МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Методические указания

к выполнению лабораторных работ
"Разработка приложений с формами средствами visual C#"
для студентов специальности 1-40 03 01 «Искусственный интеллект»

БРЕСТ 2023

УДК 681.3 (075.8)
ББК с57

Целью методических указаний является методическая, информационная поддержка лабораторных работ по изучению средств языка visual C# системы программирования Microsoft Visual Studio в части разработки и реализации оконных приложений и их графических интерфейсов на базе объектной парадигмы, объектно-ориентированного программирования.

Составители: Муравьев Г. Л., к. т. н., доцент
Мухов С. В., к. т. н., доцент
Савицкий Ю. В., к. т. н., доцент
Крапивин Ю. Б., к. т. н., доцент
Хвещук В. И., к. т. н., доцент

Рецензент: доцент кафедры Электронно-вычислительные машины, системы и сети «Брестского государственного технического университета», к. т. н. Костюк Д. А.

Оглавление

Введение

Технология .NET-программирования используется на платформе Microsoft.NET, включающей набор средств .NET Framework (библиотеки классов, используемые для разработки .NET-приложений различной специализации, и виртуальную машину для их исполнения), специализированные серверы и web-сервисы и инструментальную среду (систему программирования) Microsoft Visual Studio (VS). При разработке .NET-приложений, выполняемых с помощью виртуальной машины, используют следующие термины.

Решение (solution) – контейнер, содержащий один и более проектов. При создании нового проекта в VS автоматически создается новое решение (папка и файлы, включая файл описания контейнера). Для разработки оконных приложений Visual C# (приложений на базе форм) в VS создаются проекты Windows типа "Приложение Windows Forms". Проект – контейнер с набором данных, описаниями приложения, включая исходные коды, которые при компиляции собираются в единый файл – сборку. Сборка (assembly) – готовое приложение, созданное в среде .NET (файл с расширением exe, dll), включающее все необходимые для выполнения данные, метаданные. Манифест – описание, метаданные сборки (версия, название сборки, ссылки на другие сборки и т. д.). Среда VS дает возможность пользователю выбрать подходящий каркас (шаблон, заготовку) сборки и поддерживает работу с контейнерами типа решение и проект для хранения частей приложения в процессе разработки приложений. Форма – базовый компонент графического интерфейса приложения (ГИП). Визуально это окно, аналогичное окнам windows-приложений C++ (API-, MFC-приложения). Форма может использоваться в качестве масштабируемого окна с клиентской областью (КО) или диалогового окна. Функциональность форм обеспечивается библиотечным классом Form и пользовательскими классами, производными от него. С работой формы связано множество событий, приводящих к соответствующим сообщениям. Так, при запуске формы посылаются сообщения типа Load, Shown, Activate и др., при завершении работы формы – FormClosed. В процессе использования – сообщения типа Paint, Activate, Deactivate, сообщения о действиях пользователя и др. Приложение может обрабатывать события формы и элементов управления (ЭУ), расположенных в форме, определяя тем самым ее поведение.

Формой можно управлять методами пользовательского класса, унаследованного от класса Form, методами класса приложение Application. Например, метод Close – закрывает форму, Activate – активизирует форму, Hide – скрывает форму, Invalidate – вызывает перерисовку КО, Show – показывает форму (немодальное окно), ShowDialog – отображает форму в виде модального диалогового окна и т. д. Форма (как основное окно) активизируется методом приложения Application.Run(new Form1()). Выполнение формы завершается: методом приложения Exit одновременно с завершением его работы как Application.Exit(); кнопкой закрытия формы либо методом формы Close().

Для реализации пользовательского приложения в VS сначала выбирают и генерируют подходящий типовой каркас (шаблон) приложения (ТКП), который затем до программируют в соответствии с функциональными требованиями к приложению. В процессе разработки приложений в среде VS потребуется ряд окон. Это окна для работы с кодом, с визуальным представлением компонентов ГИП (например, Форм в виде кода соответствующего класса или в виде изображения в Конструкторе форм – как в визуальном Редакторе графических ресурсов в Windows). В последнем случае надо подключить и Панель элементов (ЭУ). Также при настройке среды следует активизировать окна: 1 – обозреватель решений (отображает файловый состав проекта); 2 – классы; 3 – окно свойств. Можно добавить окно: 4 – ресурсы и др. При необходимости просмотра фраг-

мента кода в соответствующем окне можно выбрать нужный файл в Обозревателе решений или нужный класс в Окне классов и т. д.

1 Создание каркаса оконного приложения (ТКП)

► **Задание 1**. *Создайте в Visual Studio (VS) автоматически генерируемый типовой каркас оконного приложения (ТКП). Используйте настройки – язык "visual C#", ОС "Windows", тип (архитектура) приложения "Рабочий стол", тип проекта – "Приложение Windows Forms (.NET Framework)".*

Для этого (на примере VS 2022):

1. Запустите Visual Studio. Появится главное окно ("Visual Studio"), в котором можно открыть недавно созданные проекты ("последние"), можно создать новый проект ("Начало работы. Создание проекта"). Выполните "Создание проекта".

2. Появится окно ("Создание проекта"), в котором можно запустить создание нужного проекта с настройками, указанными выше в задании. Либо, если нужных средств (шаблона приложения, соответствующего мастера) в системе нет, то можно выполнить их установку ("Установка других средств и компонентов").

Выберите "Приложение Windows Forms (.NET Framework)", выполните – "Далее".

3. Появится окно ("Настроить новый проект"), в котором надо задать Имя, Расположение проекта (а при желании и другие параметры). Выполните "Создать". Будет создан ТКП с одной формой (файлы решения-проекта в указанном месте файловой системы), появится рабочее окно VS. ТКП можно запустить, а файлы проекта ТКП посмотреть!

4. Настройте среду программирования VS для работы с текущим проектом.

4.1. К окнам отображения кодов, результатов отладки, выполнения добавьте сервисные Окна – Обозреватель решений, Классы, Окно Свойств (используйте пункт главного меню ГМ – Вид), Ресурсы (ГМ – Вид – Другие Окна).

4.2. Убедитесь (ГМ – Сборка/Построение – Диспетчер конфигураций), что выбрана отладочная конфигурация проекта Debug (или Release, если отладка завершена).

4.3. Ознакомьтесь с настройками проекта приложения (ГМ – Проект – Свойства), которыми можно доопределить вид приложения, версию используемого .NET Framework и др.

5. Создайте приложение (ТКП заданного типа). Для этого выполните "Пуск" либо ГМ-Отладка – Запуск без отладки или Начать отладку.

6. Изучите состав решения (проекта) созданного приложения – состав папок и файлов. Нарисуйте диаграмму классов приложения. Результаты приведите в Отчете.

6.1. Состав основных файлов решения приложения (как он представлен в Обозревателе решений) показан на рисунке 1 (может меняться в зависимости от версии VS). Здесь в первую очередь отображен перечень исходных (source) файлов, содержимое которых можно отобразить в отдельных окнах просмотра и редактирования кода. Это файл Program.cs с кодом главного класса приложения (здесь Program) с методом Main. И файлы Form1.cs, Form1.Designer.cs, описывающие класс Form1, унаследованный от базового Form. Из них первый файл будет отображать часть класса – его методы, включая обработчики, второй – часть класса с настройками самой формы (описания состава, дизайна формы), которую видит пользователь.

Полный состав файлов решения (можно увидеть в Обозревателе решений в развернутом виде) включает ряд служебных компонентов. Так файл AssemblyInfo.cs задает атрибуты – общие сведения о сборке проекта. Например, номер версии сборки, дополнительный номер, номер построения, номер редакции сборки и др.

6.2. Найдите папку решения в файловой системе и проанализируйте ее состав. Там есть файл типа Решение – Solution (здесь f_001_skeleton.sln) – аналог файла проекта. Для запуска среды для работы с кодом уже существующего приложения надо открывать этот файл. Там же расположено несколько папок, например, как показано на рисунке 2. Папки obj и bin в зависимости от выбранной конфигурации проекта для построения решения (Debug или Release) содержат файлы, представленные на рисунке 3.

