Лабораторная работа №1 – Windows Forms

1 Цель работы

Изучить основы построения Windows приложений на Visual Studio.NET на основе событийно-реакционной модели приложений. Обучиться основным приемам использования проекта Windows-forms.

2 Порядок выполнения работы

– Прочесть краткие теоретические сведения.
– Выполнить задания раздела.
– Составить отчет о лабораторной работе и защитить его у преподавателя.

3 Теоретическая часть

3.1 Событийно-управляемое программирование

В основу Windows положен принцип событийного управления. Это значит, что и сама система, и приложения после запуска ожидают действий пользователя и реагируют на них заранее заданным образом. Любое действие пользователя (нажатие клавиши на клавиатуре, щелчок кнопкой мыши, перемещение мыши) называется событием.

Событие воспринимается Windows и преобразуется в сообщение — запись, содержащую необходимую информацию о событии (например, какая клавиша была нажата, в каком месте экрана произошел щелчок мышью). Сообщения могут поступать не только от пользователя, но и от самой системы, а также от активного или других приложений. Определен достаточно широкий круг стандартных сообщений, образующий иерархию, кроме того, можно определять собственные сообщения.

Сообщения поступают в общую очередь, откуда распределяются по очередям приложений. Каждое приложение содержит цикл обработки сообщений, который выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки. Таким образом, Windows-приложение состоит из главной программы, обеспечивающей инициализацию и завершение приложения, цикла обработки сообщений и набора обработчиков событий.

Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу — создание шаблонов приложения и форм, заготовок обработчиков событий, организацию цикла обработки сообщений.

3.2 Создание Windows-приложения

Создадим новый проект (Файл → Создать проект), выбрав шаблон ***Приложение Windows Forms*** (рисунок 1). После этого среда сформирует шаблон Windows-приложения (рисунок 2).

