МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания к лабораторным работам

Направления подготовки:

09.03.04 «Программная инженерия», Направленность (профиль) «Инженерия сложных программных систем»

Квалификация выпускника: бакалавр

Ставрополь, 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Лабораторный практикум по дисциплине «Технологии разработки программного обеспечения» представляет собой цикл из 22 лабораторных работ, посвященный практическим аспектам проектирования, разработки и отладки программного обеспечения с использованием инструментария Microsoft Visual Studio.

Сложность освоения Технологии разработки программного обеспечения для Windows, в совокупности с постоянным ростом популярности данной операционной системы, указывает на необходимость подготовки специалистов в данной области. Увеличение количества технологий в сфере создания Windows приложений указывает на необходимость выбора отдельного инструментария, поддерживаемого непосредственно производителем операционной системы Windows. В качестве такого инструментария в данных методических указаниях используется IDE MS Visual Studio.

Актуальность обучения специалистов в области проектирования и разработки Windows приложений продиктована постоянным совершенствованием технологий Microsoft в сфере разработки интерфейса, доступа к внешним данным, двумерных и трехмерных графических технологий, сетевых технологий, а также растущим распространением этих технологий в различных прикладных сферах.

Основное внимание в первой части пособия уделено вопросам, связанным с синтаксисом языка C#; стандартной библиотеке C#; массивам; основами объектно-ориентированного программирования; основам разработки приложений с графическим интерфейсом.

ЛАБОРАТОРНАЯ РАБОТА 1. КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ C#

1. Цель и содержание

Цель лабораторной работы: научиться работать с переменными и константами простых типов в C#.

Задачи лабораторной работы:

– научиться объявлять переменные простых типов в языке C#;

– научиться объявлять константы простых типов в языке C#;

– научиться выполнять простейшие действия с переменными и константами.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

3.1 Структура консольного приложения

Рассмотрим структуру консольного приложения на языке C#, созданного с использованием средств MS Visual Studio (рис. 1.1).

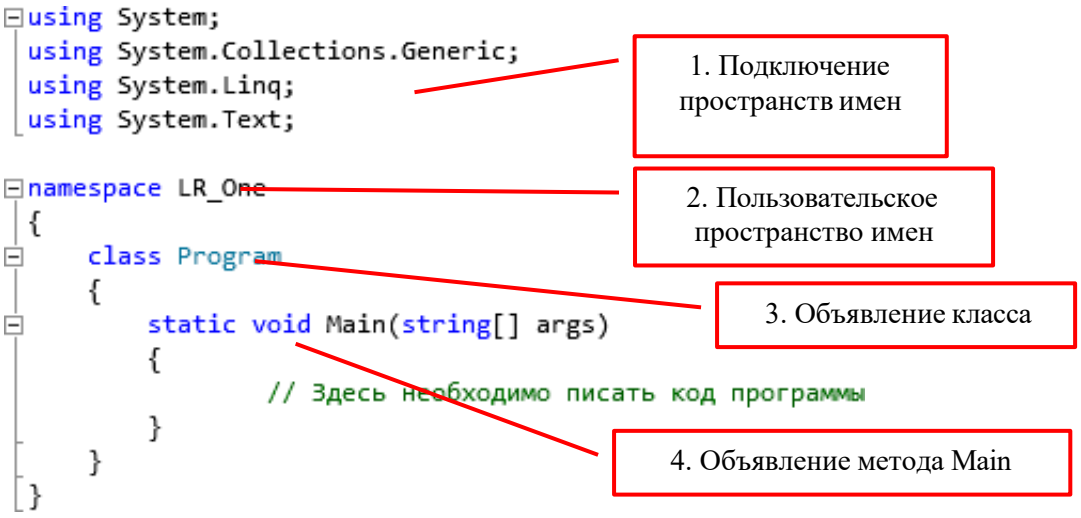


Рисунок 1.1 – Исходный код консольного приложения

В интерактивном редакторе кода среды разработки код может быть свернут/развернут с использованием кнопок +/-, внедренных в код. На рис. 1.1 в области 1 показано подключение пространств имен (библиотек, содержащих стандартные инструменты) с использованием зарезервированного слова using.

Программист сам создает свое пространство имен с именем LR_One (2), в котором объявляется класс с именем Program.

В классе Program объявлен один метод – функция Main (параметры функции не рассматриваем).

Функция Main имеет особенное значение в программировании на языках C, C++ и C#.

Функцию Main называют «точкой входа», то есть началом выполнения программы. Далее мы рассмотрим приложения, содержащие множество функций. Операционная система знает с какой именно функции начать выполнение программы – с функции Main. Очевидно, что имя этой функции менять нельзя. Это должен быть статический метод класса (или структуры), возвращающий либо значение типа int, либо void. Хотя нередко модификатор public указывается явно, поскольку по определению этот метод должен быть вызван извне программы, на самом деле неважно, какой уровень доступа вы назначите методу точки входа. Он запустится, даже если вы пометите его как private.

Когда компилируется консольное или Windows-приложение C#, по умолчанию компилятор ищет в точности один метод Main () с описанной выше сигнатурой в любом классе и делает его точкой входа программы. Если существует более одного метода Main ( ), компилятор возвращает сообщение об ошибке.

Обратите внимание на вложенность конструкций на рис. 1.1: в пространство имен LR_One вложен класс Program, в класс Program вложена функция Main. В свою очередь, в функции Main содержатся инструкции на языке C#-код, который начнет выполняться при старте программы.

В C#, как и в других C-подобных языках, большинство операторов завершаются точкой с запятой (;) и могут продолжаться в нескольких строках без

необходимости указания знака переноса. Операторы могут быть объединены в блоки с помощью фигурных скобок ({ }). Однострочные комментарии начинаются с двойного слеша (//), а многострочные – сослеша со звездочкой (/*) и заканчиваются противоположной комбинацией (*/). В этих аспектах язык C# идентичен C++ и Java.

Следует также помнить о том, что язык C# чувствителен к регистру символов. Это значит, что переменные с именами myVar и MyVar являются разными.

Причина присутствия оператора using в файле Program.cs связана с использованием библиотечных классов.

Весь код C# должен содержаться внутри класса. Объявление класса состоит из ключевого слова class, за которым следует имя класса и пара фигурных скобок. Весь код, ассоциированный с этим классом, размещается между этими скобками.

3.2 Типы значений C#

Язык C# поддерживает 8 предопределенных целочисленных типов (таблица 1.1).

Таблица 1.1 – Целочисленные типы C#

Имя типа	Тип CTS	Описание	Диапазон (минимум : максимум)
sbyte	System.SByte	8-битное целое со знаком	-128 : 127
short	System.Int16	16-битное целое со знаком	-32 768 : 32 767
int	System.Int32	32-битное целое со знаком	-2 147 483 648 : 2 147 483 647
long	System.Int64	64-битное целое со знаком	$-2^{63} : 2^{63}-1$
byte	System.Byte	8-битное целое без знака	0 : 255
ushort	System.UInt16	16-битное целое без знака	0 : 65 535
uint	System.UInt32	32-битное целое без знака	$0 : 2^{32}-1$

ulong	System.UInt64	64-битное целое без знака	0 : 2^{64}-1

Язык C# также поддерживает и типы с плавающей точкой (таблица 1.2).

Таблица 1.2 – Типы с плавающей точкой C#

Имя типа	Тип CTS	Описание	Кол-во знаков	Диапазон (минимум : максимум)
float	System.Single	32-битное с плавающей точкой одинарной точности	7	от $\pm 1.5\times10^{-45}$ до $\pm 3.4\times10^{38}$
double	System.Double	64-битное с плавающей точкой двойной точности	15/16	от $\pm 5.0\times10^{-324}$ до $\pm 1.7\times10^{308}$

В таблице 1.3 представлен десятичный тип C#. Данный тип реализован для финансовых операций.

Таблица 1.3 – Десятичный тип C#

Имя типа	Тип CTS	Описание	Кол-во знаков	Диапазон (минимум : максимум)
decimal	System.Decimal	128-битное с плавающей точкой в десятичной нотации с высокой точностью	28	от $\pm 1.0\times10^{-28}$ до $\pm 7.9\times10^{28}$

Как и во многих языках программирования существует булевский тип (таблица 1.4).

Таблица 1.4 – Булевский тип.

Имя типа	Тип CTS	Значения
bool	System.Boolean	true или false

Для хранения одиночных символов в языке C# используется тип char (таблица 1.5)

Таблица 1.5 – Булевский тип.

Имя типа	Тип CTS	Значения
char	System.Char	Представляет отдельный 16-битный (Unicode) символ

Литералы типа char записываются как одиночные, заключенные в одинарные кавычки символы: ‘F’, ‘w’, ‘ц’, ‘Я’ и т.д.

В переменных типа char можно хранить и специальные символы в виде управляющих последовательностей (таблица 1.6).

Таблица 1.6 – Представление символов в виде управляющих последовательностей.

Управляющая последовательность	Символ
\’	Одиночная кавычка
\”	Двойная кавычка
\\	Обратный слэш
\0	Пусто
\a	Предупреждение (звуковой сигнал)
\b	Забой
\f	Подача формы
\n	Новая строка
\r	Возврат каретки
\t	Символ табуляции
\v	Вертикальная табуляция

Если отдельные символы объединены в строку, то необходимо использовать тип string, который отображается на тип CTS – System.String.

3.3 Объявление и инициализация переменных в C#

Синтаксис объявления переменных в C# выглядит следующим образом:

ТипДанных ИдентификаторПеременной;

Например:

```
int a;
```

Этот код объявляет переменную типа int с именем a. Компилятор не позволит использовать эту переменную до тех пор, пока она не будет инициализирована (т.е. пока ей не будет присвоено значение).

Для инициализации переменной a необходимо написать следующий код:

```
a = 123;
```

Переменную можно инициализировать во время объявления:

```
int b = 7;
```

или

```
string str = "Hello, World!!!";
```

Синтаксис C# позволяет объявить несколько переменных (и инициализировать их) одного типа в одной синтаксической конструкции.

Например:

```
float b, i, myPerem, U_t = 0.3F, z_11 = 23.56F;
```

В данном примере объявляется 5 переменных типа float, некоторые из них инициализируются в процессе объявления.

3.4 Объявление и инициализация констант в C#

Константа – это переменная, значение которой не меняется за время выполнения программы. Для объявления константы необходимо воспользоваться ключевым словом const. Пример:

```
const char simv = 'A';
const double pi = 3.14;
```

Очевидно, что при таком объявлении, поменять значения simv и pi в дальнейшем будет нельзя.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64)

процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

6.1 Создание приложения для вывода персональной информации

1. Создайте консольное приложение, для этого выполните следующие действия:

1.1 Выберите команду главного меню *File* → *New* → *Project*...

1.2 В открывшемся диалоговом окне (рис. 1.2) выберите необходимые настройки для создаваемого проекта: язык Visual C#; фреймворк: .NET Framework 4 и выше; шаблон: Console Application.

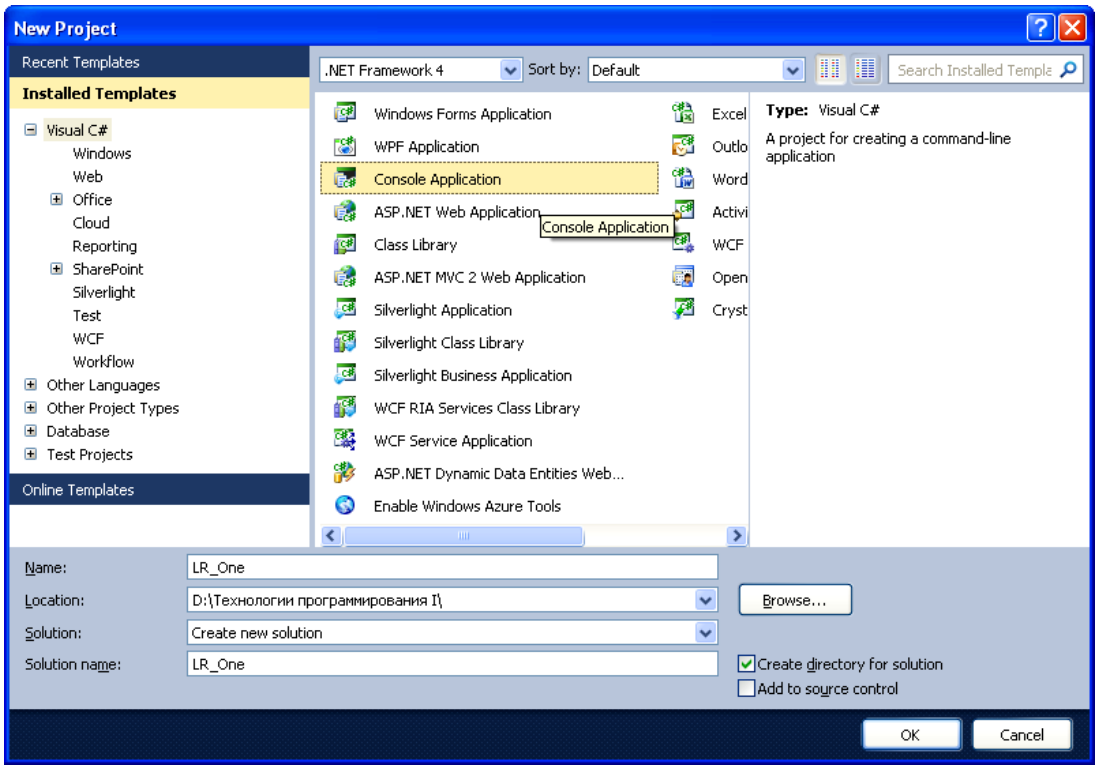


Рисунок 1.2 – Создание нового проекта консольного приложения

1.3 В текстовом поле Name введите имя проекта (например LR_One).

1.4 В текстовом поле Location выберите место сохранения нового проекта.

1.5 Установите флажок-переключатель «Create directory for solution».

1.6 Нажмите кнопку «OK».

2. После выполнения пункта 1 в среде разработки откроется новый созданный проект (рис. 1.3).

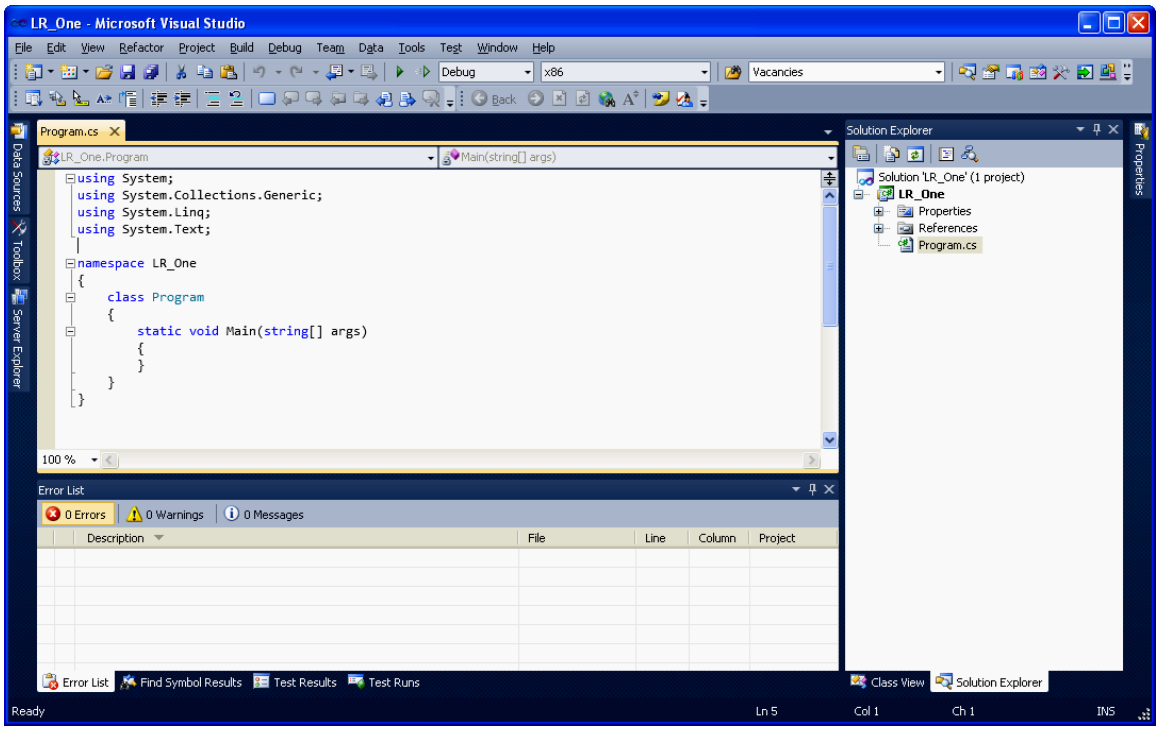


Рисунок 1.3 – Новый проект, загруженный в среду разработки: вкладка «Solution Explorer» отображает состав проекта (нас интересует только файл Program.cs); в левой части окна (сверху) открыт файл Program.cs в редакторе кода

3. На данном этапе необходимо ознакомиться со структурой исходного файла консольного приложения (рис. 1.4).

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
                // Здесь необходимо писать код программы
        }
    }
}
```

Рисунок 1.4 – Исходный файл консольного приложения

4. Весь код программы необходимо писать внутри функции Main.

5. Для построения сборки (исполняемого exe-файла) выполните команду главного меню *Build* → *Build Solution* (или использовать горячую клавишу *F6*). После этого сборка создана, но приложение не будет запущено автоматически.

6. Для создания сборки и последующего запуска программы можно воспользоваться командой *Debug* → *Start Debugging* главного меню среды разработки или нажать кнопку панели инструментов  *Start Debugging* (*F5*). Можно также использовать горячую клавишу *F5*.

7. Запустите приложение на выполнение одним из методов, указанных в пункте 6. Окно консольного приложения появится и исчезнет. Это означает, что приложение выполнило все команды, написанные программистом, и завершило свою работу.

8. Для удержания окна на экране измените исходный файл в соответствии с рисунком 1.5. В функции Main добавлен вызов только одной команды Console.DeadKey ( ) – эта функция останавливает выполнение программы и ждет, когда пользователь нажмет любую клавишу.

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ReadKey();
        }
    }
}
```

Рисунок 1.5 – Исходный файл консольного приложения для предотвращения закрытия окна консольного приложения

9. Запустите измененное приложение, убедитесь, что окно удерживается на экране.

10. Добавьте несколько строк кода в исходный файл (рис. 1.6).

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №1");
            Console.WriteLine("");
            Console.WriteLine("Выполнил: Иванов Иван иванович");
            Console.WriteLine("Группа: ИСТБ-121");
            Console.WriteLine("Наименование ЛР: Структура консольного приложения");
            Console.WriteLine("");
            Console.WriteLine("для завершения работы программы нажмите любую клавишу...");

            Console.ReadKey();
        }
    }
}
```

Рисунок 1.6 – Исходный файл консольного приложения для вывода информации на экран.

11. Внимательно изучите исходный код примера на рис. 1.5. Запустите приложение и убедитесь, что отсутствуют ошибки и информация выводится.

6.2 Приложение для вычисления значения выражения

1. Создайте новое консольное приложение.

2. Изучите материал в разделе «Теоретическое обоснование» (подпункты 3.2 – 3.4) данной лабораторной работы.

3. Модифицируйте исходный файл как показано на рис. 1.7.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace LR_One
7  {
8      class Program
9      {
10         static void Main(string[] args)
11         {
12             int a;
13             int b = 7;
14
15             string str = "Hello, World!!!";
16
17             a = 123;
18
19             Console.ReadKey();
20         }
21     }
22 }
```

Рисунок 1.7 – Объявление переменных в C#

4. Рассмотрим приведенный пример подробнее:

4.1. В строке 12 объявляется переменная типа int с именем a.

4.2. В строке 13 объявляется переменная типа int с именем b, причем при объявлении для нее устанавливается начальное значение, равное 7.

4.3. В строке 15 объявляется переменная типа string с именем str и инициализируется строковым значением «Hello, World!!!».

4.4. В строке 17 переменной a присваивается целочисленное значение 123.

5. Запустите приложение на выполнение. У вас должно появиться пустое окно консольного приложения. Очевидно, что исходный код, представленный на рис. 1.7 не предполагает вывода какой-либо информации на экран.

6. Для вывода информации на экран воспользуемся функцией Console.WriteLine. Добавим в исходный файл следующие строки (строки 19 – 21 на рис. 1.8).

```
6   namespace LR_One
7   {
8       class Program
9       {
10          static void Main(string[] args)
11          {
12              int a;
13              int b = 7;
14
15              string str = "Hello, World!!!";
16
17              a = 123;
18
19              Console.WriteLine(a);
20              Console.WriteLine(b);
21              Console.WriteLine(str);
22
23              Console.ReadKey();
24          }
25      }
26  }
27
```

Рисунок 1.8 – Добавление строк кода для вывода значений объявленных переменных на экран

7. В примере на рис. 1.8 используется простой вывод, то есть имя переменной просто передается в качестве параметра функции Console.WriteLine.

8. Для выполнения форматного вывода (изменения формата представления выводимой информации) необходимо реализовать вывод в следующем виде (рис. 1.9 изменены строки 19-21):

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

namespace LR_One
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            int b = 7;

            string str = "Hello, World!!!";

            a = 123;

            Console.WriteLine("Значение переменной a равно {0}", a);
            Console.WriteLine("a значение переменной b равно {0}", b);
            Console.WriteLine("значение a+b = {0}+{1} = {2}", a, b, a+b);
            Console.WriteLine(str);

            Console.ReadKey();
        }
    }
}
```

Рисунок 1.9 – Вывод информации с использованием форматных строк

9. Внимательно изучите код полученной программы, выполните индивидуальное задание.

Индивидуальное задание

Индивидуальное задание состоит из двух частей:

1. Измените приложение, созданное в ходе выполнения данной лабораторной работы таким образом, чтобы программа выводила на экран следующую информацию (каждый студент должен использовать персональную информацию о себе):

– Название и номер лабораторной работы;

– ФИО студента;

– Группа студента и шифр специальности;

– Дата рождения студента;

– Населенный пункт постоянного места жительства студента;

– Любимый предмет в школе;

– Краткое описание увлечений, хобби, интересов.

2. Создайте второй проект. Объявите требуемые переменные, присвойте им начальные значения (определите самостоятельно, значения какого типа могут принимать переменные), выведите на экран с использованием форматной строки значения переменных и результат вычисления выражения в соответствие с вариантом:

Вариант	Выражение для вычисления
1	$F = a\_1 + b - a \cdot (x + y5)$
2	$Se = w111 + bt - x + y \cdot w$
3	$R\_x = a \cdot b + b / t - x + f \cdot i\_2$
4	$c = gh + b \cdot q3 - x + y / w$
5	$H = \frac{g \cdot h}{d17} + b / h1 - \frac{x + y}{4}$
6	$Z = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$
7	$U = \frac{35}{a} \cdot z + z \cdot a - \frac{a + E \cdot t}{4}$
8	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
9	$Se = w111 + bt - x + y \cdot w$
10	$R\_x = a \cdot b + b / t - x + f \cdot i\_2$
11	$g = w \cdot h + b \cdot q3 - q3 + y / w$
12	$ee = \frac{g \cdot h}{d17} + d17 / h - \frac{g + d17 + h}{4}$
13	$Zze = \frac{35}{f} + y \cdot f - \frac{f + y}{4}$
14	$t = \frac{35}{a} \cdot z + z \cdot a - \frac{a + Et}{4}$
15	$A0 = \frac{35}{G\_1} \cdot Zvcw + G\_1 \cdot a - \frac{a + Zvcw}{a}$
16	$Zxy = a\_5 + b - a\_5 \cdot (b + y)$
17	$ch = \frac{6}{f1} \cdot z + z \cdot f1 - \frac{f1 + Et}{6}$
18	$ch = rx \cdot z + z \cdot f1 - \frac{f1 + Et}{rx}$

19	$\_H = k1 + k2 - k3 + \frac{k1 \cdot k2}{k3}$
20	$Fy = a\_1 + b - a\_1 \cdot (x + y)$
21	$\_G\_1 = y1 + y2 + y3 - \frac{y1 + y2 + y3}{3}$
22	$R = 2 \cdot x + 2 \cdot y - 4 \cdot x \cdot y + z$
23	$g11 = t \cdot z + z \cdot f1 - \frac{f1 + U}{t}$
24	$Y = Z \cdot (z + x) + z / Z - \frac{z + Z}{x + X}$
25	$U = rx \cdot z + z \cdot Y - \frac{Y + z}{rx}$

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какая функция имеет особенное значение при выполнении программы на языках C, C++, C#?

2. Что такое «точка входа» в программе?

3. Как вы понимаете термины «пространства имен», «класс», «метод», «функция»?

4. Что такое переменная? Как объявляется переменная?

5. Как объявляется константа? Чем константа отличается от переменной?

6. Какие типы значений применяются C#?

7. Чем тип char отличается от типа string?

8. Как производится инициализация переменных? Как производится инициализация констант?

9. Что такое управляющие последовательности?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1-3].

ЛАБОРАТОРНАЯ РАБОТА 2. УПРАВЛЕНИЕ ПОТОКОМ ВЫПОЛНЕНИЯ В ПРОГРАММЕ

1. Цель и содержание

Цель лабораторной работы: изучить операторы, позволяющие организовывать непоследовательное выполнение программного кода.

Задачи лабораторной работы:

– научиться применять условныq оператор if .. else;

– научиться применять операторs цикла for, while, do … while;

– научиться применять операторы выбора switch.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

3.1 Условный оператор

В предыдущих лабораторных работах были рассмотрены вопросы программирования на языке C# с использованием только последовательного выполнения операторов программы.

В языке C# (как и в большинстве языков) можно вводить условия, циклы, ветвления.

Условный оператор в C# позволяет организовать условие типа «если …, то …, иначе …».

Синтаксис условного оператора:

if (*условие*)
    оператор
[else if (*условие*)
    оператор
...
else
    оператор]

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки). Условный оператор содержит только одно обязательное зарезервированное слово – if. Все остальные конструкции не являются обязательными (в синтаксическом описании на это указывают скобки [ ]).

Пример использования условного оператора:

```
int a = 100, b = 13, c = 23, d = 0, e = 0;


if (a == 100) // Если a равно 100
{
    // Увеличиваем b на 1000
    b += 1000;
}
else if (a < 100) // иначе если a меньше 100
{
    // изменяем значение d
    d = b + c;
    // и изменяем значение e
    e = a + b;
}
else // иначе (остальные варианты не предусмотренные первыми двумя условиями)
    // edtkbxbdftv a на 1
    a++;

// продолжается последовательное выполнение программы
Console.WriteLine("{0}", a);
```

В представленном примере сначала проверяется является ли значение переменной a равным 100.

1. Если «Да», то выполняется единственный оператор (увеличение b на 1000). На этом выполнение всего условного оператора завершено и программа переходит к выполнению операторов, следующих за ним, то есть к выводу значения a.

2. Если первое условие не выполнилось, проверяется второе – является ли a меньше 100 (конструкция else if). Если это утверждение истинно, то выполняется составной оператор, состоящий из двух – изменение значений переменных d и e. Затем осуществляется переход к выводу переменной a.

3. Если не выполнилось ни одно условие с оператором if, то выполняется оператор, указанный после зарезервированного слова else. В данном случае это несоставной оператор (a увеличивается на 1), поэтому фигурные скобки можно не писать.

Во всей этой конструкции только оператор if является обязательным. Конструкций else if может быть любое количество, а if и else – только по одному.

Еще один пример условного оператора:

```
int a = 100;

if (a >= 100)
{
    Console.WriteLine("Значение переменной a больше или равно 100");
}
```

В данном случае вывод в консоль выполнится только если a больше либо равно 100.Иначе условный оператор ничего не выведет. Фигурные скобки в данном примере можно опустить.

Обратите внимание, что для проверки на равенство используется оператор ==, а не =. Знак «равно» используется в C# для присваивания значений. Следует понимать, что условное выражение, стоящее в конструкции if должно возвращать булево значение.

Например, следующий условный оператор всегда будет выполняться:

```
if (true)
    Console.WriteLine("Всегда выводится");
else
    Console.WriteLine("Никогда не выводится");
```

3.2 Оператор выбора

Синтаксис оператора:

```
switch (перем)
{
   case константа1:
      оператор_1;
      break;
   case константа2:
      оператор_2;
      break;
...
   default:
      оператор_n;
      break;
}
```

В данном операторе осуществляется выбор оператора (который может быть составным) в зависимости от значения переменной *перем*. Если *перем* равна значению *константа*1, то выполнится *оператор*_1, если *константа*2 – *оператор*_2, и т.д. Если ни одно значение, указанное в каком-либо операторе case, не совпало со значением *перем*, то выполнится оператор, указанный в секции default . Внутри каждой секции case следует указать оператор break, который предотвращает проверку других условий после выполнения данного оператора. Следует обратить внимание, что в секциях case следует использовать только константы (переменные не допускаются).

Пример использование оператора switch … case:

```
// Организация ответа на действие пользователя:
string text = "1 - Вывод группы \n" +
    "2 - Вывод фамилии преподавателя \n" +
    "3 - Вывод названия предмета \n" +
    "4 - Вывод приветствия \n";

Console.Write(text + "Введите команду > ");
int result = Convert.ToInt32(Console.ReadLine());

switch (result)
{
    case 1: { Console.WriteLine("ИСТБ - 101"); break; }
    case 2: Console.WriteLine("Николаев Евгений Иванович"); break;
    case 3: { Console.WriteLine("Технологии программирования"); break; }
    case 4: Console.WriteLine("Привет !"); break;
    default: Console.WriteLine("Недопустимый вариант !!!"); break;
}
```

Изучите представленный пример самостоятельно.

3.3 Цикл for

Циклы позволяют выполнять определенную последовательность операторов до тех пор, пока выполняется некоторое условие. Каждый «круг» выполнения этого блока называется итерацией.

Синтаксис оператора:

for (*инициализатор*; *условие*; *итератор*)
    оператор

инициализатор – выражение, выполняемое перед первой итерацией цикла.

условие – выражение булевого типа, которое проверяется перед каждой итерацией. Если оно истинно то итерация выполняется, иначе – цикл завершается.

итератор – выражение, вычисляемое после каждой итерации.

Пример применения оператора цикла for (пример 1.1):

```
/*
 * Вычислить сумму чисел от 1 до 1000
 */

// Объявляем переменную, которая хранит значение суммы чисел
System.Int32 Sum = 0;
// Объявляем счетчик цикла
int i;

for (i = 1; i <= 1000; i++)
    Sum += i;

Console.WriteLine("Итог: {0}", Sum);
```

Изучите код самостоятельно (создайте такую программу в VS). Еще один пример для самостоятельного изучения (пример 1.2):

```
/*
* вычислить сумму всех чисел, делящихся на 3 без остатка
* и находящихся в диапазоне от 1 до 1000000
*/

long Sum = 0;
int i;

for (i = 1; i <= 1000000; i++)
    if (i % 3 == 0)
        Sum += i;
```

Пример, отображающий возможность применения итератора (пример 1.3):

```
/*
* вычислить сумму всех чисел, делящихся на 3 без остатка
* и находящихся в диапазоне от 1 до 1000000
*/

long Sum = 0;
int i;

for (i = 0; i <= 1000000; i += 3)
        Sum += i;
```

Обратите внимание, что примеры 2 и 3 решают одну и ту же задачу.

3.4 Цикл while

Цикл while похож на for тем, что является конструкцией с предварительной проверкой условия продолжения цикла. Но синтаксис цикла while более лаконичен:

while (*условие*)
    оператор

В данном синтаксисе *оператор* может быть составным оператором (группа операторов языка, заключенные в фигурные скобки).

Следует понимать, что while используется для заранее неизвестного количества повторных выполнений операторов.

Пример выполнения цикла (пример 2.1):

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = false;
int chislo;

while (!Stop)
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        Stop = true;
}
```

Оператор break уже встречался в конструкции case оператора switch для выхода из case. Но break часто применяется внутри циклов for, while и do … while. Данный оператор прерывает выполнение цикла и передает управление оператору, следующему за циклом.

Пример 2.2, выполняющий то же самое, что и пример 2.1:

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = true;
int chislo;

while(Stop)
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        break;
}
```

Следует обратить внимание, что цикл является бесконечным (логическая переменная Stop всегда является true), но выполнение цикла все равно прервется, если пользователь введет 0.

Оператор continue также применяется внутри цикла – он останавливает выполнение текущей итерации и немедленно переходит к выполнению следующей итерации.

3.5 Цикл do … while

Цикл do … while полностью повторяет функциональные возможности while, но предполагает проверку условия окончания цикла после выполнения тела цикла. То есть блок операторов тела цикла всегда выполнится хотя бы один раз.

Синтаксис оператора:

do
{
    оператор (*операторы*)
}
while (*условие*);

Пример использования цикла (пример 3.1):

```
/*
 * Цикл будет выполняться пока пользователь не введет 0
 */

bool Stop = false;
int chislo;

do
{
    Console.Write("Введите число > ");
    chislo = Convert.ToInt32(Console.ReadLine());

    if (chislo == 0)
        Stop = true;
} while (!Stop);
```

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Выполните индивидуальное задание. Пользователю предлагается выбрать метод расчета значения выражения: 1) с помощью цикла for; 2) с помощью цикла while; или 3) с помощью цикла do ... while. После выбора способа расчета, запрашиваются входные данные и производится расчет с использованием данного вида цикла.

3. Во всех заданиях переменные X, Y являются вещественными и вводятся пользователем. Количество слагаемых также вводится пользователем. Программа должна вывести сумму заданного числа членов последовательности.

Индивидуальное задание.

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Выражение для вычисления
1	$j = -\frac{\sin^3(Y)}{1\cdot 3} + \frac{\sin^5(X^2)}{3\cdot 5} - \frac{\sin^7(Y^3)}{5\cdot 7} + \frac{\sin^9(X^4)}{7\cdot 9} - \ldots$

2	$Z=\frac{1}{1\cdot3\cdot5}-\frac{X}{2\cdot4\cdot6}+\frac{Y^2}{3\cdot5\cdot7}-\frac{X^3}{4\cdot6\cdot8}+\frac{Y^4}{5\cdot7\cdot9}-...$
3	$Z=\frac{X^2}{1\cdot3}-\frac{Y^4}{3\cdot5}+\frac{X^6}{5\cdot7}-\frac{Y^8}{7\cdot9}+...$
4	$Z=\frac{Y^2\cdot X}{2}-\frac{Y^2\cdot X^4}{4}+\frac{X^4\cdot Y^6}{6}-\frac{X^8\cdot Y^6}{8}+...$
5	$A=-\frac{\sin(X)\cdot\lg(Y)}{1!}+\frac{\ln(X^3)\cdot\cos(Y^2)}{3!}-\frac{\sin(X^5)\cdot\lg(Y^3)}{5!}+\frac{\ln(X^7)\cdot\cos(Y^4)}{7!}-...$
6	$Z=1-\frac{X}{2}+\frac{Y^2}{6}-\frac{X^3}{24}+\frac{Y^4}{120}-...$ (в знаменателе факториал)
7	$s=-\frac{Y\cdot X^2}{1\cdot3}+\frac{X\cdot Y^3}{2\cdot4}-\frac{Y\cdot X^4}{3\cdot5}+\frac{X\cdot Y^5}{4\cdot6}-...$
8	$Z=1-\frac{\sin(X^2)}{2}+\frac{\cos(Y^3)}{3}-\frac{\sin(X^4)}{4}+\frac{\cos(Y^5)}{5}-...$
9	$Z=\frac{1}{1\cdot3}-\frac{X}{2\cdot4}+\frac{X^2}{3\cdot5}-\frac{X^3}{4\cdot6}+...$
10	$Z=\frac{X^2}{1\cdot3}-\frac{Y^4}{3\cdot5}+\frac{X^6}{5\cdot7}-\frac{Y^8}{7\cdot9}+...$
11	$p=-\frac{X\cdot\sin(Y)}{1\cdot2}+\frac{X^3\cdot\cos(Y^2)}{3\cdot4}-\frac{X^5\cdot\sin(Y^3)}{5\cdot6}+\frac{X^7\cdot\cos(Y^4)}{7\cdot8}-...$
12	$p=-\frac{X}{1\cdot2\cdot3}+\frac{Y^3}{3\cdot4\cdot5}-\frac{X^5}{5\cdot6\cdot7}+\frac{Y^7}{7\cdot8\cdot9}-...$
13	$Z=1-\frac{\sin(X^2)+Y}{2}+\frac{\cos(Y^3)+X}{3}-\frac{\sin(X^4)+Y}{4}+\frac{\cos(Y^5)+X}{5}-...$
14	$Z=\frac{Y^2+X}{1\cdot2}-\frac{Y^2-X^4}{2\cdot4}+\frac{X^4+Y^6}{4\cdot6}-\frac{X^8-Y^6}{6\cdot8}+...$
15	$p=1-\frac{\cos(X)\cdot\sin^2(Y)}{1\cdot2}+\frac{\cos^4(X)\cdot\sin^3(Y)}{3\cdot4}-\frac{\cos^5(X)\cdot\sin^6(Y)}{5\cdot6}+...$
16	$s=-\frac{\sin(Y)\cdot X^2}{1\cdot3}+\frac{X\cdot\cos(Y^3)}{2\cdot4}-\frac{\sin(Y)\cdot X^4}{3\cdot5}+\frac{X\cdot\cos(Y^5)}{4\cdot6}-...$
17	$Z=-\frac{tg(Y)\cdot ctg(X^2)}{1\cdot3}+\frac{tg(X)\cdot ctg(Y^3)}{2\cdot4}-\frac{tg(Y)\cdot ctg(X^4)}{3\cdot5}+\frac{tg(X)\cdot ctg(Y^5)}{4\cdot6}-...$
18	$Z=\frac{1}{1!}-\frac{X}{2!}+\frac{Y^2}{3!}-\frac{X^3}{4!}+\frac{Y^4}{5!}-...$
19	$Z=\frac{\cos(X)+\lvert Y\rvert}{1!}-\frac{\cos^2(Y)+\lvert X\rvert}{2!}+\frac{\cos^3(X)+\lvert Y\rvert}{3!}-\frac{\cos^4(X)+\lvert Y\rvert}{4!}+...$

20	$Z = \frac{Y^2 \cdot X}{2+1} - \frac{Y^2 \cdot X^4}{4-2} + \frac{X^4 \cdot Y^6}{6+4} - \frac{X^8 \cdot Y^6}{8-6} + \ldots$
21	$Z = 1 + \frac{1-\cos(Y)}{1 \cdot 3 \cdot 5} - \frac{Y+\cos^2(X)}{2 \cdot 4 \cdot 6} + \frac{X^2-\cos^3(Y)}{3 \cdot 5 \cdot 7} - \frac{Y^3+\cos^4(X)}{4 \cdot 6 \cdot 8} + \ldots$
22	$Z = \frac{X-Y}{1 \cdot 3} - \frac{Y^2-X^2}{2 \cdot 4} + \frac{X^3-Y^3}{3 \cdot 5} - \frac{Y^4-X^4}{4 \cdot 6} + \ldots$
23	$Z = 1 + \frac{1-\cos(Y)}{1 \cdot 3 \cdot 5} - \frac{Y+\cos^2(X)}{2 \cdot 4 \cdot 6} + \frac{X^2-\cos^3(Y)}{3 \cdot 5 \cdot 7} - \frac{Y^3+\cos^4(X)}{4 \cdot 6 \cdot 8} + \ldots$
24	$Z = \frac{(X-Y)^2}{1 \cdot 3} - \frac{(Y-X)^4}{3 \cdot 5} + \frac{(X-Y)^6}{5 \cdot 7} - \frac{(Y-X)^8}{7 \cdot 9} + \ldots$
25	$Z = \frac{Y^2 \cdot \sin(X)}{2} - \frac{\cos(Y^2) \cdot X^4}{4} + \frac{\sin(X^4) \cdot Y^6}{6} - \frac{\cos(Y^6) \cdot X^8}{8} + \ldots$

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Как оформляется комментарий?

2. Опишите синтаксис условного оператора. Какие части данного оператора являются обязательными?

3. Опишите синтаксис оператора выбора. Поясните почему следует использовать оператор break внутри каждого выбора.

4. Можно ли с помощью for реализовать бесконечный цикл? Поясните ответ на примерах.

5. В разделе «Теоретическое обоснование» приведены примеры 1.1, 1.2 и 1.3. Почему в примерах 1.2 и 1.3 для переменной Sum выбран тип long, хотя в примере 1 для Sum выбран тип int? Обоснуйте ответ.

6. Опешите синтаксис оператора цикла for.

7. Опишите синтаксис оператора while.

8. Поясните назначение операторов break и continue.

9. Опешите синтаксис оператора do … while.

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2-4].

ЛАБОРАТОРНАЯ РАБОТА 3. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ

1. Цель и содержание

Цель лабораторной работы: изучить типы и принципы работы с массивами.

Задачи лабораторной работы:

– научиться работать с простыми массивами;

– научиться работать с многомерными массивами.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическое обоснование

3.1 Простые массивы

Для работы с набором значений одного типа используются массивы.

Массив – это структура данных, содержащая множество элементов одного и того же типа.

Массив определяется типом элементов, содержащимся в нем, за которым следуют квадратные скобки и имя переменной.

Примеры объявления массивов:

```
// Объявление массива целочисленных элементов
int[] mas_int;

// Объявление массива элементов типа double
double[] mas_d;

// Объявление массива элементов типа string
string[] MassivStr;
```

При объявлении массива не указывается его размер. Для выделения памяти для элементов массива необходимо использовать следующий синтаксис (для массивов, объявленных в предыдущем примере):

```
// Спецификация размера массивов
mas_d = new double[5];
mas_int = new Int32[100];
MassivStr = new string[100];
```

После данной инициализации можно обращаться к элементам массивов.

После такой инициализации нельзя изменять размер массива.

Объявить и инициализировать массив можно в одной строке:

```
float[] keof = new float[1000];
```

При инициализации можно также присвоить значения элементам массива (все три способа присвоения элементов массива эквивалентны):

```
int[] m1 = new int[5] {1, 23, 2, 100, 7};

int[] m2 = new int[] {23, 12, 100, 101, 0};

int[] m3 = {0, 1, 2, 56, 90};
```

Там, где программист не указал явно размер массива, компилятор определит его самостоятельно. Очевидно, что задать таким образом большое количество элементов неудобно. Для доступа к элементам массивов используется индексатор.

После того как массив объявлен и инициализирован, можно производить доступ к элементам массива для чтения и записи через индексатор. Индексатор – это целочисленный номер элемента массива.

Индексатор начинается с 0 (для первого элемента массива) и закачивается числом равным количеству элементов в массиве минус 1.

Например, определим массив и попробуем обратиться к его элементам:

```
// Объявление и инициализация массива
int[] mas = new int[4];

// Установка значений элементов массива
mas[0] = 1000; // для первого элемента
mas[1] = 45; // для второго элемента

// Считывание значений
mas[2] = mas[0]; // считываем знасчение первого и присваиваем его третьему элементу
int val = mas[1];
int a = mas[3]; // возникнет ОШИБКА! Т.к. четвертый элемент не инициализирован
int Oo = mas[4]; // возникнет ОШИБКА! Т.к. пятого элемента в массиве нет
```

Часто (когда размер массива большой) используют циклы для обращения к элементам массива:

```
/*
 * В цикле последовательно запрашиваются у пользователя
 * значения для инициализации элементов массива
 */

for (int i = 0; i < massiv.Length; i++)
{
    Console.Write("Введиите значение {0}-го элемента > ", i+1);
    massiv[i] = Convert.ToInt32(Console.ReadLine());
}
```

Для больших массивов есть смысл сгенерировать значения элементов массива случайным образом (используя генератор случайных чисел):

```
int[] Pro = new int[1000];

// Объявление переменной класса Random
Random rnd = new Random(1);

int i = 0;

while (i < Pro.Length)
{
    // Вызов метода Next класса Random
    Pro[i] = rnd.Next();
    i++;
}

// Вывод элементов массива в обратном порядке
for (i = Pro.Length-1; i >= 0; i--)
    Console.WriteLine("Элемент {0} равен {1}", i, Pro[i]);
```

Внимательно изучите представленный код.

3.2 Многомерные массивы

Обычные массивы (одномерные) индексируются одним числом. Многомерные массивы индексируются несколькими индексаторами. Очевидно, что двумерный массив можно отождествлять с матрицей, трехмерный – с кубом.

Принципы объявления, инициализации и обращения к элементам остаются теми же, что и для одномерных массивов:

```
// Объявляем и инициализируем двумерный массив
// - матрицу целых размером 3x3
int[,] matrix = new int[3, 3];

// Присваиваем значение диаганальным элементам
matrix[0, 0] = 1;
matrix[1, 1] = 120;
matrix[2, 2] = 8;

// Объявляем матрицу с одновременным присвоением
// значений элементов
int[,] mm = {
                {0, 23, 100},
                {6, 12, 23},
                {0, 0, 5}
            };
// Объявляем трехмерный массив
double[, ,] opr = new double[100, 100, 50];

// Присваиваем значение 34 элементу с индексами (0, 0, 0)
opr[0, 0, 0] = 34;
```

Как и в случае одномерных массивов, для работы с многомерными применяются циклы.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться

электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение.
2. Изучите примеры, представленные в разделе «Теоретическое обоснование» данной лабораторной работы, повторите теоретическую часть лабораторной работы 2 (циклические конструкции).
3. Выполните индивидуальное задание. Задания ориентированы на работу с одномерными и многомерными массивами.

Индивидуальное задание

Для инициализации исходной матрицы необходимо использовать генератор случайных чисел. Следует использовать ограничение на генерируемые числа, например, ограничить значения случайно генерируемых чисел величиной 100, 10 и т.д., чтобы вывод программы был более читабельным. Алгоритм решения должен работать для массивов любой величины, поэтому размер исходного массива запрашивается у пользователя.

Вариант	Выражение для вычисления
1	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие максимального, заменены на 0.
2	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие среднего арифметического в данной строке, заменены на минимальный элемент матрицы.
3	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое элементов матрицы и результирующую матрицу, в которой все элементы, среднее арифметическое матрицы, заменены на 0.

4	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждой строки и результирующую матрицу, в которой все четные элементы заменены на среднее арифметическое в данной строке.
5	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждого столбца и результирующую матрицу, в которой все элементы, заменяются по правилу: четные элементы заменяются на 1, нечетный на 0.
6	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент для каждой строки и результирующую матрицу, в которой все элементы, которые делятся на 3, заменены на минимальный элемент во всей матрице.
7	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент для каждого столбца и результирующую матрицу, которая является транспонированной по отношению к исходной.
8	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент делится на 2, то вывести 2; - если элемент делится на 3, то вывести 3; - если не выполняются предыдущие правила, то элемент не заменяется.
9	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент меньше чем среднее арифметическое строки, то выводится 0; - если элемент больше чем среднее арифметическое строки, то выводится 1; - если не выполняются предыдущие правила, то элемент не заменяется.
10	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный и максимальный элементы в матрице (и их индексы) и результирующую матрицу, которая является транспонированной по отношению к исходной.
11	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие максимального, заменены на 0.
12	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие среднего арифметического в данной строке, заменены на минимальный элемент матрицы.

13	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое элементов матрицы и результирующую матрицу, в которой все элементы, среднее арифметическое матрицы, заменены на 0.
14	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждой строки и результирующую матрицу, в которой все четные элементы заменены на среднее арифметическое в данной строке.
15	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждого столбца и результирующую матрицу, в которой все элементы, заменяются по правилу: четные элементы заменяются на 1, нечетный на 0.
16	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент для каждой строки и результирующую матрицу, в которой все элементы, которые делятся на 3, заменены на минимальный элемент во всей матрице.
17	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент для каждого столбца и результирующую матрицу, которая является транспонированной по отношению к исходной.
18	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент делится на 2, то вывести 2; - если элемент делится на 3, то вывести 3; - если не выполняются предыдущие правила, то элемент не заменяется.
19	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент меньше чем среднее арифметическое строки, то выводится 0; - если элемент больше чем среднее арифметическое строки, то выводится 1; - если не выполняются предыдущие правила, то элемент не заменяется.
20	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный и максимальный элементы в матрице (и их индексы) и результирующую матрицу, которая является транспонированной по отношению к исходной.
21	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие максимального, заменены на 0.

22	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие среднего арифметического в данной строке, заменены на минимальный элемент матрицы.
23	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое элементов матрицы и результирующую матрицу, в которой все элементы, среднее арифметическое матрицы, заменены на 0.
24	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждой строки и результирующую матрицу, в которой все четные элементы заменены на среднее арифметическое в данной строке.
25	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждого столбца и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент делится на 2, то вывести 2; - если элемент делится на 5, то вывести 5; - если не выполняются предыдущие правила, то вывести 0.

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое массив?

2. Как объявляется массив, как он инициализируется?

3. Как происходи обращение к элементам массива?

4. Как реализовать в программе работу с генератором случайных чисел?

5. Поясните, как связаны величины: количество элементов в массиве, максимальное значение индексатора массива, минимальное значение индексатора массива.

6. В чем преимущества и недостатки непосредственного задания значений элементов массива при его объявлении? В чем недостатки такого подхода?

7. Как изменить размер объявленного и инициализированного массива?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1-4].

ЛАБОРАТОРНАЯ РАБОТА 4. КЛАССЫ. ОБЪЕКТНОЕ МОДЕЛИРОВАНИЕ

1. Цель и содержание

Цель лабораторной работы: изучить структуру и принципы объявления классов, освоить технологию создания экземпляров классов (объектов).

Задачи лабораторной работы:

– научиться объявлять классы;

– научиться создавать объекты классов;

– научиться работать с полями данных и методами классов.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

3.1 Классы и структуры

Класс – это тип данных, объединяющий данные и методы их обработки. Класс – это пользовательский шаблон, в соответствии с которым можно создавать объекты. То есть класс – это правило, по которому будет строиться объект. Сам класс не содержит данных.

Объект класса (экземпляр класса) – переменная типа класс. Объект содержит данные и методы, манипулирующие этими данными. Класс определяет, какие данные содержит объект и каким образом он ими манипулирует.

Все что справедливо для классов можно распространить и на структуры. Отличие состоит в методе хранения объектов данных типов в оперативной памяти: структуры – это типы по значению, они размещаются в стеке; классы – это

ссылочные типы, объекты классов размещаются в куче. Структуры не поддерживают наследование.

Структуры применяются для представления небольших объемов данных. Объявление структур происходит с использованием ключевого слова struct, объявление классов – с помощью ключевого слова class.

Пример объявления класса:

```
// Объявление класса
public class MyFirstClass
{
    // Данные-члены класса

    // Доступные на уровне экземпляра
    public int a;
    public float b__;
    public string fio;

    // Доступные только на уровне класса
    private bool IsOK;
    private double precision;
}
```

При создании как классов, так и структур, используется ключевое слово new, например:

```
MyFirstClass obj = new MyFirstClass();
```

3.2 Структура класса

Данные и функции, объявленные внутри класса, называются членами класса (class members). Доступность членов класса может быть описана как public, private, protected, internal или internal protected.

3.2.1 Данные-члены

Данные-члены – это те структуры внутри класса, которые содержат данные класса – поля, константы события.

Поля – это любые переменные, ассоциированные с классом. После создания экземпляра класса к полям можно обращаться с использованием синтаксиса , например: ИмяПоля ИмяОбъекта.

```
MyFirstClass obj = new MyFirstClass();

obj.a = 5;
int cc = obj.a;

obj.fio = "Novak E.I.";
```

Аналогичным образом с классом ассоциируются константы.

События будут рассмотрены в следующих лабораторных работах.

3.2.2 Функции-члены

Функции-члены – это члены, которые обеспечивают некоторую функциональность для манипулирования данными классов. Они делятся на следующие виды: методы, свойства, конструкторы, финализаторы, операции и индексаторы.

Методы (method) – это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

Свойства (property) – это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса. В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится. Поскольку не существует какого-то отдельного синтаксиса для свойств, который отличал бы их от нормальных функций, создается иллюзия объектов как реальных сущностей, предоставляемых клиентскому коду.

Конструкторы (constructor) – это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

Финализаторы (finalizer) похожи на конструкторы, но вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды (~). Предсказать точно, когда будет вызван финализатор, невозможно.

Операции (operator) – это простейшие действия вроде + или -. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым. Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операций).

Индексаторы (indexer) позволяют индексировать объекты таким же способом, как массив или коллекцию.

В данной лабораторной работе рассматриваются только методы класса – это функции, ассоциированные с определенным классом.

В C# объявление метода класса состоит из спецификатора доступности, возвращаемого значения, имени метода, списка формальных параметров и тела метода:

```
[модификатор] тип_возврата имя_метода ([список_параметров])
{
    // тело метода
}
```

Например, добавим методы для объявленного ранее класса MyFirstClass:

```
// Объявление класса
public class MyFirstClass
{
    // Данные-члены класса

    // Доступные на уровне экземпляра
    public int a;
    public float b__;
    public string fio;

    // Доступные только на уровне класса
    private bool IsOK;
    private double precision;

    // Метод для инициализации некоторых полей класса
    public void InitClassMembers(int pA, float pB__, string pFio)
    {
        a = pA;
        b__ = pB__;
        fio = pFio;
    }

    // Метод, возвращающий значение вычисленное на основе полей класса
    public int GetAbsA()
    {
        return Math.Abs(a);
    }
}
```

Синтаксис вызова методов аналогичен синтаксису обращения к данным-членам:

```
MyFirstClass obj = new MyFirstClass();

obj.InitClassMembers(10, 0.8F, "Новиков П.Е.");

int abs_a = obj.GetAbsA();
```

В данном примере метод InitClassMembers не возвращает никаких данных, но требует передачи ему фактических параметров. В свою очередь, метод GetAbsA возвращает значение типа int и не предполагает никаких параметров.

В общем случае параметры могут передаваться методу либо по значению, либо по ссылке. Когда переменная передается по ссылке, вызываемый метод получает саму переменную, поэтому любые изменения, которым она подвергнется внутри метода, останутся в силе после его завершения. Но если переменная передается по значению, вызываемый метод получает копию этой переменной, а это значит, что все изменения в ней по завершении метода будут утеряны. Для сложных типов данных передача по ссылке более эффективна из-за большого объема данных, который приходится копировать при передаче по значению.

Если не указано обратное, то в C# все параметры передаются по значению. Тем не менее, можно принудительно передавать значения по ссылке, для чего используется ключевое слово ref. Если параметр передается в метод, и входной аргумент этого метода снабжен префиксом ref, то любые изменения этой переменной, которые сделает метод, отразятся на исходном объекте.

В C-подобных языках функции часто возвращают более одного значения. Это обеспечивается применением выходных параметров, за счет присваивания значений переменным, переданным в метод по ссылке. Часто первоначальное значение таких переменных не важно. Эти значения перезаписываются в функции, которая может даже не обращать внимания на то, что в них хранилось первоначально.

Было бы удобно использовать то же соглашение в C#. Однако в C# требуется, чтобы переменные были инициализированы каким-то начальным значением перед

тем, как к ним будет выполнено обращение. Хотя можно инициализировать входные переменные какими-то бессмысленными значениями до передачи их в функцию, которая наполнит их осмысленными значениями, этот прием выглядит в лучшем случае излишним, а в худшем – сбивающим с толку. Тем не менее, существует способ обойти требование компилятора C# относительно начальной инициализации переменных.

Это достигается ключевым словом out. Когда входной аргумент снабжен префиксом out, этому методу можно передать неинициализированную переменную. Переменная передается по ссылке, поэтому любые изменения, выполненные методом в переменной, сохраняются после того, как он вернет управление. Ключевое слово out также должно указываться при вызове метода – так же, как при его определении.

Частичные классы.

Ключевое слово partial (частичный) позволяет определить класс, структуру или интерфейс, распределенный по нескольким файлам. Но ситуации, когда множеству разработчиков требуется доступ к одному и тому же классу, или же в ситуации, когда некоторый генератор кода генерирует часть класса, такое разделение класса на несколько файлов может оказаться полезным. Ключевое слово partial просто помещается перед классом, структурой или интерфейсом.

Статические классы.

Статический класс функционально представляет собой то же самое, что и класс с приватным статическим конструктором. Создать экземпляр такого класса невозможно. Если указать ключевое слово static в объявлении класса, компилятор будет гарантировать, что к этому классу никогда не будут добавлены нестатические члены.

Класс Object.

Rлассы .NET изначально унаследованы от System.Object. Фактически, если при определении нового класса базовый класс не указан, компилятор автоматически предполагает, что он наследуется от Object.

Практическое значение этого в том, что помимо методов и свойств, которые программист определяет самостоятельно, также появляется доступ к множеству общедоступных и защищенных методов-членов, которые определены в классе Object. Эти методы присутствуют во всех определяемых классах.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение в среде Visual Studio.
2. Изучите пример выполнения задания, представленный в данном разделе.

3. Выполните индивидуальное задание. Задания ориентированы на работу с классами.

Пример выполнения задания.

Разработать класс для представления объекта «Прямоугольный параллелепипед». Реализуйте все необходимые поля данных (закрытые) и методы позволяющие:

– устанавливать и считывать значения полей данных;

– вычислять объем прямоугольного параллелепипеда;

– вычислять площадь поверхности прямоугольного параллелепипеда;

– выводить полную информацию об объекте в консоль.

Решение данной задачи состоит из двух этапов: объявление класса Parallelepiped и демонстрация использования объекта данного класса.

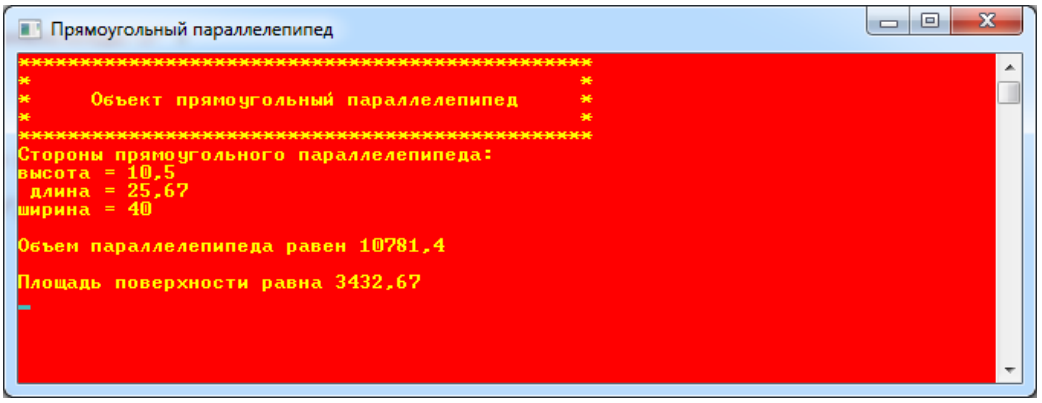
Полный листинг примера:

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

namespace LR_Three
{
    class Parallelepiped
    {
        // Поля данных представляют стороны параллелепипеда
        private double a, b, c;

        // Методы для установки и считывания значений полей
        public void Set_a(double pa) { a = pa; }
        public double Get_a() { return a; }

        public void Set_b(double pb) { b = pb; }
        public double Get_b() { return b; }

        public void Set_c(double pc) { c = pc; }
        public double Get_c() { return c; }


        // Метод для вычисления объема прямоугольного параллелепипеда
        public double GetV()
        {
            return a * b * c;
        }

        // Метод для вычисления площади поверхности прямоугольного параллелепипеда
        public double GetS()
        {
            return 2 * (a * b + b * c + a * c);
        }
```

```
        // Метод для вывода полной информации об объектк в консоль
        public void PrintFullInformation()
        {
            string str =    "*********************************************\n" +
                            "*                                           *\n" +
                            "*     Объект прямоугольный параллелепипед   *\n" +
                            "*                                           *\n" +
                            "*********************************************";
            Console.WriteLine(str);

            Console.WriteLine("Стороны прямоугольного параллелепипеда:\n" +
                "высота = {0}\n "+
                "длина = {1}"+
                "ширина = {2}", a, b, c);

            Console.WriteLine("Объем параллелепипеда равен {0}", GetV());

            Console.Write("Площадь поверхности равна {0}", GetS());
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Прямоугольный параллелепипед";
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.Red;
            Console.Clear();

            Parallelepiped p;

            p = new Parallelepiped();
            p.Set_a(10.5);
            p.Set_b(25.67);
            p.Set_c(40);

            p.PrintFullInformation();

            Console.ReadKey();
        }
    }


}
```

В данном примере необходимо обратит внимание на тот факт, что все вычисления выполняются внутри класса. Метод Main содержит только вызовы методов класса, то есть вся реализация скрыта.