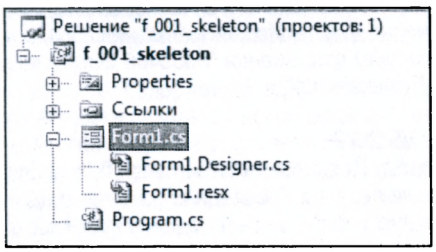

Рисунок 1 – Обозреватель решений (состав файлов решения)

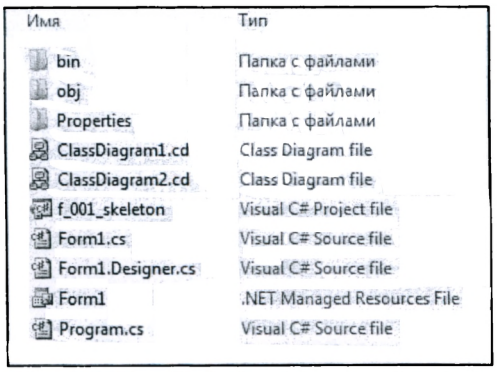

Рисунок 2 – Обозреватель решений (состав файлов решения)

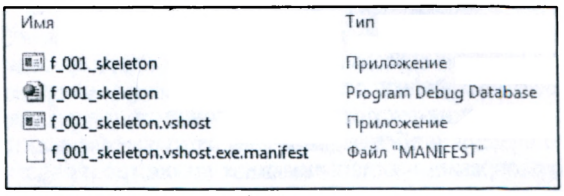

Рисунок 3 – Примерный состав папки Release

Таким образом: 1 – нет деления файлов приложения по признаку интерфейс либо реализация как в оконных приложениях visual C++; 2 – в тоже время файлы приложения разделены на те, где непосредственно описывается код – бизнес-логика приложения и те, где текст генерируется системой, хранятся соответствующие настройки. Это, например, файлы, описывающие классы форм; 3 – описания приложения, хранящиеся в указанных файлах, объединены в отдельные пространства имен (здесь это пространство namespace f_001_skeleton, содержащее непосредственно исходные коды, и namespace

f_001_skeleton.Properties). Примерный состав классов с разбиением на пространства имен – как он отображается в Окне классов – приведен на рисунке 4.

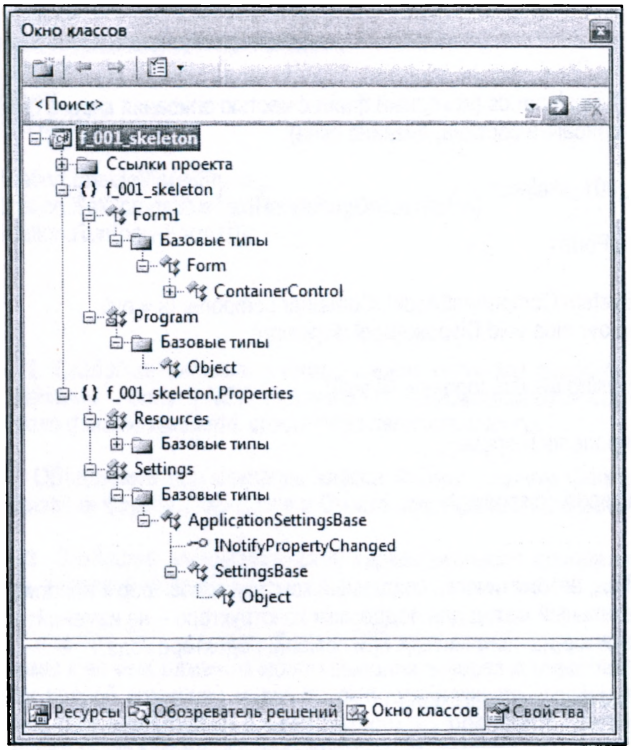

Рисунок 4 – Окно классов (состав классов)

Ниже представлены фрагменты описания кодов приложения (ТКП). Это файл Form1.cs (исходный файл с частью описания класса формы Form1)

```
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 f_001_skeleton
{
   public partial class Form1 : Form
   {
      public Form1( )
```

```
        {
            InitializeComponent( );
        }
    }
}
```

И файл Form1.Designer.cs (исходный файл с частью описания класса формы Form1 – ее "настроек" – описание состава, дизайна окна)

```
namespace f_001_skeleton
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows
        /// Обязательный метод для поддержки конструктора – не изменяйте
        /// содержимое данного метода при помощи редактора кода.

        private void InitializeComponent( )
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Text = "Form1";
        }
        #endregion
    }
}
```

Файл Program.cs (исходный файл с описанием главного класса приложения, здесь Program)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace f_001_skeleton
```

```
{
    internal static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

► **Задание 2**. *Создайте автоматический каркас оконного приложения – проект "Приложение Windows Forms (.NET Framework)". Минимизируйте состав каркаса, сохранив иерархию файлов проекта, созданного автоматически.*

Для этого: 1. Объедините оба описания класса Формы в одном файле Form1.cs. 2. Удалите файл дизайна Формы. Приведите в Отчете новое описание класса Form1.

► **Задание 3**. *Создайте автоматический каркас оконного приложения с элементом управления – кнопкой (button).*

Для этого: 1 Создайте ТКП, откройте Форму ТКП в режиме Конструктора форм (переход от кода формы к ее изображению можно выполнить через контекстное меню, вызываемое щелчком правой клавишей мыши по окну отображения и далее выбором действия – Перейти к коду или Открыть в конструкторе). 2 Подключите Панель ЭУ (используйте пункт главного меню ГМ–Вид-Панель элементов). 3 Добавьте в форму (в Конструкторе форм) элемент управления – кнопку (button) из числа Стандартных ЭУ. 4 Запустите приложение. Затем проанализируйте код каркаса и зафиксируйте в Отчете изменения, внесенные в код мастером автоматически. 5 Удалите кнопку, снова проанализируйте и зафиксируйте в Отчете изменения в коде.

2. Вывод данных в клиентскую область (КО) формы. Окно сообщений

При выводе строк и графики используются такие графические компоненты, как шрифт, цвет, кисть, перо и др.:

1. Шрифт (Font) – специальный класс для задания атрибутов: – НазваниеШрифта; – Кегль; – при необходимости Стиль (FontStyle). Создание объекта

System.Drawing.Font мойШрифт = new System.Drawing.Font (НазваниеШрифта, Кегль, [FontStyle.Стиль]) .

Например, временный объект new System.Drawing.Font("Verdana",15); объекты

```
System.Drawing.Font myFont2 = new System.Drawing.Font("Arrial",12);
Font  myFont3 = new Font("Arial",30);
Font  myFont4 = new Font("Arial",30, FontStyle.Italic ).
```

2. Цвет Color – структура для задания цвета пера, кисти и т. д. Цвет задается как System.Drawing.Color.НазваниеЦвета; System.Drawing.Color.FromArgb(Значение); или System.Drawing.Color.FromArgb(Значение1, Значение2, Значение3).

При необходимости можно создать и далее использовать переменную, например

```
System.Drawing.Color   myColor1=(System.Drawing.Color.Black);
System.Drawing.Color   myColor2=(System.Drawing.Color.FromArgb(125));
System.Drawing.Color   myColor3=(Color.FromArgb(0,0,255)).
```

3. Кисть SolidBrush – класс для определения атрибутов кисти, включая цвет т. д. Соответствующий объект создается как System.Drawing.SolidBrush мояКисть = new SolidBrush (Цвет). Например,

```
System.Drawing.SolidBrush  myBrush1 = new SolidBrush (Color.Black);
System.Drawing.SolidBrush  myBrush2 = new SolidBrush (myColor3);
myBrush1.Color = Color.Red;
myBrush1.Color = (Color.Red).
```

4. Перо Pen – специальный класс для определения атрибутов пера: Цвет, Толщина, Стиль. Объект создается как System.Drawing.Pen моеПеро = new Pen(Цвет, [Толщина]). Например,

```
System.Drawing.Pen  myPen1 = new Pen (Color.Black);
System.Drawing.Pen  myPen2 = new Pen (Color.Black, 5);
System.Drawing.Pen  myPen3 = new Pen (myBrush2, 4);
System.Drawing.Pen  myPen4 = new Pen (new SolidBrush (myColor3), 4);
myPen3.Width = 5.
```

5. Контекст устройства (КУ), который можно получить: – в самой форме и ее частях (методах, обработчиках) как System.Drawing.Graphics gr = this.CreateGraphics( ) или Graphics gr = this.CreateGraphics( ); – в обработчике события перерисовки (Paint) формы private void Form1_Paint (object sender, PaintEventArgs e) через аргумент обработчика e.Graphics.

С помощью КУ можно управлять выводом в клиентскую область (КО) формы, вызывая необходимые атрибуты и методы. Как правило, вывод информации в КО осуществляется как раз в обработчике события ее перерисовки.

6. Вывод информации в КО формы.