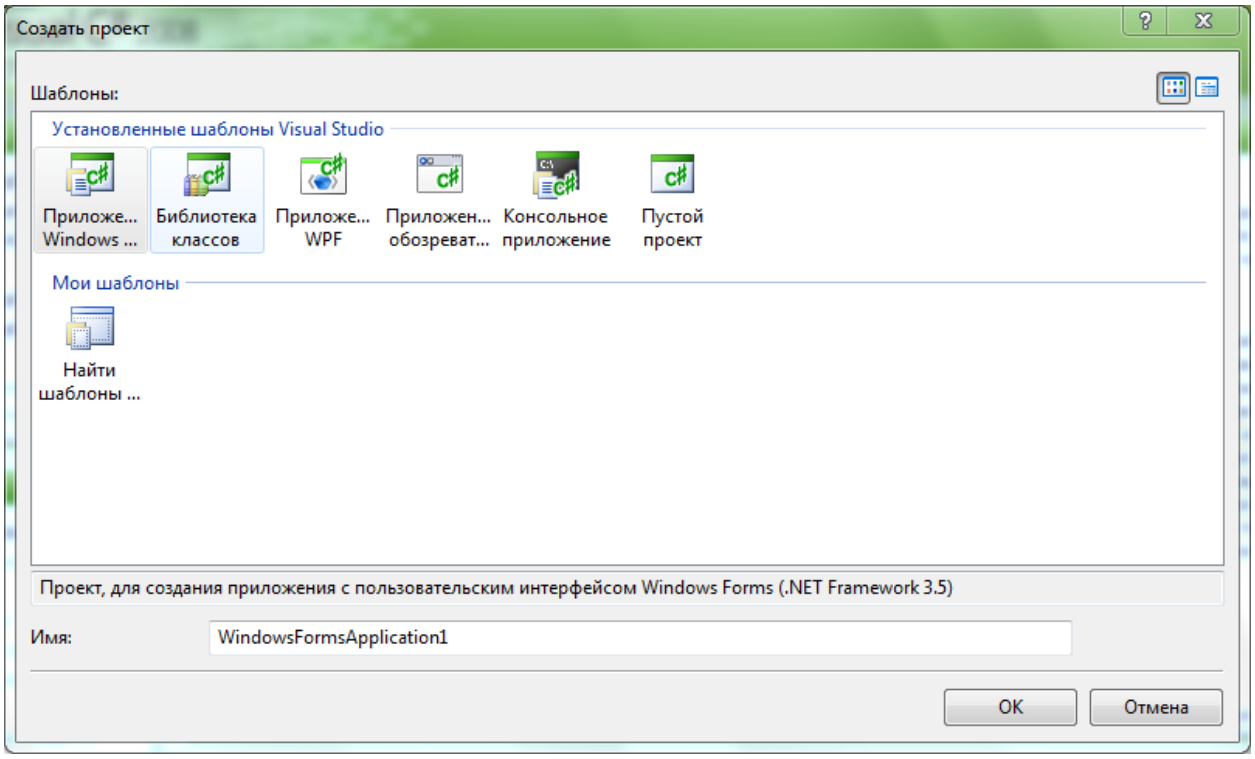

Рисунок 1 – Окно выбора шаблона программы

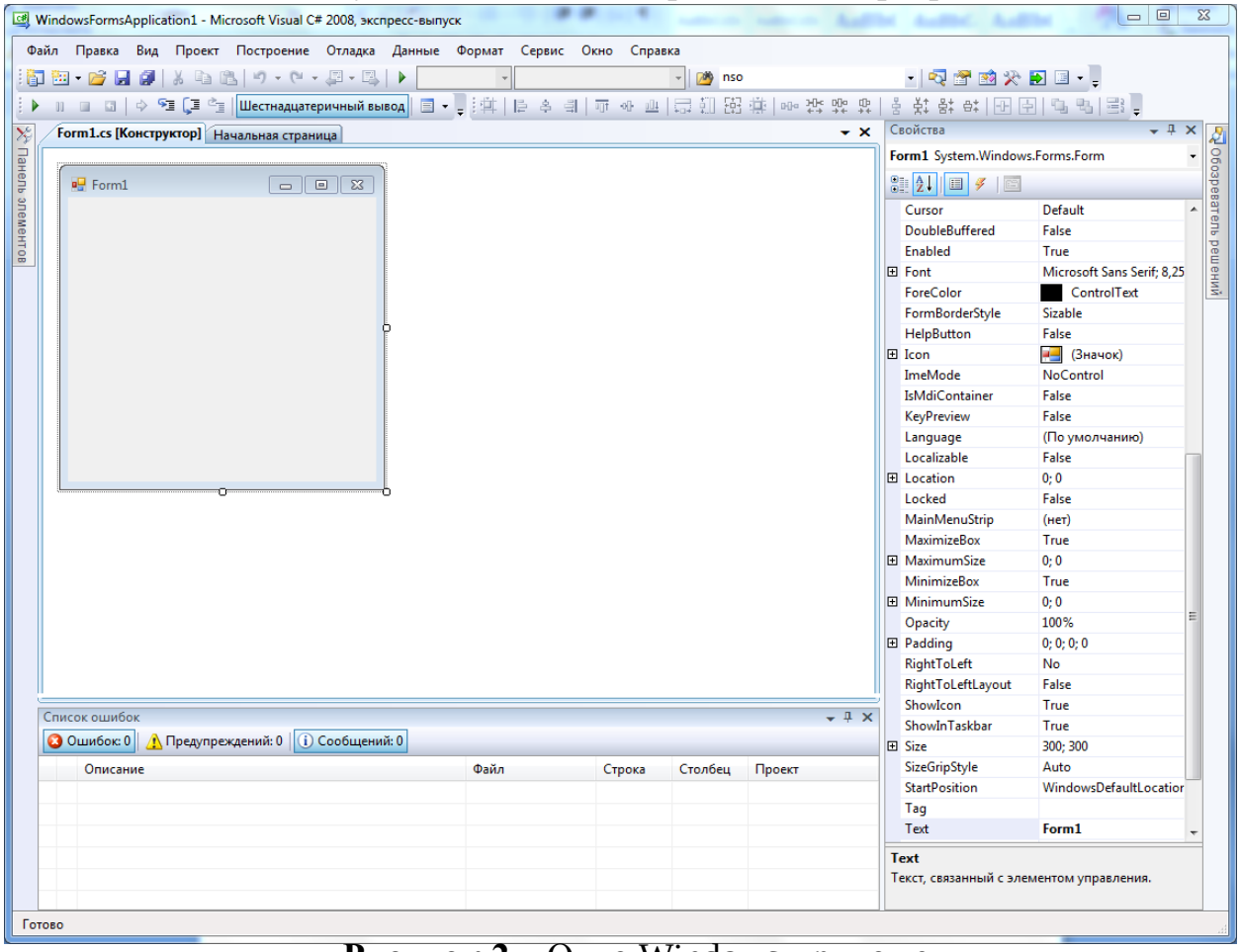

Рисунок 2 – Окно Windows-приложения

Среда создает не только заготовку формы, но и шаблон текста приложения. Перейти к нему можно, щелкнув в окне ***Обозреватель решений*** (Вид → Обозреватель решений) правой кнопкой мыши на файле Forml .cs и выбрав в