В результате выполнения программы отобразится следующее консольное окно с выводом информации:



Индивидуальное задание.

Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса.

Вариант	Выражение для вычисления
1.	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
2.	Класс «Куб». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, длины диагонали, а также вывод информации об объекте.
3.	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
4.	Класс «Точка в пространстве». Реализовать ввод и вывод полей данных, вычисление расстояния до введенной пользователем точки, расстояния от начала координат, а также вывод информации об объекте.
5.	Класс «График y=x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
6.	Класс «Шар». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
7.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод матрицы, нахождение максимального и минимального элементов, а также вывод информации об объекте.
8.	Класс «Прямоугольный треугольник». Реализовать ввод и вывод полей данных, вычисление гипотенузы, площади и периметра, а также вывод информации об объекте.

9.	Класс «Отрезок». Реализовать ввод и вывод полей данных (координаты начала и координаты конца отрезка), вычисление длины, расстояний начала и конца отрезка от начала координат, а также вывод информации об объекте.
10.	Класс «Цилиндр». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
11.	Класс «Ромб». Реализовать ввод и вывод полей данных (диагонали ромба), вычисление площади, периметра, а также вывод информации об объекте.
12.	Класс «Точка в пространстве». Реализовать ввод и вывод полей данных, вычисление расстояния до введенной пользователем точки, расстояния от начала координат, а также вывод информации об объекте.
13.	Класс «График y=3x+5». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
14.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод транспонированной матрицы, нахождение среднего арифметического всех элементов, а также вывод информации об объекте.
15.	Класс «Отрезок в пространстве». Реализовать ввод и вывод полей данных (координаты начала и координаты конца отрезка), вычисление длины, расстояний начала и конца отрезка от начала координат, а также вывод информации об объекте.
16.	Класс «График y=x-10». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
17.	Класс «Матрица MxN». Реализовать инициализацию элементов матрицы случайными числами, вывод транспонированной матрицы, нахождение и вывод среднего арифметического элементов в каждом столбце.
18.	Класс «Куб». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, длины диагонали, а также вывод информации об объекте.
19.	Класс «Сфера». Реализовать ввод и вывод полей данных, вычисление объема, диаметра и площади поверхности, а также вывод информации об объекте.
20.	Класс «Точка в пространстве». Реализовать ввод и вывод полей данных, вычисление расстояния до введенной пользователем точки, расстояния от начала координат, а также вывод информации об объекте.

21.	Класс «График y=x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
22.	Класс «Точка в пространстве». Реализовать ввод и вывод полей данных, вычисление расстояния до введенной пользователем точки, расстояния от начала координат, а также вывод информации об объекте.
23.	Класс «График y=-x». Реализовать ввод и вывод полей данных, вычисление интеграла функции от a до b (вводятся пользователем), длины отрезка функции от (a, y(a)) до (b, y(b)), а также вывод информации об объекте.
24.	Класс «Цилиндр». Реализовать ввод и вывод полей данных, вычисление объема, площади поверхности, а также вывод информации об объекте.
25.	Класс «Отрезок в пространстве». Реализовать ввод и вывод полей данных (координаты начала и координаты конца отрезка), вычисление длины, расстояний начала и конца отрезка от начала координат, а также вывод информации об объекте.

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое класс?

2. Что такое структура? Чем структура отличается от класса?

3. Что такое члены класса? Какие группы членов класса вы знаете?

4. Какие типы членов-данных вы знаете?

5. Какие типы функций-членов класса вы знаете?

6. Как поменять цвет текста в консольном приложении?

7. Как поменять цвет фона в консольном приложении?

8. Какие модификаторы доступности членов класса вы знаете?

9. Какое ключевое слово используется для создания объекта класса?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1, 2].

ЛАБОРАТОРНАЯ РАБОТА 5. КОНСТРУКТОР КЛАССА. ПЕРЕГРУЗКА КОНСТРУКТОРОВ КЛАССА.

1. Цель и содержание

Цель лабораторной работы: понять принципы работы конструктора.

Задачи лабораторной работы:

– научиться объявлять конструктор класса;

– научиться создавать перегруженные конструкторы.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

Конструктор – это метод класса, который не возвращает значения и имеет то же самое имя, что и класс. Если конструктор класса не определен программистом явно, то компилятор создаст конструктор по умолчанию.

Конструкторы подчиняются тем же правилам перегрузки, что и все методы.

В C# поддерживается перегрузка методов – то есть может существовать несколько версий одного метода, но с разными сигнатурами (методы отличаются количеством и / или типом параметров). Чтобы перегрузить метод, просто объявляются методы с одинаковыми именами, но разными сигнатурами.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше,

свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Для выполнения лабораторной работы необходимо модифицировать приложение, полученное в результате выполнения индивидуального задания лабораторной работы №4.

2. Модификация сводится к следующему: необходимо объявить и продемонстрировать использование трех-четырех перегруженных конструкторов класса.

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое класс?
2. Что такое конструктор класса?
3. Что такое перегрузка методов?
4. Может ли один конструктор класса вызывать другой конструктор? Прежде чем отвечать попробуйте реализовать такой вызов в своем разработанном классе.

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [1-4].

ЛАБОРАТОРНАЯ РАБОТА 6. ПРОЕКТИРОВАНИЕ ИЕРАРХИИ КЛАССОВ

1. Цель и содержание

Цель лабораторной работы: изучить механизм организации наследования классов.

Задачи лабораторной работы:

– научиться объявлять производные классы;

– научиться создавать иерархии классов;

– научиться использовать механизм полиморфизма.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

3.1 Наследование реализации

Наследование реализации (implementation inheritance) означает, что тип происходит от базового типа, получая от него все поля-члены и функции-члены.

Синтаксис наследования реализации:

```
class ПроизводныйКласс : БазовыйКласс
{
    // Данные – члены и функции – члены
}
```

Если при определении класса не указан базовый класс, то C# предполагает, что базовым классом является System.Object.

3.2 Создание иерархии классов

При наследовании реализации производный класс наследует реализацию каждой функции базового типа, если только в его определении не указано, что реализация функции должна быть переопределена.

Определим следующую иерархию классов (рис. 6.1) и продемонстрируем, как **наследуется реализация** и как **переопределяются** свойств и методов.

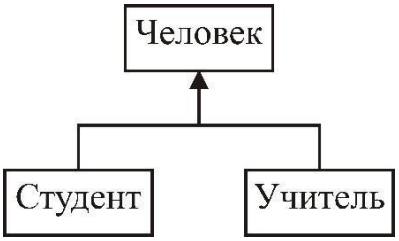


Рисунок 6.1 – Иерархия классов.

Создадим код, описывающий данную иерархию. Определим базовый класс:

```
class Человек
{
    public Человек(string Ф, string И, string О, int Возр)
    {
        Фамилия = Ф; Имя = И; Отчество = О; Возраст = Возр;
    }

    public Человек()
    {
        Фамилия = "нет данных"; Имя = ""; Отчество = "";
        Возраст = 18;
    }

    public string Фамилия, Имя, Отчество;
    private DateTime ДатаРождения;

    public virtual string ФИО
    {
        get { return Фамилия + " " + Имя + " " + Отчество; }
    }

    public int Возраст
    {
        get { return DateTime.Now.Year - ДатаРождения.Year; }
        set
        {
            int ГодРождения = DateTime.Now.Year - value;

            ДатаРождения = Convert.ToDateTime(ГодРождения.ToString()+".01.01");
        }
    }
}
```

Обратите внимание на наличие двух конструкторов, механизм хранения возраста человека и ключевое слово virtual у свойства «ФИО». Ключевое слово virtual указывает, что данное свойство будет переопределено в производном классе.

Определим производный класс «Учитель»:

```
class Учитель: Человек
{
    public Учитель()
        :base()
    {
        УченоеЗвание = УченыеЗвания.Без_Звания;
        УченаяСтепень = УченыеСтепени.Без_Степени;
    }

    public Учитель(string Ф, string И, string О, int Возр, УченыеЗвания УЗ, УченыеСтепени УС)
        :base(Ф, И, О, Возр)
    {
        УченоеЗвание = УЗ;
        УченаяСтепень = УС;
    }

    public УченыеЗвания УченоеЗвание;
    public УченыеСтепени УченаяСтепень;

    public override string ФИО
    {
        get
        {
            return УченаяСтепень.ToString() + ", " + УченоеЗвание.ToString() + ", " + base.ФИО;
        }
    }
}
```

Следует обратить внимание на использование ключевого слова override у свойства «ФИО», которое указывает на то, что данное свойство имеет новую реализацию, отличающуюся от реализации базового класса.

Определим второй производный класс «Студент»:

```
class Студент: Человек
{
    public Студент(string Ф, string И, string О, int Возр, Специальности Спец)
        :base(Ф, И, О, Возр)
    {
        Специальность = Спец;
    }

    public Специальности Специальность;

    public override string ФИО
    {
        get
        {
            return base.ФИО + ", " + Специальность.ToString();
        }
    }
}
```

В обоих производных классах следует обратить внимание на реализацию конструкторов производных классов, использование ключевого слова base в коде свойства «ФИО» и при объявлении конструкторов.

Также интерес представляют типы данных, используемые для членов-данных: «Специальности», «УченыеЗвания», «УченыеСтепени». Вот определения данных типов-перечислений:

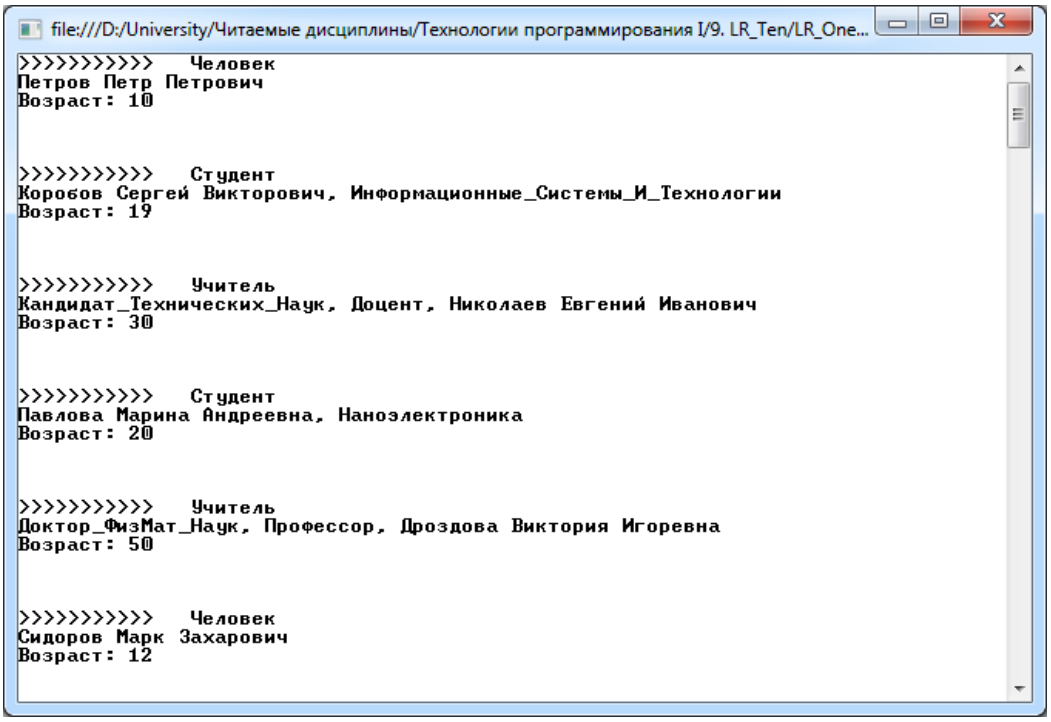
```
public enum УченыеЗвания
{
    Доцент,
    Профессор,
    Академик,
    Без_Звания
}

public enum УченыеСтепени
{
    Кандидат_Технических_Наук,
    Кандидат_ФизМат_Наук,
    Кандидат_Педагогических_Наук,
    Доктор_ФизМат_Наук,
    Без_Степени
}

public enum Специальности
{
    Информационные_Системы_И_Технологии,
    Безопасность_Информационных_Систем,
    Технология_Защиты_Информации,
    Психология,
    Наноэлектроника
}
```

Наконец, продемонстрируем использование объявленной иерархии классов.

В программе объявим массив объектов типа «Человек». Следует понимать, что при объявлении такого массива, его элементам можно присваивать объекты любого производного класса, причем каждый объект будет вести себя по-своему (за счет переопределения свойств и методов).

```
static void Main(string[] args)
{
    // Объявим массив людей:
    Человек[] mas = new Человек[6];

    // Заполним массив значениями разных (!) типов
    mas[0] = new Человек("Петров", "Петр", "Петрович", 10);

    mas[1] = new Студент("Коробов", "Сергей", "Викторович",
        19,
        Специальности.Информационные_Системы_И_Технологии);

    mas[2] = new Учитель("Николаев", "Евгений", "Иванович",
        30,
        УченыеЗвания.Доцент,
        УченыеСтепени.Кандидат_Технических_Наук);

    mas[3] = new Студент("Павлова", "Марина", "Андреевна",
        20,
        Специальности.Наноэлектроника);

    mas[4] = new Учитель("Дроздова", "Виктория", "Игоревна",
        50,
        УченыеЗвания.Профессор,
        УченыеСтепени.Доктор_ФизМат_Наук);

    mas[5] = new Человек("Сидоров", "Марк", "Захарович", 12);

    // Вывод инфорации
    for (int i = 0; i < mas.Length; i++)
    {
        Console.WriteLine(">>>>>>>>>>>    " + mas[i].GetType().Name);
        Console.WriteLine(mas[i].ФИО);
        Console.WriteLine("Возраст: " + mas[i].Возраст.ToString()+"\n");
        Console.WriteLine("\n");
    }

        Console.ReadKey();
}
```

В результате работы программы будет осуществлен следующий вывод:



Полная диаграмма типов в полученном приложении показана на рис. 6.2.

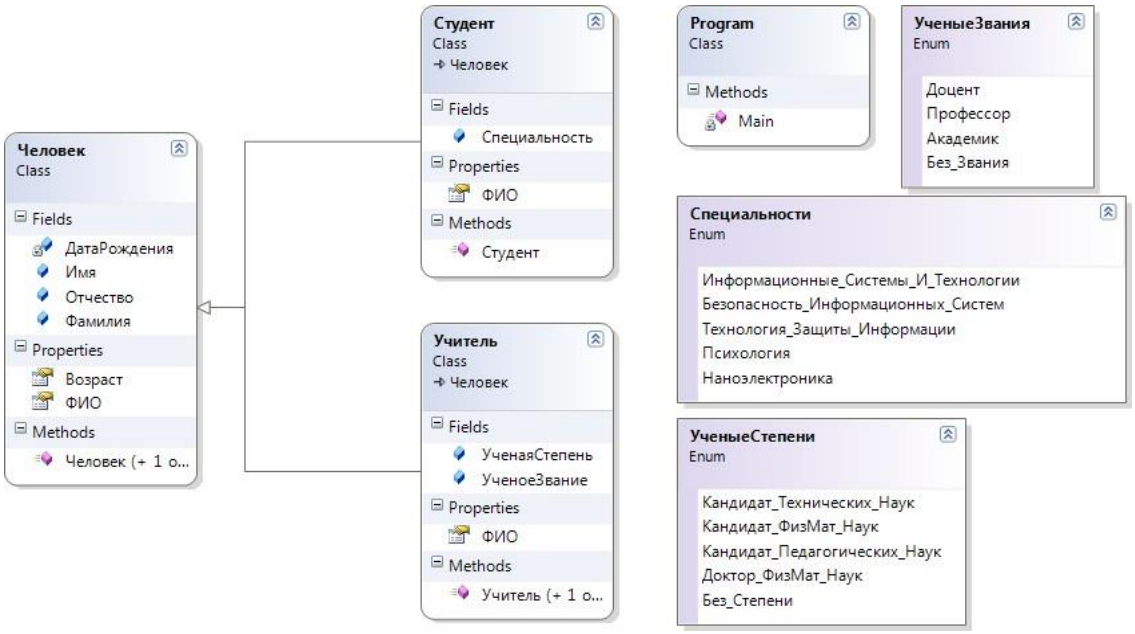


Рисунок 6.2 – Диаграмма типов приложения (создана средствами VS)

Структуры всегда наследуются от System.ValueType. Они могут также наследовать любое количество интерфейсов. Классы всегда наследуются от одного класса по вашему выбору. Они также могут наследовать любое количество интерфейсов.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение.

2. Изучите пример создания иерархии классов, представленный в разделе «Теоретическое обоснование» данной лабораторной работы.

3. Постройте свою иерархию классов в соответствии с индивидуальным заданием. В результате выполнения лабораторной работы должны быть реализованы следующие механизмы:

– использование типа-перечисления (хотя бы одного);

– использование переопределенного свойства (хотя бы одного);

– использование переопределенного метода (хотя бы одного);

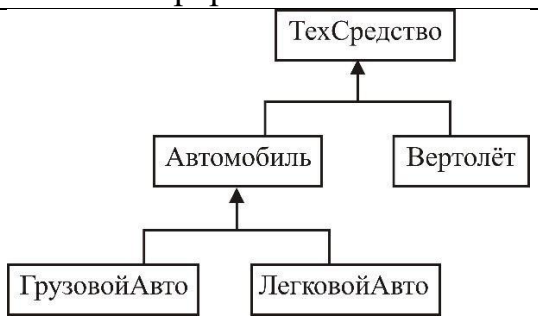
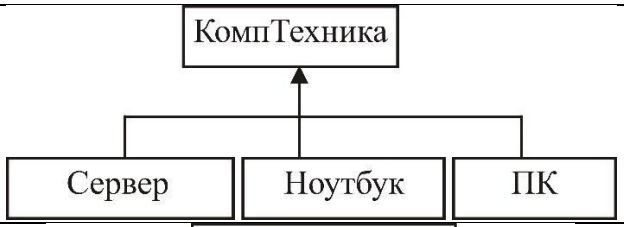
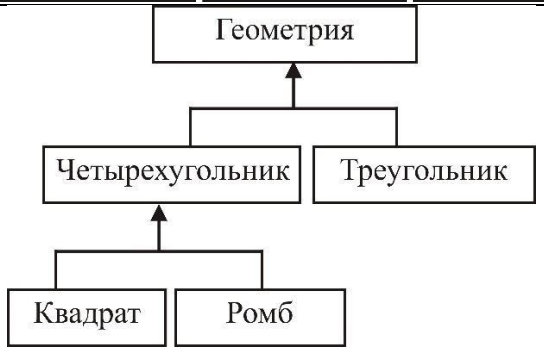
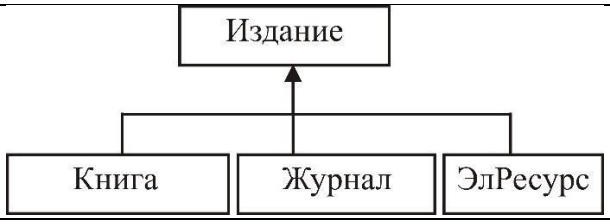
– использование вызова базового конструктора;

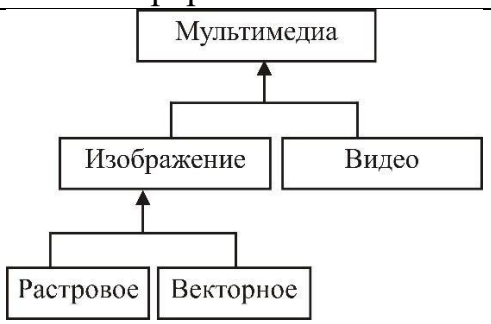
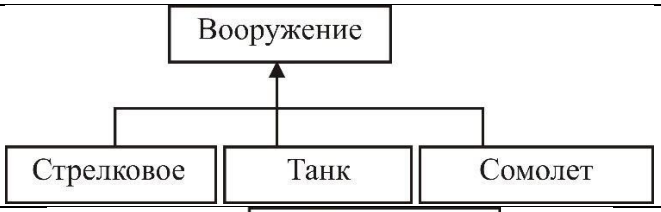
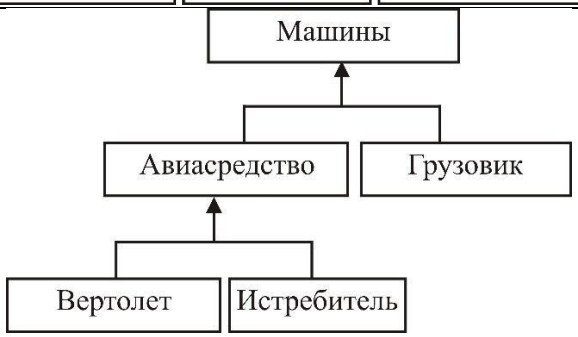
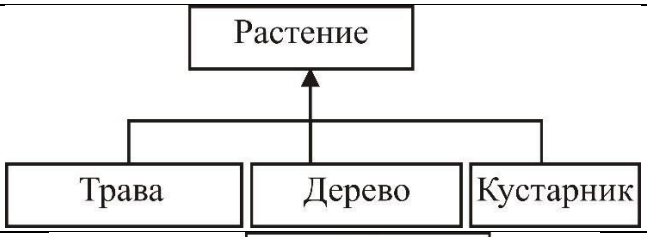
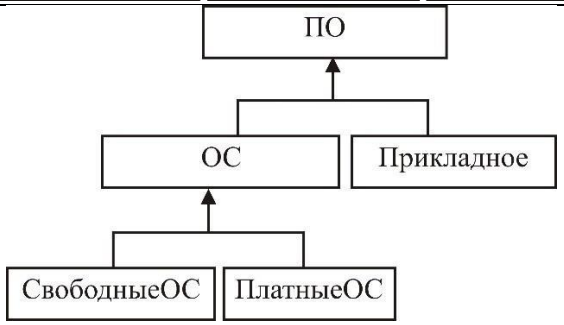
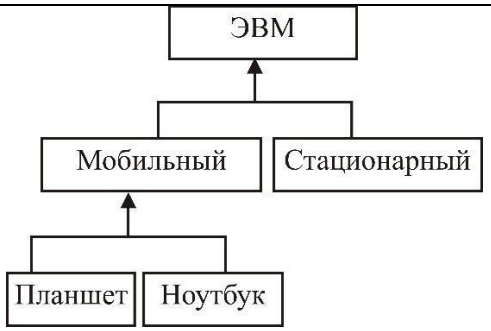
– использование вызова любого базового метода (отличного от конструктора).

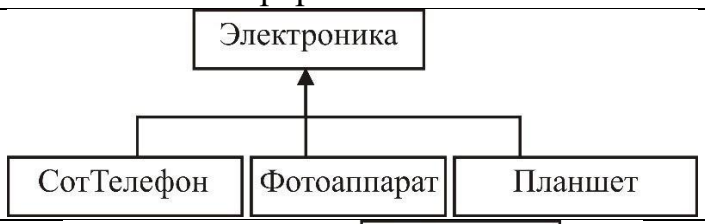
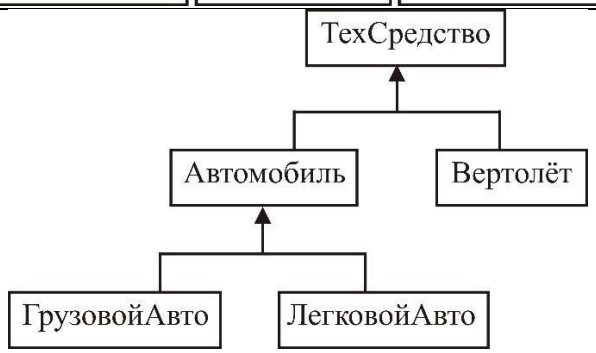
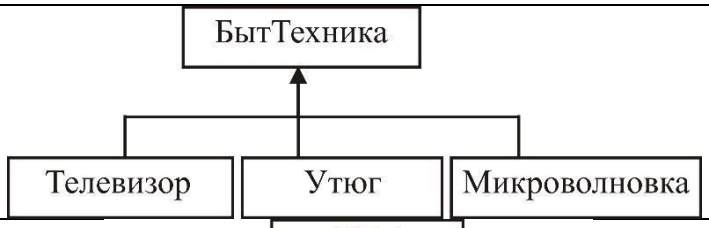
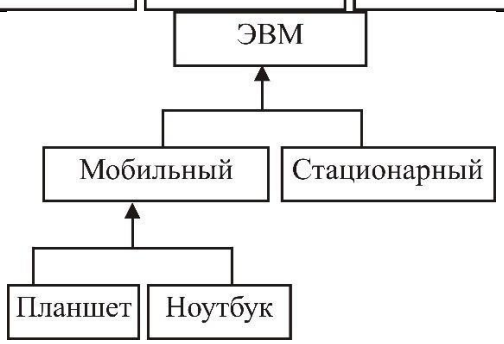
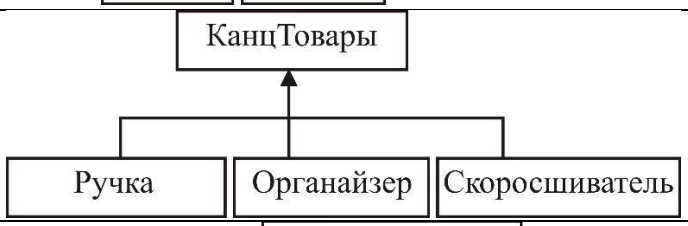
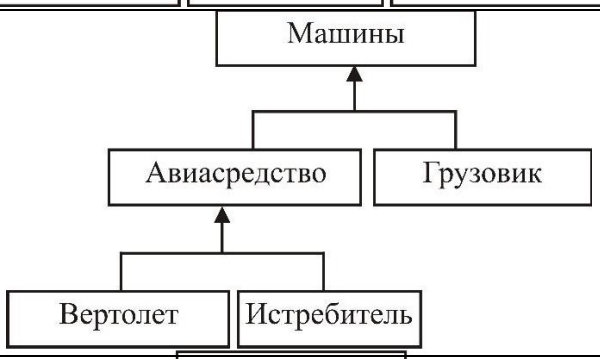
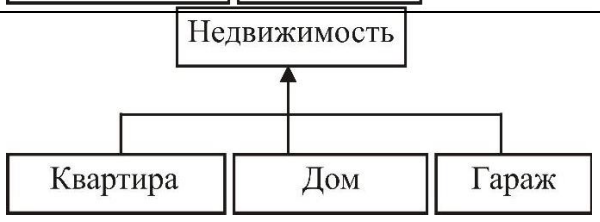
4. Продемонстрируйте использование классов, созданной иерархии (легче всего это сделать с использованием массивов). При защите работы укажите признаки присутствия полиморфного поведения в программе (реализация полиморфизма).

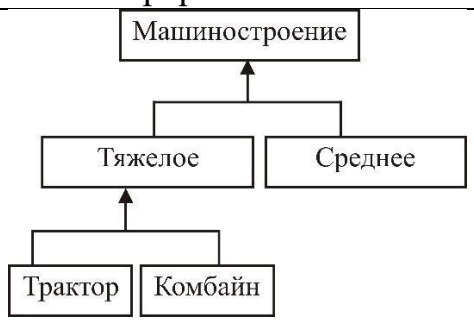
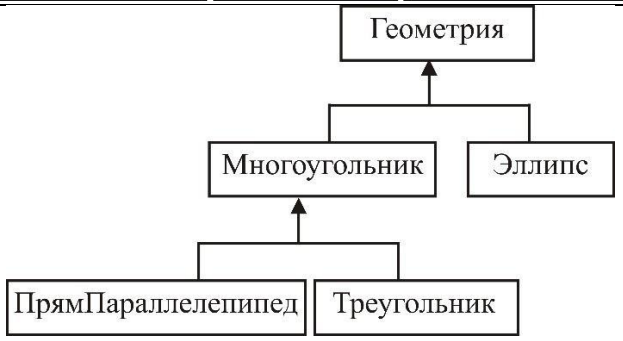
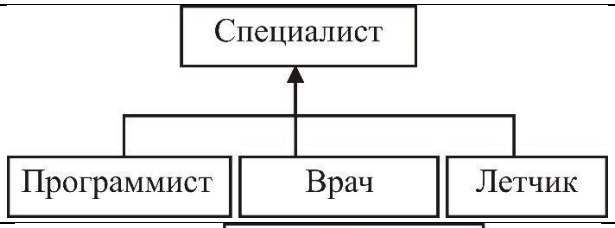
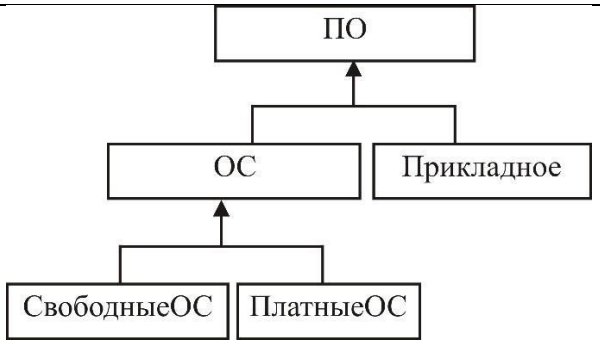
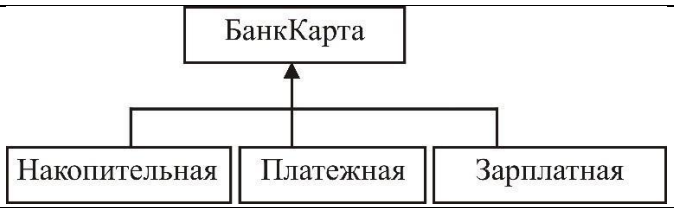
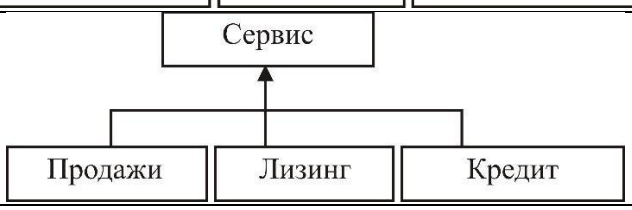
Индивидуальное задание.

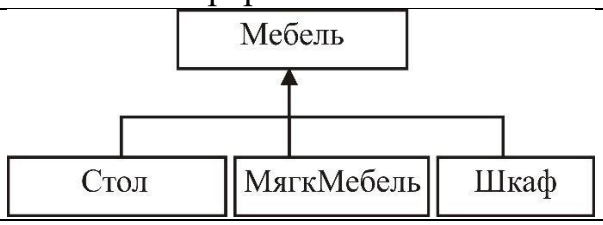
Спроектируйте класс, наполните его требуемой функциональностью, продемонстрируйте работоспособность класса.

Вариант	Иерархия классов
1	
2	
3	
4	

Вариант	Иерархия классов
5	
6	
7	
8	
9	
10	

Вариант	Иерархия классов
11	
12	
13	
14	
15	
16	
17	

Вариант	Иерархия классов
18	
19	
20	
21	
22	
23	
24	

Вариант	Иерархия классов
25	

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Что такое наследование реализации? Как описать синтаксически наследование реализации?

2. Для чего используется ключевое слово base?

3. Можно ли переопределить метод класса? Свойства класса? Данные класса?

4. Как переопределить метод в производном классе?

5. Для чего используется ключевое слово virtual?

6. Для чего используется ключевое слово override?

7. Как поменять цвет фона в консольном приложении?

8. Как построить диаграмму типов данных, используемых в приложении средствами Visual Studio 2008/ 2010?

9. Сколько базовых классов может быть у любого класса в C#?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5-7].

ЛАБОРАТОРНАЯ РАБОТА 7. ПОЛИМОРФИЗМ НА ОСНОВЕ ИНТЕРФЕЙСОВ

1. Цель и содержание

Цель лабораторной работы: изучить принципы работы с типами интерфейсов в C#.

Задачи лабораторной работы:

– научиться объявлять интерфейсы в C#;

– научиться создавать классы, реализующие интерфейсы.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

Кроме наследования реализации (implementation inheritance) в языке C# реализован механизм наследования интерфейсов (interface inheritance). Необходимо понимать, что не все объектно-ориентированные языки поддерживают интерфейсы.

Если класс наследует интерфейс, класс как бы берет на себя обязательства реализовать некоторый функционал.

Синтаксис наследования интерфейса не отличается от синтаксиса наследования реализации:

```
class КлассРеализующийИнтерфейс: Интерфейс
{
    // Данные – члены и функции – члены
}
```

Отличие от механизма наследования реализации состоит в том, что:

- класс обязательно должен реализовать методы и свойства, продекларированные в интерфейсе (никакой реализации в интерфейсе не существует);

- класс может реализовывать (наследовать) несколько интерфейсов, тогда как базовый класс может быть только один.

Для объявления типа интерфейса используется ключевое слово interface:

```
public interface ICalculate
{
    void Plus(int pPlus);
    void Minus(int pMinus);
}

public interface IVisual
{
    string Name { get; set; }
    void DrawObject();
}
```

В приведенном примере объявлены два интерфейса: ICalculate и IVisual.

Интерфейс ICalculate включает два метода с одним параметром. Из названий видно, что один метод что-то увеличивает, второй – что-то уменьшает на величину параметра.

Второй интерфейс IVisual содержит метод DrawObject без параметров, который призван нарисовать объект, а также свойство Name, которое доступно как для чтения, так и для записи.

Следует понимать, что интерфейсы не предполагают конкретную реализацию методов, а свойства интерфейсов не хранят данных. Интерфейс должен быть реализован классом, который и определит конкретное наполнение методов и свойств интерфейса.

Для использования объявленных интерфейсов создадим два класса Human (человек) и Car (автомобиль).

```
public class Human
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;
}

public class Car
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;
}
```

В каждом классе определен конструктор, который инициализирует закрытые данные-члены класса.

Требуется для каждого класса реализовать вывод в консоль, а также возможности увеличения скорости автомобиля и возможности изменения возраста у человека.

Этот общий функционал можно реализовать с использованием следующих способов:

1. Добавить все необходимые функции в каждый класс независимо. Этот метод приводит к «разбуханию кода», сложной управляемости кода.

2. Реализовать один класс, например Base (базовый), в котором определить общие методы и свойства (Plus, Minus, DrawObject). Затем использовать класс Base в качестве базового для классов Human и Car, причем в каждом классе переопределить методы базового класса с использованием ключевого слова override. Этот метод – наследование реализации. Получается иерархия классов, которые по смыслу и наполнению сильно отличаются. Хотя в данном подходе классы и код более упорядочены, все равно возникает избыточность за счет многократного переопределения методов.

3. Определение интерфейсов и реализация возможностей интерфейсов данными классами.

Именно метод 3 будет использован в данной лабораторной работе.

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте консольное приложение.

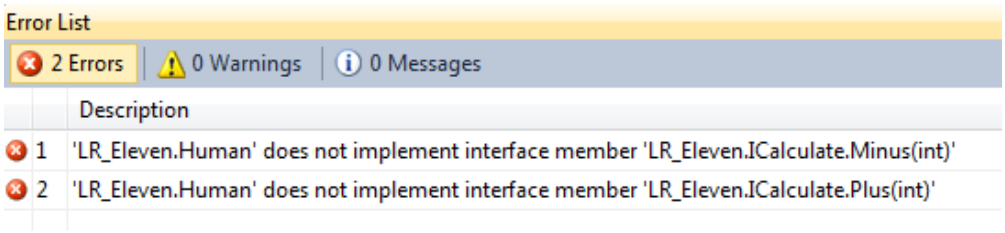
2. Определите в приложении классы Human и Car, а также интерфейсы ICalculate и IVisual, представленные в разделе «Теоретическое обоснование» данной лабораторной работы.

3. Реализуем механизм наследования интерфейса ICalculate классом Human. Для этого выполним следующие действия:

3.1 В определении класса укажем, что класс Human наследует интерфейс ICalculate.

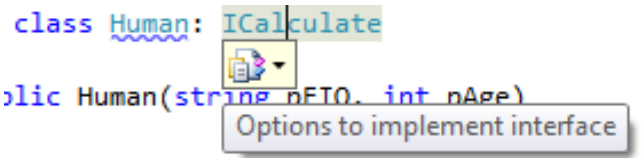
```
public class Human: ICalculate
```

3.2 Обратите внимание, что после этого попытка перекомпилировать проект приведет к ошибкам:

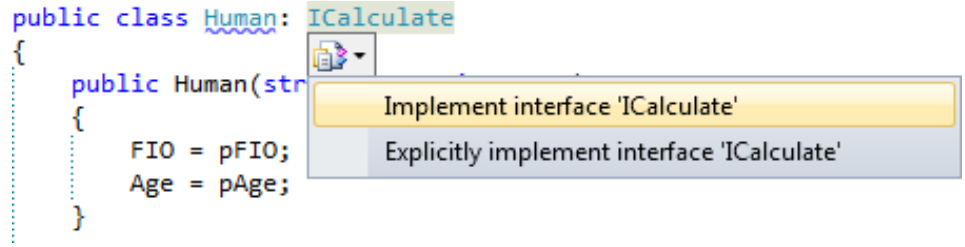


То есть компилятор сообщает, что интерфейс наследуется классом, но методы, заявленные в интерфейсе, классом не реализованы. Реализуем их.

3.3 Наведите курсор мыши на интерактивное подчеркивание и появится выпадающий список:



Раскройте его и выберите команду «Implement interface» (реализовать интерфейс).



3.3 В классе Human появятся два новых метода, как и было объявлено в интерфейсе ICalculate. Определение класса примет вид:

```
public class Human: ICalculate
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        throw new NotImplementedException();
    }

    public void Minus(int pMinus)
    {
        throw new NotImplementedException();
    }
}
```

Обратите внимание, что в каждый сгенерированный метод среда разработки добавила код вызова исключения, то есть проект скомпилируется без ошибок, но в процессе выполнения, при попытке использовать методы Plus или Minus программа завершится с ошибками. Это сделано для того, чтобы программист не забыл реализовать данные методы интерфейсов.

В процессе выполнения пп. 3.1 – 3.3 были использованы возможности Visual Studio по автоматизации реализации интерфейса. Очевидно, что интерфейс можно было реализовать, самостоятельно написав данный код.

– использование вызова базового конструктора;

– использование вызова любого базового метода (отличного от конструктора).

4. Определим окончательную реализацию для методов Plus и Minus:

```
public void Plus(int pPlus)
{
    Age += pPlus;
}

public void Minus(int pMinus)
{
    Age -= pMinus;
}
```

5. Аналогичным образом реализуем наследование интерфейса IVisual для класса Human. Окончательно для класса Human получим:

```
public class Human: ICalculate, IVisual
{
    public Human(string pFIO, int pAge)
    {
        FIO = pFIO;
        Age = pAge;
    }

    private string FIO;
    private int Age;

    public void Plus(int pPlus)
    {
        Age += pPlus;
    }

    public void Minus(int pMinus)
    {
        Age -= pMinus;
    }

    public string Name
    {
        get
        {
            return FIO + " : " +Age.ToString();
        }
        set
        {
            FIO = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "    o    \n" +
                "  -----  \n" +
                "    |    \n" +
                "   / \\   \n" +
                "  /   \\  \n"
            );
        Console.WriteLine(Name);
    }
}
```

6. Реализуем интерфейсы ICalculate и IVisual для класса Car:

```
public class Car: IVisual, ICalculate
{
    public Car(string pManufacturer, string pModel, int pVelocity)
    {
        Manufacturer = pManufacturer;
        Model = pModel;
        Velocity = pVelocity;
    }

    private string Manufacturer;
    private string Model;
    private int Velocity;

    public void Plus(int pPlus)
    {
        Velocity += pPlus;
    }

    public void Minus(int pMinus)
    {
        Velocity += pMinus;
    }

    public string Name
    {
        get
        {
            return Manufacturer + " - " + Model +
                " : " + Velocity.ToString() + "km/h";
        }
        set
        {
            Model = value;
        }
    }

    public void DrawObject()
    {
        Console.WriteLine
            (
                "     -----------\n" +
                "____/           \\_____\n" +
                "|                     |\n" +
                "---(@)------------(@)--- \n"
            );
        Console.WriteLine(Name);
    }
}
```

7. Когда все классы и интерфейсы определены и реализованы можно их использовать. Продемонстрируем использование типов данных созданием соответствующих объектов в функции main.

```
static void Main(string[] args)
{
    Console.BackgroundColor = ConsoleColor.White;
    Console.ForegroundColor = ConsoleColor.Blue;
    Console.Clear();

    Console.Title = "Лабораторная работа №11";

    Human h = new Human("Иванов Иван Иванович", 50);
    h.Plus(5);
    h.Minus(1);
    h.DrawObject();

    Console.WriteLine("\n\n\n");

    Car car = new Car("Hyundai", "ix35", 120);
    car.Plus(25);
    car.Minus(11);
    car.DrawObject();

    Console.ReadKey();
}
```

8. Вид окна разработанного приложения представлен на рис. 7.1:

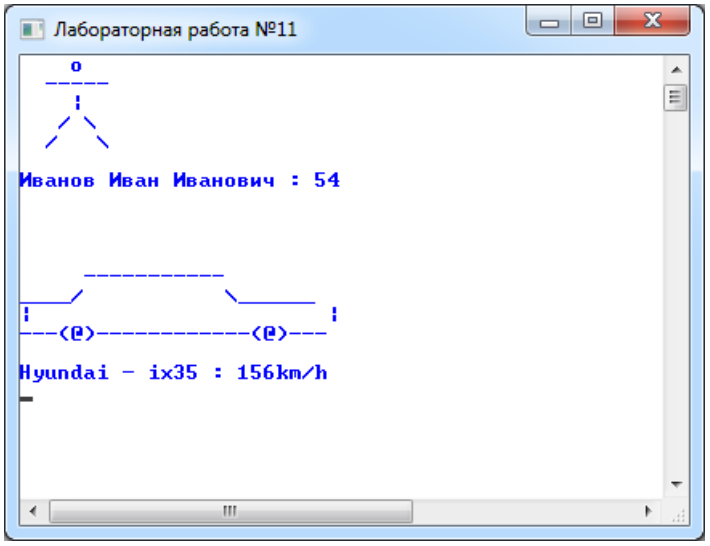


Рисунок 7.1 – Консольное приложение на основе интерфейсных типов

Диаграмма типов данных, созданная редактором Visual Studio, для разработанного приложения, показана на рис. 7.2:

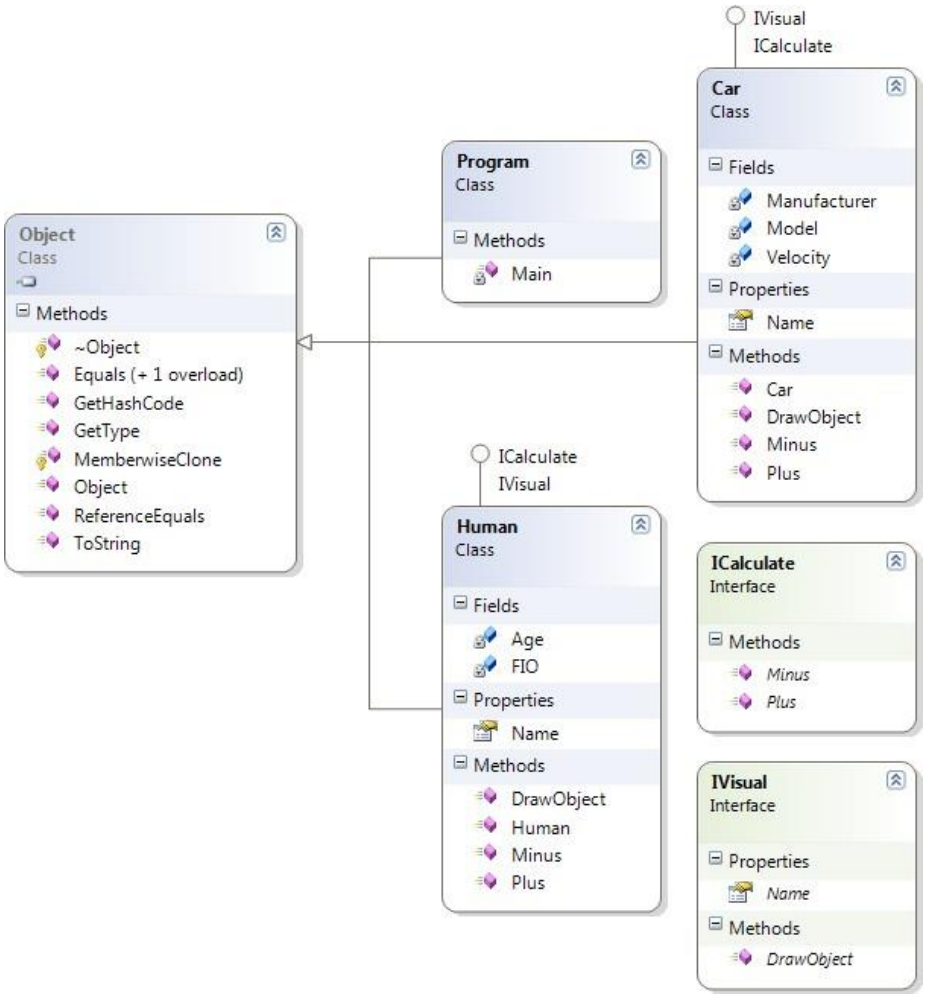


Рисунок 7.2 – Диаграмма классов приложения

Индивидуальное задание

Спроектируйте классы, наполните их требуемой функциональностью, определите интерфейс, продемонстрируйте использование объектов класса и методов интерфейса. Постройте диаграмму классов своего приложения средствами Visual Studio или с использованием сторонних редакторов UML-диаграмм.

Продемонстрируйте явление полиморфизма при использовании классов из индивидуального задания.

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Вопросы для защиты работы

1. Чем отличается наследование интерфейсов от наследования реализации?

2. Поясните, каким образом проявляется полиморфное поведение объектов при реализации классами интерфейсов?

3. Сколько интерфейсов может реализовывать класс?

4. Как объявляются интерфейсные типы? Для чего используются интерфейсы?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1-4].

ЛАБОРАТОРНАЯ РАБОТА 8. РАЗРАБОТКА ПРИЛОЖЕНИЯ WINDOWS FORMS

1. Цель и содержание

Цель лабораторной работы: научиться создавать приложение Windows в среде MS Visual Studio и программировать алгоритмы с использованием простых элементов управления.

Задачами лабораторной работы являются:

– разработка приложения с использованием Windows Forms,

– изучение основных членов класса Form;

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическая часть

Файловая структура проекта MS Visual Studio в случае использования типа поекта Windows Form Application выглядит следующим образом (рис. 8.1):

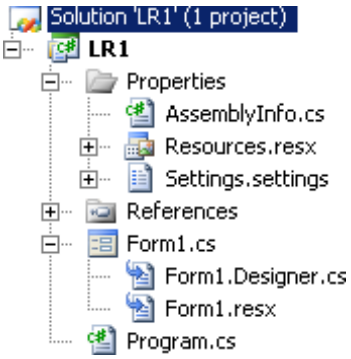


Рисунок 8.1 – Вид окна Solution Explorer для Windows Form Application

Файл Program.cs содержит класс Program и статический метод Main(), с которого начинается выполнение приложения (рис. 8.2).

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

namespace LR1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Рисунок 8.2 – Листинг Program.cs

В самом начале файла Program.cs выполняется объявление используемых пространств имен с использованием using.

Класс Program содержит метод Main(), который используя статический метод Run класса Application создает и выводит на экран главную форму приложения: Application.Run (new Form1()).

Таким образом, реализуется один из принципов объектно-ориентированного программирования: разграничение обязанностей (т.е. каждый класс выполняет минимально возможное количество операций).

Класс главной формы Form1 (по умолчанию) представлен двумя связанными C#-файлами. Для отображения содержимого Form1.cs необходимо щелкнутьправой кнопкой мыши в окне проектирования главной формы на самой форме или на пиктограмме Form1.cs в окне Solution Explorer (рис. 8.1) и в появившемся контекстном меню выбрать пункт «View Code».

Листинг Form1.cs представлен на рис 8.3.

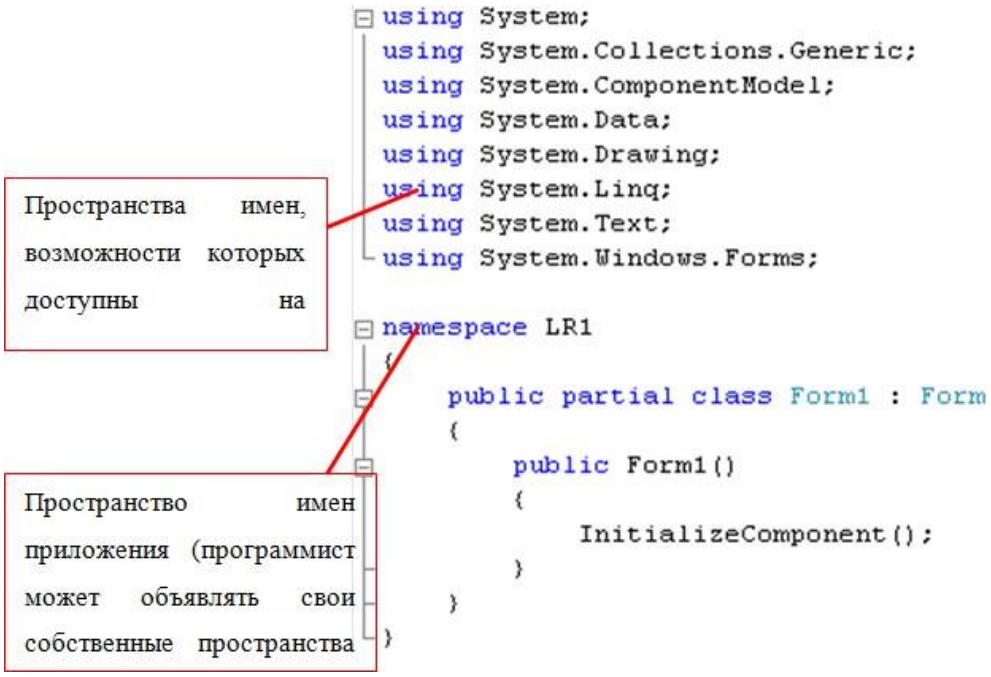


Рисунок 8.3 –Листинг Form1.cs

Конструктор, созданный по умолчанию, вызывает метод InitializeComponent, который определен в соответствующем файле Form1.Designer.cs (рис. 8.1).

Этот метод создается автоматически, в нем отображаются все изменения, производимые программистом в окне визуального проектирования формы.

Пространства имен, доступные для использования, объявляются в начале файла.

System является базовым пространством имен – в него входят все остальные типы и пространство имен. Конструкция using System; в начале файла программы указывает на то, что весь программный код будет выполняться в данном пространстве имен, поэтому при использовании типов (например Int32), определенных в пространстве System нет необходимости указывать само имя пространства (то есть нет необходимости писать System.Int32).

Программист может самостоятельно вводить пространства имен при написании приложений.

Обработка событий в режиме проектирования. Окно проектирования главной формы приложения (рис. 8.4) можно открыть дважды щелкнув на

пиктограмме Form1.cs в окне Solution Explorer (рис. 8.1). При этом откроется вкладка Form1.cs [Design].

Окно свойств примет вид, показанный на рисунке 8.4.

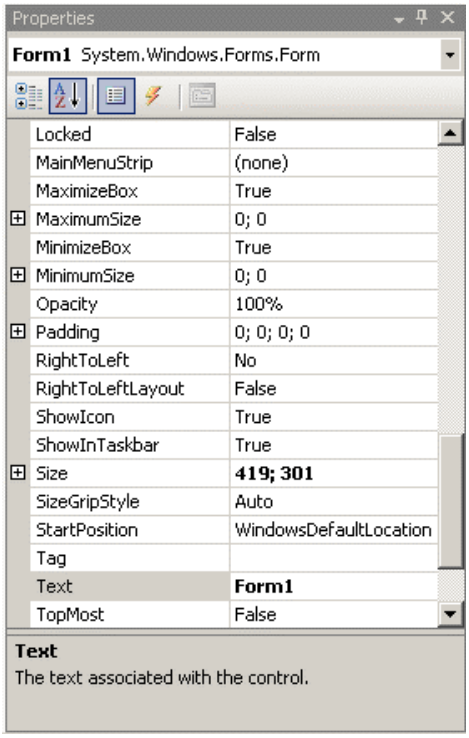


Рисунок 8.4 – Вид окна Свойств (Properties) в режиме проектирования формы

Кнопка  позволяет отобразить список событий, характерных для элемента управления, активного в данный момент в окне проектирования. Например, если выделить главную форму приложения, то отобразятся события, характерные для класса Form.

Для отработки механизма регистрации событий, рассмотрим обработку наиболее простых событий – событий от мыши.

Для добавления обработчика события мыши необходимо выбрать соответствующее событие (Click, DoubleClick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove,. MouseHover, MeaseWheel) и дважды щелкнуть на поле, расположенном в соседнем от него столбце в окне свойств (таким образом обработчику присвоится имя по умолчанию). В данном поле можно ввести свое название для нового обработчика, или выбрать из выпадающего списка уже существующие обработчики.

После регистрации события в окне «Properties», автоматически добавляются строки кода в файлы Form1.Designer.cs (регистрируется обработчик события) и Form1.cs (добавляетсяметод, ассоциированный с данным событием).

Форма приложения должна быть функциональной и максимально эргономичной. Такие простые вещи, как цветовые комбинации, размеры шрифтов и окон могут сделать приложение намного более привлекательным для пользователя.

В случае если свойство Icon формы установлено, а для свойства ControlBox не указано значение false, то значок появится в левом верхнем углу формы. Обычно принято устанавливать здесь значок приложения.

Свойство FormBorderStyle задает тип рамки, которая появляется вокруг формы. Для этого используется перечисление FormBorderStyle. Допустимые значения этого перечисления: Fixed3D, FixedDialog, FixedSingle, FixedToolWindow, None, Sizable, SizableToolWindow.

Для выполнения лабораторной работы потребуется основная информация о событиях мыши:

Все обработчики событий мыши принимают параметр типа MouseEventArgs. Поступающий на вход обработчика событий мыши объект типа MouseEventArgs позволяет получить дополнительную информацию о действии мыши, путем введения ряда специальных членов класса (табл. 5.1).

Таблица 5.1 – Свойства типа MouseEventArgs

Свойство	Описание
Button	Содержит информацию о том, какая клавиша была нажата, в соответствии с определением перечня MouseButtons
Clicks	Содержит информацию о том, сколько раз была нажата и отпущена клавиша мыши
Delta	Содержит информацию со знаком, соответствующее числу щелчков, произошедших при вращении колесика мыши
X	Содержит информацию о координате X
Y	Содержит информацию о координате Y

4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 20112 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

1. Создайте в среде разработки MS VS проект. В качестве типа проекта выбрать «Windows Application» (или «Windows Forms Application» в зависимости от версии .NET Framework)

2. После создания и сохранения проекта измените программу таким образом, чтобы координаты курсора мыши выводились в заголовке главного окна приложения.

Для этого в обработчик события движения мыши над оконной формой (MouseMove) добавьте строку кода:

Text = string.Format("Координаты:{0}, {1}", e.X, e.Y);

3. Добавьте текстовое поле (TextBox) в режиме разработки (для этого необходимо использовать панель элементов управления «ToolBox»). Дополните обработчик движения мыши таким образом, чтобы в текстовом поле отображалась сумма координат указателя мыши.

4. Работающую программу необходимо представить преподавателю.

5. После этого выполните индивидуальное задание в соответствие с вариантом. В каждом задании необходимо вывести значение выражения, предварительно введя значения переменных в соответствующие текстовые поля формы главного окна приложения. Результат выводится в заголовок окна в ответ на нажатие кнопки (кнопку также необходимо поместить на форму).