Вывод данных (в строковом формате) осуществляется рядом перегруженных методов DrawString с прототипом DrawString (ВыводимаяСтрока, Шрифт, Кисть, X, У); . Выводимая строка имеет тип String. Например,

```
gr.DrawString("СтрокаВывода",myFont,myBrush1,35,200);
gr.DrawString("СтрокаВывода",myFont,myBrush1,35,250).
```

Для вывода графики (фигур) используются специальные методы. Например,

```
DrawLine( Перо, float X1, float Y1, float X2, float Y2 );
DrawLine( Перо, int X1, int Y1, int X2, int Y2 );
DrawLine(new Pen(Color.Brown),1,1,25,56);
Pen myPen = new Pen(new SolidBrush(Color.DarkViolet), 2);
gr.DrawEllipse(myPen, 15, 5, 60, 70);
Rectangle r = new Rectangle(147, 17, 23, 23);
gr.DrawRectangle(myPen, r).
```

► **Задание 4.** *Обеспечьте вывод данных по образцу на рисунке 5. Данные вывести как инициализирующие для окна, т. е. в конструкторе формы.*

СПРАВКА. Для отображения данных в окне формы необходимо до первого использования метода Draw… выполнить метод формы Show( ) или this.Show( ).

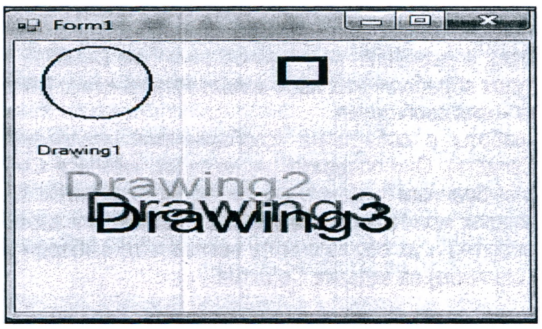

Рисунок 5 – Образец вывода

Для вывода кратких сообщений используют окно сообщений, управляемое методом Show класса MessageBox. Прототип метода – Show("СтрокаВывода", "ЗаголовокОкна", СоставКнопок,ВидПиктограммыОкна) или упрощенно System.Windows.Forms. MessageBox.Show("СтрокаВывода"). Метод обеспечивает визуализацию окна сообщения с набором кнопок и пиктограмм. Для настройки состава кнопок окна используйте MessageBoxButtons.<НаборКнопок>, тип пиктограммы окна задавайте как MessageBoxIcon. <Пиктограмма>. Для анализа возвращаемого окном результата – кода нажатой кнопки используйте поименованное значение DialogResult.<ТипНажатойКнопки>. Например, MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation, DialogResult.OK.

3. Сообщения и их обработка

Во время работы оконного приложения могут происходить различные события, приводящие к сообщениям, посылаемым приложению. При наличии соответствующего обработчика выполняются запланированные пользователем действия. Так, при запуске оконного приложения инициируются сообщения типа Load, Paint, Shown, Activate. В процессе работы при смене активного компонента (например, формы) посылаются сообщения Activate, Deactivate, при необходимости перерисовки – Paint, а также другие сообщения, сообщения ввода. При закрытии формы инициируется сообщение Closed. Чтобы приложение реагировало на событие, необходимо создать каркас метода для его обра-

ботки – обработчика, "подписать" его на сообщение (связать обработчик с конкретным типом события), описать сам код обработчика. При необходимости можно снять "подписку" в свойствах формы – но сам обработчик при этом останется в коде формы.

Обработчики событий добавляются: – автоматически выбором необходимого события на одноименной вкладке События в окне Свойства текущей Формы; – автоматически двойным щелчком по компоненту ГИП "мышью" можно добавить обработчик того события, которое установлено для этого компонента по умолчанию (например, для Формы – это обработчик события Load, для кнопки – Click и т. д.); – при необходимости "вручную", то есть программирования кода. При автоматическом добавлении обработчика каркас обработчика и "подписка" генерируются системой. Код обработчика пишется пользователем. Сам обработчик хранится в основном описании класса формы (Form1.cs), команда подписки хранится во второй части описания класса формы (Form1.Designer.cs).

► **Задание 5.** *Создайте каркас приложения. Добавьте обработчики сообщений загрузки, вывода, закрытия формы – Load, FormClosed, Shown, перерисовки Paint.*

В коде обработчика Paint следует наращивать переменную, фиксирующую число или номер перерисовки, и выводить ее значение методом DrawString (см. параграф 2, пункты 5, 6), а в других обработчиках надо использовать класс MessageBox, а в качестве строки вывода – имя сообщения.

СПРАВКА. Для работы с событиями (сообщениями) используйте Окно Свойств (ГМ – Вид – Окно Свойств). Оно содержит ряд вкладок, включая События и Свойства, ассоциированные с выбранным в Конструкторе форм элементом ГИП (формой, ЭУ). Содержимое этих вкладок может быть выведено в алфавитном порядке или по категориям! Добавление событий и их обработчиков выполняйте выбором соответствующего элемента (двойным щелчком) на вкладке События.

Для этого:

1 Создайте ТКП. Убедитесь, что Окно свойств подключено (или подключите его).

2 Откройте Форму ТКП в Конструкторе (для визуальной работы). В Окне свойства на вкладке Свойства найдите заголовок Формы (например, свойство Text – "Form1") и измените его по своему усмотрению, ее внутреннее имя (например, свойство Name – Form1), на вкладке События найдите подключенные к Форме (например, событие Load с обработчиком Form1_Load).

3. На вкладке События последовательно найдите и включите чувствительность ко всем заданным событиям (двойным щелчком), добавив затем необходимый код в их обработчики. При каждом добавлении запускайте приложение, чтобы убедиться в его работоспособности.

4 Запустите приложение и зафиксируйте в Отчете порядок появления сообщений. Найдите в коде приложения команды "подписки" на соответствующее сообщение (событие) и зафиксируйте их в Отчете.

5 Удалите обработчик какого-либо сообщения и затем восстановите его.

► **Задание 6.** *Повторите Задание 4. Обеспечьте вывод тех же данных, но с учетом поддержки перерисовки. При запуске формы выводить соответствующее сообщение методом MessageBox.Show("Запуск формы!!! ").*

Для этого: 1. После создания каркаса откройте Форму в редакторе – Конструктор форм.

2. В Окне Свойств выберите вкладку События, а в ней – событие Paint (например, двойным щелчком). В код Формы автоматически добавится каркас обработчика.

3. В обработчике используйте контекст устройства e.Graphics, передаваемый в него через параметр PaintEventArgs e, и реализуйте вывод данных как в Задании 4.

► **Задание 7.** *Обеспечьте вывод в КО формы набора линий (не менее 6), образующих плоскую фигуру – многоугольник. Многоугольник выводить по заданному константному набору координат точек с поддержкой перерисовки. Для хранения координат использовать массив (на базе класса Point, т.е. как массив объектов класса Point). Вывод каждой линии выполнять методом DrawLine.*

► **Задание 8.** *Создайте каркас приложения. Добавьте обработчик сообщений нажатия клавиш мыши MouseClick (или MouseDoubleClick, MouseDown) с анализом и выводом методом MessageBox координат курсора мыши и типа нажатой клавиши.*

СПРАВКА. Координаты указателя мыши передаются в параметре обработчика MouseEventArgs e как e.X и e.Y. Для перевода координат в строковый формат можно использовать метод ToString( ), например, как (e.X).ToString( ).

Информация о нажатой клавише содержится в параметре обработчика e.Button, а для ее анализа используйте поименованные значения типа System.Windows.Forms. MouseButtons.Клавиша (где Клавиша – Left, Right, Middle, None).

Координаты мыши относительно экрана определяются как атрибуты формы this.MousePosition.X и this.MousePosition.Y.

► **Задание 9.** *Добавьте обработчик сообщений KeyPress о нажатии клавиш на клавиатуре с анализом и выводом методом MessageBox названия клавиши. Обеспечьте завершение работы приложения по нажатии определенной клавиши методом Application.Exit .*

СПРАВКА. Код нажатой клавиши передается в параметре обработчика KeyPressEventArgs e как e.KeyChar.

► **Задание 10.** *Создайте каркас приложения (по указанию преподавателя повторите предыдущие пункты 7–9). Добавьте обработчики сообщений активизации Activated и деактивизации Deactivate формы с выводом в КО значений двух переменных A и DA в соответствии с образцом на рисунке 6. Опишите и приведите в Отчете диаграмму классов приложения и сам класс формы.*

СПРАВКА. Здесь переменная A устанавливается в 1 при активизации, DA – при деактивизации формы. Начальные значения соответственно – 0 и 1. Метод MessageBox.Show в этих обработчиках и в обработчике перерисовки не использовать! Значения выводить в обработчике Paint, который активизируется методом формы Invalidate. Фрагмент кода для события Activated выглядит, например, как A =1; DA =0; Invalidate( ).