контекстном меню команду ***Просмотреть код***. При этом откроется вкладка с кодом формы, который приведен ниже:

```
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();
        }
    }
}
```

Приложение начинается с директив использования пространств имен библиотеки .NET.

```
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;
```

Далее следует описание пространства имен `WindowsFormsApplication1`, описание класса `Form1`, наследника класса `Form`, и описание конструктора класса `Form1()` с единственным оператором `InitializeComponent()`, осуществляющим инициализацию компонентов фармы.
Добавим с проект новый класс (клик правой кнопкой по проекту Добавить →Класс) и в открывшемся окне редактора кода, модифицируем новый класс следующим образом:

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

namespace WindowsFormsApplication1
{
    class Class1
    {
        private int real;
        private int img;
        public void Input(int r, int i)
        {
            real = r;
            img = i;
        }
        public string Output()
        {
            string OutStr;
            if (real == 0 && img == 0)
            {
                OutStr = "Комплексное число не введено!";
                return OutStr;
            }
            else
            {
                OutStr = "Целая часть = " + real + ", мнимая часть
= " + img + ".";
                return OutStr;
            }
        }
    }
}
```

Затем добавим на форму новые компоненты, а именно 2 кнопки(Button), 2 поля ввода(TextBox), и одну метку(Label)

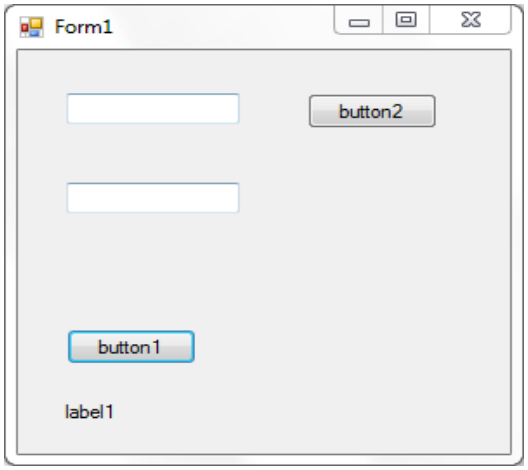

Рисунок 3 – Форма программы

Далее модифицируем код формы следующим образом (обратите внимание на то, что обработчики событий `button1_Click` и `button2_Click` генерируются средой автоматически, при двойном щелчке на соответствующие кнопки):

```
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();
        }

        Class1 r1 = new Class1();

        private void button1_Click(object sender, EventArgs e)
        {
            label1.Text =  r1.Output();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "" && textBox2.Text != "")
r1.Input(Convert.ToInt32(textBox1.Text),
Convert.ToInt32(textBox2.Text));
            textBox1.Clear();
            textBox2.Clear();
        }
    }
}
```

Теперь запустим программу, и увидим, что она представляет собой простой пример реализации элементарного класса, осуществляющего формирование комплексного числа из строк, содержащихся в полях ввода. Заметим, что туда необходимо вводить ТОЛЬКО ЦИФРЫ. При несоблюдении этого правила возникнет исключение и ошибка. Данное исключение можно будет обработать, и устранить подобную ошибку, но механизм обработки исключений не является темой данной лабораторной работы и будет рассмотрен в оной из следующих.

3.3 Некоторые приемы работы с формой

3.3.1 Получение доступа к методам и свойствам формы из ее кода

Получить доступ к методам и свойствам формы можно, используя ключевое слово ***this***. Например:

```
this.Text = "Я заголовок формы!";
```

Кроме того, можно создать новые экземпляры форм во время выполнения, описав переменную, представляющую тип формы, и создав экземпляр данной формы.

Свойства форм Windows
Внешний вид пользовательского интерфейса очень важен для приложения. Плохо разработанный пользовательский интерфейс трудно понять. Его изучение займет много времени и потребует больших затрат. Изменить внешний вид пользовательского интерфейса можно с помощью свойств форм Windows.
Формы Windows содержат множество свойств, позволяющих настраивать их внешний вид и поведение. Просматривать и изменять эти свойства можно в окне Properties конструктора.

В таблице 1 перечислены некоторые свойства форм Windows, отвечающие за внешний вид и поведение приложения. Обратите внимание, что здесь представлен не полный список свойств форм Windows, а их избранная подгруппа.
Таблица 1 – Некоторые свойства класса *Form*

Свойство	*Описание*
Name	Задает имя классу ***Form,*** показанному в конструкторе. Данное свойство задается исключительно во время разработки
BackColor	Указывает цвет фона формы
Backgroundlmage	Указывает фоновое изображение формы
Background-Image Layout	Определяет, как изображение, указанное свойством ***Backgroundlmage,*** будет располагаться в форме. Если фоновое изображение не выбрано, данное свойство игнорируется
ControlBox	Определяет, имеет ли форма оконное меню Control/System
Cursor	Указывает курсор, который появляется при наведении мыши на форму

Enabled	Указывает, может ли форма принимать ввод от пользователя. Если свойству ***Enabled*** задано значение ***False,*** все элементы управления формы также блокируются
Font	Задает шрифту формы значение по умолчанию. Если отдельно не указать значение свойства *Font* элементов управления формы, они примут это же значение по умолчанию
Fore Color	Указывает цвет переднего плана формы, то есть цвет выводимого текста. Если отдельно не указать значение свойства *Fore Color* элементов управления формы, они примут то же значение
FormBorderStyle	Указывает вид и поведение границы и строки заголовка формы
HelpButton	Указывает, есть ли у формы кнопка Help
Icon	Указывает значок, расположенный в заголовке формы
Location	Когда свойству *StartPosition* задано значение *Manual,* это свойство указывает исходное положение формы относительно верхнего левого угла экрана
MaximizeBox	Указывает, есть ли у формы кнопка MaximizeBox
MaximumSize	Устанавливает максимальный размер формы. Если задать этому свойству размер *0; 0,* у формы не будет верхнего ограничения размера
MinimizeBox	Указывает, есть ли у формы кнопка MinimizeBox
MinimumSize	Устанавливает минимальный размер формы, который пользователь может задать
Opacity	Устанавливает уровень непрозрачности или прозрачности формы от *0* до *100%*. Форма, непрозрачность которой составляет 100%, -полностью непрозрачна, а форма, имеющая 0% непрозрачности, — наоборот, полностью прозрачна

3.3.2 Размещение формы поверх пользовательского интерфейса

Порой вам нужно будет расположить одну форму поверх других форм в пользовательском интерфейсе. К примеру, вы разработали форму, в которой

представлена важная информация о выполнении программы, и хотите, чтобы она всегда была у пользователя перед глазами. Можно сделать так, чтобы форма всегда располагалась поверх пользовательского интерфейса, задав свойству *TopMost* значение *True.* Если *TopMost* присвоить значение *True,* форма всегда будет расположена поверх всех остальных форм, свойству *TopMost* которых задано значение *Fake,* являющееся значением по умолчанию. Обратите внимание, что если значение *True* свойства *TopMost* задано нескольким формам, они могут закрыть друг друга.

3.3.3 Прозрачность и непрозрачность форм

С помощью свойства *Opacity* можно создавать удивительные визуальные эффекты внутри формы. Свойство *Opacity* определяет степень непрозрачности формы. При настройке значения непрозрачности в окне Properties оно может варьироваться от 0 до 100 %. Непрозрачность в 100 % означает, что форма полностью непрозрачна (сплошная и видимая), а значение 0 % — что форма полностью прозрачна. Значения между 0 и 100 % делают форму частично прозрачной.

Кроме того, свойство *Opacity* можно задать в коде. При этом ему задается значение от 0 до 1, где 0 означает, что форма будет полностью прозрачной, а 1 — полностью непрозрачной.

```
aForm.Opacity =0.5;
```

Свойство *Opacity* может пригодиться, когда необходимо расположить одну форму на переднем плане, наблюдая при этом за работой формы на заднем плане, или же для создания интересных визуальных эффектов. Чаще всего элемент управления наследует прозрачность той формы, которая его содержит.

3.3.4 Создание непрямоугольных форм Windows

Возможно, для получения дополнительных визуальных эффектов вы захотите создать непрямоугольные формы, например овальную или форму в виде логотипа компании. Хотя создать непрямоугольную форму довольно просто, относительно окончательного вида и поведения формы существует несколько соображений.

Создать непрямоугольную форму можно, задав свойство *Region* формы в обработчике события *FormLoad.* Поскольку изменения в форме фактически происходят во время выполнения, вы не сможете их увидеть во время разработки. Таким образом, вам придется несколько раз запускать приложение и смотреть на форму, пока не настроите внешний вид и расположение элементов управления.

Свойство *Region* является экземпляром класса *System.Drawing.Region.* Данный класс представляет собой область экрана, которая является внутренней частью графической формы, определенной с помощью прямоугольников и графических путей. Самый простой способ создать непрямоугольную форму — это создать новый экземпляр класса *GraphicsPath,* а затем с его помощью создать новый класс *Region.* Вот простой пример кода:

```
System.Drawing.Drawing2D.GraphicsPath myPath = new
System.Drawing.Drawing2D.GraphicsPath();
// Добавление эллипса, вписанного в прямоугольную форму // заданной
ширины и высоты
myPath.AddEllipse(0, 0, this.Width, this.Height);
// Создание нового класса Region из GraphicsPath Region myRegion = new
Region(myPath);
// Присвоение свойству Region формы нового региона this.Region = myRegion;
```

Поскольку непрямоугольные формы будут иметь ограниченные границы (если таковые вообще будут), было бы неплохо задать свойству ***FormBorderStyle*** формы значение ***None.*** Так, если какие-либо части формы пересекут исходные границы прямоугольника формы, они не изменятся и не примут нежелательную форму. Однако если задать свойству ***FormBorderStyle*** значение ***None,*** пользователь не сможет изменить размеры, передвинуть или закрыть форму, а вам придется встроить эти возможности в свой проект.

3.4 Создание непрямоугольной формы:

1. В окне Properties задайте свойству ***FormBorderStyle*** значение ***None.***
2. Дважды щелкните форму в конструкторе, чтобы открыть обработчик события по умолчанию ***Form_Load.***
3. В обработчике события ***FormLoad*** создайте новый экземпляр класса ***Region,*** как показано в предыдущем примере.
4. Если нужно, добавьте в форму функциональность для закрытия, передвижения или изменения размеров, поскольку, возможно, пользователь не сможет получить доступ к границам или строке заголовка формы.
5. Задайте форму как стартовую и нажмите F5, чтобы увидеть форму Настройте по мере необходимости внешний вид и расположение элементов управления.

4 Задания для выполнения работы

Общая часть задания: написать Windows-приложение, заголовок главного окна, которое содержит Ф. И. О., группу и номер варианта. В программе должна быть предусмотрена обработка исключений, возникающих из-за ошибочного ввода пользователя

1. Вычислить сумму

$$\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!},$$

где n - натуральное число, x - действительное.

2. Вычислить сумму

$$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots$$

с точностью 0,01 на интервале [-1;1] с шагом 0,1, где x - действительное число.

3. Вычислить сумму

$$1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots$$

с точностью 0,01 на интервале [-2;1] с шагом 0,1, где x - действительное число.

4. Дана функция

$$y(x)=x+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots.$$

Определить число членов ряда этой функции для *x,* выбираемого из интервала [-1;1], при точности вычисления функции с 0,001 с шагом 0,0001.

5. Дана функция

$$y(x)=1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots.$$

Определить число членов ряда этой функции для *x,* выбираемого из интервала [-1;1], при изменении точности от 0,1 до 0,001 с шагом 0,0001.

6. Вычислить сумму

$$\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!},$$

где n - натуральное число, x — действительное.

7. Найти сумму целых положительных чисел из промежутка от a до b, кратных 4.

8. Найти сумму целых положительных чисел, больших 20, меньших 100, кратных 3 и заканчивающихся на 2, 4 или 8.

9. Дано натуральное число n. Получить все его натуральные делители.

10. Даны натуральные числа m и n. Получить все кратные им числа, меньшие m*n.

11. Сумма цифр трехзначного числа кратна 7, само число также делиться на 7. Найти все такие числа.

12. Дано натуральное число n. Определить является ли оно палиндромом (перевертышем), с учетом четырех цифр. Например, палиндромами являются следующие числа: 1111, 6116, 0440.

13. Даны натуральные числа n и k. Из чисел от n до k выбрать те, запись которых содержит ровно три одинаковые цифры. Например, числа 6766, 5444 содержат ровно три одинаковые цифры.

14. Среди четырехзначных чисел выбрать те, у которых все четыре цифры различны.

15. Дано четырехзначное число n. Выбросить из его записи цифры 0 и 5, оставив прежним порядок остальных цифр. Например, из числа 1509 должно получиться число 19.

16. Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведенная в n – ю степень, равна самому числу (например, $153=1^3+5^3+3^3$). Получить все трехзначные числа Армстронга.

17. Поменять местами первую и последнюю цифры числа.

18. Поменять порядок цифр числа на обратный. Например, было 12345, стало 54321.

19. Найти количество четных цифр целого положительного числа.

20. Найти самую большую цифру целого числа.

21. Найти сумму цифр целого числа, больших 5.

22. Сколько раз данная цифра k встречается в числе n.

23. Составить программу, проверяющую, является ли заданное натуральное число совершенным, то есть равным сумме своих положительных делителей, кроме самого этого числа.

24. Найти натуральное число от 1 до n с максимальной суммой делителей.