6. Для выполнения индивидуального задания необходимо использовать математические функции, которые доступны в виде статических методов класса Math.

Индивидуальное задание.

Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Выражение для вычисления
1	$Z = \sqrt{\left\|\frac{a \cdot x}{w^a}\right\|} + \sqrt{\|b\|} - \|x + \cos(y)\|$
2	$F = \sqrt{\|d2\|} \cdot x + \sqrt{\left\|b^3\right\|} - \left\|x^2 + \cos(y)\right\|$
3	$A = -d \cdot \frac{z}{\sqrt{\|e\|}} + \|\sin(e) + \cos(y)\|$
4	$U = \sqrt{\left\|\frac{f-e}{w}\right\|} + \left\|\sin^2(\frac{e}{w}) + \cos(y)\right\|$
5	$t = \lg(f) - e + \left\|\sin(\frac{w}{t}) + \sqrt{\|e\|}\right\|$

6	$V = \frac{h-e}{q+a} + \left\lvert \sin(f) + \sqrt{\lvert \sin(b) \rvert} \right\rvert$
7	$Q = \sin(h + \frac{d}{e^{o1}}) - o1 + \left\lvert \sin(f) + \sqrt{\lvert \sin(b) \rvert} \right\rvert$
8	$E = \sin(h11 + \frac{d12}{\ln(h11)}) - v6 + \left\lvert \sin(h3) + \sqrt{\lvert \sin(f) \rvert} \right\rvert$
9	$Z = z \cdot ax1 + \sqrt{\left\lvert e7 + \frac{x}{y} \right\rvert} - \lvert x + \cos(as\_9 + y) \rvert$
10	$RES = \frac{\arg 01 - \arg 02 + \lvert \sin(\arg 03) + \sqrt{\lvert \arg 04 \rvert} \rvert}{\arg 01^{\arg 5} - \lg(\arg 02)}$
11	$U = \frac{v - e\_2 + \lvert \cos(\arg 3) + \sqrt{\lvert t \rvert} \rvert}{a\_del * tg(t)}$
12	$t = W + \frac{\cos(g \cdot q)}{W} - e + \left\lvert \sin(e) + \sqrt{\lvert o \rvert} \right\rvert$
13	$Z = z \cdot \cos(y) + \sqrt{e7} - \lvert x + \cos(as\_9 + y) \rvert$
14	$R = q + \lvert \sin^2(e) + \cos(y) \rvert \cdot \cos(s + g)$
15	$F = \sin(\frac{v \cdot x}{y}) \cdot x + \sqrt{\lvert b^3 \rvert} - \lvert x^2 + \cos(y) \rvert$
16	$res = \frac{F}{\cos(w)} - e + \frac{\left\lvert \sin(\frac{w}{t}) + \sqrt{\lvert e \rvert} \right\rvert}{e}$
17	$t = W + \frac{tg(g \cdot q)}{\ln(20 \cdot x)} - \frac{e}{W} + \left\lvert \sin(e) + \sqrt{\lvert W - e \rvert} \right\rvert$
18	$t = W + \frac{tg(g + q)}{\ln(20 \cdot x)} - \frac{e}{W} + \left\lvert \sin(e) + \sqrt{\lvert W - e \rvert} \right\rvert$
19	$F = \sqrt{\lvert d2 - x \rvert} \cdot x + \sqrt{\lvert b^3 \rvert} - \lvert x^2 + \cos(y) \rvert$
20	$t = W + \frac{\cos(g \cdot q)}{W} - e + \left\lvert \sin(e) + \sqrt{\left\lvert \frac{o}{e^4} \right\rvert} \right\rvert$
21	$U = \frac{v^3 - e\_2 + \lvert \cos(\arg 3) + \sqrt{\lvert t \rvert} \rvert}{a\_del * tg(t)}$
22	$F = \sin(\frac{v \cdot x}{y}) \cdot x + \sqrt{\lvert b^3 \rvert} - \lvert x^2 + \cos(y) \rvert$
23	$v = \sqrt[4]{e\_2} \cdot \sqrt{\frac{\sin(12 * \arg 3)}{e\_2 - 1}} + \left\lvert \cos(\arg 3) + \sqrt{\lvert t \rvert} \right\rvert$

24	$res = \frac{F^{\lvert e \rvert}}{\cos(w+e)} - e + \frac{\left\lvert \sin(\frac{w}{t}) \right\rvert}{25 + \sqrt{\lvert e \rvert}}$
25	$t = \frac{az}{\sqrt{g^3}} + \frac{\cos(g \cdot q)}{az} - g + \left\lvert \sin(q) + \sqrt{\lvert 12 - q \rvert} \right\rvert$

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

8. Контрольные вопросы

1. Какие файлы описывают класс формы?

2. Какие действия необходимо выполнить для создания обработчика события?

3. Где описывается код обработчика события? В каком файле регистрируется обработчик события (метод привязывается к событию)?

4. Как получить доступ к координатам курсора мыши?

5. Какой класс содержит методы, реализующие математические функции?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [6].

ЛАБОРАТОРНАЯ РАБОТА 9. МЕХАНИЗМ СОБЫТИЙ. ПРОСТЫЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

1. Цель и содержание

Цель лабораторной работы: изучить принципы использования стандартных элементов управления в приложениях Windows.

Задачи лабораторной работы:

– научиться использовать кнопки, флажки, переключатели;

– научиться использовать списки, выпадающие списки.

– обработка событий от простых элементов управления.

2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

3. Теоретическое обоснование

Для выполнения лабораторной работы необходимо изучить теоретические основы проектирования с использованием стандартных элементов управления библиотеки .NET Framework.

Класс. Button. Класс Button представляет простую командную кнопку и наследуется от класса ButtonBase. Наиболее часто приходится писать код для обработки события Click кнопки. В следующем фрагменте кода реализован обработчик события Click (обработка данного события не представляет сложностей). С помощью метода PerformClick можно эмулировать событие Click на кнопке без реального щелчка на ней пользователем, что может пригодиться для тестирования построенного пользовательского интерфейса. В окне также имеется понятие кнопки по умолчанию, на которой автоматически совершается щелчок, если пользователь нажимает в этом окне клавишу «Enter». Чтобы

идентифицировать кнопку как кнопку по умолчанию, в форме, содержащей эту кнопку, необходимо установить свойство AcceptButton в объект кнопки.

Класс CheckBox (рис. 6.1). Элемент управления CheckBox также унаследован от ButtonBase и используется для приема двух или трех состояний от пользователя.

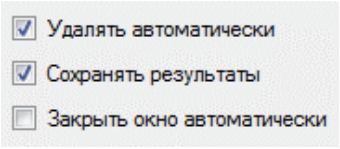


Рисунок 6.1 – Внешний вид CheckBox

Если установить свойство ThreeState в true, то свойство CheckState элемента CheckBox сможет принимать одно их трех значений перечисления CheckState, описанных в табл. 6.1.

Таблица 6.1 – Состоятия CheckBox (возможные значения перечисления CheckState)

Значение	Описание
Checked	Флажок имеет отметку.
Unchecked	Флажок не имеет отметки.
Indeterminate	В этом состоянии флажок становится серым (неопределенное состояние).

Состояние Indeterminate удобно, когда пользователь должен быть уведомлен, что опция не установлена. Если нужно булевское значение, можно проверить свойство Checked.

События CheckedChanged и CheckStateChanged возникают, когда изменяются значения свойств CheckState и Checked. Для флажка с тремя состояниями понадобится присоединиться к событию CheckStateChanged. Перехват этих событий может быть полезен для установки других значений, основанных на новом состоянии CheckBox.

Класс RadioButton (рис. 6.2). Переключатель (кнопки опций) – элемент управления, унаследованный от ButtonBase.

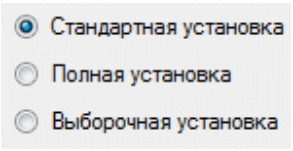


Рисунок 6.2 – Внешний вид RadioButton

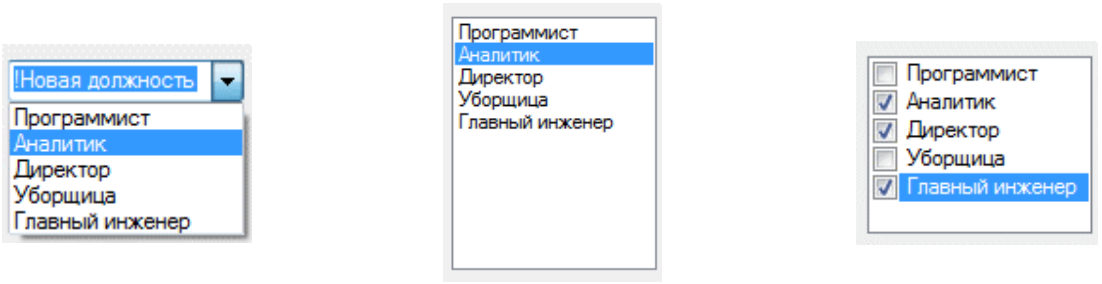
Переключатели обычно используются в группе. Переключатели позволяют пользователю выбирать одну из нескольких опций. При наличии нескольких элементов управления RadioButton в одном контейнере, только один из них может быть выбран в один и тот же момент времени.

Свойство Appearance принимает значение перечисления Appearance, которым может быть Button либо Normal. В случае установки значения Normal переключатель выглядит как маленький кружочек с меткой рядом с ним. Выбор переключателя заполняет этот кружок, а выбор другого переключателя снимает отметку с текущего выбранного переключателя и кружок делается пустым. В случае установки значения Button элемент управления выглядит подобно стандартной кнопке, но работает как переключатель – его выбор означает включение (вдавливание кнопки), а отказ от выбора – выключение (стандартное положение кнопки).

Свойство CheckedAlign определяет местоположение кружка по отношению к тексту метки. Он может быть над меткой, с любой стороны от нее либо под меткой.

Событие CheckedChanged инициируется при любом изменении свойства Checked. Это позволяет предпринимать другие действия на основе нового значения элемента управления.

Классы ComboBox, ListBox и CheckedListBox (рис. 6.3). Элементы управления ComboBox, ListBox и CheckedListBox унаследованы от класса ListControl. Этот класс предоставляет некоторую базовую функциональность управления списками. Наиболее важные аспекты использования списочных элементов управления состоят в добавлении данных и выборе данных из списка.



а) б) в)

Рисунок 6.3 – Внешний вид списков: а – ComboBox; б – ListBox; в – CheckedListBox

На выбор списка влияет то, как он должен использоваться, и тип данных, которые будут помещены в список. Если есть необходимость во множественном выборе, или если пользователю нужно видеть несколько элементов в списке в одно и то же время, то для этого лучше всего подойдут ListBox и CheckedListBox. Если в любой момент времени в списке может быть выбран только один элемент, то предпочесть следует ComboBox.

Для добавления элементов в списки необходимо обратиться к коллекции ListBox.ObjectCollection, которая доступна через свойство Items списка.

Поскольку коллекция хранит объекты, в список может быть добавлен любой допустимый тип .NET. Чтобы идентифицировать элементы, необходимо установить два важных свойства. Первым свойством является DisplayMember. Его настройка сообщает ListControl, какое свойство объекта должно быть отображено в списке. Другое свойство – ValueMember. Оно представляет собой свойство объекта, которое должно возвращаться в качестве значения. Если в список были добавлены строки, для обоих свойств по умолчанию используются строковые значения.

После загрузки данных в список для их получения могут быть использованы свойства Selectedltem и Selectedlndex. Свойство Selectedltem возвращает объект, выбранный в данный момент. Если список предполагает множественный выбор, то должна применяться коллекция Selectedltems для получения выбранных элементов.

Если для наполнения списка применяется DataBinding, то свойство SelectedValue вернет значение свойства выбранного объекта, которое было установлено в свойстве ValueMember.

Элемент ComboBox – это комбинация поля редактирования и окна списка. Стиль ComboBox задается установкой в свойстве DropDownStyle значения перечисления DropDownStyle (таблица 6.2).

Таблица 6.2 – Стили ComboBox (возможные значения перечисления DropDownStyle)

Значение	Описание
DropDown	Текстовая часть комбинированного списка допускает редактирование, и пользователи могут вводить значение. Также они могут щелкать на кнопке со стрелочкой, чтобы отобразить раскрывающийся список элементов.
DropDownList	Текстовая часть не допускает редактирование. Пользователи должны выбирать значения из списка.
Simple	Подобно DropDown, но список является постоянно видимым.

Класс DateTimePicker (рис. 6.4). Элемент управления DateTimePicker позволяет выбирать значение даты или времени (или то и другое) во множестве разных форматов. Значение DateTime можно отобразить в любом стандартном формате времени и даты. Свойство Format принимает значение перечисления DateTimePickerFormat, которое устанавливает формат Long, Short, Time или Custom. Если свойство Format установлено в DateTimePickerFormat .Custom, с помощью свойства CustomFormat можно задать строку, представляющую формат.

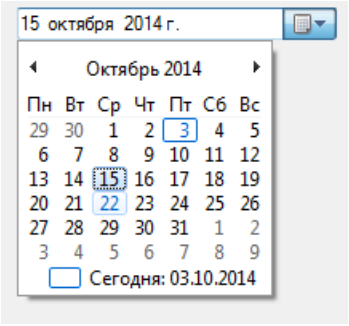


Рисунок 6.4 – Внешний вид DateTimePicker

Свойство Text возвращает строковое представление значения DateTime, а свойство Value – объект DateTime. С помощью свойств MinDate и MaxDate можно устанавливать максимальное и минимальное значения дат. Когда пользователь щелкает на стрелке вниз, отображается календарь, позволяющий выбрать дату. Доступны свойства, которые позволяют изменить внешний вид календаря, указать фоновые цвета заголовка и месяца, а также цвета переднего плана.

4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2013 и выше.

5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

6. Методика и порядок выполнения работы

Для выполнения лабораторной работы необходимо выполнить следующую последовательность действий:

1. Создайте проект приложения Windows Forms в среде MS Visual Studio.

2. Реализуйте методы для вычисления выражений в соответствии с вариантом индивидуального задания.

3. При проектировании формы необходимо учитывать следующие особенности задачи:

– выражение может быть вычислено двумя различными способами, то есть пользователь должен иметь возможность выбрать метод расчета (подумайте, какой элемент управления подходит для этого больше всего?);

– в правой части выражения присутствуют неизвестные переменные, которые пользователь может вводить с клавиатуры в текстовые поля;

– в некоторых выражениях присутствуют коэффициенты, выбор значений которых необходимо реализовать с помощью различных списков.

Варианты индивидуальных заданий

Вариант	Выражение для вычисления
1	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{3 + a \cdot j}{b \cdot i}. \end{cases}$
2	$Z = \begin{cases} -\frac{1}{2} + \frac{1}{6} - \frac{X}{24} + \frac{Y}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j}{i}. \end{cases}$
3	$Z = \begin{cases} 1 - \frac{1}{2} + \frac{X}{3} - \frac{Y}{4} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{j^2 + h \cdot i}{i + c \cdot j}. \end{cases}$

4	$Z = \begin{cases} -\frac{Y}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{c \cdot i}. \end{cases}$
5	$Z = \begin{cases} -\frac{p}{2} + \frac{p}{6} - \frac{p}{24} + \frac{p}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + b \cdot j}{c \cdot i}. \end{cases}$
6	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + a \cdot j^2}{i^3 + j^3}. \end{cases}$
7	$Z = \begin{cases} -\frac{X}{2} + \frac{X^2}{6} - \frac{X^3}{24} + \frac{X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i^2}{i^3 + b \cdot j^3}. \end{cases}$
8	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i}{i^3 + j^3}. \end{cases}$
9	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot (i + j)}{i \cdot (i + 2)}. \end{cases}$
10	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{3 + j}{c \cdot i}. \end{cases}$
11	$Z = \begin{cases} -\frac{1}{2} + \frac{1}{6} - \frac{X}{24} + \frac{1}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j}{a^j}. \end{cases}$

12	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{2} - \frac{X^2}{3} + \frac{Y^3}{4} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{j^2 + c \cdot i}{i + d \cdot j}. \end{cases}$
13	$Z = \begin{cases} -\frac{}{2} + \frac{}{6} - \frac{}{24} + \frac{}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{c \cdot i}. \end{cases}$
14	$Z = \begin{cases} -\frac{p}{2} + \frac{p}{6} - \frac{p}{24} + \frac{p}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + b \cdot j}{c \cdot i}. \end{cases}$
15	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j^2}{i + j}. \end{cases}$
16	$Z = \begin{cases} -\frac{X}{2} + \frac{2 \cdot X^2}{6} - \frac{3 \cdot X^3}{24} + \frac{4 \cdot X^4}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{(i+1)^2}{a \cdot i + b \cdot j}. \end{cases}$
17	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i}{i + b \cdot j}. \end{cases}$
18	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot (i + j)}{b \cdot i \cdot (i + 2)}. \end{cases}$
19	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{3} - \frac{X^2}{5} + \frac{Y^3}{7} - ..; \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot j^2 + 2}{b \cdot i + 3}. \end{cases}$

20	$Z = \begin{cases} -\frac{Y}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{a \cdot i + b \cdot j}{c \cdot i}. \end{cases}$
21	$Z = \begin{cases} -\frac{p}{1 \cdot 2} + \frac{p}{2 \cdot 6} - \frac{p}{3 \cdot 24} + \frac{p}{4 \cdot 120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + d \cdot j}{c \cdot i^3}. \end{cases}$
22	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + j^2}{i^3}. \end{cases}$
23	$Z = \begin{cases} -\frac{3 \cdot Y}{2} + \frac{5 \cdot Y^2}{6} - \frac{7 \cdot X^3}{24} + \frac{9 \cdot Y^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i + k \cdot j}{c \cdot i}. \end{cases}$
24	$Z = \begin{cases} -\frac{3 \cdot p^2}{2} + \frac{5 \cdot p^3}{6} - \frac{7 \cdot p^4}{24} + \frac{9 \cdot p^5}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{i^2 + b}{c \cdot (i+1)^3}. \end{cases}$
25	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - ..;. \\ \sum_{i=1}^{N}\sum_{j=1}^{R} \frac{(a \cdot i + j)^2}{i^3 + b \cdot j^3}. \end{cases}$

7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

8. Вопросы для защиты работы

1. Какое событие элемента управления Button обрабатывается в программах чаще всего?

2. Для чего предназначен компонент CheckBox? Назовите основные свойства класса CheckBox.

3. Опишите назначение элемента RadioButton. Какой внешний вид может принимать данный компонент? Назовите основные свойства класса RadioButton.

4. Назовите классы компонентов для представления списочной информации.

5. Опишите основные свойства класса ListBox. Чем комопнет ListBox отличается от CheckedListBox?

9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 10. ТЕХНОЛОГИЯ GDI+ В ПРИЛОЖЕНИЯХ WINDOWS FORMS

10.1. Цель и содержание

Цель лабораторной работы: научиться использовать инструменты визуализации двумерной графики, предоставляемые пространством имен GDI+.

Задачами лабораторной работы являются:

– научится использовать основные типы, предоставляемые библиотекой GDI+;

– научиться использовать различные режимы вывода графики;

– освоить принципы построения графиков, диаграмм;

– научиться создавать интерактивные приложения;

– научиться обрабатывать события в графических приложениях;

– научиться использовать класс Timer;

– научиься реализовывать механизм анимации в приложениях Windows.

10.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

10.3. Теоретическое обоснование

10.3.1 Основы GDI+

В таблице 10.1 приводится описание базовых пространств имен GDI+.

Таблица 10.1 – Базовые пространства имен GDI+

Пространство имен	Описание
System.Drawing	Базовое пространство имен GDI+, определяющее множество типов для основных операций визуализации (шрифты, перья, кисти и т.д.), а также тип Graphics.
System.Drawing.Drawing2D	Содержит описание типов, используемых для более сложной двумерной/векторной графики (градиентные кисти, стили окончания линий, геометрические трансформации и т.д.)
System.Drawing.Imaging	Содержит описание типов, обеспечивающих обработку графических изображений (изменение палитры, извлечение метаданных изображения, работа с метафайлами и т.д.)
System.Drawing.Printing	Содержит описание типов, обеспечивающих отображение графики на печатной странице, непосредственное взаимодействие с принтером.
System.Drawing.Text	Дает возможность управлять коллекциями шрифтов

При создании приложения Windows с использованием мастера, ссылка на System.Drawing.dll устанавливается автоматически.

Обзор пространства имен System.Drawing.

В таблице 10.2 приводится описание основных типов, необходимых при работе с графикой.

Таблица 10.2 – Базовые типы пространства имен System.Drawing

Тип	Описание
Bitmap	Тип, инкапсулирующий данные изображения
Brush, Brushes, SolidBrush, SystemBrushes, TextureBrush	Объекты Brush используются для заполнения внутренних областей графических форм (прямоугольников, эллипсов и т.д.)
BufferedGraphics	Новый тип .NET 2.0, обеспечивающий графический буфер для двойной буферизации, которая используется для уменьшения или полного исключения эффекта мелькания, возникающего при перерисовке

Color SystemColors	Определяют ряд статических свойств, доступных только для чтения и используемых для получения нужного цвета при использовании перьев и кистей
Font, FontFamily	Тип Font инкапсулирует характеристики данного шрифта (название, плотность, размер и т.д.). FontFamily предлагает абстракцию для группы шрифтов, имеющих аналогичный дизайн, но определенные вариации стиля
Graphics	Представляет реальную поверхность нанесения изображения, а также предлагает ряд методов для визуализации текста, изображений и геометрических шаблонов
Pen Pens SystemPens	Pens – это объекты, используемые для построения линий, кривых. Тип Pen определяет ряд статических свойств, возвращающих новый объект Pen заданного цвета.
Point, PointF	Структуры, представляющие точку
Rectangle RectangleF	Структуры, представляющие прямоугольник
Size SizeF	Структуры, представляющие размер (в виде целого или вещественного числа)

Утилитарные типы System.Drawing.

Многие из методов рисования, определенные объектом System.Drawing.Graphics, требуют указать позицию или область, в которой требуется отобразить данный элемент. Другие методы требуют указания размеров, координат, границ геометрического образа.

Тип Point(F) поддерживает ряд полезных членов:

- перегруженные операции: +, -, ==, != ;

- доступ к значениям (x,y): X, Y;

- поле IsEmpty возвращает значение true, если x и y установлены в 0.

Пример:

Типы **Rectangle(F)**, как и Point, необходимы в большинстве GUI-приложений. Одним из наиболее полезных методов типа Rectangle является Contains(), который позволяет выяснить, находится ли данный тип Point или Rectangle в рамках границ некоторого другого объекта.

Пример:

Класс Region представляет внутреннюю часть геометрической фигуры, поэтому конструкторы класса Region требуют, чтобы им на вход был предоставлен некоторый геометрический объект.

Например, имея внутреннюю часть фигуры, вы можете манипулировать ею с использованием различных членов.

Класс Graphics.

Класс System.Windows.Graphics – это интерфейс для функциональных возможностей GDI+. Этот класс не только предоставляет поверхность, на которой размещаются изображения (поверхность формы, поверхность элемента управления или область в памяти), но определяет также члены, которые позволяют отображать текст, изображения (пиктограммы, точечные рисунки и т.д.) и самые разные геометрические формы.

Класс Graphics не допускает непосредственного создания своего экземпляра с помощью ключевого слова new, поскольку класс не имеет открытых конструкторов.

Для получения доступа к объекту Graphics необходимо создатьобработчик события WM_PAINT, для этого для формы необходимо зарегистрировать обработчик события Paint. В качестве параметра метода-обработчика будет передан объект типа PaintEventArgs. В обработчике события Paint можно будет получить объект Graphics следующим образом:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics new_graf = e.Graphics;

    new_graf.DrawRectangle(10, 10, 100, 100);

}
```

Рисунок 10.1 – Получение объекта Graphics через параметр PaintEventArgs обработчика события Paint

10.3.2 Обработка события таймера

Для организации анимации в приложениях Windows часто используется класс Timer. Класс достаточно простой и после создания объекта класса

программисту требуется только одно событие Tick, которое наступает когда указанный интервал таймера заканчивается. Интрвал устанавливается свойством Interval.

Существую еще два метода, необходимых для работы таймера: Start() – запускает таймер; Stop() – останавливает выполнение таймера.

10.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

10.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

10.6. Методика и порядок выполнения работы

10.6.1 Рисование графика функции

Для овладения практическими навыками визуализации результатов расчетов реализуйте пример: создайте приложение для отображения графика по заданной функции: $F = a \cdot x^{-p} \cdot \sin(k \cdot x + b)$.

Для решения поставленной задачи реализуйте следующие действия:

1. Создайте приложение Windows Forms.

2. Спроектируйте форму в соответствии с представленным шаблоном (рис. 10.2).

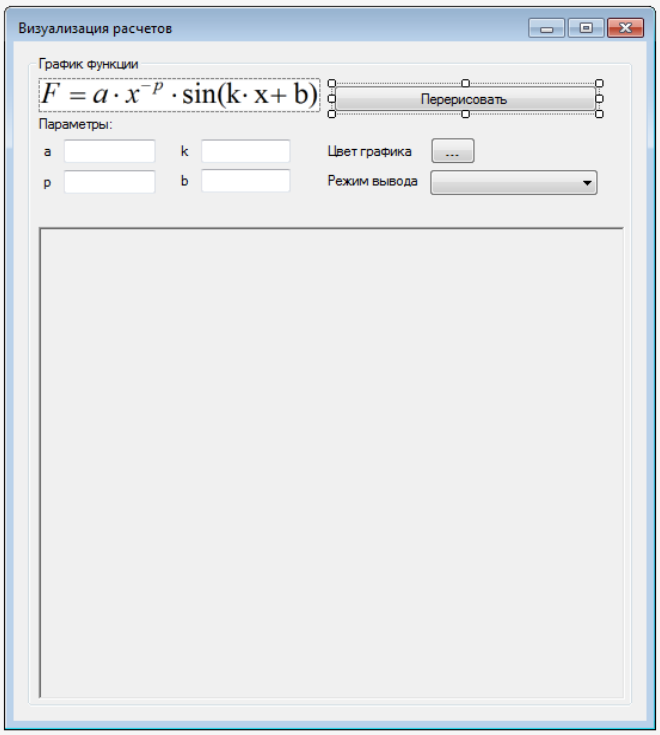


Рисунок 10.2 – Проект формы приложения

В таблице 10.1 приведена дополнительная информация о компонентах, размещенных на форме приложения.

Таблица 10.1 – Элементы управления формы главного окна

Имя элемента управления	Описание
txtA	Элемент управления типа TextBox. Предназначен для ввода коэффициента функции a.
txtP	Элемент управления типа TextBox. Предназначен для ввода коэффициента функции p.

txtK	Элемент управления типа TextBox. Предназначен для ввода коэффициента функции k.
txtB	Элемент управления типа TextBox. Предназначен для ввода коэффициента функции b.
btnChooseColor	Элемент управления типа Button. Предназначен для открытия диалога выбора цвета графика.
comboMode	Элемент управления типа ComboBox. Предназначен для выбора режима отображения графика: точками или в виде линии. Заполните свойство Items и добавьте два элемента: «Точки», «Линия».
panelGraf	Элемент управления типа Panel. Предназначен для отображения графика функции. Свойство Height установлено в 400.
btnRedraw	Элемент управления типа Button. Предназначен для перерисовки графика.

3. В теле класса главной формы приложения определите следующие свойства (рис. 10.3).

```
public partial class Form1 : Form
{
    // Цвет графика
    Color color = Color.Red;
    // Режим отображения
    int mode = 1;
    // Шаг расчета функции
    double dx = 5;
    // Интервал изменения x
    double x0 = 0.0, xn = 400.00;
    // Коэффициенты функции
    double a = 5, b = 100, p = -0.5, k = 2;
```

Рисунок 10.3 – Добавление полей класса

4. В конструктре экранной формы установите значения элементов управления в соответствии со значениями служебных переменных (рис. 10.4).

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

    // Инициализация компонентов формы значениями переменных
    // Выбор режима
    comboMode.SelectedIndex = mode;
    // Установка коэффициентов
    txtA.Text = a.ToString();
    txtB.Text = b.ToString();
    txtK.Text = k.ToString();
    txtP.Text = p.ToString();
}
```

Рисунок 10.4 – Конструктор класса

5. Реализуйте метод вычисления значений функции в соответствии с условием задачи (рис. 10.5).

```
private double MyFunc(double x)
{
    if (x == 0)
        return 0;
    else
        return a * Math.Pow(x, -p) * Math.Sin(k * x + b);
}
```

Рисунок 10.5 – Метод для вычисления значений функции

6. Реализуйте обработчики: нажатия кнопки «Перерисовать» (рис. 10.6), книпки выбора цвета графика (рис. 10.7) и выбора режима рисования графика (рис. 10.8).

```
private void btnRedraw_Click(object sender, EventArgs e)
{
    a = Convert.ToDouble(txtA.Text);
    b = Convert.ToDouble(txtB.Text);
    k = Convert.ToDouble(txtK.Text);
    p = Convert.ToDouble(txtP.Text);
    panelGraf.Invalidate();
}
```

Рисунок 10.6 – Обработчик нажатия кнопки btnRedraw

```
private void btnChooseColor_Click(object sender, EventArgs e)
{
    ColorDialog dlg = new ColorDialog();
    dlg.Color = color;
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // еустанавливаем новый цвет
        color = dlg.Color;
        // перерисовываем панель с графиком
        panelGraf.Invalidate();
    }
}
```

Рисунок 10.7 – Обработчик нажатия кнопки btnChooseColor

```
private void comboMode_SelectedIndexChanged(object sender, EventArgs e)
{
    // Устанавливаем режим рисования графика
    mode = ((ComboBox)sender).SelectedIndex;
    // Перерисовываем панель
    panelGraf.Invalidate();
}
```

Рисунок 10.8 – Обработчик выбора пункта в comboMode

Внешний вид экранной формы разработанного приложения представлен на рис. 10.9

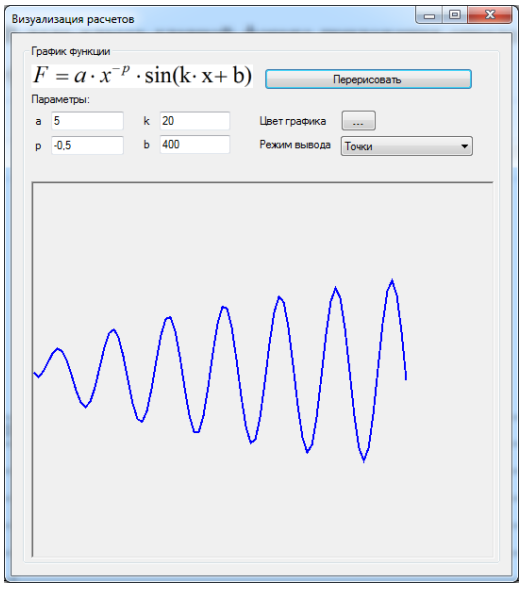


Рисунок 10.9 – Приложение в режиме выполнения

7. Запустите приложение. Протестируйте работу программы при различных значениях параметров.

8. Доработайте приложение: реализуйте механизм вывода графика отдельными точками, так как в представленном варианте график выводится в виде отрезков, соединяющих отдельные точки графика.

9. Разработайте приложение, в котором визуализируются данные сложной функции (учащийся может определить график функции самостоятельно).

10.6.2 Анимация средствами GDI+

1. Создайте приложение Windows Forms.

2. Дополните класс экранной формы вспомогательными переменными (рис. 10.10).

```
public partial class Form2 : Form
{
    private Point pos = new Point(100, 100);
    private int R = 50;
    double T = 0;
```

Рисунок 10.10 – Переменные класса формы

3. Перетащите из панели инструментов компонент Timer на экранную форму. Присвойте ему имя MainTimer. Установите свойство Interval в значение 500.

4. Модифицируйте обработчик события Paint экранной формы (рис. 10.11).

```
private void Form2_Paint(object sender, PaintEventArgs e)
{
    Graphics gr = e.Graphics;
    gr.DrawEllipse(new Pen(Color.Blue, 10), pos.X - R, pos.Y - R, R, R);
}
```

Рисунок 10.11 – Обработчик события Paint

5. Реализуйте обработчик события Tick таймера MainTimer (рис. 10.12).

```
private void MainTimer_Tick(object sender, EventArgs e)
{
    T += 0.1;
    pos.X = (int)(300 + 150 * Math.Sin(T));
    pos.Y = (int)(300 + 100 * Math.Cos(T));
    Invalidate();

}
```

Рисунок 10.12 – Обработчик события Tick

6. Для запуска и остановки таймера добавьте вызов метода Start() в конструктор класса, и вызов метода Stop() в обработчик события FormClosing главной формы (рис. 10.13).

```
public Form2()
{
    InitializeComponent();
    MainTimer.Start();
}
private void Form2_FormClosing(object sender, FormClosingEventArgs e)
{
    MainTimer.Stop();
}
```

Рисунок 10.13 – Запуск и остановка таймера

10.6.3 Интерактивная двумерная графика

Выполните и разберите технологию решения учебного задания: разработайте приложение, которое позволяет пользователю взаимодействовать с GDI-объектами. Необходимо вывести на форму простую фигуру и сделать доступным ее перетаскивание.

Для выполнения задания необходимо выполнить следующие действия:

1. Создайте приложение Windows Forms.

2. Добавить в класс формы служебные переменные (рис. 10.14).

3. Модифицируйте метод обработки события перерисовки окна (рис. 10.15).

```
public partial class Form1 : Form
{
    private Point location = new Point(50,50);
    private int R = 50;
    Boolean DropStarted = false;
```

Рисунок 10.14 – Служебные переменные класса

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // Рисуем круг в точке location
    Graphics gr = e.Graphics;
    gr.DrawEllipse(new Pen(Color.Red, 5), location.X - R, location.Y - R, R, R);
}
```

Рисунок 10.15 – Обработчик Paint

4. Реализуйте обработчик события MouseDown для формы приложения (рис. 10.16).

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    int curX = e.X, curY = e.Y;
    Boolean InX = (curX >= location.X - R && curX <= location.X + R);
    Boolean InY = (curY >= location.Y - R && curY <= location.Y + R);

    if (InX && InY)
        DropStarted = true;
}
```

Рисунок 10.16 – Обработчик MouseDown

5. Реализуйте обработчик события MouseUp для формы приложения (рис. 10.17).

```
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    if (DropStarted)
    {
        location.X = e.X + R/2;
        location.Y = e.Y + R/2;
        Invalidate();
    }
}
```

Рисунок 10.17 – Обработчик MouseUp

6. Запустите приложение, протестируйте механиз перетаскивания объектов.

Варианты индивидуальных заданий

1. Создайте приложение Windows в среде Visual Studio.

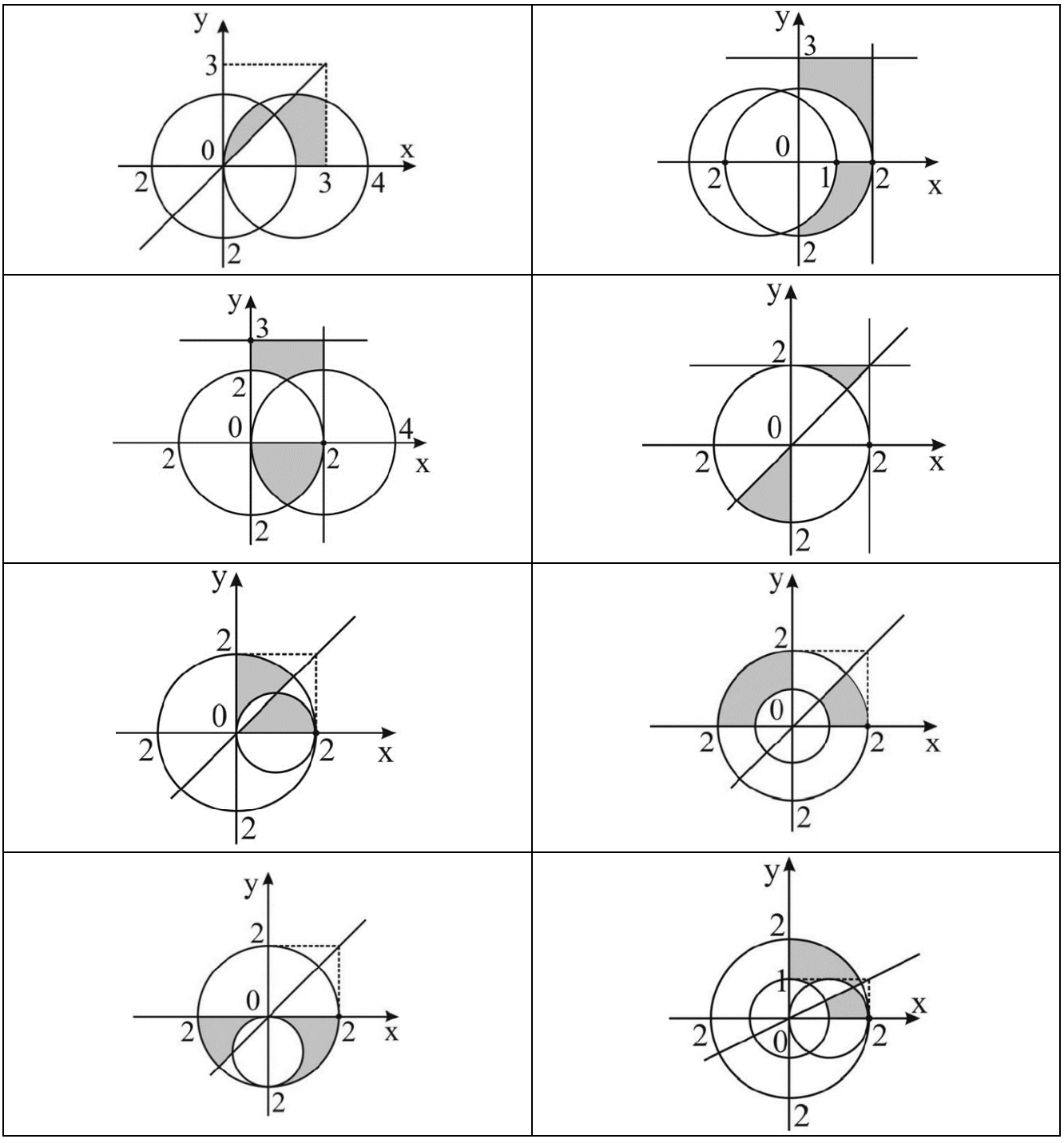
2. Выведите на поверхность формы рисунок, в соответствие с индивидуальным заданием.

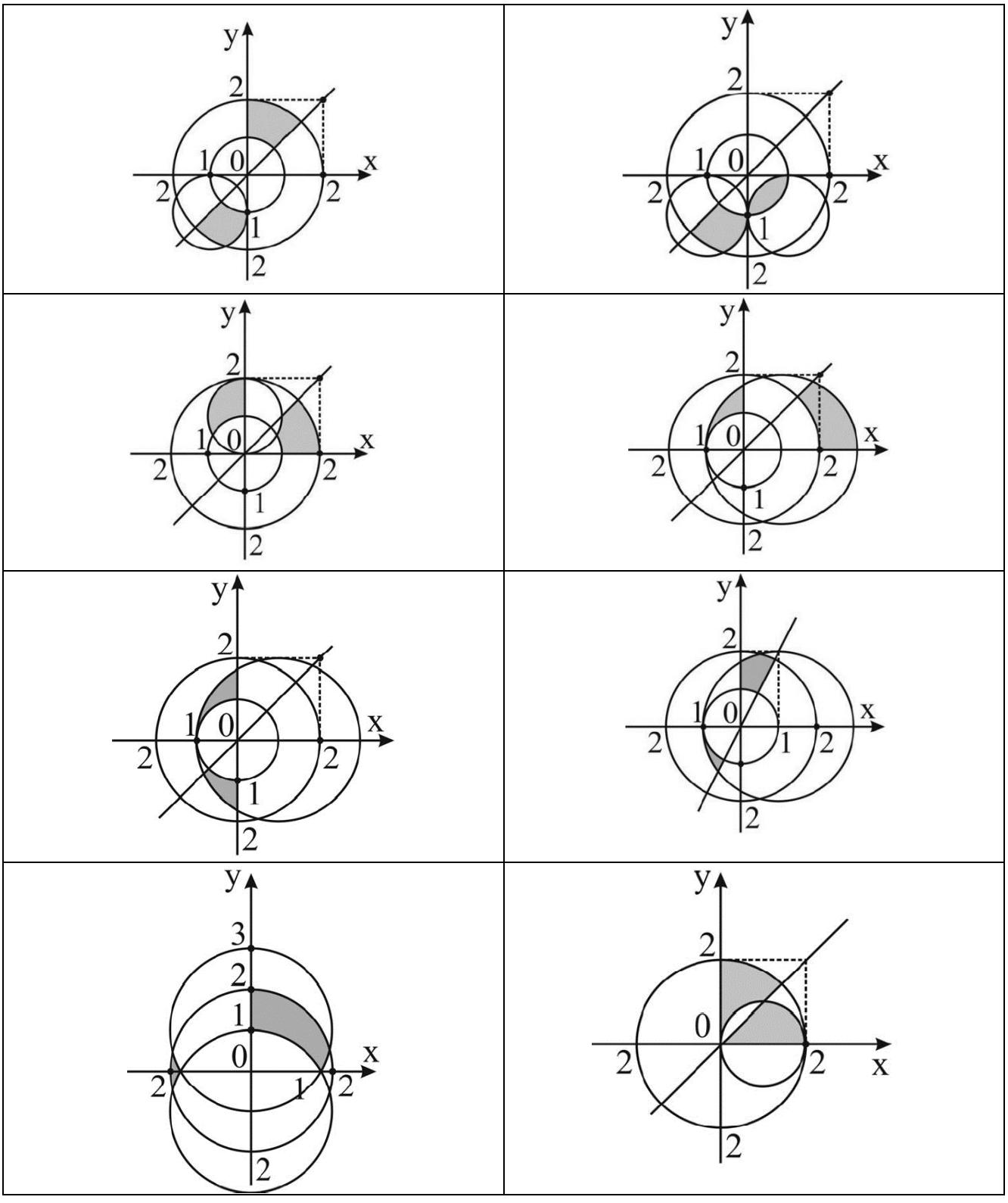
3. Реализуйте обработчик события изменения координат курсора мыши таким образом, чтобы при попадании курсора мыши в закрашенную область, соответствующий закрашенный регион менял свой цвет.

4. Реализуйте возможность выбора цвета цвета закрашенных областей с использованием стандартного диалога операционной системы для выбора цвета.

5. Реализуйте возможность перетаскивания пользователем подписей к рисункам (названия осей, цифры).

6. Реализуйте анимацию: циклическое изменение размеров шрифта подписей, анимацию цвета элементов и т.п. (что анимировать – выбирает учащийся). Необходимо предусмотреть возможность остановки и запуска анимации.





10.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

10.8. Вопросы для защиты работы

1. Назовите основные типы, необходимые для работы с GDI+.

2. Какие пространства имен доступны для работы с графикой?

3. Опишите возможные конструкторы класса Point.

4. В чем состоит особенность конструктора класса Region?

5. Как получить доступ к объекту класса Graphics?

6. Опишите способ, с помощью которого можно реализовать перехват соытия мыши над фигурами GDI+.

7. Для чего предназначе класс GraphicPath?

8. Чем отличаются типы Point и PointF?

9. Какие элементы могут обрабатывать событие Paint?

10. Для чего используется тип PathPointType?

11. Через какой тип происходит рисование в обработчике события Paint?

12. Опишите механизм выбора объекта GDI.

13. Какие события участвуют в процессе интрактивного взаимодействия с GDI объектами графики?

14. Какое событие необходимо задействовать, если добавить в программу отображение контура объекта при перетаскивании?

15. Опишите механизм реализации анимации с использованием GDI+.

16. Опишите назначение основных свойств и методов класса Timer.

17. Для чего предназначено событие Tick класса Timer?

18. Сколько таймеров может присутствовать в рпограмме?

10.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 11. МЕНЮ И СТРОКА СОСТОЯНИЯ В ПРИЛОЖЕНИЯХ WINDOWS FORMS

11.1. Цель и содержание

Цель лабораторной работы: научиться использовать в программах контекстные меню, главное меню приложения и строки состояния.

Задачами лабораторной работы являются:

– научится использовать контекстные меню для различных визуальных элементов управления,

– научиться использовать главное меню приложения,

– научиться координировать главное меню приложения с контекстными меню и другими элементами управления;

– научится создавать строки состояния с различными функциональными возможностями;

– получить практические навыки программирования обработчиков событий от элементов управления «панель инструментов» и «строка состояния».

11.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

11.3. Теоретическое обоснование

11.3.1 Меню в приложениях Windows Forms

В рамках платформы .NET элементом управления для создания системы меню является MenuStrip. Этот элемент управления позволяет создавать как пункты меню, представляющие собой любые подходящие элементы управления. Некоторые общие элементы интерфейса, которые могут содержаться в MenuStrip:

ToolStripMenuItem - традиционный пункт меню;

ToolStripComboBox - встроенный элемент ComboBox;

ToolStripSeparator - простая линия, разделяющая содержимое.

ToolStripTextBox - встроенный элемент TextBox.

MenuStrip содержит строго типизированную коллекцию ToolStripItemCollection. Подобно другим типам коллекции, этот объект поддерживает методы Add(), AddRange(), Remove() и свойство Count. Эта коллекция обычно заполняется не напрямую, а с помощью различных инструментов режима проектирования.

11.3.2 Строка состояния в приложениях Windows Forms

Поддержка элемента управления «строка состояния» реализована в .NET 2.0 использованием класса StatusStrip (также доступен класс StatusBar для проектирования строки состояния). Строка состояния StatusStrip может состоять из произвольного количества панелей, которые могут содержать текстовые и графические данные. Данные панели представлены типом ToolStripStatus. В отличие от StatusBar, элемент управления класса StatuStrip может содержать дополнительные элементы управления, такие как:

1) ToolStripProgressBar – встроенный индикатор выполнения (хода задания);

2) ToolStripDropDownButton – встоенная кнопка, отображающая при щелчке на ней список вариантов;

3) ToolStripSplitButton – отображает варианты списка только при щелчке непосредственно в области раскрывающегосясписка.

11.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

11.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

11.6. Методика и порядок выполнения работы

11.6.1 Изучение классов для создания контекстного и главного меню

1. Создайте приложение Windows Forms в среде MS Visual Studio.
2. Поместите элемент MenuStrip в главную форму в окне проектирования.
3. Создайте меню со следующей структурой:

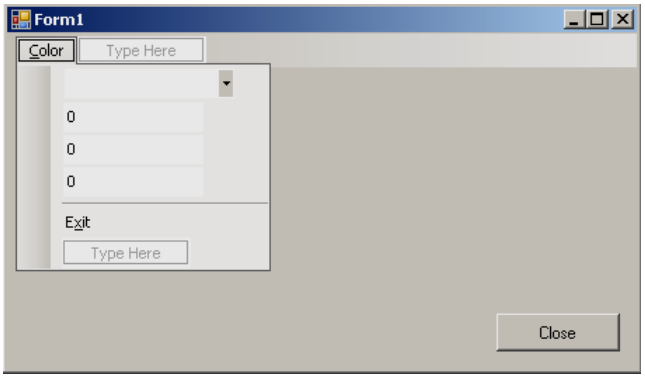


Рисунок 11.1 – Главное меню приложения.

3.1. Для этого свойстве Text пункта меню верхнего уровня установите равным «&Color».

3.2. Первым элементом меню выберите элемент типа ToolStripComboBox, для которого, в режиме проектирования, установите следующие свойства:

DropDownStyle = DropDownList

Items = список значений: белый, красный, черный, синий, желтый

ToolTipText = «Готовые цвета»

3.3. Далее следуют три элемента типа ToolStripTextBox, для каждого задается свойство Text = 0. Дополнительно, для toolStripTextBox1 свойство ToolTipText устанавливается равным «Красный», для toolStripTextBox2 – «Зеленый», toolStripTextBox3 – «Синий».

3.4. Следующий пункт меню типа ToolStripSeparator.

3.5. Последний пункт меню типа ToolStripMenuItem, для которого устанавливаются свойство Text = “E&xit”.

4. После создания меню, необходимо инициализировать начальные значения пункта ToolStripComboBox. Для этого в обработчике события Load главной формы добавьте код:

```
private void Form1_Load(object sender, EventArgs e)
{
    toolStripComboBox1.SelectedIndex = 0;
}
```

5. Для обработки выбора пункта Exit, добавьте в обработчик события Click данного пункта следующий код:

```
private void closeToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

6. Запустите приложение, в случае необходимости исправьте ошибки. Обратите внимание на работу пункта меню Exit.

Следующим этапом является обработка выбора пунктов меню:

Для обработки выбора цвета в первом пункте меню выполните следующие действия:

1. Создайте обработчик события SelectedIndexChanged:

```
private void toolStripComboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    switch (toolStripComboBox1.Text)
    {
        case "белый": ; BackColor = Color.White; break;
        case "красный": ; BackColor = Color.Red; break;
        case "черный": ; BackColor = Color.Black; break;
        case "синий": ; BackColor = Color.Blue; break;
        case "желтый": ; BackColor = Color.Yellow; break;

        default: BackColor = SystemColors.Control;  break;
    }
}
```

Рисунок 11.2 – Обработка выбора цвета в списке.

2. Запустите приложение, протестируйте обработчик.

Для задания RGB-компонент цвета с использованием пунктов меню типа ToolStripTextBox выполните следующие действия:

1. Для первого текстового поля в меню создайте обработчик события TextChanged:

```
private void toolStripTextBox1_TextChanged(object sender, EventArgs e)
{
    try
    {
        BackColor = Color.FromArgb(
            Convert.ToInt32(toolStripTextBox1.Text),
            Convert.ToInt32(toolStripTextBox2.Text),
            Convert.ToInt32(toolStripTextBox3.Text));
    }
    catch
    {
        MessageBox.Show("Необходимо ввести целое чило от 0 до 255", "Ошибка в задании цвета");
    }
```

Рисунок 11.3 – Обработка события изменения значения компоненты цвета.

2. Для второго и третьего полей установите обработчики событий TextChanged. В качестве функции-обработчика события для обоих оставшихся диалоговых окон выбрать из списка обработчик toolStripTextBox1_TextChanged первого окна (которое отвечает за ввод компоненты красного цвета).

Создайте кнопку Close, установите для нее обработчик события Click такой же как у пункта меню Exit.

По аналогии с созданием главного меню приложения создайте контекстное приложения, открывающееся при щелчке правой кнопкой мыши на окне приложения. Для этого:

1. Спроектируйте контекстное меню на основе элемента управления ContextMenuStrip.

2. Укажите значение свойства ContextMenuStrip главной формы равным ContextMenuStrip1 (необходимо указать имя созданного вами контекстного меню).

3. Подключите обработчики событий, созданные для главного меню к пунктам контекстного меню.

11.6.2 Изучение классов для создания строки состояния

1. Создайте приложение Windows Forms в среде MS Visual Studio 2005.

2. Поместите элемент StatusStrip в главную форму в окне проектирования.

3. Поменяйте имя данного элемента управления на MainStatusStrip.

4. Для добавления панелей в строке состояния можно использовать следующие подходы:

– создавать необходимый программный код без использования мастеров;

– добавить нужные элементы в диалоговом окне, появляющемся при выборе ссылки Edit Items (Редактирование элементов) из меню контекстного редактора StatusStrip (рис. 11.4).

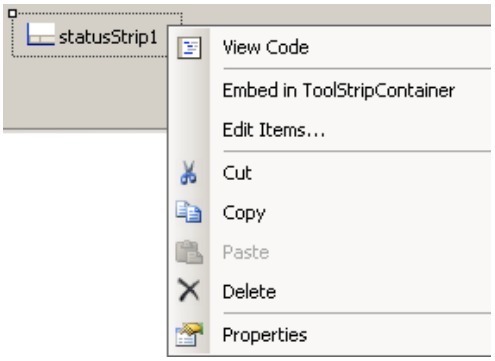


Рисунок 11.4 – Контекстное меню для выбора команды редактирования элементов строки состояния.

– добавить нужные элементы по одному с помощью раскрывающегося меню новых элементов StatusStrip (рис. 11.5).

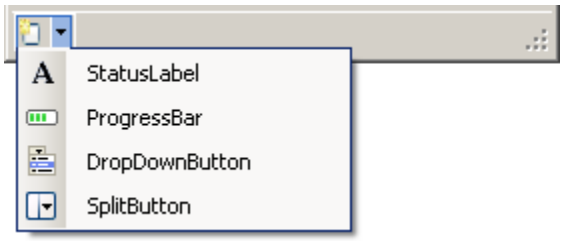


Рисунок 11.5 – Добавление элементов с помощью меню новых элементов.

5. Создайте с использованием вышеописанных методов строку состояния со следующей структурой:

– первый элемент типа ToolStripStatusLabel, для которого необходимо установить следующие свойства: Name = toolStripStatusLabelState, Spring = true, Text = (пусто), TextAlign = TopLeft.

– второй элемент типа ToolStripStatusLabel, для которого необходимо установить следующие свойства: Name = toolStripStatusLabelClock, BorderSides = All, Text = (пусто).

– третий элемент типа ToolStripDropDownButtonDateTime, для которого необходимо создать два элемента меню типа ToolStripMenuItem: первый элемент с Name = toolStripMenuItemDate, Text = «Текущая дата»; второй элемент с Name = toolStripMenuItemTime, Text = «Текущее время».

6. Создайте элемент управления типа Timer с именем timerDateTimeUpdate. Используя окно свойств установите значение свойства Interval в значение 1000 (значение в миллисекундах), а значение Enabled равным True.

7. Определите в проекте новый тип перечня DateTimeFormat, для выяснения того, что должен отображать второй элемент строки состояния – текущее время или текущую дату:

```
public enum DateTimeFormat
{
    ShowTime,
    ShowDate
}
```

8. В классе главного окна определите следующие поля и методы:

8.1. Поле dtFormat типа DateTimeFormat.

8.2. Поле currentCheckedItem типа ToolStripMenuItem.

8.3. Создайте обработчик события Tick для таймера timerDateTimeUpdate с именем timeDateTimeUpdate_Tick.

8.4. Выполните необходимые действия по присвоению начальных значений переменным dtFormat и currentCheckedItem.

8.5. Создайте обработчики события Click (выбор пункта меню) для элементов toolStripMenuItemTime и toolStripMenuItemDate.

После выполнения действий 8.1 – 8.5 класс главного окна будет содержать поля и методы, показанные на рисунке 11.6.