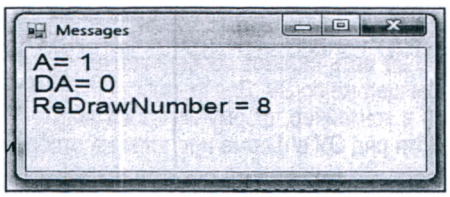

Рисунок 6 – Образец вывода

► **Задание 11.** *Разработать диаграммы состояний (привести в Отчете) и соответствующие приложения для рисования плоских фигур – многоугольников – отрезками прямых в вариантах, представленных ниже.*

11.1. *Координаты вершин или линий задаются левой клавишей мыши, при вводе новой координаты изображение заданной фигуры выводится (обновляется) сразу. При этом первая и последняя из заданных точек (координат) также соединяются прямой линией.*

11.2. *Координаты задаются левой клавишей мыши, окончательное изображение фигуры выводится правой клавишей мыши (сообщения MouseClick или MouseDoubleClick). При этом первая и последняя из заданных точек (координат) также соединяются прямой линией.*

11.3. *Модифицируйте приложение (Задание 11.2) – для работы с фигурой используйте пользовательские классы (Точка, Линия, Фигура, а при сохранении данных дополнительно – Файл, т. е. класс поддержки файлового ввода-вывода). Привести в Отчете диаграмму классов.*

11.4. *Модифицируйте приложение (Задание 11.2 или 11.3) – для хранения координат используйте динамические структуры C#.*

4 Элементы управления (ЭУ)

Элементы управления (ЭУ) являются производными от класса Object и сами образуют фрагменты иерархии классов. Все ЭУ обладают как индивидуальными, так и такими общими свойствами, атрибутами как: Anchor – определяет "поведение" размеров, положение ЭУ при изменении размеров его контейнера; Bottom, Left – координаты верхнего левого угла ЭУ; Height, Width – высота и ширина ЭУ; Dock – характер прижатия ЭУ к границе контейнера; Enabled – определяет, может ли ЭУ воспринимать действия пользователя – события; TabStop, TabIndex – определяют соответственно чувствительность ЭУ к нажатию клавиши табуляции и номер обхода ЭУ при использовании этой клавиши; Name, Text – название ЭУ (для использования в коде) и ассоциируемый заголовок и др.

С ЭУ связаны события (сообщения), в том числе общие для всех или большинства ЭУ. Это сообщения Click, DoubleClick (не для кнопок), иногда Enter. Это события KeyDown, ReyUp, KeyPressed (аналоги WM_CHAR), MouseDown, MouseUp манипуляции с мышью. Это события MouseEnter, MouseLeave, MouseMove, MouseHover, связанные с определенным характером перемещения и положением курсора мыши (входом и выходом за видимые границы ЭУ, при наведении указателя мыши на компонент, при неподвижности в пределах ЭУ в течение некоторого времени).

События Validate, Validating – для организации корректного пользовательского ввода – как контроля ввода на отдельном ЭУ, так и контроля по группе задействованных для ввода ЭУ. Последний случай означает, что кнопка итогового ввода (типа OK) разблокируется только, если на всех ЭУ группы будет зафиксирован корректный ввод.

Кроме этого у каждого ЭУ есть базовое сообщение, его обработчик подключается автоматически при двойном щелчке по ЭУ. Для формы это - Load, для кнопки – Click и т. д.

Сами ЭУ включаются в контейнер, форму пользователем, как правило, визуально в Конструкторе формы. Хотя ряд ЭУ в форме при этом не отображается. Если ЭУ добавлен – перенесен из панели ЭУ в разрабатываемую форму, то он входит в состав коллекции ЭУ формы (как this.Controls.Add(this.<ЭУ>).

В § 1 в Задании 3 это было показано на примере вставки в форму кнопка (button). При этом автоматически: 1 – в класс формы добавлен атрибут-ссылка private Button button1;

2 – создан объект this.button1 = new System.Windows.Forms.Button() в конструкторе формы;
3 – выполнена инициализация в соответствии со свойствами кнопки, например,

```
// button1
this.button1.Location = new System.Drawing.Point(216, 21);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(70, 43);
this.button1.TabIndex = 0;
this.button1.Text = "button1";
this.button1.UseVisualStyleBackColor = true;
```

4 – объект будет добавлен в состав коллекции ЭУ формы командой this.Controls.Add (this.button1).

Фрагмент итогового кода приложения (файл Form1.Designer.cs) представлен ниже

```
partial class Form1
{
   private Button button1;
        ...
   #region Код, автоматически созданный конструктором форм Windows
   /// для поддержки конструктора

   private void InitializeComponent()
   {
      this.button1 = new System.Windows.Forms.Button();
      this.SuspendLayout();
      // button1
      this.button1.Location = new System.Drawing.Point(216, 21);
      this.button1.Name = "button1";
      this.button1.Size = new System.Drawing.Size(70, 43);
      this.button1.TabIndex = 0;
      this.button1.Text = "button1";
      this.button1.UseVisualStyleBackColor = true;
      // Form1
      this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
      this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
      this.ClientSize = new System.Drawing.Size(296, 262);
      this.Controls.Add(this.button1);
      this.Name = "Form1";
      this.Text = "EXAMPLE";
      this.ResumeLayout(false);
   }
   #endregion
}
```

► **Задание 12.** *Создать приложение с двумя кнопками. По нажатию первой кнопки выводить сообщение (MessageBox с кнопкой OK) о ее выборе.*

По нажатию второй кнопки с названием Выход выводить приглашение о завершении приложения (MessageBox с кнопками OK и Cancel). И, если пользователь отвечает OK, то завершать приложение методом Application.Exit.

► **Задание 13.** *Разработайте приложение для работы с числами и строками с интерфейсом, приведенным на рисунке 7. Спроектируйте диаграммы прецедентов, классов, автоматов (приведите в Отчете).*

Для этого последовательно выполните указанные группы действий:

А. Создайте ТКП. Реализуйте интерфейсную форму (в роли диалогового окна) указанного вида (добавьте необходимые ЭУ). Для надписей используйте ЭУ Label. Для группирования элементов работы со списком – ЭУ GroupBox из группы "контейнеры".

Б. Реализуйте поддержку работы с левой группой ЭУ – двумя окнами редактирования (TextBox) и кнопкой (Button) с именем Enter. Число из верхнего окна редактирования по нажатию кнопки должно вводиться в приложение, из него извлекается квадратный корень, результат выводится в нижнем окне. Содержимое верхнего окна при этом очищается. При отсутствии числа ввода вычисление не производится.

СПРАВКА. Используйте методы и атрибуты для управления окном редактирования, обработки числа, для преобразования типов данных, приведенные ниже

Math.Sqrt(X) , this.textBox1.Text.Length , this.textBox2.Clear( ) ,
this.textBox2.Text = X.ToString( ), System.Convert.ToSingle(Строка).

В. Реализуйте поддержку работы с правой группой ЭУ – со списком (List), окнами и кнопками для управления списком и его содержимым. Строки списка хранятся в коллекции Items как в массиве (this.listBox1.Items[Индекс]).

Для этого: 1. Инициализируйте ЭУ список значениями. В Редакторе форм для списка это выглядит как добавление строк в Коллекцию его свойства Items (рисунок 8).

