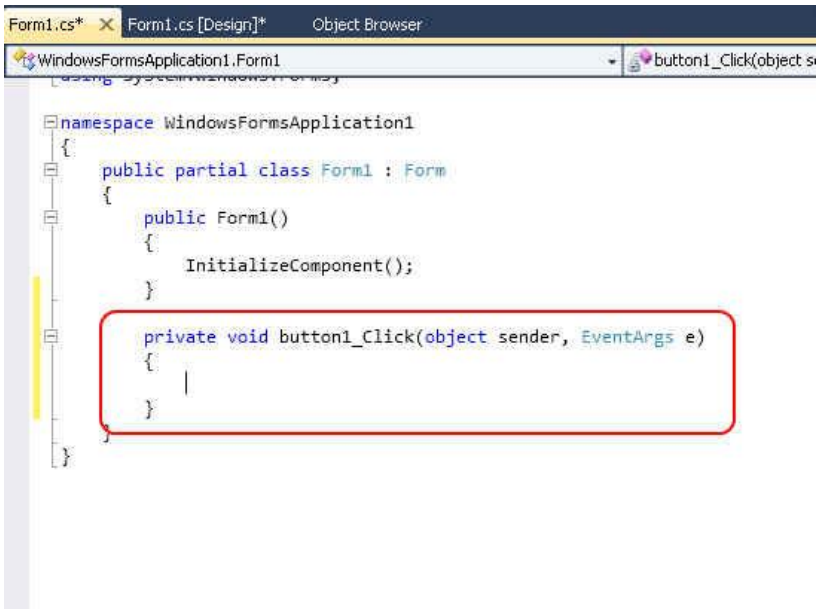


Рисунок 17 – Участок кода

Запускаем приложение.

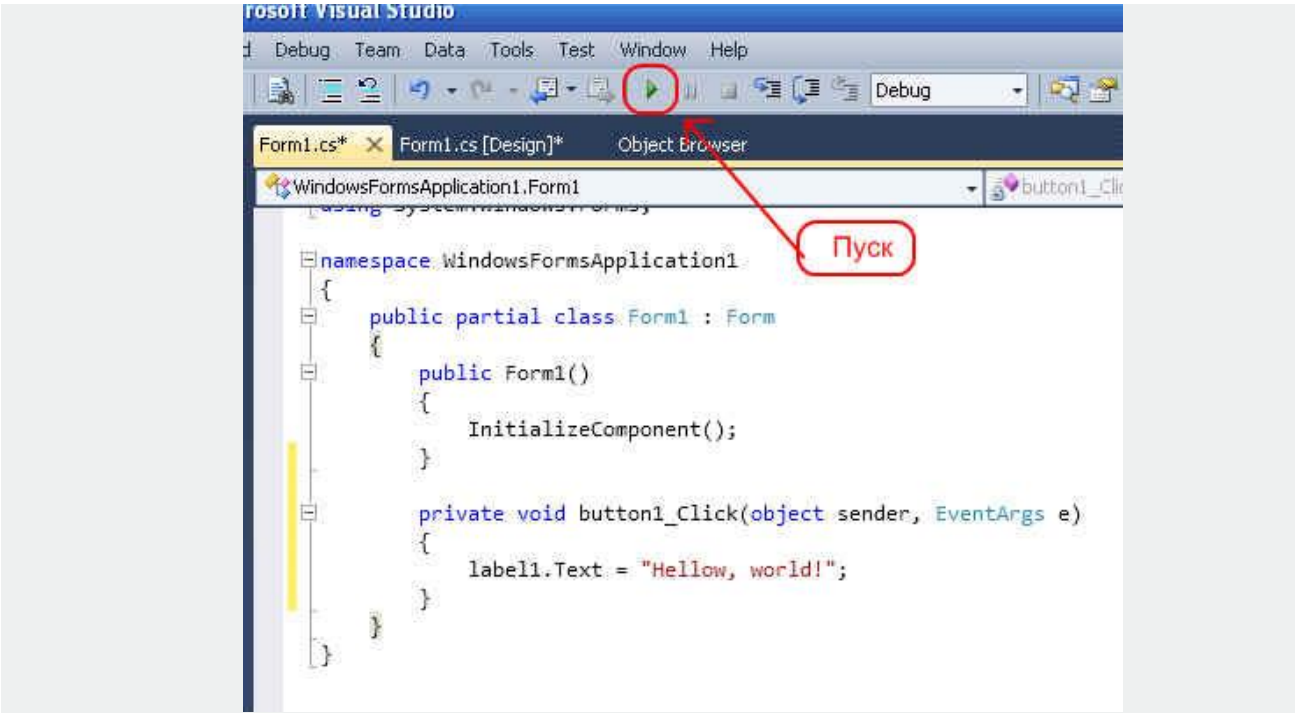


Рисунок 18 – Участок кода

Наша программа скомпилируется и запустится:

Рисунок 19 – Участок кода

Нажмем на Button1 и увидим надпись "Hello, world!":

2. Создание консольного приложения

"File" -> "New Project...", затем в "Other languages" -> "Visual C#" -> "Windows". Но на этот раз создадим не "Windows forms Application", а "Console Application":

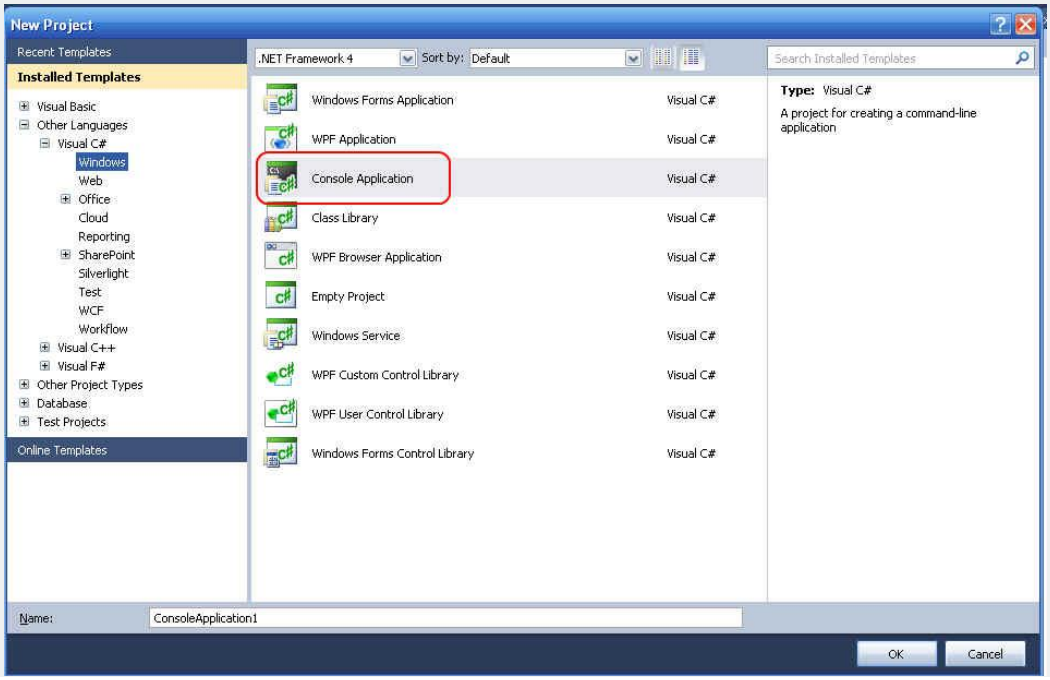


Рисунок 20 – Участок кода

У нас появится шаблон консольного приложения, формы, естественно, нет, так как это у нас будет консольное приложение. В шаблоне присутствует только заготовка текста программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

наш код мы прописываем в процедуру Main:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите ваше имя:");
            string sName = Console.ReadLine();
            Console.WriteLine("Привет, " + sName);
            Console.WriteLine("Нажмите ENTER что бы выйти из программы");
            Console.Read();
        }
    }
}
```

Эта коротенькая программа при своем запуске попросит ввести ваше имя, а потом поздоровается с вами:

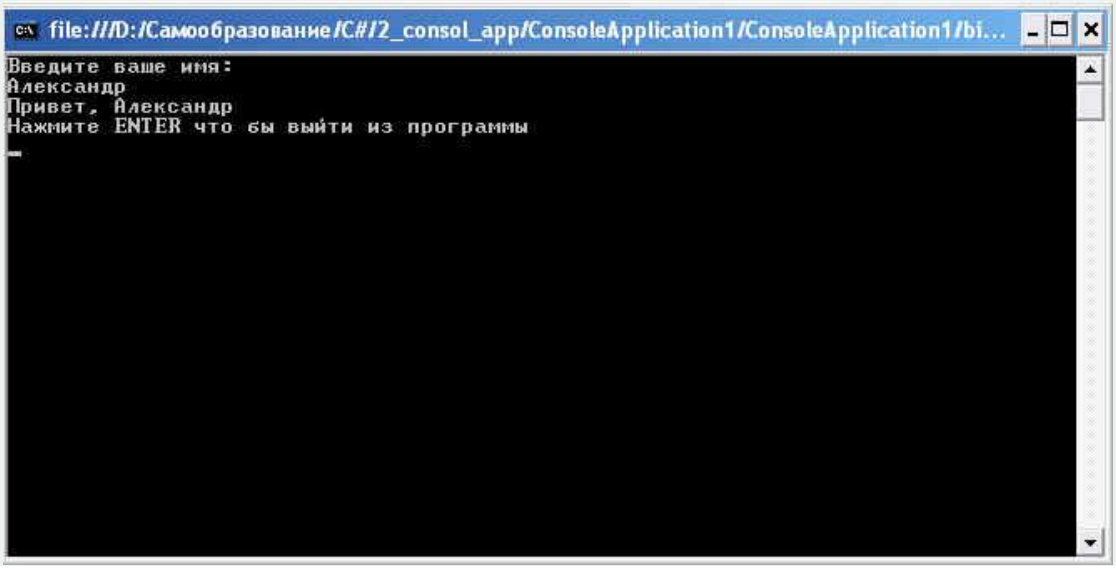


Рисунок 21 – Участок кода

Теперь несколько пояснений к коду программы.

```
Console.WriteLine("Введите ваше имя:");
```

данная команда, как вы уже догадались, выводит заданный текст на экран.

```
string sName = Console.ReadLine();
```

а эта команда ожидает, когда пользователь введет что-то в компьютер и то, что он ввел помещает в переменную sName.

```
Console.WriteLine("Привет, " + sName);
```

а тут мы выводим составную строку. Первая часть состоит из надписи "Привет, ", вторая из того, что ввел пользователь. Знак "+" складывает два значение. Если это числа - складывает числа, а если строки - то соединяет их вместе (потому что мы не может строку прибавить к строке, а соединить можем).

У объекта есть методы (встречные процедуры и функции) и свойства - встроенные в объекты поля. WriteLine и ReadLine это как раз пример методов.

5.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

6. ЛАБОРАТОРНАЯ РАБОТА № 5. ПРИМЕР ИСПОЛЬЗОВАНИЯ ДЕЛЕГАТА ДЛЯ ВЫЗОВА АНОНИМНОГО МЕТОДА.

6.1 Общие сведения

Цель: научиться использовать делегаты для вызова анонимного метода в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 4.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:

Аудиторное время выполнения (под руководством преподавателя): 1 ч.

Время самостоятельной подготовки: 1 ч.

6.2 Теоретические сведения

Формула Герона имеет вид:

$$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$$ где

- S – площадь треугольника;
- a, b, c – длины сторон треугольника;
- p – полупериметр, который вычисляется по формуле:

$$p = \frac{a+b+c}{2}$$

6.3 Задание к лабораторной работе

Разработать приложение, которое находит площадь треугольника по формуле Герона. В приложении реализовать вызов анонимного метода с помощью делегата. Метод должен осуществлять вычисление площади треугольника. Приложение реализовать в Microsoft Visual Studio по шаблону Windows Forms Application.

6.4 Методические указания и порядок выполнения работы

1. Запустить систему Microsoft Visual Studio. Создать проект по шаблону Windows Forms Application. Сохранить проект в произвольной папке, например:

D:\Programs\C_Sharp\TrainDelegates03

В результате будет создана новая форма приложения, как показано на рисунке 22.

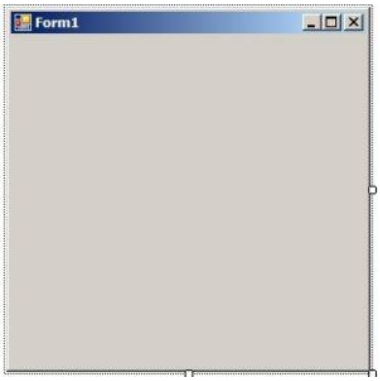


Рисунок 22 - Форма приложения после создания проекта

2. Разработка формы приложения

2.1. Размещение элементов управления на форме

Разместить на форме следующие элементы управления:

- 4 элемента управления типа Label. В результате будет создано 4 объекта (переменные) с именами label1, label2, label3, label4;
- 1 элемент управления типа Button. Будет создан объект с именем button1;
- 3 элемента управления типа TextBox. В результате будет создано 3 объекта с именами textBox1, textBox2, textBox3.

После размещения элементов управления, форма приложения будет иметь приблизительный вид, как показано на рисунке 23.

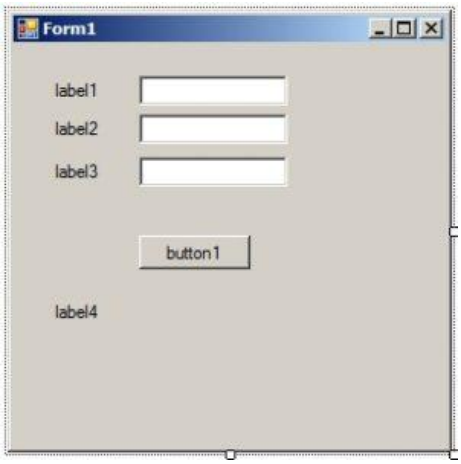


Рисунок 23 - Форма приложения после размещения элементов управления

2.2. Настройка элементов управления

Настроить следующие свойства элементов управления:

- в элементе управления label1 свойство Text = «a = « (label1.Text = «a = «);
- label2.Text = «b = «;
- label3.Text = «c = «;
- в элементе управления button1 свойство Text = «Вычислить» (button1.Text = «b = «);
- в объекте, который соответствует форме Form1, свойство Text = «Площадь треугольника».

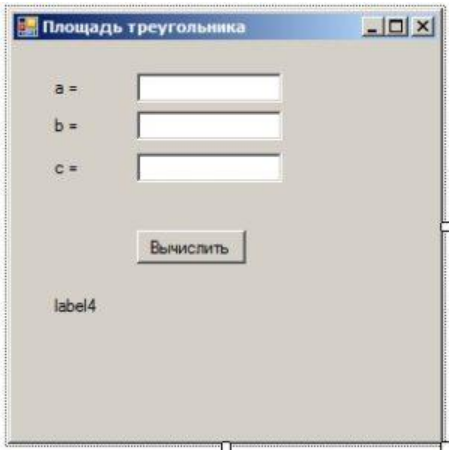


Рисунок 24 - Главная форма приложения после настройки

3. Написание программного кода

Для написания программного кода нужно перейти в файл Form1.cs, который соответствует главной форме приложения (программы).

3.1. Объявление типа делегата

В теле класса формы Form1 объявить тип делегата:

```
// объявление типа делегата
delegate float SquareTriangle(float a, float b, float c);
```

Об объявленном типе делегата можно сказать следующее:

- тип делегата носит имя SquareTriangle;
- делегат этого типа будет получать три параметра типа float и возвращать значение типа float.

После объявления типа текст модуля «Form1.cs» следующий:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TrainDelegates03
{
   public partial class Form1 : Form
   {
      // объявление типа делегата
      delegate float SquareTriangle(float a, float b, float c);

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

3.2. Программирование обработчика события клика на кнопке «Вычислить»

Текст обработчика события клика на кнопке button1:

```
private void button1_Click(object sender, EventArgs e)
{
   // Объявление делегата с именем ST, который вычисляет площадь треугольника
   SquareTriangle ST;

   ST = delegate(float a, float b, float c)
   {
      float s, p, d;

      p = (a + b + c) / 2.0f;
      d = p * (p - a) * (p - b) * (p - c);
      if (d < 0) return -1.0f;
```

```
        s = (float)Math.Sqrt(p * (p - a) * (p - b) * (p - c));

        return (float)s;
    };

    // получить значения длин a, b, c
    float aa, bb, cc;

    aa = (float)Convert.ToDouble(textBox1.Text);
    bb = (float)Convert.ToDouble(textBox2.Text);
    cc = (float)Double.Parse(textBox3.Text); // так тоже можно преобразовывать

    // вызов делегата
    float area;
    area = (float)ST(aa, bb, cc);

    // вывод результата на форму
    label4.Text = "S = " + area.ToString();
}
```

Объясним некоторые фрагменты кода. В обработчике события объявляется делегат с именем ST типа SquareTriangle. Делегат ST ссылается на анонимный метод, который получает входными 3 параметра типа float. В анонимном методе осуществляется вычисление площади треугольника по формуле Герона.

Результат (площадь) возвращается с помощью оператора return.
Длины сторон, которые вводятся с клавиатуры (элементы управления textBox1, textBox2, textBox3) размещаются в переменных aa, bb, cc.

Согласно синтаксису C# переменные, которые объявлены в анонимном методе, имеют видимость на весь блок кода обработчика события button1_Click(). Поэтому, не может быть одинаковых имен в обработчике события и анонимном методе, который описан в этом обработчике. То есть, объявление
float aa, bb, cc;
является верным. Если в тексте обработчика события попробовать написать

float a, b, c;
то выйдет ошибка компиляции, поскольку такие имена уже используются в анонимном методе.

3.3. Текст модуля Form1.cs

Весь программный код модуля Form1.cs имеет вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace TrainDelegates03
```

```
{
   public partial class Form1 : Form
   {
      // объявление типа делегата
      delegate float SquareTriangle(float a, float b, float c);

      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         // Объявление делегата с именем ST, вычисляющего площадь треугольника
         SquareTriangle ST;

         ST = delegate(float a, float b, float c)
         {
            float s, p, d;

            p = (a + b + c) / 2.0f;
            d = p * (p - a) * (p - b) * (p - c);
            if (d < 0) return -1.0f;
            s = (float)Math.Sqrt(p * (p - a) * (p - b) * (p - c));

            return (float)s;
         };

         // взять значения длин a, b, c
         float aa, bb, cc;

         aa = (float)Convert.ToDouble(textBox1.Text);
         bb = (float)Convert.ToDouble(textBox2.Text);
         cc = (float)Double.Parse(textBox3.Text); // так тоже можно конвертировать

         // вызов делегата
         float area;
         area = (float)ST(aa, bb, cc);

         // вывод результата на форму
         label4.Text = "S = " + area.ToString();
      }
   }
}
```

4. Запуск программы на выполнение

После выполненных действий можно запускать программу на выполнение и тестировать ее работу.

6.5 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

7. ЛАБОРАТОРНАЯ РАБОТА № 6. ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ ДЛЯ ВЫЧИСЛЕНИЯ ХАРАКТЕРИСТИК ГЕОМЕТРИЧЕСКИХ ФИГУР

7.1 Общие сведения

Цель: научиться использовать делегаты в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 5.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 1 ч.
Время самостоятельной подготовки: 1 ч.

7.2 Теоретические сведения:

Делегат – это объект, который может ссылаться на метод. Фактически делегат инкапсулирует метод. После создания делегата, получается объект, который содержит ссылку на метод. С помощью делегата можно вызвать метод, на который этот делегат ссылается (рисунок 25).

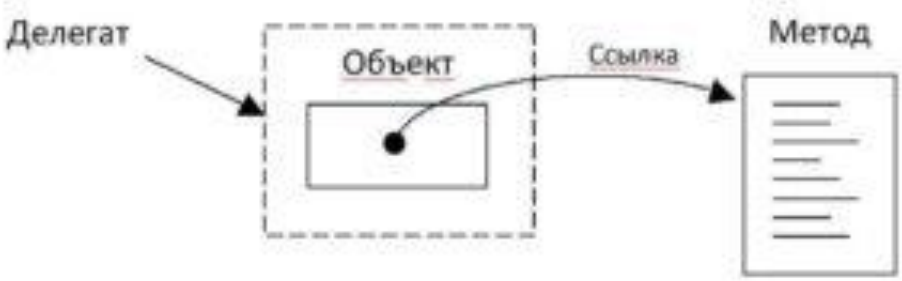


Рисунок 25 - Назначение делегата

Чтобы получить делегат, сначала нужно объявить его тип. Тип делегата объявляется с помощью ключевого слова delegate. Ниже приведена общая форма объявления делегата:
delegate *возвращаемый_тип* ***имя***(*список_параметров*);
где:

- *возвращаемый_тип* – обозначает тип значения, которое возвращается методами, которые будут вызваться с помощью делегата;
- *имя* – непосредственное имя типа делегата. С помощью этого имени объявляются делегаты таким самым образом, как объявляются обычные переменные;
- *список_параметров* – параметры, которые необходимы для методов, которые будут вызываться с помощью делегата.

После объявления типа делегата можно объявлять сам делегат. Общая форма объявления делегата (объекта) точно такая же, как в случае объявления переменной некоторого типа:

тип_делегата **имя1 [, имя2, ...]**;
где:

- *тип_делегата* – название типа делегата, который объявляется с помощью ключевого слова delegate (см. п. 2);
- *имя1* , *имя2*, … – имена делегатов (переменных типа «делегат», объектов, и т.п.). С помощью этих имен можно иметь доступ к методам, на которые ссылаются делегаты.

Делегат может вызвать разные методы. Если делегат ссылается на *метод1*, то чтобы вызвать *метод2* с помощью делегата, нужно в этом делегате изменить ссылку на этот метод (*метод2*)(рисунок 26).

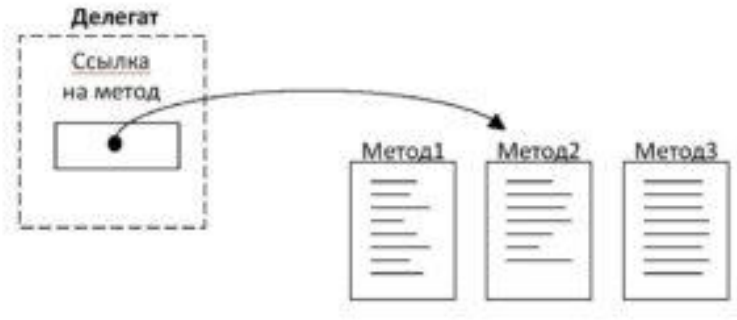


Рисунок 26 - Делегат ссылается на *Метод2*

7.3 Задание к лабораторной работе

Разработать приложение, которое вычисляет характеристики геометрических фигур с использованием делегатов. Приложение реализовать в Microsoft Visual Studio по шаблону Windows Forms Application.

В приложении типа Windows Forms Application объявить тип делегата, который ссылается на метод. Требования к сигнатуре метода следующие:

- метод получает входным параметром переменную типа double;
- метод возвращает значение типа double, которое есть результатом вычисления.

Реализовать вызов методов с помощью делегата, которые получают радиус *R* и вычисляют:

- длину окружности по формуле $\mathbf{D = 2 \cdot \pi \cdot R}$;
- площадь круга по формуле $\mathbf{S = \pi \cdot R^2}$;
- объем шара. Формула: $\mathbf{V = 4/3 * \pi \cdot R^3}$.

Методы должны быть объявлены как статические (с использованием ключевого слова static). Для работы программы выбираем такие имена:

- название типа делегата – CalcFigure;

- название делегата (экземпляра объекта) – CF;
- название метода, который вычисляет длину окружности – Get_Length();
- название метода, который вычисляет площадь круга – Get_Area();
- название метода, который вычисляет объем шара – Get_Volume().

Объявление типа делегата и методов осуществляется в классе Form1 главной формы приложения типа Windows Forms Application.

Объявление типа делегата

Тип делегата объявляется в некотором классе. Это может быть, например, класс основной формы в случае, если приложение создано по шаблону Windows Forms Application.
В классе нужно объявить тип делегата с именем CalcFigure в соответствии с условием задачи. Фрагмент такого объявления приведен ниже:

```
// объявление типа делегата CalcFigure
delegate double CalcFigure(double r);
```

5.3.2. Объявление методов в классе

Методы в классе объявляются как статические с ключевым словом static.

```
// объявление статических методов в классе
// длина окружности
public static double Get_Length(double r)
{
   double length;
   length = 2 * 3.1415 * r;
   return length;
}

// площадь круга
public static double Get_Area(double r)
{
   double area;
   area = 3.1415 * r * r;
   return area;
}

// объем шара
public static double Get_Volume(double r)
{
   double volume;
   volume = 3.1415 * r * r * r * 4.0 / 3.0;
   return volume;
}
```

Демонстрация вызова методов с помощью делегата

Демонстрация вызова методов с помощью делегата из другого программного кода, например из обработчика события клика на кнопке (шаблон Windows Forms Application).

```
// обработчик события button1_Click()
private void button1_Click(object sender, EventArgs e)
```

```
{
   double radius, length, area, volume;
   radius = Convert.ToDouble(textBox1.Text);

   // вызов делегата
   // сконструировать делегат
   CalcFigure CF = new CalcFigure(Get_Length);
   length = CF(radius); // 1. Вызов метода Get_Length()

   CF = new CalcFigure(Get_Area);
   area = CF(radius); // 2. Вызов метода Get_Area()

   CF = new CalcFigure(Get_Volume);
   volume = CF(radius); // 3. Вызов метода Get_Volume()

   label2.Text = length.ToString();
   label3.Text = area.ToString();
   label4.Text = volume.ToString();
}
```

На рисунке27 схематично изображена работа делегата CF.