```
public partial class MainForm : Form
    {
        DateTimeFormat dtFormat = DateTimeFormat.ShowTime;
        ToolStripMenuItem currentCheckedItem;

        public MainForm()
        {
            InitializeComponent();
            Text = "Пример строки состояния";
            CenterToScreen();
            BackColor = Color.CadetBlue;
            currentCheckedItem = toolStripMenuItemTime;
            currentCheckedItem.Checked = true;
        }

        private void timerDateTimeUpdate_Tick(object sender, EventArgs e)
        {
            string info = "";
            if (dtFormat == DateTimeFormat.ShowTime)
                info = DateTime.Now.ToLongTimeString();
            else
                info = DateTime.Now.ToLongDateString();
            toolStripStatusLabelClock.Text = info;
        }

        private void toolStripMenuItemDate_Click(object sender, EventArgs e)
        {
            // установка даты
            currentCheckedItem.Checked = false;
            dtFormat = DateTimeFormat.ShowDate;
            currentCheckedItem = toolStripMenuItemDate;
            currentCheckedItem.Checked = true;
        }

        private void toolStripMenuItemTime_Click(object sender, EventArgs e)
        {
            // установка времени
            currentCheckedItem.Checked = false;
            dtFormat = DateTimeFormat.ShowTime;
            currentCheckedItem = toolStripMenuItemTime;
            currentCheckedItem.Checked = true;
        }
    }
```

Рисунок 11.6 – Поля и методы класса главного окна, определяемые программитом.

9. Исправьте ошибки и запустите приложение.

10. Самостоятельно выполните задание: добавьте код, показывающий в строке состояния (на панели toolStripStatusLabelState) координаты курсора мыши.

Выполните задание в соответствие с индивидуальным вариантом, дополнив главное и контекстное меню своими пунктами.

Варианты индивидуальных заданий

Вариант 1.

1.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot x + b \cdot y + \sin(z)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

1.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = x + y$ или $z = x^2 + y^2$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 2.

2.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a^2 \cdot x + (b - a) \cdot y + \sin(z)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

2.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю

выбрать один из вариантов расчета выражения: $z = \frac{x^3 + y}{y^2 + 1}$, $f = \sqrt{y^3}$ или $z = \sqrt{\frac{x}{y+1}} + y$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 3.

3.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot x^2 + b \cdot (y - x) + \lg(a \cdot z)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

3.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \frac{y - \sqrt{x}}{|x - y|}$, $f = \sqrt{\left|\sqrt{x} - \sqrt{y}\right|}$ или $z = \cos x^2 + \sin^2 y$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 4.

4.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{b} + b \cdot \frac{y}{x} + \sin(z)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

4.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и

ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = x^4$, $f = y^3$ или $z = \sqrt{\frac{x}{y}} + y^2$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 5.

5.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot z + b \cdot y + \sin(z)}{a + b}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

5.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \sqrt{|x^3 - y^2|}$, $f = \frac{x^2}{y^3}$ или $z = \sin x$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 6.

6.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot \frac{x}{y} + b \cdot \frac{y}{z} + \sin(\frac{z}{x})$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

6.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю

выбрать один из вариантов расчета выражения: $z = \frac{x^2 + y^3}{|y^3 - x|}$ или «квадраты координат курсора равны (x^2 , y^2)», где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 7.

7.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{b - y} + \frac{b \cdot y + \sin(z)}{a \cdot x}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

7.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \frac{x + y}{|y - x|}$ или «координаты курсора равны (x , y)», где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 8.

8.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot \sqrt{x} + b \cdot y + \cos(x) \cdot \sin(z)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

8.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю

выбрать один из вариантов расчета выражения: $z = \frac{y}{x^2}$, $f = \sqrt{\left|\sqrt{x} - \sin y\right|}$ или $z = \cos x^2 + \sin^2 \sqrt{y}$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 9.

9.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\lg(a \cdot x) + \sin(z) - \cos(y)$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

9.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \frac{y - \sqrt{x}}{|x - y|}$, $f = \sqrt{\left|\sqrt{x} - \sqrt{y}\right|}$ или $z = \cos x^2 + \sin^2 y$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 10.

10.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot x + \frac{b \cdot y}{\lg(z)} + \frac{\sin(z)}{\cos(y)}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

10.1 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю

выбрать один из вариантов расчета выражения: $z = \frac{x}{|y - x^2|}$, $f = \sqrt{|x - \sqrt{y}|}$ или $z = \cos x + \sin y$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 11.

11.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{\cos(z)} + \frac{b \cdot y + \sin(z)}{\sqrt{|x - y|}}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

11.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \frac{x^4}{y + 1}$, $f = \sqrt{|y^3 + x|}$ или $z = \sqrt{|x - y|}$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 12.

12.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{|z|} + b \cdot y + \frac{\sin(z)}{\sqrt{|x^2 - y|}}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

12.1 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и

ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = x^4$, $f = y^3$ или $z = \sqrt{\frac{x}{y}} + y^2$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 13.

13.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{y} + \frac{b \cdot y + \sin(z)}{\left|x - y^2\right|}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

13.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \frac{x^3 + y}{y^2 + 1}$, $f = \sqrt{y^3}$ или $z = \sqrt{\frac{x}{y+1}} + y$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 14.

14.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\sqrt{\left|a \cdot x + b \cdot y + \sin(z)\right|}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

14.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и

ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = x^3$, $f = \sqrt{y}$ или $z = \sqrt{x} + y^2$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

Вариант 15.

15.1 Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\sqrt{\left|a \cdot x \cdot \sqrt{\left|b \cdot y \cdot \sqrt{|z|}\right|}\right|}$, где x, y, z - неизвестные, вводимые в поля ToolStripTextBox; a, b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

15.2 Создайте элемент «строка состояния», который содержит панели следующих типов: ToolStripDropDownButton (панель выбора) и ToolStripStatusLabel (отображения текста). Первая панель предлагает пользователю выбрать один из вариантов расчета выражения: $z = \sqrt{x} + \sqrt{y}$ или $z = \left|x^2 - y^2\right|$, где x и y – текущие координаты указателя мыши. Значение выбранного выражения на второй панели должно изменяться при смене положения курсора мыши.

11.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

11.8. Вопросы для защиты работы

1. Какие элементы управления используются для создания главного меню приложения?
2. Какие элементы управления используются для создания контекстного меню элемента управления?
3. Как в режиме разработки указать элементу управления его контекстное меню?
4. Какие типы данных используются для создания пунктов меню?
5. Поясните механизм синхронизации событий от разных элементов управления.
6. Поясните назначение конструкции try … catch …
7. Какие классы используются для создания строки состояния в .NET?
8. Какие классы используются для создания панелей строки состояния? Каково назначение каждого из этих классов?
9. Какой класс используется для работы с таймером? Какое событие элемента-таймера необходимо обрабатывать для реагирования на смену системного времени?
10. С помощью какого класса можно получить текущие дату и время? Какие методы содержит данный класс?
11. Что позволяет настраивать свойство Spring элементов-панелей строки состояния?

11.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 12. ДИАЛОГОВЫЕ ОКНА

12.1. Цель и содержание

Цель лабораторной работы: Научиться создавать формы верхнего уровня и модальные диалоговые окна.

Задачами лабораторной работы являются:

– создание окон различного типа (модальные и нет) в ответ на события мыши,

– научиться реализовывать обмен данными между модальными диалоговыми окнами и главной формой;

– изучить принципы работы с диалоговым окном «Выбор цвета»;

– изучить принципы работы с диалоговым окном «Выбор шрифта»;

– изучить принципы работы с диалоговыми окнами, предназначенными для работы с каталогами;

– изучить принципы работы с диалоговыми окнами, предназначенными для работы с файлами.

12.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

12.3. Теоретическое обоснование

12.3.1 Пространство имен System.Windows.Forms

Основные компоненты пространства имен System.Windows.Forms:

1. System.Windows.Forms компонуется из различных классов, структур, делегатов, интерфейсов и перечней. Сотни типов пространства имен System.Windows.Forms можно объединить в следующие большие категорий.

2. Базовая инфраструктура. Это типы, представляющие базовые операции программы .NET Forms (Form, Application и т.д.), а также различные типы,

обеспечивающие совместимость с разработанными ранее элементами управления ActiveX.

3. Элементы управления. Все типы, используемые для создания пользовательского интерфейса (Button, MenuStrip, ProgressBar, DataGridView и т.д.), являются производными базового класса Control, Элементы управления конфигурируются в режиме проектирования и оказываются видимыми (по умолчанию) во время выполнения.

4. Компоненты. Это типы, не являющиеся производными базового класса Control, но тоже предлагающие визуальные инструменты (ToolTip, ErrorProvider и т.д.) для программ .NET Forms, Многие компоненты (например, Timer) во время выполнения не видимы, но они могут конфигурироваться визуально в режиме проектирования.

5. Диалоговые окна общего вида. Среда Windows Forms предлагает целый ряд стандартных заготовок диалоговых окон для выполнения типичных действий (OpenFileDialog, PrintDialog и т.д.). Кроме того, вы можете создавать свои собственные пользовательские диалоговые окна, если стандартные диалоговые окна по какой-то причине вам не подойдут.

Общее число типов в System.Windows.Forms намного больше 100. В таблице 12.1 описаны наиболее важные из типов System.Windows.Forms, предлагаемых в .NET Framework.

Таблица 12.1 – Базовые типы пространства имен System.Windows.Forms

Классы	Описание
Application	Класс, инкапсулирующий средства поддержки Windows Forms, необходимые любому приложению
Button, CheckBox, ComboBox, DateTimePicker, ListBox, LinkLabel, MaskedTextBox, MonthCalendar, PictureBox, TreeView	Классы, которые (вместе со многими другими классами) определяют различные GUI- элементы
FlowLayoutPanel, TableLayoutPanel	Платформа .NET 2.0 предлагает целый набор администраторов оформления, выполняющих автоматическую корректировку размещения

	элементов управления в форме при изменении ее размеров
Form	Тип, представляющий главное окно, диалоговое окно или дочернее окно MDI в приложении Windows Forms
ColorDialog, OpenFileDialog, SaveFileDialog, FontDialog, PrintPreviewDialog, FolderBrowserDialog	Представляют различные диалоговые окна, соответствующие стандартным операциям в рамках GUI
Menu, MainMenu, Menultem, ContextMenu, MenuStrip, ContextMenuStrip	Типы, используемые для построения оконных и контекстно-зависимых систем меню. Новые (появившиеся в .NET 2.0) элементы управления MenuStrip и ContextMenuStrip позволяют строить меню, содержащие как традиционные пункты меню, так и другие элементы управления (окна текста, комбинированные окна и т.д.)
StatusBar, Splitter, ToolBar, ScrollBar, StatusStrip, ToolStrip	Типы, используемые для добавления в форму стандартных элементов управления

12.3.2. Создание окон.

Все окна являются объектами типа Form из пространства имен System.Windows.Forms. Поэтому создание диалогового или модального окна отличается только свойствами, устанавливаемыми для данного окна в окне Properties среды разработки (свойство FormBorderStyle) и методом, которым форма выводится на экран:

Form2.Show() – для немодальной формы.

Form2.ShowDialog() – для модального диалогового окна.

В случае использования диалоговых окон, доступны различные варианты возвращения результатов функцией ShowDialog(). Результат имеет тип DialogResult.

12.3.3 Стандартные диалоговые окна библиотеки

При программировании графических приложений выделяют стандартные задачи, которые часто повторяются на практике: выбор объекта файловой системы, выбор шрифта, выбор цвета. Для решения подобных задач широко используются стандартные диалоговые окна библиотке .NET Framework.

На рис. 12.1 представлен фрагмен панели инструментов, содержащий невизуальные компоненты, которые могут быть размещены в лотке элементов управления формы в режиме проектирвоания.

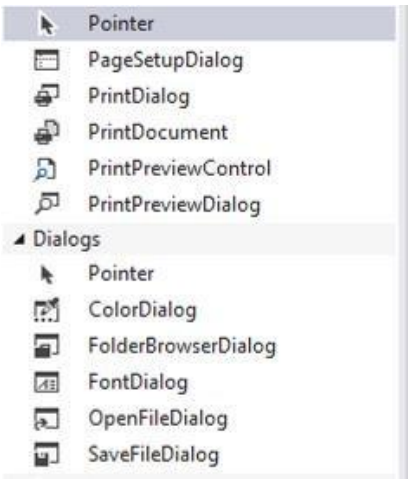


Рисунок 12.1 – Компоненты, обеспечивающие работу с встроенными диалоговыми окнами.

На рис. 10 показаны иконки диалогов, которые доступны программисту: PageSetupDialog, PrintDialog, PrintPreviewDialog, ColorDialog, FolderBrowserDialog, FontDialog, OpenFileDialog, SaveFileDialog.

После переноса иконки из панели инструментов на экранную форму в режиме проектирвания, в лотке компонентов появляется соответствующий компонент. Следует понимать, что подобные диалоги можно создавать и настраивать как в визуальном режиме (режим проектирования), так и динамически непосредственно в коде (режим выполнения).

12.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

12.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

12.6. Методика и порядок выполнения работы

12.6.1 Пользовательские диалоговые окна

1. Измените программу, созданную в лабораторной работе №9 таким образом, чтобы все поля для ввода значений неизвестных переменных были доступны только для чтения (установить свойство ReadOnly). Рядом с полями TextBox разместите кнопки для открытия диалогового окна, например:

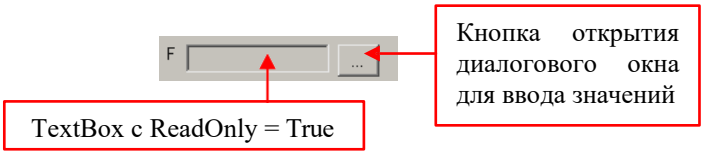


Рисунок 12.2 – Элемент управления, составленный из нескольких стандартных компонентов.

2. Ввод значений в каждое окно должен производиться путем открытия диалогового окна с полем ввода значения и двумя кнопками OK и Cancel. (для обоих кнопок необходимо установить свойства DialogResult для последующего анализа возвращаемых значений методом ShowDialog).

3. Данное диалоговое окно открывается по нажатию специальной кнопки, расположенной рядом с каждым текстовым окном для ввода значений. После указания значения и нажатия кнопки OK, диалоговое окно закрывается и введенное значение переносится в соответствующее поле.

4. После разработки всех визуальных компонент и форм, вычислите выражения в соответствии с вариантом индивидуального задания. Результат вывести в текстовое поле.

12.6.2 Стандартные диалоговые окна

Для изчения принципов работы со стандартными диалоговыми окнами реализуем приложение, которое демонстрирует работу со встроенными диалоговыми окнами.

1. Перемстите на форму приложения компоненты в соотвтестви с рис. 12.3.

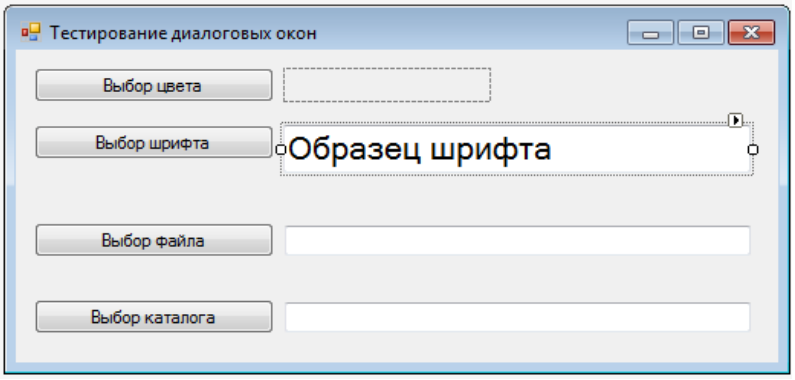


Рисунок 12.3 – Внешний вид экранной формы для работы со стандартными диалоговыми окнами

В таблице 12.2 перечислены компоненты формы и указаны некоторые из их свойств.

Таблица 12.2 – Компоненты формы.

Компонент	Описание
btnChooseColor	Тип Button. Создается для выбора открытия диалога выбора цвета. Свойство Text установлено в «Выбор цвета».
panelChooseColor	Тип Panel. Создается для демонстрации выбранного цвета.
btnChooseFont	Тип Button. Создается для открытия диалога выбора шрифта. Свойство Text установлено в «Выбор шрифта».

txtChooseFont	Тип TextBox. Создается для демонстрации внешнего вида выбранного шрифта.
btnChooseFile	Тип Button. Создается для открытия диалога выбора файла. Свойство Text установлено в «Выбор файла».
txtChoseFile	Тип TextBox. Отображает полный путь к выбранному файлу.
btnChooseFolder	Тип Button. Создается для открытия диалога выбора каталога. Свойство Text установлено в «Выбор каталога».
txtChooseFolder	Тип TextBox. Отображает полный путь к выбранному каталогу.

2. Создайте обработчики нажатия каждой кнопки.

На рис. 12.4 представлен листинг обработчика нажатия btnChooseColor.

```
private void btnChooseColor_Click(object sender, EventArgs e)
{
    // Создание диалогового окна
    ColorDialog dlg = new ColorDialog();
    // Открытие полной версии диалога
    dlg.FullOpen = true;
    // Разрешить пользователю получать справку
    dlg.ShowHelp = true;
    // Установка цвета панели в качестве начального цвета диалога
    dlg.Color = panelChooseColor.BackColor;

    // Получение результата выполнения диалога
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // Установка цвета панели
        panelChooseColor.BackColor = dlg.Color;
    }
}
```

Рисунок 12.4 – Обработчик btnChooseColor

На рис. 12.5 представлен обработчик выбора шрифта.

```
private void btnChooseFont_Click(object sender, EventArgs e)
{
    // Создание диалогового окна
    FontDialog dlg = new FontDialog();
    dlg.ShowColor = true;
    dlg.ShowHelp = true;
    dlg.Font = txtChooseFont.Font;

    // Получение результата выполнения диалога
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // Установка шрифта текстового поля и его цвета
        txtChooseFont.Font = dlg.Font;
        txtChooseFont.ForeColor = dlg.Color;
    }
}
```

Рисунок 12.5 – Обработчик btnChooseFont

На рис. 12.6 представлен код обработчика нажатия кнопки btnChooseFolder.

```
private void btnChooseFolder_Click(object sender, EventArgs e)
{
    // Создание диалогового окна
    FolderBrowserDialog dlg = new FolderBrowserDialog();
    dlg.Description = "выберите папку для демонстрации работы диалога";
    dlg.ShowNewFolderButton = true;
    dlg.SelectedPath = Application.StartupPath;

    // Получение результата выполнения диалога
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // Установка выбранного пути
        txtChooseFolder.Text = dlg.SelectedPath;
    }
}
```

Рисунок 12.6 – Обработчик btnChooseFolder

На рис. 12.7 показан диалог выбора файла.

```
private void btnChooseFile_Click(object sender, EventArgs e)
{
    // Создание диалогового окна
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.InitialDirectory = Application.StartupPath;
    dlg.Filter = "txt files (*.txt)|*.txt |" +
        "Мои файлы (расширения не придумал)|*.xxx|" +
        "Сборки (*.exe)|*.exe";
    dlg.FilterIndex = 3;
    dlg.Title = "Выбор моего файла";


    // Получение результата выполнения диалога
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        // Установка выбранного пути к файлу
        txtChoseFile.Text = dlg.FileName;
    }
}
```

Рисунок 12.7 – Обработчик btnChooseFile

3. Реализуйте все обработчики, запустите приложение, убедитесь в его работоспособности. Измените различные настройки диалоговых окон.

Варианты индивидуальных заданий

1. Вычислить значение выражения: $U = 1 - \frac{\cos^2(x \cdot t)}{2} + \frac{\sin^3(x \cdot z)}{3} - \frac{\cos^4(x \cdot t)}{4} + \frac{\sin^5(x \cdot z)}{5} - ...,$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

2. Вычислить значение выражения: $U = 1 - \frac{\cos^2(x)}{2!} + \frac{\sin^3(x)}{3!} - \frac{\cos^4(x)}{4!} + \frac{\sin^5(x)}{5!} - ...$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

3. Вычислить значение выражения: $y = \frac{\sin(x) \cdot x^2}{2!} - \frac{x^2 \cdot \sin(x^3)}{3!} + \frac{\sin(x^3) \cdot x^4}{4!} - \frac{x^4 \cdot \sin(x^5)}{5!} + ...,$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

4. Вычислить значение выражения: $h = \frac{x^2}{1 \cdot 3} - \frac{\sin(x^4)}{3 \cdot 5} + \frac{x^6}{5 \cdot 7} - \frac{\sin(x^8)}{7 \cdot 9} + ...,$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

5. Вычислить значение выражения: $U = 1 - \frac{\sin_2(x) \cdot y}{2} + \frac{\sin_3(x) \cdot y_3}{3} - \frac{\sin_4(x) \cdot y_5}{4} + \frac{\sin_5(x) \cdot y_7}{5} - ...$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

6. Вычислить значение выражения: $Z = 1 - \frac{\sin(x^2)}{2} + \frac{\cos(x^3)}{3} - \frac{\sin(x^4)}{4} + \frac{\cos(x^5)}{5} - ...,$ если количество слагаемых в правой части выражения и целое x пользователь вводит с клавиатуры.

7. Вычислить значение выражения: $U = 1 - \frac{\sin^2(x)}{2} + \frac{\sin^3(x)}{3} - \frac{\sin^4(x)}{4} + \frac{\sin^5(x)}{5} - ...,$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

8. Вычислить значение выражения: $V = \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + ...,$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

9. Вычислить значение выражения: $s = -\frac{x^2}{2 \cdot 3} + \frac{x^3}{3 \cdot 4} - \frac{x^4}{4 \cdot 5} + \frac{x^5}{5 \cdot 6} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

10. Вычислить значение выражения: $h = \frac{y_3 \cdot x_2}{1 \cdot 3} - \frac{y_5 \cdot x_4}{3 \cdot 5} + \frac{y_7 \cdot x_6}{5 \cdot 7} - \frac{y_9 \cdot x_8}{7 \cdot 9} + \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

11. Вычислить значение выражения: $p = -\frac{x}{1 \cdot 2 \cdot 3} + \frac{x^3}{3 \cdot 4 \cdot 5} - \frac{x^5}{5 \cdot 6 \cdot 7} + \frac{x^7}{7 \cdot 8 \cdot 9} - \ldots$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

12. Вычислить значение выражения: $T = -\frac{x}{1 \cdot 3} + \frac{x^3}{3 \cdot 5} - \frac{x^5}{5 \cdot 7} + \frac{x^7}{7 \cdot 9} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

13. Вычислить значение выражения: $j = -\frac{\sin^3(x)}{1 \cdot 3} + \frac{\sin^5(x^2)}{3 \cdot 5} - \frac{\sin^7(x^3)}{5 \cdot 7} + \frac{\sin^9(x^4)}{7 \cdot 9} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

14. Вычислить значение выражения: $Z = 1 - \frac{\sin(x^2)}{2} + \frac{\cos(x^3)}{3} - \frac{\sin(x^4)}{4} + \frac{\cos(x^5)}{5} - \ldots$, если количество слагаемых в правой части выражения и целое x пользователь вводит с клавиатуры.

15. Вычислить значение выражения: $F = -\frac{x^2}{2 \cdot 3} + \frac{y^3}{3 \cdot 4} - \frac{x^4}{4 \cdot 5} + \frac{y^5}{5 \cdot 6} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

16. Вычислить значение выражения: $s = -\frac{y \cdot x^2}{1 \cdot 3} + \frac{z \cdot x^3}{2 \cdot 4} - \frac{y \cdot x^4}{3 \cdot 5} + \frac{z \cdot x^5}{4 \cdot 6} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

17. Вычислить значение выражения: $E = -\frac{\sin(y)\cdot x^2}{2\cdot 3} + \frac{\sin(x^3)}{3\cdot 4} - \frac{\cos(y)\cdot x^4}{4\cdot 5} + \frac{\sin(x^5)}{5\cdot 6} - \frac{\sin(y)\cdot x^6}{4\cdot 5} + ...$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

18. Вычислить значение выражения: $p = -\frac{y\cdot x}{1\cdot 2\cdot 3} + \frac{y^2\cdot x^3}{3\cdot 4\cdot 5} - \frac{y^3\cdot x^5}{5\cdot 6\cdot 7} + \frac{y^4\cdot x^7}{7\cdot 8\cdot 9} - ...$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

19. Вычислить значение выражения: $p = -\frac{t\cdot \sin(a)}{1\cdot 2} + \frac{t^3\cdot \cos(a^2)}{3\cdot 4} - \frac{t^5\cdot \sin(a^3)}{5\cdot 6} + \frac{t^7\cdot \cos(a^4)}{7\cdot 8} - ...$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

20. Вычислить значение выражения: $A = -\frac{\sin(t)\cdot \lg(a)}{1!} + \frac{\ln(t^3)\cdot \cos(a^2)}{3!} - \frac{\sin(t^5)\cdot \lg(a^3)}{5!} + \frac{\ln(t^7)\cdot \cos(a^4)}{7!} - ...$ если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

12.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

12.8. Вопросы для защиты работы

1. Как создаются модальные диалоговые окна?
2. Что такое пространство имен?
3. Какие пространства имен использованы в вашей программе? Какие типы данных из этих пространств использовались?
4. Опишите механизм обработки событий для визуальных элементов управления.
5. С каким модификатором доступности создаются элементы управления формы?
6. Как возвращается информация о нажатой кнопке закрытия модального диалогового окна? Какое свойство диалогового окна используется для этого? Значение какого типа возвращает метод ShowDialog?
7. Для чего предназначен класс OpenFileDialog? Назовите основные методы и свойства класса.
8. Для чего предназначен класс SaveFileDialog? Опишите основные методы и свойства данного класса.
9. Опишите принцип работы с классом ColorDialog. Опишите основные методы и свойства данного класса.
10. Опишите принципы работы с классом FontDialog. Опишите основные методы и свойства данного класса.

12.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 13. МНОГОМОДУЛЬНЫЕ ПРИЛОЖЕНИЯ

13.1. Цель и содержание

Цель лабораторной работы: научиться проектировать консольные приложения на основе нескольких сборок.

Задачи лабораторной работы:

– научиться управлять несколькими разрабатываемыми проектами в одном решении;

– научиться организовывать взаимодействие нескольких модулей приложения.

13.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

13.3. Теоретическое обоснование

В терминологии технологической платформы .NET не используется понятие «исполняемый файл». Подобные файлы (*.dll или *.exe) характерны для компиляторов небезопасного кода.

В технологии .NET используется термин «сборка». Сборка – это модуль, в котором хранится компилированный управляемый код. Она похожа на классический исполняемый файл (*.exe) или dll-библиотеку, но имеет важное свойство – она полностью себя описывает. Сборки содержат метаданные, которые включают в себя сведения о сборке и обо всех определенных внутри нее типах, методах и т.д. Сборка может быть частной (доступной только одному приложению) или разделяемой (доступной любому приложению Windows).

В предыдущих лабораторных работах студентам предлагалось спроектировать консольные приложения, которые представляют собой одномодульные приложения – exe-файлы.

В данной лабораторной работе требуется спроектировать приложение, которое состоит из нескольких сборок (один модуль – файл *.exe, остальные – dll).

13.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

13.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

13.6. Методика и порядок выполнения работы

Для выполнения лабораторной работы спроектируем следующее приложение:

Требуется разработать приложение-опрос. Пользователю последовательно задаются вопросы, за каждый вариант ответа начисляются определенные баллы. После завершения опросы выводится сумма набранных баллов и какое-либо резюме. Такая программа может использоваться для оценки остаточных знаний в форме тестирования, для составления опросов, для самообследования и т.п.

Перед началом создания приложения, то есть перед непосредственным программированием, необходимо определиться: какие объекты и явления предметной области и связи между ними должен выявить программист. Очевидно, что это:

– вопрос;

– ответ (несколько для каждого вопроса);

– балл соответствующий конкретному ответу;

– итоговые резюме или оценки, которые выводятся в зависимости от количества набранных баллов.

Второе, что необходимо сделать осуществить физическую декомпозицию данного приложения, то есть сколько отдельных модулей будет содержать приложение. В данном случае:

– 1-ый модуль в виде файла *.exe для запуска приложения;

– 2-ой модуль в виде файла *.dll, содержащий вопросы и ответы;

– 3-ий модуль в виде файла *.dll, содержащий сообщения о результатах тестирования (опроса).

Декомпозиция, логическая и физическая, завершена. Для реализации прложения необходимо выполнить следующие шаги:

1. Создайте консольное приложение в среде MS VS (проект назовите Quiz). После выполнения всех необходимых действий окно Solution Explorer примет следующий вид:

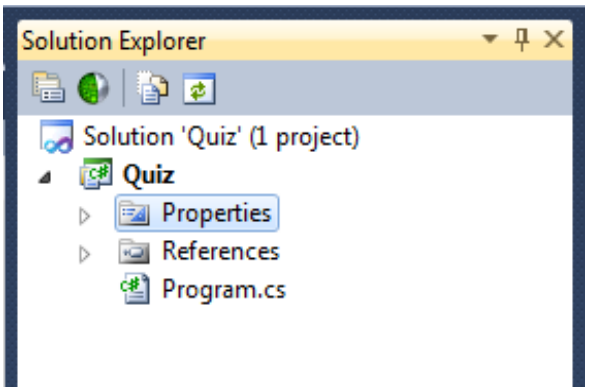


Рисунок 13.1 – Структура одномодульного приложения

В окне отображено одно решение (Solution) с именем «Quiz» и один проект в рамках решения (с именем «Quiz»). Проект содержит файл кода Program.cs, в котором определена функция Main. Этот проект будет откомпилирован в файл Quiz.exe (то есть файл для запуска).

2. Создадим еще один файл в рамках проекта Quiz, который будет содержать класс Testing, позволяющий реализовывать логику тестирования, то есть вывод вопросов в определенной последовательности, ввод выбора пользователя и т.д. Для этого вызовем контекстное меню проекта (рис. 13.2) и создадим новый файл с именем Testing.cs.

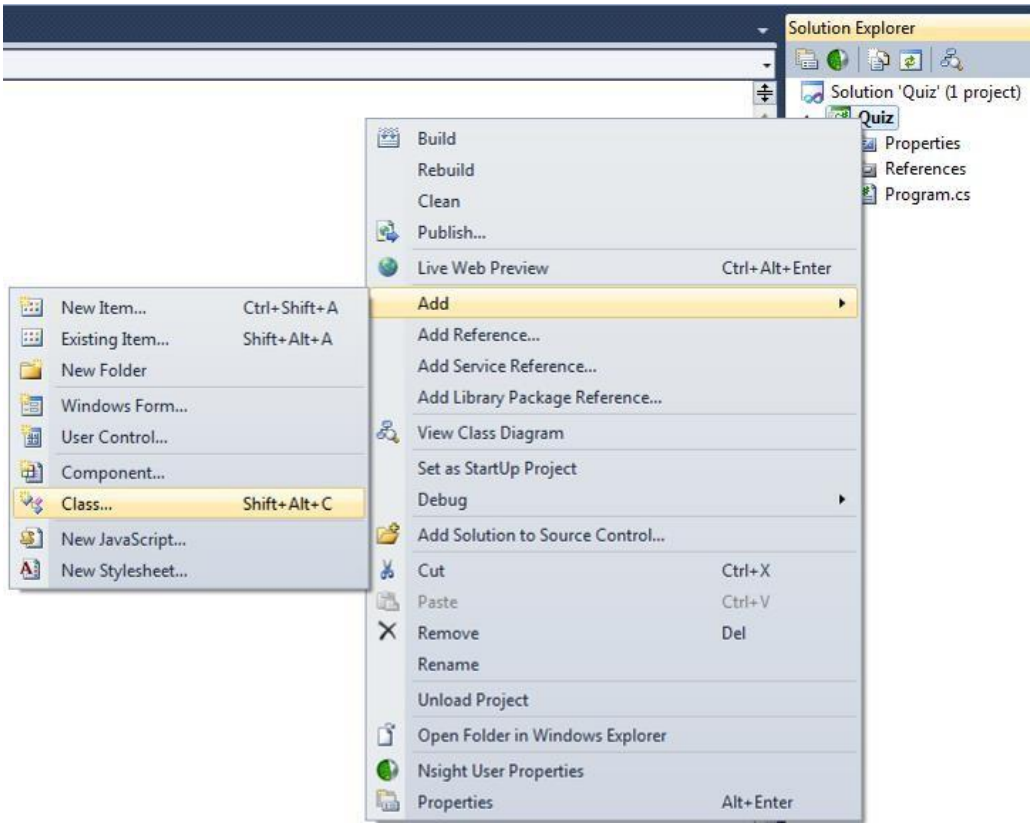


Рисунок 13.2 – Добавления нового класса к проекту

После выполнения данного действия в окне Solution Explorer состав проекта изменится (рис. 13.3).

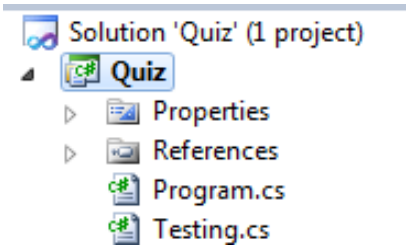


Рисунок 13.3 – Проект с несколькими файлами исходного кода

На данном этапе класс Testing не содержит никакой логики.

3. Создадим второй модуль в виде dll-библиотеки, которая будет содержать вопросы и ответы. Для этого вызовем команду контекстного меню решения Quiz (рис. 13.4).

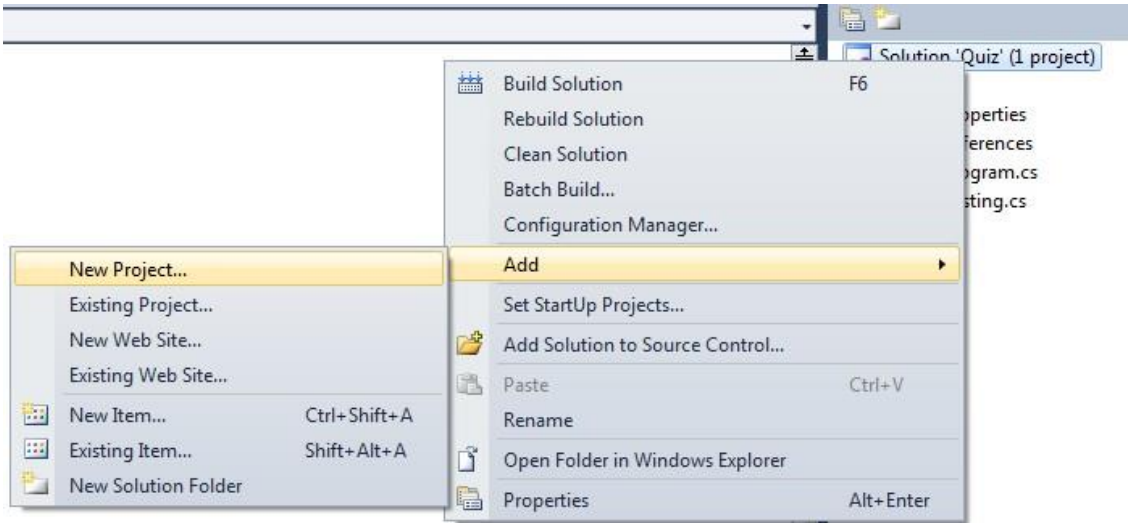


Рисунок 13.4 – Добавление проекта к решению VS

4. В появившемся диалоговом окне выберем тип проекта «Class Library», в качестве имени проекта укажем «Task» (рис. 13.5)

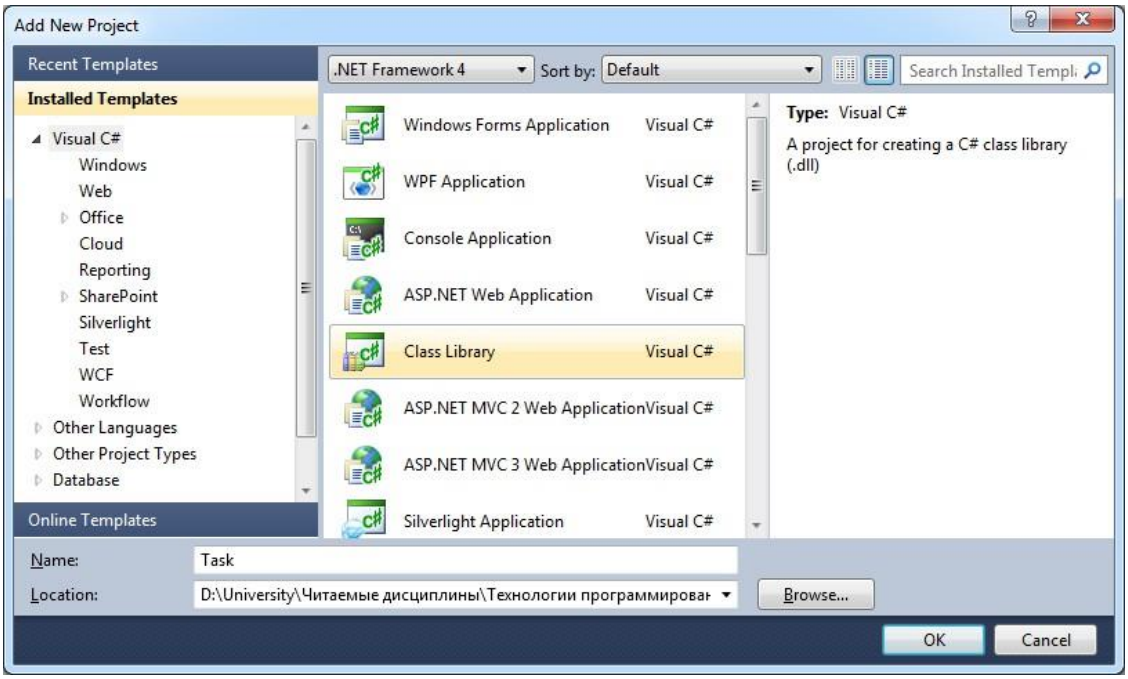


Рисунок 13.5 –Добавление библиотеки классов к решению Quiz.

После добавления нового проекта окно Solution Explorer примет вид, показанный на рис. 13.6

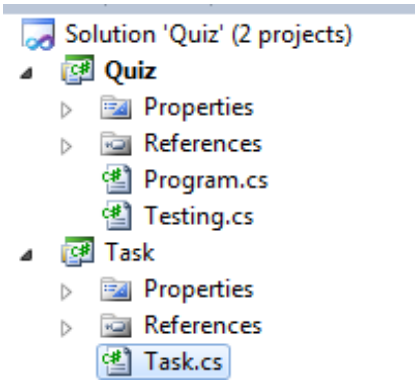


Рисунок 13.6 –Решение с несколькими проектами.

Два проекта в одном решении Quiz, при этом проект Quiz помечен полужирным шрифтом. Это означает, что при вызове команды Run будет запущен именно этот проект. Следует обратить внимание, что файл Task.cs проекта Task содержит только пустой класс без кода. Проект Task в целом не содержит функции Main, то есть не может быть запушен на выполнение.

5. Аналогичным образом добавим к решению еще один проект с именем «Result» и типом «Class Library». Окно Solution Explorer примет вид:

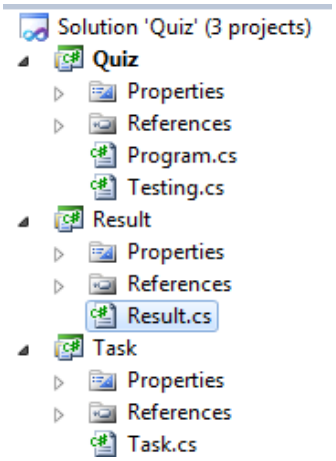


Рисунок 13.7 – Все требуемые проекты, в соответствие с физической декомпозицией, добавлены к решению

Далее переходим к наполнению классов требуемым функционалом.

6. Модифицируем файл Task.cs проекта Task следующим образом:

```
namespace Task
{
    // Класс для представления одного тестового задания
    public class SingleTest
    {
        public string Question {get; set;} // вопрос
        public List<string> Answers; // Ответы
        public List<int> Balls; // Баллы за ответы
    }

    // Статический класс для представления списка тестовых заданий
    public static class Task
    {
        // Список отдельных вопросов с ответами
        public static List<SingleTest> AllQuestions;

        // Статический конструктор для инициализации коллекции
        static Task()
        {
            AllQuestions = new List<SingleTest>()
            {
                new SingleTest()
                {
                    Question = "Main - точка входа программы ...",
                    Answers = new List<string>() { "1", "7", "Не знаю" },
                    Balls = new List<int>() { 20, 5, 0 }
                },
                new SingleTest()
                {
                    Question = "Какой оператор не является оператором цикла?",
                    Answers = new List<string>() { "while", "for", "if" },
                    Balls = new List<int>() { -5, 0, 10 }
                },
                new SingleTest()
                {
                    Question = "Как обозначить составной оператор?",
                    Answers = new List<string>() { "{ ... }", "( ... )", "< ... >" },
                    Balls = new List<int>() { 50, 0, 0 }
                },
                new SingleTest()
                {
                    Question = "Что обозначает символ ; ",
                    Answers = new List<string>() { "Пустой оператор", "Конец программы", "Не знаю" },
                    Balls = new List<int>() { 50, 20, 0 }
                }
            };
        }
    }
}
```

Рисунок 13.8 – Классы проекта Task

Следует обратить внимание на то, что класс Task статический, то есть не требует создания объектов.

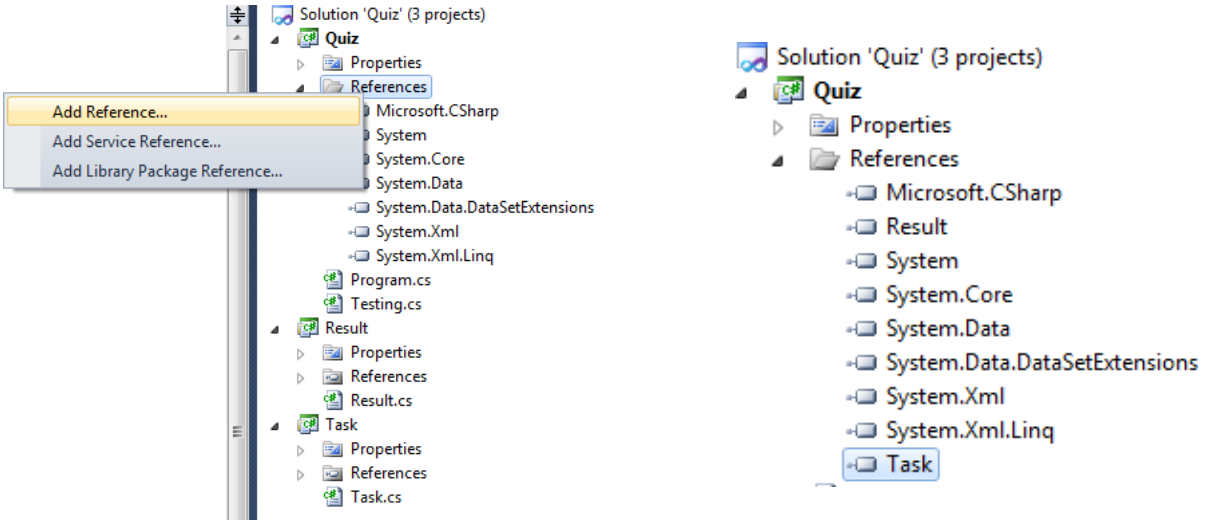
7. Модифицируйте файл Result.cs проекта Result следующим образом:

```
namespace Result
{
    public static class Result
    {
        public static List<string> Messages;

        // Статический конструктор
        static Result()
        {
            Messages = new List<string>()
            {
                "Вы набрали менее 10 баллов! Плохо!",
                "Вы набрали не менее 10 и менее 40 баллов! Нормально!",
                "Вы набрали не менее 40 баллов! Отлично"
            };
        }

        // Метод для возвращения сообщения
        public static string GetMessage(int Ball)
        {
            string mes = "";

            if (Ball < 10)
                mes = Messages[0];
            else if (Ball < 40)
                mes = Messages[1];
            else
                mes = Messages[2];

            return mes;
        }
    }
}
```

Рисунок 13.9 – Класс проекта Result

Создан один статический файл, для представления результирующих сообщений.

8. Теперь модифицируем класс Tasting проекта Quiz. В своей работе класс будет использовать ранее созданные классы Task и Result, которые находятся в других проектах. Поэтому необходимо установит ссылки на данные проекты. Для этого вызовите команду (рис. 13.10а) контекстного меню папки References проекта Quiz. После этого ссылки на сборки (откомпилированные проекты Task и Result) появятся в списке ссылок Reference (рис. 13.10б).



а *б*

Рисунок 13.10 – Добавление ссылки на проекты: а – команда контекстного меню; б – список ссылок на проекты

9. Модифицируем файл Testing.cs для реализации логики тестирования (рис. 13.11).

```
class Testing
{
    public string DoTesting()
    {
        int SumBal = 0;
        int ans = 0;

        foreach (Task.SingleTest p in Task.Task.AllQuestions)
        {
            Console.WriteLine("////////////////////////////");
            Console.WriteLine(p.Question);
            Console.WriteLine("////////////////////////////");
            Console.WriteLine("1. "+p.Answers[0]);
            Console.WriteLine("2. " + p.Answers[1]);
            Console.WriteLine("3. " + p.Answers[2]);
            Console.Write("Выберите номер ответа > ");
            ans = Convert.ToInt32(Console.ReadLine());
            SumBal += p.Balls[ans-1];
        }

        return Result.Result.GetMessage(SumBal);
    }
}
```

Рисунок 13.11 – Класс Testing

10. Модифицируем функцию Main:

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

namespace Quiz
{
    class Program
    {
        static void Main(string[] args)
        {
            Testing newTest = new Testing();

            Console.WriteLine("\n\n"+newTest.DoTesting());

            Console.ReadKey();
        }
    }
}
```

Рисунок 13.12 – Функция Main

Запустите полученное приложение. Внимательно проанализируйте код приложения. Попробуйте изменить вопросы и их количество.

Затем, выполните индивидуальное задание.

Варианты индивидуальных заданий

Спроектируйте многомодульное приложение (**3 модуля**), реализующее поставленную задачу. Перед выполнением приложения необходимо выполнить декомпозицию задачи и выявить основные объекты предметной области.

Варианты	Структура данных
1, 6, 11, 16, 21	Приложение по двум числам a и b (стороны прямоугольника) рассчитывает периметр, диагональ и площадь прямоугольника.
2, 7, 12, 17, 22	Приложение по трем сторонам (a, b, c) треугольника рассчитывает периметр и площадь треугольника.
3, 8, 13, 18, 23	Приложение по значениям R и h (радиус и высота цилиндра) рассчитывает площадь поверхности и объем цилиндра.

Варианты	Структура данных
4, 9, 14, 19, 24	Приложение из нескольких прямоугольников со сторонами a и b выбирает фигуру наибольшей площади и фигуру, имеющую наибольшую диагональ.
5, 10, 15, 20, 25	Приложение из нескольких треугольников со сторонами a, b и c выбирает фигуру наименьшей и наибольшей площади.

13.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

13.8. Вопросы для защиты работы

1. Что такое статический класс?

2. Что такое сборка?

3. Как определить проект по умолчанию в многомодульном решении?

4. Какими способами можно разместить в файлах определение класса?

5. Какой проект начнет выполняться первым если несколько из них в одном решении содержат функцию Main?

6. Что необходимо сделать для того, чтобы появилась возможность использовать классы, определенные в другом проекте?

13.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5], [6-8].

ЛАБОРАТОРНАЯ РАБОТА 14. ОСНОВЫ ФАЙЛОВОГО ВВОДА-ВЫВОДА

14.1. Цель и содержание

Цель лабораторной работы: научиться использовать механизмы файлового ввода-вывода.

Задачи лабораторной работы:

– научиться применять классы для работы с файлами;

– научиться применять классы для работы с каталогами;

– научиться использовать потоки ввода-вывода.

14.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

14.3. Теоретическая часть

14.3.1 Классы .NET Framework для реализации операций ввода-вывода

Весь ввод и вывод в .NET Framework подразумевает использование потоков. Поток – это абстрактное представление последовательного устройства. Последовательное устройство – это нечто такое, что хранит данные в линейной структуре и точно таким же образом обеспечивает доступ к ним: считывает или записывает по одному байту за одну единицу времени. Сохранение устройства абстрактным означает, что лежащие в основе источник/приемник данных могут быть скрыты. Такой уровень абстракции обеспечивает повторное использование кода и позволяет писать более обобщенные процедуры, потому что нет необходимости заботиться о действительной специфике передачи данных.

Для обработки файлов в C# необходима ссылка на пространство имен System.IO. При открытии файла создается объект, с которым ассоциируется поток. Потоки обеспечивают механизмы связи между файлами и программами.

Среда .NET Framework предоставляет все необходимые инструменты для эффективного использования файлов в приложениях.

Основные классы, необходимые программисту:

1. Object – исходный базовый класс для всех классов платформы .NET Framework и корень иерархии типов.

2. File – статический служебный класс, предоставляющий множество статических методов, которые дают возможность перемещать, создавать, копировать, удалять файлы, опрашивать и обновлять атрибуты, а также создавать объекты потоков FileStream.

3. Directory – статический служебный класс, предоставляющий множество статических методов для перемещения, копирования и удаления каталогов.

4. Path – служебный класс, используемый для манипулирования путевыми именами.

5. MarshalByRefObject – разрешает доступ к объектам через границы доменов приложения в приложениях, поддерживающих удаленное взаимодействие, это базовый класс для всех классов .NET, позволяющих удаленное взаимодействие.

6. FileInfo – представляет физический файл на диске, имеет методы для манипулирования этим файлом. Для любого объекта, который читает или пишет в этот файл, должен быть создан объект Stream. Все методы FileInfo доступны из объектной переменной, поэтому, если необходимо выполнить только одно действие, более эффективным может оказаться использование метода File, а не соответствующего экземпляра метода FileInfo. Для всех методов FileInfo требуется путь к файлу, с которым проводится операция. Все статические методы класса FileInfo выполняют проверку безопасности для всех методов. Если необходимо использовать объект неоднократно, рекомендуется использовать соответствующий метод экземпляра FileInfo, поскольку в этом случае проверка безопасности будет требоваться не всегда.

7. DirectoryInfo – представляет физический каталог на диске и предоставляет методы уровня экземпляра для манипулирования каталогом. Класс DirectoryInfo работает точно так же, как класс FileInfo. Это объект, представляющий отдельный каталог на машине. Подобно классу FileInfo, многие из вызовов методов дублируются между Directory и DirectoryInfo.

8. FileSystemInfo – служит базовым классом для FileInfo и DirectoryInfo, обеспечивая возможность работы с файлами и каталогами одновременно, используя полиморфизм.

9. Stream – предоставляет универсальное представление последовательности байтов. Класс Stream является абстрактным базовым классом всех потоков.

10. FileStream – представляет файл, который может быть записан, прочитан или то и другое.

11. TextReader – представляет средство чтения, позволяющее считывать последовательные наборы знаков. Этот класс является абстрактным базовым классом для StreamReader, который считывает символы из потоков.

12. TextWriter – представляет средство записи, позволяющее записывать последовательные наборы символов. Этот класс является абстрактным базовым классом для StreamWriter, который записывают символы в потоки.

13. StreamReader – читает символьные данные из потока и может быть создан с использованием класса FileStream в качестве базового.

14. StreamWriter – пишет символьные данные в поток и может быть создан с использованием класса FileStream в качестве базового.

15. Component – предоставляет базовую реализацию интерфейса IComponent и делает возможным совместное использование объектов разными приложениями.

16. FileSystemWatcher – используется для мониторинга файлов и каталогов и представляет события, которые приложение может перехватить, когда в этих объектах происходят какие-то изменения.

Таким образом, эта система классов включает в себя классы для работы с файлами (File, FileInfo), каталогами (Directory, DirectoryInfo) и потоками (FileStream, StreamReader, StreamWriter).

В большинстве случаев для разработки бизнес-приложений достаточно лишь четырех классов для манипулирования файловой системой. Эти классы расположены пространстве имен System.IO и предназначены для работы с файловой системой компьютера, то есть для создания, удаления переноса файлов и каталогов.

Первые два типа – Directory и Fi1e реализуют свои возможности с помощью статических методов, поэтому данные классы можно использовать без создания соответствующих объектов (экземпляров классов).

Следующие типы – DirectoryInfo и FileInfo обладают схожими функциональными возможностями с Directory и Fi1e, но порождены от класса FileSystemInfo, поэтому реализуются путем создания соответствующих экземпляров классов.

Класс FileSystemInfo предоставляет базовый функционал. Значительная часть членов FileSystemInfo предназначена для работы с общими характеристиками файла или каталога (метками времени, атрибутами и т. п.). Рассмотрим некоторые свойства FileSystemInfo (таблица 14.1).

Таблица 14.1 – Свойства класса FileSystemInfo

Свойство	Описание
Attributes	Позволяет получить или установить атрибуты для данного объекта файловой системы. Для этого свойства используются значения и перечисления FileAttributes
CreationTime	Позволяет получить или установить время создания объекта файловой системы
Exists	Может быть использовано для того, чтобы определить, существует ли данный объект файловой системы
Extension	Позволяет получить расширение для файла
FullName	Возвращает имя файла или каталога с указанием пути к нему в файловой системе
LastAccessTime	Позволяет получить или установить время последнего обращения к объекту файловой системы
LastWriteTime	Позволяет получить или установить время последнего внесения изменений в объект файловой системы
Name	Возвращает имя указанного файла. Это свойство доступно только для чтения. Для каталогов возвращает имя последнего каталога в иерархии, если это возможно. Если нет, возвращает полностью определенное имя

В FileSystemInfo предусмотрен набор методов. Например, метод Delete() – позволяет удалить объект файловой системы с жесткого диска, а Refresh() – обновить информацию об объекте файловой системы.

14.3.2 Классы для работы с каталогами файловой системы

Класс DirectoryInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, которые предназначены для создания, перемещения, удаления, получения информации о каталогах и подкаталогах в файловой системе. Наиболее важные члены класса содержатся в таблице 14.2.

Таблица 14.2 – Доступные члены класса DirectoryInfo

Член	Описание
Create() CreateSubDirectory()	Создают каталог (или подкаталог) по указанному пути в файловой системе
Delete()	Удаляет пустой каталог
GetDirectories()	Позволяет получить доступ к подкаталогам текущего каталога (в виде массива объектов DirectoryInfo)
GetFiles()	Позволяет получить доступ к файлам текущего каталога (в виде массива объектов FileInfo)
MoveTo()	Перемещает каталог и все его содержимое на новый адрес в файловой системе
Parent	Возвращает родительский каталог в иерархии файловой системы

Работа с типом DirectoryInfo начинается с того, что создается экземпляр класса (объект), при вызове конструктора в качестве параметра указывается путь к нужному каталогу. Если необходимо обратиться к текущему каталогу (то есть каталогу, в котором в настоящее время производится выполнение приложения), вместо параметра используется обозначение ".". Например:

```
// Создаем объект DirectoryInfo,
// которому будет обращаться к текущему каталогу
DirectoryInfo dir1 = new DirectoryInfo(".");

// Создаем объект DirectoryInfo,
// которому будет обращаться к каталогу d:\prim
DirectoryInfo dir2 = new DirectoryInfo(@"d:\prim");
```

Если создается объект DirectoryInfo, который связывается с несуществующим каталогом, то будет сгенерировано исключение System.IO.DirectoryNotFoundException.

Свойство Attributes класса DirectoryInfo позволяет получить информацию об атрибутах объекта файловой системы. Возможные значения данного свойства приведены в следующей таблице 14.3.

Таблица 14.3 – Значения свойства Attributes

Значение	Описание
Archive	Этот атрибут используется приложениями при проведении резервного копирования, а в некоторых случаях - удаления старых файлов
Compressed	Определяет, что файл является сжатым
Directory	Определяет, что объект файловой системы является каталогом
Encrypted	Определяет, что файл является зашифрованным
Hidden	Определяет, что файл является скрытым (такой файл не будет выводиться при обычном просмотре каталога)
Normal	Определяет, что файл находится в обычном состоянии и для него установлены любые другие атрибуты. Этот атрибут не может использоваться с другими атрибутами
Offline	Файл (расположенный на сервере) кэширован в хранилище off-line на клиентском компьютере. Возможно, что данные этого файла уже устарели
Readonly	Файл доступен только для чтения
System	Файл является системным (то есть файл является частью операционной системы или используется исключительно операционной системой)

Через класс DirectoryInfo программист может собрать информацию о дочерних подкаталогах. Например:

```
static void printDirect(DirectoryInfo dir)
{
    Console.WriteLine("***** " + dir.Name + " *****");
    Console.WriteLine("FullName: {0}", dir.FullName);
    Console.WriteLine("Name: {0}", dir.Name);
    Console.WriteLine("Parent: {0}", dir.Parent);
    Console.WriteLine("Creation: {0}", dir.CreationTime);
    Console.WriteLine("Attributes: {0}", dir.Attributes.ToString());
    Console.WriteLine("Root: {0}", dir.Root);
}

static void Main(string[] args)
{
    DirectoryInfo dir = new DirectoryInfo(@"d:\prim");
    printDirect(dir);
    DirectoryInfo[] subDirects = dir.GetDirectories();
    Console.WriteLine("Найдено {0} подкаталогов", subDirects.Length)
    foreach (DirectoryInfo d in subDirects)
    {
        printDirect(d);
    }
}
```

Метод CreateSubdirectory() позволяет создать в выбранном каталоге как единственный подкаталог, так и множество подкаталогов (в том числе, и вложенных друг в друга). Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim");

//создали подкаталог
dir.CreateSubdirectory("doc");

//создали вложенный подкаталог
dir.CreateSubdirectory(@"book\2008");
```

Метод MoveTo() позволяет переместить текущий каталог по заданному в качестве параметра адресу. При этом возможно произвести переименование каталога. Например:

```
DirectoryInfo dir = new DirectoryInfo(@"d:\prim\bmp");
dir.MoveTo(@"d:\prim\letter\Николаев");
```

В данном случае каталог «bmp» перемещается по адресу «d:\prim\letter\Николаев». Так как имя перемещаемого каталога не совпадает с крайним правым именем в адресе нового местоположения каталога, то производится переименование.

Работать с каталогами файловой системы компьютера можно и при помощи класса Directory, функциональные возможности которого во многом совпадают с возможностями DirectoryInfo. Следует учитывать, что члены данного класса

реализованы статически, поэтому для их использования нет необходимости создавать объект. Например:

```
// Создание подкаталога Новый
Directory.CreateDirectory(@"d:\prim\Новый");

// Перемещение каталога Новый в каталог 2012
Directory.Move(@"d:\prim\Новый",
               @"d:\prim\2012\Новый");

//переименовали каталог Новый в Старый
Directory.Move(@"d:\prim\2012\Новый",
               @"d:\prim\2012\Старый");
```

Следует учитывать, что удаление каталога возможно только когда он пуст. На практике комбинируют использование классов Directory и DirectoryInfo.

14.3.3 Классы для работы с файлами

Класс FileInfo предназначен для организации доступа к физическому файлу, который содержится на жестком диске компьютера. Он позволяет получать информацию об этом файле (например, о времени его создания, размере, атрибутах), а также производить различные операции, например, по созданию файла или его удалению. Класс FileInfo наследует члены класса FileSystemInfo и содержит дополнительный набор членов, который приведен в следующей таблице 14.4

Таблица 14.4 – Члены класса FileInfo

Член	Описание
AppendText()	Создает объект StreamWriter для добавления текста к файлу.
CopyTo()	Копирует уже существующий файл в новый файл.
Create()	Создает новый файл и возвращает объект FileStream для взаимодействия с этим файлом.
CreateText()	Создает объект StreamWriter для записи текстовых данных в новый файл.
Delete()	Удаляет файл, которому соответствует объект FileInfo.
Directory	Возвращает каталог, в котором расположен данный файл.
DirectoryName	Возвращает полный путь к данному файлу в файловой системе.
Length	Возвращает размер файла.
MoveTo()	Перемещает файл в указанное пользователем место (этот метод позволяет одновременно переименовать данный файл).
Name	Позволяет получить имя файла.

Член	Описание
Open()	Открывает файл с указанными пользователем правами доступа на чтение, запись или совместное использование с другими пользователями.
OpenRead()	Создает объект FileStream, доступный только для чтения.
OpenText()	Создает объект StreamReader (о нем также будет рассказано ниже), который позволяет считывать информацию из существующего текстового файла.
OpenWrite()	Создает объект FileStream, доступный для чтения и записи.

Большинство методов FileInfo возвращает объекты классов FileStream, StreamWriter, StreamReader и т. п., которые позволяют различным образом взаимодействовать с файлом, например, производить чтение или запись в него. Например:

```
// Создание объекта FileInfo
FileInfo f = new FileInfo("text.txt");

// Создание файла и потока
StreamWriter fOut =
    new StreamWriter(f.Create());

// Запись текста в файл
fOut.WriteLine("ОДИН ДВА ТРИ...");

// Закрытие файла
fOut.Close();

// Получение информации о файле
Console.WriteLine("Name: {0}", f.Name);
Console.WriteLine("File size: {0}",
    f.Length);
Console.WriteLine("Creation: {0}",
    f.CreationTime);
Console.WriteLine("Attributes: {0}",
    f.Attributes.ToString());
```

Доступ к физическим файлам можно получать и через статические методы класса File. Большинство методов объекта FileInfo представляют в этом смысле зеркальное отражение методов объекта File.

14.3.4 Потоки в системе ввода-вывода

Программы на языке C# выполняют операции ввода-вывода посредством потоков, которые построены на иерархии классов. Поток (stream) – это абстракция, которая генерирует и принимает данные. С помощью потока можно читать данные

из различных источников (клавиатура, файл, память) и записывать в различные источники (принтер, экран, файл, память).

Центральную часть потоковой системы C# занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов. Производными от класса Stream являются классы потоков:

1. FileStream – байтовый поток, разработанный для файлового ввода-вывода.

2. BufferedStream – заключает в оболочку байтовый поток и добавляет буферизацию, которая во многих случаях увеличивает производительность программы.

3. MemoryStream – байтовый поток, который использует память для хранения данных.

Программист может реализовать собственные потоковые классы. Однако для подавляющего большинства приложений достаточно встроенных потоков.

14.3.4.1 Байтовый поток

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. При этом в классе определено несколько конструкторов. Чаще всего используется конструктор, который открывает поток для чтения и (или) записи:

```
public FileStream(string path, FileMode mode);
```

Параметр path определяет имя файла, с которым будет связан поток ввода-вывода данных. Параметр mode определяет режим открытия файла, который может принимать одно из возможных значений, определенных перечислением FileMode:

– FileMode.Append – предназначен для добавления данных в конец файла;

– FileMode.Create – предназначен для создания нового файла, при этом если существует файл с таким же именем, то он будет предварительно удален;

– FileMode.CreateNew – предназначен для создания нового файла, при этом файл с таким же именем не должен существовать;

– FileMode.Open – предназначен для открытия существующего файла;

– FileMode.OpenOrCreate – если файл существует, то открывает его, в противном случае создает новый;

– FileMode.Truncate – открывает существующий файл, но усекает его длину до нуля.

Если попытка открыть файл оказалась неудачной, то генерируется одно из исключений:

– FileNotFoundException – файл невозможно открыть по причине его отсутствия;

– IOException – файл невозможно открыть из-за ошибки ввода-вывода;

– ArgumentNullException – имя файла представляет собой null-значение;

– ArgumentException – некорректен параметр mode;

– SecurityException – пользователь не обладает правами доступа;

– DirectoryNotFoundException – некорректно задан каталог.

Другая версия конструктора позволяет ограничить доступ только чтением или только записью:

```
public FileStream(string path, FileMode mode, FileAccess access);
```

Параметры path и mode имеют то же назначение, что и в предыдущей версии конструктора. Параметр access, определяет способ доступа к файлу и может принимать одно из значений, определенных перечислением FileAccess:

– FileAccess.Read – только чтение;

– FileAccess.Write – только запись;

– FileAccess.ReadWrite – и чтение, и запись.

После установления связи байтового потока с физическим файлом внутренний указатель потока устанавливается на начальный байт файла.

Для чтения очередного байта из потока, связанного с физическим файлом, используется метод ReadByte( ). После прочтения очередного байта внутренний указатель перемещается на следующий байт файла. Если достигнут конец файла, то метод ReadByte( ) возвращает значение -1.

Для побайтовой записи данных в поток используется метод WriteByte( ).

По завершении работы с файлом его необходимо закрыть. Для этого достаточно вызвать метод Close( ). При закрытии файла освобождаются системные ресурсы, ранее выделенные для этого файла, что дает возможность использовать их для работы с другими файлами.