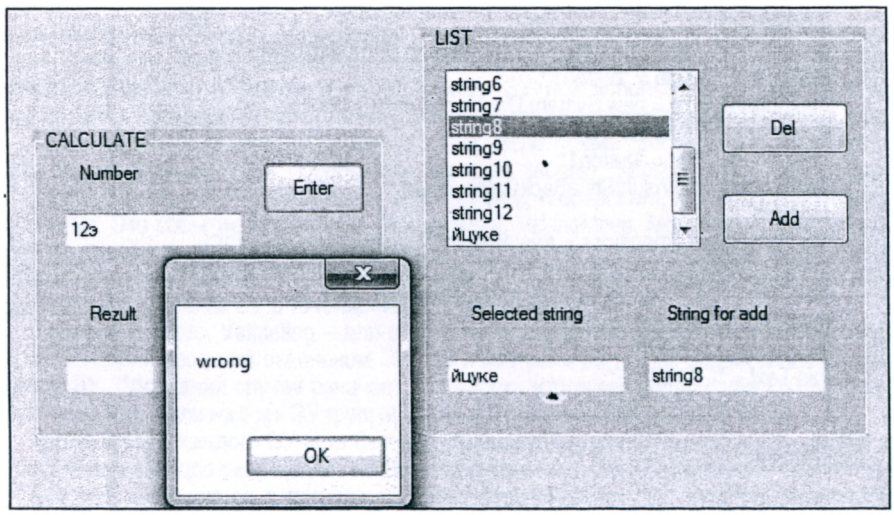

Рисунок 7 – Образец интерфейсной формы

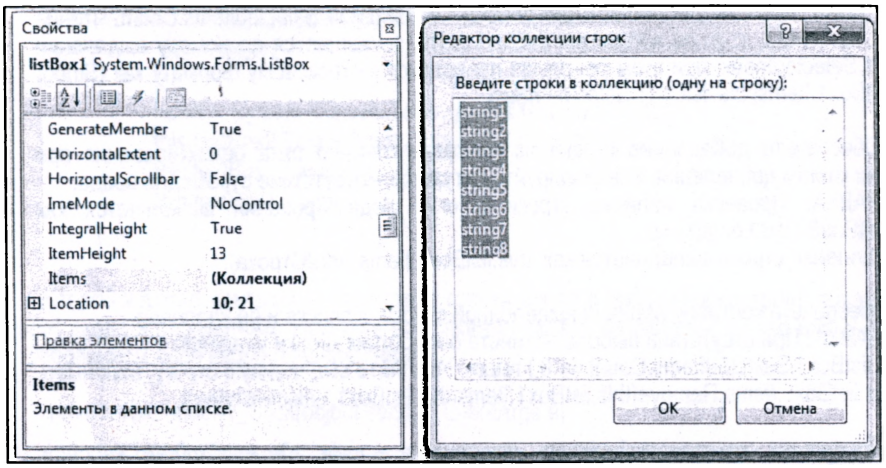

Рисунок 8 – Редактирование списка

При этом автоматически генерируется код в методе формы InitializeComponent. Результаты создания и инициализации списка (фрагмент кода) представлены ниже

```
        ...
private System.Windows.Forms.ListBox listBox1;
        ...
private void InitializeComponent()
{
        ...
  this.listBox1 = new System.Windows.Forms.ListBox();
        ...
        // listBox1
        this.listBox1.FormattingEnabled = true;
        this.listBox1.Items.AddRange(new object[] {
        "string1", "string2", "string3",  "string4",  "string5",  "string6",
        "string7",  "string8",  "string9", "string10", "string11",  "string12"});
        this.listBox1.Location = new System.Drawing.Point(15, 26);
        this.listBox1.Name = "listBox1";
        this.listBox1.Size = new System.Drawing.Size(158, 108);
        this.listBox1.TabIndex = 11;
        ...
}
```

Можно выполнить эту же операцию в конструкторе формы или в обработчике события – загрузка формы, используя для этого метод списка – this.listBox1.Items.Add(Строка).

2. Обеспечьте контроль события списка – SelectedIndexChanged – выводите сообщение о смене выбора, саму выбранную строку выводите в соответствующем окне редактирования. Контролируйте ситуацию – отсутствие выбора в списке (отсутствие выделения)!

СПРАВКА. Число выделенных строк списка – this.listBox1.SelectedItems.Count. Чтение строки из списка, т. е. из коллекции listBox1.Items выполняется по номеру выделения listBox1.SelectedIndex, при этом результат приводится к строковому формату как (String) (this.listBox1.Items[this.listBox1.SelectedIndex] .

3. Обеспечьте добавление строки из соответствующего окна редактирования при нажатии кнопки добавления. Контролируйте ситуацию – отсутствие строки для ввода.

СПРАВКА. Проверка наличия строки в окне редактирования выполняется как This.textBox3.Text.Length != 0.

Добавление строки выполняется как this.listBox1.Items.Add(Строка).

4. Обеспечьте контроль наличия выделенной строки в списке и ее удаление.

СПРАВКА. При отсутствии выбора покиньте обработчик кнопки удаления if (this.listBox1.SelectedItems.Count == 0) return. Удаляйте выделенную строку, например, как this.listBox1.Items.Remove(this.listBox1.Items[this.listBox1.SelectedIndex]).

Г. Проверьте отказоустойчивость приложения – там, где необходимо вводить числовые данные задайте вместо чисел недопустимые символы!

► **Задание 14.** *Создайте приложение, аналогичное предыдущему, но для работы только с левой группой ЭУ. Модифицируйте работу с левой группой ЭУ. Для этого используйте оба варианта контроля ввода некорректных данных, описанных ниже.*

Здесь это ввод нечисловых и отрицательных числовых данных, что касается работы обработчика кнопки Enter и содержимого textBox1 с названием Number.

СПРАВКА. Для обеспечения ввода корректного данного (в Задании 14 – чисел) можно применить две стратегии. Первая – после ввода данного в окне редактирования произвести его разбор с анализом на соответствие необходимому числовому типу (например, методом System.Single.TryParse). Если данное некорректно, то вывести сообщение и произвести очистку поля ввода для повтора.

Вторая – осуществлять поразрядный контроль вводимого данного (числа) в обработчике KeyPress для окна ввода (здесь textBox1) прямо на этапе ввода в окне редактирования.

Первый вариант контроля – отсутствие строки ввода в textBox1 и ввод не числового данного с последующим требованием пере ввода приведен ниже.

```
private void button1_Click(object sender, EventArgs e)
    {
        float newX;
        if ( this.textBox1.Text.Length == 0  )
        {
                this.textBox2.Clear( );
                return;
        };
        bool IsNumber = System.Single.TryParse(this.textBox1.Text,out newX);
        if ( ! IsNumber )
        {
                MessageBox.Show("Invalid value");
```

```
            this.textBox1.Clear( );
            return;
        };
        newX = System.Convert.ToSingle(this.textBox1.Text);
        this.textBox2.Text = (Math.Sqrt( newX )).ToString();
        this.textBox1.TabIndex = 0;
        this.textBox1.Clear( );
    }
```

Второй вариант контроля – отсутствие строки ввода в textBox1 и контроль поразрядного ввода в обработчике KeyPress для textBox1 (здесь разрешены только цифры) – с требованием пере ввода приведен ниже.

```
private void button1_Click(object sender, EventArgs e)
    {
        float newX;
        if ( this.textBox1.Text.Length == 0 )
        {
            this.textBox2.Clear( );
            return;
        };
        newX = System.Convert.ToSingle(this.textBox1.Text);
        this.textBox2.Text = (Math. Sqrt ( newX )).ToString( );
        this.textBox1.TabIndex = 0;
        this.textBox1.Clear( );
    }

private void textBox1_KeyPress (object sender, KeyPressEventArgs e)
    {
      if ( ( Char.IsDigit( e.KeyChar ) == true)  ) return;
      if ( e.KeyChar == Char( Keys.Back )  ) return;
     MessageBox::Show("Invalid value");
      e.Handled = true;
    }
```

или

```
private void textBox1_KeyPress (object sender, KeyPressEventArgs e)
    {
     if (  e.KeyChar >= 48 &&  e.KeyChar <= 57  ) return;
     MessageBox::Show("Invalid value");
      e.Handled = true;
    } .
```

5 Меню, стандартные формы, ЭУ RichTextBox

Рассматривается использование меню в составе формы. Чтобы создать меню: 1. Из панели элементов перенесите в Форму ЭУ menuStrip. Появится поле ввода первого пункта меню – введите его название. 2. Отредактируйте меню в соответствие с его конфигурацией. При выделении добавленного пункта меню новый подпункт задается выбором в поле ввода "Вводите здесь", расположенным справа или внизу от него. 3. Добавьте обработчики конечных пунктов меню (MenuItem) – двойным щелчком по пункту или задайте в окне свойств для выбранного пункта меню чувствительность к событию Click.

► **Задание 15.** *Создайте приложение, содержащее меню с системой обработчиков. Структура меню (как она выглядит в Конструкторе форм) показана на рисунке 9 (включает 5 пунктов, часть из них – конечные).*

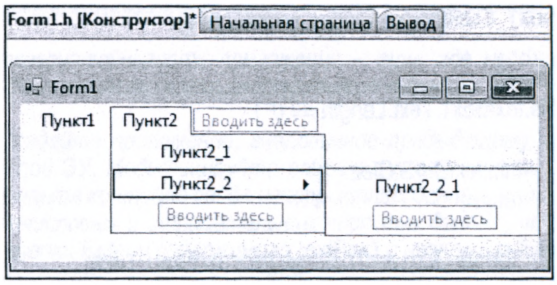

Рисунок 9 – Редактирование меню

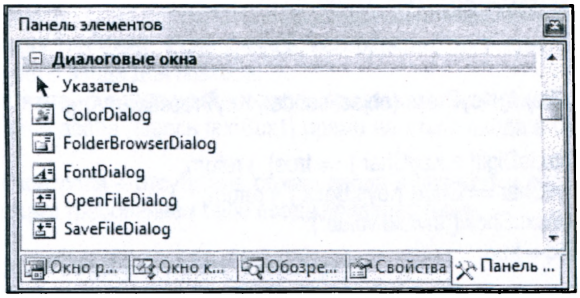

Рисунок 10 – Добавление стандартных окон

Для этого: 1. Создайте ТКП. 2. Добавьте ЭУ menuStrip. 3. Отредактируйте меню (см. рисунок). 4. Настройте свойства пунктов (задайте их имена – свойство Text). 5. Добавьте обработчики с выводом сообщений о выборе пункта меню.