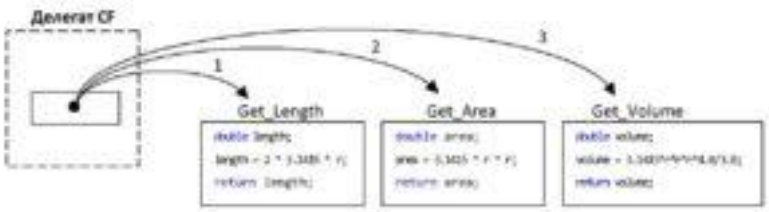


Рисунок 27 - Делегат CF

Групповое преобразование делегированных методов позволяет упростить строку присвоения делегату метода. В этом случае ключевое слово new опускается.
В вышеприведенном примере (см. п. 5) строку

```
CF = new CalcFigure(Get_Length);
```

можно заменить строкой

```
CF = Get_Length;
```

что упрощает программный код и наглядность его отображения.

Учитывая вышесказанное, соответствующий обработчик события button1_Click() может иметь следующий вид:

```
// групповое преобразование делегатов
private void button1_Click(object sender, EventArgs e)
{
   double radius, length, area, volume;
   radius = Convert.ToDouble(textBox1.Text);

   // замена оператора new на оператор присваивания
```

```
    // сконструировать делегат с именем CF
    CalcFigure CF = Get_Length; // не нужно оператора new - упрощение кода
    length = CF(radius); // Вызов метода Get_Length()

    CF = Get_Area; // делегат CF ссылается на Get_Area
    area = CF(radius); // Вызов метода Get_Area()

    CF = Get_Volume; // делегат CF ссылается на Get_Volume
    volume = CF(radius); // Вызов метода Get_Volume()

    label2.Text = length.ToString();
    label3.Text = area.ToString();
    label4.Text = volume.ToString();
}
```

7.4 Методические указания и порядок выполнения работы

1. Запустить Microsoft Visual Studio. Создать проект (приложение) по шаблону Windows Forms Application

Создать проект по шаблону Windows Forms Application. Имя модуля главной формы «Form1.cs». Имя проекта можно задать, например, «Delegates01». Имя экземпляра объекта формы приложения Form1.

Окно новосозданной формы приложения изображено на рисунке 28.

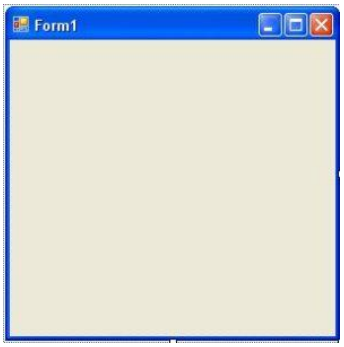


Рисунок 28 - Окно главной формы приложения Form1

После создания проекта текст модуля основной формы приложения следующий:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Delegates01
{
    public partial class Form1 : Form
```

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

2. Разработка формы приложения

Разместить на форме приложения следующие элементы управления:

- четыре элемента управления типа Label. В результате будет создано 4 экземпляра (объекта) класса Labelс именами label1, label2, label3, label4;
- элемент управления типа TextBox с именем textBox1;
- элемент управления типа Button с именем button1.

С помощью окна Properties настроить следующие свойства элементов управления:

- в элементе управления label1 свойство Text = «R = « (label1.Text = «R = «);
- label2.Text = «Длина окружности = «;
- label3.Text = «Площадь круга = «;
- label4.Text = «Объем шара = «;
- в элементе управления Form1 (главная форма) свойство Text = «Расчет»;
- в Form1 свойство StartPosition = CenterScreen;
- в элементе управления button1 свойство Text = «Вычислить».

После выполненных действий, форма приложения будет иметь вид как изображено на рисунке 29.

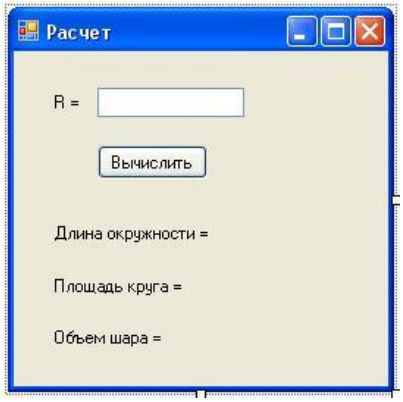


Рисунок 29 - Форма приложения после настройки и размещения элементов управления

3. Написание программного кода

3.1. Объявление типа делегата

В соответствии с соображениями, тип делегата носит имя CalcFigure и объявляется в классе Form1 (файл«Form1.cs») перед конструктором формы Form1(). Текст объявления делегата следующий:

```
// объявление типа делегата CalcFigure
delegate double CalcFigure(double r);
```

При объявлении типа делегата используется ключевое слово delegate.

3.2. Объявление методов вычисления характеристик геометрических фигур

Методы объявляются в классе формы после реализации конструктора формы Form1(). В соответствии с условием задачи, методы в классе объявляются как статические с ключевым словом static.

После объявления методов текст модуля «Form1.cs» имеет вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Delegates01
{
   public partial class Form1 : Form
   {
      // объявление типа делегата CalcFigure
      delegate double CalcFigure(double r);

      public Form1()
      {
         InitializeComponent();
      }

      // объявление статических методов в классе
      // длина окружности
      public static double Get_Length(double r)
      {
         double length;
         length = 2 * 3.1415 * r;
         return length;
      }

      // площадь круга
      public static double Get_Area(double r)
      {
         double area;
         area = 3.1415 * r * r;
         return area;
      }

      // объем шара
      public static double Get_Volume(double r)
      {
         double volume;
         volume = 3.1415 * r * r * r * 4.0 / 3.0;
         return volume;
      }
   }
}
```

3.3. Программирование события Click, которое вызывается при клике на кнопке button1(вариант 1)

Следующий шаг – программирование события клика на кнопке «Вычислить» (элемент управления button1).

Текст обработчика события клика на кнопке button1 следующий:

```
private void button1_Click(object sender, EventArgs e)
{
   double radius, length, area, volume;

   // считать радиус
   radius = Convert.ToDouble(textBox1.Text);

   // вызов делегата
   // сконструировать делегат
   CalcFigure CF = new CalcFigure(Get_Length);

   length = CF(radius); // 1. Вызов метода Get_Length()

   CF = new CalcFigure(Get_Area);
   area = CF(radius); // 2. Вызов метода Get_Area()

   CF = new CalcFigure(Get_Volume);
   volume = CF(radius); // 3. Вызов метода Get_Volume()

   // вывод результатов на форму
   label2.Text = "Длина окружности = " + length.ToString();
   label3.Text = "Площадь круга = " + area.ToString();
   label4.Text = "Объем шара = " + volume.ToString();
}
```

Объясним некоторые фрагменты кода.

Радиус окружности считывается во внутреннюю переменную radius. Преобразование из строки в текст осуществляется с помощью класса Convert (метод ToDouble()).

Следующим шагом строится делегат с именем CF. Этот делегат инициализируется значением Get_Length. Это означает, что делегат ссылается на метод Get_Length().

На следующих шагах происходит переопределение делегата CF соответственно методами Get_Area() и Get_Volume().

Следующий пункт отображает другой вариант реализации обработчика события.

3.4. Программирование события Click, которое вызывается при клике на кнопке button1(вариант 2)

Обработчик события клика на кнопке button1 может иметь и другую реализацию, которая есть более наглядной и упрощенной.

```
private void button1_Click(object sender, EventArgs e)
{
   double radius, length, area, volume;
```

```
    // взять радиус
    radius = Convert.ToDouble(textBox1.Text);

    // вызов делегата
    // сконструировать делегат
    CalcFigure CF = Get_Length;

    length = CF(radius); // 1. Вызов метода Get_Length()

    CF = Get_Area;
    area = CF(radius); // 2. Вызов метода Get_Area()

    CF = Get_Volume;
    volume = CF(radius); // 3. Вызов метода Get_Volume()

    // вывод результата на форму
    label2.Text = "Длина окружности = " + length.ToString();
    label3.Text = "Площадь круга = " + area.ToString();
    label4.Text = "Объем шара = " + volume.ToString();
}
```

В этом случае происходит так называемое групповое преобразование делегированных методов. При таком преобразовании ключевое слово new опускается. Программный код упрощается.

3.5. Текст модуля Form1.cs

Текст всего модуля Form1.cs решения данной задачи (вариант 2) имеет вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Delegates01
{
    public partial class Form1 : Form
    {
        // объявление типа делегата CalcFigure
        delegate double CalcFigure(double r);

        public Form1()
        {
            InitializeComponent();
        }

        // объявление статических методов в классе
        // длина окружности
        public static double Get_Length(double r)
```

```
        {
            double length;
            length = 2 * 3.1415 * r;
            return length;
        }

        // площадь круга
        public static double Get_Area(double r)
        {
            double area;
            area = 3.1415 * r * r;
            return area;
        }

        // объем шара
        public static double Get_Volume(double r)
        {
            double volume;
            volume = 3.1415 * r * r * r * 4.0 / 3.0;
            return volume;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double radius, length, area, volume;

            // взять радиус
            radius = Convert.ToDouble(textBox1.Text);

            // вызов делегата
            // сконструировать делегат
            CalcFigure CF = Get_Length;
            length = CF(radius); // 1. Вызов метода Get_Length()

            CF = Get_Area;
            area = CF(radius); // 2. Вызов метода Get_Area()

            CF = Get_Volume;
            volume = CF(radius); // 3. Вызов метода Get_Volume()

            // вывод результата на форму
            label2.Text = "Длина окружности = " + length.ToString();
            label3.Text = "Площадь круга = " + area.ToString();
            label4.Text = "Объем шара = " + volume.ToString();
        }
    }
}
```

4. Запуск программы на выполнение

Окно программы после запуска изображено на рисунке 30.

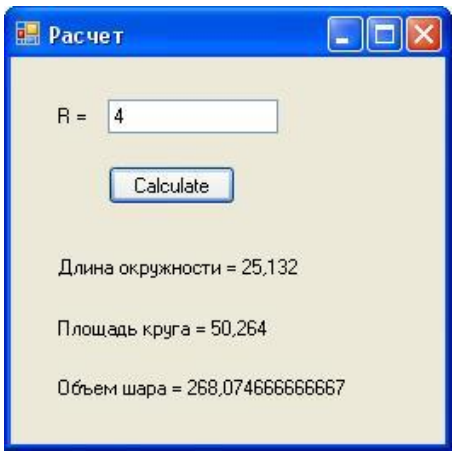


Рисунок 30 - Выполнение программы

7.5 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

8. ЛАБОРАТОРНАЯ РАБОТА № 7. РАЗРАБОТКА ПРОГРАММЫ ЧТЕНИЯ И ЗАПИСИ ТЕКСТОВОГО ФАЙЛА.

8.1 Общие сведения

Цель: научиться создавать приложение для чтения и записи текстового файла в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной работы № 6.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 2 ч.
Время самостоятельной подготовки: 2 ч.

8.2 Задание к лабораторной работе

Разработать программу чтения/записи текстовых файлов. В программе должна быть возможность выбора файла для чтения, его корректировка и запись этого файла на диск.

8.3 Методические указания и порядок выполнения работы

1. Запустить MS Visual Studio. Создать проект по шаблону Windows Forms Application
2. Разработка формы приложения

Создать форму как показано на рисунке 31.

На форме размещаются следующие элементы управления:

- два элемента управления типа Button. Автоматически будут созданы два объекта (экземпляры класса Button) с именами button1 и button2;
- один элемент управления типа RichTextBox. Автоматически создается объект с именем richTextBox1;
- элемент управления типа Label. Создается объект с именем label1.

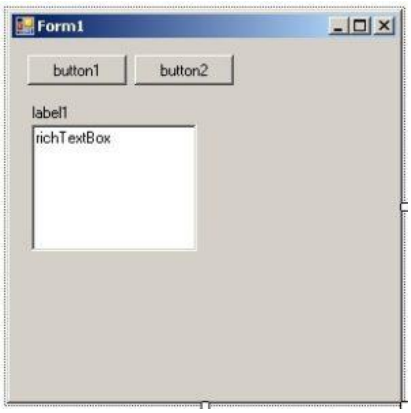


Рисунок 31 - Элементы управления формы приложения

Осуществить настройку элементов управления типа Button следующим образом:

- в элементе управления button1 свойство Text = “Открыть файл …»;
- в элементе управления button2 свойство Text = “Сохранить файл».

Настройка формы приложения Form1:

- свойство Text = “Чтение/запись текстовых файлов”;
- свойство MaximizeBox = false (кнопка развертывания на весь экран становится неактивной);
- свойство FormBorderStyle = “Fixed3D”;
- свойство StartPosition = “CenterScreen”.

Также нужно откорректировать размеры формы и элементов управления на форме приблизительно так, как показано на рисунке 32.

В элементе управления RichTextBox (рисунок 33):

- свойство WordWrap = «false» (перенос длинных строк в пределах границ окна редактора).

Элемент управления RichTextBox представляет собой многострочное редактированное текстовое поле, которое работает с текстом в формате RTF (Rich Text Format – расширенный текстовый формат). Текст формата RTF сохраняет дополнительную служебную информацию, которая управляет свойствами каждого абзаца и изменением шрифта по ходу текста.

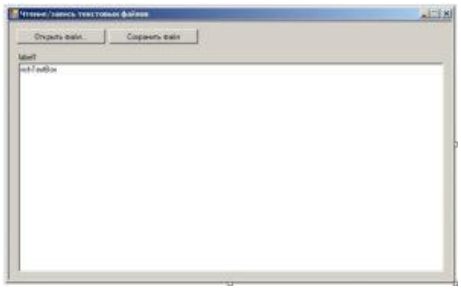


Рисунок 32 - Форма приложения после корректировки и настройки свойств

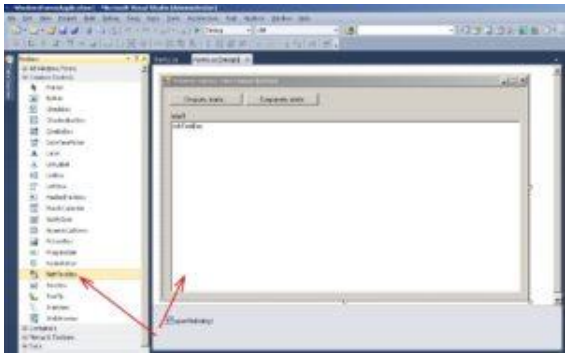


Рисунок 33 - Элемент управления типа RichTextBox

3. Элемент управления OpenFileDialog

Для того, чтобы выбрать текстовый файл для чтения, нужно использовать элемент управления типа OpenFileDialog. Этот элемент управления представляет собой стандартное диалоговое окно Windows,предназначенное для открытия файлов.
Разместить на форме компонент OpenFileDialog (рисунок 34).

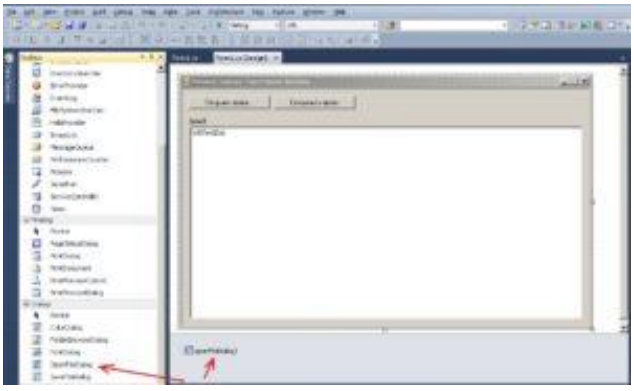


Рисунок 34 - Элемент управления OpenFileDialog

4. Добавление внутренних переменных в текст приложения

Для работы программы нужно ввести следующие дополнительные внутренние переменные:

- f_open – определяет, выбрал ли пользователь файл командой «Открыть файл…»;
- f_save – определяет, был ли сохранен ранее открытый файл (f_open = true).

Поэтому, в текст модуля “Form1.cs” нужно ввести следующий код:

...

```
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      bool f_open, f_save; // определяют выбор файла и его сохранение

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

...

5. Программирование события Load класса формы Form1

Событие Load возникает в момент загрузки формы сразу после запуска приложения на выполнение. В обработчик события целесообразно включать начальную инициализацию переменных.

В данном случае, начальной инициализации подлежат такие переменные и объекты как f_open, f_save, label1, richTextBox1.

В компоненте label1 будет отображаться путь к выбранному файлу. В компоненте richTextBox1 будет отображаться содержимое (текст) выбранного файла.

Листинг обработчика Form1_Load() события Load загрузки формы следующий:

```
private void Form1_Load(object sender, EventArgs e)
{
  f_open = false; // файл не открыт
  f_save = false;
  label1.Text = "-";
  richTextBox1.Text = "";
}
```

6. Импорт пространства имен System.IO

В данной работе для чтения и записи файлов будут использованы возможности классов StreamWriter и StreamReader из библиотеки классов языка C#.

Поэтому, для использования методов этих классов, нужно в начале кода модуля “Form1.cs” добавить строку:

```
using System.IO;
```

Таким образом, верхняя часть файла “Form1.cs” имеет вид:

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
```

7. Программирование события клика на кнопке button1 (“Открыть файл…»)

Для вызова стандартного окна Windows выбора файла, пользователь должен выбрать команду «Открыть файл…» (button1). (из лабы про шар)
Листинг обработчика button1_Click() события Click кнопки button1, осуществляющей открытие окна выбора файла, следующий:

```
private void button1_Click(object sender, EventArgs e)
{
  // 1. Открытие окна и проверка, выбран ли файл
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    // 2. Вывести имя файла на форме в компоненте label1
    label1.Text = openFileDialog1.FileName;

    // 3. Установить флажки f_open и f_save
    f_open = true;
    f_save = false;

    // 4. Прочитать файл в richTextBox1
    // очистить предыдущий текст в richTextBox1
    richTextBox1.Clear();

    // 5. Создать объект класса StreamReader и прочитать данные из файла
    StreamReader sr = File.OpenText(openFileDialog1.FileName);

    // дополнительная переменная для чтения строки из файла
    string line = null;
    line = sr.ReadLine(); // чтение первой строки

    // 6. Цикл чтения строк из файла, если строки уже нет, то line=null
    while (line != null)
    {
      // 6.1. Добавить строку в richTextBox1
      richTextBox1.AppendText(line);

      // 6.2. Добавить символ перевода строки
      richTextBox1.AppendText("\r\n");

      // 6.3. Считать следующую строку
      line = sr.ReadLine();
    }
```

```
    // 7. Закрыть соединение с файлом
    sr.Close();
  }
}
```

Для вызова стандартного окна Windows выбора файла используется метод ShowDialog() компонента openFileDialog1. Выбранный файл сохраняется в свойства FileName объекта openFileDialog.

Для чтения текстового файла используется класс StreamReader, который осуществляет чтение символьных данных из файла. Чтобы создать экземпляр класса StreamReader используется метод OpenText() класса File.Класс File содержит ряд методов, которые хорошо подходят для упрощенного чтения символьных данных.
Чтобы считать строку из файла в программе используется метод ReadLine(), считывающий строку символов из текущего потока и возвращающий данные в виде строки. Если достигнут конец файла, то метод возвращает null.

Чтение строк осуществляется в локальную переменную line.

Чтобы добавить строку к объекту richTextBox1, используется метод AppendText().

8. Программирование события изменения текста в компоненте RichTextEdit

В соответствии с логикой программы, если в тексте файла происходят изменения, то флажок f_save должен быть равен значению false.

В компоненте richTextBox1 есть событие TextChanged, которое вызывается в момент изменения текста в редакторе (рисунок 35).

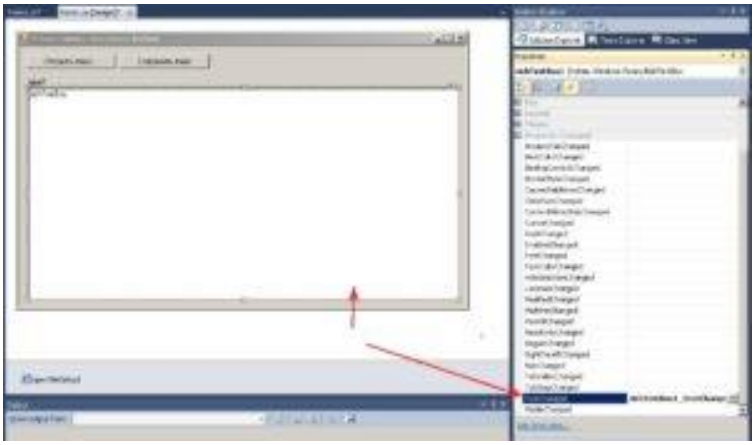


Рисунок 35 - Событие TextChanged элемента управления richTextBox1

Обработчик события TextChanged имеет следующий вид:

```
private void richTextBox1_TextChanged(object sender, EventArgs e)
{
  f_save = false;
}
```

9. Программирование события клика на кнопке “Сохранить файл”

Для сохранения измененного текста в файле нужно выбрать команду «Сохранить файл» (кнопка button2).Для сохранения измененного в richTextBox1 файла используются методы класса StreamWriter.

Листинг обработчика события клика на кнопке button2 следующий:

```
private void button2_Click(object sender, EventArgs e)
{
  // 1. Проверка, открыт ли файл
  if (!f_open) return;

  // 2. Если файл открыт, то проверка – сохранен ли он
  if (f_save) return;

  // 3. Создание объекта типа StreamWriter и получение строчных данных
  StreamWriter sw = File.CreateText(openFileDialog1.FileName);

  // 4. Чтение строк с richTextBox1 и добавление их в файл
  string line;
  for (int i = 0; i < richTextBox1.Lines.Length; i++)
  {
    // 4.1. Чтение одной строки
    line = richTextBox1.Lines[i].ToString();