```
static void Main(string[] args)
{
    try
    {
        // Создание потока для чтения из файла
        FileStream fileIn =
            new FileStream("ФайлСТекстом.txt",
                FileMode.Open,
                FileAccess.Read);
        // Создание потока для записи в файл
        FileStream fileOut =
            new FileStream("ФайлКопия.txt",
                FileMode.Create,
                FileAccess.Write);
        int i;

        // В цикле производится чтение одного файла
        // и одновременная запись содержимого
        // во второй файл
        while ((i = fileIn.ReadByte()) != -1)
        {
            // запись очередного байта в поток
            fileOut.WriteByte((byte)i);
        }

        // Закрыте файлов
        fileIn.Close();
        fileOut.Close();
    }
    catch (Exception EX)
    {
        Console.WriteLine(EX.Message);
    }
}
```

В данном примере создаются два потока fileIn (для чтения байтов из файла в поток) и fileOut (для записи байтов из потока в файл). Каждый из потоков ассоциируется со своим файлом. Затем в цикле производится побайтовое чтение файла «ФайлСТекстом.txt» до момента, когда функция ReadByte( ) вернет значение -1 (то есть достигнут конец файла). При этом на каждой итерации цикла считывается один байт и на этой же итерации этот байт записывается в файл «ФайлКопия.txt». После всех операция оба потока закрываются. Весь код,

осуществляющий работу с файлами заключен в конструкцию try … catch. Это позволяет повысить устойчивость программы к ошибкам.

14.3.4.2 Символьный поток

Чтобы создать символьный поток нужно поместить объект класса Stream (например FileStream) внутрь объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если он (поток) имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр path определяет имя открываемого файла.

Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamWriter fileOut =
    new StreamWriter(
        new FileStream(
            "ФайлНиколаева.txt",
            FileMode.Create,
            FileAccess.Write));
```

И еще один вариант конструктора StreamWriter:

```
public StreamWriter(string path, bool append);
```

Параметр path определяет имя открываемого файла, а параметр append может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```
// Добавляем символы в конец файла
StreamWriter fileOut_1 =
    new StreamWriter("МойФ.txt", true);
```

Объявив таким образом переменную fileOut_1 для записи данных в поток можно обратиться к методу WriteLine:

```
// Использование WriteLine
fileOut_1.WriteLine("Строка для записи");
```

В данном случае для записи используется метод, аналогичный статическому методу класса Console. Это действительно схожие механизмы ввода-вывода.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода. Создать экземпляр класса StreamReader можно следующим образом:

```
StreamReader fileIn =
    new StreamReader(
        new FileStream(
            "text.txt",
            FileMode.Open,
            FileAccess.Read));
```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр path определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =
    new StreamReader
        (@"c:\temp\Николаев.txt");
```

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn =
    new StreamReader(
        @"c:\temp\t.txt",
        Encoding.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Тип Encoding реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлСТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```

В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

14.3.4.3 Перенаправление стандартных потоков.

Тремя стандартными потоками, доступ к которым осуществляется через свойства Console.Out, Console.In и Console.Error, могут пользоваться все программы, работающие в пространстве имен System. Свойство Console.Out относится к стандартному выходному потоку. По умолчанию это консоль. Например, при вызове метода Console.WriteLine() информация автоматически передается в поток Console.Out. Свойство Console.In относится к стандартному входному потоку, источником которого по умолчанию является клавиатура. Например, при вводе данных с клавиатуры информация автоматически передается потоку Console.In, к которому можно обратиться с помощью метода Console.ReadLine(). Свойство Console.Error относится к ошибкам в стандартном потоке, источником которого также по умолчанию является консоль. Однако эти потоки могут быть перенаправлены на любое совместимое устройство ввода-вывода, например, на работу с физическими файлами.

Перенаправить стандартный поток можно с помощью методов SetIn(), SetOut() и SetError(), которые являются членами класса Console:

```
static void SetIn(TextReader input);
static void SetOut(TextWriter output);
static void SetError(TextWriter output);
```

Пример перенаправления потоков представлен в следующей программе, демонстрирующей, что стандартный поток вывода перенаправляется в один файл, а поток ввода – в другой:

```
static void Main(string[] args)
{
    int[,] MyArray;
    StreamReader file = new StreamReader("input.txt");
    // Перенаправление на file
    Console.SetIn(file);
    string line = Console.ReadLine();
    string[] mas = line.Split(' ');
    int n = int.Parse(mas[0]);
    int m = int.Parse(mas[1]);
    MyArray = new int[n, m];
    for (int i = 0; i < n; i++)
    {
        line = Console.ReadLine();
        mas = line.Split(' ');
        for (int j = 0; j < m; j++)
        {
            MyArray[i, j] = int.Parse(mas[j]);
        }
    }
    PrintArray("Исходный массив:", MyArray, n, m);
    file.Close();
}

static void PrintArray(string a, int[,] mas, int n, int m)
{
    // Перенаправление на file
    StreamWriter file=new StreamWriter("output.txt");
    Console.SetOut(file);
    Console.WriteLine(a);
    for (int i = 0; i < n; i++)
    {
        for (int j=0; j<m; j++) Console.Write("{0} ", mas[i,j]);
        Console.WriteLine();
    }
    file.Close();
}
```

14.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

14.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

14.6. Методика и порядок выполнения работы

Для выполнения лабораторной работы необходимо спроектировать многомодульное приложение, использующее файлы для ввода и вывода информации.

Например, требуется разработать приложение, которое позволяет:

– генерировать матрицу случайных чисел с сохранением ее в файл;

– считывать матрицу из файла;

– выводить матрицу на экран.

Для выполнения данного задания-примера достаточно в отдельном проекте реализовать класс. Этот проект следует скомпилировать в виде dll-файла. Затем, в основной программе просто необходимо использовать данную библиотеку.

Класс библиотеки назовем Matrix. В нем определим следующие функции:

```
class Matrix
{
    private float[,] matrix;
    int m, n;

    // Конструктор
    public Matrix()...

    // Генерация матрицы заданного размера
    public void GenerateMatrix(int M, int N)...

    // Сохранение сгенерированной матрицы в файл
    public void SaveMatrix(string pFileName)...

    // Загрузка сохраненной матрицы из файла
    public Boolean LoadMatrix(string pFileName)...

    // Вывод матрицы в консоль
    public void PrintMatrix()...
}
```

Рисунок 14.1 – Основные методы класса Matrix

Листинг метода GenegateMatrix:

```
// Генерация матрицы заданного размера
public void GenerateMatrix(int M, int N)
{
    m = M; n = N;

    Random r = new Random(DateTime.Now.Millisecond);

    matrix = new float[M, N];

    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            matrix[i, j] = (float)r.Next(1000) / 973f;
}
```

Рисунок 14.2 – Метод для создания матрицы заданного размера и заполнения ее случайными числами

Листинг метода SaveMatrix:

```
// Сохранение сгенерированной матрицы в файл
public void SaveMatrix(string pFileName)
{
    if (matrix.Length > 0)
    {
        if (File.Exists(pFileName))
            File.Delete(pFileName);

        FileInfo f = new FileInfo(pFileName);
        TextWriter tw = f.CreateText();

        tw.WriteLine(m.ToString());
        tw.WriteLine(n.ToString());

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                tw.WriteLine(i.ToString() + " " + j.ToString() + " " + matrix[i, j].ToString("E10"));

        tw.Close();
    }
}
```

Рисунок 14.3 – Метод для сохранения матрицы в файл

```
// Загрузка сохраненной матрицы из файла
public Boolean LoadMatrix(string pFileName)
{
    if (File.Exists(pFileName))
    {
        try
        {
            TextReader tr = File.OpenText(pFileName);
            m = Convert.ToInt32(tr.ReadLine());
            n = Convert.ToInt32(tr.ReadLine());

            matrix = new float[m, n];
            string line;
            string[] substring;

            for (int i = 0; i < m; i++)
                for (int j = 0; j < n; j++)
                {
                    line = tr.ReadLine();
                    substring = line.Split(new char[] {' '}, 3);
                    matrix[i, j] = Convert.ToSingle(substring[2]);
                }

            tr.Close();

            return true;

        }
        catch
        {
            return false;
        }
    }

    return false;
}
```

Рисунок 14.4 – Метод загрузки матрицы из файла

На рис. 14.4 представлен листинг метода LoadMatrix. Листинг метода PrintMatrix:

```
// Вывод матрицы в консоль
public void PrintMatrix()
{
    if (matrix.Length > 0)
    {
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
                Console.Write(matrix[i, j].ToString("E3") + "  ");

            Console.WriteLine();
        }

    }
}
```

Рисунок 14.5 – Метод для вывода матрицы

Основная программа для использования класса-матрицы представлена на рис. 14.6

```
class Program
{
    static void Main(string[] args)
    {
        Console.BackgroundColor = ConsoleColor.White;
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Clear();

        Matrix m = new Matrix();

        m.GenerateMatrix(10, 5);
        m.SaveMatrix("FileForMatrix.txt");

        if (m.LoadMatrix("FileForMatrix.txt"))
            m.PrintMatrix();

        Console.ReadKey();
    }
}
```

Рисунок 14.6 – Работа с библиотечным классом

В результате работы данного кода будет создан файл FileForMatrix.txt следующего содержания:

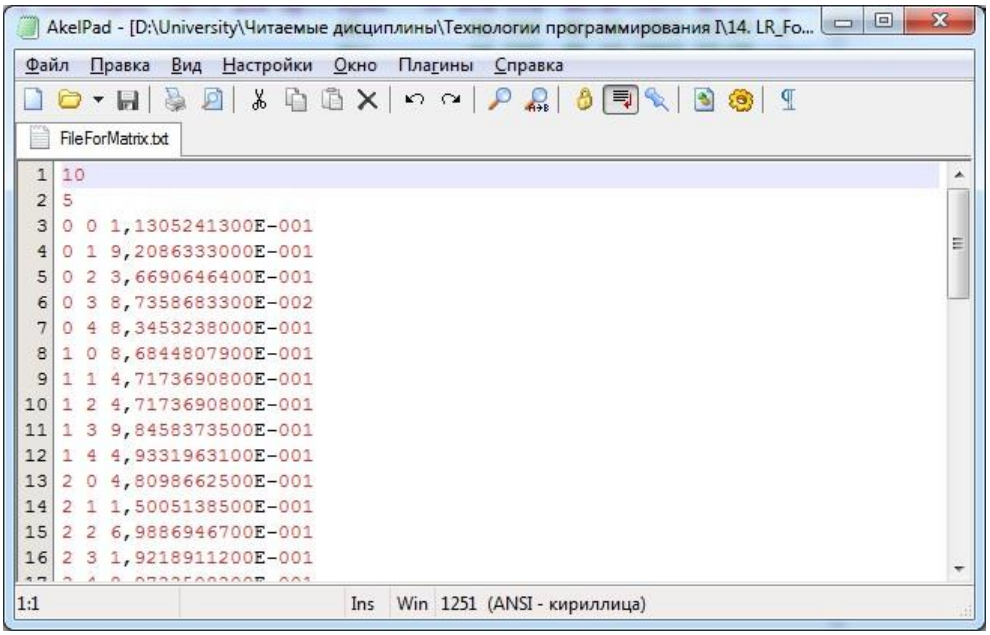


Рисунок 14.7 – Результирующий файл

На экран будет выведена следующая информация:

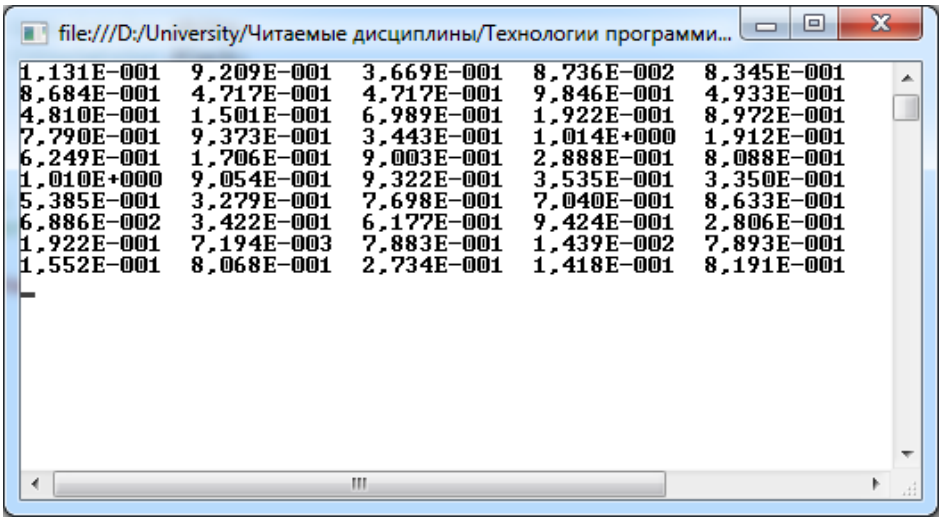


Рисунок 14.8 – Вывод программы в консоль

Выполните индивидуальное задание, согласно предложенному варианту. В каждом варианте необходимо спроектировать многомодульное приложение (оптимально 2 модуля). Исходные данные в каждом варианте программа получает из входного файла.

Варианты индивидуальных заданий

Вариант	Структура данных
1	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
2	Программа рассчитывает сумму двух матриц, которые хранятся в разных файлах.
3	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.
4	Программа рассчитывает сумму диагональных элементов двух матриц, которые хранятся в разных файлах.
5	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
6	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.
7	Программа рассчитывает сумму элементов главной диагонали первой матрицы и сумму элементов второстепенной диагонали второй матрицы. Матрицы хранятся в разных файлах.
8	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
9	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
10	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
11	Программа рассчитывает сумму элементов , находящихся в четном столбце и нечетной строке, в двух матрицах, которые хранятся в разных файлах.
12	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
13	Программа рассчитывает произведение элементов в двух матрицах, которые хранятся в разных файлах.
14	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
15	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
16	Программа рассчитывает сумму элементов четных строк в двух матрицах, которые хранятся в разных файлах.
17	Программа находит максимальный элемент в двух матрицах, которые хранятся в разных файлах.

Вариант	Структура данных
18	Программа рассчитывает сумму элементов четных строк и расположенных на второстепенной диагонали в обеих матрицах, которые хранятся в разных файлах.
19	Программа рассчитывает сумму первой матрицы и матрицы транспонированной относительно второй. Обе матрицы хранятся в отдельных файлах.
20	Программа рассчитывает произведение двух матриц, которые хранятся в разных файлах.
21	Программа рассчитывает сумму элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
22	Программа рассчитывает сумму диагональных элементов четных столбцов в двух матрицах, которые хранятся в разных файлах.
23	Программа рассчитывает произведение элементов диагонали первой матрицы и сумму всех элементов второй матрицы. Обе матрицы хранятся в отдельных файлах.
24	Программа рассчитывает сумму элементов с четной суммой индексов в двух матрицах, которые хранятся в разных файлах.
25	Программа рассчитывает разность сумм элементов матриц, которые хранятся в разных файлах.

14.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

14.8. Контрольные вопросы

1. Какие классы для работы с файловой системой вы знаете?

2. Что такое сборка?

3. Как определить проект по умолчанию в многомодульном решении?

4. Какие классы отвечают за представление файлов в программе?

5. Что такое поток? Какие типы классов потоков используются при работе с файлами?

6. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов.

14.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5], [6-8].

ЛАБОРАТОРНАЯ РАБОТА 15. ВЫЧИСЛИТЕЛЬНАЯ ЗАДАЧА С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВОГО ВВОДА-ВЫВОДА

15.1. Цель и содержание

Цель лабораторной работы: научиться использовать механизмы файлового ввода-вывода при работе с данными.

Задачи лабораторной работы:

– научиться применять классы для работы с файлами;

– научиться использовать возможности ввода-вывода для решения практических задач;

– научиться обрабатывать данные в приложениях с файловым вводом-выводом.

15.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

15.3. Теоретическая часть

Повторите теоретический материал лабораторной работы №14. При выполнении данной лабораторной работы рекомендуется использовать символьный поток.

Чтобы создать символьный поток нужно поместить объект класса Stream (например FileStream) внутрь объекта класса StreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока. В нем определено несколько конструкторов. Один из них записывается следующим образом:

```
public StreamWriter(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока. Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для вывода, и исключение типа ArgumentNullException, если он (поток) имеет null-значение.

Другой вид конструктора позволяет открыть поток сразу через обращения к файлу:

```
public StreamWriter(string path);
```

Параметр path определяет имя открываемого файла.

Например, создать экземпляр класса StreamWriter можно следующим образом:

```
StreamWriter fileOut =
    new StreamWriter(
        new FileStream(
            "ФайлНиколаева.txt",
            FileMode.Create,
            FileAccess.Write));
```

И еще один вариант конструктора StreamWriter:

```
public StreamWriter(string path, bool append);
```

Параметр path определяет имя открываемого файла, а параметр append может принимать значение true – если нужно добавлять данные в конец файла, или false – если файл необходимо перезаписать.

Например:

```
// Добавляем символы в конец файла
StreamWriter fileOut_1 =
    new StreamWriter("МойФ.txt", true);
```

Объявив таким образом переменную fileOut_1 для записи данных в поток можно обратиться к методу WriteLine:

```
// Использование WriteLine
fileOut_1.WriteLine("Строка для записи");
```

В данном случае для записи используется метод, аналогичный статическому методу класса Console. Это действительно схожие механизмы ввода-вывода.

Класс StreamReader предназначен для организации входного символьного потока. Один из его конструкторов выглядит следующим образом:

```
public StreamReader(Stream stream);
```

Параметр stream определяет имя уже открытого байтового потока.

Этот конструктор генерирует исключение типа ArgumentException, если поток stream не открыт для ввода. Создать экземпляр класса StreamReader можно следующим образом:

```
StreamReader fileIn =
    new StreamReader(
        new FileStream(
            "text.txt",
            FileMode.Open,
            FileAccess.Read));
```

Как и в случае с классом StreamWriter у класса StreamReader есть и другой вид конструктора, который позволяет открыть файл напрямую:

```
public StreamReader(string path);
```

Параметр path определяет имя открываемого файла. Обратиться к данному конструктору можно следующим образом:

```
StreamReader fileIn =
    new StreamReader
        (@"c:\temp\Николаев.txt");
```

В C# символы реализуются кодировкой Unicode. Для того, чтобы можно было обрабатывать текстовые файлы, содержащие русские символы, созданные, например, в Блокноте, рекомендуется вызывать следующий вид конструктора StreamReader:

```
StreamReader fileIn =
    new StreamReader(
        @"c:\temp\t.txt",
        Encoding.GetEncoding(1251));
```

Параметр Encoding.GetEncoding(1251) говорит о том, что будет выполняться преобразование из кода Windows-1251 (одна из модификаций кода ASCII, содержащая русские символы) в Unicode. Тип Encoding реализован в пространстве имен System.Text.

Для чтения данных из потока fileIn можно воспользоваться методом ReadLine. При этом если будет достигнут конец файла, то метод ReadLine вернет значение null.

Рассмотрим пример, в котором данные из одного файла копируются в другой, но уже с использованием классов StreamWriter и StreamReader.

```
static void Main(string[] args)
{
    // Создание символьных потоков
    StreamReader fileIn =
        new StreamReader(
            "ФайлСТекстом.txt",
            Encoding.GetEncoding(1251));
    StreamWriter fileOut =
        new StreamWriter(
            "НовыйФайл.txt",
            false);
    string line;

    // Построчное считывание
    while ((line = fileIn.ReadLine()) != null)
    {
        // ... и запись строки
        fileOut.WriteLine(line);
    }

    fileIn.Close();
    fileOut.Close();
}
```

В данном примере осуществляется копирование содержимого одного символьного файла в другой.

Таким образом, данный способ копирования одного файла в другой, дает тот же результат, что и при использовании байтовых потоков. Однако, его работа будет менее эффективной, т.к. будет тратиться дополнительное время на преобразование байтов в символы. У символьных потоков есть и свои преимущества. Например, можно использовать регулярные выражения для поиска заданных фрагментов текста в файле.

15.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство

DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, Microsoft Visual Studio 2013 и выше.

15.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы определяется общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

15.6. Методика и порядок выполнения работы

Перед выполнением индивидуального задания выполните учебную задачу.

Учебная задача. Разработать приложение, отражающее процессы поступления товара в магазине. Пользователь приложения формирует редактируемый список товарных групп («Продукты», «Бытовая химия», «Одежда», «Фрукты», «Полиграфия»). По факту поступления товара пользователь формирует запись со следующими полями: «Наименование товара», «Партия», «Товарная группа», «Цена» «Количество». Описание полей представлены в таблице 15.1. В программе должен быть реализован автоматический расчет сумм по каждой партии и товарной группе.

Таблица 15.1 – Поля записи регистра поступления товаров

Поле записи	Описание	Тип данных
Наименование товара	Пользователь заполняет самостоятельно.	String

Партия	Документ (приходная накладная) в соответствии с которым принят товар. Пользователь заполняет самостоятельно.	String
Товарная группа	Пользователь может выбрать только из имеющегося справочника товарных групп.	String
Цена	Пользователь вводит самостоятельно.	float
Количество	Пользователь вводит самостоятельно.	float

Для выполнения лабораторной работы необходимо спроектировать многомодульное приложение, использующее файлы для ввода и вывода информации.

На рис. 15.1 представлен фрагмент текстового файла, содержащий исходные данные.

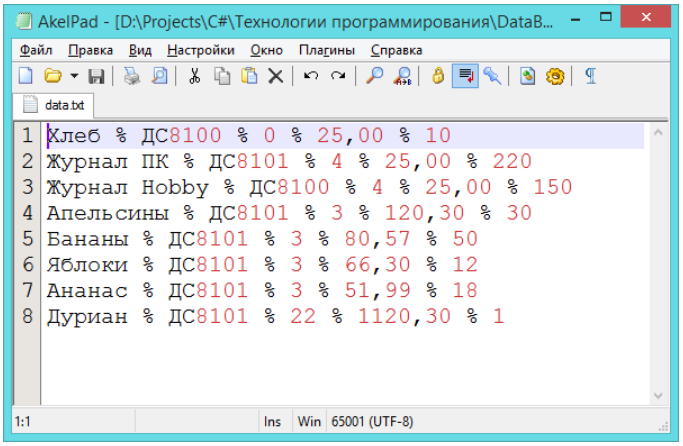


Рисунок 15.1 – Фрагмент исходного файла

Решение. Решение задачи будем реализовывать на базе приложения Windows Forms.

1. Создайте новый проект в среде MS VS типа Windows Forms с именем DataBase. В приложении модифицируйте экранную форму в соответствии с рисунком 15.2.

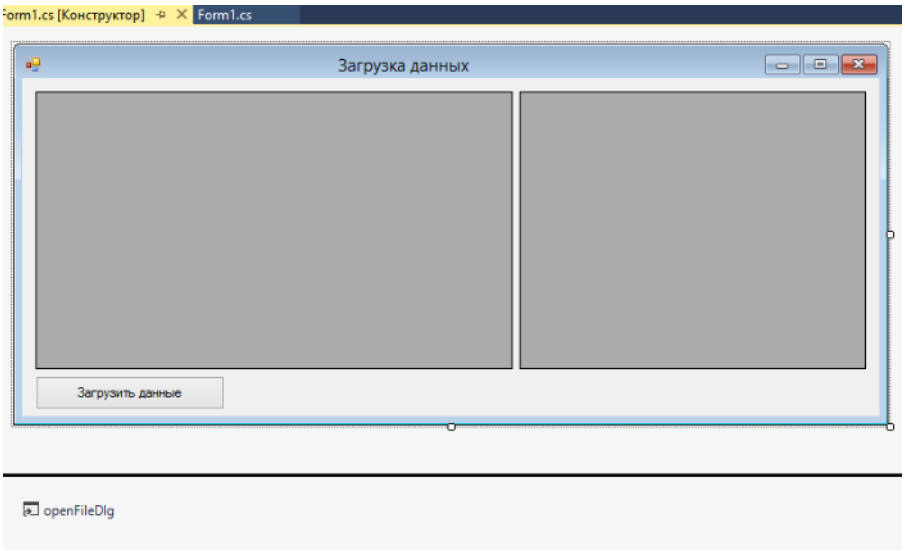


Рисунок 15.2 – Внешний вид экранной формы Form1

В таблице 15.2 представлено описание компонентов экранной формы.

Таблица 15.2 – Компоненты экранной формы Form1

Свойство Name объекта	Тип объекта	Описание
dgvRaw	DataGridView	Таблица для отображения данных из текстового файла. Данные отображаются в том виде, в котором они хранятся в текстовом файле – без дополнительных преобразований
dgvSummary	DataGridView	Таблица для отображения агрегированных данных: суммарная стоимость по каждой товарной категории
btnLoad	Button	Кнопка с текстом «Загрузить данные». Выполняет открытие диалогового окна для выбора файла с данными.
openFileDlg	OpenFileDialog	Невизуальный компонент. Представляет собой стандартное диалоговое окно для выбора файла

2. Выполним декомпозицию решаемой задачи и определим, какие классы понадобятся нам для выполнения поставленной задачи. В таблице 15.3 представлены имена добавляемых классов и их описание.

Таблица 15.3 – Классы проекта

Имя класса	Описание
RawDataItem	Класс для представления одной строки в файле с данными. Этот же класс будет являться источником данных для строки таблицы dgvRaw
SummaryDataItem	Класс для представления одной строки агрегированных данных. Этот же класс будет предоставлять данные для одной строки таблицы dgvSummary
Utils	Вспомогательный класс для отображения номера категории в название товарной категории (в файле хранятся не имена товарных категорий, а их коды), класс Utils будет преобразовывать номера категорий в их наименования
DataInterface	Это не класс, а интерфейс, который будет предоставлять экранной форме необходимые данные
DataStorage	Класс, который будет реализовывать интерфейс DataInterface

3. Самый простой класс – RawDataItem. На рис. 15.3 представлен листинг кода объявления класса RawDataItem для представления данных одной строки исходного файла.

```
7   namespace DataBase
8   {
9       class RawDataItem
10      {
11          public String Name { get; set; }
12          public int Group { get; set; }
13          public String Part { get; set; }
14          public float Price { get; set; }
15          public float Count { get; set; }
16          public float Sum
17          {
18              get { return Count * Price;  }
19          }
20      }
21  }
```

Рисунок 15.3 – Исходный код класса RawDataItem

4. Следующий простой класс – SummaryDataItem, листинг которого представлен на рис. 15.4.

```
7   namespace DataBase
8   {
9       class SummaryDataItem
10      {
11          public String GroupName { get; set; }
12          public float GroupSum { get; set; }
13      }
14  }
```

Рисунок 15.4 – Исходный код класса SummaryDataItem

5. Реализуйте класс Utils (рис. 15.5).

```
9   class Utils
10  {
11      private static Dictionary<int, String> dict;
12      static Utils()
13      {
14          if(dict == null)
15          {
16              dict = new Dictionary<int, string>(5);
17              dict.Add(0, "Продукты");
18              dict.Add(1, "Бытовая химия");
19              dict.Add(2, "Одежда");
20              dict.Add(3, "Фрукты");
21              dict.Add(4, "Полиграфия");
22          }
23      }
24
25      public static String GetGroupByNumber(int number)
26      {
27          if (dict.ContainsKey(number))
28              return dict[number];
29          else
30              return "???";
31      }
32  }
```

Рисунок 15.5 – Исходный код класса Utils

Класс Utils содержит единственный статический метод GetGroupByNumber(int number),который возвращает наименование товарной категории по ее номеру. Пары «номер-наименование» хранятся в закрытом словаре, построение которого происходит в статическом конструкторе класса Utils.

6. Реализуем интерфейс DataInterface, который будет использоваться для получения данных в экранной форме (рис. 15.6).

```
7   namespace DataBase
8   {
9       interface DataInterface
10      {
11          List<RawDataItem> GetRawData();
12          List<SummaryDataItem> GetSummaryData();
13      }
14  }
```

Рисунок 15.6 – Исходный код интерфейсного типа DataInterface

Интерфейс содержит два метода:

1) GetRawData() – возвращает данные для таблицы dgvRaw;

2) GetSummaryData() – возвращает данные для таблицы dgvSummary.

7. Рассмотрим определение класса DataStorage (рис. 15.7).

```
10  class DataStorage : DataInterface
11  {
12      public bool IsReady...
20      private List<RawDataItem> rawdata;
21      private List<SummaryDataItem> sumdata;
22      private char devider = '%';
23      private DataStorage(){ }
24
25      private bool InitData(String datapath)...
59
60      private void BuildSummary()...
81
82      public static DataStorage DataCreator(String path)...
90
91      public List<RawDataItem> GetRawData()...
98
99      public List<SummaryDataItem> GetSummaryData()...
106 }
```

Рисунок 15.7 – Исходный код класса DataStorage

Описание полей и методов класса DataStorage представлено в таблице 15.4.

Таблица 15.4 – Поля и методы класса DataStorage

Имя поля/метода	Тип / тип возврата	Описание
rawData	List<RawDataItem>	Закрытое свойство для хранения списка элементов типа RawDataItem
sumData	List<SummaryDataItem>	Закрытое свойство для хранения списка элементов типа SummaryDataItem
devider	char	Поле для хранения символа-разделителя, которым отделяются поля в исходном файле
IsReady	bool	Возвращает true, если поле rawdata заполнено, в противном случае – false
DataStorage( )		Закрытый конструктор (пустой)
InitData(String datapath)	bool	Метод считывает данные из файла datapath и заполняет коллекцию rawData
BuildSummary()	void	Метод для расчета агрегирующих показателей по отдельным категориям: создает коллекцию sumData
DataCreator(String path)	DataStorage	Статический метод для возврата нового объекта типа DataStorage
GetRawData( )	List<RawDataItem>	Метод является реализацией интерфейса DataInterface. Возвращает данные для таблицы dgvRaw
GetSummaryData( )	List<SummaryDataItem>	Метод является реализацией интерфейса DataInterface. Возвращает данные для таблицы dgvSummary

Листинг класса DataStorage представлен на рис. 15.8.

```
10      class DataStorage : DataInterface
11      {
12          public bool IsReady
13          {
14              get
15              {
16                  if (rawdata == null) return false;
17                  else return true;
18              }
19          }

21          private List<RawDataItem> rawdata;
22          private List<SummaryDataItem> sumdata;
23          private char devider = '%';
24          private DataStorage(){ }
```

```
60          private void BuildSummary()
61          {
62              Dictionary<int, float> tmp = new Dictionary<int, float>();
63              foreach (var item in rawdata)
64              {
65                  if (tmp.ContainsKey(item.Group))
66                      tmp[item.Group] += item.Sum;
67                  else
68                      tmp[item.Group] = item.Sum;
69              }
70
71              sumdata = new List<SummaryDataItem>();
72              foreach (var item in tmp)
73              {
74                  sumdata.Add(new SummaryDataItem()
75                  {
76                      GroupName = Utils.GetGroupByNumber(item.Key),
77                      GroupSum = item.Value
78                  });
79              }
80          }
```

```
26          private bool InitData(String datapath)
27          {
28              rawdata = new List<RawDataItem>();
29
30              try
31              {
32                  StreamReader sr = new StreamReader(datapath, Encoding.UTF8);
33                  String line;
34                  while ((line = sr.ReadLine()) != null)
35                  {
36                      string[] items = line.Split(devider);
37                      var item = new RawDataItem()
38                      {
39                          Name = items[0].Trim(),
40                          Part = items[1].Trim(),
41                          Group = Convert.ToInt32(items[2].Trim()),
42                          Price = Convert.ToSingle(items[3].Trim()),
43                          Count = Convert.ToSingle(items[4].Trim())
44                      };
45                      rawdata.Add(item);
46
47                  }
48
49                  sr.Close();
50                  BuildSummary();
51
52              } catch(IOException ex)
53              {
54                  return false;
55              }
56
57              return true;
58          }
```

```
82          public static DataStorage DataCreator(String path)
83          {
84              DataStorage d = new DataStorage();
85              if (d.InitData(path))
86                  return d;
87              else
88                  return null;
89          }
90
91          public List<RawDataItem> GetRawData()
92          {
93              if (this.IsReady)
94                  return rawdata;
95              else
96                  return null;
97          }
98
99          public List<SummaryDataItem> GetSummaryData()
100         {
101             if (this.IsReady)
102                 return sumdata;
103             else
104                 return null;
105         }
```

Рисунок 15.8 – Листинг класса DataStorage

8. В классе Form1 добавьте обработчик события btnLoad_Click( ) и метод ShowData( ). Листинги методов представлены на рисунках 15.9 и 15.10.

```
22          private void btnLoad_Click(object sender, EventArgs e)
23          {
24              openFileDlg.InitialDirectory = Application.StartupPath;
25              if(openFileDlg.ShowDialog() == DialogResult.OK)
26              {
27                  ShowData(openFileDlg.FileName);
28              }
29          }
```

Рисунок 15.9 – Обработчик нажатия кнопки btnLoad

```
31          private void ShowData(String datapath)
32          {
33              try
34              {
35                  data = DataStorage.DataCreator(datapath);
36              }
37              catch (Exception ex)
38              {
39                  MessageBox.Show("При загрузке данных что-то сломалось");
40              }
41
42              dgvRaw.DataSource = data.GetRawData();
43              dgvRaw.ReadOnly = true;
44              dgvSummary.DataSource = data.GetSummaryData();
45              dgvSummary.ReadOnly = true;
46
47          }
```

Рисунок 15.10 – Метод ShowData класса Form1

9. После реализации всех классов, запустите приложение. Приложение выглядит как указано на рис. 15.11.

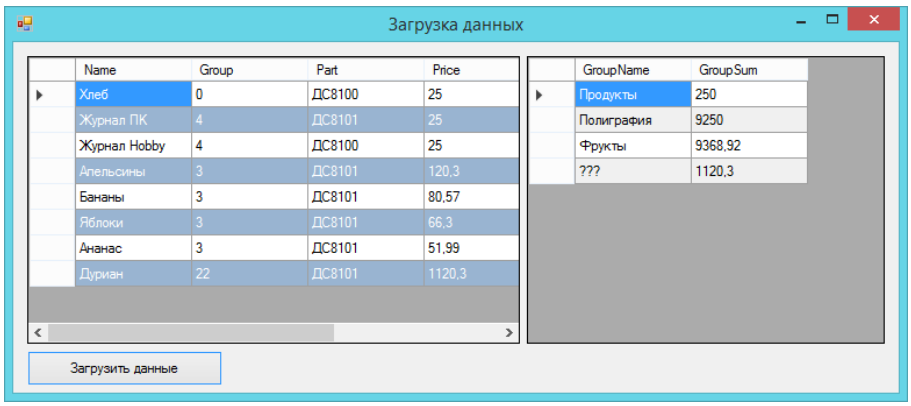


Рисунок 15.11 – Внешний вид окна приложения в режиме выполнения

Варианты индивидуальных заданий

Вариант	Задание
1	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк): 1 Иванов С.Е.| доцент | ИСТ | 56.00 2 Петров А.А. | профессор | ИБАС | 100.23 3 Новиков П.С. | ст. преподаватель | ИСТ | 35.00 4 Ганотченко Е.П. | профессор | ИБАС | 200.45 5 Петров А.А. | профессор | ИБАС | 105.67 Необходимо вычислить: количество преподавателей; сумму зарплат преподавателей по каждой кафедре.

2	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк): 1 BAS-2107; Автоваз; 140000.57; 16000 2 Гога-1; Хуань Вей Трейд; 50.00; 1000 3 BAS-5000; Автоваз; 1400000.00; 1000 4 X35; Hyundai; 750000; 1 Необходимо вычислить: суммарную стоимость всех автомобилей; автомобиль с наименьшим пробегом.
3	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк): 1 Самолет*Москва*Пекин*30000.00 2 Автобус*Ставрополь*Москва*3000.00 3 Автобус*Невинномысск*Ставрополь*10.00 4 Автобус*Москва*Владивосток*50000 5 Самолет*СПб*Москва*1500 Необходимо вычислить: количество автобусных рейсов; суммарную стоимость билетов рейсов самолетов; самый дорогой билет.
4	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк): 1 Иванов Е.А.*ИСТ-б-о-121*3*2 2 Петров А.А.*ИСТ-б-о-111*4*1 3 Кононов А.Е.*БАС-б-о-111*4*0 4 Катченко Е.А.*БАС-б-о-121*3*1 Необходимо вычислить: суммарное количество задолженностей по каждому курсу; количество студентов без задолженностей.
5	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк): 1 Молоко, 0.5 л % Продукты % 10.00 % 23.00 2 Хлеб белый % Продукты % 15.00 % 22.00 3 Молоко, 1.0 л % Продукты % 18.00 % 30.00 4 Туалетный утенок % Химия % 11.00 % 17.58 5 Фейри, 0.5 л % Химия % 17.00 % 88.00 Необходимо вычислить: суммарную стоимость товаров по каждой группе; самый дорогой товар в каждой товарной группе.
6	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк): 1 Кабель сетевой % Сетевое оборудование % 10.00 % Пулково 1 2 Коммутатор ASUS X100 % Сетевое оборудование % 15.00 % Склад №7 3 ПО Windows 8.1, 1.0 л % ПО % 18.00 % Склад №19 4 ПО Office 365 % ПО % 11.00 % Склад №19 5 Коннектор 234511 % Сетевое оборудование % 17.00 % Склад №7 Необходимо вычислить: среднюю стоимость товаров по каждой группе; склад, содержащий максимальное количество продукции.

7	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк):
	`Дом*6*150*3500000`
	`Квартира*1*49*790000`
	`Квартира*2*59*1500000`
	`Квартира*3*70*3000000`
	`Комната*1*12*500000`
	Необходимо вычислить: для каждого строения рассчитать стоимость квадратного метра; суммарную стоимость объектов по каждому типу.

8	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк):
	`Привет с того света | Детектив | 150.00`
	`Кто подставил кролика | Детектив | 200.10`
	`Почемучка | Детская литература | 60.22`
	`Капитошка | Детская литература | 230.14`
	`Звезда 112 | Фантастика | 12.90`
	Необходимо вычислить: самую дешевую книгу в каждом жанре; среднюю стоимость книг в каждом жанре.

9	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк):
	`Молоко \ Продукты \ 40.00 \ 1`
	`Тряпка \ Хозтовары \ 10.00 \ 5`
	`Порошок \ Хозтовары \ 750.00 \ 10`
	`Мясо \ Продукты \ 350 \ 5`
	`Швабра \ Хозтовары \ 900.00 \ 5`
	Необходимо вычислить: цену каждого товара за вычетом скидки; суммарную стоимость товаров по каждой группе.

10	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк):
	`Рис 1 кг. / 120.11 / 54 / Магнит`
	`Ручка шариковая / 23.00 / 15 / ТЦ Карандаш`
	`Тетрадь 12 л. / 15.00 / 200 / ГеоКнига`
	`Тетрадь 60 л. / 90.70 / 300 / ГеоКнига`
	`Торт 1 кг. / 450 / 4 / Магнит`
	Необходимо вычислить: среднюю цену товара по каждому магазину; суммарную стоимость для каждого товара.

11	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк):
	`Иванов С.Е.| доцент | ИСТ | 56.00`
	`Петров А.А. | профессор | ИБАС | 100.23`
	`Новиков П.С. | ст. преподаватель | ИСТ | 35.00`
	`Ганотченко Е.П. | профессор | ИБАС | 200.45`
	`Петров А.А. | профессор | ИБАС | 105.67`
	Необходимо вычислить: суммарный фонд заработной платы; среднюю зарплату по каждой кафедре.

12	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк):
	`ВАЗ-2107; Автоваз; 140000.57; 16000`
	`Гога-1; Хуань Вей Трейд; 50.00; 1000`
	`ВАЗ-5000; Автоваз; 1400000.00; 1000`
	`X35; Hyundai; 750000; 1`
	Необходимо вычислить: средний пробег по каждому производителю; среднюю стоимость автомобилей по каждому производителю.

13	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк):
	`Самолет*Москва*Пекин*30000.00`
	`Автобус*Ставрополь*Москва*3000.00`
	`Автобус*Невинномысск*Ставрополь*10.00`
	`Автобус*Москва*Владивосток*50000`
	`Самолет*СПб*Москва*1500`
	Необходимо вычислить: среднюю стоимость билета по каждому виду транспорта; количество рейсов из Москвы.

14	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк):
	`Иванов Е.А.*ИСТ-б-о-121*3*2`
	`Петров А.А.*ИСТ-б-о-111*4*1`
	`Кононов А.Е.*БАС-б-о-111*4*0`
	`Катченко Е.А.*БАС-б-о-121*3*1`
	Необходимо вычислить: суммарное количество задолженностей по каждой группе; среднее количество задолженностей по каждому курсу.

15	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк):
	`Молоко, 0.5 л % Продукты % 10.00 % 23.00`
	`Хлеб белый % Продукты % 15.00 % 22.00`
	`Молоко, 1.0 л % Продукты % 18.00 % 30.00`
	`Туалетный утенок % Химия % 11.00 % 17.58`
	`Фейри, 0.5 л % Химия % 17.00 % 88.00`
	Необходимо вычислить: суммарную прибыль от продажи товаров; среднюю цену закупки по каждой группе.

16	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк):
	`Кабель сетевой % Сетевое оборудование % 10.00 % Пулково 1`
	`Коммутатор ASUS X100 % Сетевое оборудование % 15.00 % Склад №7`
	`ПО Windows 8.1, 1.0 л % ПО % 18.00 % Склад №19`
	`ПО Office 365 % ПО % 11.00 % Склад №19`
	`Коннектор 234511 % Сетевое оборудование % 17.00 % Склад №7`
	Необходимо вычислить: среднюю стоимость товаров по каждому складу; суммарную стоимость по каждой группе.

17	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк):
	`Дом*6*150*3500000`
	`Квартира*1*49*790000`
	`Квартира*2*59*1500000`
	`Квартира*3*70*3000000`
	`Комната*1*12*500000`
	Необходимо вычислить: вывести объект с максимальной стоимостью квадратного метра; среднюю стоимость квадратного метра по объектам типа «Квартира».

18	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк):
	`Привет с того света | Детектив | 150.00`
	`Кто подставил кролика | Детектив | 200.10`
	`Почемучка | Детская литература | 60.22`
	`Капитошка | Детская литература | 230.14`
	`Звезда 112 | Фантастика | 12.90`
	Необходимо вычислить: суммарную стоимость книг по каждому жанру; среднюю стоимость книг по жанру «Фантастика».

19	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк):
	`Молоко \ Продукты \ 40.00 \ 1`
	`Тряпка \ Хозтовары \ 10.00 \ 5`
	`Порошок \ Хозтовары \ 750.00 \ 10`
	`Мясо \ Продукты \ 350 \ 5`
	`Швабра \ Хозтовары \ 900.00 \ 5`
	Необходимо вычислить: среднюю цену товара (с учетом скидки) по каждой товарной группе; товар с минимальной скидкой.

20	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк):
	`Рис 1 кг. / 120.11 / 54 / Магнит`
	`Ручка шариковая / 23.00 / 15 / ТЦ Карандаш`
	`Тетрадь 12 л. / 15.00 / 200 / ГеоКнига`
	`Тетрадь 60 л. / 90.70 / 300 / ГеоКнига`
	`Торт 1 кг. / 450 / 4 / Магнит`
	Необходимо вычислить: суммарную стоимость товаров в каждом магазине; магазин с минимальным товарным ассортиментом.

15.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

15.8. Контрольные вопросы

1. Какие классы для работы с файловой системой вы знаете?

2. Что такое класс потока? Перечислите классы потоков для работы с файлами?

3. Для чего используются интерфейсные типы? Приведите примеры.

4. Какие классы отвечают за представление файлов в программе?

5. Опишите последовательность действий при необходимости записать одну строку в файл. Приведите примеры использования различных классов.

6. Опишите принципы работы с байтовым потоком. Приведите пример кода для записи и считывания файла с использованием байтового потока.

7. Чем байтовый поток отличается от символьного?

15.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следущие источники: [5], [6-8].

ЛАБОРАТОРНАЯ РАБОТА 16. НЕСВЯЗНЫЙ УРОВЕНЬ ADO.NET В ПРИЛОЖЕНИЯХ WINDOWS FORMS

16.1. Цель и содержание

Цель лабораторной работы: научиться использовать классы, представляющие возможности несвязного доступа в рамках ADO.NET.

Задачами лабораторной работы являются:

– научится настраивать объекты соединения;

– получить практические навыки работы с адаптерами данных;

– научиться использовать информацию из таблицы базы данных в элементах управления Windows;

– научиться создавать представления;

– получить практические навыки обработки данных из связанных таблиц.

16.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

16.3. Теоретическое обоснование

При работе в рамках несвязного уровня ADO.NET используются объекты соединения, команд и адаптеры данных. Данные из базы данных получают с помощью адаптера данных. Для перемещения данных между клиентским приложением и источником данных объекты адаптера используют объекты DataSet. Тип DataSet – это контейнер, используемый для любого числа объектов DataTable. Каждый объект DataTable содержит коллекцию объектов DataRow и DataColumn.

Адаптеры данных сохраняют соединение открытым минимально возможное время, чтобы не загружать канал связи. Как только клиентское приложение

получает объект DataSet, соединение с СУБД разрывается. На стороне клиентского приложения остается только локальная копия удаленных данных. На стороне клиентского приложения может производиться вставка, удаление и модификация данных DataTable, но физически БД не будет обновлена.

Для обновления базы данных необходимо передать адаптеру данных объект DataSet. То есть DataSet позволяет имитировать постоянно открытое соединение. На самом деле, все операции выполняются локально с наборами данных в памяти (рис. 16.1).

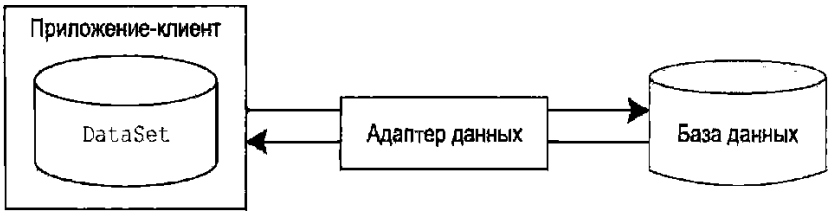


Рисунок 16.1 – Объекты адаптера данных передают объекты DataSet клиенту и возвращают их обратно в БД

На основе данного теоретического материала и лекционного материала по дисциплине «Основы разработки Windows приложений» (тема «Доступ к данным посредством ADO.NET»), выполните задание лабораторной работы.

Для организации связей между связанными таблицами в Visual Studio используются классы производные от BindingSource, которые, в общем случае, позволяют выполнять синхронизацию данных между различными элементами управления формы. Общая схема функционирования объектов класса BindingSource представлена на рис. 16.2.

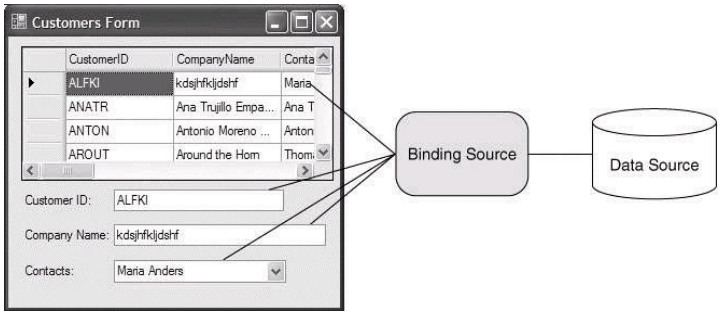


Рисунок 16.2 – Общая схема функционирования объектов BindingSource

16.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

16.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

16.6. Методика и порядок выполнения работы

16.6.1 Создание приложения для редактирования данных в таблице БД

1. Создайте проект Windows Forms Application в среде MS Visual Studio. Присвойте проекту имя: «MyDBProject».

2. Перейдите в режим проектирования главной формы приложения. Переименуйте класс главной формы приложения в окне свойств: Name = MyMainForm.

3. Поместите на форму элемент класса DataGridView. Установите свойство следующим образом: Name = MyDataGridView.

4. Создайте в MS Access базу данных с именем MyDB.mdb, которая содержит одну таблицу с именем «goods» («Товары»).

Таблица содержит поля: id (как первичный ключ, идентификатор), name (наименование товара), price (стоимость). После завершения редактирования структуры таблицы, внесите в базу данных несколько значений и закройте базу данных. Таблица имеет структуру, представленную на рис. 16.3.

Поля таблицы следующих типов:

1) поле «id» – счетчик (выбрать данное поле в качестве первичного ключа);

2) поле «name» – текстовый с длинной 100 символов;

3) поле «price» – денежный.

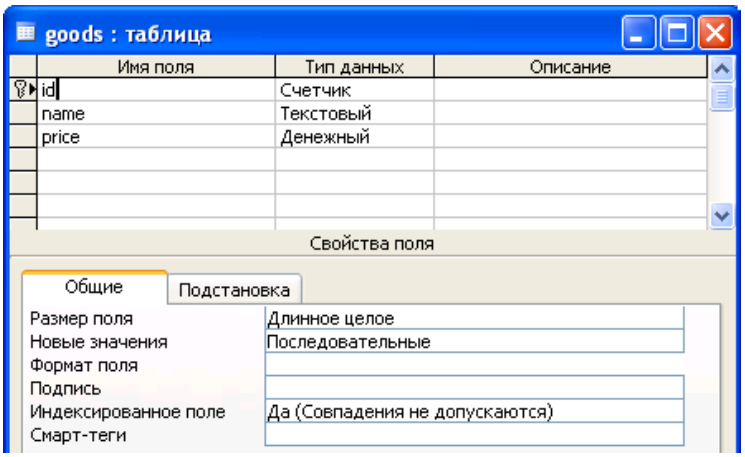


Рисунок 16.3 – Структура таблицы Access

5. Для получения данных в режиме проектирования необходимо отредактировать свойство DataSource элемента MyDataGridView. Нажав на выпадающий список, расположенный напротив данного свойства в окне Properties, выберите ссылку «Add Project Data Source …».

6. В открывшемся диалоговом окне выберите тип источника данных – DataBase. Нажмите «Next».

7. В следующем окне нажмите кнопку «New Connection …» для создания нового объекта соединения.

8. В открывшемся диалоговом окне «Choose Data Source» выберите в окне Data Source тип источника «Microsoft Access Database File». В окне «Database File Name» выберите созданную базу данных MyDB.mdb. Нажмите кнопку «Test Connection». Должно появиться сообщение «Test connection succeeded». Затем кнопку OK.

9. Нажмите «Next». Появится сообщение, в котором спрашивается, желаете ли вы скопировать указанный файл базы данных в папку проекта. Выберите «Нет».

10. В следующем окне введите имя строки соединения MyDBConnectionString. Нажмите «Next».

11. В следующем окне необходимо указать, какие из объектов базы данных будут помещены в DataSet. Отметьте Tables. То есть будут использоваться все таблицы. Нажмите «Finish».

12. В окне проектирования формы приложения появятся объекты myDBDataSet, goodsBindingSource, goodsTableAdapter. Также автоматически будет сгенерирован код данных классов, который можно просмотреть открыв соответствующие элементы в окнах «Solution Explorer» и «Class View».

13. Запустите приложение, убедитесь, что подключение к БД и загрузка данных осуществляется успешно (рис. 16.4). Перед этим необходимо в программе Access ввести данные в таблицу.

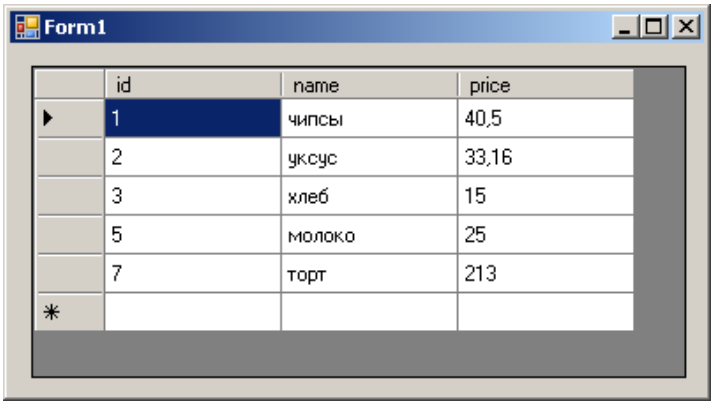


Рисунок 16.4 – Внешний вид окна приложения

14. Выполните удаление нескольких строк в режиме выполнения приложения. Затем откройте базу данных и убедитесь, что строки из источника данных (MyDB.mdb) удалены не были. Открывать базу данных через MS Access не обязательно – можно воспользоваться вкладкой «Database Explorer» среды разработки. Выберите в данном окне таблицу goods и в контекстном меню команду «Show Table Data». Обновление источника данных не произошло из-за того, что после загрузки данных соединение с базой данных было разорвано, а команда на пересылку данных из локального DataSet в источник данных не посылалась.

15. Для внесения изменений в базу данных при удалении, вставки или модификации записей, необходимо выполнить следующие действия:

15.1 Создайте обработчик события FormClosing главной формы приложения.

15.2 В обработчике добавьте строку:

```
this.goodsTableAdapter.Update(this.myDBDataSet.goods).
```

15.3 Откройте класс MyBDDataSet.goodsDataTable, используя вкладку ClassView. В методе InitClass() перед строкой:

base.Columns.Add(this.columnid);

добавьте следующий код:

this.columnid.AutoIncrement = true;

this.columnid.AutoIncrementStep = 5;

16. Запустите приложение, проверьте возможность добавления, модификации и вставки новых записей.

16.6.2 Приложение для обработки данных из связанных таблиц

1. В качестве основы для создания приложения необходимо использовать проект, разработанный в подразделе 16.6.1.

2. Откройте базу данных MyDB.mdb в программе MS Access. Добавьте таблицу «storages» («склады») со следующей структурой:

Имя поля	Тип	Описание
id	Счетчик	Идентификатор. Ключевое поле.
name	Текстовый(50)	Наименование склада (хранилища)

3. В таблицу «goods» добавьте поле, которое будет являться внешним ключом, с именем storage_id и типом «Числовой».

4. В MS Access установите связь между двумя таблицами, как показано на рисунке 16.5.

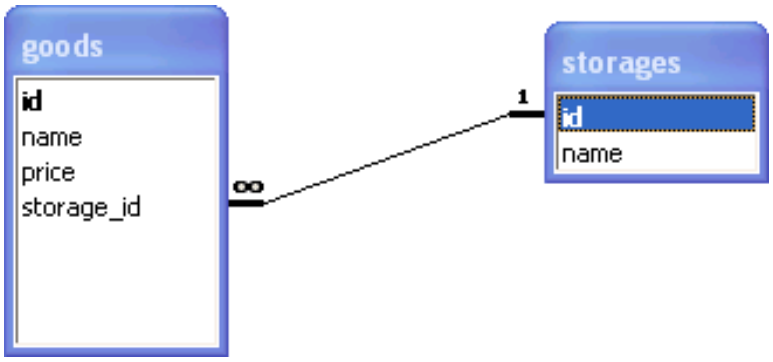


Рисунок 16.5 – Связь между таблицами базы данных

5. Откройте проект приложения в среде Visual Studio. Откройте окно дизайнера главной формы. Перейдите на вкладку «Data Sources» среды разработки, щелкните правой кнопкой мыши на объекте MyDBDataSet и выберите команду «Configure DataSet with Wizard…».

6. Отметьте таблицу «storages», тем самым добавив ее в набор данных MyDBDataSet. Нажмите кнопку «Finish».

7. Создайте DataGridView на форме приложения. Присвойте данному элементу имя «ParentDataGridView». В качестве источника данных (свойство «DataSource») выберите таблицу «storages».

8. Запустите приложение и убедитесь, что информация в обоих DataGridView отображается.

9. Для корректного отображения информации, необходимо чтобы при выборе склада в одном окне DataGridView, в другом окне выводились только товары, принадлежащие этому складу. Для этого необходимо установить свойства объектов приложения следующим образом:

9.1. Установить свойство DataSource объекта storagesBindingSource равным MyDBDataSource.

9.2. Установить свойство DataMember объекта storagesBindingSource равным storages.

9.3. Установить свойство DataSource объекта ParentDataGridView равным storagesBindingSource.

9.4. Установить свойство DataSource объекта goodsBindingSource равным storagesBindingSource.

9.5. Установить свойство DataMember объекта goodsBindingSource равным FK_storages_goods.

9.6. Установить свойство DataSource объекта MyDataGridView равным goodsBindingSource.

10. Запустите приложение, убедитесь, что связь между представлениями связанных таблиц настроена.

11. Исправьте ошибки в приложении, переименуйте названия столбцов в DataGridView, поле «id» необходимо скрыть в обоих представлениях, так как они не несут смысловой нагрузки.

12. Модернизируйте базу данных и приложение в соответствие с вариантом. Необходимо добавить в базу не менее двух таблиц, реализовать связи между ними, корректно отобразить данные из всех таблиц с учетом связей.

Варианты индивидуальных заданий

Выполните задание в соответствие с индивидуальным вариантом. Каждый вариант предусматривает выполнение аналогичного задания по разработке приложения для обработки данных из связанных таблиц и базы данных.

База данных должна содержать минимум две связанных таблицы; в каждой таблице должно быть минимум 5 полей.

При проектировании элемента типа DataGridView необходимо указать заголовки столбцов, изменив используемые заголовки по умолчанию. Добавьте на форму кнопку «Сохранить», при нажатии на которую будет выводиться сообщение «Вы действительно хотите сохранить все изменения?». В случае положительного ответа, все изменения должны отразиться в источнике данных. В случае отрицательного ответа, необходимо отменить изменения, сделанные в DataGridView.

Варианты заданий:

1. Учет сетевого и компьютерного оборудования.
2. Учет работы с пластиковыми картами.
3. Учет работы поликлиники.
4. Учет работы пассажирского автотранспорта.
5. Учет поступлений книг на склад.
6. Учет поставок и реализации продуктов питания.
7. Учет поставок и реализации компьютеров.
8. Учет поставок и реализации автомобилей ВАЗа.
9. Учет материальных ценностей.
10. Учет заявок на производство кондитерских и хлебобулочных изделий.
11. Учет материальных ценностей при реализации металла и металлоизделий.
12. Учет заказов и продаж.
13. Учет бегущих строк на телевидении для частных лиц.
14. Трудоустройство.
15. Учет банковских операций с валютными вкладами юридических лиц.
16. Учет курсовых работ.
17. Поставка и реализация программного обеспечения на компакт-дисках.
18. Подписка.
19. Санкции ГИБДД.
20. Регистратура поликлиники.
21. Пункт проката видео кассет.
22. Спортивные клубы.
23. Гостиница.
24. Поставка и реализация ювелирных изделий.
25. Автоматизация работы станции технического обслуживания автомобилей.
26. Купля-продажа квартир.
27. Каталог микросхем.

16.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю.

16.8. Вопросы для защиты работы

1. Что такое несвязный доступ к БД?
2. Какие типы данных используются при реализации несвязного доступа к БД?
3. Что представляет собой класс DataSet?
4. Что представляет собой класс DataTable?
5. Как выполняется обновление источника данных при несвязном доступе?
6. Для чего в приложениях .NET используется адаптер данных?

16.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 17. ОСНОВЫ ПОСТРОЕНИЯ ПРИЛОЖЕНИЙ WINDOWS PRESENTATION FOUNDATION

17.1. Цель и содержание

Цель лабораторной работы: научиться проектировать интерфейс приложений на основе технологии WPF.

Задачами лабораторной работы являются:

– изучить возможности работы с классом Window;

– изучить принципы использования контейнеров компоновки;

– научиться обрабатывать события простых элементов управления.

17.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

17.3. Теоретическое обоснование

17.3.1 Класс Window.

Объект Window – основной элемент традиционных приложений, в нем отображается их основное содержимое. В Windows Presentation Foundation (WPF) за объектом Window скрывается обычное окно Win32. Операционная система не различает окна с WPF- и Win32-содержимым: обрамление рисуется одинаково, на панели задач они неотличимы друг от друга и т. д. Обрамление (Chrome) – это просто другое название области окна, которая обрамляет клиентскую область и в числе прочего содержит кнопки Minimize (Свернуть), Maximize (Развернуть) и Close (Закрыть).

Объект Window – это абстракция окна Win32 (так же, как класс Form в Windows Forms), содержащая ряд простых методов и свойств. Создав объект Window, программист может показать его на экране с помощью метода Show,

скрыть – методом Hide() (это то же самое, что присвоить свойству Visibility значение Hidden или Collapsed) и закрыть навсегда – методом Close().

Внешним видом Window можно управлять с помощью таких свойств, как Icon (иконка приложения), Title (интерпретируется как заголовок окна) и WindowStyle. Для управления положением на экране служат свойства Left и Top. Более осмысленного поведения можно добиться, присваивая свойству WindowStartupLocation значение CenterScreen или CenterOwner. Короче говоря, с помощью установки свойств можно делать почти все, что принято ожидать от окна. Скажем, если установить для Topmost значение true, то окно будет всегда отображаться поверх других, а если присвоить ShowInTaskbar значение false, то значок окна не будет показываться на панели задач.

Всякий раз, как окно становится активным или неактивным (например, из-за того, что пользователь переключается между разными окнами), возникает событие Activated или Deactivated объекта Window.

На рис. 17.1 представлено определение окна с использованием языка разметки XAML. Данное определение генерируется Visual Studio по умолчанию.