► **Задание 16.** *Создайте приложение с меню, состоящим из одного пункта File и трех подпунктов в нем – Open, SaveAs, Exit.*

Подключите к основной форме стандартные ЭУ для работы с файлами – диалоговые окна (ДО) типа OpenFileDialog и SaveFileDialog. Каждое из них активизируется в обработчике соответствующего пункта меню.

ктивизируется в

СПРАВКА. Стандартные ДО подключаются – "вставляются" в форму как обычные ЭУ из панели элементов (рисунок 10), но в форме не отображаются. Однако соответствующие пиктограммы можно увидеть в Окне конструктора формы ниже вида самой формы. А сами операции открытия и сохранения файлов с использованием этих окон автоматически не поддерживаются и требуют допрограммирования. Таким образом, указанные окна – это готовые формы, отображаемые в виде модальных ДО, их функциональность обеспечивается классами OpenFileDialog, SaveFileDialog соответственно. При этом они (их объекты) подключаются – агрегируются в форму как атрибуты ее класса автоматически, если указанные ДО добавляются к форме в Редакторе форм как обычные ЭУ.

Для этого:

1. Создайте ТКП, добавьте в Форму меню и стандартные ДО.

Система добавит код в класс формы: private System.Windows.Forms.OpenFileDialog openFileDialog1 и private System.Windows.Forms.SaveFileDialog saveFileDialog1, а в ее конструкторе InitializeComponent инициализирует эти объекты

```
this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog( );
this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog( ).
```

2. Настройте ДО вручную, например, как

```
this.openFileDialog1.FileName = "\"D:\\\\Text1.txt\"";
this.openFileDialog1.Filter = "\"Текстовые файлы (*.txt) | *.txt | All files (*.*) | *.*\"";
...
this.saveFileDialog1.Filter = "\"Текстовые файлы (*.txt) | *.txt | All files (*.*) | *.*\"";
```

или в окне Свойств как показано на рисунках 11, 12. Для открытия этих окон надо в Конструкторе формы выделить соответствующую пиктограмму ниже самой формы.

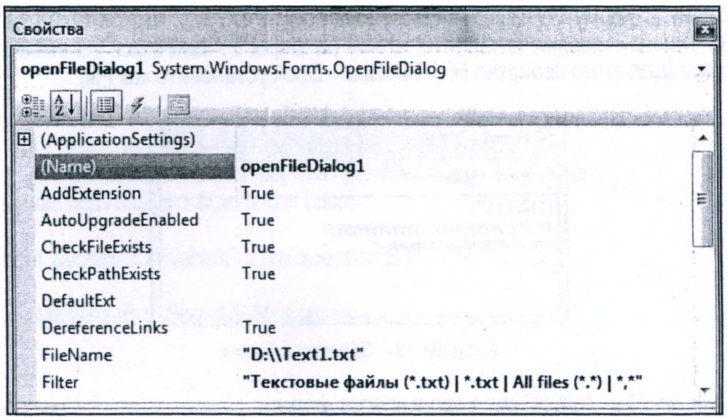

Рисунок 11 – Настройка окна OpenFileDialog

3. Организуйте их запуск в модальном режиме командой ShowDialog в обработчиках соответствующих пунктов меню, например, как this.openFileDialog1.ShowDialog и this.saveFileDialog1.ShowDialog. Для выхода из приложения используйте this.Close.

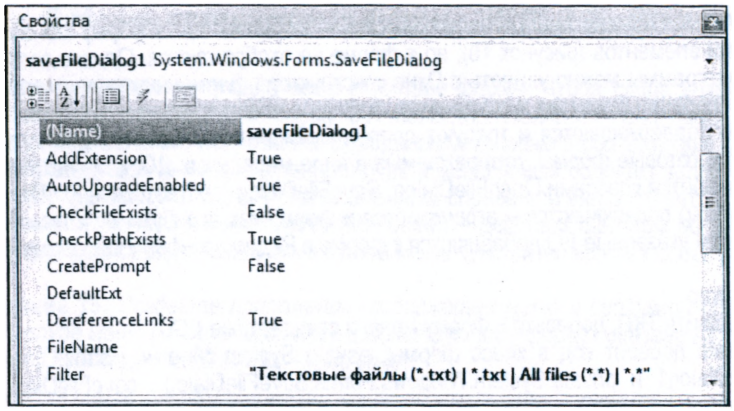

Рисунок 12 – Настройка окна SaveFileDialog

► **Задание 17.** *Создайте аналогичное Заданию 16 приложение с меню (рисунок 13), выполняющее роль упрощенного текстового редактора на базе ЭУ RichTextBox, расположенного поверх КО главной формы.*

Пункт меню Open обеспечивает открытие и загрузку в ЭУ RichTextBox выбранного rtf-файла, а Close обеспечивает сохранение текста из ЭУ RichTextBox в выбранном rtf-файле и месте файловой системы. Завершение работы приложения выполняется системной кнопкой закрытия окна.

1. Разработайте диаграмму состояний приложения (приведите в Отчете).
2. Создайте ТКП, добавьте в Форму меню. Для поддержки загрузки-выгрузки файла с расширением rtf подключите стандартные ДО типа OpenFileDialog и SaveFileDialog (см. Задание 16).
3. Добавьте в форму ЭУ RichTextBox, разместите его так, чтоб он был расположен поверх КО главной формы, настройте, чтобы размеры ЭУ менялись бы вместе с размерами формы (для этого свойство RichTextBox – Dock установить как Fill).

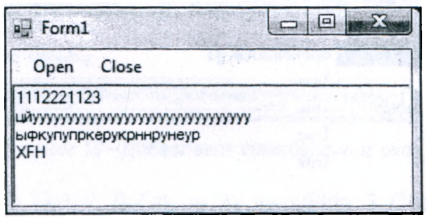

Рисунок 13 – Образец вывода

При этом система сгенерирует код в классе формы

```
private System.Windows.Forms.OpenFileDialog openFileDialog1;
private System.Windows.Forms.SaveFileDialog saveFileDialog1;
private System.Windows.Forms.RichTextBox richTextBox1;
```

и в ее конструкторе

```
private void InitializeComponent()
    {
        ...
        this.richTextBox1 = new System.Windows.Forms.RichTextBox();
        this.openFileDialog1 = new System.Windows.Forms.OpenFileDialog();
        this.saveFileDialog1 = new System.Windows.Forms.SaveFileDialog();

        // richTextBox1
        this.richTextBox1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.richTextBox1.Location = new System.Drawing.Point(12, 27);
        this.richTextBox1.Name = "richTextBox1";
        this.richTextBox1.Size = new System.Drawing.Size(259, 220);
        this.richTextBox1.TabIndex = 0;
        this.richTextBox1.Text = "";

        // openFileDialog1
        this.openFileDialog1.FileName = "openFileDialog1";

        // Form1
        this.Controls.Add(this.richTextBox1);
        ...
    }
```

4. Управление ДО реализуется методами-обработчиками пунктов меню

```
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        this.openFileDialog1.ShowDialog( );
        if ( (this.openFileDialog1.FileName) == null ) return;
        this.richTextBox1.LoadFile( this.openFileDialog1.FileName );
        this.richTextBox1.Modified = false;
    }
    catch( System.IO.FileNotFoundException S )
    {
        MessageBox.Show( S.Message );
    }
}

private void closeToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.saveFileDialog1.FileName = this.openFileDialog1.FileName;
    if ( this.saveFileDialog1.ShowDialog( ) == System.Windows.Forms.DialogResult.OK )
        try
```

```
        {
            this.richTextBox1.SaveFile(this.saveFileDialog1.FileName);
            this.richTextBox1.Modified = false;
        }
        catch(System.IO.FileNotFoundException S)
        {
            MessageBox.Show(S.Message);
        }
} .
```

5 *Модифицируйте приложение. Контролируйте событие FormClosing, в его обработчике определите – была ли модификация текста. Обеспечьте пользователю возможность сохранить измененный текст. Процедуру сохранения текста оформите как внутренний, закрытый метод.*

СПРАВКА. Для контроля изменения текста – содержимого ЭУ richTextBox – используйте его свойство – флаг richTextBox1.Modified. Для запроса на сохранение измененного текста в файле используйте MessageBox.

6 Многооконные приложения

Основу ГИП приложения составляют окна. Это формы и диалоговые окна (ДО), реализуемые на базе форм. ДО используются как модальные и немодальные окна, бывают пользовательские и стандартные. Формы и ряд других компонентов ГИП выполняют роль контейнеров, т. е. содержат другие компоненты, а также специализированные окна – элементы управления (ЭУ).