    // 4.2. Добавление этой строки в файл
    sw.WriteLine(line);
  }

  // 5. Закрыть объект sw
  sw.Close();
}
```

Объясним некоторые фрагменты кода.

В обработчике события button2_Click после проверки на наличие открытого файла создается объект (переменная) с именем sw типа StreamWriter.

При создании объекта используется метод CreateText() из типа File, возвращающего экземпляр типа StreamWriter. Имя файла сохраняется в свойстве openFileDialog1.FileName.

Для доступа к введенным строкам компонента richTextBox используется свойство Lines, которое есть массивом строк.

Чтобы добавить одну строку, нужно вызвать метод WriteLine() объекта sw типа StreamWriter.

8.1 8.5 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

9. ЛАБОРАТОРНАЯ РАБОТА № 8. ПОДКЛЮЧЕНИЕ БАЗЫ ДАННЫХ MICROSOFT ACCESS

9.1 Общие сведения

Цель: научиться подключать базу данных Microsoft Access к приложению в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной работы № 7.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 1 ч.
Время самостоятельной подготовки: 1 ч.

9.2 Задание к лабораторной работе

Пусть имеется некоторая база данных, созданная в СУБД Microsoft Access. Файл базы данных имеет имя «db1.mdb«. Путь к файлу базы данных

E:\Programs\C_Sharp\WindowsFormsApplication1\db1.mdb

База данных имеет одну таблицу с именем «Tovar».

Необходимо осуществить подключение базы данных к Windows-приложению на языке C# средствами Microsoft Visual Studio 2010. Приложение должно быть реализовано как Windows Forms Application.

9.3 Методические указания и порядок выполнения работы

1. Создание приложения типа Windows Forms Application.
Запустить MS Visual Studio. **Создать приложение Windows Forms Application**.

2. Вызов мастера подключения.

Для доступа к файлу базы данных необходимо сделать его подключение к приложению. Это осуществляется путем вызова команды «Add New Data Source…» из меню «Data» (рисунок 36) либо кликом на крайней левой кнопке с панели инструментов Data Source.

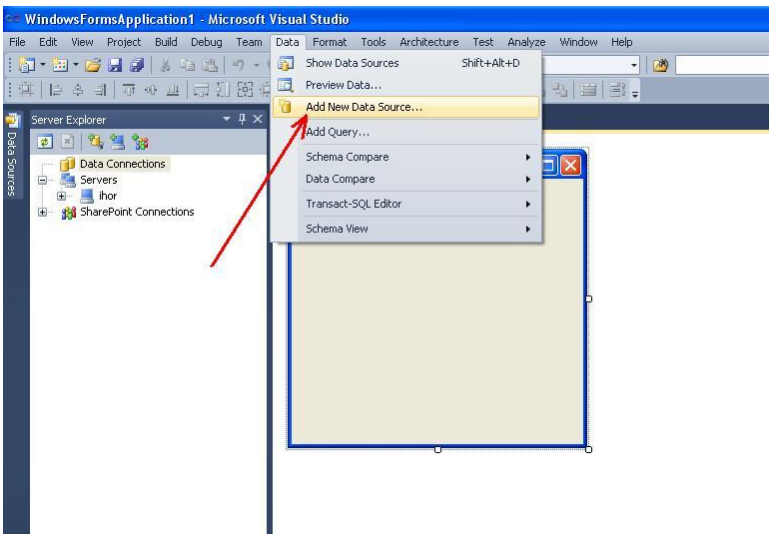


Рисунок 36 - Вызов мастера подключения к файлу базы данных

3. Выбор типа источника данных.

В результате откроется окно мастера для подключения к источнику данных которое изображено на рисунке 37.

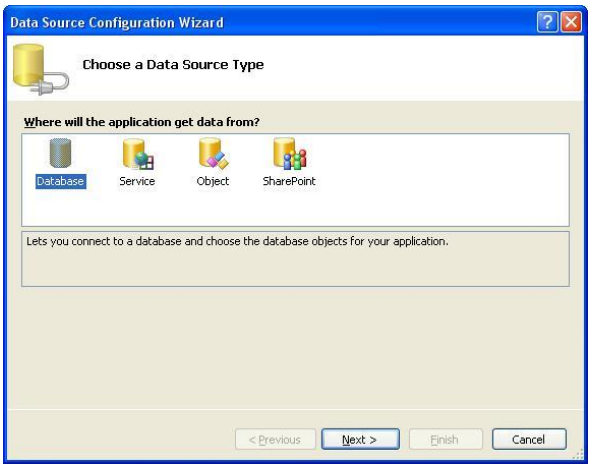


Рисунок 37 - Выбор типа подключения из которого приложение будет получать данные

В окне необходимо выбрать один из четырех возможных вариантов подключения к источнику данных. В MS Visual Studio существует четыре типа подключения к источникам данных:

- Database – подключение к базе данных и выбор объектов базы данных;
- Service – открывает диалоговое окно Add Service Reference позволяющее создать соединение с сервисом, который возвращает данные для вашей программы;
- Object – позволяет выбрать объекты нашего приложения, которые в дальнейшем могут быть использованы для создания элементов управления (controls) с привязкой к данным;
- Share Point – позволяет подключиться к сайту SharePoint и выбрать объекты для вашей программы.

В нашем случае выбираем элемент Database и продолжаем нажатием на кнопке Next.

4. Выбор модели подключения к базе данных.

Следующий шаг – выбор модели подключения к базе данных (рисунок 38).

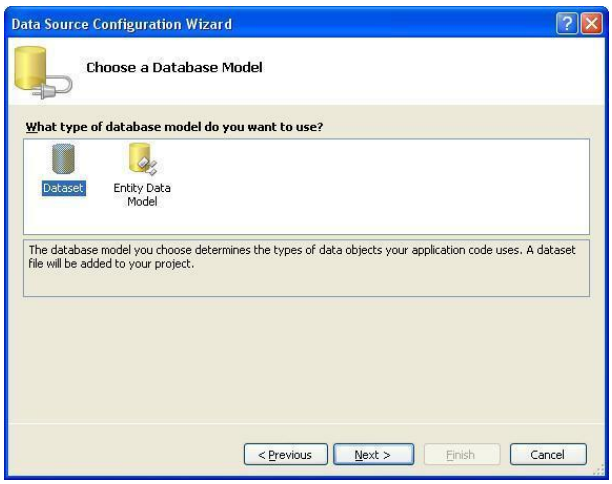


Рисунок 38 - Выбор модели подключения к базе данных

Система предлагает выбор одного из двух вариантов:

- модели данных на основе набора данных (Dataset);
- модели данных Entity, что означает, что система может сгенерировать модель данных из базы данных которой могут выступать сервера баз данных Microsoft SQL Server, Microsoftt SQL Server Compact 3.5 или Microsoft SQL Server Database File, либо создать пустую модель как отправную точку для визуального проектирования концептуальной модели с помощью панели инструментов.

В нашем случае выбираем тип модели данных DataSet.

5. Задание соединения с БД.

Следующим шагом мастера (рисунок 39) есть выбор соединения данных, которое должно использоваться приложением для соединения с базой данных.

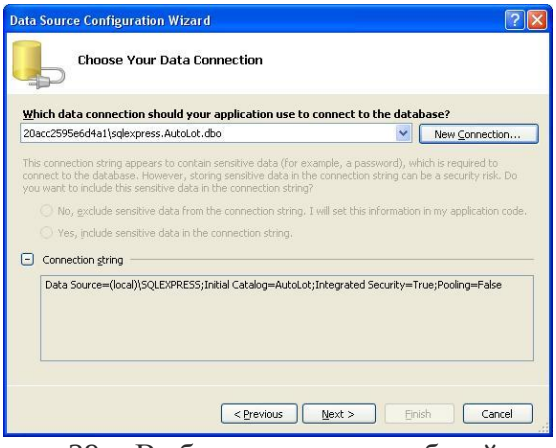


Рисунок 39 - Выбор соединения с базой данных

Для создания нового соединения необходимо выбрать кнопку «New Connection…«. В результате откроется окно «Add Connection» (рисунок 5) в котором нужно добавить новое соединение Microsoft Access и выбрать маршрут к файлу базы данных.
В нашем случае поле «Data source» уже содержит нужный нам тип соединения «Microsoft Access Database File (OLE DB) ».

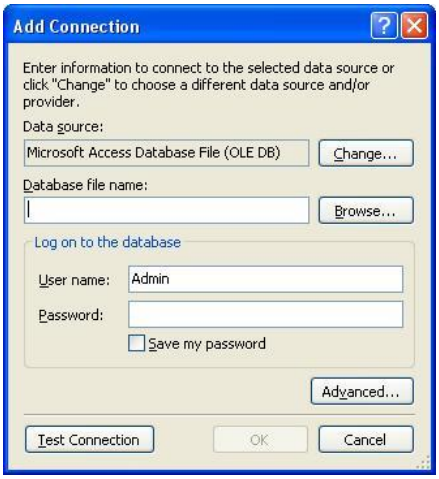


Рисунок 40 - Добавление нового соединения и выбор файла базы данных

Если нужно выбрать другую базу данных, то для этого используется кнопка «Change…«, которая открывает окно, изображенное на рисунке 42.

Рисунок 42 - Смена источника данных

В окне на рисунке 6 системой Microsoft Visual Studio будет предложено следующие виды источников данных:

- Microsoft Access Database File – база данных Microsoft Access;
- Microsoft ODBC Data Source – доступ к базе данных с помощью программного интерфейса ODBC (Open Database Connectivity);
- Microsoft SQL Server;
- Microsoft SQL Server Compact 3.5;
- Microsoft SQL Server Database File;
- Oracle Database – база данных Oracle.

Нажимаем кнопку «Browse…» и в открывшемся окне (рисунок 43) «Add Connection» выбираем маршрут к файлу базы данных «db1.mdb«. Целесообразно размещать файл базы данных в каталоге содержащим исполняемый модуль приложения.
Для проверки правильности установленного соединения можно воспользоваться кнопкой «Test Connection«.

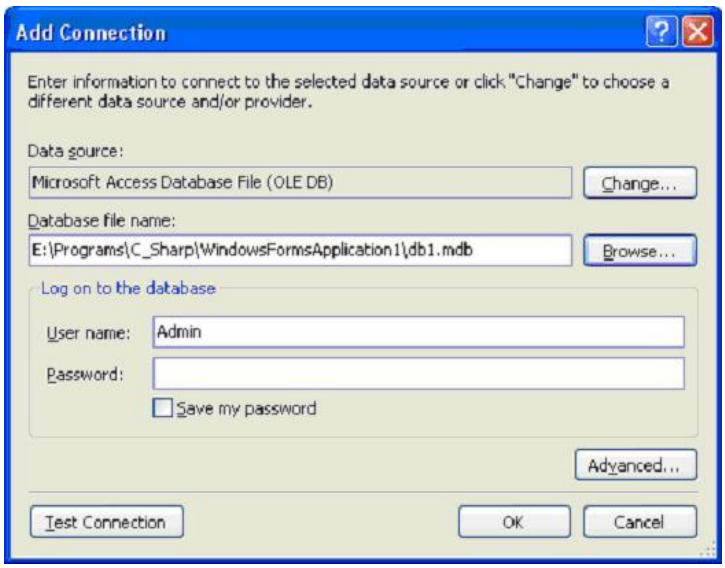


Рисунок 43 - Окно «Add Connection» с выбранной базой данных «db1.mdb«

После нажатия на кнопке ОК система сгенерирует строку «Connection string» (рисунок 44) который в дальнейшем будет использован для программного подключения к базе данных.
Кликаем на «Next» для продолжения работы мастера.

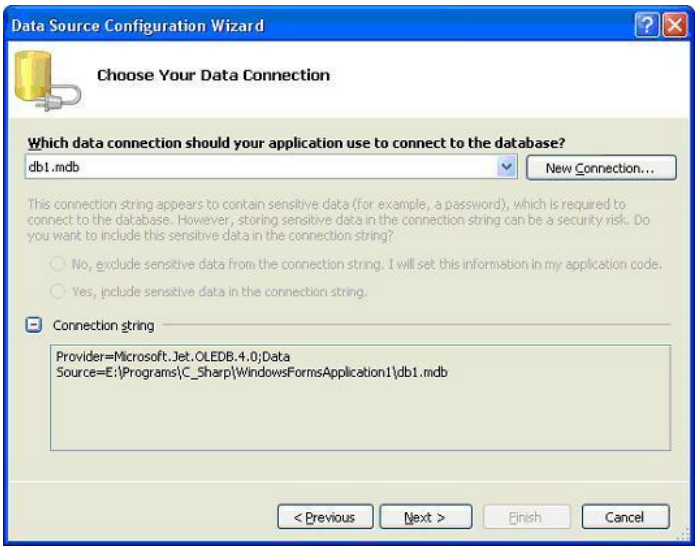


Рисунок 44 - Строка Connection string

После выбора Next система выдаст информационное окно следующего вида (рисунок 45). Если выбрать «Да», то файл базы данных «db1.mdb» будет копироваться в выходной каталог приложения каждый раз при его запуске в среде MS Visual Studio. Как правило, это каталог, содержащий основные модули приложения. В нашем случае каталог

E:\Programs\C_Sharp\WindowsFormsApplication1\WindowsFormsApplication1

В этом каталоге размещаются все основные исходные модули проекта, например Program.cs (модуль, содержащий основную функцию WinMain()), Form1.cs (содержит исходный код обработки главной формы приложения) и другие.

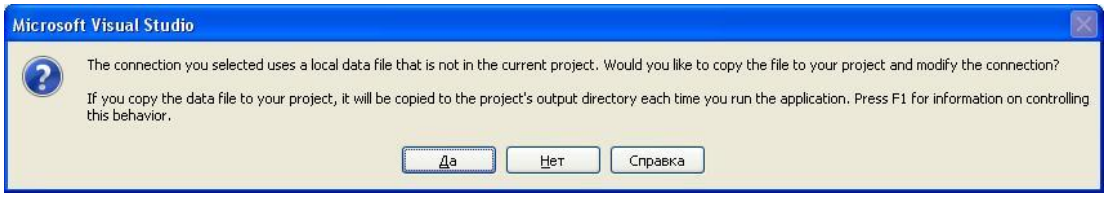


Рисунок 45 - Окно добавления файла базы данных в проект

6. Формирование конфигурационного файла приложения.

После выбора кнопки «Next» мастера откроется следующее окно, в котором предлагается сохранить строку соединения в конфигурационный файл приложения (рисунок 46).

Рисунок 46 - Предложение записи строки подключения к базе данных в конфигурационный файл приложения

Ничего не изменяем, оставляем все как есть (кликаем на Next).

7. Выбор объектов базы данных для использования в программе

Последнее окно мастера (рисунок 47) предлагает выбрать список объектов (таблиц, запросов, макросов, форм и т. д.), которые будут использоваться в наборе данных. Как правило выбираем все таблицы базы данных. В нашем примере база данных содержит всего одну таблицу с именем Tovar.

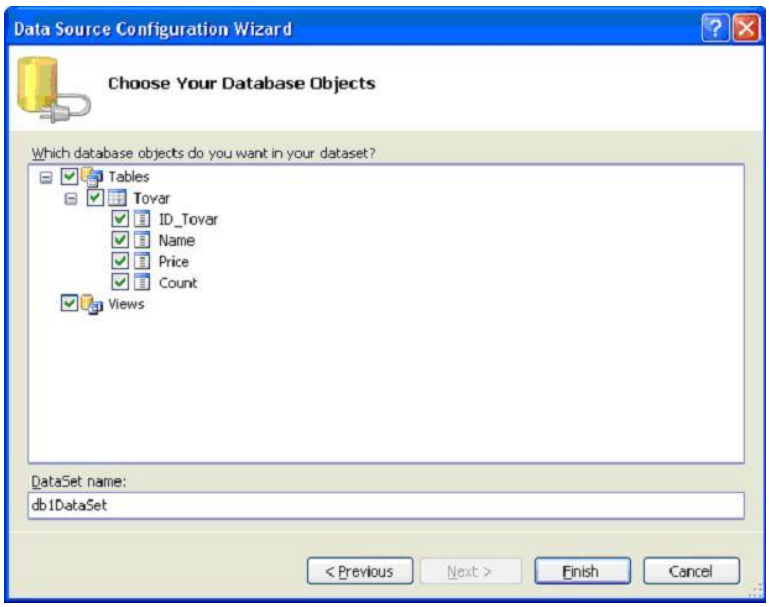


Рисунок 47 - Выбор объектов базы данных, которые будут использоваться в данном наборе данных

После выбора кнопки «Finish» заканчиваем работу с мастером подключения. Теперь база данных подключена к приложению и будет автоматически подключаться при его запуске или при его проектировании в MS Visual Studio.

8. Что же изменилось в программе после выполнения мастера?

Если выбрать панель Data Source (рисунок 48), то можно увидеть, как подключен набор данных с именем db1Dataset , в котором есть таблица с именем Tovar.

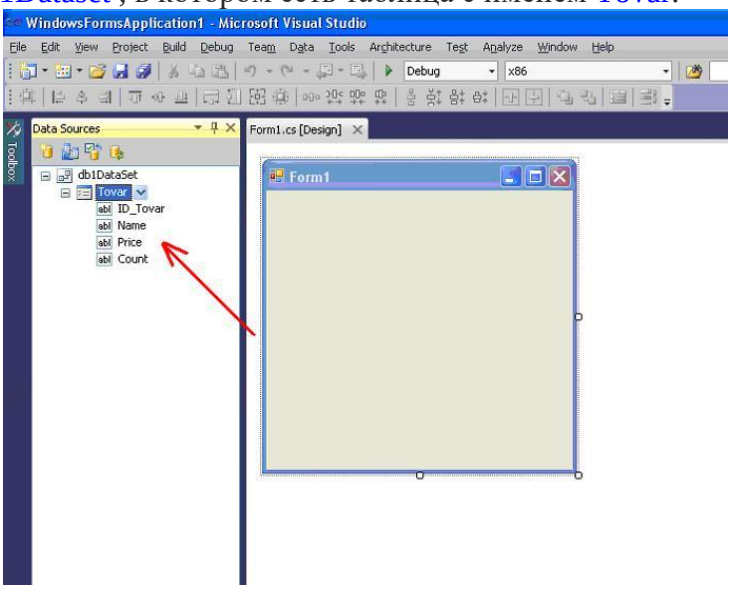


Рисунок 48 - Окно DataSources содержит подключение к базе данных

Точно также можно увидеть изменентия в панели Server Explorer (рисунок 49), где появилась база данных «db1.mdb» с таблицей Tovar и ее полями. Приложение может подключать не только одну, но и несколько баз данных.

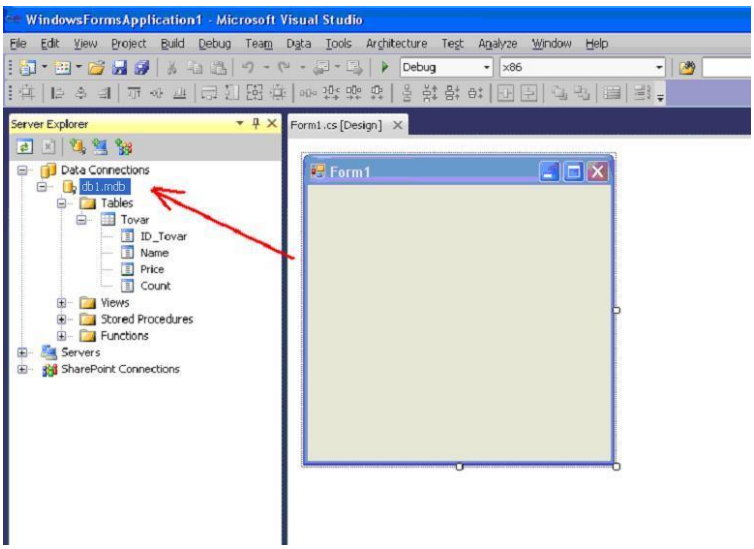


Рисунок 49 - Окно приложения с изменениями в панели Server Explorer

9. Подключение методов оперирования базой данных.

Для того, чтобы использовать методы, которые будут работать с базой данных MS Access (и не только MS Access), необходимо подключить пространство имен System.Data.OleDb.
Для этого в основной форме (Form1.cs) в Solution Explorer выбираем режим просмотра кода (View Code) из контекстного меню (рисунок 50) и вначале файла добавляем следующую строку:

```
using System.Data.OleDb;
```

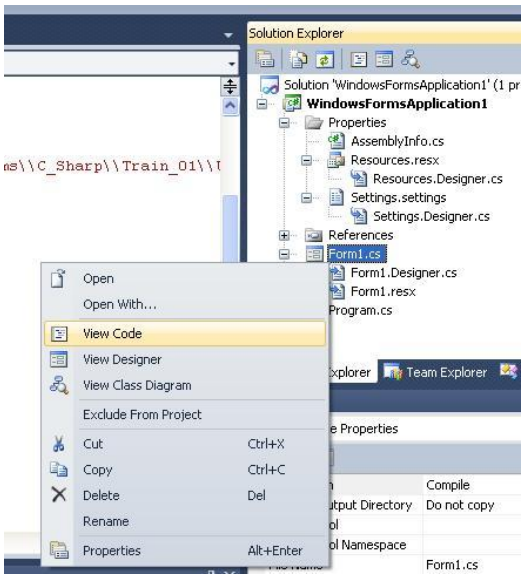


Рисунок 50 - Вызов программного кода главной формы приложения (Form1.cs) с помощью Solution Explorer

Общий вид верхней части файла Form1.cs будет следующим:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
```

На этом этапе подключение к базе данных db1.mdb выполнено. Дальнейшими шагами есть создание программного кода для оперирования данными в базе данных.

9.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

10. ЛАБОРАТОРНАЯ РАБОТА № 9. ВЫВОД ТАБЛИЦЫ БАЗЫ ДАННЫХ MICROSOFT ACCESS В КОМПОНЕНТЕ DATAGRIDVIEW

10.1 Общие сведения

Цель: научиться выводить таблицы базы данных Microsoft Access в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 8.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 1 ч.
Время самостоятельной подготовки: 1 ч.

10.2 Задание к лабораторной работе

Пусть имеется база данных, созданная в приложении Microsoft Access.
Имя файла базы данных “mydb.mdb”. Файл размещается на диске по следующему пути:

C:\Programs\C_Sharp\WindowsFormsApplication1\mydb.mdb

База данных имеет несколько таблиц, одна из которых имеет название “Order”.
Задача состоит в том, чтобы с помощью средств языка C# осуществить подключение к базе данных и вывести таблицу с именем «Order» на форму.
Приложение реализовать как Windows Forms Application.

Общий вид таблиц и связей между ними изображен на рисунке 52.

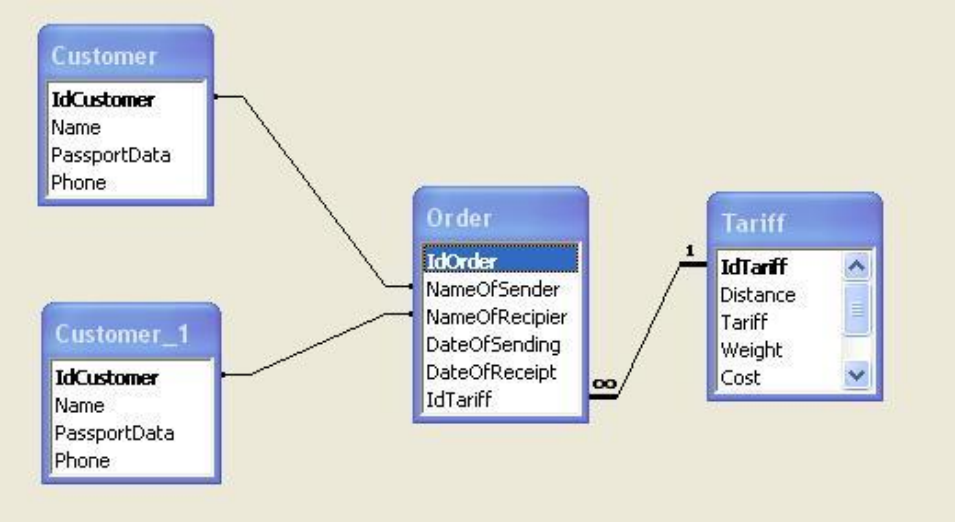


Рисунок 51 - Связи между таблицами базы данных

10.3 Методические указания и порядок выполнения работы

1. Создание приложения.

Загружаем MS Visual Studio. Подробный пример создания приложения по шаблону Windows Forms описывается **здесь**.
Исходный код формы приложения имеет вид (файл Form1.cs):

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
  }
}
```

Как видно из листинга, в пространстве имен WindowsFormsApplication1 есть только конструктор формы, в котором вызывается метод InitializeComponent().

---

2. Подключение к базе данных. Чтение строки подключения Connection String.

Осуществим подключение базы данных MS Access к нашему приложению.
В итоге получаем строку подключения к базе данных Connection String. Эта строка в дальнейшем будет использована в нашем приложении.

Чтобы получить корректную строку подключения к базе данных, нужно выделить базу данных в панели Server Explorer (mydb.mdb) и в окне “Properties” прочитать (скопировать) значение свойства “Connection String” (рисунок 52, красное выделение). Следует учесть, что слеш ‘\’ в строке на C# нужно заменить на ‘\\’ (два слеша) согласно синтаксису языка.

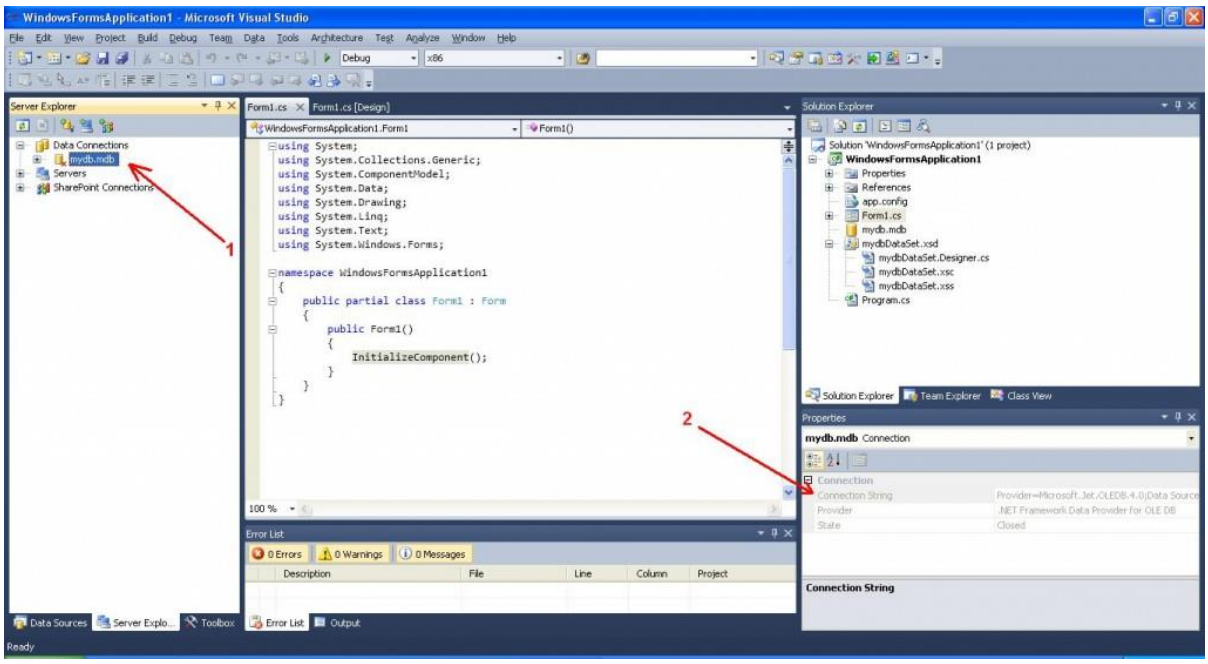


Рисунок 52 - Чтение свойства Connection String

3. Размещение компонента типа dataGridView.

Выносим на форму компонент dataGridView (рисунок 53), представляющий компонент-таблицу, в которой будет выведена наша таблица “Order” из базы данных. Получаем объект-переменную под названием dataGridView.

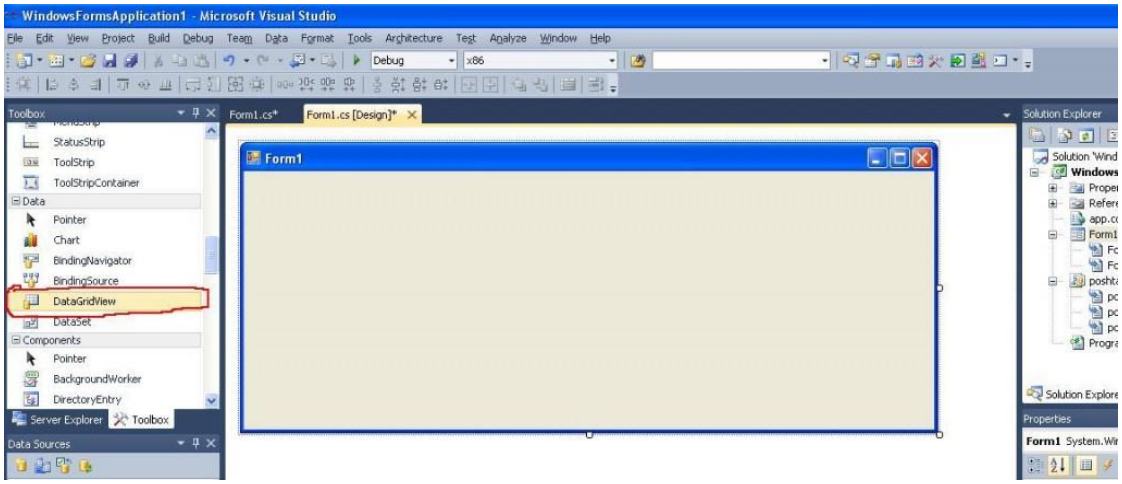


Рисунок 53 - Компонент DataGridView на панели Toolbox

Размещение компонента dataGridView на форме изображено на рисунке 54.

Рисуок 54 - Компонент dataGridView на главной форме приложения

4. Изменение программного кода.

4.1. Добавление переменных SQL-запроса и строки подключения к базе данных. Активируем текст модуля Form1.cs (главная форма) с помощью Solution Explorer. В программный код формы вводим дополнительные переменные CmdText и ConnString. Переменная CmdText будет содержать строку SQL-запроса для вывода всех записей таблицы “Order”.Переменная ConnString представляет собой строку подключения к базе данных (см. п. 2). Общий вид программного кода класса формы следующий:

```
public partial class Form1 : Form
{
 public string CmdText = "SELECT * FROM [Order]";
 public string ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;
DataSource=C:\\Programs\\C_Sharp\\WindowsFormsApplication1\\mydb.mdb";
  public Form1()
  {
    InitializeComponent();
  }
}
```

4.2. Подключение пространства имен OleDb.

В Microsoft Visual Studio взаимодействие с файлом данных Microsoft Access осуществляется с помощью поставщика данных OLE DB или ODBC. Поставщик данных OLE DB обеспечивает доступ к данным, находящимся в любом хранилище данных, если оно поддерживает классический протокол OLE DB на основе технологии COM. Этот поставщик состоит из типов, которые определены в пространстве имен System.Data.OleDb.

В последующих шагах мы будем использовать методы из этого пространства имен. Поэтому, вначале файла Form1.cs после строки
using System.Windows.Forms;
нужно добавить строку подключения пространства имен OleDb:
using System.Data.OleDb;

4.3. Создание объекта типа OleDbDataAdapter.
В конструкторе формы после вызова

InitializeComponent();
Добавляем строку создания объекта типа OleDbDataAdapter:

OleDbDataAdapter dA = new OleDbDataAdapter(CmdText, ConnString);
Объект типа OleDbDataAdapter организовывает пересылку наборов данных с вызываемым процессом. Адаптеры данных содержат набор из четырех внутренних объектов команд. Это команды чтения, вставки, изменения и удаления информации. Как видно из программного кода, конструктор объекта получает входящими параметрами строку запроса на языке SQL (переменная CmdText) и строку подключения к базе данных (переменная ConnString). Таким образом, после выполнения данного кода, объект адаптера уже связан с нашей базой данных.

4.4. Создание объекта набора данных DataSet.

После создания адаптера данных (OleDbDataAdapter) создаем объект типа DataSet (набор данных):
DataSet ds = new DataSet();
Набор данных представляет что-то вроде промежуточного буфера для данных, которые могут отображаться. Набор данных представляет удобный механизм чтения и обновления данных а также инкапсулирует множество таблиц и связей между ними.

4.5. Заполнение таблицы “Order” на основе SQL-запроса.

Следующая команда – это заполнение набора данных (переменная ds) значениями записей из базы данных на основе SQL-запроса, содержащегося в адаптере данных dA с помощью метода Fill():
dA.Fill(ds, "[Order]");

4.6. Визуализация данных в dataGridView1.

На данный момент данные из таблицы “Order” считаны в объекте ds (типа DataSet), представляющем собой набор данных. Для их отображения необходимо чтобы свойство DataSource компонента dataGridView1 ссылалось на первую таблицу (в нашем случае одна таблица) набора данных ds. Программный код этой операции имеет следующую реализацию:
dataGridView1.DataSource = ds.Tables[0].DefaultView;
После этого данные из таблицы “Order” отобразятся на форме (рисунок 55).

5. Весь програмний код.

Общий листинг класса главной формы приложения имеет следующий вид:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication1
{
 public partial class Form1 : Form
 {
  public string CmdText = "SELECT * FROM [Order]";
  public string ConnString =
"Provider=Microsoft.Jet.OLEDB.4.0;DataSource=C:\\Programs\\C_Sharp\\WindowsFormsA
pplication1\\mydb.mdb";
  public Form1()
  {
   InitializeComponent();
   OleDbDataAdapter dataAdapter = new OleDbDataAdapter(CmdText, ConnString);
   // создаем объект DataSet
   DataSet ds = new DataSet();
   // заполняем таблицу Order
   // данными из базы данных
   dataAdapter.Fill(ds, "[Order]");
   dataGridView1.DataSource = ds.Tables[0].DefaultView;
  }
 }
}
```

Результат выполнения приложения изображен на рисунке 55.

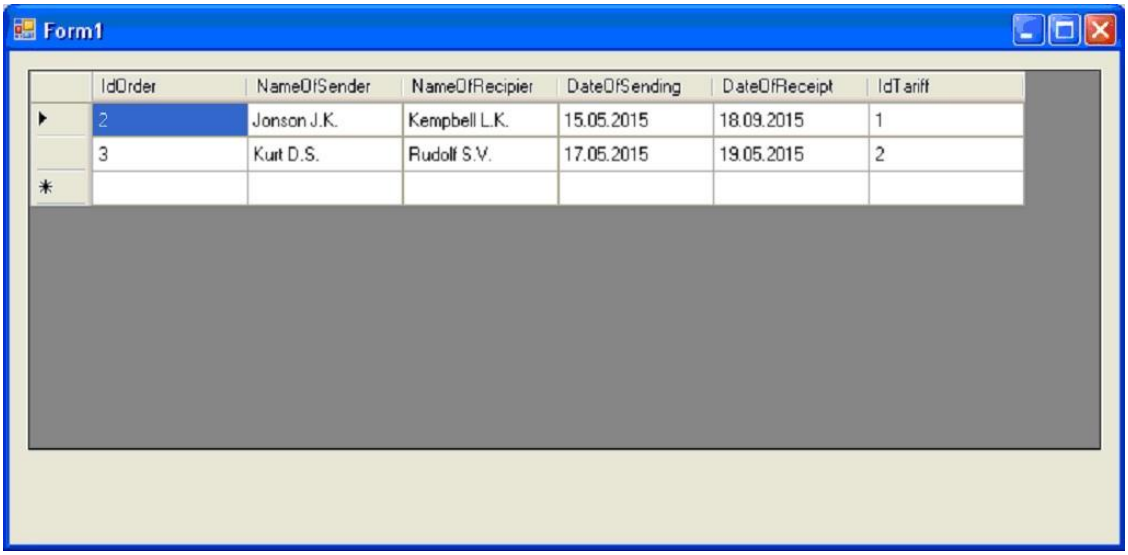


Рисунок 55 - Результат выполнения приложения

6. Схема взаимодействия.

Общая схема взаимодействия между объектами изображена на рисунке 56.

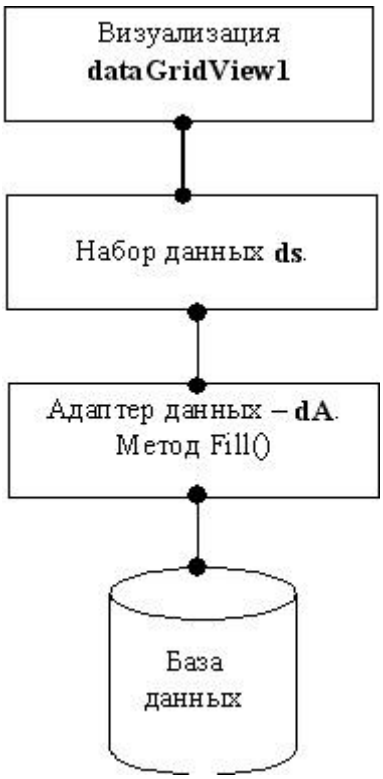


Рисунок 56 - Схема взаимодействия между объектами для доступа к базе данных

Таким образом, можно выводить на форму любую таблицу базы данных.

Условия выведения данных из базы данных задаются в строке SQL-запроса в переменной CmdText.
Например, если в CmdText задать следующую строку:

CmdText = "SELECT * FROM [Order] WHERE [NameOfSender] LIKE 'I%'";
то в результате из базы данных будут извлекаться записи, начинающиеся с символа '**I**'.

10.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

11. ЛАБОРАТОРНАЯ РАБОТА № 10. СОЗДАНИЕ СЛУЖБЫ WCF

11.1 Общие сведения

Цель: научиться создавать службы WCF в среде Visual Studio на языке C#.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной № 9.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 6ч.
Время самостоятельной подготовки: 2 ч.

11.2 Задание к лабораторной работе

Для начала необходимо создать новый проект WCF. Пусть наша Windows Communication Foundation служба будет возвращать количество оставшихся дней до нового года.

11.3 Методические указания и порядок выполнения работы (рисунок 57,58,59,60)

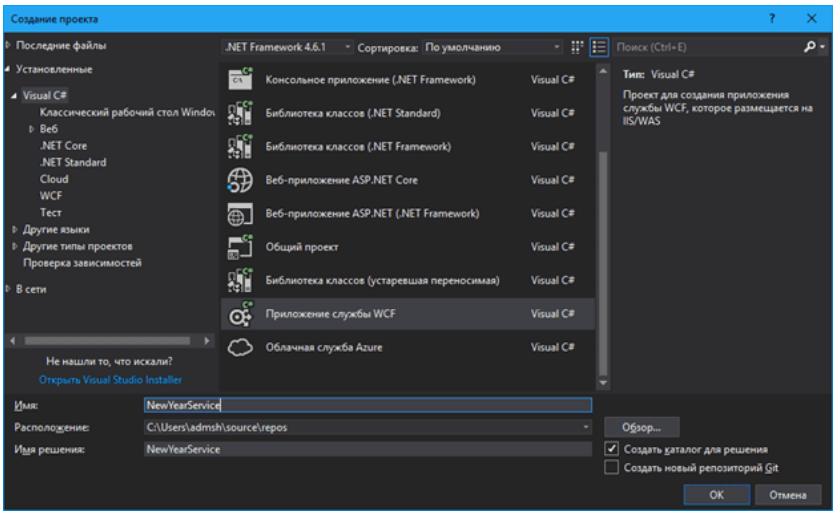


Рисунок 57 - Интерфейс

Visual studio создаст интерфейс и класс службы по умолчанию с именем IService1.cs и Service1.svc.

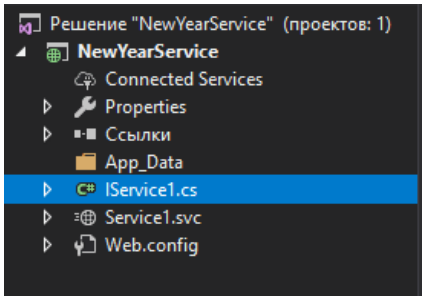


Рисунок 58 - Интерфейс

Нам необходимо переименовать их в соответствии с нашей предметной областью.

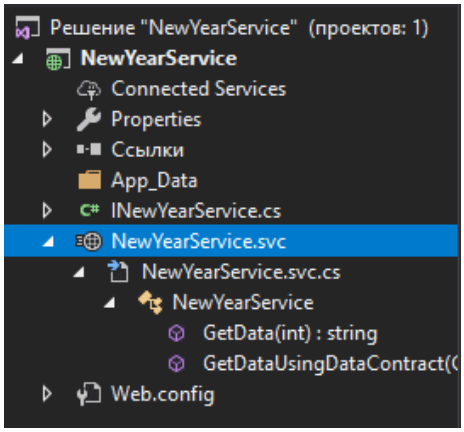


Рисунок 59 - Интерфейс

Давайте рассмотрим интерфейс INewYearService. Для начала нам необходимо в теле интерфейса объявить метод, который будет предоставлять служба для вызова. Для этого его необходимо пометить атрибутом [OperationContract].

```
using System;
using System.ServiceModel;

namespace NewYearService
{
    /// <summary>
    /// Интерфейс, определяющий методы, которые предоставляет служба.
    /// </summary>
    [ServiceContract]
    public interface INewYearService
    {
        /// <summary>
        /// Получить  количество времени до нового года от указанной даты.
        /// </summary>
        /// <param name="start">Дата от которой ведется отсчет до нового года.</param>
        /// <returns>Временные промежутки до нового года.</returns>
        [OperationContract]
        TimeToNewYear GetDaysToNewYear(DateTime start);
    }
}
```

Как вы видите данный метод возвращает экземпляр класса TimeToNewYear. Это вспомогательный класс, содержащий значения времени до нового года. Ниже приведена его структура. Для того, чтобы данный класс можно было использовать в качестве возвращаемого аргумента, его необходимо пометить атрибутом [DataContract], а свойства, доступные для чтения клиенту в возвращаемом значении помечаются атрибутом [DataMember].

```
using System;
using System.Runtime.Serialization;

namespace NewYearService
```

```
{
    /// <summary>
    /// Класс, содержащий временные промежутки до нового года.
    /// </summary>
    [DataContract]
    public class TimeToNewYear
    {
        /// <summary>
        /// Момент времени, от которого отсчитывается время.
        /// </summary>
        private DateTime _start;

        /// <summary>
        /// Дата нового года.
        /// </summary>
        private DateTime _newYear;

        /// <summary>
        /// Количество дней до нового года.
        /// </summary>
        [DataMember]
        public int Days { get; private set; }

        /// <summary>
        /// Количество часов до нового года.
        /// </summary>
        [DataMember]
        public int Hours { get; private set; }

        /// <summary>
        /// Количество минут до нового года.
        /// </summary>
        [DataMember]
        public int Minutes { get; private set; }

        /// <summary>
        /// Количество секунд до нового года.
        /// </summary>
        [DataMember]
        public int Seconds { get; private set; }

        /// <summary>
        /// Создать новый экземпляр отсчета времени до нового года.
        /// </summary>
        /// <param name="point">Момент времени, от которого ведется отсчет.</param>
        public TimeToNewYear(DateTime point)
        {
            _start = point;
            _newYear = DateTime.Parse($"{point.Year}-12-31 23:59:59.999");
            SetPeriods();
        }
```

```
        /// <summary>
        /// Метод устанавливающий временные промежутки до нового года.
        /// </summary>
        private void SetPeriods()
        {
            var interval = _newYear - _start;
            Days = (int)interval.TotalDays;
            Hours = (int)interval.TotalHours;
            Minutes = (int)interval.TotalMinutes;
            Seconds = (int)interval.TotalSeconds;
        }
    }
}
```

Теперь нам остается реализовать интерфейс Windows Communication Foundation службы в классе NewYearService.svc.cs следующим образом:

```
using System;

namespace NewYearService
{
    /// <summary>
    /// Класс службы реализующий интерфейс взаимодействия со службой.
    /// </summary>
    public class NewYearService : INewYearService
    {
        /// <summary>
        /// Получить  количество времени до нового года от указанной даты.
        /// </summary>
        /// <param name="start">Дата от которой ведется отсчет до нового года.</param>
        /// <returns>Временные промежутки до нового года.</returns>
        public TimeToNewYear GetDaysToNewYear(DateTime start)
        {
            var timeToNewYear = new TimeToNewYear(start);
            return timeToNewYear;
        }
    }
}
```

Давайте проверим работу нашей службы wcf. Для этого нажмем кнопку Начать отладку. Обратите внимание, что возможные два варианта поведения системы. Если мы начнем отладку находясь в NewYearService.svc, от откроется отладчик службы. Во всех остальных случаях откроется окно браузера. Давайте рассмотрим каждый из вариантов подробнее.

Браузер

После запуска отладки отобразится браузер с файловой структурой нашей службы wcf.

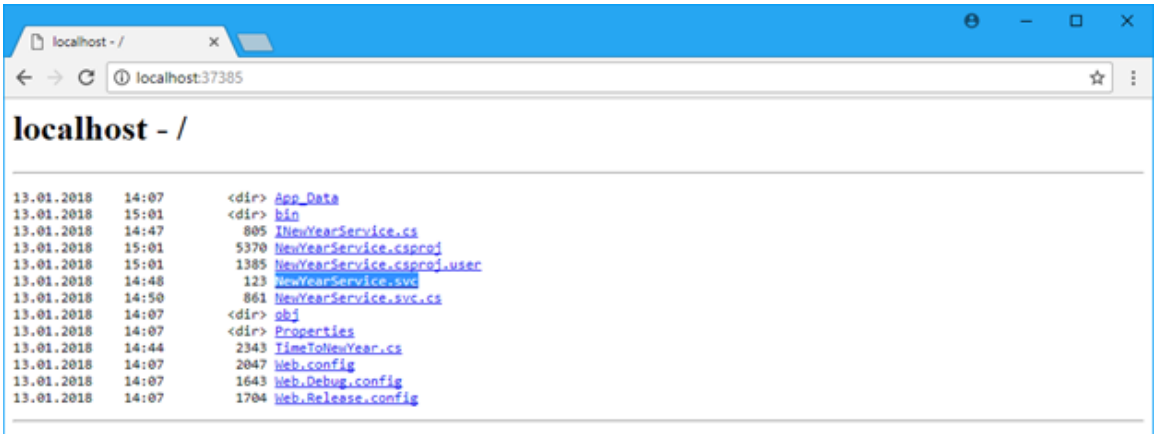


Рисунок 60 - Интерфейс

Нам необходимо нажать на ссылку с именем нашей службы NewYearService.svc. Если все работает корректно, то мы увидим следующее окно, иначе будет показано сообщение с ошибкой.

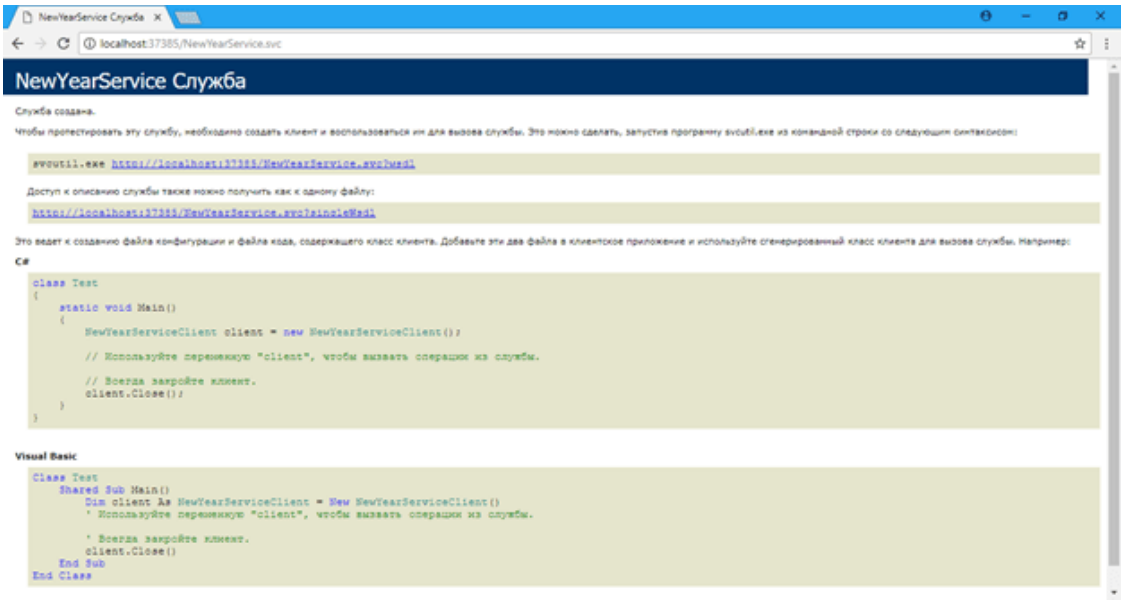


Рисунок 61 - Интерфейс

Тестовый клиент WCF

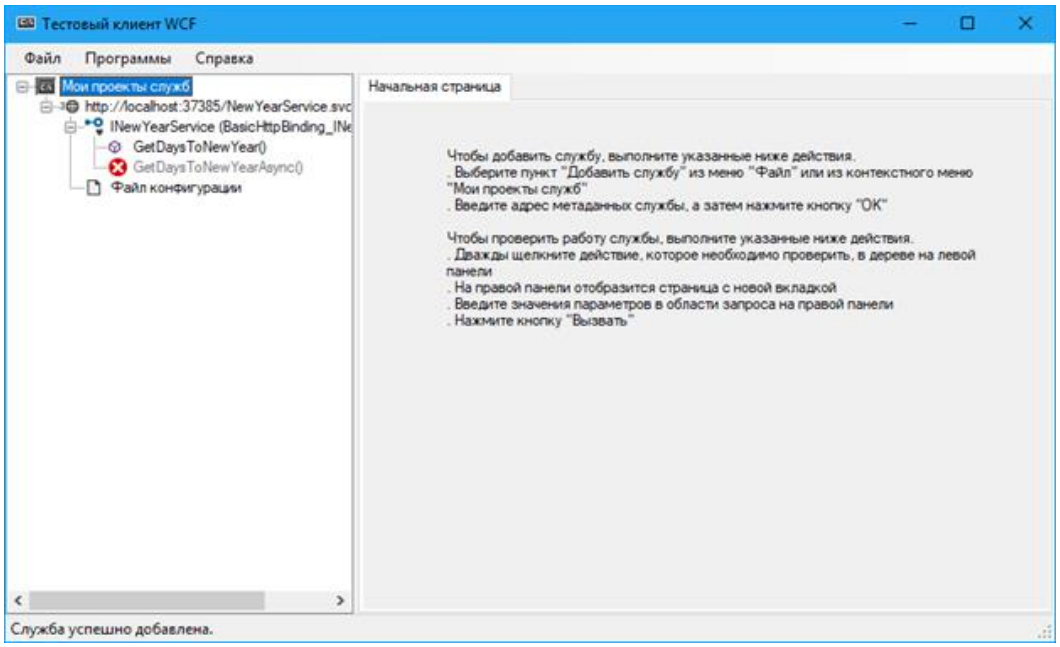


Рисунок 62 - Интерфейс

В левой верхней части отладчика можно увидеть структура нашей службы wcf. Для проверки нашего метода выполним двойной щелчок левой кнопкой мыши по его имени. В правой части отладчика откроется форма запроса. Мы можем указать значение, которое будет передано в метод.

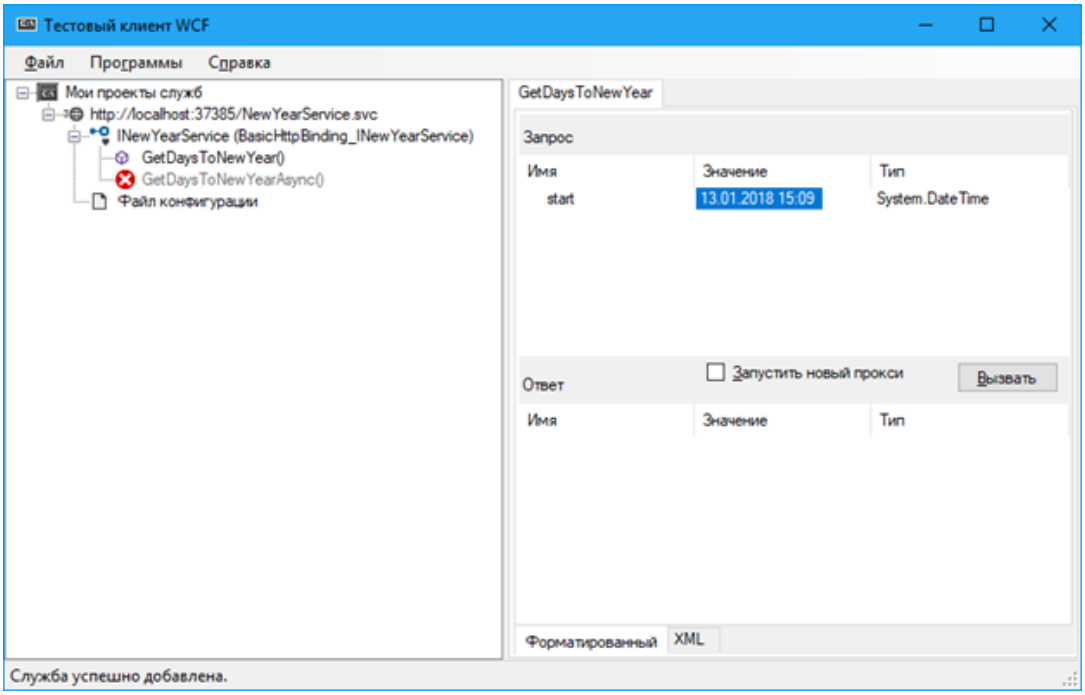


Рисунок 63 - Интерфейс

После установки передаваемых значений необходимо нажать кнопку Вызвать. Появится предупредительное сообщение. Можно смело ставить галочку Не выводить это сообщение в дальнейшем и нажимать кнопку ОК.