```
1  <Window x:Class="WpfExample.Window1"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="Window1" Height="300" Width="300">
5      <Grid>
6
7      </Grid>
8  </Window>
```

Рисунок 17.1 – Шаблон окна приложения, генерируемого по умолчанию.

Окно может содержать только один дочерний элемент. Для размещения в окне множества дочерних элементов управления в WPF используется подход: сначала в фрейме окна размещается специализированная панель (контейнер компоновки), а затем, в данном контейнере располагаются элементы управления.

17.3.2 Контейнеры компоновки

Для размещения элементов управления в WPF используются специализированные компоненты: контейнеры компоновки. В технологии

Windows Forms применяется механизм абсолютного позиционирования элементов управления: задаются координаты верхнего левого угла элемента (свойства Top, Left) и его размер (свойства Width и Height).

В WPF компоновка определяется используемым контейнером. Окно в WPF должно реализовывать следующие принципы:

1. Элементы (такие как элементы управления) не должны иметь явно установленных размеров. Вместо этого они растут, чтобы уместить свое содержимое. Например, кнопка увеличивается при добавлении в нее текста. Можно ограничить элементы управления приемлемыми размерами, устанавливая максимальное и минимальное их значение.

2. Элементы не указывают свою позицию в экранных координатах. Вместо этого они упорядочиваются своим контейнером на основе размера, порядка и (необязательно) другой информации, специфичной для контейнера компоновки. Для добавления пробелов между элементами служит свойство Margin.

3. Контейнеры компоновки «разделяют» доступное пространство между своими дочерними элементами. Они пытаются обеспечить для каждого элемента его предпочтительный размер (на основе его содержимого), если только позволяет свободное пространство. Они могут также выделять дополнительное пространство одному или более дочерним элементам.

4. Контейнеры компоновки могут быть вложенными. Типичный пользовательский интерфейс начинается с Grid – наиболее развитого контейнера, и содержит другие контейнеры компоновки, которые организуют меньшие группы элементов, такие как текстовые поля с метками, элементы списка, значки в панели инструментов, колонка кнопок и т.д.

Все контейнеры компоновки WPF являются панелями, которые унаследованы от абстрактного класса System.Windows.Controls.Panel (рис. 17.2). Класс Panel добавляет небольшой набор возможностей всем производным классам.

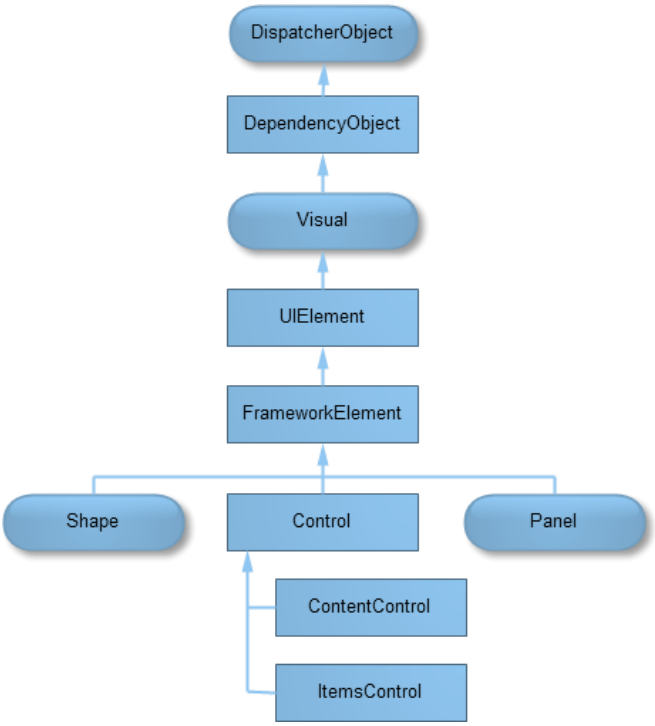


Рисунок 17.2 – Иерархия классов WPF.

Сам по себе базовый класс Panel – это абстрактный класс, не допускающий создания объектов. WPF предлагает набор производных от Panel классов, которые можно использовать для организации компоновки. Основные контейнеры компоновки перечислены в табл. 17.1. Как и все элементы управления WPF, а также большинство визуальных элементов, эти классы находятся в пространстве имен System.Windows.Controls.

Таблица 17.1 – Контейнеры компоновки WPF.

Контейнер	Описание
StackPanel	Размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна.
WrapPanel	Размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации WrapPanel располагает элементы в строке слева направо, затем переходит к следующей строке. В вертикальной ориентации WrapPanel располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов.
DockPanel	Выравнивает элементы по краю контейнера.

Grid	Выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки.
UniformGrid	Помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто.
Canvas	Позволяет элементам позиционироваться абсолютно — по фиксированным координатам. Этот контейнер компоновки более всего похож на традиционный компоновщик Windows Forms, но не предусматривает средств привязки и стыковки. В результате это неподходящий выбор для окон переменного размера.

17.3.2 Контейнер StackPanel

На рис. 17.3 представлен простой пример компоновки элементов с использованием контейнера StackPanel.

```
<Window x:Class="WpfExample.Window31"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window31" Height="127" Width="300">
    <StackPanel>
        <Button>Кнопка 1</Button>
        <Button>Кнопка 2</Button>
        <Button>Кнопка 3</Button>
        <Button>Кнопка 4</Button>
    </StackPanel>
</Window>
```

Рисунок 17.3 – Использование StackPanel.

Результат подобной компоновки представлен на рис. 17.4. Без дополнительной информации о порядке, координатах и размерах кнопок с помощью контейнера StackPanel удалось расположить элементы вертикально.

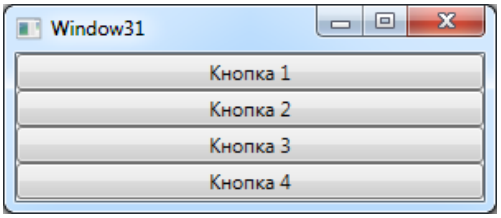


Рисунок 17.4 – Окно с контейнером StackPanel.

По умолчанию панель StackPanel располагает элементы сверху вниз, устанавливая высоту каждого из них такой, которая необходима для отображения его содержимого.

В данном примере это значит, что размер кнопок устанавливается достаточно большим для спокойного размещения текста внутри них. Все элементы растягиваются на полную ширину StackPanel, которая равна ширине окна. Если вы расширите окно, StackPanel также расширится, и кнопки растянутся, чтобы заполнить ее.

StackPanel может также использоваться для упорядочивания элементов в горизонтальном направлении за счет установки свойства Orientation (рис. 17.5):

<StackPanel Orientation=llHorizontalll>

```
<Window x:Class="WpfExample.Window31"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window31" Height="127" Width="300">
    <StackPanel Orientation="Horizontal">
        <Button>Кнопка 1</Button>
        <Button>Кнопка 2</Button>
        <Button>Кнопка 3</Button>
        <Button>Кнопка 4</Button>
    </StackPanel>
</Window>
```

Рисунок 17.5 – Использование StackPanel.

Результат представлен на рис. 17.6

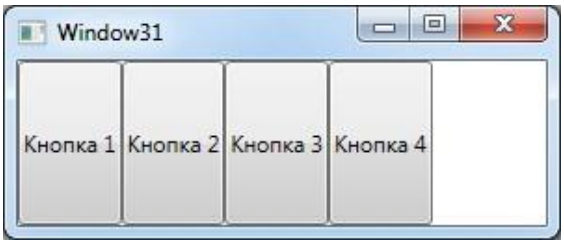


Рисунок 17.6 – Использование горизонтального размещения StackPanel.

Размещение элементов управления определяется контейнером, но элементы управления могут повлиять на этот процесс, явно задавая свойства. Свойства элементов, которые влияют на размещение в контейнере представлены в таблице 17.2.

Таблица 17.2 – Свойства компоновки

Наименование свойства	Описание
HonzontalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по горизонтали. Доступные значения: Center, Left, Right или Stretch.
VerticalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по вертикали. Доступные значения: Center, Top, Bottom или Stretch.
Margin	Добавляет некоторое пространство вокруг элемента. Свойство Margin – это экземпляр структуры System.Windows.Thickness, с отдельными компонентами для верхней, нижней, левой и правой граней.
MinWidth и MinHeight	Устанавливает минимальные размеры элемента. Если элемент слишком велик, чтобы поместиться в его контейнер компоновки, он будет усечен.
MaxWidth и MaxHeight	Устанавливает максимальные размеры элемента. Если контейнер имеет свободное пространство, элемент не будет увеличен сверх указанных пределов, даже если свойства HonzontalAlignment и VerticalAlignment установлены в Stretch.
Width и Height	Явно устанавливают размеры элемента. Эта установка переопределяет значение Stretch для свойств HorizontalAlignment и VerticalAlignment. Однако данный размер не будет установлен, если выходит за пределы, заданные в MinWidth, MinHeight, MaxWidth и MaxHeight.

17.3.3 Элемент Border

Border не является одной из панелей компоновки, но это удобный элемент, который часто используется.

Класс Border принимает единственную порцию вложенного содержимого (которым часто является панель компоновки) и добавляет фон или рамку вокруг него. Для работы с Border понадобятся свойства, перечисленные в табл. 17.3.

Таблица 17.3 – Свойства класса Border

Наименование свойства	Описание

Background	С помощью объекта Brush устанавливает фон, который появляется под содержимым. Можно использовать сплошной цвет либо что-нибудь более сложное.
BorderBrush и BorderThickness	Устанавливают цвет рамки, который появляется на границе объекта Border, и толщину рамки. Для отображения рамки потребуется установить оба свойства.
CornerRadius	Позволяет скруглить углы рамки. Чем больше значение CornerRadius, тем заметнее эффект.
Padding	Добавляет пространство между рамкой и содержимым внутри нее.

На рис. 17.7 представлен фрагмент кода XAML, использующий данные свойства.

```
<Window x:Class="WpfExample.Window31"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window31" Height="127" Width="300">
    <Border Background="GreenYellow" BorderBrush="Red"
            BorderThickness="5" CornerRadius="10" Padding="10">
        <StackPanel Orientation="Vertical">
            <Button>Кнопка 1</Button>
            <Button>Кнопка 2</Button>
            <Button>Кнопка 3</Button>
            <Button>Кнопка 4</Button>
        </StackPanel>
    </Border>

</Window>
```

Рисунок 17.7 – Использование свойств Border

На рис. 17.8 показано результирующее окно.

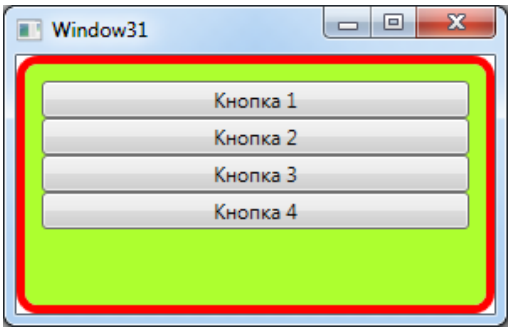


Рисунок 17.8 – Результат применения Border

17.3.4 Элементы WrapPanel и DockPanel

Панель WrapPanel располагает элементы управления в доступном пространстве – по одной строке или колонке за раз. По умолчанию свойство

WrapPanel.Orientation установлено в Horizontal; элементы управления располагаются слева направо, затем – в следующих строках. Установка значения Vertical для свойства WrapPanel. Orientation приводит к размещению элементов в нескольких колонках.

Подобно StackPanel, панель WrapPanel действительно предназначена для управления мелкими деталями пользовательского интерфейса, а не компоновкой всего окна. Например, WrapPanel можно использовать для удержания вместе кнопок в элементе управления, подобном панели инструментов.

Панель DockPanel обеспечивает более интересный вариант компоновки. Эта панель растягивает элементы управления вдоль одной из внешних границ. Простейший способ представить это – вообразить панель инструментов, которая присутствует в верхней части многих Windows-приложений. Такие панели инструментов пристыковываются к верхней части окна. Как и в случае StackPanel, пристыкованные элементы должны выбрать один аспект компоновки. Например, если вы пристыковать кнопку к верхней части DockPanel, она растянется на всю ширину DockPanel, но получит высоту, которая ей понадобится (на основе своего содержимого и свойства MinHeight). С другой стороны, если пристыковать кнопку к левой стороне контейнера, ее высота будет растянута для заполнения контейнера, но ширина будет установлена по необходимости.

Дочерние элементы DockPanel прикрепляются к одной из сторон панели посредством задания присоединенного свойства Dock. Элементы на самом деле не имеют этого свойства – они приобретают его после помещения в контейнер.

На рис. 17.9 показан пример использования DockPanel.

```
<Window x:Class="WpfExample.DockPanelExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="DockPanelExample" Height="300" Width="300">
    <DockPanel>
        <Button Content="кнопка 1" DockPanel.Dock="Top"/>
        <Button Content="кнопка 2" DockPanel.Dock="Bottom"/>
        <Button Content="кнопка 3" DockPanel.Dock="Left"/>
        <Button Content="кнопка 4" DockPanel.Dock="Right"/>
        <Button Content="кнопка 5"/>
    </DockPanel>
</Window>
```

Рисунок 17.9 – Использование DockPanel

На рис. 17.10 представлен результат.

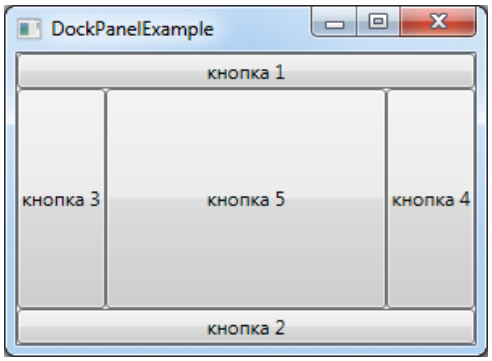


Рисунок 17.10 – Контейнер DockPanel в окне приложения

17.3.5 Контейнер Grid

Элемент управления Grid – это наиболее мощный контейнер компоновки в WPF. Большая часть того, что можно достичь с помощью других элементов управления компоновкой, также возможно и в Grid. Контейнер Grid является идеальным инструментом для разбиения окна на меньшие области, которыми можно управлять с помощью других панелей.

При добавлении в Visual Studio нового документа XAML для окна автоматически добавляются дескрипторы Grid в качестве контейнера первого уровня, вложенного внутрь корневого элемента Window.

Хотя Grid задуман как невидимый элемент, можно установить свойство Grid.ShowGridLines в true и получить наглядное представление о нем. Это средство на самом деле не предназначено для украшения окна. В действительности это средство для облегчения отладки, которое предназначено для того, чтобы наглядно показать, как Grid разделяет пространство на отдельные области. Благодаря ему, появляется возможность точно контролировать то, как Grid выбирает ширину колонок и высоту строк.

Создание компоновки на основе Grid – поэтапный процесс. Сначала выбирается необходимое количество колонок и строк. Затем каждому содержащемуся элементу назначается соответствующая строка и колонка, тем самым помещая его в правильное место.

Колонки и строки создаются путем заполнения объектами коллекции Grid.ColumnDefinitions и Grid.RowDefinitions. Например, если необходимо создать две строки и три колонки, то используются следующие дескрипторы:

```
<Window x:Class="WpfExample.GridExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="GridExample" Height="300" Width="300">
    <Grid>
        <Grid Background="GreenYellow">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <Button Grid.Column="0" Grid.Row="0" Content="Кнопка (0, 0)"/>
            <Button Grid.Column="1" Grid.Row="1" Content="Кнопка (1, 1)"/>
            <Button Grid.Column="2" Grid.Row="2" Content="Кнопка (2, 2)"/>
            <Button Grid.Column="2" Grid.Row="0" Content="Кнопка (2, 0)"/>
            <Button Grid.Column="0" Grid.Row="2" Content="Кнопка (0, 2)"/>
        </Grid>
    </Grid>
</Window>
```

Рисунок 17.11 – Контейнер Grid

Результирующее окно представлено на рис. 17.12.

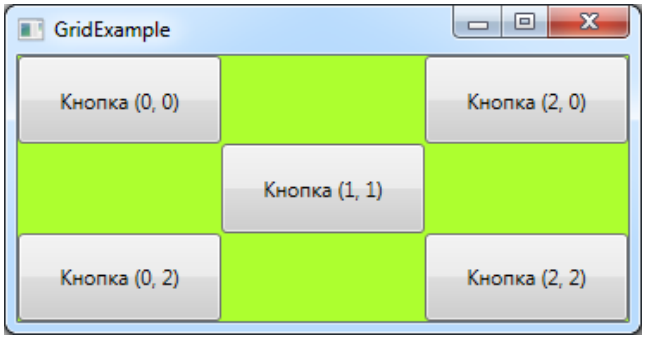


Рисунок 17.12 – Расположение кнопок в контейнере Grid

17.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды

разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

17.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

17.6. Методика и порядок выполнения работы

Для закрепления теоретического материала и получения практических навыков работы с технологий WPF решим следующую задачу:

Условие. Реализовать приложение для вычисления выражения:

$$S = \begin{cases} \sin(f \cdot a),\ f \in \{4,5,6,7,8,9\}; \\ \cos(f \cdot a) + \sin(f \cdot b),\ f \in \{10,20,30,40\}; \\ c \cdot a^{2} + d \cdot b^{2},\ c \in \{0,1\},\ d \in \{-1,0,1\}; \\ \sum_{i=0}^{d} (c+a)^{i},\ d = const,\ c \in \{0,1,2,3,4,5\}. \end{cases}$$

Решение. В примере вычисление выражения может быть организовано по четырем возможным направлениям, причем критерии выбора одной из формул не определены. Это значит, что необходимо дать пользователю возможность

производить выбор формулы для расчета. Решение задачи реализуем последовательно выполняя следующие стадии:

1. Создайте приложение WPF Application. В качестве имени проекта необходимо указать «WpfExample». При разработке приложения необходимо придерживаться какого-либо наброска интерфейса, например, изображенного на рис. 17.13.

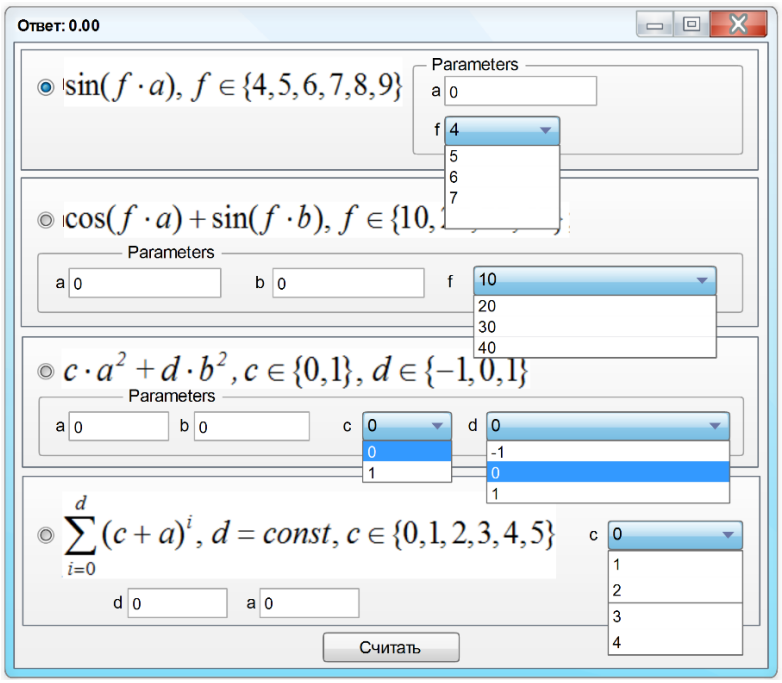


Рисунок 17.13 – Проект формы: приблизительное расположение элементов управления формы

На рис. 17.13 представлен приблизительный интерфейс, но даже из этого макета можно выделить следующие особенности:

– форма разбита на 4 области, каждая из которых представляет собой панель с параметрами для вычисления выражения;

– каждая из четырех панелей формы может быть отмечена пользователем посредством выбора с помощью RadioButton;

– при нажатии кнопки «Считать» результат вычисления, выбранного пользователем выражения, выводится в заголовок окна.

2. Перейдем к созданию интерфейса с использованием языка XAML. Для начала определим контейнер компоновки верхнего уровня. По умолчанию в классе

Window установлен контейнер Grid. Удалите его и добавьте контейнер StackPanel. Измените свойства Window. В результате этих действий получим код:

```
1  <Window x:Class="WpfExample.MainWindow"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="Ответ: "
5          Height="420" Width="480"
6          WindowStartupLocation="CenterScreen"
7          SizeToContent="Height">
8      <StackPanel>
9          <RadioButton IsChecked="True" Margin="5">
10             <WrapPanel...>
37         </RadioButton>
38         <RadioButton Margin="5">
39             <StackPanel...>
57         </RadioButton>
58         <RadioButton Margin="5">
59             <StackPanel...>
81         </RadioButton>
82         <RadioButton Margin="5">
83             <WrapPanel...>
96         </RadioButton>
97         <Button Content="Считать" Width="100"/>
98     </StackPanel>
99 </Window>
```

Рисунок 17.14 – Создание контейнера и добавление элементов в контейнер

Рассмотрим данный пример подробнее:

– в строке 1 расположен открывающий дескриптор <Window> главного окна приложения; в строках 4-7 устанавливаются атрибуты главного окна (поясните, какие именно свойства окна задаются данными атрибутами);

– в строке 8 определяется контейнер <StackPanel> (контейнер <Grid> был удален);

– внутри контейнера StackPanel определены четыре элемента RadioButton и кнопка «Считать»; обратите внимание, что внутри каждого элемента RadioButton добавляется свой контейнер компановки (в первом и четвертом – WrapPanel, во втором и третьем – StackPanel).

3. Для удобства отображения вложенности элементов в Visual Studio предусмотрен просмотр элементов окна в виде дерева. Откройте окно «Document

Outline» (рис. 17.15) и убедитесь, что иерархия элементов соответствует задуманной.

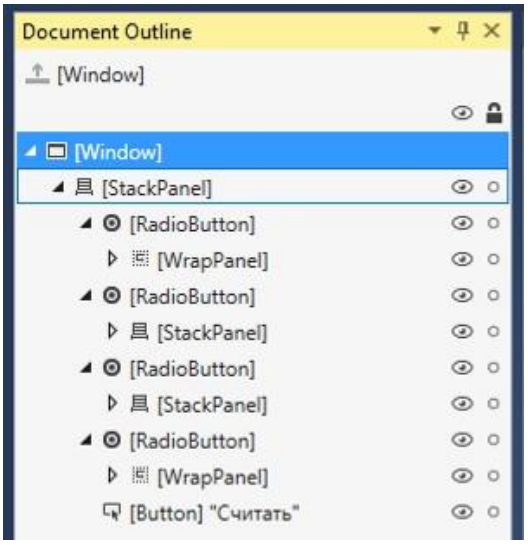


Рисунок 17.15 – Просмотр иерархии элементов главного окна

4. Далее произведем наполнение каждого элемента RadioButton. При дальнейшей разработке присвоим элементам имена, чтобы обращаться к ним из кода на языке C#.

4.1 На рис. 17.16 представлено содержимое первого RadioButton. После создания данного фрагмента кода необходимо проконтролировать вложенность элементов с использованием окна «Document Outline» (рис. 17.17).

```
9         <RadioButton IsChecked="True" Margin="5" Name="Radio1">
10            <WrapPanel>
11                <Image Source="p1.png" Height="30"/>
12                <GroupBox Header="Parameters">
13                    <Grid>
14                        <Grid.RowDefinitions>
15                            <RowDefinition/>
16                            <RowDefinition/>
17                        </Grid.RowDefinitions>
18                        <Grid.ColumnDefinitions>
19                            <ColumnDefinition/>
20                            <ColumnDefinition Width="100"/>
21                        </Grid.ColumnDefinitions>
22                        <Label Content="a" Grid.Column="0" Grid.Row="0"/>
23                        <TextBox Text="0,00" Grid.Column="1" Grid.Row="0"
24                                 MinWidth="70" Name="R1TextA"/>
25                        <Label Content="f" Grid.Column="0" Grid.Row="1"/>
26                        <ComboBox Grid.Column="1" Grid.Row="1" SelectedIndex="0"
27                                  Name="R1ComboF">
28                            <ComboBoxItem>4</ComboBoxItem>
29                            <ComboBoxItem>5</ComboBoxItem>
30                            <ComboBoxItem>6</ComboBoxItem>
31                            <ComboBoxItem>7</ComboBoxItem>
32                            <ComboBoxItem>8</ComboBoxItem>
33                            <ComboBoxItem>9</ComboBoxItem>
34                        </ComboBox>
35                    </Grid>
36                </GroupBox>
37            </WrapPanel>
38        </RadioButton>
```

Рисунок 17.16 – Дескриптор RadioButton для представления первой формулы

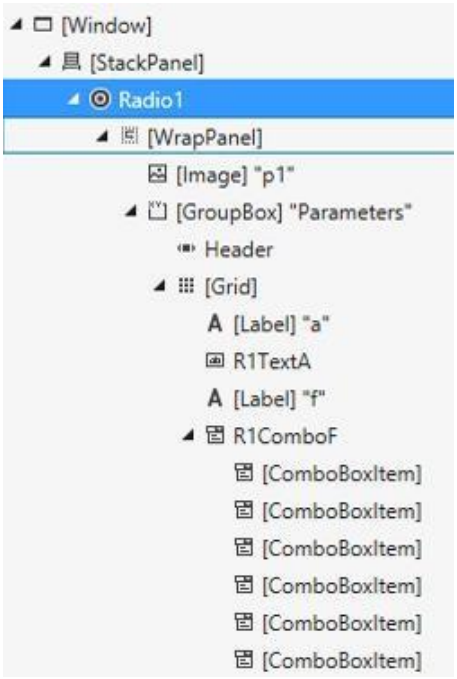


Рисунок 17.17 – Дескриптор RadioButton (иерархическое представление)

4.2 На рис. 17.18 представлено наполнение второго RadioButton для представления параметров вычисления второй формулы выражения. И снова необходимо просмотреть иерархию элементов управления с использованием окна «Document Outline».

```
9   <RadioButton IsChecked="True" Margin="5" Name="Radio1"...>
39  <RadioButton Margin="5" Name="Radio2">
40      <StackPanel>
41          <Image Height="23" Source="p2.png"/>
42          <GroupBox Header="Parameters">
43              <StackPanel Orientation="Horizontal">
44                  <Label Content="a"/>
45                  <TextBox Text="0,00" MinWidth="70" Name="R2TextA"/>
46                  <Label Content="b"/>
47                  <TextBox Text="0,00" MinWidth="70" Name="R2TextB"/>
48                  <Label Content="f"/>
49                  <ComboBox MinWidth="100" Name="R2ComboF">
50                      <ComboBoxItem>10</ComboBoxItem>
51                      <ComboBoxItem>20</ComboBoxItem>
52                      <ComboBoxItem>30</ComboBoxItem>
53                      <ComboBoxItem>40</ComboBoxItem>
54                  </ComboBox>
55              </StackPanel>
56          </GroupBox>
57      </StackPanel>
58  </RadioButton>
```

Рисунок 17.18 – Дескриптор RadioButton для представления второй формулы

4.3 На рис. 17.19 представлен код XAML для третьего дескриптора RadioButton.

```
9   <RadioButton IsChecked="True" Margin="5" Name="Radio1"...>
39  <RadioButton Margin="5" Name="Radio2"...>
59  <RadioButton Margin="5" Name="Radio3">
60      <StackPanel>
61          <Image Source="p3.png" Height="30"/>
62          <GroupBox Header="Parameters">
63              <StackPanel Orientation="Horizontal">
64                  <Label Content="a"/>
65                  <TextBox Text="0,00" MinWidth="50" Name="R3TextA"/>
66                  <Label Content="b"/>
67                  <TextBox Text="0,00" MinWidth="50" Name="R3TextB"/>
68                  <Label Content="c"/>
69                  <ComboBox MinWidth="50" SelectedIndex="0" Name="R3ComboC">
70                      <ComboBoxItem Content="0"/>
71                      <ComboBoxItem Content="1"/>
72                  </ComboBox>
73                  <Label Content="d"/>
74                  <ComboBox MinWidth="70" SelectedIndex="0" Name="R3ComboD">
75                      <ComboBoxItem Content="-1"/>
76                      <ComboBoxItem Content="0"/>
77                      <ComboBoxItem Content="1"/>
78                  </ComboBox>
79              </StackPanel>
80          </GroupBox>
81      </StackPanel>
82  </RadioButton>
```

Рисунок 17.19 – Дескриптор RadioButton для представления третьей формулы

4.4 Затем наполним содержимым четвертый дескриптор RadioButton (рис. 17.20).

```
84  <RadioButton Margin="5" Name="Radio4">
85      <WrapPanel>
86          <Image Source="p4.png" Height="60"/>
87          <Label Content="c" VerticalAlignment="Center"/>
88          <ComboBox MinWidth="70" VerticalAlignment="Center" Name="R4ComboC"
89                    SelectedIndex="0">
90              <ComboBoxItem Content="0"/>
91              <ComboBoxItem Content="1"/>
92              <ComboBoxItem Content="2"/>
93          </ComboBox>
94          <Label Content="d" VerticalAlignment="Center"/>
95          <TextBox Text="0,00" Width="70"
96                   VerticalAlignment="Center" Name="R4TextD"/>
97          <Label Content="a" VerticalAlignment="Center"/>
98          <TextBox Text="0,00" Width="70"
99                   VerticalAlignment="Center" Name="R4TextA"/>
100     </WrapPanel>
101 </RadioButton>
```

Рисунок 17.20 – Дескриптор RadioButton для представления четвертой формулы

5. В результате выполнения этапов 1 – 4 получим главное окно приложения, внешний вид которого показан на рис. 17.21 (показано окно в режиме

проектирования). На рис. 17.22 показано это же окно в режиме выполнения приложения.

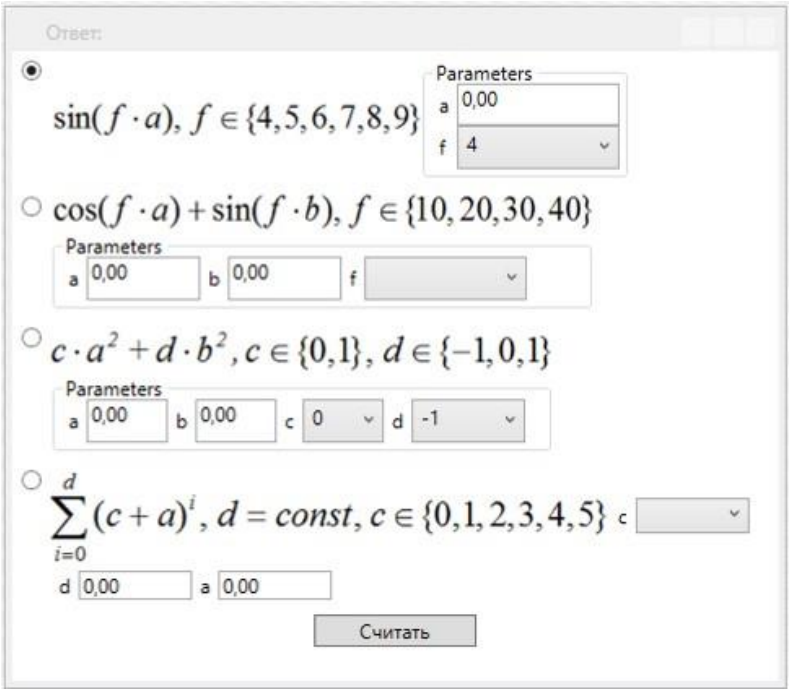


Рисунок 17.21 – Главное окно приложения в режиме проектирования

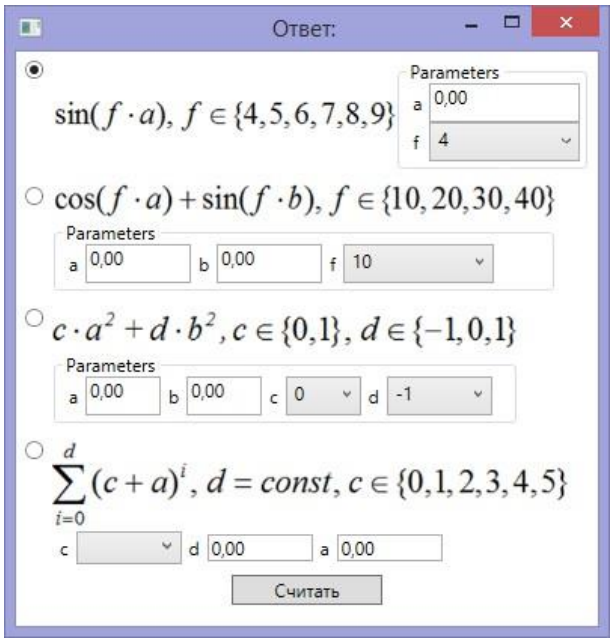


Рисунок 17.22 – Главное окно приложения в режиме выполнения

6. Окно приложения создано, теперь необходимо написать код обработчика события нажатия кнопки «Считать». Для этого просто добавьте соответствующий атрибут в определение кнопки (рис. 17.23).

```
99          <Button Content="Считать" Width="100"
100                 Click="Calc_Click"/>
```

Рисунок 17.23 – Добавление обработчика нажатия кнопки.

В результате выполнения данного действия Visual Studio откроет редактор кода C# и загрузит в него файл MainWindow.xaml.cs. В отличие от файла MainWindow.xaml (в котором определен код XAML окна), в данном файле содержится листинг кода на языке C#.

Добавьте в обработчик события нажатия кнопки следующий код:

```
29          private void Calc_Click(object sender, RoutedEventArgs e)
30          {
31              // Определяем, какой RadioButton выбран
32              // и в зависимости от этого расчитываем нужную формулу
33              if (Radio1.IsChecked.GetValueOrDefault())
34              {
35                  double a = Convert.ToDouble(R1TextA.Text);
36                  double f = Convert.ToDouble(R1ComboF.Text);
37                  this.Title = "Ответ: " + Math.Sin(f*a).ToString("F");
38              }
39
40              if (Radio2.IsChecked.GetValueOrDefault())
41              {
42                  double a = Convert.ToDouble(R2TextA.Text);
43                  double b = Convert.ToDouble(R2TextB.Text);
44                  double f = Convert.ToDouble(R2ComboF.Text);
45                  this.Title = "Ответ: "
46                      + (Math.Cos(f*a) + Math.Sin(f*b)).ToString("F");
47              }
48
49              if (Radio3.IsChecked.GetValueOrDefault())
50              {
51                  double a = Convert.ToDouble(R3TextA.Text);
52                  double b = Convert.ToDouble(R3TextB.Text);
53                  double c = Convert.ToDouble(R3ComboC.Text);
54                  double d = Convert.ToDouble(R3ComboD.Text);
55                  this.Title = "Ответ: "
56                      + (c*a*a + d*b*b).ToString("F");
57              }
58
```

```
59              if (Radio4.IsChecked.GetValueOrDefault())
60              {
61                  double a = Convert.ToDouble(R4TextA.Text);
62                  double d = Convert.ToDouble(R4TextD.Text);
63                  double c = Convert.ToDouble(R4ComboC.Text);
64                  double res = 1;
65                  for (int i = 0; i < d; i++)
66                  {
67                      res = res*(c + a) + 1;
68                  }
69                  this.Title = "Ответ: "
70                      + res.ToString("F");
71              }
72          }
```

Рисунок 17.24 – Код обработчика нажатия кнопки «Считать»

Представленный код простой и в пояснениях не нуждается.

7. Запустите приложение. Протестируйте работу программы при различных значениях входных параметров.

8. Выполните задание в соответствии с индивидуальным вариантом. В каждом варианте необходимо реализовать вычисление выражения. Окно приложения необходимо разрабатывать с использованием технологии WPF. Необходимо самостоятельно спроектировать внешний вид окна, реализовать адаптивное расположение элементов управления.

Индивидуальное задание

1.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{p^t\cdot x^3+f^J\cdot y^2}{i\cdot j}\cdot$	2.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{t^t\cdot x^t+f^J\cdot y^J}{t\cdot i\cdot j}\cdot$
3.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{p^i\cdot y^j}{i\cdot j}$	4.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\sin(x)\cdot x^t+f^J\cdot y^J}{i\cdot j}$
5.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\sin(x)\cdot x^i+\cos(y)\cdot y^j}{i\cdot j}$	6.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\lvert x^i\rvert+\cos(y)\cdot y^j}{i\cdot j}$
7.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{i\cdot x^t+j\cdot y^J}{i\cdot j}$	8.	$Z=\sum_{i=1}^{N}\sum_{j=1}^{K}\frac{i\cdot\cos(x^t)+j\cdot\sin(y^J)}{i\cdot j}$

9.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\cos(x^i)+j}{i\cdot j}$	10.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\cos(y^i)+j\cdot x}{i\cdot j}$
11.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\sin(y^i)+i\cdot x}{(i+1)\cdot j}$	12.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\sin(y\cdot x)+i\cdot y}{(i+1)\cdot(j+2)}$
13.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{2\cdot x^i+3\cdot y^j}{i^2\cdot j^2}$	14.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{i\cdot x^{2\cdot i}+j\cdot y^{2\cdot j}}{i^2\cdot j^2}$
15.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{p^{i-1}\cdot x+f^{j-1}\cdot y^2}{i\cdot j}$	16.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{a^{i-1}\cdot x^i+f^j\cdot y^j}{(i+1)\cdot j}$
17.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{5\cdot x+3\cdot y}{i^2\cdot j^2}$	18.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\cos(y^i)+j\cdot\sin(x)}{i\cdot(j-1)}$
19.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{\lvert x^i\rvert+\cos(y)\cdot y^j}{(i+1)\cdot j}$	20.	$Z = \sum_{i=1}^{N}\sum_{j=1}^{K}\frac{(p-1)^i\cdot y^j}{i\cdot j}$

17.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю.

17.8. Вопросы для защиты работы

1. Опишите основные свойства класса Window.
2. Что такое контейнер компоновки? Для чего они используются? Опишите свойства класса Panel.
3. Назовите основные контейнеры компоновки. Дайте характеристику и опишите особенности контейнеров компоновки.
4. Что такое присоединенное свойство? Приведите примеры.
5. Какой контейнер применяется в окне приложения как контейнер компоновки по умолчанию?
6. Какие свойства элементов управления влияют на положение элементов внутри контейнера компоновки?

17.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1], [3].

ЛАБОРАТОРНАЯ РАБОТА 18. ОБРАБОТКА СОБЫТИЙ WPF

18.1. Цель и содержание

Цель лабораторной работы: научиться проектировать интерфейс приложений на основе технологии WPF.

Задачами лабораторной работы являются:

– изучить основные классы для представления элементов управления;

– научиться использовать элементы управления;

– изучить механизм событий WPF;

– научиться использовать событийную модель WPF.

18.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

18.3. Теоретическое обоснование

18.3.1 Простые элементы управления

Элементы управления (control) являются потомками класса System.Windows.Control.

Типы элементов управления:

1. Элементы управления содержимым. Эти элементы могут содержать вложенные элементы, что дает им практически неограниченные визуальные возможности. К ним относятся классы Label, Button и ToolTip.

2. Элементы управления содержимым с заголовками. Эти элементы позволяют добавлять главный раздел содержимого и отдельную заглавную часть. Обычно они используются для упаковки больших блоков пользовательского интерфейса. К ним относятся классы Tabltem, GroupBox и Expander.

3. Элементы управления текстом. Этот небольшой набор элементов позволяет пользователям вводить текст. Текстовые элементы поддерживают ввод обычного текста (TextBox), паролей (PasswordBox) и форматированного текста (RichTextBox).

4. Элементы управления списком. Эти элементы отображают коллекцию элементов в виде списка. К ним относятся классы ListBox и ComboBox.

5. Элементы выбора из диапазона. У всех этих элементов есть общая черта: свойство Value, которое может принимать любое значение из заранее указанного диапазона. Примерами могут служить классы Slider и ProgressBar.

6. Элементы управления датами. В эту категорию входят два класса, позволяющие выбрать дату: Calendar и DatePicker.

Существуют также элементы для создания меню, инструментальных панелей и лент; элементы для вывода визуальных таблиц и древовидного отображения данных; и элементы для просмотра и редактирования форматированных документов.

В окнах WPF много разных элементов, но лишь некоторые из них являются элементами управления.

В мире WPF элемент управления обычно описывается как элемент для интерактивной связи с пользователем – т.е. он может принимать фокус и получать входные данные от клавиатуры или мыши. Очевидными примерами таких элементов являются текстовые поля и кнопки. Однако различие не всегда бывает четким. Всплывающая подсказка считается элементом управления, т.к. она появляется и пропадает в зависимости от движений мыши. Метка считается элементом управления из-за ее поддержки мнемоники (нажатия клавиш, которые передают фокус соответствующим элементам).

Все элементы управления происходят от класса System.Windows.Control, который наделяет их базовыми характеристиками:

– они позволяют определять расположение содержимого внутри элемента управления;

– они позволяют определять порядок передачи фокуса при использовании клавиши табуляции;

– они позволяют отображать фон, передний план и рамку;

– они позволяют форматировать размер и шрифт текстового содержимого.

18.3.1.1 Цвет элементов управления

Все элементы управления имеют фон (background) и передний план (foreground). Как правило, фоном является поверхность элемента управления (например, белая или серая область внутри кнопки), а передним планом – текст. В WPF цвет (но не содержимое) этих двух областей определяется с помощью свойств Background и Foreground соответственно.

Естественно ожидать, что свойства Background и Foreground должны использовать объекты цвета, как в приложениях на основе Windows Forms. Однако на самом деле эти свойства используют более универсальный объект – Brush (кисть) или чем-то экзотическим (например, используя кисти LinearGradientBrush или TileBrush).

Указание цвета в коде C#:

```
cmd.Background = new SolidColorBrush(Colors.AliceBlue);
```

Можно также пользоваться системными цветами (они могут выбираться исходя из предпочтений пользователя) из перечисления System.Windows.SystemColors.

Например:

```
cmd.Background = new SolidColorBrush(SystemColors.ControlColor);
cmd.Background = SystemColors.ControlBrush;
```

Классы Colors и SystemColors предлагают удобные сокращения, однако это не единственный способ задать цвет. Вы можете, например, создать объект Color, указав значения R, G и B (красной, зеленой и синей составляющих). Каждое из этих значений является числом из диапазона 0-255:

```
int red = 0; int green = 255, int blue = 0;
cmd.Foreground = new SolidColorBrush(Color.FromRgb(red, green, blue));
```

Можно также сделать цвет частично прозрачным, используя значение альфа-канала и вызвав метод Color.FromArgb(). Значение альфа-канала, равное 255, соответствует полной непрозрачности, а значение 0 – полной прозрачности.

При задании цвета фона или переднего плана в XAML можно воспользоваться удобным сокращением. Вместо определения объекта Brush можно указать наименование или значение цвета. Компилятор WPF автоматически создаст объект SolidColorBrush с выбранным цветом и будет применять этот объект для фона или переднего плана:

```
<Button Background="Red">A Button</Button>
```

Он эквивалентен следующему многострочному фрагменту:

```
<Button>A Button
<Button.Background>
<SolidColorBrush Color="Red" />
</Button.Background>
</Button>
```

Можно использовать другую форму задания цвета:

```
<Button Background="#FFFF0000">A Button</Button>
```

Здесь значением альфа-канала является FF (255).

18.3.1.2 Шрифты

Класс Control определяет небольшой набор свойств, связанных со шрифтами.

FontFamily	Имя шрифта
FontSize	Размер шрифта в не зависящих от устройства единицах (по 1/96 дюйма). Это небольшое отклонение от традиции предназначено для поддержки новой модели прорисовки в WPF, которая не зависит от разрешения Обычные Windows-приложения измеряют шрифты с помощью пунктов (point), которые равны 1/72 дюйма на стандартном мониторе для ПК. Если вам нужно преобразовать размер шрифта WPF в более

	знакомый размер в пунктах, просто умножьте его на 3/4. Например, традиционные 38 пунктов эквивалентны 48 единицам в WPF.
FontStyle	Наклон текста, представленный объектом FontStyle. Возможные значения свойства FontStyle содержатся в статических свойствах класса FontStyles, который включает написание символов Normal, Italic или Oblique. (Oblique – это искусственный способ создания курсивного текста на компьютере, в котором нет необходимого курсивного шрифта. Буквы берутся из обычного шрифта и скашиваются с помощью специального преобразования. Как правило, результат получается неважным.)
FontWeight	Плотность текста, представленная объектом FontWeight. Вначале свойство FontWeight устанавливается из статических свойств класса FontWeight. Наиболее популярен вариант Bold (жирный), хотя в некоторых гарнитурах имеются и другие, такие как Heavy, Light, ExtraBold и т.д.
FontStretch	Коэффициент растяжения или сжатия текста, представленный объектом FontStretch. Возможные значения свойства FontStretch содержатся в статических свойствах класса FontStretches. Например, UltraCondensed сжимает текст до 50% от обычной ширины, а UltraExpanded растягивает его до 200%. Растяжение шрифта является особенностью OpepType, которая не поддерживается многими гарнитурами. (Чтобы поэкспериментировать с этим свойством, попробуйте шрифт Rockwell, который поддерживает его.)

Задание шрифтов:

<Button Name="cmd" FontFamily="Times New Roman" FontSize="12">

```
A Button
</Button>
cmd.FontFamily = "Times New Roman";
cmd.FontSize = "18";
<Button FontFamily="Technical Italic, Comic Sans MS, Arial">
A Button
</Button>
```

18.3.1.3 Курсор мыши

В любом приложении обычно требуется изменять курсор мыши, чтобы он показывал, когда приложение занято, или отражал работу разных элементов управления.

Курсор мыши можно задать для любого элемента с помощью свойства Cursor, унаследованного от класса FrameworkElement. Каждый курсор представляется объектом System.Windows.Input.Cursor. Получить объект Cursor проще всего с помощью статических свойств класса Cursors (из пространства имен System.Windows.Input). Они включают все стандартные указатели Windows, такие как песочные часы, рука, стрелки изменения размеров и т.д.:

```
this.Cursor = Cursors.Wait;
<Button Cursor="Help">Help</Button>
```

Родительский элемент может перекрыть параметры курсора своих потомков с помощью свойства ForceCursor. Если требуется применить параметры курсора к каждому элементу в каждом окне приложения, то свойство FrameworkElement.Cursor бесполезно. Вместо него вам следует использовать статическое свойство Mouse.OverrideCursor,

18.3.2 Элементы управления содержимым

Элемент управления содержимым – еще более специализированный вид элементов управления, который может хранить (и отображать) фрагмент содержимого. Технически элементы управления содержимым представляют собой

элементы, которые могут содержать один вложенный элемент. Это ограничение на наличие лишь одного потомка и отличает элементы управления содержимым от контейнеров компоновки, которые могут содержать сколько угодно вложенных элементов.

Иерархия элементов WPF:

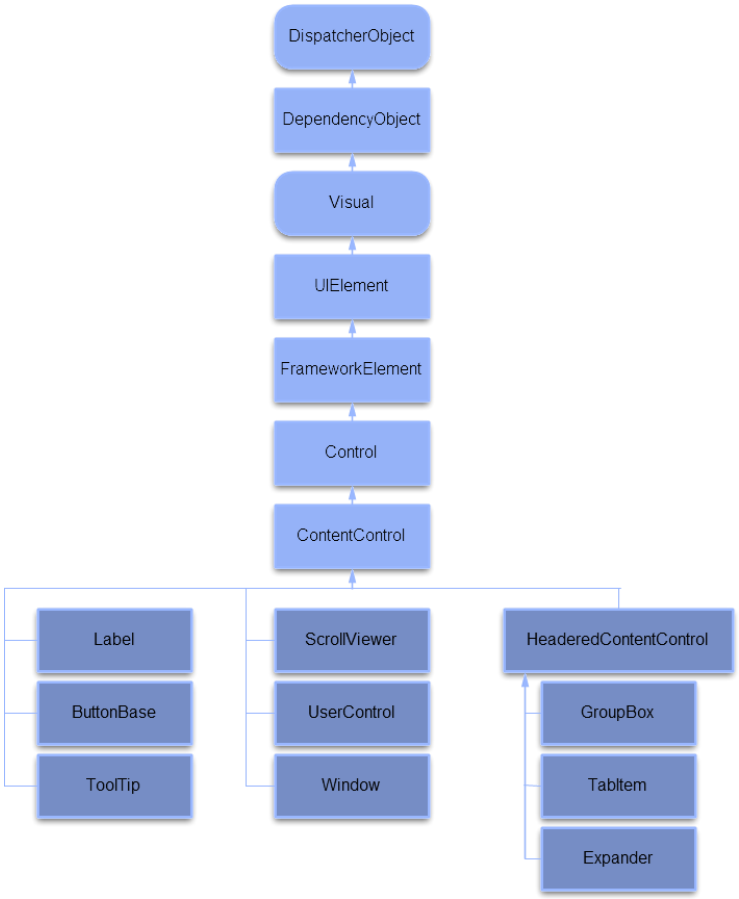


Рисунок 18.1 – Иерархия классов элементов управления.

Класс ContentControl добавляет свойство Content, которое принимает единственный объект. Свойство Content поддерживает объекты любых типов, но разделяет их на две группы и обрабатывает эти группы по-разному:

1. Объекты, не порожденные от UIElement. Элементы управления содержимым получают текст из таких элементов с помощью метода ToString(), а затем выводят этот текст.

2. Объекты, порожденные от UIElement. Эти объекты (к которым относятся все визуальные элементы, входящие в WPF) выводятся внутри элемента управления содержимым с помощью метода UIElement.OnRender().

18.3.2.1 Метки

Простейшим элементом управления содержимым является Label – метка. Отличительной чертой элемента Label является его поддержка мнемонических команд – нажатий клавиш, которые передают фокус соответствующему элементу управления.

Для поддержки этой функции в элементе Label добавлено свойство Target. Чтобы задать это свойство, необходимо воспользоваться выражением привязки, которое указывает на другой элемент управления. Ниже показан необходимый для этого синтаксис:

```
<Label Target="{Binding ElementName=txtA}">Choose _A</Label>
<TextBox Name=MtxtA"></TextBox>
<Label Target="{Binding ElementName=txtB}">Choose _B</Label>
<TextBox Name="txtBM></TextBox>
```

Если нужно лишь вывести содержимое без поддержки мнемонических команд, можно воспользоваться более облегченным элементом TextBlock. В отличие от элемента Label, TextBlock поддерживает перенос текста с помощью свойства TextWrapping.

18.3.2.2 Кнопки

WPF распознает три типа кнопок: Button, CheckBox и RadioButton. Все эти кнопки являются наследниками класса ButtonBase.

Все кнопки поддерживают клавиши доступа, которые работают подобно мнемоническим командам в элементе управления Label. Для обозначения клавиши доступа служит символ подчеркивания. Когда пользователь нажмет клавишу <Alt> и клавишу доступа, возникнет событие Click данной кнопки.

Класс Button представляет кнопку Windows. Он добавляет всего два доступных для записи свойства: IsCancel и IsDefault.

Существует элемент RepeatButton, который в прижатом состоянии непрерывно генерирует события Click. Обычные кнопки генерируют событие Click только при полном щелчке на кнопке.

ToggleButton – кнопка с двумя состояниями (нажата и отпущена). Если щелкнуть на кнопке ToggleButton, она будет оставаться нажатой до тех пор, пока вы не щелкнете на ней снова. Иногда такое поведение называют залипающим щелчком (sticky click).

Классы RepeatButton и ToggleButton определены в пространстве имен System.Windows.Controls.Primitives, что означает, что сами по себе они применяются редко. ToggleButton применяется для порождения более полезных классов Checkbox и RadioButton.

Класс CheckBox не добавляет никаких членов, поэтому базовый интерфейс CheckBox определяется в классе ToggleButton. Более важно то, что ToggleButton добавляет свойство IsChecked. Свойство IsChecked является расширенным логическим, т.е. оно может принимать значения true, false или null. Понятно, что true представляет установленный флажок, а false – сброшенный. Значение null используется для представления неопределенного состояния, которое отображается в виде серого квадратика.

Неопределенное состояние обычно служит для представления не заданных значений или областей, в которых возможны противоречия. Например, если имеется флажок, который позволяет применять жирный шрифт в текстовом приложении, а выбранный фрагмент содержит как жирный, так и обычный текст, можно присвоить флажку значение null, чтобы обозначить неопределенное состояние. Чтобы присвоить значение null в разметке WPF, нужно использовать расширение разметки Null:

```
<CheckBox IsChecked="{x:Null}">
        A check box in indeterminate state
</CheckBox>
```

Наряду со свойством IsChecked класс ToggleButton добавляет свойство IsThreeState, которое определяет, может ли пользователь установить флажок в неопределенное состояние.

Класс RadioButton также порожден от класса ToggleButton и использует то же свойство IsChecked и те же события Checked, Unchecked и Indeterminate. Кроме того, RadioButton добавляет еще одно свойство GroupName, которое позволяет управлять группировкой переключателей.

WPF предлагает гибкую модель для всплывающих подсказок (tooltip – желтые окошки, которые появляются при наведении указателя мыши на какой-то объект). Поскольку в WPF всплывающие подсказки относятся к элементам управления содержимым, в них можно поместить практически что угодно. Можно также настроить различные временные параметры, чтобы задать время, после которого подсказка появляется и исчезает.

Самый простой способ вывода всплывающих подсказок – отнюдь не непосредственное использование класса ToolTip. Вместо этого достаточно просто определить свойство ToolTip нужного элемента. Свойство ToolTip определено в классе FramworkElement, поэтому оно доступно для любого элемента, которое может разместиться в окне WPF.

Например, вот кнопка с простой всплывающей подсказкой:

```
<Button ToolTip="This is my tooltip">
        I have a tooltip
</Button>
```

Пример более сложной подсказки:

```
<Button>
        <Button.ToolTip>
                <StackPanel>
                        <TextBlock Margin="3">Hello</TextBlock>
                        <Image Source="face.ipg" Stretch="None" />
                        <TextBlock Margin="3">Image and text</TextBlock>
                </StackPanel>
```

```
        </Button.ToolTip>
        <Button.Content>I have a fancy tooltip</Button.Content>
    </Button>
```

Нельзя помещать во всплывающую подсказку элементы управления, поддерживающие интерактивную связь с пользователями, т.к. окно ToolTip не может получить фокус. Например, если поместить в элемент ToolTip кнопку, эта кнопка будет отображаться, но не будет реагировать на щелчки.

Свойства подсказок:

IsEnabledM IsOpen – позволяют управлять поведением подсказки с помощью кода. Свойство IsEnabled позволяет временно отключить всплывающую подсказку, а IsOpen – программно отображать и скрывать подсказку (или просто проверять, открыто ли ее окно).

HasDropShadow – определяет, имеет ли окно подсказки размытую темную тень, которая "приподнимает" его над находящимся под ним окном.

Существуют свойства всплывающих подсказок, которые нельзя задать с помощью свойств класса ToolTip. Для этого предназначен другой класс – ToolTipService. Он позволяет задать длительность задержек при отображении всплывающей подсказки. Все свойства этого класса являются прикрепленными свойствами, поэтому их можно указывать прямо в дескрипторе элемента управления:

```
<Button ToolTipService.InitialShowDelay=Hl">
    …
</Button>
```

Свойства класса ToolTipService:

InitialShowDelay – задает задержку (в миллисекундах) перед выводом подсказки после наведения указателя мыши на элемент.

ShowDuration – задает время (в миллисекундах), в течение которого будет отображаться подсказка, если пользователь не сдвинет указатель мыши.

Существует также элемент Popup, который в отличие от ToolTip способен принимать фокус.

18.3.3 Модель событий WPF

Программист, работающий в .NET, знаком с понятием события: это сообщение, которое посылается объектом (например, элементом WPF) для уведомления кода о том, что произошло что-то важное. WPF дополняет модель событий .NET новой концепцией маршрутизации событий.

Маршрутизация позволяет событию возникать в одном элементе, а генерироваться в другом: например, щелчок на кнопке панели инструментов генерируется в панели инструментов, а затем в содержащем эту панель окне, и только тогда передается на обработку коду.

Маршрутизация событий дает возможность писать лаконичный и простой код, который может обрабатывать события в наиболее удобном для этого месте. Она необходима также для работы с моделью содержимого WPF, позволяющей создавать простые элементы (например, кнопки) из десятков отдельных ингредиентов, каждый из которых имеет свой собственный набор событий.

Создать обработчик события (на примере кнопки) можно несколькими способами:

1. Добавляеть соответствующий атрибут события в разметку XAML (рис. 18.2).