Форма – универсальное окно, содержит клиентскую область (КО) для вывода данных и размещения ЭУ. Как правило, имеет системное меню, системные кнопки, может включать пользовательское меню и панели инструментов. Окно масштабируемое и перерисовываемое по желанию пользователя.

Диалоговые окна предназначены для размещения ЭУ, отличаются общепринятым стилем: окно не имеет меню, содержит рекомендуемые кнопки типа OK и Cancel, имеет постоянный размер (устанавливается значением FixedDialog свойства формы FormBorderStyle, при этом свойства MinimizeBox и MaximizeBox для кнопок изменения размера окна задаются как false).

Для кнопок, которые предполагается использовать в стандартном контексте, например, как OK, Cancel, Yes и т. д., надо задать ассоциируемый с ними код возврата (возвращаемое формой значение при их нажатии). Можно также указать для кнопок OK, Cancel аналогичные по действию клавиши Enter и Esc. То есть форма может реагировать на клавишу Esc как на кнопку Отмены (если установлено свойство CancelButton), а на клавишу Enter как на кнопку приема данных формы (если установлено свойство AcceptButton).

Соответственно при реализации ДО: 1 – для кнопок, выполняющих роль OK и Cancel, установите свойство DialogResult как OK и Cancel соответственно, и при необходимости скорректируйте свойство TabIndex; 2 – в самой форме установите, по какой кнопке выполняется прием данных и по какой – выполняется отмена работы с ДО (для этого свойство формы AcceptButton устанавливается идентификатором (name) кнопки OK, а свойство CancelButton идентификатором (name) кнопки Cancel.

Разработка форм (окон) предполагает работу как с графическим ресурсом – изображением (видом окна, его дизайном) так и с классом, поддерживающим его функциональность. Основные этапы разработки: 1) проектирование окна (дизайн, поведение, запуск, завершение, передача данных и т. д.); 2) создание ресурса окна (в редакторе ресурсов или программно), т. е. формы и составляющих ее элементов. При этом код класса окна генерируется автоматически либо прописывается вручную; 3) настройка свойств окна – в окне Свойства или программным способом (в первом случае код класса окна корректируется автоматически); 4) настройка поведения окна (добавление обработчиков) – через окно Свойства либо программным способом; 5) организация использования окна в приложении – создание объекта класса окна, запуск окна, передача данных, завершение работы окна.

6.1 Графический интерфейс многооконного приложения

► **Задание 18.** *Создайте приложение с формой с названием MAIN (в качестве главного окна) и диалоговым окном с названием DIALOG, загружаемым в модальном режиме из формы по щелчку клавиши мыши (событие Click в площади формы).*

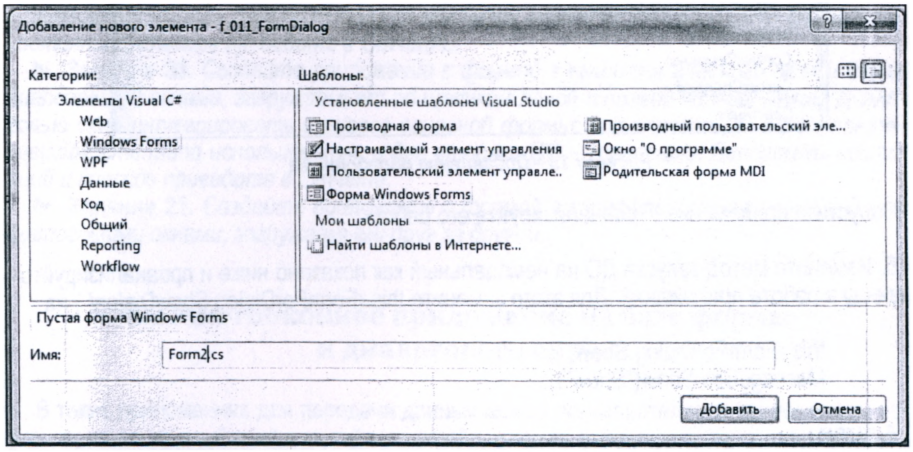

Рисунок 14 – Добавление формы

Для этого:

1. Спроектируйте диаграммы классов и состояний приложения (приведите в Отчете). Создайте ТКП.

2. Добавьте к проекту еще одну форму для использования в качестве ДО. Для этого выполните ГМ – Проект – ДобавитьНовыйЭлемент – Элемент Visual C# – Форма (Windows Forms) как показано на рисунке 14. Автоматически будет добавлен графический ресурс – новая форма (окно) и class Form2 для его поддержки. В Обозревателе решений будет показан добавленный автоматически класс и файлы. Теперь в составе проекта есть две несвязанные формы в одном пространстве имен!).

3. Подготовьте запуск ДО – внесите изменения в класс главной формы:

– в атрибуты класса Form1 добавьте ссылку на ДО как public Form2 rFormForDialog;

– в конструктор Form1 вставьте команду создания объекта формы Form2

```
public Form1( )
{
        InitializeComponent( );
         //TODO: добавьте код конструктора
         rFormForDialog = new Form2( );
}
```

4. Организуйте вызов ДО в модальном режиме. Для этого добавьте в главную форму обработчик щелчка клавиши мыши – Click (рисунок 15) и выполните в нем запуск ДО

```
private void Form1_Click(object  sender, EventArgs  e)
                        {     this.rFormForDialog.ShowDialog( );        } .
```

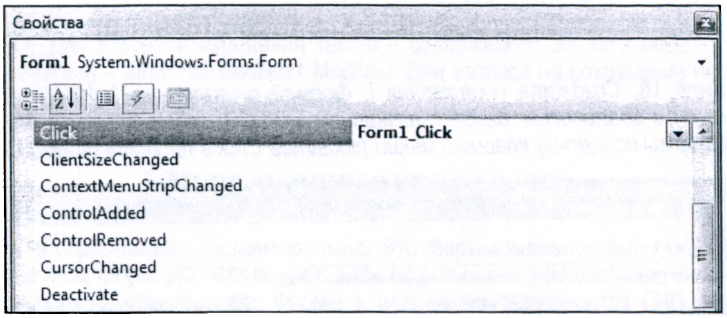

Рисунок 15 – Добавление обработчика

Запустите приложение. Проанализируйте его работу.

5. Измените метод запуска ДО на немодальный как показано ниже и проанализируйте разницу в работе приложений. Для этого замените this.rFormForDialog.ShowDialog( ) на

```
this.rFormForDialog.Show( );
MessageBox.Show("Show").
```

А затем на

```
this.rFormForDialog.Show( );
MessageBox.Show("Show");
this.rFormForDialog.Hide( );
MessageBox.Show("Hide");
this.rFormForDialog.Show( );    .
```

Верните модальный режим.

6. Настройте свойства второй формы для ее работы в качестве диалогового окна. Для этого откройте ДО (вторую форму) в Конструкторе:

– зафиксируйте размеры ДО (установите свойство FormBorderStyle как FixedDialog);

– отключите системные кнопки (установите свойства MinimizeBox, MaximizeBox как false);

– добавьте кнопки и назовите их myOK и myCancel;
– сообщите системе, что они должны восприниматься именно как OK и Cancel (установите их свойства DialogResult как OK и Cancel соответственно);
– для самой формы определите как реагировать на их использование (установите свойства AcceptButton = button1 для кнопки myOK и CancelButton – button2 для кнопки myCancel).

7. "Цивилизуйте" запуск ДО, выполняя в точке его запуска анализ кода возврата – кода нажатой в ДО кнопки (здесь myOK или myCancel). Выводите соответствующее сообщение по образцу ниже

```
if (this.rFormForDialog.ShowDialog( ) == System.Windows.Forms.DialogResult.OK)
{
    MessageBox.Show("Back from Dialog by button - OK");
    // сегмент обработки данных из ДО
    ...
} .
```

► **Задание 19**. *Создайте приложение с формой в качестве главного окна и двумя диалоговыми окнами, загружаемыми по нажатию кнопок из главного окна. Диаграммы состояний и классов приведите в Отчете.*
► **Задание 20.** *Создайте приложение с формой в качестве главного окна и двумя диалоговыми окнами, загружаемыми по щелчкам левой и правой клавиш мыши. Диалоговые окна интегрировать в классе основной формы: для первого ДО использовать зависимость типа использования, а для второго ДО – агрегацию. Диаграммы состояний и классов приведите в Отчете.*
► **Задание 21.** *Создайте приложение с формой в качестве главного окна и двумя диалоговыми окнами, загружаемыми друг за другом.*

6.2 Многооконное приложение на базе формы и диалогового окна

В таких приложениях для передачи данных между пользователем и приложением через соответствующий графический интерфейс, а также между формами приложения используются соответствующие классы и их атрибуты. Поэтому следует учитывать: 1. Даже при деактивизации окна текущая информация может сохраняться в атрибутах его класса и участвовать в дальнейшей обработке, в других процессах. Для этого надо обеспечить доступ к таким атрибутам, как минимум – сделать их открытыми. 2. Многие ЭУ, поддерживаемые соответствующими системными классами, имеют открытые атрибуты для доступа к ассоциированным с ними данным. Например, атрибут Text окна редактирования, атрибут – коллекция Items ЭУ список и т. д.