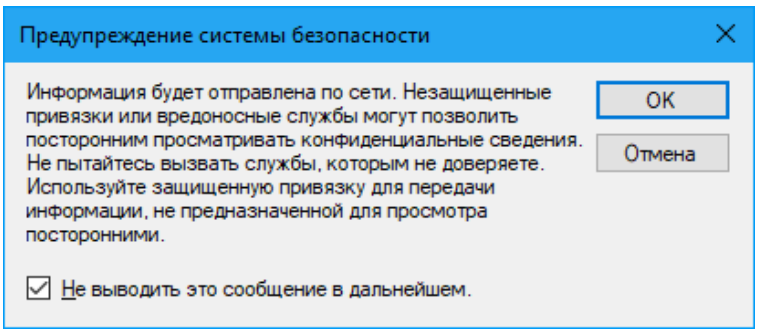


Рисунок 64 - Интерфейс

После этого в нижней правой части отладчика будет отображены значения возвращаемые нашей службой wcf (отсунок 65,66,67,68,69).

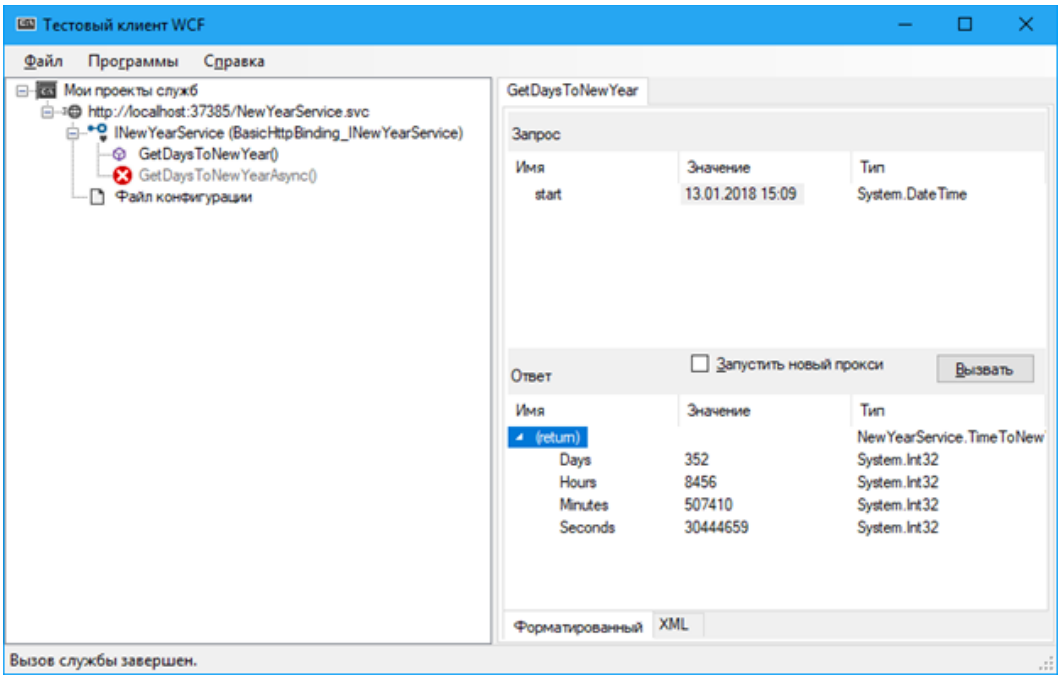


Рисунок 65 - Интерфейс

Консольный клиент для WCF

Теперь нам необходимо создать клиент, который будет обращаться к нашей службе wcf. Для этого для начала создадим новое консольное приложение.

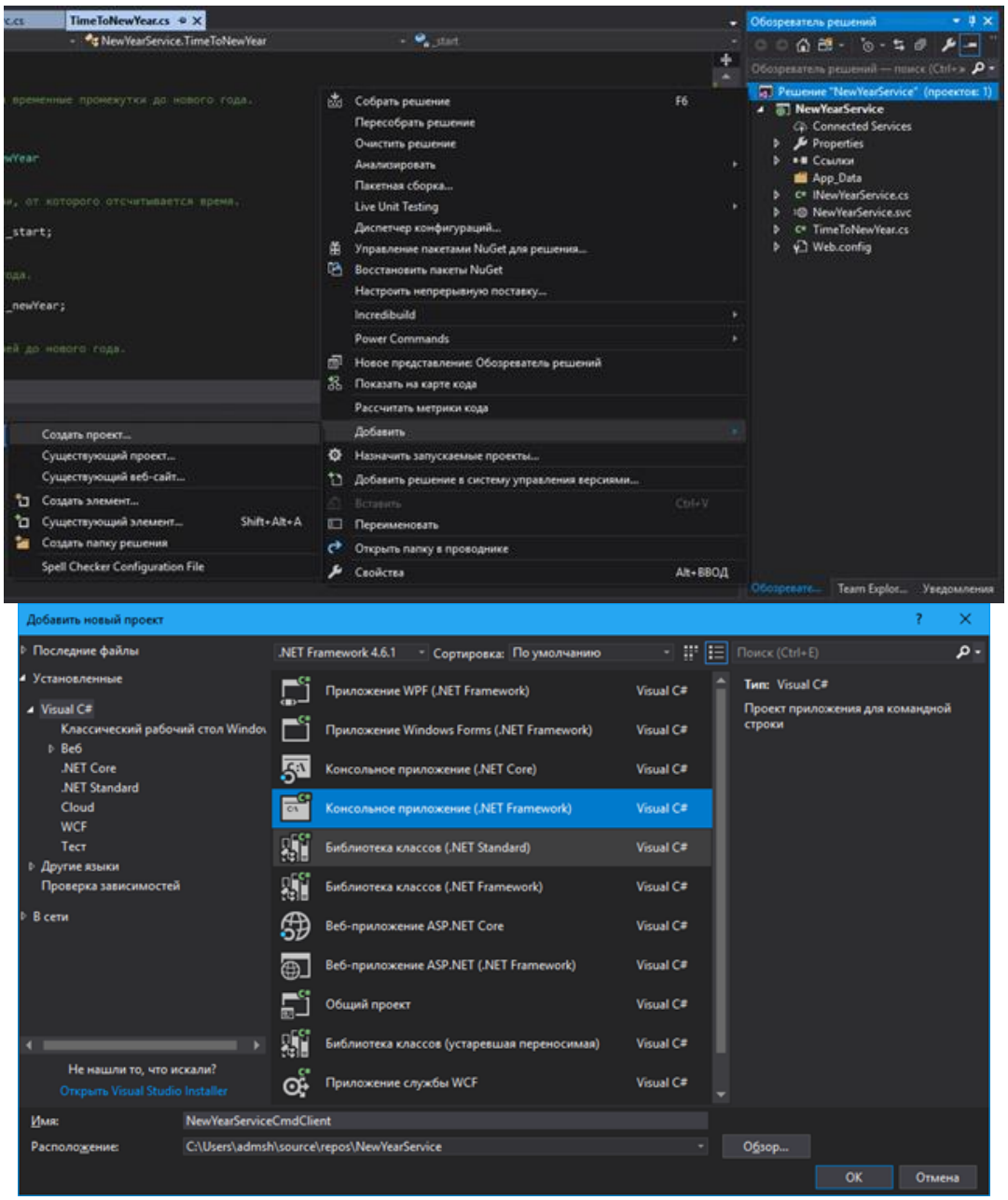


Рисунок 66 - Интерфейс

В созданном консольном приложении нам необходимо добавить ссылку на службу wcf.

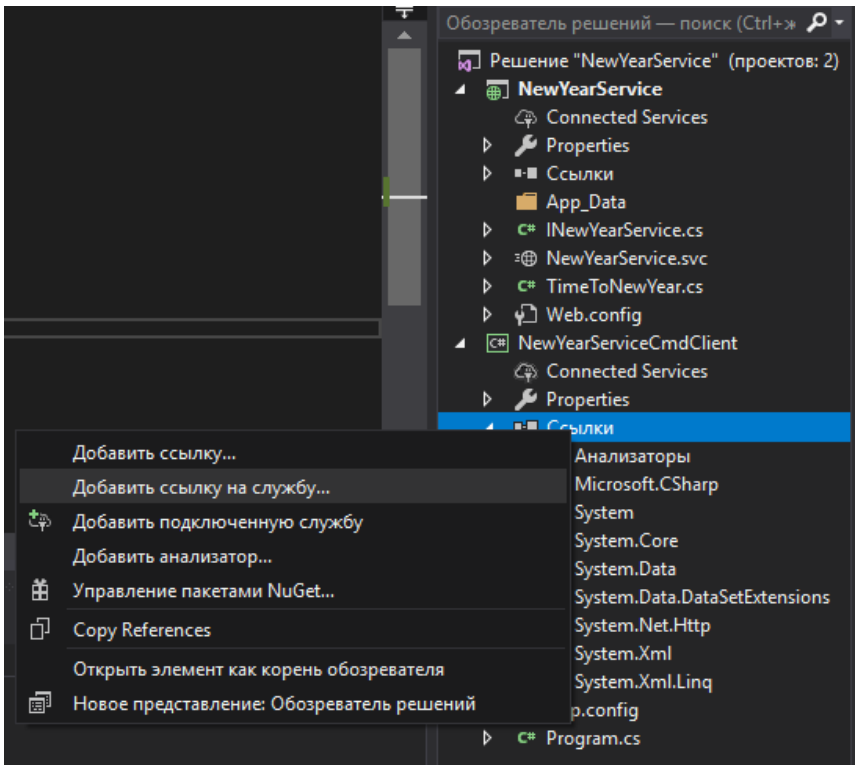


Рисунок 67 - Интерфейс

В открывшемся окне службы необходимо указать имя службы wcf и ввести ее адрес.

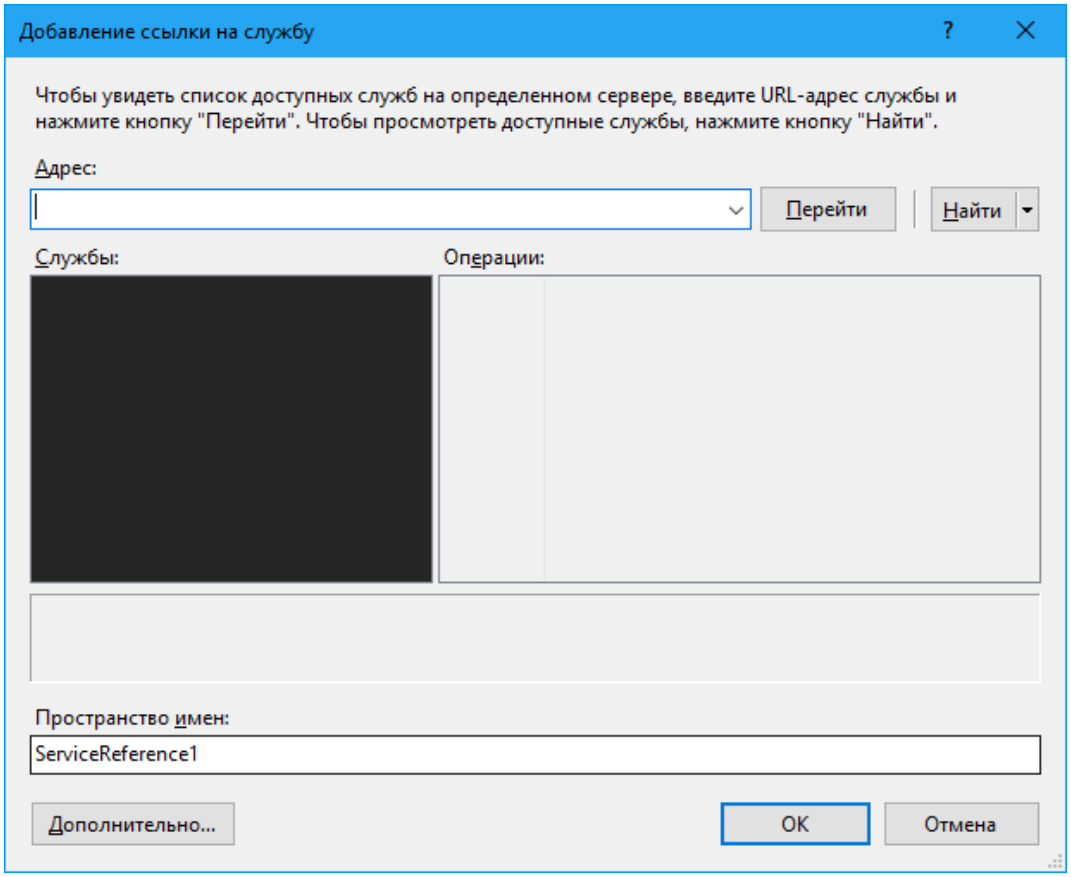


Рисунок 68 - Интерфейс

Для простоты можно нажать кнопку Найти, тогда адрес службы wcf будет определен автоматически.

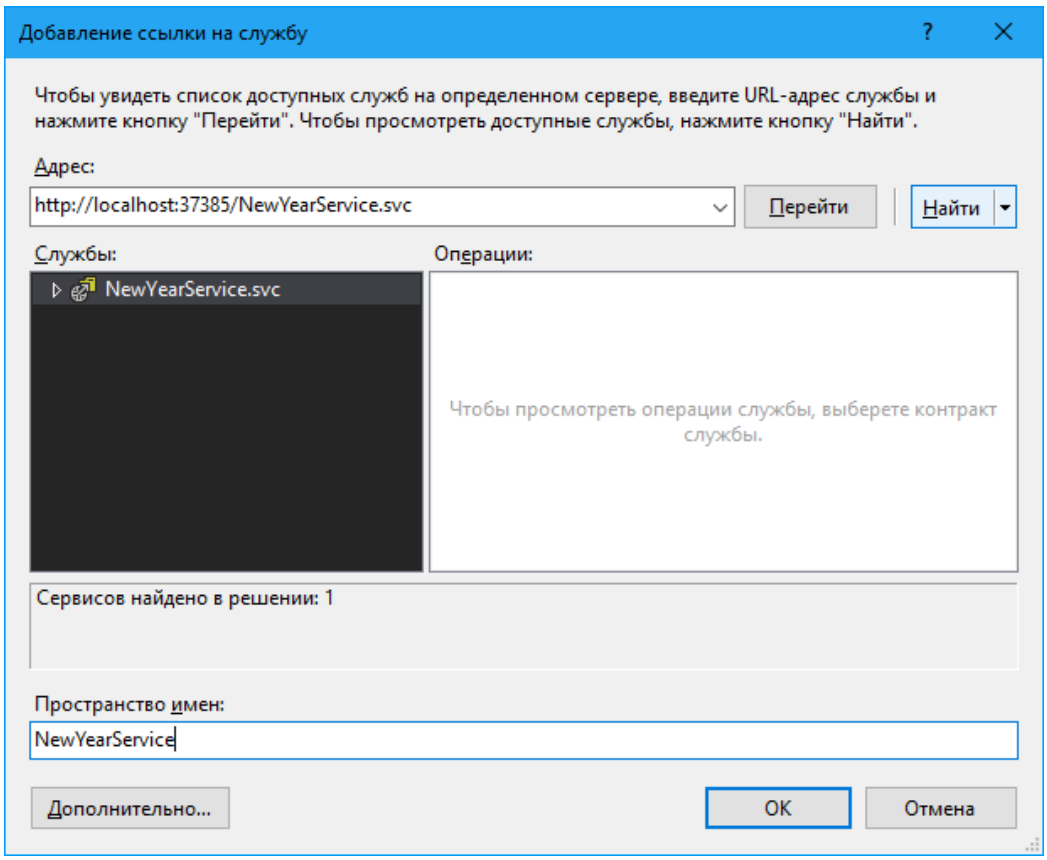


Рисунок 69 - Интерфейс

После этого необходимо развернуть дерево Windows Communication Foundation службы, чтобы удостоверится что выбран правильность выбора. В правой части должен быть отображен вызываемый метод.

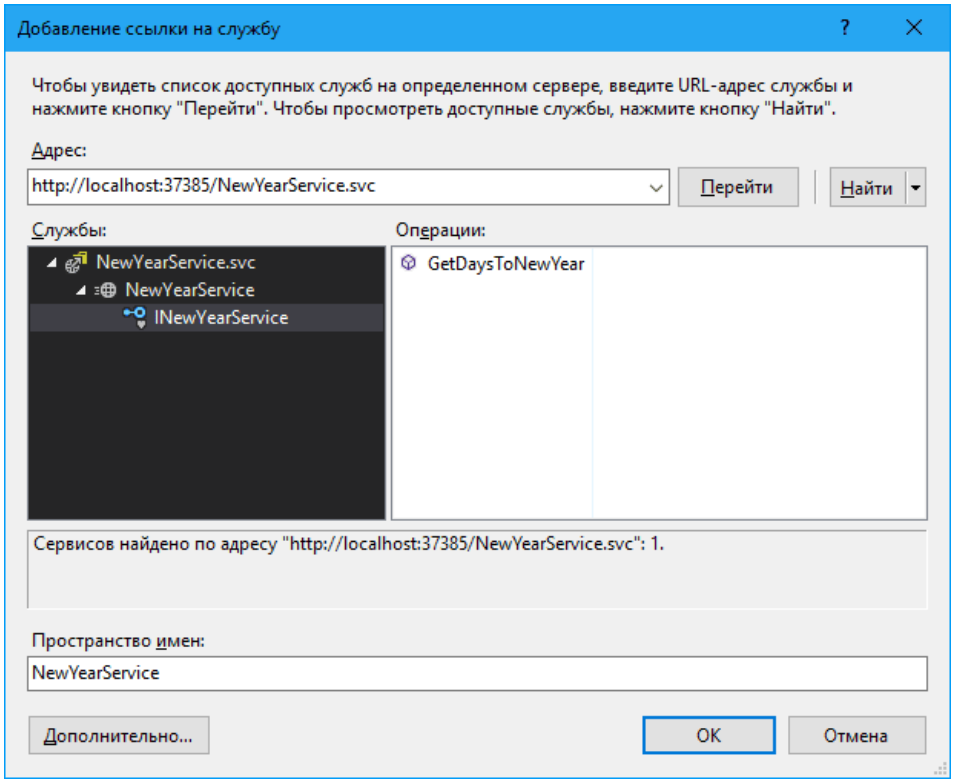
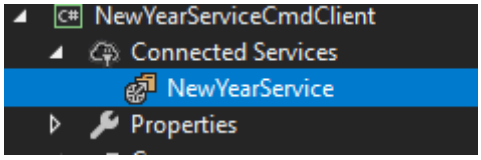


Рисунок 70 - Интерфейс

Если настройка прошла корректно, то в обозревателе решения в консольном приложении отобразится ссылка на нашу службу wcf.



Теперь нам остается только обратиться к нашей службе, чтобы взывать метод и вывести результат на экран.

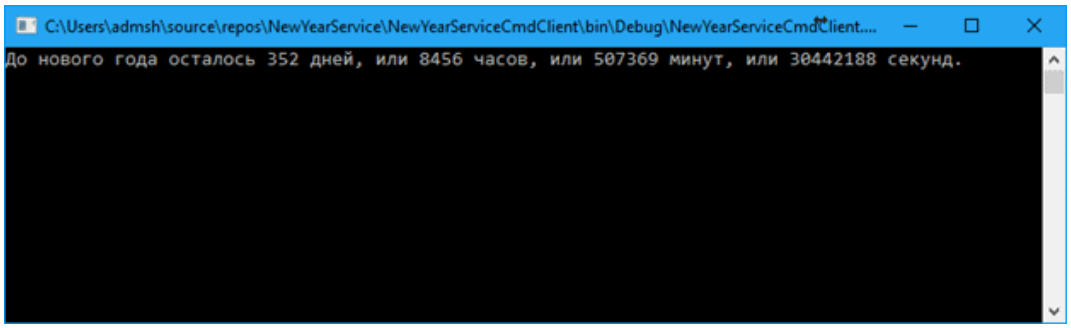
```
using NewYearServiceCmdClient.NewYearService;
using System;

namespace NewYearServiceCmdClient
{
   class Program
   {
      static void Main(string[] args)
      {
         // Создадим клиент для обращения к службе.
         var client = new NewYearServiceClient();

         // Вызовем метод службы и сохраним результат.
         var result = client.GetDaysToNewYear(DateTime.Now);

         // Выводим результат на консоль.
         Console.WriteLine($"До нового года осталось {result.Days} дней, или {result.Hours} ча
сов, или {result.Minutes} минут, или {result.Seconds} секунд.");
         Console.ReadKey();
      }
   }
}
```

Перед началом отладки не забудьте установить консольное приложение автозагружаемым проектом. Получаем следующий результат.



Web клиент для WCF (рисунок 71,72,73,74).

Теперь рассмотрим, как нам обратиться к службе wcf из веб-приложения. Процесс подключения службы не отличается от подключения в консольном приложении. Давайте рассмотрим как можно настроить авторизацию с помощью Windows. Это потребует дополнительной настройки приложения. Для начала создадим проект нового MVC приложения.

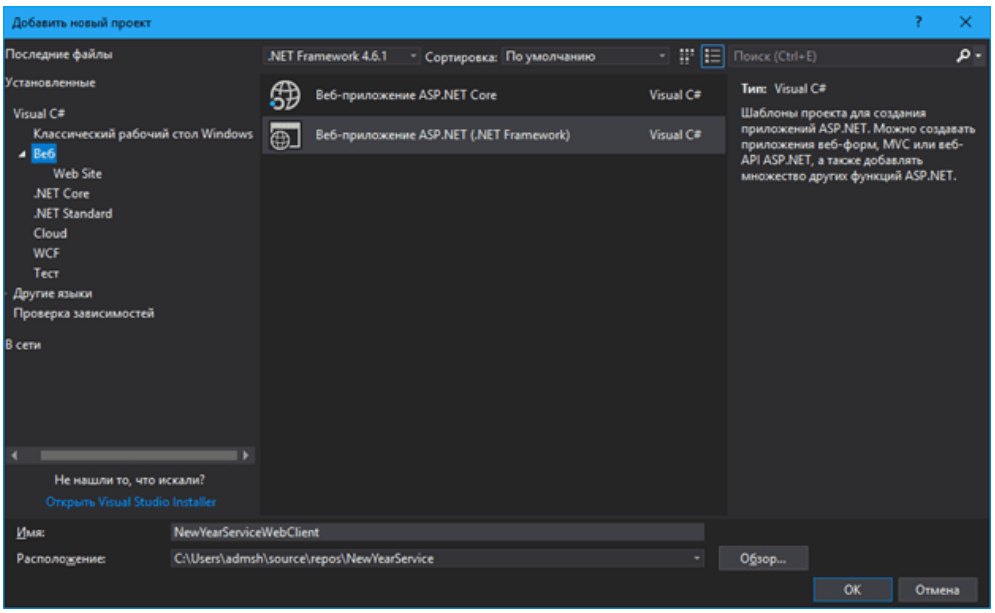


Рисунок 71 - Интерфейс

Нажимаем кнопку ОК, и попадаем в меню настройки создания веб-приложения. Выберем MVC шаблон и изменим способ авторизации. Для этого нажмем на кнопку Изменить способ проверки подлинности.

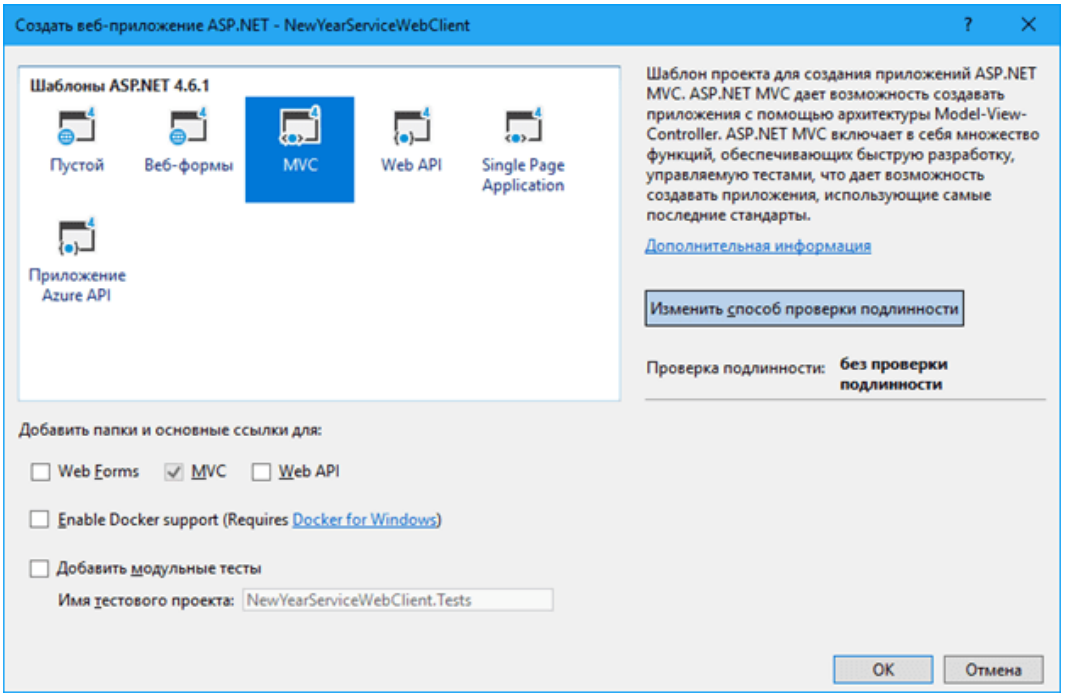


Рисунок 72 - Интерфейс

Выбираем авторизацию с помощью Windows и нажимаем ОК в обоих окнах.