```
<Window x:Class="WpfExample.EventExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="EventExample" Height="300" Width="300">
    <Grid>
        <Button Width="150" Height="150">
            <Border BorderThickness="2" BorderBrush="Black">
                <Image Source="Images/1.png" Stretch="Fill" Width="100" Height="100"
                   MouseLeave="Image_OnMouseLeave"
                   MouseEnter="Image_OnMouseEnter"/>
            </Border>
        </Button>
    </Grid>
</Window>
```

Рисунок 18.2 – Добавление обработчика в разметке XAML.

В данном случае добавлены два обработчика: для события попадания мыши в область рисунка, и для события покидания указателем мыши пределов рисунка.

2. Событие можно соединить и с кодом. Вот эквивалент приведенного на рис. 18.2 кода разметки XAML:

```
20  public partial class EventExample : Window
21  {
        0 references
22      public EventExample()
23      {
24          InitializeComponent();
25
26          Image.MouseEnter += Image_OnMouseEnter;
27          Image.MouseLeave += Image_OnMouseLeave;
28      }
```

Рисунок 18.3 – Добавление обработчика в коде C#

Если необходимо отсоединить обработчик события, то сделать это можно только в коде:

```
40  private void Button_OnClick(object sender, RoutedEventArgs e)
41  {
42      Image.MouseEnter -= Image_OnMouseEnter;
43      Image.MouseLeave -= Image_OnMouseLeave;
44  }
```

Рисунок 18.4 – Отключение обработчика в коде C#.

18.3.4 Маршрутизируемые события

Многие элементы управления в WPF являются элементами управления содержимым, которые могут иметь разный тип и разный объем вложенного содержимого.

Например, на рис. 18.5 показан подобный элемент:

```
<Window x:Class="WpfExample.RoutedEventEx"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="RoutedEventEx" Height="300" Width="300">
    <Label HorizontalAlignment="Center">
        <StackPanel VerticalAlignment="Center">
            <TextBlock Margin="10">
                Пример маршрутизируемого события
            </TextBlock>
            <Button Width="150">
                <Button.Content>
                    <StackPanel>
                        <Image Source="Images/1.png" Width="100"/>
                        <Border>
                            <TextBlock FontWeight="ExtraBold" FontSize="14">
                                Кнопка с яблоком
                            </TextBlock>
                        </Border>
                    </StackPanel>
                </Button.Content>
            </Button>
        </StackPanel>
    </Label>
</Window>
```

Рисунок 18.5 – Вложенность элементов управления WPF

На рис. 18.6 показано дерево элементов WPF.

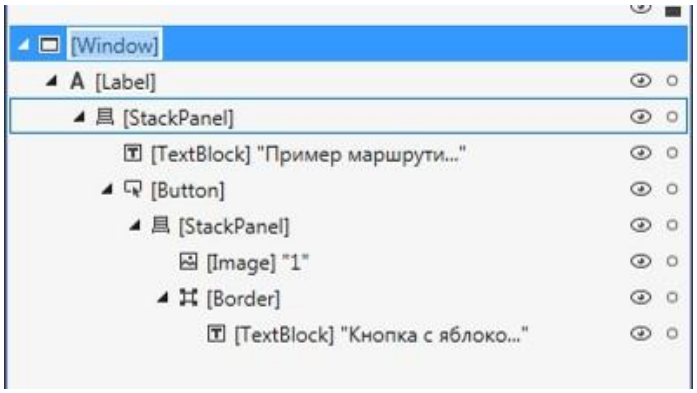


Рисунок 18.6 – Вложенность элементов управления WPF

На рис. 18.7 представлен результат такой компановки.

Рисунок 18.7 – Результирующее окно WPF

Каждый компонент, помещаемый в окно WPF, так или иначе является наследником класса UIElement, включая Label, StackPanel, TextBlock и Image. Класс UIElement определяет несколько ключевых событий. Например, каждый класс, являющийся потомком UIElement, обеспечивает события MouseUp и MouseDown.

А теперь подумайте, что произойдет при щелчке на изображении в такой метке. Понятно, что при этом возникнут события Image.MouseDown и Image.MouseUp. А если вам нужно обрабатывать все щелчки на метке одинаковым образом? То есть неважно, где щелкнул пользователь: на изображении, на тексте или на пустом месте в области метки. В любом из этих случаев нужно реагировать на щелчок с помощью одного и того же кода.

Понятно, что к событиям MouseDown и MouseUp каждого элемента можно привязать один и тот же обработчик, однако это может загромоздить код и усложнить сопровождение разметки. WPF предлагает более удобное решение с помощью модели маршрутизируемых событий.

Маршрутизируемые события бывают трех видов:

1. Прямые (direct) события подобны обычным событиям .NET. Они возникают в одном элементе и не передаются в другой. Например, прямым является событие MouseEnter, которое возникает, когда указатель мыши наводится на элемент.

2. Пузырьковые (bubbling) события поднимаются по иерархии содержания. Например, пузырьковым событием является MouseDown. Оно возникает в элементе, на котором был произведен щелчок, потом передается от этого элемента к родителю, затем к родителю этого родителя, и т.д., пока WPF не достигнет вершины дерева элементов.

3. Туннелируемые (tunneling) события опускаются по иерархии содержания. Они позволяют предварительно просматривать (и, возможно, останавливать) событие, прежде чем оно дойдет до подходящего элемента управления. Например,

PreviewKeyDown позволяет перехватить нажатие клавиши, сначала на уровне окна, а затем в более специфических контейнерах, вплоть до элемента, содержавшего фокус в момент нажатия клавиши.

Поскольку события MouseUp и MouseDown являются пузырьковыми событиями, можно определить, что произойдет в примере с составной меткой. При щелчке на изображении с яблоком (рис. 4.7) событие MouseDown возникнет в следующем порядке:

1. Image.MouseDown.
2. StackPanel.MouseDown
3. Button.MouseDown.
4. StackPanel.MouseDown.
5. Label.MouseDown.

После того как событие MouseDown возникнет в метке, оно передается следующему элементу управления (в данном случае это окно Window). Окно находится на самом верху иерархии содержания и в самом конце в последовательности пузырькового распространения события. Здесь последний шанс обработать пузырьковое событие наподобие MouseDown. Если пользователь отпускает кнопку мыши, в такой же последовательности возникает событие MouseUp.

Пузырьковые события не обязательно обрабатывать в одном месте: например, ничто не мешает обрабатывать события MouseDown и MouseUp на каждом уровне. Однако, как правило, для каждой задачи выбирается наиболее подходящая маршрутизация событий.

Класс RoutedEventArgs. Объект данного класса передается в качестве параметра обработчика события. При обработке пузырькового события параметр отправителя содержит ссылку на последнее звено в цепочке. Например, если событие перед обработкой всплывает от изображения до метки, то параметр отправителя будет ссылаться на объект метки.

В некоторых случаях требуется знать, где первоначально произошло событие. Эту информацию, а также другие подробности, можно получить из свойств класса RoutedEventArgs (таблица 18.1).

Таблица 18.1 – Свойства класса RoutedEventArgs.

Имя свойства	Описание
Source	Указывает, какой объект сгенерировал событие. Если речь идет о событии клавиатуры, то это элемент управления, имевший фокус ввода в момент возникновения события (например, когда была нажата клавиша). В случае события мыши это самый верхний элемент под указателем мыши в момент возникновения события (например, когда был произведен щелчок кнопкой мыши).
OriginalSource	Указывает, какой объект первоначально сгенерировал событие. Как правило, совпадает с Source. Однако в некоторых случаях OriginalSource спускается глубже по дереву объектов, чтобы дойти до внутреннего элемента, являющегося частью элемента более высокого уровня. Например, если вы щелкнете кнопкой мыши близко к границе окна, то получите объект Window в качестве источника события и Border в качестве первоначального источника. Это объясняется тем, что Window состоит из отдельных меньших элементов.
RoutedEvent	Предоставляет объект RoutedEvent для события, сгенерированного вашим обработчиком события (например, статический объект UIElement.MouseUpEvent). Эта информация бывает полезна при обработке разных событий одним и тем же обработчиком.
Handled	Позволяет остановить процесс пузырькового распространения или туннельного распространения события. Если элемента управления заносит в свойство Handled значение true, событие прекращает продвижение и не будет возникать в любых других элементах.

Туннелируемые события работают так же, как и пузырьковые, но в обратном направлении. Например, если бы событие MouseUp было туннельным (а это не так), то при щелчке на изображении в примере с меткой событие MouseUp возникло бы сначала в окне, затем в элементе Label, затем в StackPanel и так далее до достижения источника Image, т.е. изображения кнопки.

Туннелируемые события легко распознать: они начинаются со слова Preview. Более того, WPF обычно определяет события попарно. Это означает, что если имеется пузырьковое событие MouseUp, то, скорее всего, существует и туннелируемое событие PreviewMouseUp. Туннелируемые событие всегда возникает перед пузырьковым событием, как показано на рис. 18.8.

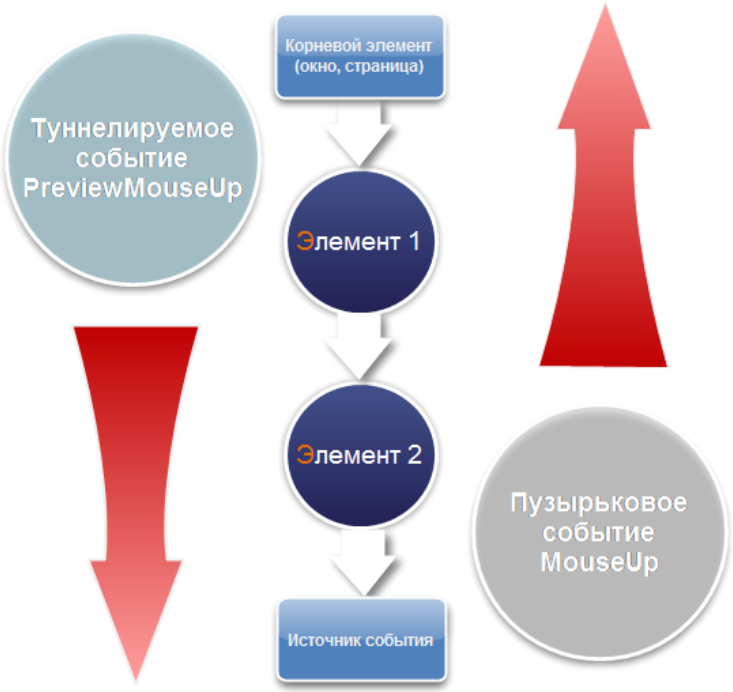


Рисунок 18.8 – Механизм пузырьковых и туннелируемых событий WPF.

18.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

18.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление

программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

18.6. Методика и порядок выполнения работы

Для закрепления материала о туннелируемых и пузырьковых событиях реализуйте приложение для вывода информации о распространении события:

1. Модифицируйте код XAML, представленный на рис. 18.5 следующим образом:

```
<Window x:Class="WpfExample.RoutedEventEx"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="RoutedEventEx" Height="300" Width="300"
        MouseUp="Somewhere_OnMouseUp"
        PreviewMouseUp="Somewhere_OnMouseUp">
    <Label HorizontalAlignment="Center" MouseUp="Somewhere_OnMouseUp"
           PreviewMouseUp="Somewhere_OnMouseUp">
        <StackPanel VerticalAlignment="Center" MouseUp="Somewhere_OnMouseUp"
                    PreviewMouseUp="Somewhere_OnMouseUp">
            <TextBlock Margin="10">
                Пример маршрутизируемого события
            </TextBlock>
            <Button Width="150" MouseUp="Somewhere_OnMouseUp"
                    PreviewMouseUp="Somewhere_OnMouseUp">
                <Button.Content>
                    <StackPanel MouseUp="Somewhere_OnMouseUp"
                                PreviewMouseUp="Somewhere_OnMouseUp">
                        <Image Source="Images/1.png" Width="100"
                               MouseUp="Somewhere_OnMouseUp"
                               PreviewMouseUp="Somewhere_OnMouseUp"/>
                        <Border>
                            <TextBlock FontWeight="ExtraBold" FontSize="14">
                                Кнопка с яблоком
                            </TextBlock>
                        </Border>
                    </StackPanel>
                </Button.Content>
            </Button>
        </StackPanel>
    </Label>
</Window>
```

Рисунок 18.9 – Проверка механизма событий WPF

В данном коде для каждого элемента иерархии добавлен один и тот же обработчик для двух событий: MouseUp и PreviewMouseUp.

2. В класс RoutedEventEx (главный класс окна, производный от Window) добавьте закрытую переменную:

```
public partial class RoutedEventEx : Window
{
    private int count = 1;
```

Рисунок 18.10 – Добавление переменной в класс.

3. Определите обработчик события Somewhere_OnMouseUp:

```
29          private void Somewhere_OnMouseUp(object sender, MouseButtonEventArgs e)
30          {
31              String mes = "№" + count + ": " + "\r\n" +
32                  "Sender: " + sender.ToString() + ": " + "\r\n" +
33                  "Source: " + e.Source + ": " + "\r\n" +
34                  "Original Source: " + e.OriginalSource + ": " + "\r\n" +
35                  "Event: " + e.RoutedEvent + ": " + "\r\n";
36              MessageBox.Show(mes);
37              count++;
38          }
```

Рисунок 18.11 – Обработчик событий MouseUp и PreviewMouseUp.

Запустите приложение, щелкните на кнопке и проследите распространение события.

4. Замените определение кнопки Button на Label. Снова запустите приложение и отследите обработку событий. Теперь отработает полная цепочка событий. Такое поведение обусловлено тем, что событий Click кнопки перекрывало обработку события MouseUp.

5. Теперь удалите событие PreviewMouseUp и рассмотрите распространение события и его тип.

6. Выполните индивидуальное задание

Варианты индивидуальных заданий

Самостоятельно спроектируйте структуру главного окна приложения. Структура должна содержать элементы с уровнем вложенности не менее десяти. Затем выберите событие (туннельное или пузырьковое) и выполните его обработку. В отчете по лабораторной работе добавьте протокол регистрации событий, который должен содержать информацию, представленную в примере (рис. 18.11).

18.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю.

18.8. Вопросы для защиты работы

1. Что такое событие?
2. Что такое контейнер компоновки? Для чего они используются? Опишите свойства класса Panel.
3. Какие типы событий WPF вы знаете? Дайте определение каждого типа WPF
4. Какими способами можно подключить событие? Как можно отсоединить обработчик события от элемента управления?
5. Какие элементы управления вы знаете?
6. Что означают понятия «элемент управления содержимым», «контент», «панель», «контейнер компоновки»?
7. Какие элементы управления содержимым вы знаете? Поясните назначение и отличительные особенности каждого из них.
8. Поясните принципы работы с классом ToggleButton. Как используется данный класс?

18.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [3].

ЛАБОРАТОРНАЯ РАБОТА 19. 2D-ГРАФИКА В ПРИЛОЖЕНИЯХ WPF

19.1. Цель и содержание

Цель лабораторной работы: научиться использовать классы библиотеки для работы с кистями, трансформациями и фигурами.

Задачами лабораторной работы являются:

– изучить фигуры WPF,

– научиться использовать кисти WPF;

– изучить механизм задания трансформаций WPF.

19.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

19.3. Теоретическое обоснование

19.3.1 Фигуры

Классы фигур наследуются непосредственно от FrameworkElement и используются при необходимости рисования двумерных графических примитивов.

Фигуры являются полноценными элементами WPF, что влечет следующие особенности их применения:

– фигуры рисуют сами себя;

– фигуры организованы таким же образом, как и другие элементы;

– фигуры поддерживают те же события, что и другие элементы.

На рис. 19.1 представлены основные классы фигур:

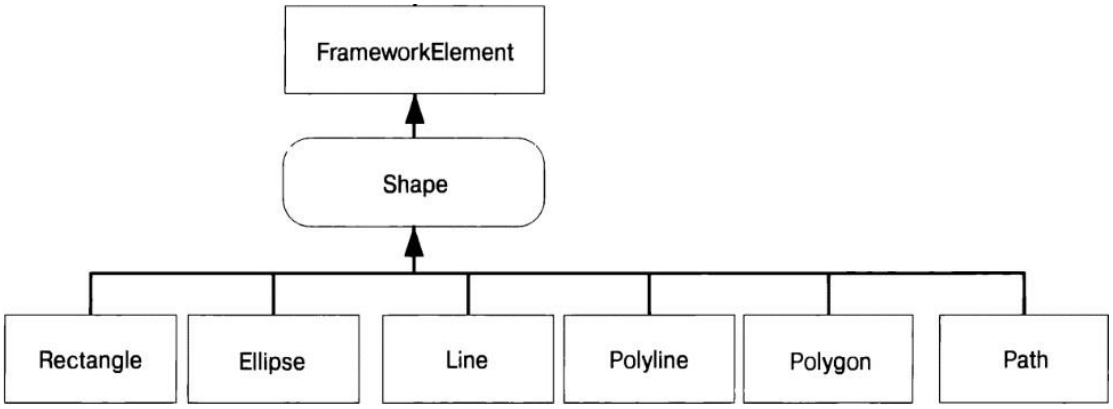


Рисунок 19.1 – Фрагмент иерархии классов с фигурами WPF

Класс Shape обеспечивает все фигуры следующими свойствами:

Таблица 19.1 – Свойства фигур и их описания

Свойство	Описание
Fill	Устанавливает объект кисти, рисующей поверхность фигуры (все, что расположено в ее границах)
Stroke	Устанавливает объект кисти, рисующей границу фигуры
StrokeThickness	Устанавливает толщину границы в единицах, независимых от устройства.
StrokeStartLineCap и StrokeEndLineCap	Определяют контур краев в начале и конце линии. Эти свойства имеют эффект только для фигур Line, Polyline и (иногда) Path. Все прочие фигуры замкнуты, а потому начальной и конечной точки не имеют
StrokeDashArray, StrokeDashOffset и StrokeDashCap	Позволяют создавать заштрихованную границу вокруг фигуры. Можно управлять размером и частотой штриховки, а также контуром, ограничивающим начало и конец каждой штриховой линии
StrokeLineJoin и StrokeMiterLimit	Определяют контур углов фигуры. Технически эти свойства затрагивают вершины, где стыкуются разные линии, такие как углы Rectangle. Эти свойства не имеют эффекта для фигур без углов, подобных Line и Ellipse
Stretch	Определяет способ заполнения фигурой доступного пространства.
GeometryTransform	Позволяет применять объект Transform, который изменяет координатную систему, используемую для рисования фигуры. Это дает возможность искажать, вращать или перемещать фигуру.

Значения для свойства Stretch представлены в таблице 19.2.

Таблица 19.2 – Значения перечисления Stretch

Свойство	Описание
Fill	Фигура растягивается по ширине и высоте для полного заполнения контейнера.
None	Фигура не растягивается.
Uniform	Ширина и высота увеличиваются пропорционально, пока фигура не достигнет границ контейнера.
UniformToFill	Ширина и высота устанавливаются пропорционально, пока фигура не заполнит всю доступную высоту и ширину.

19.3.2 Кисти

Кисти заполняют области – будь то фон, передний план или граница элемента, или штрих фигуры. Простейшим типом кисти является SolidColorBrush, которая рисует сплошным цветом.

Доступные в WPF классы кистей представлены в таблице 19.3.

Таблица 19.3 – Классы кистей

Класс	Описание
SolidColorBrush	Рисует область с использованием одного сплошного цвета
LinearGradientBrush	Рисует область с использованием линейного градиентного заполнения
RadialGradientBrush	Рисует область с использованием радиального градиентного заполнения
ImageBrush	Рисует область с использованием графического изображения, которое может растягиваться, масштабироваться или многократно повторяться
DrawingBrush	Рисует область с использованием объект Drawing. Этот объект может включать определенные программистом фигуры и растровые изображения
VisualBrush	Рисует область с использованием объекта Visual. Поскольку все элементы WPF наследуются от класса Visual, эту кисть можно применять для копирования части пользовательского интерфейса (такого как поверхность кнопки) в другую область.
BitmapCacheBrush	Рисует область с использованием кэшированного содержимого из объекта Visual. Это делает его похожим на VisualBrush, но этот класс более эффективен, если графическое содержимое должно

	повторно использоваться во множестве мест либо часто перерисовываться

Класс SolidColorBrush является достаточно примитивным, рассмотрим использование классов LinearGradientBrush и ImageBrush. На рис. 19.2 представлен исходный код, демонстрирующий определение кисти с линейным градиентом и кисти с изображениями.

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <sys:Double x:Key="FontSize">60</sys:Double>
        <LinearGradientBrush x:Key="LinearGrBrush">
            <GradientStop Offset="0" Color="Red"/>
            <GradientStop Offset="0.3" Color="Green"/>
            <GradientStop Offset="0.6" Color="Blue"/>
            <GradientStop Offset="0.9" Color="Red"/>
        </LinearGradientBrush>
        <ImageBrush x:Key="BrushWithTitle" ImageSource="Images/example1.png"
                    TileMode="Tile" Stretch="Uniform"
                    Viewport="0,0 0.2, 0.3"
                    ViewportUnits="RelativeToBoundingBox" />
    </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions...>
        <Grid.RowDefinitions...>
        <TextBlock HorizontalAlignment="Center"
                   FontSize="{StaticResource FontSize}"
                   VerticalAlignment="Center"
                   FontWeight="Bold" TextWrapping="WrapWithOverflow"
                   Foreground="{StaticResource LinearGrBrush}"
                   >
            Colored Text!
        </TextBlock>
        <Ellipse Grid.Column="1"
                 Fill="{StaticResource BrushWithTitle}"
                 Stroke="Black"
                 StrokeThickness="2"/>
        <Rectangle Grid.Column="0" Grid.Row="1" Margin="5"
                   Fill="{StaticResource BrushWithTitle}"/>
        <Polyline Grid.Column="1" Grid.Row="1"
            Points="0,0 30,100 60,30 90,150 110,80 140,120 200,0"
                  Fill="{StaticResource LinearGrBrush}"/>
    </Grid>
</Window>
```

Рисунок 19.2 – Разметка с использованием кистей

Для создания этого градиента необходимо добавить по одному объекту GradientStop для каждого цвета. Также нужно поместить каждый цвет в градиент, используя значение Offset от 0 до 1.

Результат работы подобной кисти представлен на рис. 19.3.

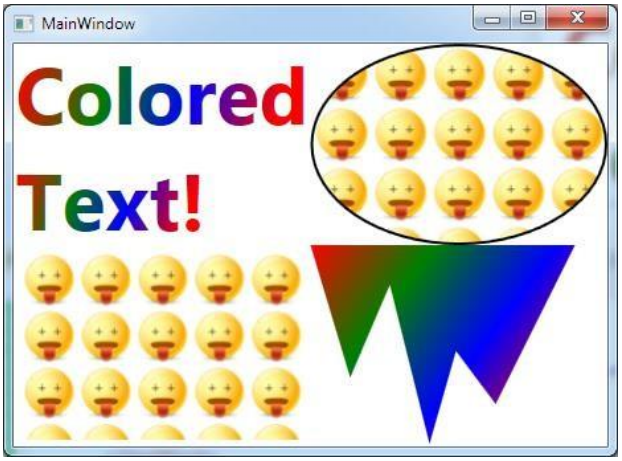


Рисунок 19.3 – Приложение с использованием различных видов кистей

19.3.3 Трансформации

Огромный объем задач, связанных с рисованием, может быть упрощен благодаря применению трансформации (transform) – объекта, изменяющего способ рисования фигуры или элемента посредством скрытого сдвига используемой им координатной системы.

В таблице 19.4 представлены основные классы трансформаций и их свойства.

Таблица 19.4 – Классы трансформаций

Класс	Описание	Определяющие свойства
TranslateTransform	Смещает координатную систему на определенную величину. Эта трансформация удобна, когда нужно нарисовать ту же самую фигуру в разных местах	X, Y
RotateTransform	Поворачивает координатную систему. Нормально нарисованные фигуры поворачиваются вокруг заданной точки	Angle, CenterX, CenterY
ScaleTransform	Масштабирует координатную систему в большую и меньшую сторону, так что фигуры становятся больше или меньше.	ScaleX, ScaleY, CenterX, CenterY
SkewTransform	Деформирует координатную систему, наклоняя ее на определенное число градусов.	AngleX, AngleY, CenterX, CenterX

MatrixTransform	Модифицирует координатную систему, используя матричное умножение с указанной матрицей.	Matrix
TransformGroup	Комбинирует несколько трансформаций – таким образом, что они могут применяться одновременно. Порядок применения трансформаций важен – он влияет на конечный результат.	

Для применения трансформации к элементам управления и фигурам необходимо установить трансформацию в качестве значения свойства RenderTransform.

19.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2015 и выше.

19.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

19.6. Методика и порядок выполнения работы

Рассмотрим пример выполнения задания:

Реализуйте приложение с использованием трансформаций, кистей и фигур для схематичного представления фотоизображения на форме приложения.

На рис. 19.4 представлено исходное фото изображения.

Рисунок 19.4 – Исходное изображение

На рис. 19.5 представлена XAML-разметка для построения стилизованного изображения.

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="400" Width="400">
    <Canvas>
        <Rectangle Fill="#92CA72" Width="392" Height="370"></Rectangle>
        <Ellipse Fill="White" Width="392" Height="370"></Ellipse>
    </Canvas>
</Window>
```

Рисунок 19.5 – Код XAML

Результат работы приложения представлен на рис. 19.6.

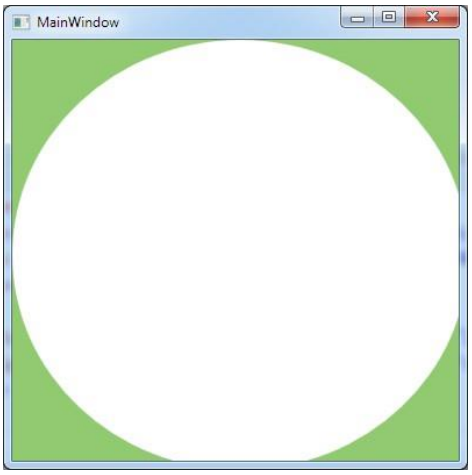


Рисунок 19.5 – Код XAML

На рис. 19.6 представлено начало выполнения задания.

Варианты индивидуальных заданий

Продолжите выполнение задания. Каждый студент должен реализовать свою версию исходного изображения. При выполнении задания необходимо использовать несколько различных классов фигур, различные типы кистей, трансформации. Важно! Необходимо реализовать анимацию изображения (например, в ответ на действие пользователя или непрерывное анимирование). Для построения изображения используйте XAML (естественно, что для реализации анимации необходимо применит код C# и обработчики событий действий пользователя).

19.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо

привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

19.8. Вопросы для защиты работы

1. Для чего предназначены классы фигур?
2. Чем отличаются технологии работы с двумерной графикой в рамках WPF и Windows Forms?
3. Опишите основные классы фигур.
4. От какого класса унаследованы все фигуры? Опишите общие свойства всех фигур.
5. Что такое трансформации? Опишите основные классы для реализации трансформации. Поясните, каким образом применяется трансформация к элементам WPF.
6. Какие классы кистей вы знаете?
7. Поясните основные свойства, которые необходимо задействовать при использовании класса SolidColorBrush?
8. Поясните основные свойства, которые необходимо задействовать при использовании класса LinearGradientBrush?
9. Поясните основные свойства, которые необходимо задействовать при использовании класса ImageBrush?

19.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 20. РЕСУРСЫ, СТИЛИ, ТРИГГЕРЫ

20.1. Цель и содержание

Цель лабораторной работы: научиться использовать инструменты XAML, позволяющие гибко настраивать и управлять внешним видом приложения.

Задачами лабораторной работы являются:

– научиться использовать ресурсы в приложениях WPF,

– научиться использовать стили;

– научится задействовать механизм триггеров.

20.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

20.3. Теоретическое обоснование

20.3.1 Ресурсы

Система ресурсов WPF представляет собой просто способ поддержания вместе набора полезных объектов, таких как наиболее часто используемые кисти, стили или шаблоны, что существенно упрощает работу с ними.

В WPF ресурсы можно определять в коде или в различных местах внутри разметки (вместе с отдельными элементами управления, внутри отдельных окон или во всем приложении).

Ресурсы обладают рядом важных преимуществ:

1. Эффективность. Ресурсы позволяют определять объект один раз и затем использовать его в нескольких местах внутри разметки. Это упрощает код и делает его намного эффективнее.

2. Сопровождаемость. Ресурсы позволяют переносить низкоуровневые детали форматирования (вроде размеров шрифтов) в центральное место, где их легко изменять. Это своего рода XAML-эквивалент создания констант в коде.

3. Адаптируемость. После отделения определенной информации от остальной части приложения и ее помещения в раздел ресурсов появляется возможность ее динамической модификации.

Каждый элемент включает свойство Resources, в котором хранится словарная коллекция ресурсов (представляющая собой экземпляр класса ResourceDictionary). Эта коллекция ресурсов может хранить объект любого типа с индексацией по строке. Хотя каждый элемент имеет свойство Resources (которое определено в классе FrameworkElement), чаще всего ресурсы определяются на уровне окна.

Рассмотрим пример (рис. 19.1) использования ресурсов для хранения данных различного типа.

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <SolidColorBrush x:Key="SolidBrush" Color="Black"/>
        <sys:Double x:Key="FontSize">32</sys:Double>
        <sys:Double x:Key="StrokeSize">5</sys:Double>
        <ImageBrush x:Key="BrushWithoutStretch" ImageSource="Images/example.png"
                    TileMode="Tile" Stretch="None"/>
        <ImageBrush x:Key="BrushWithTitle" ImageSource="Images/example1.png"
                    TileMode="Tile" Stretch="Uniform"
                    Viewport="0,0 0.2, 0.3"
                    ViewportUnits="RelativeToBoundingBox" />
    </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions...>
        <Grid.RowDefinitions...>

        <Button Foreground="{StaticResource SolidBrush}"
                Background="{StaticResource BrushWithoutStretch}"
                FontSize="{StaticResource FontSize}"
                Margin="5">
            Click Me!
        </Button>
        <Ellipse Grid.Column="1"
```

```
34              Fill="{StaticResource BrushWithTitle}"
35              Stroke="{StaticResource SolidBrush}"
36              StrokeThickness="{StaticResource StrokeSize}"/>
37      <Rectangle Grid.Column="0" Grid.Row="1" Margin="5"
38                 Stroke="{DynamicResource OwnBrush}"
39                 StrokeThickness="50">
40          <Rectangle.Resources>
41              <LinearGradientBrush x:Key="OwnBrush">
42                  <GradientStop Color="Red" Offset="0"/>
43                  <GradientStop Color="DeepPink" Offset="0.2"/>
44                  <GradientStop Color="GreenYellow" Offset="0.4"/>
45                  <GradientStop Color="DarkCyan" Offset="0.6"/>
46                  <GradientStop Color="Aqua" Offset="0.8"/>
47                  <GradientStop Color="Black" Offset="1"/>
48              </LinearGradientBrush>
49          </Rectangle.Resources>
50      </Rectangle>
51      <Button Grid.Row="1" Grid.Column="1" Margin="5"
52              Background="{StaticResource BrushWithTitle}"
53              FontSize="{StaticResource FontSize}"
54              BorderBrush="{StaticResource SolidBrush}">
55          Another!
56      </Button>
57  </Grid>
58 </Window>
```

Рисунок 20.1 – Код XAML, использующий ресурсы

Обратите внимание на места определения ресурсов и синтаксические конструкции для их подключения.

Результат подобной разметки представлен на рис. 20.2.

Рисунок 20.2 – Снимок окна приложения для демонстрации использования ресурсов

20.3.2 Стили

Стили (style) представляют собой важный инструмент для организации и повторного использования вариантов форматирования. Вместо того чтобы заполнять XAML-файл повторяющимся кодом разметки для установки деталей

вроде полей, отступов, цветов и шрифтов, можно просто создавать набор охватывающих все эти детали стилей, а затем применять эти стили по мере необходимости, устанавливая единственное свойство.

Стилем называется коллекция значений свойств, которые могут применяться к элементу. Система стилей WPF играет ту же роль, которую играет стандарт каскадных таблиц стилей (Cascading Style Sheet — CSS) в HTML-разметке.

Стили WPF поддерживают триггеры, которые позволяют изменять стиль элемента управления при изменении какого-то свойства.

Рассмотрим пример определения внешнего вида элемента с использованием ресурсов (рис. 20.3).

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="150" Width="400">
    <Window.Resources>
        <sys:Double x:Key="DefaultWidth">150</sys:Double>
        <sys:Double x:Key="DefaultHeght">50</sys:Double>
        <Thickness x:Key="StrokeSize">10</Thickness>
        <FontFamily x:Key="FF">Times New Roman</FontFamily>
        <LinearGradientBrush x:Key="DefaultBrush">
            <GradientStop Color="AliceBlue" Offset="0.2"/>
            <GradientStop Color="Aquamarine" Offset="0.7"/>
        </LinearGradientBrush>
    </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Grid.Column="0" FontFamily="{StaticResource FF}"
                Width="{StaticResource DefaultWidth}"
                Height="{StaticResource DefaultHeght}"
                BorderThickness="{StaticResource StrokeSize}"
                Background="{StaticResource DefaultBrush}">
            Button One
        </Button>
        <Button Grid.Column="1">
            Button Two
        </Button>
    </Grid>
</Window>
```

Рисунок 20.3 – Определение внешнего вида элементов управления с использованием ресурсов

Результат подобной разметки представлен на рис. 20.4.

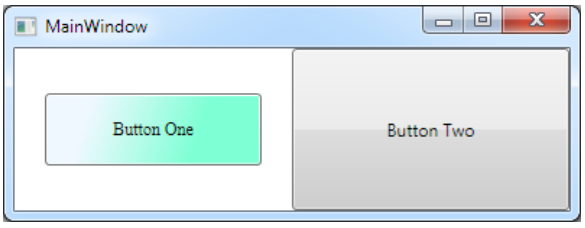


Рисунок 20.4 – Внешний вид элементов управления при использовании ресурсов

Для того, чтобы изменить внешний вид второй кнопки в соответствии с выбранным внешним видом, необходимо установить все свойства, примененные к первой кнопке (то есть повторить строки 21–25 листинга на рис. 20.3 применительно ко второй кнопке). Такой код XAML будет больше, обслуживать его будет сложнее. Еще сильнее усложнится код если в приложении будет множество элементов управления, оформленных в одном визуальном стиле. Для решения этой задачи необходимо применить стили WPF.

На рис. 20.5 представлен рефакторинг кода (из рис. 20.3) с использованием механизма стилей.

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="150" Width="400">
    <Window.Resources>
        <LinearGradientBrush x:Key="DefaultBrush">
            <GradientStop Color="AliceBlue" Offset="0.2"/>
            <GradientStop Color="Aquamarine" Offset="0.7"/>
        </LinearGradientBrush>
        <Style x:Key="DefaultStyle">
            <Setter Property="Control.Width" Value="150"/>
            <Setter Property="Control.Height" Value="50"/>
            <Setter Property="Control.BorderThickness" Value="10"/>
            <Setter Property="Control.FontFamily" Value="Times New Roman"/>
            <Setter Property="Control.Background" Value="{StaticResource DefaultBrush}"/>
        </Style>

    </Window.Resources>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Button Grid.Column="0"
                Style="{StaticResource DefaultStyle}">
            Button One
        </Button>
        <Button Grid.Column="1"
                Style="{StaticResource DefaultStyle}">
            Button Two
        </Button>
    </Grid>
</Window>
```

Рисунок 20.5 – Код XAML с использованием стилей

Код существенно сократился. Стили также могут использовать ссылки на ресурсы. Внешний вид двух кнопок установлен всего одной строкой – установкой свойства Style.

Стиль (фактически, это класс System.Windows.Style) определяется в словаре ресурсов. Как и ресурс, стиль может быть определен у различных элементов управления (хотя чаще определяется у элементов управления верхнего уровня).

В таблице 20.1 представлены основные свойства класса Style.

Таблица 20.1 – Свойства класса Style

Свойство	Описание
Setters	Коллекция объектов Setter или EventSetter, которые устанавливают значения для свойств и присоединяют обработчики событий автоматически

Triggers	Коллекция объектов, унаследованных от класса TriggerBase, которые позволяют автоматически изменять настройки стиля. Настройки стиля могут модифицироваться, например, при изменении значения какого-то другого свойства или при поступлении какого-нибудь события
Resources	Коллекция ресурсов, которые должны использоваться со стилями.
BasedOn	Свойство, которое позволяет создавать более специализированный стиль, наследующий (и дополнительно переопределяющий) параметры другого стиля
TargetType	Свойство, которое идентифицирует тип элемента, к которому применяется данный стиль. Это свойство позволяет создавать объекты Setter, влияющие только на определенные элементы, а также объекты Setter, автоматически вступающие в силу для всех элементов подходящего типа

В объекте типа Style размещается коллекция Setters с объектами Setter, по одному для каждого свойства, которое подлежит установке. В каждом объекте Setter указывается имя свойства Property, на которое он влияет, и значение Value, которое он должен применять к этому свойству. Как и все ресурсы, объект стиля имеет ключевое имя, по которому его можно при необходимости извлекать из коллекции.

Если бы в примере на рис. 20.5 было много кнопок, то можно обойтись без установки Style для каждой кнопки. Для этого необходимо описать стиль следующим образом:

<Style x:Key="{x:Type Button}">

Стиль с таким ключом будет применен ко всем кнопкам данного окна. Можно также установить свойство TargetType:

<Style TargetType="Button">

20.3.3 Триггеры

С помощью триггеров можно автоматизировать процесс внесения простых изменений в стили, для которых требуется написание рутинной логики обработки

событий. Например, можно обеспечить реакцию на изменение значения свойства и соответствующим образом автоматически подстроить стиль.

Триггеры связываются со стилями через коллекцию Style.Triggers. Каждый стиль может иметь любое количество триггеров, а каждый триггер является экземпляром класса, унаследованного от System.Windows.TriggerBase (таблица 20.2).

Таблица 20.2 – Классы, унаследованные от TriggerBase

Класс	Описание
Trigger	Простейшая форма триггера. Он следит за изменением в свойстве зависимости и затем использует средство установки для изменения стиля
MultiTrigger	Похож на Trigger, но поддерживает проверку множества условий. Этот триггер вступает в действие, только если удовлетворены все заданные условия
DataTrigger	Работает с привязкой данных. Он похож на Trigger, но следит за изменением в любых связанных данных
MultiDataTrigger	Объединяет множество триггеров данных
EventTrigger	Наиболее сложный триггер. Он применяет анимацию, когда возникает соответствующее событие

Продемонстрировать использование простого триггера можно с использованием примера (рис. 20.6), который дополняет стиль, представленный на рис. 20.5.

```
<Style x:Key="{x:Type Button}">
    <Style.Triggers>
        <Trigger Property="Control.IsMouseOver" Value="True">
            <Setter Property="Control.Foreground" Value="Red" />
            <Setter Property="Control.RenderTransform">
                <Setter.Value>
                    <RotateTransform Angle="10" CenterX="75" CenterY="25"></RotateTransform>
                </Setter.Value>
            </Setter>
        </Trigger>
    </Style.Triggers>
    <Setter Property="Control.Width" Value="150"/>
    <Setter Property="Control.Height" Value="50"/>
    <Setter Property="Control.BorderThickness" Value="10"/>
    <Setter Property="Control.FontFamily" Value="Times New Roman"/>
    <Setter Property="Control.Background" Value="{StaticResource DefaultBrush}"/>
</Style>
```

Рисунок 20.6 – Простой триггер

Из рис. 20.6 очевидно на какое событие среагирует триггер, а также как изменится внешний вид любой кнопки окна при наступлении условного события (или при срабатывании триггера). Свойства изменяются путем добавления объектов типа Setter в коллекцию триггера Trigger.Setters.

К одному элементу может быть применено любое количество триггеров (главное, чтобы отдельные триггеры не конфликтовали).

При использовании мульти триггера можно задать несколько условий срабатывания (рис. 20.7).

```
<Style x:Key="{x:Type Button}">
    <Style.Triggers>
        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition Property="Button.IsKeyboardFocused" Value="true"/>
                <Condition Property="Button.IsMouseOver" Value="True"/>
            </MultiTrigger.Conditions>
            <MultiTrigger.Setters>
                <Setter Property="Button.Width" Value="160"/>
                <Setter Property="Button.Height" Value="55"/>
                <Setter Property="Button.FontSize" Value="26"/>
            </MultiTrigger.Setters>
        </MultiTrigger>
        <Trigger Property="Control.IsMouseOver" Value="True">
            <Setter Property="Control.Foreground" Value="Red" />
            <Setter Property="Control.RenderTransform">
                <Setter.Value>
                    <RotateTransform Angle="10" CenterX="75" CenterY="25"/>
                </Setter.Value>
            </Setter>
        </Trigger>
    </Style.Triggers>
    <Setter Property="Control.Width" Value="150"/>
    <Setter Property="Control.Height" Value="50"/>
    <Setter Property="Control.BorderThickness" Value="10"/>
    <Setter Property="Control.FontFamily" Value="Times New Roman"/>
    <Setter Property="Control.Background" Value="{StaticResource DefaultBrush}"/>
</Style>
```

Рисунок 20.7 –Триггер, срабатывающий при наступлении нескольких условий

В приведенном листинге MultiTrigger срабатывает только в случае, если наступает одновременно два условия. При этом определенный ранее триггер не отменен – он продолжает функционировать. Нутри класса MultiTrigger присутствуют две коллекции: Setters и Conditions.

20.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2015 и выше.

20.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

20.6. Методика и порядок выполнения работы

Рассмотрим пример выполнения задания с использованием механизмов ресурсов, стилей и триггеров.

Условие задания: разработайте приложение, демонстрирующее N графических примитивов (эллипсов, прямоугольников) со случайными значениями положения, размера и стиля. Число N задается пользователем. Фигуры должны быть интерактивны, т.е. реагировать на какие-либо действия пользователя (любое событие).

Для решения данной задачи необходимо выполнить следующие действия:

1. Создайте проект WPF-приложения. Реализуйте XAML-разметку (рис. 20.8), основным назначением которой является задание стилей и поведения (триггеров) фигур.

```
<Window x:Class="ComplexDataExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="400" Width="800">
    <Window.Resources>
        <Style x:Key="style1">
            <Setter Property="Shape.Fill">
                <Setter.Value>
                    <LinearGradientBrush>
                        <GradientStop Offset="0" Color="Aqua"/>
                        <GradientStop Offset="0.5" Color="WhiteSmoke"/>
                        <GradientStop Offset="1.5" Color="DarkGreen"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="Shape.Stroke" Value="Blue"/>
            <Setter Property="Shape.StrokeThickness" Value="5"/>
        </Style>
        <Style x:Key="style2">
            <Setter Property="Shape.Fill" Value="Blue"/>
            <Setter Property="Shape.Stroke" Value="BlueViolet"/>
            <Setter Property="Shape.StrokeThickness" Value="7"/>
        </Style>
        <Style x:Key="style3">
            <Setter Property="Shape.StrokeThickness" Value="3"/>
            <Setter Property="Shape.Fill" Value="LawnGreen"/>
            <Setter Property="Shape.Stroke" Value="ForestGreen"></Setter>
        </Style>
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <TextBox Name="FigureCount"
                     ToolTip="Inter figere Count"
                     Margin="5" Text="10"/>
            <Button Grid.Column="1" Click="Button_Click">Generate Shapes</Button>
        </Grid>
        <Canvas Name="MainCanvas" Grid.Row="1" Margin="5">

        </Canvas>
    </Grid>
</Window>
```

Рисунок 20.8 – Разметка XAML для каркаса приложения

2. Добавьте обработчик события для кнопки «Generate Shapes» (рис. 20.9)

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    int N = 10;
    try
    {
        N = Convert.ToInt32(FigureCount.Text);
    }
    catch (Exception ee)
    {
        this.Title = "Только целое число!";
    }

    GenerteShapes(N);
}
```

Рисунок 20.9 – Обработчик события нажатия кнопки «Generate Shapes»

3. Реализуйте функцию GenerateShapes(int N), которая случайным образом выберет фигуру, ее стиль, размер и положение, а также добавит фигуру в Canvas (рис. 20.10).

```
private void GenerteShapes(int N)
{
    Random rndShapeType = new Random(DateTime.Now.Millisecond);
    Random rndStyle = new Random(DateTime.Now.Second);
    Random rndPosition = new Random(DateTime.Now.Minute);
    Random rndSize = new Random(DateTime.Now.Minute);

    for (int i = 0; i < N; i++)
    {
        Shape currentShape;
        int shapeType = rndShapeType.Next(0, 2);
        if(shapeType == 0)
            currentShape = new Ellipse();
        else
            currentShape = new Rectangle();

        int shapeStyle = rndStyle.Next(0, 3) + 1;
        String styleName = "style" + shapeStyle.ToString();
        Style currentStyle = (Style) this.FindResource(styleName);
        currentShape.Style = currentStyle;

        currentShape.Width = rndSize.Next(10, 200);
        currentShape.Height = rndSize.Next(10, 100);

        MainCanvas.Children.Add(currentShape);
        Canvas.SetLeft(currentShape, rndPosition.Next(5, 750));
        Canvas.SetTop(currentShape, rndPosition.Next(5, 370));
    }
}
```

Рисунок 20.10 – Реализация функции GenerateShapes(int N)

Результат (снимок работающего приложения) показан на рис 20.11.

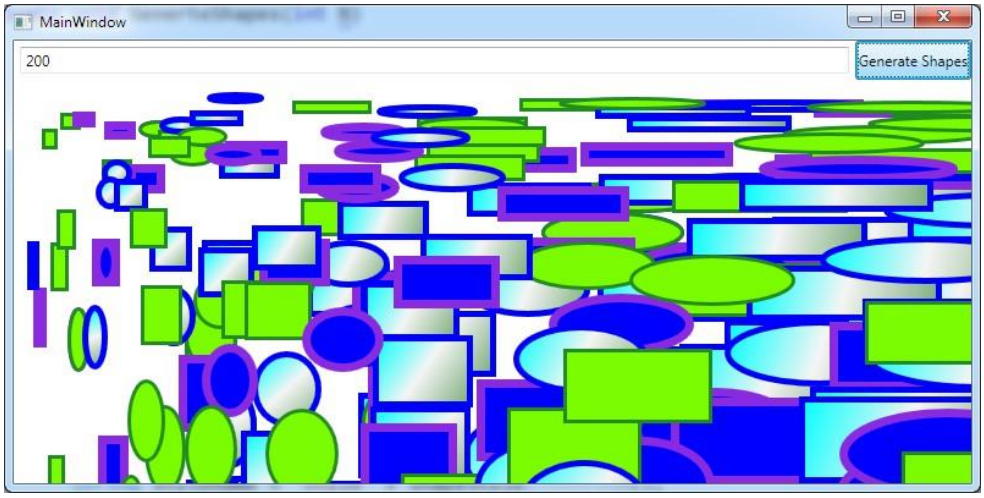


Рисунок 20.11 – Скриншот работающего приложения

На рис. 20.11 видно, что в процессе работы приложения используются оба типа фигур, а также задействованы все три стиля.

Варианты индивидуальных заданий

Разработайте собственные стили. По аналогии с задачей, рассмотренной в рамках лабораторной работы реализуйте генерацию люьых стилизованных объектов (фигур, элементов управления).

20.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю. Студент защищает лабораторную работу, отвечает на вопросы преподавателя, касающиеся разработанного приложения.

20.8. Вопросы для защиты работы

1. Что такое стиль?

2. Что такое ресурс?

3. Что такое триггер?

4. Опишите механизмы применения стилей

5. Опишите механизмы применения триггеров.

6. Какие типы ресурсов вы знаете?

7. У каких элементов WPF может быть коллекция ресурсов?

20.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [4].

ЛАБОРАТОРНАЯ РАБОТА 21. МЕХАНИЗМ ПРИВЯЗКИ WPF

21.1. Цель и содержание

Цель лабораторной научиться применять механизм привязки элементов управления.

Задачами лабораторной работы являются:

– изучить теоретические аспекты функционирования механизма привязки;

– научиться осуществлять привязку элементов управления в различных режимах.

21.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

21.3. Теоретическое обоснование

21.3.1 Основы привязки элементов управления

Простейший сценарий привязки данных подразумевает ситуацию, когда исходным объектом является элемент WPF, а исходным свойством – свойство зависимости. Свойства зависимости имеют встроенную поддержку уведомлений об изменениях. В результате, когда значение свойства зависимости изменяется в исходном объекте, привязанное свойство целевого объекта немедленно обновляется. Это именно то, что требуется, и происходит оно без необходимости построения любой дополнительной инфраструктуры.

Рассмотрим пример выполнения привязки. Окно содержит два элемента: Slider (ползунок) и TextBlock (текстовый блок) с единственной строкой текста. Перемещение ползунка вправо приводит к увеличению размера шрифта текста, а перемещение влево – к уменьшению размера шрифта. Такое поведение реализуется через Slider.ValueChanged.

Привязка WPF позволяет реализовать данную задачу проще. На рис. 21.1 представлен листинг XAML для окна WPF.

```
1  <Window x:Class="Task5.MainWindow"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="MainWindow" Height="200" Width="500">
5      <Grid>
6          <Grid.RowDefinitions>
7              <RowDefinition Height="Auto"/>
8              <RowDefinition/>
9          </Grid.RowDefinitions>
10         <Slider Name="sldSource" Value="14" Maximum="80" Minimum="1"
11                 TickFrequency="6" TickPlacement="Both"
12                 Grid.Row="0" Margin="5" AutoToolTipPlacement="TopLeft"/>
13         <TextBlock Name="txtTarget" Text="Образец текста" Grid.Row="1"
14                    TextAlignment="Center" VerticalAlignment="Center"
15                    FontSize="{Binding ElementName=sldSource, Path=Value}"
16                    Foreground="Red"/>
17     </Grid>
18 </Window>
```

Рисунок 21.1 – Листинг XAML главного окна приложения

В данном листинге на холсте главного окна размещен двухстрочный контейнер компоновки Grid. В него помещены два элемента управления:

– элемент Slider (с именем sldSource);

– элемент TextBlock (с именем txtTarget).

Кроме визуальных свойств (отступов, размеров и т.п.) в приведенном коде присутствует синтаксическая конструкция языка XAML, позволяющая связать размер шрифта элемента txtTarget к текущему значению ползунка sldSource.

В строке 15 листинга показано выражение привязки. Синтаксис данного выражения следующий:

ИмяЦелевогоСвойства =

"{

Binding ElementName = *ИмяЭлементаИсточника*,

Path = *ИмяСвойстваИсточника*

}"

Целевое свойство должно быть свойством зависимости.

Таким образом, свойство FontSize элемента txtTarget устанавливается не явно через указание численного значения, а с использованием привязки к свойству-источнику Value элемента-источника sldSource.

Рассмотренную привязку можно представить графически следующим образом:

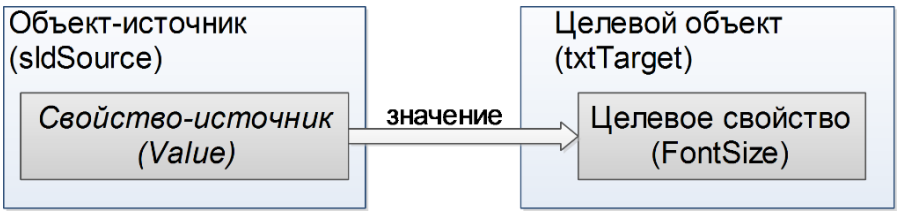


Рисунок 21.2 – Привязка целевого свойства к свойству-источнику значения, содержащемуся в элементе-источнике

Если во время привязки возникла ошибка (например, вместо свойства Value указано несуществующее свойство) – обрабатываться она не будет. WPF не генерирует исключения для уведомления о проблемах привязки данных. Во время отладки приложения эта информация появляется в выходном окне Visual Studio.

На рис. 21.3 представлен внешний вид окна работающего приложения. В результате включения привязки в код XAML изменения размера шрифта элемента достигнуты без использования кода на языке C#.

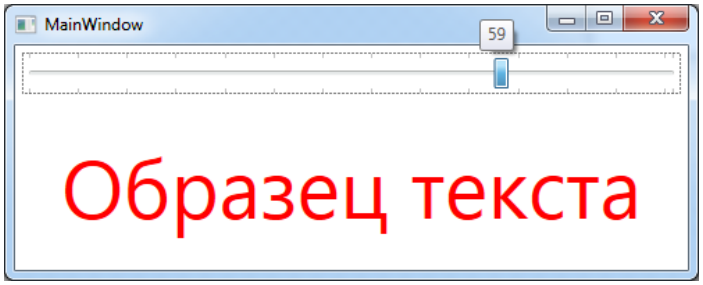


Рисунок 21.3 – Результат применения привязки

21.3.2 Режимы привязки

Особенность привязки данных заключается в том, что целевое свойство обновляется автоматически, независимо от того, как модифицируется источник.

Усложним ранее рассмотренный пример: в главное окно добавлены кнопки «Нормальный размер», «Крупный размер». Данные кнопки устанавливают по нажатию размер шрифта элемента txtTarget в значения 20 и 60 соответственно.

На рис. 21.4 приведен листинг XAML полученного окна.