► **Задание 22.** *Создайте приложение (с интерфейсом, подобным в Задании 21) с формой в качестве главного окна и дочерним диалоговым окном, загружаемым из формы в модальном режиме нажатием кнопки (событие Click). Образец интерфейса приведен на рисунке 16.*

В диалоговом окне можно ввести строку в поле редактирования. По нажатии кнопки ОК (событие Click) диалоговое окно теряет активность и в главное окно в качестве кода завершения передается код нажатой кнопки – здесь ОК.

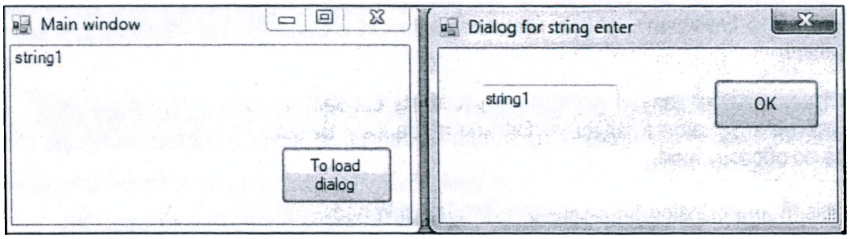

Рисунок 16 – Примерный вид интерфейса

Для главной формы это послужит признаком того, что пользователем в диалоговом окне было введено новое данное – строка. При этом предполагается, что текущее значение строки было считано и сохранено (например, в открытом атрибуте класса ДО), а следовательно ее можно обрабатывать методами главного окна (формы). Здесь необходимо вывести заданную строку в клиентскую область главной формы, активизировав ее перерисовку.

В противном случае, если работа с ДО была окончена по системной кнопке завершения, то результат ввода не сохраняется (или не обновляется) и соответственно вывод не требуется, т. е. КО главного окна не перерисовывается.

СПРАВКА

1. Для организации совместной работы окон приложения в качестве отношения между их классами (Form1 и Form2) используется композиция (агрегация) – ссылка на объект класса Form2 будет атрибутом класса Form1, например, как private Form2 rForm2.

2. Для сохранения введенной строки в объекте класса ДО (Form2) и доступа к ней извне в этом классе используется открытая переменная (атрибут) – public String X.

Для разработки приложения:

1. Спроектируйте диаграммы классов и состояний приложения (в Отчет).

2. Создайте проект приложения (каркас).

3. Добавьте в главную форму (Form1) кнопку для загрузки дочерней формы (Form2), а также пустые обработчики события нажатия кнопки Click и события перерисовки КО.

4. Добавьте к проекту дочернюю форму (class Form2) для использования в качестве ДО для ввода строки. Для этого выполните ГМ – Проект – ДобавитьНовыйЭлемент – Элемент Visual C# – Форма (Windows Forms).

5. Доработайте дочернюю форму (Form2) для ввода строки. Для этого добавьте кнопку, окно редактирования, а также пустой обработчик события Click для кнопки.

Кнопку настройте для применения в режиме "ОК", установив свойство DialogResult как ОК. В обработчик нажатия кнопки добавьте команду сохранения результата ввода в атрибуте объекта класса Form2 как X = this.textBox1.Text.

6. Реализуйте запуск ДО. Для этого:

– в класс Form1 добавьте атрибут – ссылку private Form2 rForm2;

– в конструктор Form1 инициализируйте ссылку как rForm2 = new Form2( ).

7. Организуйте вызов ДО в модальном режиме в обработчике сообщения Click кнопки главной формы. При этом, если в ДО будет введена новая строка (признак – код возврата кнопки ОК), то вызовите перерисовку данных в КО главной формы

```
if ( this.rForm2.ShowDialog( ) == System.Windows.Forms.DialogResult.OK )
{
    MessageBox.Show("Back from Dialog by button OK");
    // сегмент обработки данных из ДО
    this.Invalidate( );
    ...
}
```

8. Организуйте вывод строки X, введенной в ДО, в КО главной формы по событию ее перерисовки. Здесь X – открытый атрибут объекта Form2, являющегося, в свою очередь, атрибутом класса Form1. Используйте, например, метод DrawString

```
private void Form1_Paint(object  sender, PaintEventArgs  e)
 {
     e.Graphics.DrawString(...);
 } .
```

9 Внесите изменения – во этом варианте реализации для доступа к результату ввода в классе ДО (Form2) используйте собственный атрибут поля редактирования этого класса.

► **Задание 23**. *Внесите изменения в Задание 22 для запуска ДО в немодальном режиме (hForm2.Show( )).*

► **Задание 24**. *Внесите изменения в Задание 22, чтобы при запуске ДО в нем отображался новый ЭУ типа Label, в котором выводился номер текущего вызова ДО.*

Реализуйте указанное, например, программно в обработчике сообщения Click кнопки запуска ДО.

► **Задание 25**. *Разработать приложение, аналогичное Заданию 22 с интерфейсом из трех ДО. ДО в роли главного должно содержать кнопки запуска двух других – одного для ввода строки, другое для ее отображения.*

► **Задание 26**. *Повторить Задание 25 для ввода набора чисел и их обработки.*

► **Задание 27.** *Разработать диаграммы прецедентов, классов, состояний (привести в Отчете) и само приложение для рисования плоских фигур – многоугольников – отрезками прямых. Для ввода массива координат использовать отдельное (немодальное) диалоговое окно. Изображение фигуры выводить в КО главного окна. При этом первая и последняя из заданных точек (координат) соединяются прямой линией. Для работы с фигурой используйте пользовательские классы. Для хранения координат используйте динамические структуры C#.*

► **Задание 28.** *Модифицировать приложение – Задание 27: добавить отдельное окно для вывода таблицы координат точек.*

► **Задание 29.** *Модифицировать приложение – Задание 27: – добавить сохранение-загрузку координат; – управление приложением реализовать через меню. Разработать диаграммы прецедентов, классов, состояний (привести в Отчете).*

Порядок выполнения работы

Цикл лабораторных в основном варианте (для пользователей, ранее не работавших с формами, но имеющими представление о базовых средствах языка C#) предусматривает около 20 часов занятий, включая 10–12 часов аудиторных работ и самостоятельную работу. Ниже представлен один из возможных сценариев выполнения заданий.

Лабораторно выполняются задания 1, 2, 3 – ТКП; 4 – вывод в КО; 5, 7–9 – сообщения; 15, 16 – меню; 18, 19, 22, 23, 25 – формы, ДО. В отчет выносится описание реализованных приложений (задания 1–3, 7, 16, 18, 19, 22).

Далее лабораторно выполняются задания 12–14 – ЭУ; 6, 11.1, 11.2 – рисование в КО; 17 – типовые окна, редактор; 10 – сообщения, активность окон; 11.3 – рисование в КО; 20 – формы, ДО. В отчет выносится описание реализованных приложений (задания 10, 13, 17, 20).

Самостоятельно выполняются задания 11.4, 21, 24, 26, 27, 28, 29.

Учебное издание

Составители:
Муравьев Геннадий Леонидович
Мухов Сергей Владимирович
Савицкий Юрий Викторович
Крапивин Юрий Борисович
Хвещук Владимир Иванович

Методические указания

к выполнению лабораторных работ
"Разработка приложений с формами средствами visual C#"
для студентов специальности 1-40 03 01 «Искусственный интеллект»

Ответственный за выпуск: Муравьев Г. Л.
Редактор: Митлошук М. А.
Компьютерная вёрстка: Соколюк А. П.
Корректор: Дударук С. А.

Подписано в печать 24.10.2023 г. Формат 60x84 $^{1}/_{16}$. Бумага «Performer».
Гарнитура «Arial Narrow». Усл. печ. л. 1,86. Уч. изд. л. 2,0. Заказ № 1170. Тираж 22 экз.
Отпечатано на ризографе учреждения образования «Брестский государственный технический университет». 224017, г. Брест, ул. Московская, 267.
Свидетельство о государственной регистрации издателя, изготовителя, распространителя печатных изданий № 1/235 от 24.03.2014 г.