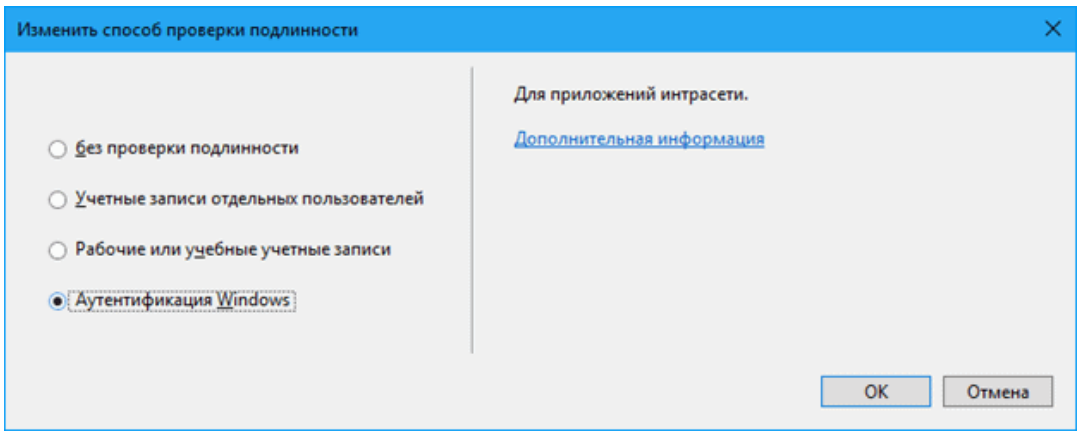


Рисунок 73 - Интерфейс

Теперь необходимо будет подождать кое-то время, чтобы создался проект и все библиотеки были загружены и подключены.

Для того, чтобы IISExpres перестала ругаться на нас за попытку создания windows-аутентификации нужно сделать ряд действий в наше службе.

Нужно дополнить наш web.config
Добавляем в любую точку раздела <configuration>, если отсутствует раздел webServer или дополняем уже существующий.

```
<system.webServer>
   <security>
     <authentication>
       <windowsAuthentication enabled="true" />
     </authentication>
   </security>
</system.webServer>
```

Находим раздел <system.web> и в нем вставляем следующее.

```
<authentication mode="Windows"></authentication>
<authorization>
     <allow users="domen\"/> //каких пользователей пускать
    <!--<deny users="*"/>--> //запрет пользователей
</authorization>
```

Далее нужно указать разделы биндинг и сервис

```
<bindings>
     <basicHttpBinding>
       <binding name="BasicHttpEndpoinBinding">
         <security mode="TransportCredentialOnly">
           <transport clientCredentialType="Windows" />
         </security>
       </binding>
     </basicHttpBinding>
</bindings>
<services>
     <service name="Имя сервиса">
       <endpoint address=""
```

```
            binding="basicHttpBinding"
            bindingConfiguration="BasicHttpEndpoinBinding"
            name="BasicHttpEndpoint"
            contract="Интерфейс в сервисе (выпадет само)">
        <identity>
          <dns value="localhost"/>
        </identity>
      </endpoint>
      <endpoint address="mex"
            binding="mexHttpBinding"
            contract="IMetadataExchange"/>
    </service>
</services>
```
Раздел <serviceBehavior><behavior> дополняем такой вот строчкой. В ней мы говорим, что именно такая схема аутентификации будет использоваться у нас.

```
<serviceAuthenticationManager authenticationSchemes="Negotiate"/>
```
На этом настройка web.config заканчивается. В итоге у нас должен получиться файл примерно следующего содержания.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.6.1" />
    <httpRuntime targetFramework="4.6.1"/>
    <authentication mode="Windows"></authentication>
    <authorization>
      <allow users="*"/> <!-
- Каких пользователей пускать (при необходимости можно указать домен в формате domen\*
) -->
      <!--<deny users="*"/>--> <!-- Запрет пользователей -->
    </authorization>
  </system.web>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpEndpoinBinding">
          <security mode="TransportCredentialOnly">
            <transport clientCredentialType="Windows" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <services>
      <service name="NewYearService.NewYearService">        <!-- Имя сервиса -->
        <endpoint address=""
            binding="basicHttpBinding"
            bindingConfiguration="BasicHttpEndpoinBinding"
```

```
            name="BasicHttpEndpoint"
            contract="NewYearService.INewYearService"> <!-
- Интерфейс в сервисе (выпадет само) -->
        <identity>
          <dns value="localhost"/>
        </identity>
      </endpoint>
      <endpoint address="mex"
            binding="mexHttpBinding"
            contract="IMetadataExchange"/>
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior>
        <!--
Чтобы избежать раскрытия метаданных, до развертывания задайте следующим параметрам з
начение "false". -->
        <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
        <!-
- Чтобы при сбое получать подробные сведения об исключении для целей отладки, установит
е для нижеприведенного параметра значение true.  Перед развертыванием установите значен
ие false, чтобы избежать раскрытия информации об исключении -->
        <serviceDebug includeExceptionDetailInFaults="false"/>
        <!-- Говорим, что именно такая схема аутентификации будет использоваться у нас. -->
        <serviceAuthenticationManager authenticationSchemes="Negotiate"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <protocolMapping>
      <add binding="basicHttpsBinding" scheme="https" />
  </protocolMapping>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled
="true" />
 </system.serviceModel>
 <system.webServer>
  <modules runAllManagedModulesForAllRequests="true"/>
  <directoryBrowse enabled="true"/>
  <security>
    <authentication>
      <windowsAuthentication enabled="true" />
    </authentication>
  </security>
 </system.webServer>
</configuration>
```

Настройка applicationhost.config

Далее идем в папку vs нашего проекта (она скрыта по умолчанию). В ней ищем папку config, а уже в ней находим файл applicationhost.config, его то нам нужно будет поправить.

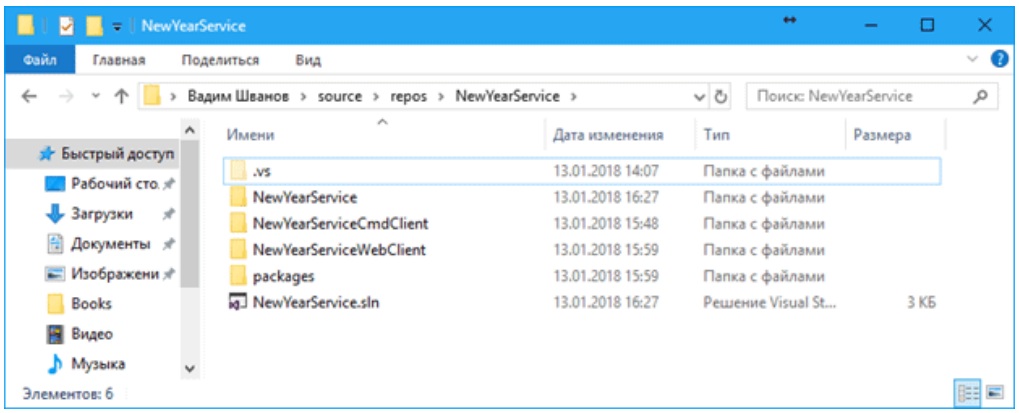


Рисунок 74 - Интерфейс

Находим вот такой раздел. Все Deny меняем на Allow, разрешая изменение установленного по умолчанию режима аутентификации.

```
<sectionGroup name="authentication">
   <section name="anonymousAuthentication" overrideModeDefault="Deny" />
   <section name="basicAuthentication" overrideModeDefault="Deny" />
   <section name="clientCertificateMappingAuthentication" overrideModeDefault="Deny" />
   <section name="digestAuthentication" overrideModeDefault="Deny" />
   <section name="iisClientCertificateMappingAuthentication" overrideModeDefault="Deny" />
   <section name="windowsAuthentication" overrideModeDefault="Deny" />
</sectionGroup>
```

Далее находим данную настройку. В ней false меняем на true, разрешая механизму работать.

```
<windowsAuthentication enabled="false">
   <providers>
      <add value="Negotiate" />
      <add value="NTLM" />
   </providers>
</windowsAuthentication>
```

И под конец находим вот эту настройку. Тут мы true меняем на false. Говоря нашему IISExpres, чтобы он не блокировал службу windows-аутентификации.

```
<add name="WindowsAuthenticationModule" lockItem="true" />
```

После этого сохраняем все конфигурационные файлы и пробуем сделать ссылку на службу wcf аналогично, как при добавлении службы в консольное приложение (рисунок 75).

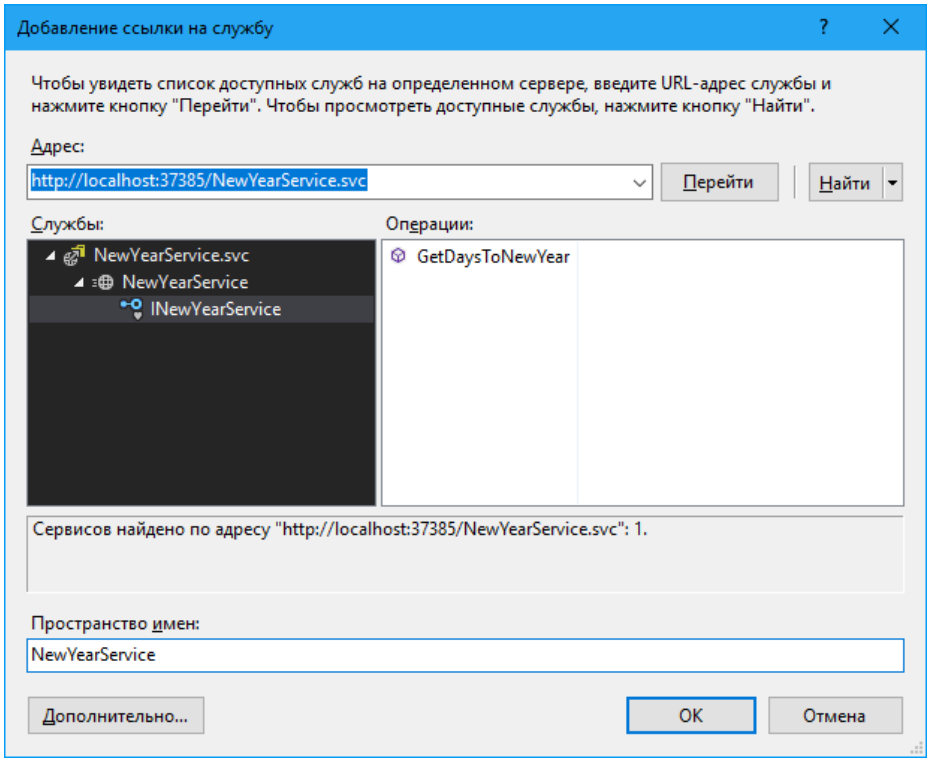


Рисунок 75 - Интерфейс

Изменим контроллер главной страницы web-приложения, чтобы взывать нашу службу wcf.

```
using NewYearServiceWebClient.NewYearService;
using System;
using System.Web.Mvc;

namespace NewYearServiceWebClient.Controllers
{
    /// <summary>
    /// Контроллер домашних страниц.
    /// </summary>
    public class HomeController : Controller
    {
        /// <summary>
        /// Главная страница.
        /// </summary>
        /// <returns>Представление главная страница.</returns>
        public ActionResult Index()
        {
            // Создадим клиент для обращения к службе.
            var client = new NewYearServiceClient();

            // Вызовем метод службы и сохраним результат.
            var result = client.GetDaysToNewYear(DateTime.Now);

            // Помещаем результат в представление.
            ViewBag.DaysToNewYear = result;
```

```
            return View();
        }

        /// <summary>
        /// О программе.
        /// </summary>
        /// <returns>Представление о программе.</returns>
        public ActionResult About()
        {
            ViewBag.Message = "Тестовое приложение для работы с WCF в MVC.";

            return View();
        }

        /// <summary>
        /// Контактные данные.
        /// </summary>
        /// <returns>Представление контакты.</returns>
        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
}
```

Теперь нам осталось только изменить представление, чтобы вывести результат работы Windows Communication Foundation службы на экран пользователя.

```
@{
    ViewBag.Title = "Главная";
}

<div class="jumbotron">
    <h1>Использование WCF в MVC</h1>
    <p class="lead">Чтобы подробнее узнать о том, как создавать WCF службы и использовать их в MVC приложениях нажмите на кнопку.</p>
    <p><a href="https://shwanoff.ru/wcf/" class="btn btn-primary btn-lg">Узнать больше &raquo;</a></p>
</div>

<div class="row">
    <div class="col-md-3">
        <h2>Количество Дней до нового года</h2>
        <p>@ViewBag.DaysToNewYear.Days</p>
    </div>
    <div class="col-md-3">
        <h2>Количество Часов до нового года</h2>
        <p>@ViewBag.DaysToNewYear.Hours</p>
    </div>
    <div class="col-md-3">
        <h2>Количество Минут до нового года</h2>
```

```
        <p>@ViewBag.DaysToNewYear.Minutes</p>
    </div>
    <div class="col-md-3">
        <h2>Количество Секунд до нового года</h2>
        <p>@ViewBag.DaysToNewYear.Seconds</p>
    </div>
</div>
```

Получаем следующий результат работы веб-приложения.

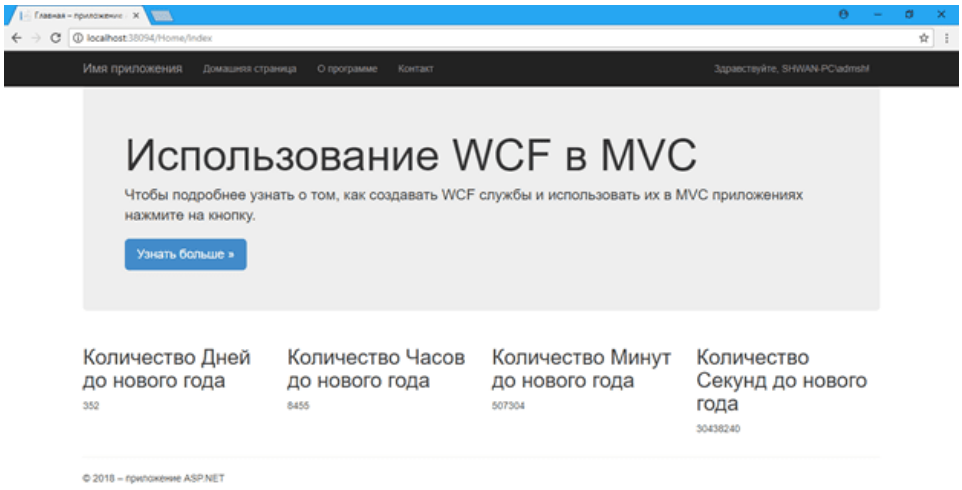


Рисунок 77 - Интерфейс

11.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

12. ЛАБОРАТОРНАЯ РАБОТА № 11. СОЗДАНИЕ ПРОСТОГО СЕРВИСА КАЛЬКУЛЯТОР WCF

12.1 Общие сведения

Цель: научиться создавать службы WCF в среде Visual Studio на языке C# и размещать их.

Материалы, оборудование, программное обеспечение: персональный компьютер, среда Visual Studio.

Условия допуска к выполнению: успешная защита лабораторной работы № 10.

Критерии положительной оценки: разработанная программа приложения и ответы на вопросы по коду и программированию событий.

Планируемое время выполнения:
Аудиторное время выполнения (под руководством преподавателя): 4ч.
Время самостоятельной подготовки: 2 ч.

12.2 Задание к лабораторной работе

В данном уроке мы создадим простой WCF-сервис «калькулятор», реализующий четыре базовые арифметические операции, научимся размещать сервис на IIS и публиковать на сервере. Кроме того, расскажем о том, как подключиться к сервису из другого приложения на C# или с помощью бесплатной программы SoapUI, а также добавим функционал для вызова сервиса через веб (например, из браузера или Ajax).

12.3 Методические указания и порядок выполнения работы

Создание проекта сервиса

В работе используется Visual Studio 2013 Professional, версия фреймворка 4.5.1. Для заметки: за некоторыми небольшими исключениями, разницы между версиями 4.0. и 4.5.1 нет, но использование последней предпочтительней, т.к. добавляет полезную функцию генерации WSDL одним файлом, но об этом ниже.

Итак, запускаем студию и создаем проект типа **WCF Service Application** в соответствующем разделе, называем его **SampleService**.

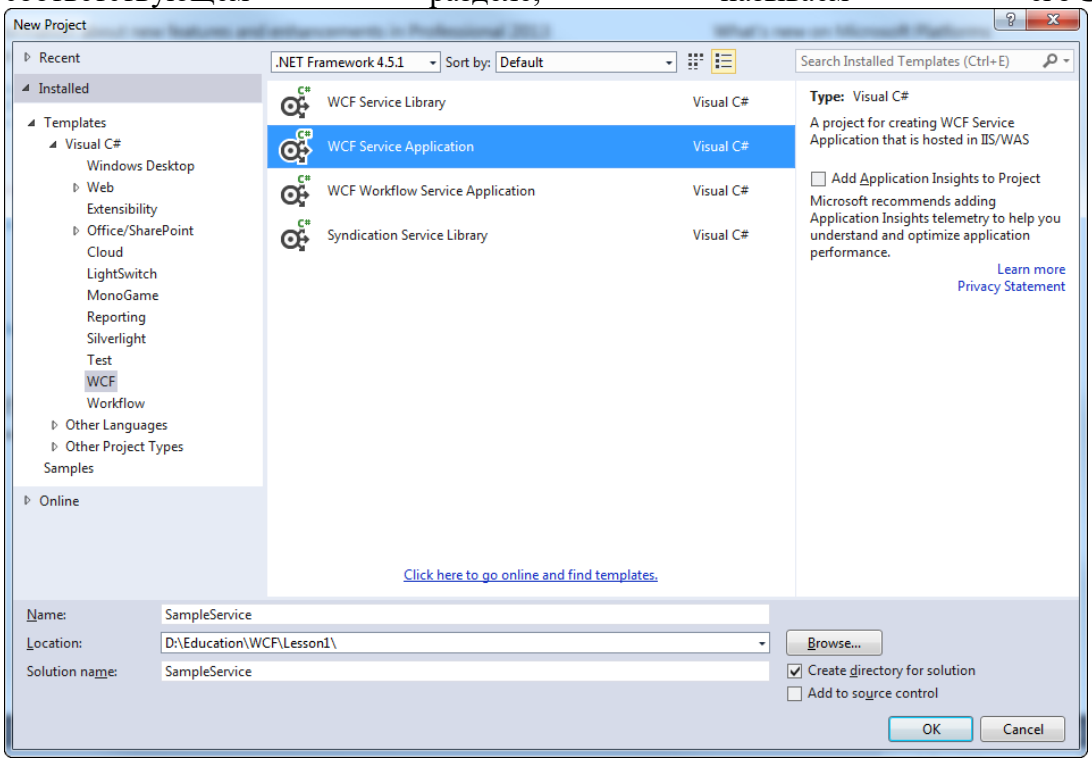
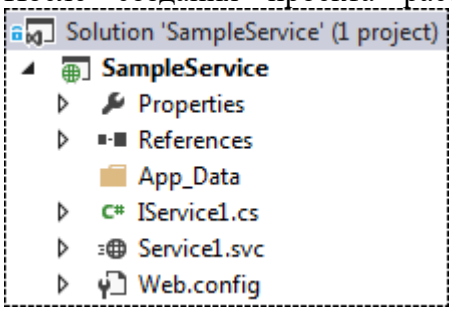


Рисунок 78 - Интерфейс

После создания проекта рассмотрим структуру решения в обозревателе решений:



1. *IService1.cs* содержит описание интерфейса сервиса, т.е. набор методов, которые наш сервис предоставляет.
2. *Service1.svc* состоит из двух частей — реализация сервиса (Service1.svc.cs) и разметка (Markup), доступная в меню по клику правой кнопкой мыши.
3. *Web.config* — конфигурация сервиса.
4. Папка *App_Data* нам пока не нужна — удаляем.

Для начала переименуем сервис и его интерфейс, придав им имена Calculator и ICalculator соответственно, а также изменим строчку в разметке сервиса (файл Calculator.svc):

```
<%@ ServiceHost Language="C#" Debug="true" Service="SampleService.Calculator" CodeBehind="Calculator.svc.cs" %>
```

Теперь опишем сам интерфейс нашего калькулятора. Объявим стандартные арифметические операции, такие как сложение (Addition), вычитание (Subtraction), умножение (Multiplication)

и деление (Division). Также добавим дополнительный метод TestConnection, возвращающий строку. С помощью этого метода клиенты смогут проверить, что сервис функционирует, так как стандартного «пинга» WCF не предоставляет. Все методы интерфейса должны быть помечены атрибутом [OperationContract], иначе они не будут видны клиентам.

В итоге, интерфейс примет следующий вид:

ICalculator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace SampleService
{
    [ServiceContract]
    public interface ICalculator
    {
        #region Common Methods

        /// <summary>
        /// проверка соединения
        /// </summary>
        /// <returns> OK </returns>
        [OperationContract]
        string TestConnection();

        #endregion

        #region Arithmetic

        /// <summary>
        /// сложение
        /// </summary>
        /// <param name="a"> слагаемое 1 </param>
        /// <param name="b"> слагаемое 2 </param>
        /// <returns> сумма </returns>
        [OperationContract]
        double Addition(double a, double b);

        /// <summary>
        /// вычитание
        /// </summary>
```

```
        /// <param name="a"> уменьшаемое </param>
        /// <param name="b"> вычитаемое </param>
        /// <returns> разность </returns>
        [OperationContract]
        double Subtraction(double a, double b);

        /// <summary>
        /// умножение
        /// </summary>
        /// <param name="a"> множитель 1 </param>
        /// <param name="b"> множитель 2 </param>
        /// <returns> произведение </returns>
        [OperationContract]
        double Multiplication(double a, double b);

        /// <summary>
        /// деление
        /// </summary>
        /// <param name="a"> делимое </param>
        /// <param name="b"> делитель </param>
        /// <returns> частное </returns>
        [OperationContract]
        double Division(double a, double b);

        #endregion
    }
}
```

Теперь реализуем интерфейс в классе сервиса:

Calculator.svc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

namespace SampleService
{
    public class Calculator : ICalculator
    {
        #region Common Methods

        /// <summary>
```

```
        /// проверка соединения
        /// </summary>
        /// <returns> OK </returns>
        public string TestConnection()
        {
            return "OK";
        }

        #endregion

        #region Arithmetic

        /// <summary>
        /// сложение
        /// </summary>
        /// <param name="a"> слагаемое 1 </param>
        /// <param name="b"> слагаемое 2 </param>
        /// <returns> сумма </returns>
        public double Addition(double a, double b)
        {
            return a + b;
        }

        /// <summary>
        /// вычитание
        /// </summary>
        /// <param name="a"> уменьшаемое </param>
        /// <param name="b"> вычитаемое </param>
        /// <returns> разность </returns>
        public double Subtraction(double a, double b)
        {
            return a - b;
        }

        /// <summary>
        /// умножение
        /// </summary>
        /// <param name="a"> множитель 1 </param>
        /// <param name="b"> множитель 2 </param>
        /// <returns> произведение </returns>
        public double Multiplication(double a, double b)
        {
            return a * b;
        }
```

```
        /// <summary>
        /// деление
        /// </summary>
        /// <param name="a"> делимое </param>
        /// <param name="b"> делитель </param>
        /// <returns> частное </returns>
        public double Division(double a, double b)
        {
            return a / b;
        }