```
<Window x:Class="Task5.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="200" Width="500">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Slider Name="sldSource" Value="14" Maximum="80" Minimum="1"
                TickFrequency="6" TickPlacement="Both"
                Grid.Row="0" Margin="5" AutoToolTipPlacement="TopLeft"/>
        <TextBlock Name="txtTarget" Text="Образец текста" Grid.Row="1"
                   TextAlignment="Center" VerticalAlignment="Center"
                   FontSize="{Binding ElementName=sldSource, Path=Value}"
                   Foreground="Red"/>
        <StackPanel Orientation="Horizontal" Grid.Row="2" HorizontalAlignment="Center">
            <Button Name="btnNormal" Content="Нормальный размер" Margin="5" Width="150"
                    Click="btnNormal_OnClick"/>
            <Button Name="btnLarge" Content="Крупный размер" Margin="5" Width="150"
                    Click="btnLarge_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

Рисунок 21.4 – Окно приложения с добавленными элементами управления

Для добавленных кнопок созданы обработчики нажатия. На рис. 5.5 представлен листинг обработчика нажатия кнопки «Нормальный размер». На рис. 21.6 – кнопки «Крупный размер».

```
private void btnNormal_OnClick(object sender, RoutedEventArgs e)
{
    sldSource.Value = 30;
}
```

Рисунок 21.5 – Обработчик нажатия кнопки «Нормальный размер», устанавливающий значение Value ползунка sldSource

```
private void btnLarge_Click(object sender, RoutedEventArgs e)
{
    txtTarget.FontSize = 60;
}
```

Рисунок 21.6 – Обработчик нажатия кнопки «Крупный размер», устанавливающий значение FontSize элемента txtTarget.

Обработчик на рис. 21.5 установит положение ползунка sldSource в положение 30. Так как значение Value ползунка изменится, то механизм привязки обновит значение размера шрифта элемента txtTarget автоматически.

Обработчик на рис. 21.6 вместо выражения привязки (рис. 21.4 строка 16) запишет числовое значение 60. Размер шрифта будет задан, но привязка будет аннулирована и дальнейшее перемещение ползунка пользователем или нажатием кнопки «Нормальный размер» не будет приводить к изменению размера шрифта TextBlock.

Такое поведение обусловлено режимом привязки. Привязка в рассмотренном примере – односторонняя (рис. 21.2). При изменении значения свойства-источника значение целевого свойства меняется, но обратное утверждение не действительно. В WPF существуют различные варианты привязок (таблица 21.1).

Устранить недостаток, проявленный в примере можно модифицировав выражение привязки следующим образом (рис. 21.7):

```
14        <TextBlock Name="txtTarget" Text="Образец текста" Grid.Row="1"
15                   TextAlignment="Center" VerticalAlignment="Center"
16                   FontSize="{Binding ElementName=sldSource, Path=Value, Mode=TwoWay}"
17                   Foreground="Red"/>
```

Рисунок 21.7 – Явное указание режима привязки.

В качестве значения режима привязки Mode указывается значение перечисления BindingMode. После указанной на рис. 21.7 модификации все кнопки будут работать в окне, не разрушая привязку.

Таблица 21.1 – Значения перечисления BindingMode.

Значение	Описание
OneWay	Целевое свойство обновляется при изменениях исходного свойства.
TwoWay	Целевое свойство обновляется при изменениях исходного свойства, а исходное свойство обновляется при изменении целевого свойства.
OneTime	Целевое свойство устанавливается изначально на основе значения исходного свойства. Однако с этого момента изменения игнорируются (если только привязка не устанавливается на совершенно другой объект или не вызывается BondingExpression.UpdateTarget()). Обычно этот режим используется для сокращения накладных расходов, если известно, что целевое свойство не изменится.

OneWayToSource	Подобно OneWay, но действует в обратном направлении. Исходное свойство обновляется, когда изменяется целевое свойство (что может показаться несколько странным), но целевое свойство никогда не обновляется.
Default	Этот тип привязки зависит от целевого свойства. Это либо TwoWay (для устанавливаемых пользователем свойств, таких как TextBox.Text), либо OneWay (для всего остального). Все привязки используют данный подход, если только не указано иное.

Таким образом, можно применять синтаксис привязки элементов управления следующего вида:

ИмяЦелевогоСвойства =
"{
Binding ElementName = *ИмяЭлементаИсточника*,
Path = *ИмяСвойстваИсточника*,
Mode = *РежимПривязки*
}"

На рис. 21.8 представлена схема взаимодействия компонентов привязки.

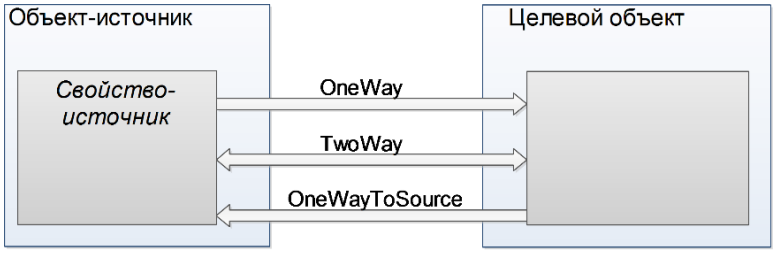


Рисунок 21.8 – Варианты привязок

Привязку можно реализовать в коде C#. На рис. 21.9 показан вариант реализации привязки на языке C#, синтаксис которой представлен на рис. 21.7. Оба варианта (с использованием XAML или C#) реализуют один и тот же механизм.

```
public MainWindow()
{
    InitializeComponent();
    Binding b = new Binding();
    b.Source = sldSource;
    b.Path = new PropertyPath("Value");
    b.Mode = BindingMode.TwoWay;
    txtTarget.SetBinding(TextBlock.FontSizeProperty, b);
}
```

Рисунок 21.9 – Реализация привязки на языке C#

Существуют методы для удаления привязок. Метод ClearBinding() принимает ссылку на свойство зависимости, которое имеет привязку, подлежащую удалению, а метод ClearAllBindings() удаляет все привязки данных элемента (рис. 21.10).

```
private void MainWindow_OnUnloaded(object sender, RoutedEventArgs e)
{
    BindingOperations.ClearAllBindings(txtTarget);
    // или
    BindingOperations.ClearBinding(txtTarget, TextBlock.FontSizeProperty);
}
```

Рисунок 21.10 – Удаление привязки в коде C#

Привязка чаще реализуется в коде XAML, но использование реализации на языке C# эффективно в случаях:

1. Создание динамических привязок. Если необходимо тонко настраивать привязку на основе другой информации времени выполнения или создавать разные привязки в зависимости от обстоятельств, имеет смысл делать это в коде. (В качестве альтернативы можно было бы определить все необходимые привязки в коллекции Resources окна и просто добавить код, который вызывает SetBinding() с соответствующим объектом привязки.)

2. Удаление привязки. Чтобы удалить привязку и получить возможность установки свойства обычным образом, понадобится помощь метода ClearBinding() или ClearAllBindings(). Недостаточно просто присвоить новое значение свойству: в случае использования двунаправленной привязки установленное значение распространится на привязанный объект, и оба свойства останутся синхронизированными.

21.3.3 Множественные привязки

Программист может осуществлять не только одиночные привязки (один целевой объект – один источник), разработчикам WPF доступны множественные привязки (один целевой элемент – множество источников). На рис. 21.11 показан вариант кода XAML, демонстрирующий привязку различных свойств элемента txtTarget к различным свойствам-источникам.

```
1  <Window x:Class="Task5.MainWindow"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="MainWindow" Height="400" Width="500" Unloaded="MainWindow_OnUnloaded">
5      <Grid>
6          <Grid.RowDefinitions>
7              <RowDefinition Height="Auto"/>
8              <RowDefinition/>
9              <RowDefinition Height="Auto"/>
10             <RowDefinition Height="Auto"/>
11         </Grid.RowDefinitions>
12         <Slider Name="sldSource" Value="14" Maximum="80" Minimum="1"
13                 TickFrequency="6" TickPlacement="Both"
14                 Grid.Row="0" Margin="5" AutoToolTipPlacement="TopLeft"/>
15         <DockPanel Grid.Row="1">
16             <ListBox Name="listColor" DockPanel.Dock="Left" Width="150" SelectedIndex="0">
17                 <ListBoxItem Content="Красный" Tag="Red"/>
18                 <ListBoxItem Content="Зеленый" Tag="Green"/>
19                 <ListBoxItem Content="Сисний" Tag="Blue"/>
20                 <ListBoxItem Content="Другой" Tag="#58aa45"/>
21                 <ListBoxItem Content="Розовый" Tag="Pink"/>
22             </ListBox>
23             <!--Целевой элемент привязки: -->
24             <TextBlock Name="txtTarget" DockPanel.Dock="Right"
25                    TextAlignment="Center" VerticalAlignment="Center"
26                    FontSize="{Binding Value, ElementName=sldSource, Mode=TwoWay}"
27                    Foreground="{Binding ElementName=listColor, Path=SelectedItem.Tag}"
28                    Text="{Binding Text, ElementName=txtTextSource}" />
29         </DockPanel>
30         <TextBox Name="txtTextSource" Text="Введите текст" Grid.Row="2"/>
31         <StackPanel Orientation="Horizontal" Grid.Row="3" HorizontalAlignment="Center">
32             <Button Name="btnNormal" Content="Нормальный размер" Margin="5" Width="150"
33                     Click="btnNormal_OnClick"/>
34             <Button Name="btnLarge" Content="Крупный размер" Margin="5" Width="150"
35                     Click="btnLarge_Click"/>
36         </StackPanel>
37     </Grid>
38 </Window>
```

Рисунок 21.11 – Применение множественной привязки

Обратите внимание на строки 24-28, в которых реализуются привязки, листинга на рис. 21.11.

На рис. 21.12 представлен внешний вид полученного окна приложения.

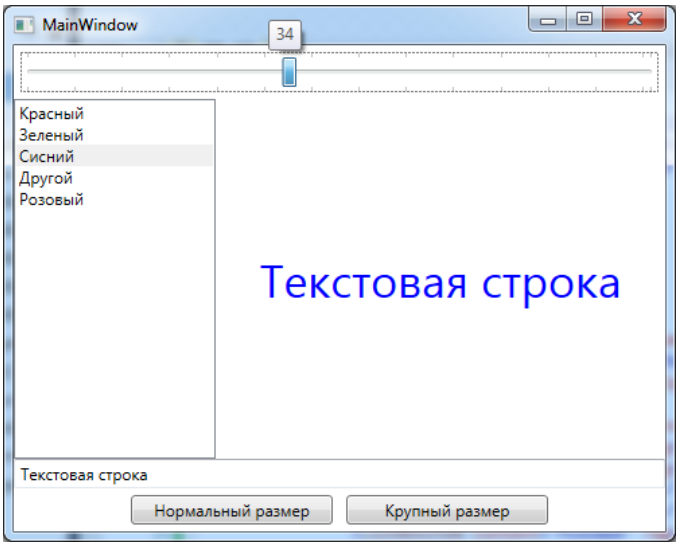


Рисунок 21.12 – Применение множественной привязки

21.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

21.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

21.6. Методика и порядок выполнения работы

Для закрепления материала о привязках элементов управления и получения практических навыков программирования приложений с использованием привязки к элементам управления, реализуйте приложение, удовлетворяющее следующим требованиям:

– в программе должно быть реализовано не менее 5 привязок к элементам управления;

– каждая привязка должна получать данные из свойства-источника, причем все свойства-источники должны быть различного типа: строковые, логические, целочисленные, вещественные, цвет, имя шрифта и т.д.

– приложение должно демонстрировать различные режимы привязок;

– одна из привязок должна создаваться динамически.

21.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю.

21.8. Вопросы для защиты работы

1. Что такое привязка WPF?
2. Поясните синтаксис привязки. Какие ключевые слова и для каких целей используются при оформлении привязки с помощью XAML?
3. Какие режимы привязки существуют? Поясните назначение всех режимов привязки.
4. Поясните схему привязки элементов управления. Назовите основные элементы данной схемы.
5. Что означают понятия «целевой элемент», «элемент-источник привязки», «целевое свойство», «свойство-источник»?
6. Какое ограничение на источник привязки существует в WPF?

21.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2], [3].

ЛАБОРАТОРНАЯ РАБОТА 22. ИСТОЧНИКИ ПРИВЯЗКИ ПРОИЗВОЛЬНОГО ТИПА

22.1. Цель и содержание

Цель лабораторной работы: научиться использовать механизм привязки с объектами-источниками пользовательского типа.

Задачами лабораторной работы являются:

– научиться осуществлять привязку элементов к источнику, который является объектом пользовательского класса;

– научиться разрабатывать многослойные (multi tier) приложения WPF.

22.2. Формируемые компетенции

ОПК-2, ПК-4, ПК-6, ПК-10, ПК-11, ПК-16

22.3. Теоретическое обоснование

22.3.1 Многомодульные приложения

При разработке реальных информационных программных систем очень редко используются одномодульные приложения (в виде единственного исполняемого файла .exe). Оптимальным вариантом является разбиение приложения на отдельные уровни, каждый из которых представляется отдельной сборкой (модулем). На рис. 22.1 представлена стандартная схема многоуровнего приложения (multi tier application), которая включает: уровень доступа к данным (Data tier), уровень бизнес-логики (Logic tier) и уровень презентаций (Presentation Tier).

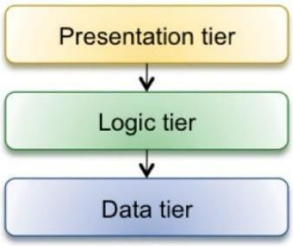


Рисунок 22.1 – Multy tier Application

Такое разделение на отдельные слои позволяет осуществлять многопользовательскую параллельную разработку сложного приложения.

22.3.2 Привязка к произвольным объектам

При реализации механизма привязки необходимо помнить, что целевое свойство должно быть свойством зависимости. Но на объект-источник ограничения отсутствуют. Это означает, что программист может привязать к свойству зависимости любое другое свойство любого другого объекта, независимо от природы его создания.

В предыдущих лабораторных работах рассматривались простые механизмы привязки, основанные на отношении между двумя элементами управления WPF, в котором один элемент – целевой объект привязки, другой – источник привязки.

Но гораздо чаще программисту приходится извлекать данные не из визуального элемента управления, а из объекта определенного бизнес-класса (то есть класса, описанного программистом и предоставляющего необходимые данные, а также реализующего определенный набор действий). В данном механизме необходимо учитывать ограничение – данные для привязки должны находиться в общедоступных свойствах.

При привязке к объекту, который не является элементом управления WPF, пользоваться свойством привязки ElementName не представляется возможным. Вместо него программист должен использовать одно из следующих свойств:

1. Source. Ссылка, указывающая на исходный объект; другими словами, это объект, поставляющий данные. Существует несколько подходов: извлечение

объекта источника из ресурса, программная генерация объекта источника или получение источника от поставщика данных.

2. RelativeSource. Указывает на исходный объект, основываяь на отношениях между исходным и целевым объектом. То есть, можно привязать целевой объект к самому себе (сам целевой объект используется в качестве объекта-источника), к родительскому элементу или к определенному элементу, отстоящему от целевого на N уровней в дереве элементов.

3. DataContext. Если источник не был указан с помощью свойства Source или RelativeSource, то среда WPF производит поиск в дереве элементов, начиная с текущего элемента. Она проверяет свойство DataContext каждого элемента и использует первый из них, который не равен null. Свойство DataContext исключительно полезно, когда нужно привязать несколько свойств одного объекта к разным элементам, потому что можно установить свойство DataContext высокоуровневого объекта контейнера, вместо его установки непосредственно на целевой элемент.

В рамках данной лабораторной работы будет изучен механизм использования свойства DataContext, как наиболее простого и широко применяемого.

22.4. Аппаратура и материалы

Для выполнения лабораторной работы необходим компьютер с установленной операционной системой Windows 7, 8, 8.1 и выше. В качестве среды разработки используется интегрированная среда разработки Microsoft Viasual Studio 2012 и выше.

22.5. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не

производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

22.6. Методика и порядок выполнения работы

Необходимо написать приложение для обработки информации о товарах. О каждом товаре необходимо обрабатывать информацию:

– код;

– наименование;

– цена;

– количество;

– описание.

Для освоения методики привязки к объектам произвольного класса, а также получения навыков разработки многомодульных приложений, необходимо реализовать три этапа:

1. Разработать слой доступа к данным.

2. Разработать слой бизнес-логики.

3. Разаработать слой презентаций.

Каждый слой должен быть реализовываться в отдельном проекте.

Для решения задачи выполните следующую последовательность действий:

1. В окне Visual Studio создайте пустое решение с именем «Shop» (не проект!). Для этого выберите тип проекта «Visual Studio Solution» (Blank Solution) (рис. 22.1).

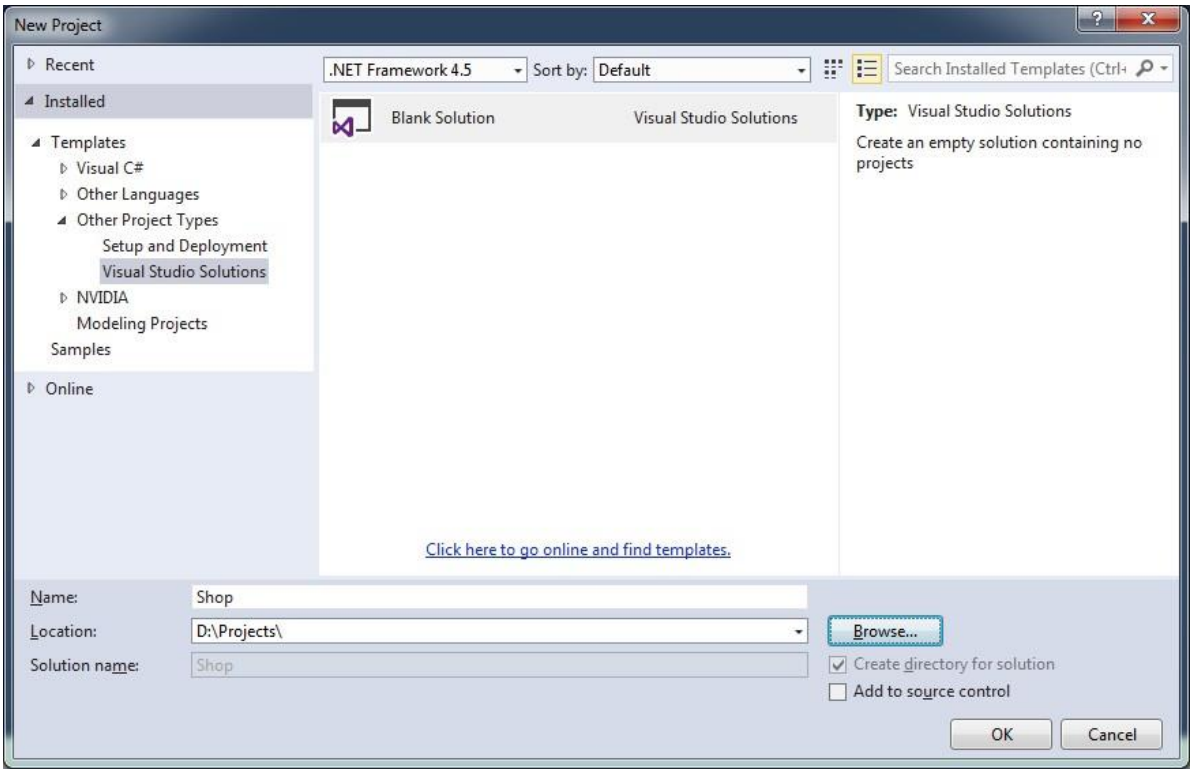


Рисунок 22.1 – Создание пустого решения Visual Studio

2. В обозревателе проектов появится новое пустое решение с именем «Shop» (рис. 22.2).

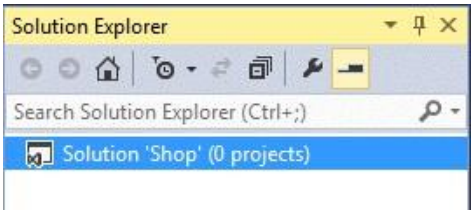


Рисунок 22.2 – Вид обозревателя проекто после создания пустого решения «Shop»

3. Добавьте в решение «Shop» три проекта (таблица 22.1).

Таблица 22.1 – Проекты, добавляемые в решение «Shop»

Имя проекта	Тип проекта	Назначение
DataTier	C# → Windows → Class Library	Библиотека классов для работы с данными. Данный проект представляет уровень доступа к данным в трехуровневой архитектуре.
LogicTier	C# → Windows → Class Library	Библиотека классов, представляющая все необходимые данные и методы для пользовательского интерфейса. Проект представляет уровень бизнес-логики в трехуровневой архитектуре.

PresentationTier	C# → Windows → WPF Application	Проект оконного приложения Windows Presentation Foundation. Данный проект представляет собой уровенб презентаций и содержит интерфейс программы.

На рис. 22.3 показан внешний вид обозревателя проектов после добавления новых проектов.

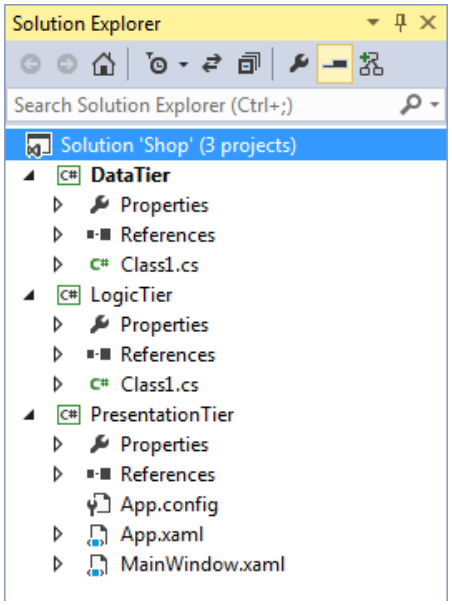


Рисунок 22.3 – Вид обозревателя проекто после добавления отдельных проектов

Наиболее простой способ решения данной задачи – реализовать уровни (tiers) в следующей последовательности с целью избежания архитектурных ошибок разработки:

1) Программирование уровня DataTier. Легче всего реализовать данный уровень первым, т.к. он в большей степени зависит от данных (в данном случае от предметной области из условия задания), чем от Windows-приложения.

2) Программирование уровня PresentationTier. Реализация данного уровня не требует знаний в области программирования, т.к. презентация WPF – это внешний вид оконного приложения Windows. Данный уровень предполагает, в основном, работу дизайнера.

3) Программирование уровня LogicTier. Цель данного слоя приложения – связать уровни DataTier и PresentationTier.

4. Проект DataTier будет содержать классы, необходимые для представления данных (таблица 22.2).

Таблица 22.2 – Классы проекта «DataTier»

Имя проекта	Назначение
Товар	Представляет отдельный товар; содержит свойства, определяемые заданием
ВсеТовары	Представляет список товаров; содержит методы сохранения и загрузки товаров на носителе

На рис. 22.4 представлен листинг класса «Товар».

```
7  namespace DataTier
8  {
       11 references
9      public class Товар
10     {
           7 references
11         public String Код { get; set; }
           7 references
12         public String Наименование { get; set; }
           8 references
13         public float Цена { get; set; }
           7 references
14         public int Количество { get; set; }
           6 references
15         public String Описание { get; set; }
16     }
17 }
```

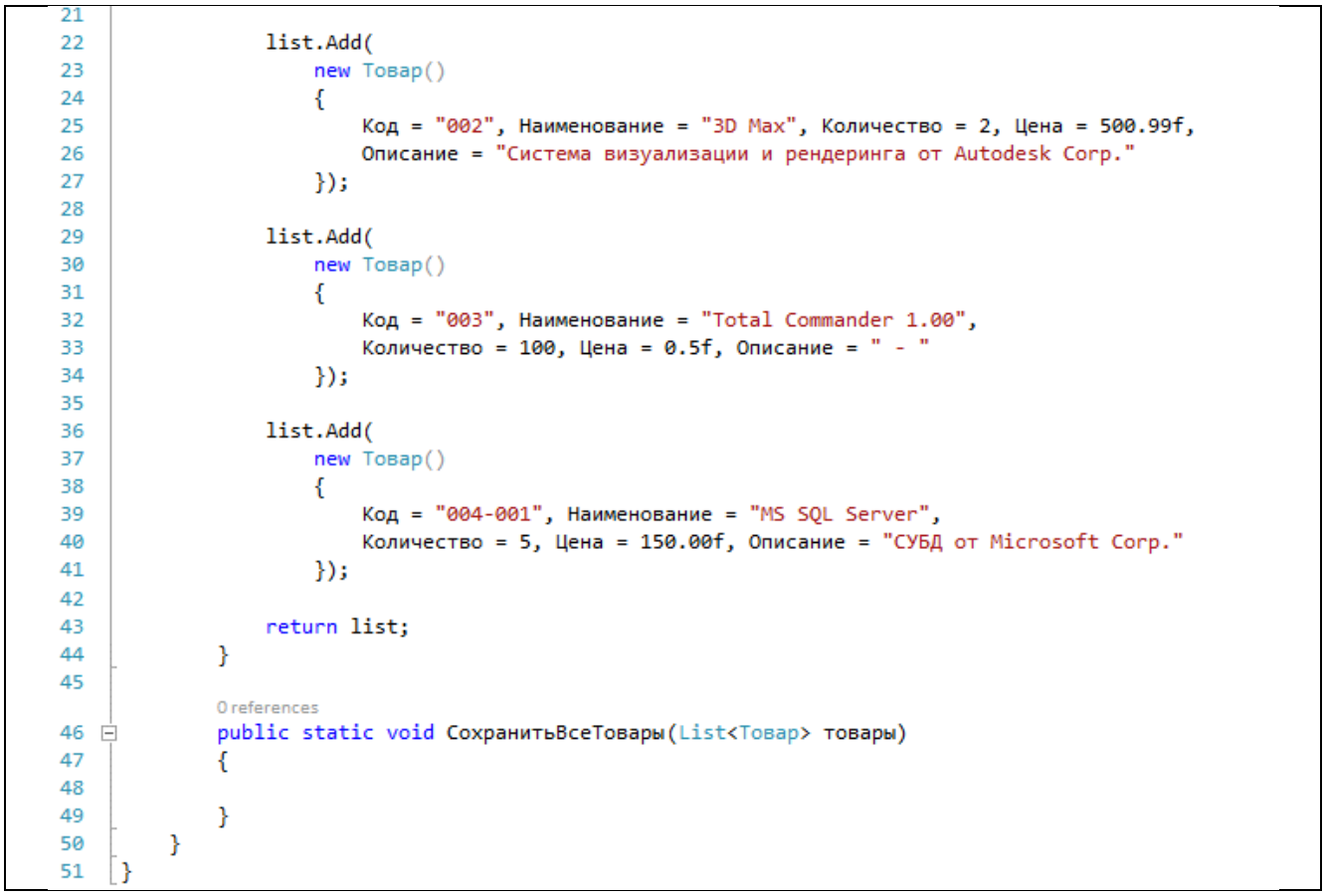
Рисунок 22.4 – Определение класса «Товар»

На рис. 22.5 представлен листинг класса «ВсеТовары». Класс «ВсеТовары» содержит два метода:

– метод ПолучитьВсеТовары(), возвращающий список товаров;

– метод СохранитьВсеТовары() для сохранения списка товаров на внешнем носителе.

```
7  namespace DataTier
8  {
       1 reference
9      public class ВсеТовары
10     {
           1 reference
11         public static List<Товар> ПолучитьВсеТовары()
12         {
13             List<Товар> list = new List<Товар>();
14
15             list.Add(
16                 new Товар()
17                 {
18                     Код = "001", Наименование = "ОС Windows 8", Количество = 10, Цена = 40.99f,
19                     Описание = "Современная операционная система. Версия 8"
20                 });
```



```
21
22              list.Add(
23                  new Товар()
24                  {
25                      Код = "002", Наименование = "3D Max", Количество = 2, Цена = 500.99f,
26                      Описание = "Система визуализации и рендеринга от Autodesk Corp."
27                  });
28
29              list.Add(
30                  new Товар()
31                  {
32                      Код = "003", Наименование = "Total Commander 1.00",
33                      Количество = 100, Цена = 0.5f, Описание = " - "
34                  });
35
36              list.Add(
37                  new Товар()
38                  {
39                      Код = "004-001", Наименование = "MS SQL Server",
40                      Количество = 5, Цена = 150.00f, Описание = "СУБД от Microsoft Corp."
41                  });
42
43              return list;
44          }
45
            0 references
46          public static void СохранитьВсеТовары(List<Товар> товары)
47          {
48
49          }
50      }
51  }
```

Рисунок 22.5 – Определение класса «ВсеТовары»

Классы, представленные на рис. 22.4 и 22.5, являются минимальной абстракцией объектов предметной области. Метод ПолучитьВсеТовары() должен считывать список товаров из файла или получать его из базы данных, но в данном примере осуществляется построение списка в коде. Метод СохранитьвсеТовары() оставим пустым. Все операции сохранения и загрузки данных, взаимодействие с внешними носителями, осуществляются через слой DataTier.

5. Уровни DataTier и LogicTier реализованы в виде библиотек, в которых отсутствует метод Main(). То есть данные проекты не предполагают запуска. Библиотеки LogicTier и DataTier после выполнения построения будут оформлены в сборки *.dll. На данном шаге решения задачи уровень DataTier реализован, откомпилируйте данный проект и исправьте ошибки, если они есть.

6. Для взаимодействия с пользователем необходимо реализовать уровень презентаций. Проект «PresentationTier» будет содержать главную форму оконного приложения.

На рис. 22.6 представлен внешний вид формы MainWindow.xaml в режиме проектирования.

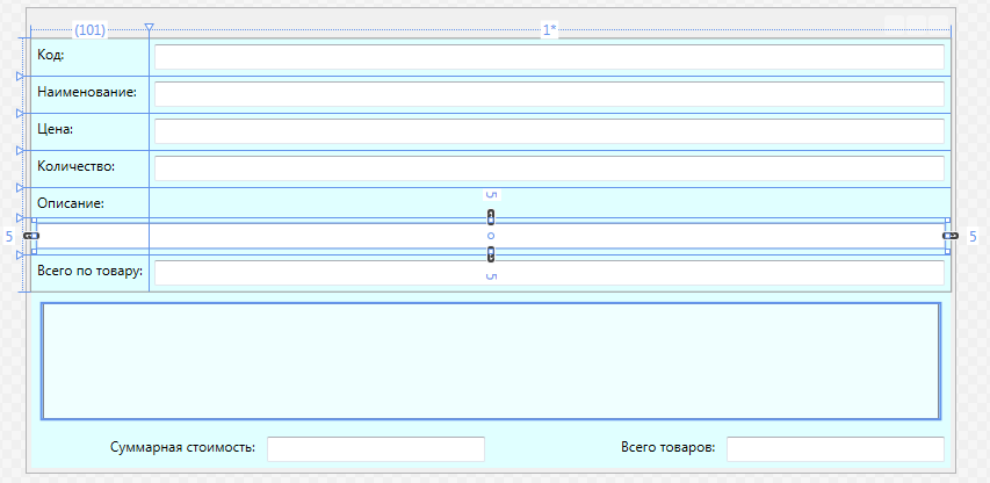


Рисунок 22.6 – Дизайн MainWindow.xaml

Рисунок 22.7 иерархию элементов управления в главном окне.

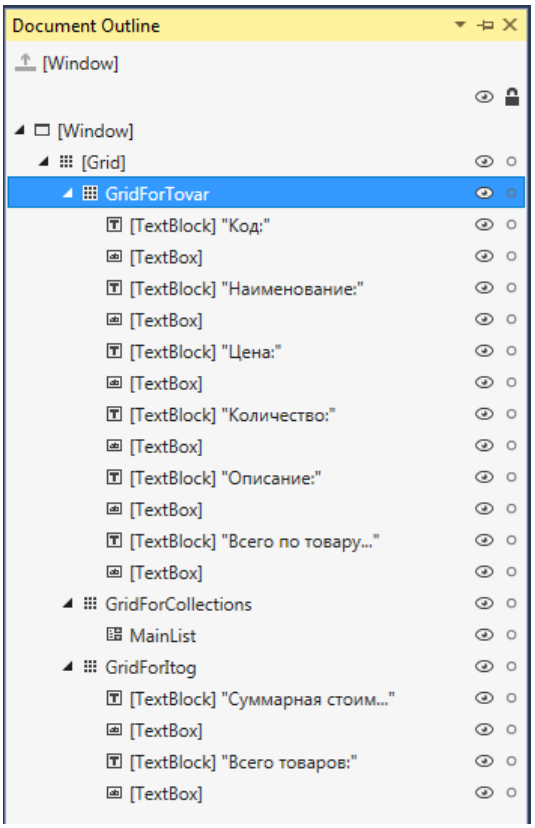


Рисунок 22.7 – Иерархическая структура элементов в окне MainWindow.xaml

Из рисунка 22.7 видно, что окно Window является корневым элементом в визуальном дереве. На втором уровне иерархии расположен элемент Grid без установленного свойства Name, который вмещает в себя все остальные элементы.

На третьем уровне расположены три элемента типа Grid: GridForTovar (содержит поля для отображения информации о конкретном товаре), GridForCollections (содержит поле MainList типа ListBox для отображения списка товаров), GridForItog (область для вывода аггрегирующих показателей).

На рис. 22.8 представлен код XAML для представленного на рисунках 22.6, 22.7 окна.

```
<Window x:Class="PresentationTier.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="{Binding НаименованиеМагазина}" Height="400" Width="800"
        WindowStartupLocation="CenterScreen" Background="LightCyan">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid Grid.Row="0" Name="GridForTovar"
              DataContext="{Binding ElementName=MainList, Path=SelectedItem}">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"></ColumnDefinition>
                <ColumnDefinition></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <TextBlock Margin="5" Grid.Row="0" Text="Код:"/>
            <TextBox Margin="5" Grid.Row="0" Grid.Column="1"
                     Text="{Binding КодТовара}"/>
            <TextBlock Margin="5" Grid.Row="1" Text="Наименование:"/>
            <TextBox Margin="5" Grid.Row="1" Grid.Column="1"
                     Text="{Binding НаименованиеТовара}"/>
            <TextBlock Margin="5" Grid.Row="2" Text="Цена:"/>
            <TextBox Margin="5" Grid.Row="2" Grid.Column="1"
                     Text="{Binding ЦенаТовара}"/>
            <TextBlock Margin="5" Grid.Row="3" Text="Количество:"/>
            <TextBox Margin="5" Grid.Row="3" Grid.Column="1"
                     Text="{Binding КоличествоТовара}"/>
            <TextBlock Margin="5" Grid.Row="4" Text="Описание:"/>
            <TextBox Margin="5" Grid.Row="5"
                     Grid.Column="0" Grid.ColumnSpan="2"
                     Text="{Binding ОписаниеТовара}"/>
            <TextBlock Margin="5" Grid.Row="6" Text="Всего по товару:"/>
            <TextBox Margin="5" Grid.Row="6" Grid.Column="1"
                     Text="{Binding СуммарнаяСтоимостьПозиции, Mode=OneWay}"/>
        </Grid>
```

```
47          <Grid Grid.Row="1" Name="GridForCollections">
48              <ListBox Name="MainList" ItemsSource="{Binding СписокТоваров, Mode=OneWay}"
49                       DisplayMemberPath="ПредставлениеТовара" Background="Azure"
50                       Margin="10"/>
51          </Grid>
52          <Grid Grid.Row="2" Name="GridForItog">
53              <Grid.ColumnDefinitions>
54                  <ColumnDefinition/>
55                  <ColumnDefinition/>
56                  <ColumnDefinition/>
57                  <ColumnDefinition/>
58              </Grid.ColumnDefinitions>
59              <TextBlock Margin="5" Text="Суммарная стоимость:" Grid.Column="0"
60                         HorizontalAlignment="Right"/>
61              <TextBox Margin="5" Grid.Column="1"
62                       Text="{Binding Path=СуммарнаяСтоимость, Mode=OneWay}" />
63              <TextBlock Margin="5" Text="Всего товаров:" Grid.Column="2"
64                         HorizontalAlignment="Right"/>
65              <TextBox Margin="5" Grid.Column="3"
66                       Text="{Binding Path=СуммарноеКоличество, Mode=OneWay}"/>
67          </Grid>
68      </Grid>
69  </Window>
```

Рисунок 22.8 – Код XAML главного окна

Как и в случае использования привязки к элементам управления в данном коде также присутствуют привязки. В некоторых элементах управления устанавливаются различные свойства с использованием {Binding …}. В связи с этим необходимо дать некоторые пояснения по коду XAML, представленному на рис. 22.8:

Таблица 22.3 – Особенности кода XAML, связанные с выполнением привязки данных

Номер строки	Исходный код и пояснения
4	Title="{Binding НаименованиеМагазина}" Производится установка заголовка окна. Значение свойства Title привязывается к полю данных НаименованиеМагазина. Такого поля на данном этапе проектирования приложения не существует, поля данных НаименованиеМагазина будет реализовано позже.
29	Text="{Binding КодТовара}" Значение свойства Text элемента управления TextBox привязывается к полю данных КодТовара. Такого поля на данном этапе проектирования приложения не существует, поля данных КодТовара будет реализовано позже.
32, 35, 38, 42	Значение свойства Text элемента управления TextBox привязывается к полю данных, значение которого будет отображено в данном TextBox. Аналогично с механизмом привязки, продемонстрированным в строке 29.

45	Text="{Binding СуммарнаяСтоимостьПозиции, Mode=OneWay} Значение свойства Text элемента управления TextBox привязывается к полю данных СуммарнаяСтоимостьПозиции в режиме OneWay. Аналогичен предыдущим привязкам, но не предполпгает обновления источника привязки при изменении целевого свойства.
48, 49	ItemsSource="{Binding СписокТоваров, Mode=OneWay}" DisplayMemberPath="ПредставлениеТовара" Элемент ListBox отображает не единственное значение, а целый список элементов. Поэтому для привязки к источнику этих элементов применяется свойство ItemsSource. Очевидно, что если нам необходимо выводить список товаров, то нужно указать, какое именно поле будет выводиться в этом списке пользователю; для этого устанавливает свойство DisplayMemberPath. Поле данных «ПредставлениеТовара» на данном этапе не реализовано.
62, 66	Text="{Binding Path=СуммарнаяСтоимость, Mode=OneWay}" Text="{Binding Path=СуммарноеКоличество, Mode=OneWay}" Механизм привязки аналогичен продимонстрированому в строке 45.
13	DataContext="{Binding ElementName=MainList, Path=SelectedItem}" Фактически, выполняется привязка одного элемента управления к другому. Производится привязка свойства DataContext элемента Grid к элементу MainList типа ListBox. Но не ко всему списку, а только к объекту, заданному свойством Path, то есть к свойству ListBox.SelectedItem (текущий выбранный элемент).

После верстки главного окна с использование инструментария XAML приложение WPF может быть запущено. Внешний вид созданного приложения представлен на рис. 22.9.

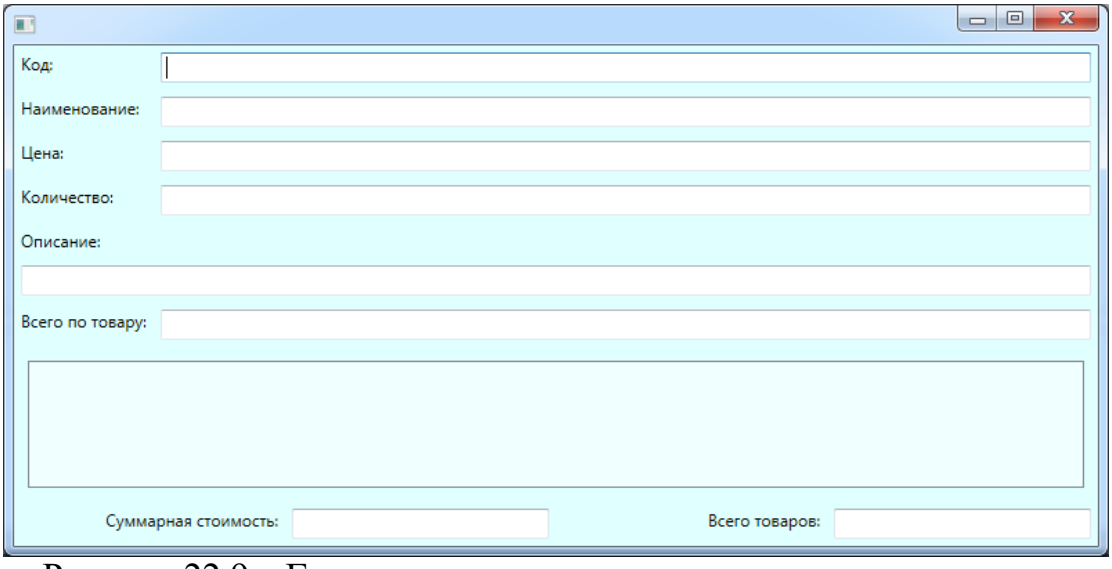


Рисунок 22.9 – Главное окно приложения в режиме выполнения

На данном этапе приложение функционирует, но нет связи с данными. Несмотря на то, что осуществлялась привязка к несуществующим данным, приложение все равно работает. Теперь в нашем приложении две сборки: DataTier.dll и PresentationTier.exe. Реализуем связующее звено – LogicTier.dll.

7. В проекте LogicTier создадим классы, представленные в таблице 22.4.

Таблица 22.4 – Особенности кода XAML, связанные с выполнением привязки данных

Имя класса	Описание
ТоварнаяПозиция	Это класс, который «оборачивает» класс «Товар» из проекта DataTier. Класс «ТоварнаяПозиция» дополняет «сырой» класс данных всем необходимым для вышестоящих уровней приложения.
Магазин	Данный класс «оборачивает» класс «ВсеТовары» из проекта DataTier.

На рис. 22.10 представлен листинг класса «Товарная позиция».

```
8  namespace LogicTier
9  {
10     public class ТоварнаяПозиция
11     {
12         private Товар _товар;
13
14         public ТоварнаяПозиция(Товар p)
15         {
16             _товар = p;
17         }
18         public String КодТовара
19         {
20             get { return _товар.Код; }
21             set { _товар.Код = value; }
22         }
23
24         public String НаименованиеТовара
25         {
26             get { return _товар.Наименование; }
27             set { _товар.Наименование = value; }
28         }
29
```

```
30          public float ЦенаТовара
31          {
32              get { return _товар.Цена; }
33              set { _товар.Цена = value; }
34          }
35
36          public int КоличествоТовара
37          {
38              get { return _товар.Количество; }
39              set { _товар.Количество = value; }
40          }
41
42          public String ОписаниеТовара
43          {
44              get { return _товар.Описание; }
45              set { _товар.Описание = value; }
46          }
47
48          public float СуммарнаяСтоимостьПозиции
49          {
50              get { return _товар.Цена * _товар.Количество; }
51          }
52
53          public String ПредставлениеТовара
54          {
55              get { return _товар.Код + " : " + _товар.Наименование
56                  + " (" + _товар.Цена.ToString("C") + ")"; }
57          }
58      }
59  }
```

Рисунок 22.10 – Код класса «ТоварнаяПозиция»

В представленном коде производится связь между уровнем презентаций и уровнем данных: для каждой привязки к единичному товару в коде XAML (табл. 22.3) в данном классе создается одноименное поле. Если привязка производилась в режиме OneWay, поле может быть объявлено только для чтения (например, поле ПредставлениеТовара).

В свою очередь, каждое поле из класса «ТоварнаяПозиция» является оберткой над соответствующим полем класса «Товар» (т.е. представляет это поле или вычисляется на основе нескольких полей).

Листинг класса «Магазин» представлен на рис. 22.11.

```
8   namespace LogicTier
9   {
10      public class Магазин
11      {
12          private List<ТоварнаяПозиция> _товары = new List<ТоварнаяПозиция>();
13
14          public Магазин()
15          {
16              List<Товар> tmp = ВсеТовары.ПолучитьВсеТовары();
17              foreach (var t in tmp)
18              {
19                  _товары.Add(new ТоварнаяПозиция(t));
20              }
21          }
22
```

```
23          public List<ТоварнаяПозиция> СписокТоваров
24          {
25              get { return _товары; }
26          }
27
28          public String НаименованиеМагазина {
29              get { return "Наш магазин"; }
30          }
31
32          public float СуммарнаяСтоимость {
33              get
34              {
35                  return _товары.Sum(p => p.СуммарнаяСтоимостьПозиции);
36              }
37          }
38
39          public float СуммарноеКоличество
40          {
41              get
42              {
43                  return _товары.Sum(p => p.КоличествоТовара);
44              }
45          }
46      }
47  }
```

Рисунок 22.11 – Определение класса «Магазин»

Класс «Магазин» – это «обертка» над классом «ВсеТовары».

После компиляции и исправления ошибок получим три сборки: DataTier.dll, LogicTier.dll, PresentationTier.exe.

8. Завершающий этап разработки приложения: связь класса MainWindow и класса LogicTier. Для выполнения данной связи необходимо задать свойство DataContext главного окна. На рис. 22.12 представлен измененный код конструктора MainWindow.

```
17  namespace PresentationTier
18  {
19      /// <summary>
20      /// Interaction logic for MainWindow.xaml
21      /// </summary>
22      public partial class MainWindow : Window
23      {
24          public MainWindow()
25          {
26              Магазин logicTier = new Магазин();
27              this.DataContext = logicTier;
28              InitializeComponent();
29          }
30      }
31  }
```

Рисунок 22.12 – Конструктор MainWindow

Из листинга видно, что создается объект класса «Магазин» и устанавливается в качестве источника привязки через свойство DataContext для окна MainWindow. После этого все привязки начинают получать данные. Внешний вид окна полученного приложения представлен на рис. 22.13.

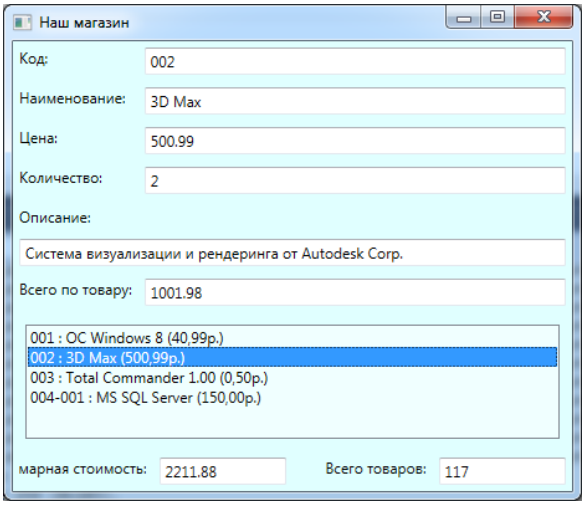


Рисунок 22.13 – Результат решения задачи

9. Выполните задание в соответствии с индивидуальным вариантом.

Варианты индивидуальных заданий

В каждом варианте необходимо реализовать многоуровневое приложение для обработки данных из файла. Формат текстового файла представлен в каждом варианте. В отличии от рассмотренного примера, приложение должно считывать информацию из файла. Приложение должно быть многомодульным (как в реализованном примере).

Вариант	Задание

1	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк):
	`Иванов С.Е.| доцент | ИСТ | 56.00`
	`Петров А.А. | профессор | ИБАС | 100.23`
	`Новиков П.С. | ст. преподаватель | ИСТ | 35.00`
	`Ганотченко Е.П. | профессор | ИБАС | 200.45`
	`Петров А.А. | профессор | ИБАС | 105.67`
	Необходимо вычислить: количество преподавателей; сумму зарплат преподавателей по каждой кафедре.

2	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк):
	`ВАЗ-2107; Автоваз; 140000.57; 16000`
	`Гога-1; Хуань Вей Трейд; 50.00; 1000`
	`ВАЗ-5000; Автоваз; 1400000.00; 1000`
	`X35; Hyundai; 750000; 1`
	Необходимо вычислить: суммарную стоимость всех автомобилей; автомобиль с наименьшим пробегом.

3	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк):
	`Самолет*Москва*Пекин*30000.00`
	`Автобус*Ставрополь*Москва*3000.00`
	`Автобус*Невинномысск*Ставрополь*10.00`
	`Автобус*Москва*Владивосток*50000`
	`Самолет*СПб*Москва*1500`
	Необходимо вычислить: количество автобусных рейсов; суммарную стоимость билетов рейсов самолетов; самый дорогой билет.

4	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк):
	`Иванов Е.А.*ИСТ-б-о-121*3*2`
	`Петров А.А.*ИСТ-б-о-111*4*1`
	`Кононов А.Е.*БАС-б-о-111*4*0`
	`Катченко Е.А.*БАС-б-о-121*3*1`
	Необходимо вычислить: суммарное количество задолженностей по каждому курсу; количество студентов без задолженностей.

5	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк):
	`Молоко, 0.5 л % Продукты % 10.00 % 23.00`
	`Хлеб белый % Продукты % 15.00 % 22.00`
	`Молоко, 1.0 л % Продукты % 18.00 % 30.00`
	`Туалетный утенок % Химия % 11.00 % 17.58`
	`Фейри, 0.5 л % Химия % 17.00 % 88.00`
	Необходимо вычислить: суммарную стоимость товаров по каждой группе; самый дорогой товар в каждой товарной группе.

6	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк):
	`Кабель сетевой % Сетевое оборудование % 10.00 % Пулково 1`
	`Коммутатор ASUS X100 % Сетевое оборудование % 15.00 % Склад №7`
	`ПО Windows 8.1, 1.0 л % ПО % 18.00 % Склад №19`
	`ПО Office 365 % ПО % 11.00 % Склад №19`
	`Коннектор 234511 % Сетевое оборудование % 17.00 % Склад №7`
	Необходимо вычислить: среднюю стоимость товаров по каждой группе; склад, содержащий максимальное количество продукции.

7	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк):
	`Дом*6*150*3500000`
	`Квартира*1*49*790000`
	`Квартира*2*59*1500000`
	`Квартира*3*70*3000000`
	`Комната*1*12*500000`
	Необходимо вычислить: для каждого строения рассчитать стоимость квадратного метра; суммарную стоимость объектов по каждому типу.

8	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк):
	`Привет с того света | Детектив | 150.00`
	`Кто подставил кролика | Детектив | 200.10`
	`Почемучка | Детская литература | 60.22`
	`Капитошка | Детская литература | 230.14`
	`Звезда 112 | Фантастика | 12.90`
	Необходимо вычислить: самую дешевую книгу в каждом жанре; среднюю стоимость книг в каждом жанре.

9	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк):
	`Молоко \ Продукты \ 40.00 \ 1`
	`Тряпка \ Хозтовары \ 10.00 \ 5`
	`Порошок \ Хозтовары \ 750.00 \ 10`
	`Мясо \ Продукты \ 350 \ 5`
	`Швабра \ Хозтовары \ 900.00 \ 5`
	Необходимо вычислить: цену каждого товара за вычетом скидки; суммарную стоимость товаров по каждой группе.

10	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк):
	`Рис 1 кг. / 120.11 / 54 / Магнит`
	`Ручка шариковая / 23.00 / 15 / ТЦ Карандаш`
	`Тетрадь 12 л. / 15.00 / 200 / ГеоКнига`
	`Тетрадь 60 л. / 90.70 / 300 / ГеоКнига`
	`Торт 1 кг. / 450 / 4 / Магнит`
	Необходимо вычислить: среднюю цену товара по каждому магазину; суммарную стоимость для каждого товара.

11	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк):
	`Иванов С.Е.| доцент | ИСТ | 56.00`
	`Петров А.А. | профессор | ИБАС | 100.23`
	`Новиков П.С. | ст. преподаватель | ИСТ | 35.00`
	`Ганотченко Е.П. | профессор | ИБАС | 200.45`
	`Петров А.А. | профессор | ИБАС | 105.67`
	Необходимо вычислить: суммарный фонд заработной платы; среднюю зарплату по каждой кафедре.

12	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк):
	`ВАЗ-2107; Автоваз; 140000.57; 16000`
	`Гога-1; Хуань Вей Трейд; 50.00; 1000`
	`ВАЗ-5000; Автоваз; 1400000.00; 1000`
	`X35; Hyundai; 750000; 1`
	Необходимо вычислить: средний пробег по каждому производителю; среднюю стоимость автомобилей по каждому производителю.

13	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк):
	`Самолет*Москва*Пекин*30000.00`
	`Автобус*Ставрополь*Москва*3000.00`
	`Автобус*Невинномысск*Ставрополь*10.00`
	`Автобус*Москва*Владивосток*50000`
	`Самолет*СПб*Москва*1500`
	Необходимо вычислить: среднюю стоимость билета по каждому виду транспорта; количество рейсов из Москвы.

14	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк):
	`Иванов Е.А.*ИСТ-б-о-121*3*2`
	`Петров А.А.*ИСТ-б-о-111*4*1`
	`Кононов А.Е.*БАС-б-о-111*4*0`
	`Катченко Е.А.*БАС-б-о-121*3*1`
	Необходимо вычислить: суммарное количество задолженностей по каждой группе; среднее количество задолженностей по каждому курсу.

15	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк):
	`Молоко, 0.5 л % Продукты % 10.00 % 23.00`
	`Хлеб белый % Продукты % 15.00 % 22.00`
	`Молоко, 1.0 л % Продукты % 18.00 % 30.00`
	`Туалетный утенок % Химия % 11.00 % 17.58`
	`Фейри, 0.5 л % Химия % 17.00 % 88.00`
	Необходимо вычислить: суммарную прибыль от продажи товаров; среднюю цену закупки по каждой группе.

16	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк):
	`Кабель сетевой % Сетевое оборудование % 10.00 % Пулково 1`
	`Коммутатор ASUS X100 % Сетевое оборудование % 15.00 % Склад №7`
	`ПО Windows 8.1, 1.0 л % ПО % 18.00 % Склад №19`
	`ПО Office 365 % ПО % 11.00 % Склад №19`
	`Коннектор 234511 % Сетевое оборудование % 17.00 % Склад №7`
	Необходимо вычислить: среднюю стоимость товаров по каждому складу; суммарную стоимость по каждой группе.

17	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк): 1 Дом*6*150*3500000 2 Квартира*1*49*790000 3 Квартира*2*59*1500000 4 Квартира*3*70*3000000 5 Комната*1*12*500000 Необходимо вычислить: вывести объект с максимальной стоимостью квадратного метра; среднюю стоимость квадратного метра по объектам типа «Квартира».
18	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк): 1 Привет с того света | Детектив | 150.00 2 Кто подставил кролика | Детектив | 200.10 3 Почемучка | Детская литература | 60.22 4 Капитошка | Детская литература | 230.14 5 Звезда 112 | Фантастика | 12.90 Необходимо вычислить: суммарную стоимость книг по каждому жанру; среднюю стоимость книг по жанру «Фантастика».
19	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк): 1 Молоко \ Продукты \ 40.00 \ 1 2 Тряпка \ Хозтовары \ 10.00 \ 5 3 Порошок \ Хозтовары \ 750.00 \ 10 4 Мясо \ Продукты \ 350 \ 5 5 Швабра \ Хозтовары \ 900.00 \ 5 Необходимо вычислить: среднюю цену товара (с учетом скидки) по каждой товарной группе; товар с минимальной скидкой.
20	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк): 1 Рис 1 кг. / 120.11 / 54 / Магнит 2 Ручка шариковая / 23.00 / 15 / ТЦ Карандаш 3 Тетрадь 12 л. / 15.00 / 200 / ГеоКнига 4 Тетрадь 60 л. / 90.70 / 300 / ГеоКнига 5 Торт 1 кг. / 450 / 4 / Магнит Необходимо вычислить: суммарную стоимость товаров в каждом магазине; магазин с минимальным товарным ассортиментом.

22.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; цель и задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты,

полученные в ходе её выполнения (в данной лабораторной работе необходимо привести листинг кода обработчиков событий, которые были изменены разработчиком).

Отчет о выполнении лабораторной работы с датой, подписанный студентом, сдается преподавателю.

22.8. Вопросы для защиты работы

1. Чем отличается привязка к элементам управления от привязки к произвольным объектам?
2. Поясните синтаксис привязки к данным. Какие ключевые слова и для каких целей используются при оформлении привязки с помощью XAML?
3. Какие режимы привязки существуют? Поясните назначение всех режимов привязки.
4. Поясните схему привязки элементов управления. Назовите основные элементы данной схемы.
5. Что означают понятия «целевой элемент», «элемент-источник привязки», «целевое свойство», «свойство-источник»?
6. Какое ограничение на источник привязки существует в WPF?
7. Какое ограничение существует на целевое свойство привязки?

22.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1], [2].

СПИСОК ЛИТЕРАТУРЫ

Основная литература

1. Павлова, Е. А. Технологии разработки современных информационных систем на платформе Microsoft .NET : учебное пособие / Павлова Е. А. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 128 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-9963-0003-7.

2. Пугачев, С. В.
    Разработка приложений для Windows 8 на языке C# / С. Пугачев, А. Шериев, К. Кичинский. - СПб. : БХВ-Петербург, 2013. - 416 с. : ил. - (Профессиональное программирование). - ISBN 978-56-9775-0846-9.

Дополнительная литература

1. Зиборов, В. В. Visual C# 2012 на примерах / Виктор Зиборов. - Санкт-Петербург : БХВ-Петербург, 2013. - 473 с. : ил. ; 24 см. - ISBN 978-5-9775-0888-9.

2. Прайс, Д. Visual C#2.0.NET. Полное руководство : [учеб. пособие] : пер. с англ. / Дж. Прайс, М. Гандрэрлой. - СПб. : КОРОНА-век, 2011. - 736 с. : ил. - Прил.: с. 673. - ISBN 978-5-7931-0555-2.

3. Чеповский, А. Common Intermediate Language и системное программирование в Microsoft .NET / А. Чеповский ; А. Макаров ; С. Скоробогатов. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 399 с. - (Основы информатики и математики). - ISBN 5-94774-410-4.

Перечень Интернет-ресурсов

1. http://metanit.com - сайт посвящен различным языкам и технологиям программирования, компьютерам, мобильным платформам и ИТ-технологиям.

2. https://msdn.microsoft.com/magazine/ - Интернет-журнал о технологиях разработки Microsoft.

3. https://professorweb.ru - информационный ресурс, посвященный разработке приложений на основе технологии .NET.

Учебное пособие (лабораторный практикум) по дисциплине

«Технологии разработки программного обеспечения»

для студентов направления

09.03.04 «Программная инженерия»

Составитель: Е.И. Николаев