        #endregion
    }
}
```

Логика сервиса описана, теперь необходимо его разместить на хостинге.

2. Размещение сервиса

Сам по себе сервис представляет библиотеку (в нашем случае файл SampleService.dll), и для ее запуска в виде сервиса необходимо воспользоваться одним из предоставляемых WCF методов:

- 1. Хостинг на IIS.
- 2. Запуск в виде службы Windows.
- 3. Self hosting (сервис выполнен в виде консольного приложения, запускающего сервис).

Принципиального различия в этих методах нет, поэтому выберем простейший вариант — размещение на IIS. Для этого откроем конфигурацию проекта и выберем вкладку «Web». Параметры по умолчанию вы можете наблюдать у себя, а вот так выглядят настройки для

нашего сервиса:

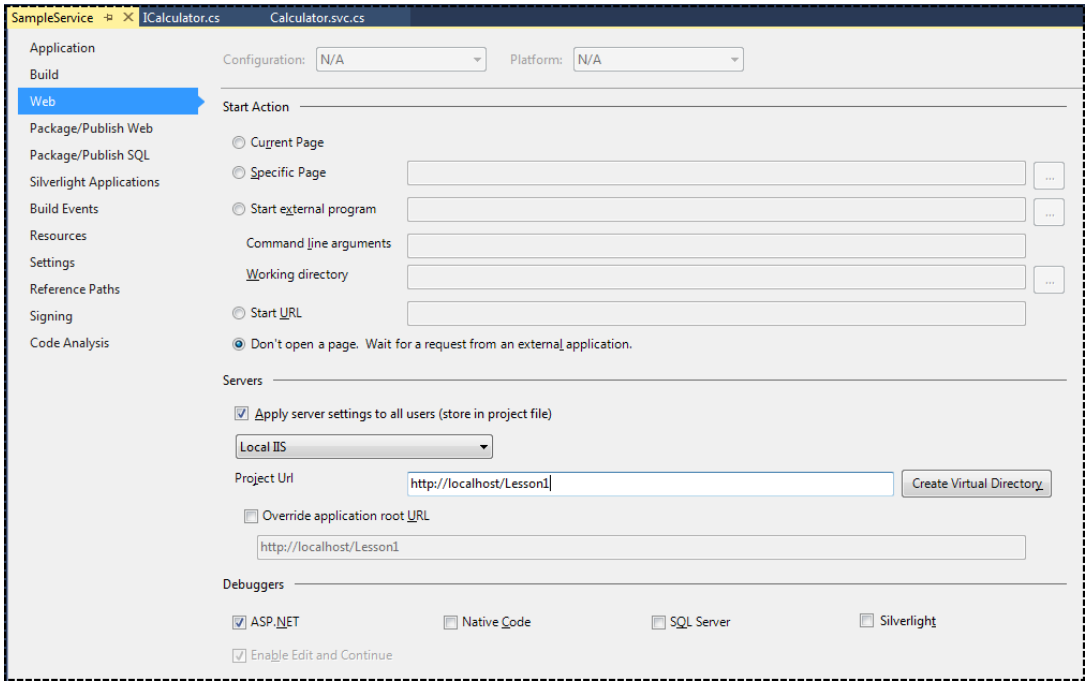


Рисунок 79 – Настройки свойств

Раздел **Start Action** определяет, что произойдет при запуске проекта в студии:

- Current Page (по умолчанию) – запуск последней открытой страницы в браузере. У нашего сервиса пока нет настроенных url-адресов методов, поэтому этот вариант не подходит.
- Specific Page – запуск конкретной страницы.
- Start External program – запуск указанного исполняемого файла, который инициирует сервис, например, вызовет метод, нуждающийся в отладке.
- Start URL – открытие указанного в поле URL адреса в браузере.
- Don't open a page – ничего не открывать, но ждать запроса от стороннего приложения.

Во всех случаях сервис «запускается» в режиме отладки и ведет себя как обычное приложение в студии, т.е. реагирует на контрольные точки. Последний вариант удобен тем, что ничего не открывается в браузере, т.к. отладка с помощью браузера — не лучший вариант для WCF сервиса, но об этом ниже.

Раздел **Servers** определяет, где будет размещен сервис. По умолчанию выбран IIS Express (строенный в студию вариант IIS), но мы будем продвинутыми и разместим сервис на нормальном IIS. Разумеется, для этого необходимо установить IIS в панели управления и включить необходимые компоненты (ASP .NET, Basic-авторизация и другие).

Под спойлером указан один из вариантов набора компонентов для работы сервисов (рисунок 81).

Компоненты IIS

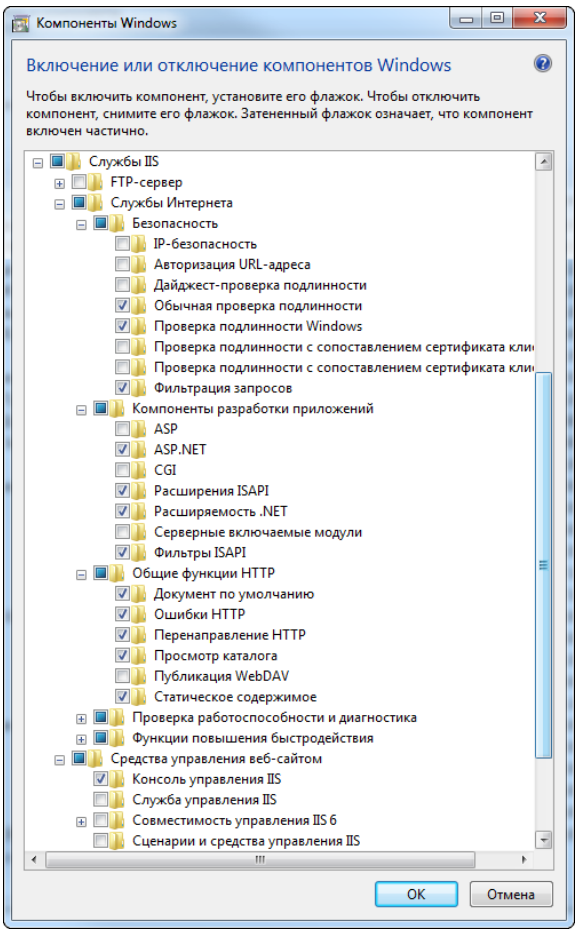


Рисунок 80 – Выбор компонента

После указания адреса в настройках необходимо нажать кнопку «Create Virtual Directory». Затем можно скомпилировать проект и запустить диспетчер служб IIS. Наше приложение уже должно появиться в списке приложений сайта по умолчанию (Default Web Site), а также ему должен быть назначен пул (DefaultAppPool). Рекомендуется создать отдельный пул для ваших сервисов, указав в качестве платформы .NET 4.0 (рисунок 81):

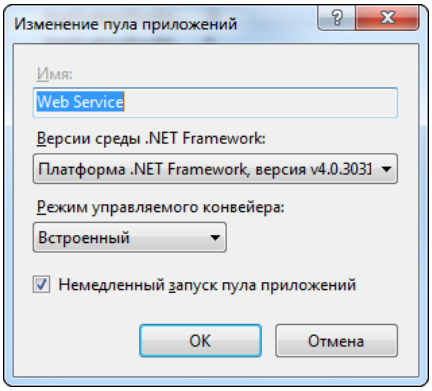


Рисунок 81 – Выбор платформы

После создания пула, назначьте его вашему приложению (Default Web Site > Lesson1 > Дополнительные параметры > Пул приложений).

Последний штрих — проверка вашего приложения. Для этого откройте в браузере строку, указанную в конфигурации проекта, добавив в конце имя сервиса, в нашем случае это Calculator.svc: http://localhost/Lesson1/Calculator.svc. Итог должен быть примерно следующим:

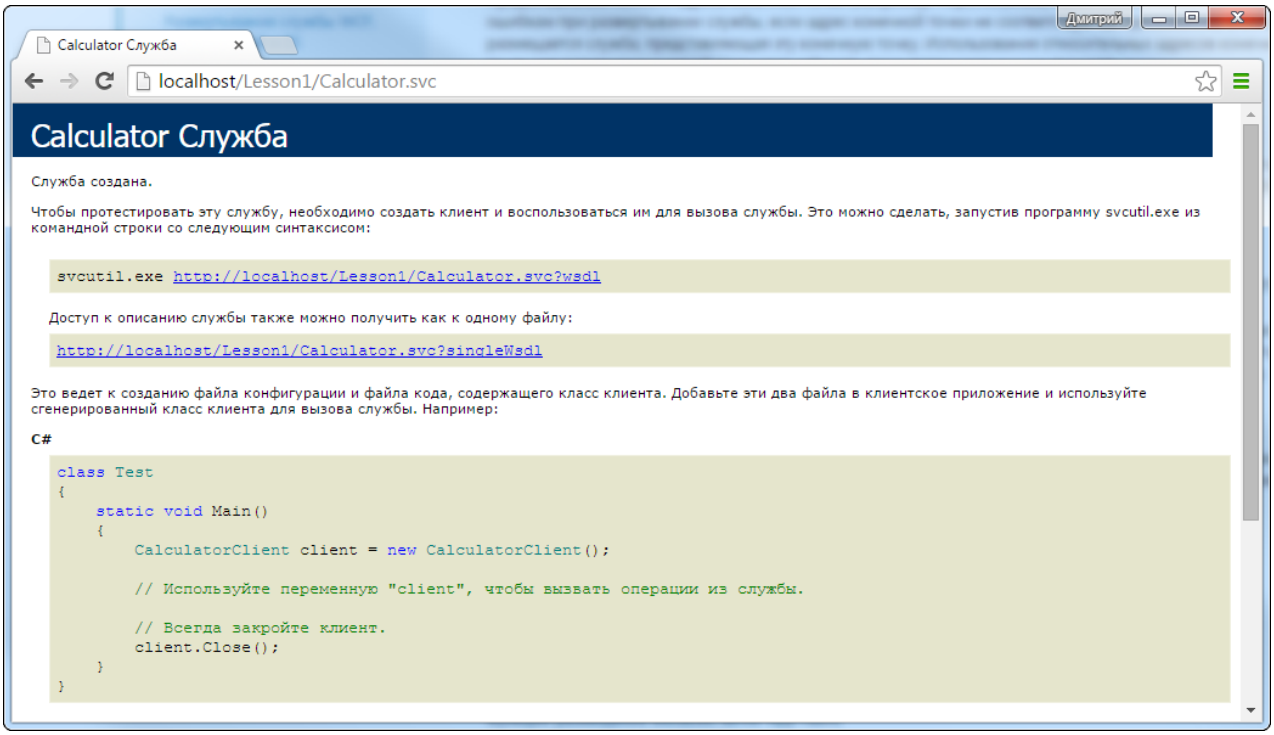


Рисунок 81 - Интерфейс

Внимание! Если в качестве платформы вы выбрали .NET 4.0 и при этом на машине, где размещается сервис установлен .NET 4.0, ссылки на WSDL одним файлом (http://localhost/Lesson1/Calculator.svc?singleWsdl) не будет. Поэтому рекомендуется установить на сервер фреймворк версии не ниже 4.5, даже если сам сервис скомпилирован на .NET 4.0. Это достаточно забавная особенность, но за эту ссылку отвечает именно фреймворк на сервере, а не тот, для которого компилировался сервис. Это, пожалуй, единственное существенное отличие .NET 4.0 от .NET 4.5 с точки зрения интерфейса сервиса.

Теперь немного о WSDL. По сути это XML, содержащая описание сервиса, его методов и точек подключения. Чтобы сторонние интеграторы смогли подключиться к вашему сервису, им нужна ссылка на WSDL. WSDL может быть разбита на части (по умолчанию) или быть одним файлом (см. выше). Важно помнить, что не все клиенты (например, SoapClient на PHP) могут работать с WSDL, разбитой на части, поэтому наличие однофайловой WSDL приветствуется.

3. Публикация сервиса

Под публикацией сервиса подразумевается его компиляция для дальнейшего выкладывания сборки на сервер. Рассмотрим на конкретном примере, выложив сервер на демонстрационный сервер компании.

Для начала, необходимо создать профиль публикации, для этого кликнем правой кнопкой мыши на имени проекта и выберем пункт Publish...:

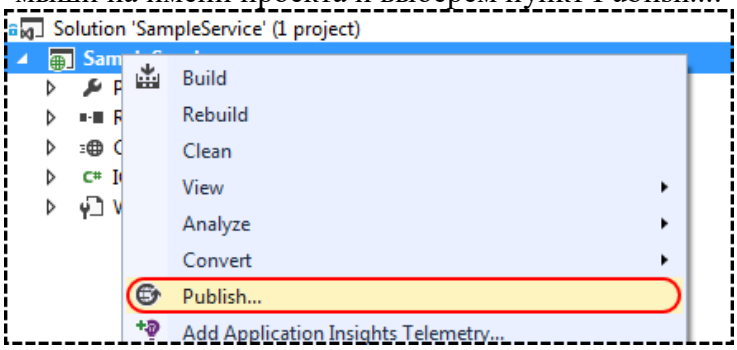


Рисунок 82 - Интерфейс

Мастер предложит нам несколько вариантов публикации, выберем Custom, профиль назовем Demo.
Затем выберем метод публикации File System и укажем путь, куда будет сохраняться наша сборка:

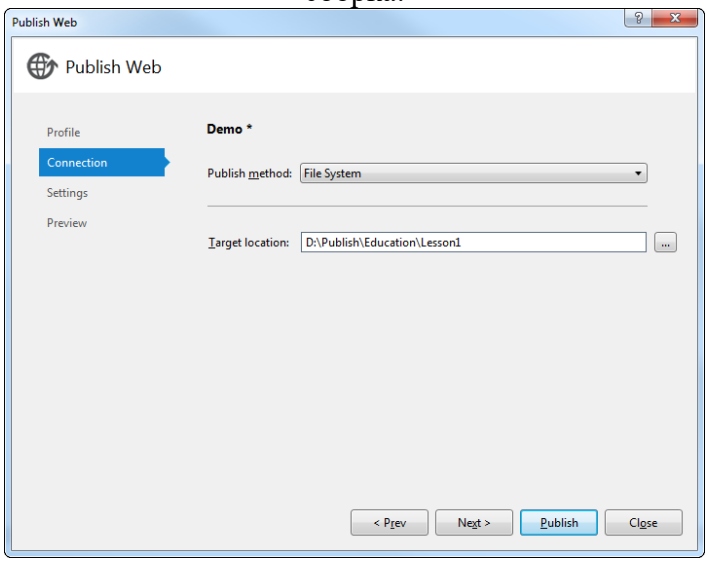


Рисунок 83 - Интерфейс

Проект готов к публикации, но желательно еще перейти к следующей вкладке (Settings) и отметить галочку «Delete all existing files prior to publish» (удалять все файлы

перед публикацией) и выбрать конфигурацию (рисунок 84).

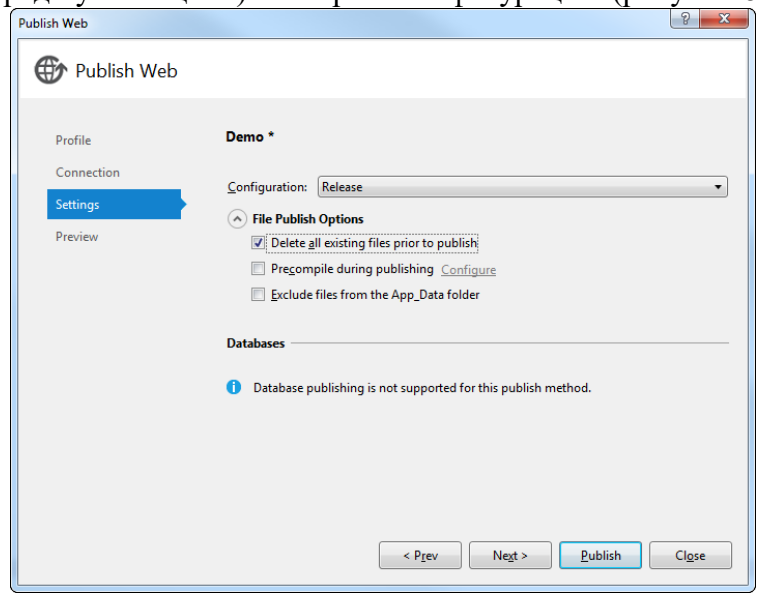


Рисунок 84 - Интерфейс

Выполняем публикацию и смотрим, что получилось в итоге (рисунок 85):

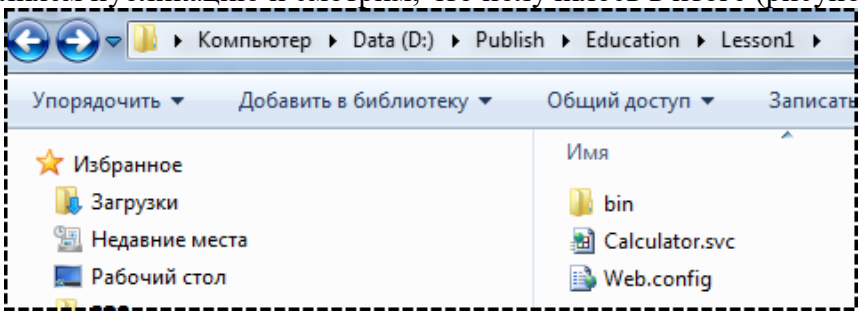


Рисунок 85 - Интерфейс

Чтобы разместить сервис на вашем сайте, скопируйте папку на сервер и подключите к IIS как приложение, как если бы это был обычный сайт (Default Web Site > Add Application...)

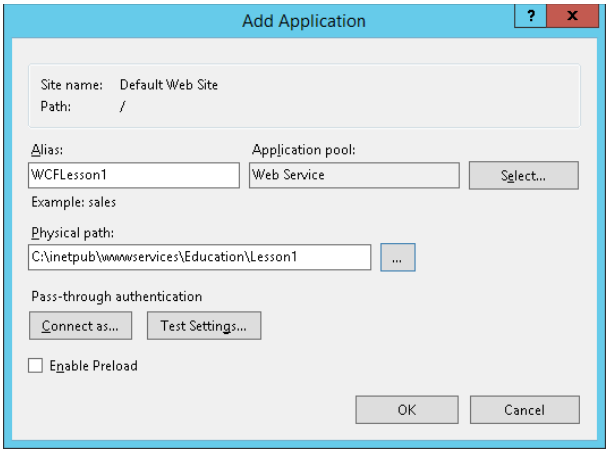


Рисунок 86 – Выбор приложения

Готово! Сервис размещен по адресу http://dszss.proitr.ru/WCFLesson1/Calculator.svc.

Вызов сервиса

В данном разделе мы рассмотрим три способа использования нашего сервиса: вызов через приложение на C#, обращение через SoapUI и запрос посредством URL (WebInvoke).

Клиент на C#

Для начала рассмотрим обращение к нашему сервису через клиент на C#. Так как WCF интегрирован в Visual Studio, создать клиент не составит труда. Все, что нам надо знать — это адрес WSDL сервиса, в которому мы хотим подключиться.

Создадим в нашем решении еще один проект типа Console Application и назовем его TestClient. Назначим его исполняемым и добавим ссылку на сервис (рисунок 87):

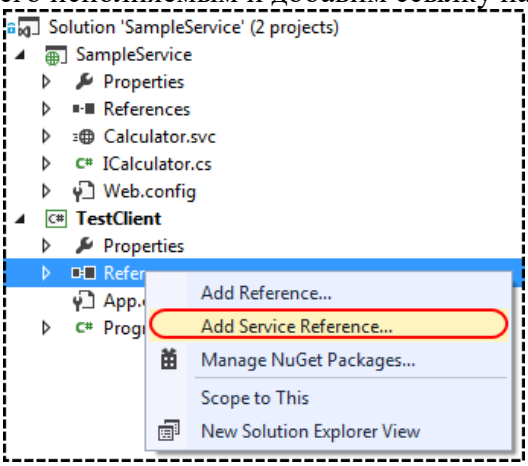


Рисунок 87 - Интерфейс

В открывшемся окне введем адрес локальный сервиса, по желанию укажем имя пространства имен и дополнительные настройки. Здесь же можно посмотреть, какие методы предоставляет наш сервис (рисунок 88):

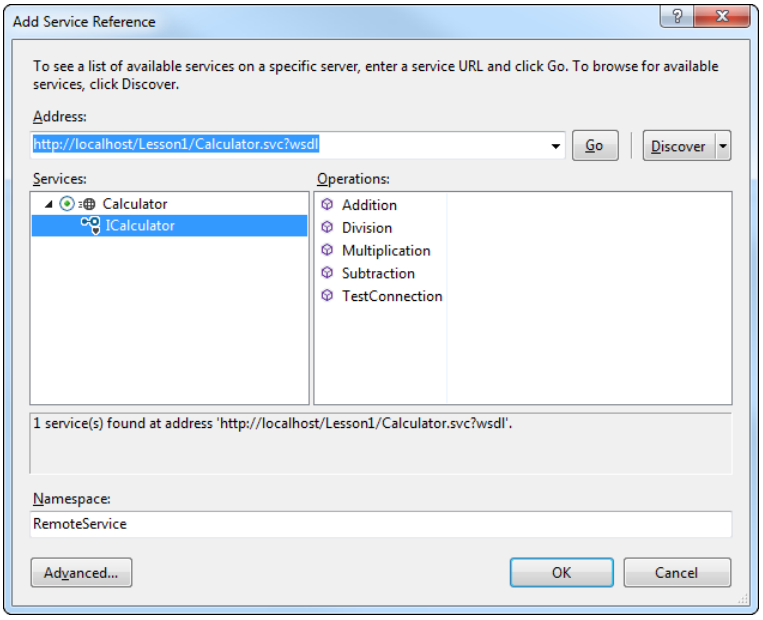


Рисунок 83 - Интерфейс

После создания ссылки на сервис, в проекте появится папка «Service References», в которой будет находиться сгенерированный сутидей клиент. Его код при желании вы можете посмотреть самостоятельно, надо только включить отображение скрытых файлов или перейти по F12 внутрь кода клиента.

Теперь напишем код для подключения к сервису и вызов арифметических операций, предоставляемых им:

Тестовый клиент

Program.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

// ссылка на пространство имен сервиса
using TestClient.RemoteService;

namespace TestClient
{
    class Program
    {
        static void Main(string[] args)
        {
            // помощник вывода в консоль
            ConsoleWriter writer = new ConsoleWriter();
```

```
            // создадим клиент сервиса
            CalculatorClient client = new CalculatorClient("BasicHttpBinding_ICalculator");

            try
            {
                // проверка соединения
                writer.Write("Проверка соединения с сервисом... ");
                if (!string.Equals(client.TestConnection(), "OK", StringComparison.InvariantCultureIgnoreCase))
                {
                    throw new Exception("Проверка соединения не удалась");
                }
                writer.WriteLineSuccess();
                writer.WriteLine();

                // лямбда-функция проверки метода
                var CheckArithmeticOperation = new Action<Func<double, double, double>, string, double, double, double>
                (
                    (operation, operationName, arg1, arg2, expectedResult) =>
                    {
                        writer.Write("Проверка операции '");
                        writer.Write(ConsoleColor.White, operation.Method.Name);
                        writer.Write("', {0} {1} {2} = ", arg1.ToString(CultureInfo.InvariantCulture), operationName, arg2.ToString(CultureInfo.InvariantCulture));
                        double result = operation(arg1, arg2);
                        if (result == expectedResult)
                        {
                            // проверка пройдена
                            writer.Write("{0} ", result.ToString(CultureInfo.InvariantCulture));
                            writer.WriteLineSuccess();
                        }
                        else
                        {
                            // ошибка
                            throw new Exception(string.Format("Ошибка проверки метода '{0}': {1} {2} {3} != {4}",
                                operation.Method.Name,  arg1.ToString(CultureInfo.InvariantCulture), operationName, arg2.ToString(CultureInfo.InvariantCulture), expectedResult.ToString(CultureInfo.InvariantCulture)));
                        }
                    }
                );

                // проверка метода Addition
```

```
            CheckArithmeticOperation(client.Addition, "+", 2.5, 5, 2.5 + 5);

            // проверка метода Subtraction
            CheckArithmeticOperation(client.Subtraction, "-", 2.5, 5, 2.5 - 5);

            // проверка метода Multiplication
            CheckArithmeticOperation(client.Multiplication, "*", 2.5, 5, 2.5 * 5);

            // проверка метода Division
            CheckArithmeticOperation(client.Division, "/", 2.5, 5, 2.5 / 5);

            // в конце работы закрываем клиент
            client.Close();
        }
        catch (Exception ex)
        {
            // в случае ошибки необходимо принудительно закрыть клиент методом Abo
rt()
            client.Abort();

            // выводим информацию об ошибке
            writer.WriteLine();
            writer.WriteLineError("Ошибка: {0}", ex.Message);
        }

        Console.WriteLine();
        Console.WriteLine("Нажмите любую клавишу для продолжения...");
        Console.ReadKey();
    }
  }
}
```

ConsoleWriter.cs

```
using System;

namespace TestClient
{
  /// <summary>
  /// консольный писатель
  /// </summary>
  public class ConsoleWriter
  {
    #region Declarations
```

```
        private ConsoleColor _successColor;     // цвет сообщений для успешных операций
        private ConsoleColor _errorColor;       // цвет сообщений об ошибках
        private ConsoleColor _warningColor;     // цвет сообщений-предупреждений

        private string _successText;            // текст сообщений для успешных операций
        private string _errorText;              // текст сообщений об ошибках
        private string _warningText;            // текст сообщений-предупреждений

        #endregion
        #region Properties

        /// <summary>
        /// цвет сообщений для успешных операций
        /// </summary>
        public ConsoleColor SuccessColor { get { return _successColor; } set { _successColor
= value; } }

        /// <summary>
        /// цвет сообщений об ошибках
        /// </summary>
        public ConsoleColor ErrorColor { get { return _errorColor; } set { _errorColor = value
; } }

        /// <summary>
        /// цвет сообщений-предупреждений
        /// </summary>
        public ConsoleColor WarningColor { get { return _warningColor; } set { _warningCol
or = value; } }

        /// <summary>
        /// текст сообщений для успешных операций
        /// </summary>
        public string SuccessText { get { return _successText; } set { _successText = value; } }

        /// <summary>
        /// текст сообщений об ошибках
        /// </summary>
        public string ErrorText { get { return _errorText; } set { _errorText = value; } }

        /// <summary>
        /// текст сообщений-предупреждений
        /// </summary>
```

```
        public string WarningText { get { return _warningText; } set { _warningText = value; } }

        /// <summary>
        /// цвет текста
        /// </summary>
        public ConsoleColor ForegroundColor { get { return Console.ForegroundColor; } set { Console.ForegroundColor = value; } }

        /// <summary>
        /// цвет фона
        /// </summary>
        public ConsoleColor BackgroundColor { get { return Console.BackgroundColor; } set { Console.BackgroundColor = value; } }

        #endregion

        #region Constructors

        /// <summary>
        /// конструктор
        /// </summary>
        public ConsoleWriter()
        {
            _successColor = ConsoleColor.Green;
            _errorColor = ConsoleColor.Red;
            _warningColor = ConsoleColor.Blue;

            _successText = "OK";
            _errorText = "ERROR";
            _warningText = "WARNING";
        }

        #endregion

        #region Private methods
        #endregion
        #region Protected methods
        #endregion
        #region Public methods

        #region Write | WriteLine

        /// <summary>
```

```
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void Write(string value)
        {
            Console.Write(value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="color"> цвет сообщения </param>
        /// <param name="value"> сообщение </param>
        public void Write(ConsoleColor color, string value)
        {
            ConsoleColor oldColor = Console.ForegroundColor;
            Console.ForegroundColor = color;
            Console.Write(value);
            Console.ForegroundColor = oldColor;
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void Write(string format, params object[] args)
        {
            Console.Write(string.Format(format, args));
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="color"> цвет сообщения </param>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void Write(ConsoleColor color, string format, params object[] args)
        {
            ConsoleColor oldColor = Console.ForegroundColor;
            Console.ForegroundColor = color;
            Console.Write(string.Format(format, args));
            Console.ForegroundColor = oldColor;
        }
```

```
        /// <summary>
        /// записать перевод строки в консоль
        /// </summary>
        public void WriteLine()
        {
            Console.WriteLine();
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteLine(string value)
        {
            Console.WriteLine(value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="color"> цвет сообщения </param>
        /// <param name="value"> сообщение </param>
        public void WriteLine(ConsoleColor color, string value)
        {
            ConsoleColor oldColor = Console.ForegroundColor;
            Console.ForegroundColor = color;
            Console.WriteLine(value);
            Console.ForegroundColor = oldColor;
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteLine(string format, params object[] args)
        {
            Console.WriteLine(string.Format(format, args));
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
```

```
        /// <param name="color"> цвет сообщения </param>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteLine(ConsoleColor color, string format, params object[] args)
        {
            ConsoleColor oldColor = Console.ForegroundColor;
            Console.ForegroundColor = color;
            Console.WriteLine(string.Format(format, args));
            Console.ForegroundColor = oldColor;
        }

        #endregion
        #region WriteSuccess | WriteLineSuccess

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        public void WriteSuccess()
        {
            Write(_successColor, _successText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteSuccess(string value)
        {
            Write(_successColor, value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteSuccess(string format, params object[] args)
        {
            Write(_successColor, string.Format(format, args));
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
```

```
        public void WriteLineSuccess()
        {
            WriteLine(_successColor, _successText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteLineSuccess(string value)
        {
            WriteLine(_successColor, value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteLineSuccess(string format, params object[] args)
        {
            WriteLine(_successColor, string.Format(format, args));
        }

        #endregion
        #region WriteError | WriteLineError

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        public void WriteError()
        {
            Write(_errorColor, _errorText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteError(string value)
        {
            Write(_errorColor, value);
        }
```

```
        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteError(string format, params object[] args)
        {
            Write(_errorColor, string.Format(format, args));
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        public void WriteLineError()
        {
            WriteLine(_errorColor, _errorText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteLineError(string value)
        {
            WriteLine(_errorColor, value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteLineError(string format, params object[] args)
        {
            WriteLine(_errorColor, string.Format(format, args));
        }

        #endregion
        #region WriteWarning | WriteLineWarning

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        public void WriteWarning()
```

```
        {
            Write(_warningColor, _warningText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteWarning(string value)
        {
            Write(_warningColor, value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
        /// <param name="args"> аргументы </param>
        public void WriteWarning(string format, params object[] args)
        {
            Write(_warningColor, string.Format(format, args));
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        public void WriteLineWarning()
        {
            WriteLine(_warningColor, _warningText);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="value"> сообщение </param>
        public void WriteLineWarning(string value)
        {
            WriteLine(_warningColor, value);
        }

        /// <summary>
        /// записать сообщение в консоль
        /// </summary>
        /// <param name="format"> строка формата </param>
```

```
        /// <param name="args"> аргументы </param>
        public void WriteLineWarning(string format, params object[] args)
        {
            WriteLine(_warningColor, string.Format(format, args));
        }

        #endregion

        #endregion
    }
}
```

Результат исполнения:

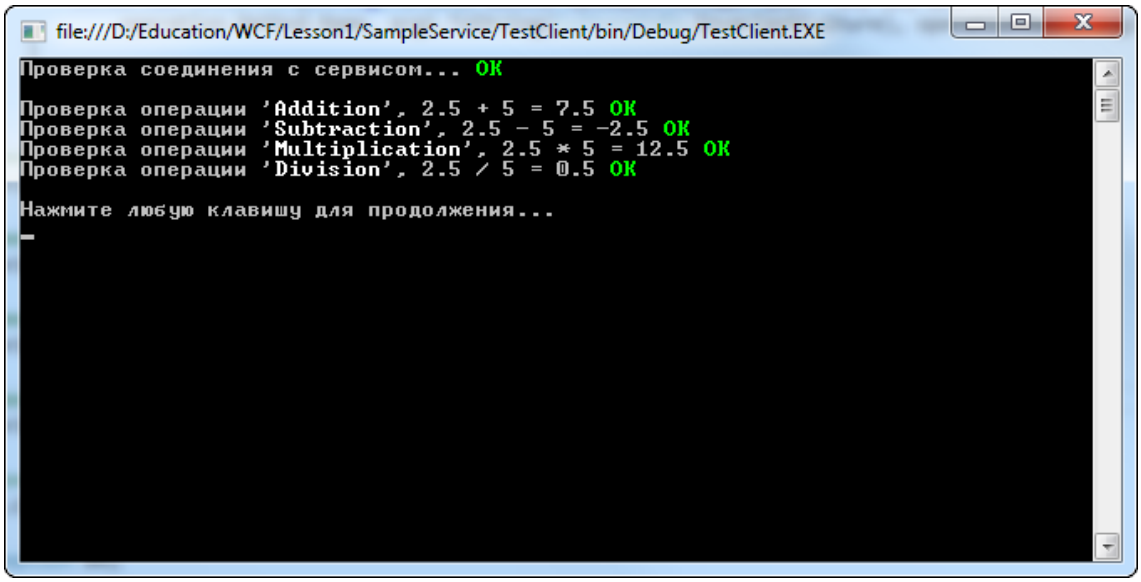


Рисунок 84 – Командная строка

В целом, вызов методов сервиса ничем не отличается от вызова методов обычного интерфейса C#, а т.к. клиент и сервис находятся в пределах одного решения, вы можете поставить контрольные точки в методах сервиса, которые сработают при их вызове. Также вы можете запустить отдельный проект сервиса в режиме ожидания запроса от стороннего приложения.

Технически, при обращении к сервису клиент формирует Soap сообщение (обычная XML), которое предается по HTTP-каналу и может быть перехвачено сниффером. Разумеется, в арсенале WCF существуют и другие методы передачи, а также возможность шифрования и/или подписи сообщений, но это уже тема другого урока.

Переходим к другому способу вызова сервиса.

SoapUI

SoapUI – бесплатная программа для тестирования Soap-сервисов. Хороша тем, что позволяет заглянуть во внутреннее устройство сообщений, что может быть очень полезно в разрешении сложных проблем, когда клиент и сервис используют разные платформы. В любом случае, SoapUI — альтернативный клиент, знакомство с которым необходимо, если вы решили серьезно заняться разработкой сервисов.

Запускаем программу и создаем новый проект. В поле адреса сервиса указываем на WSDL и вводим название проекта:

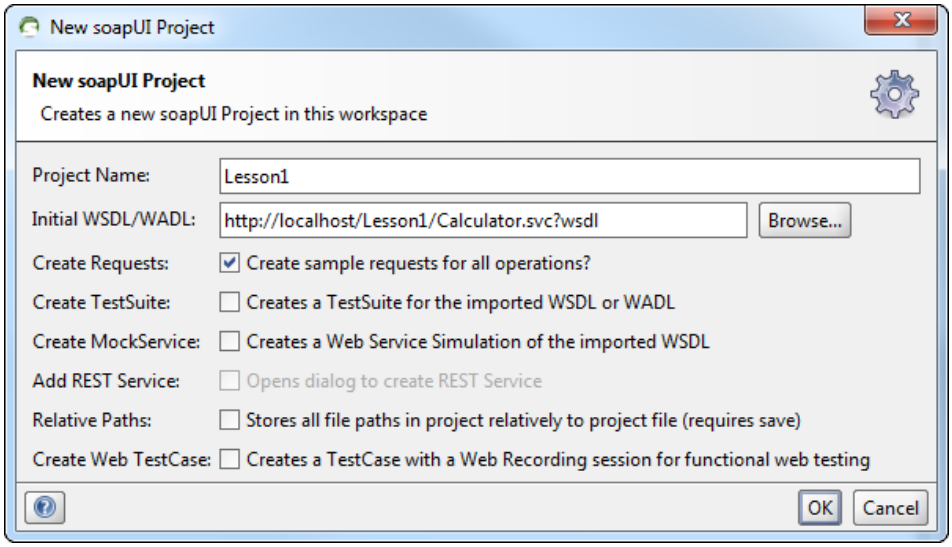
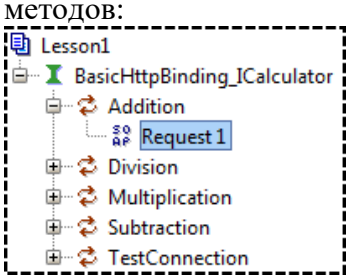


Рисунок 85 - Интерфейс

Как видим, SoapUI извлек интерфейсы сервиса и создал тестовые запросы для каждого из методов:



Осталось заполнить числовые значения и вызвать сервис:

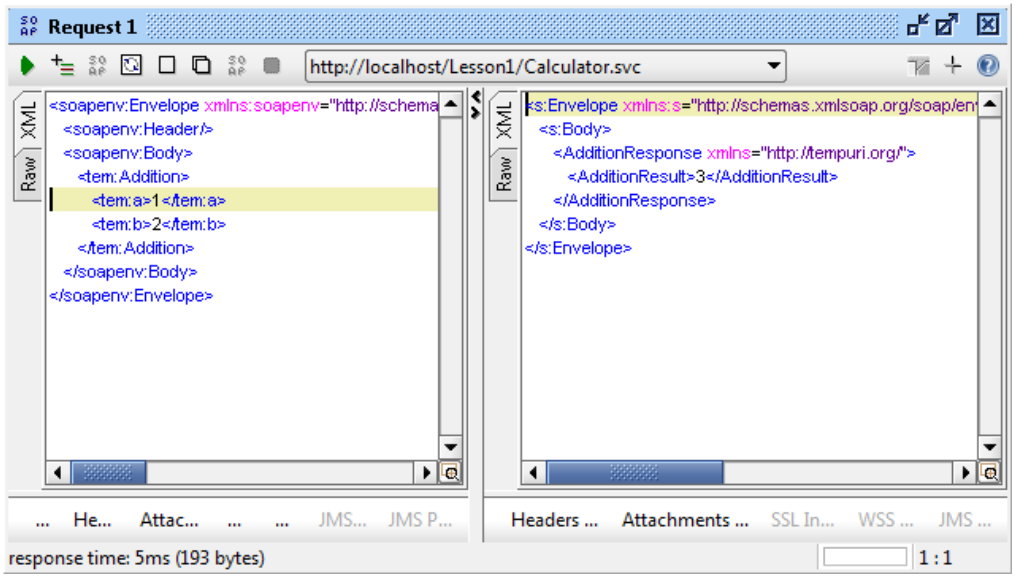


Рисунок 86 - Интерфейс

Возможно, вы обратили внимание на странное пространство имен xmlns=«tempuri.org». Это результат того, что настройки пространств имен сервиса нами не были заданы в самом начале. Поскольку оставлять tempuri.org крайне не рекомендуется, изменим его прямо сейчас. Для этого пропишем желаемое нами пространство имен в атрибуте интерфейса сервиса **ServiceContract**:

```
[ServiceContract(Namespace = "http://dszss.proitr.ru/WCF")]
public interface ICalculator
{
    ...
}
```

После компиляции сервиса, тестовый клиент и SoapUI станут выдавать ошибку «Сообщение с Action „tempuri.org/ICalculator/Addition“ не может быть обработано на стороне получателя из-за несоответствия ContractFilter на EndpointDispatcher». В тестовом клиенте необходимо обновить конфигурацию сервиса (RemoteService > Update Service Reference), а в SoapUI – обновить описание (Update Definition), нажав F5 на имени точки подключения BasicHttpBinding_ICalculator и обновить запрос.

Запрос-ответ после изменения пространства имен (рисунок 87):

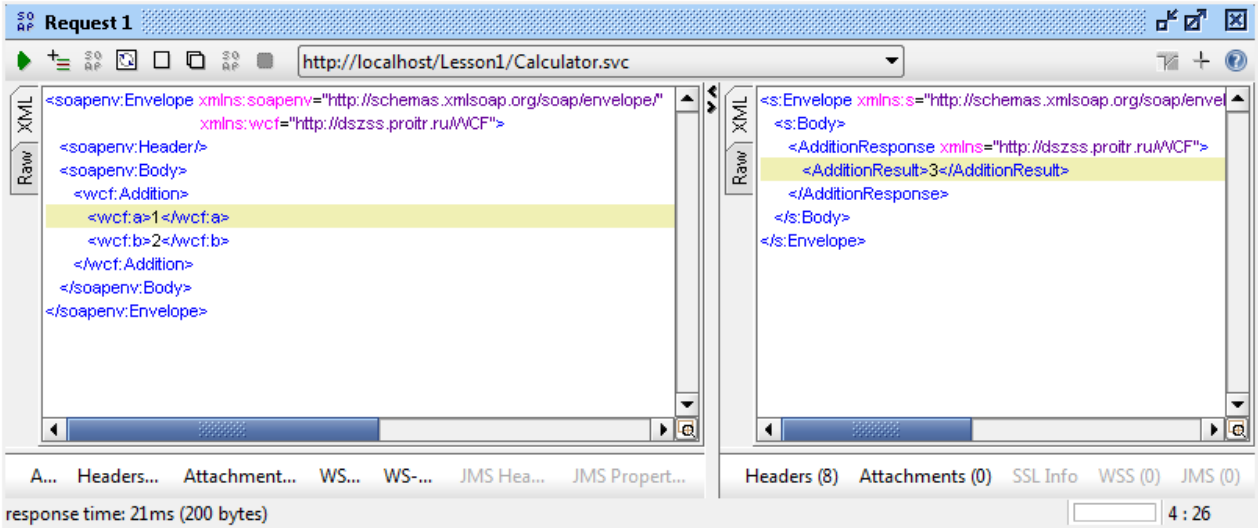


Рисунок 87 - Интерфейс

С остальными возможностями SoapUI предлагаем ознакомиться самостоятельно.

WebInvoke

WebInvoke – это возможность вызова метода WCF-сервисов «по ссылке», например с помощью Ajax.

Для начала, изменим конфигурацию:

Web.config

```
<?xml version="1.0"?>
<configuration>

  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>

  <system.web>
    <compilation debug="true" targetFramework="4.5.1" />
    <httpRuntime targetFramework="4.5.1"/>
  </system.web>

  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="Basic" />
        <behavior name="WebJson">
          <webHttp defaultOutgoingResponseFormat="Json" faultExceptionEnabled="true" />
        </behavior>
```

```
        <behavior name="WebXML">
          <webHttp defaultOutgoingResponseFormat="Xml" faultExceptionEnabled="true" />
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>

    <bindings>
      <basicHttpBinding>
        <binding name="Basic" closeTimeout="00:01:00" openTimeout="00:01:00"
                 receiveTimeout="00:20:00" sendTimeout="00:20:00" maxBufferPoolSize="5242
88000"
                 maxBufferSize="65536000" maxReceivedMessageSize="65536000">
          <readerQuotas maxDepth="65536000" maxStringContentLength="65536000"
                        maxArrayLength="65536000" maxBytesPerRead="65536000" maxNameTa
bleCharCount="65536000" />
        </binding>
      </basicHttpBinding>
      <webHttpBinding>
        <binding name="WebJson" closeTimeout="00:01:00" openTimeout="00:01:00"
                 receiveTimeout="00:20:00" sendTimeout="00:20:00" maxBufferSize="6553600
0"
                 maxBufferPoolSize="524288000" maxReceivedMessageSize="65536000">
          <readerQuotas maxDepth="65536000" maxStringContentLength="65536000"
                        maxArrayLength="65536000" maxBytesPerRead="65536000" maxNameTa
bleCharCount="65536000" />
        </binding>
        <binding name="WebXML" closeTimeout="00:01:00" openTimeout="00:01:00"
                 receiveTimeout="00:20:00" sendTimeout="00:20:00" maxBufferSize="6553600
0"
                 maxBufferPoolSize="524288000" maxReceivedMessageSize="65536000">
          <readerQuotas maxDepth="65536000" maxStringContentLength="65536000"
                        maxArrayLength="65536000" maxBytesPerRead="65536000" maxNameTa
bleCharCount="65536000" />
        </binding>
      </webHttpBinding>
```

```
    </bindings>

    <services>
      <service name="SampleService.Calculator">
        <!-- конечная точка SOAP -->
        <endpoint address="basic" binding="basicHttpBinding" behaviorConfiguration="Basic
"
                  bindingConfiguration="Basic" name="Basic" bindingNamespace="http://dszss.p
roitr.ru/WCF"
                  contract="SampleService.ICalculator" />

        <!-- конечная точка REST (формат Json) -->
        <endpoint address="json" binding="webHttpBinding" behaviorConfiguration="WebJs
on"
                  bindingConfiguration="WebJson" name="WebJson" bindingNamespace="http://
dszss.proitr.ru/WCF"
                  contract="SampleService.ICalculator" />

        <!-- конечная точка REST (формат XML) -->
        <endpoint address="xml" binding="webHttpBinding" behaviorConfiguration="WebX
ML"
                  bindingConfiguration="WebXML" name="WebXML" bindingNamespace="http
://dszss.proitr.ru/WCF"
                  contract="SampleService.ICalculator" />
      </service>
    </services>

    <protocolMapping>
      <add binding="basicHttpsBinding" scheme="https" />
    </protocolMapping>

    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsE
nabled="true" />
  </system.serviceModel>

  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true" />
    <directoryBrowse enabled="false" />
  </system.webServer>

</configuration>
```

Добавленные разделы:

- **system.serviceModel/behaviors/endpointBehaviors** – поведения конечных точек. Ранее у нас была одна конечная точка, которая принимала запросы от тестового клиента или SoapUI, теперь точки три, и для каждой надо создать свое поведение. defaultOutgoingResponseFormat определяет формат ответных сообщений — XML или Json. В целях демонстрации создадим дополнительные точки для каждого из форматов.
- **system.serviceModel/bindings** — добавлены привязки для каждой из конечных точек. Несложно догадаться, что чему соответствует.
- **system.serviceModel/services** — определения конечных точек. Здесь связаны интерфейсы сервиса связаны со своими конечными точками. До этого у нас была всего одна конечная точка типа basicHttpBinding, поэтому данный раздел не требовался.

Одного изменения конфигурации недостаточно, требуется также описать маппинги для методов интерфейса ICalculator. Пример для метода Addition:

```
[OperationContract]
[WebInvoke(Method = "GET", UriTemplate = "Add?a={a}&b={b}")]
double Addition(double a, double b);
```

Добавлен атрибут WebInvoke, в котором мы можем указан HTTP-метод (GET, POST, PUT или DELETE), а также часть URL, в которой будут указаны аргументы метода. Т.к. аргументы не строковые, их необходимо указывать в параметрах.

Если бы, допустим, у нас был метод, принимающий строки, можно было бы указать аргументы через слеш, например так:

```
[OperationContract]
[WebInvoke(Method = "GET", UriTemplate = "Add/{a}/{b}")]
double Addition(string a, string b);
```

Но у нас числа, поэтому прописываем по аналогии атрибут WebInvoke в прочим методам, в итоге получаем следующий код:

ICalculator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
```

```
namespace SampleService
{
    [ServiceContract(Namespace = "http://dszss.proitr.ru/WCF")]
    public interface ICalculator
    {
        #region Common Methods

        /// <summary>
        /// проверка соединения
        /// </summary>
        /// <returns> OK </returns>
        [OperationContract]
        [WebInvoke(Method = "GET")]
        string TestConnection();

        #endregion

        #region Arithmetic

        /// <summary>
        /// сложение
        /// </summary>
        /// <param name="a"> слагаемое 1 </param>
        /// <param name="b"> слагаемое 2 </param>
        /// <returns> сумма </returns>
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Add?a={a}&b={b}")]
        double Addition(double a, double b);

        /// <summary>
        /// вычитание
        /// </summary>
        /// <param name="a"> уменьшаемое </param>
        /// <param name="b"> вычитаемое </param>
        /// <returns> разность </returns>
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Sub?a={a}&b={b}")]
        double Subtraction(double a, double b);

        /// <summary>
        /// умножение
        /// </summary>
        /// <param name="a"> множитель 1 </param>
```

```
        /// <param name="b"> множитель 2 </param>
        /// <returns> произведение </returns>
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Mul?a={a}&b={b}")]
        double Multiplication(double a, double b);

        /// <summary>
        /// деление
        /// </summary>
        /// <param name="a"> делимое </param>
        /// <param name="b"> делитель </param>
        /// <returns> частное </returns>
        [OperationContract]
        [WebInvoke(Method = "GET", UriTemplate = "Div?a={a}&b={b}")]
        double Division(double a, double b);

        #endregion
    }
}
```

После компиляции сервиса мы получаем возможность вызывать методы через URL, не самую приятную на вид, но все же.

Вызов метода сложения с результатом в XML http://localhost/Lesson1/Calculator.svc/xml/Add?a=2&b=3.5:

Аналогичный метод, но с результатом в Json http://localhost/Lesson1/Calculator.svc/json/Add?a=2&b=3.5:

Сам адрес, как видим, состоит из пути к сервису (http://localhost/Lesson1/Calculator.svc), имени конечной точки (xml или json) и части адреса из UriTemplate. Для метода TestConnection UriTemplate не задан, поэтому используется значение по умолчанию — имя метода: http://localhost/Lesson1/Calculator.svc/xml/TestConnection

12.4 Требования к отчету и защите:

Необходимо выложить отчет о выполнении работы в ЭИОС. Отчет содержит титульный лист и скриншоты работоспособности программы и ее выполнения в различных вариациях ввода значений.

13 ЗАКЛЮЧЕНИЕ

Учебно-методическое пособие позволяет студентам освоить базовые знания и приобрести навыки по разработке, защите и эксплуатации распределенных приложений для открытых систем на платформе WCF.

Для углубления своих знаний в области создания распределенных приложений и служб и обеспечения их безопасности студенты могут использовать учебные ресурсы, часть из которых представлена в списке литературы.

14 ЛИТЕРАТУРА

1. Резник, С. Основы Windows Communication Foundation для .NET Framework 3.5: Пер. с англ. А. А. Слинкина / С. Резник, Р. Крейн, К. Боуэн. – Москва: ДМК Пресс, 2008. – 480 с.

2. Ахтырченко, К. В. Распределенные объектные технологии в информационных системах / К. В. Ахтырченко, В. В. Леонтьев. – Москва : Мир, 2001. – 560 с.

3. Коцюба, И. Ю. Основы проектирования информационных систем: учеб. пособие / И. Ю. Коцюба, А. В.Чунаев, А. Н.Шиков. – Санкт-Петербург: Университет ИТМО, 2015. – 206 с.

4. Таненбаум, Э. М. Распределенные системы. Принципы и парадигмы / Э. М. Таненбаум. – Санкт-Петербург : ПИТЕР, 2003. – 877 с.

5. Технологии разработки программного обеспечения: учебник / С. Орлов. — Санкт-Петербург: Питер, 2002. — 464 с.: ил.

6. Электронная информационная образовательная среда БГАРФ ФГБОУ ВО «КГТУ»: http://83.171.112.16/login/index.php

7. Интернет-ресурсы:

https://studbooks.net/2222776/informatika/

https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/feature-details/security

https://coderlessons.com/tutorials/microsoft-technologies/uznaite-wcf/wcf-bezopasnost

http://www.javable.com/docs/articles/dist/

http://bgarf.ru/academy/biblioteka/elektronnyj-katalog/

Локальный электронный методический материал

Алина Андреевна Бабаева

ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ ПРИЛОЖЕНИЙ ДЛЯ ОТКРЫТЫХ СИСТЕМ

Редактор Г. А. Смирнова

Уч.-изд. л. 7,5. Печ. л. 7,5

Издательство федерального государственного бюджетного образовательного учреждения высшего образования
«Калининградский государственный технический университет».
236022, Калининград, Советский проспект, 1