

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Кузбасский государственный технический университет
имени Т.Ф. Горбачева»

Составитель:
Д. Е. Турчин

АРХИТЕКТУРА СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Лабораторный практикум

Рекомендовано учебно-методической комиссией
направления подготовки 230400.68
«Информационные системы и технологии»
в качестве электронного издания
для использования в учебном процессе

	What	How	Where	Who	When	Why	
Contextual							Contextual
Conceptual							Conceptual
Logical							Logical
Physical							Physical
As Built							As Built
Functioning							Functioning
	What	How	Where	Who	When	Why	

Кемерово 2013



Рецензенты:

Ванеев О. Н. -доцент кафедры информационных и автоматизированных производственных систем

Полетаев В. А. – д.т.н., профессор, председатель учебно-методической комиссии направления подготовки 230400.68 «Информационные системы и технологии»

Турчин Денис Евгеньевич. Архитектура современных информационных систем. [Электронный ресурс]: лабораторный практикум для студентов направления подготовки 230400.68 «Информационные системы и технологии» очной формы обучения / Сост. Д. Е. Турчин - Электрон. дан. – Кемерово: КузГТУ, 2012. – Систем. требования: Pentium IV; ОЗУ 256 Мб; WindowsXP; мышь. – Загл. с экрана.

В данных методических указаниях изложены содержание лабораторных работ, порядок и примеры их выполнения, а также контрольные вопросы к ним.

© КузГТУ

© Турчин Д. Е.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	2
ЛАБОРАТОРНЫЕ РАБОТЫ	3
1. Основы создания запросов к коллекциям объектов с помощью LINQ	3
1.1. Цель работы	3
1.2. Основные теоретические сведения	3
1.3. Порядок выполнения работы.....	14
1.4. Контрольные вопросы	18
2. Основы работы с XML-документами с помощью LINQ to XML	19
2.1. Цель работы	19
2.2. Основные теоретические сведения	19
2.3. Порядок выполнения работы.....	31
2.4. Контрольные вопросы	34
3. Основы создания приложений WPF с использованием языка XAML	35
3.1. Цель работы	35
3.2. Основные теоретические сведения	35
3.3. Порядок выполнения работы.....	57
3.4. Контрольные вопросы	58
4. Основы привязки и форматирования данных в приложениях WPF	59
4.1. Цель работы	59
4.2. Основные теоретические сведения	59
4.3. Порядок выполнения работы.....	71
4.4. Контрольные вопросы	72
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА	73
ПРИЛОЖЕНИЕ	74
П.1. Пример разработки XML-документа	74

ПРЕДИСЛОВИЕ

Пособие предназначено для студентов направления подготовки магистра 230400.68 «Информационные системы и технологии», изучающих дисциплину «Архитектура современных информационных систем».

Целью лабораторных занятий является формирование умений, связанных с использованием технологий и средств разработки архитектуры информационных систем.

В качестве используемых средств и технологий выступают:

- технология доступа к данным LINQ;
- интегрированная среда разработки программного обеспечения MS Visual Studio 2010 и язык программирования Visual C# 2010;
- платформа разработки клиентских приложений MS Windows Presentation Foundation (WPF) и язык XAML;

Выписка из ФГОС ВПО по направлению 230400.68

Код	Компетенции, формируемые при освоении дисциплины	Результаты освоения
Б2.Б.5	<ul style="list-style-type: none"> • ПК-1 умение разрабатывать стратегии проектирования, определение целей проектирования, критериев эффективности, ограничений применимости; • ПК-2 умение разрабатывать новые методы и средства проектирования информационных систем; • ПК-9 умение проводить разработку и исследование методик анализа, синтеза, оптимизации и прогнозирования качества процессов функционирования информационных систем и технологий; • ПК-10 умение осуществлять моделирование процессов и объектов на базе стандартных пакетов автоматизированного проектирования и исследований. 	<p>Знать:</p> <ul style="list-style-type: none"> • понятие архитектуры информационной системы и ее контекст (ПК-2); • основные домены предметные области архитектуры информационной системы (ПК-2); • методики описания архитектуры (ПК-9); • общую схему процесса разработки архитектуры информационной системы и методы управления этим процессом (ПК-1). <p>Уметь:</p> <ul style="list-style-type: none"> • моделировать бизнес-процессы предприятия с использованием различных языков (ПК-10); • разрабатывать архитектурное описание информационной системы (ПК-6); • определять цели и задачи проектирования архитектуры информационной системы (ПК-1).

ЛАБОРАТОРНЫЕ РАБОТЫ

1. ОСНОВЫ СОЗДАНИЯ ЗАПРОСОВ К КОЛЛЕКЦИЯМ ОБЪЕКТОВ С ПОМОЩЬЮ LINQ

1.1. Цель работы

Цель работы – приобрести умение выполнять запросы к источникам данных в форме коллекций объектов с помощью технологии LINQ to Objects.

Работа рассчитана на 4 часа.

1.2. Основные теоретические сведения

1.2.1. Общие сведения о технологии LINQ. Операции запросов LINQ. Расширяющие методы

Общие сведения о технологии LINQ. Запросы LINQ.

Одной из наиболее важных технологий для доступа к данным на платформе .NET является LINQ (*Language Integrated Query*) – язык интегрированных запросов.

Под **LINQ** понимают набор средств, появившийся в .NET Framework 3.5, который предоставляет стандартные технологии для работы с различными типами источников данных. Для запросов и преобразований данных в LINQ используются одинаковые базовые шаблоны кодирования, напоминающие SQL.

В состав Visual Studio, начиная с версии 2008, входят средства для использования LINQ с коллекциями объектов, поддерживающих интерфейс IEnumerable, базами данных SQL Server, наборами данных ADO.NET и XML-файлами.

По типу источника данных выделяют следующие разновидности LINQ:

- ***LINQ to Objects*** – позволяет применять запросы к массивам и коллекциям объектов;
- ***LINQ to DataSet*** – позволяет применять запросы LINQ к объектам DataSet из ADO.NET;
- ***LINQ to Entities*** – позволяет применять запросы LINQ внутри API-интерфейса ADO.NET Entity Framework (EF);

- ***LINQ to XML*** – позволяет применять запросы LINQ к документам XML и манипулировать XML-данными;
- ***Parallel LINQ (PLINQ)*** – позволяет выполнять параллельную обработку данных, возвращенных запросом LINQ.

Для применения средств LINQ to Objects в исходный код программы следует импортировать пространство имен **System.Linq**. Для этого в файле C# должна присутствовать следующая директива:

```
using System.Linq;
```

В основу LINQ положено понятие *запроса*, в котором определяется сведения, получаемые из источника данных. При необходимости, запрос также указывает способ сортировки и группировки этих сведений. В запросе LINQ работа всегда осуществляется с объектами.

Для сохранения результатов запросов LINQ удобно использовать *неявно типизируемые локальные переменные*, задаваемые ключевым словом **var**. Тип данных таких переменных определяется во время компиляции на основе первоначального значения.

Неявная типизация применима только для локальных переменных в контексте какого-то метода или свойства. Не допускается применять ключевое слово **var** для определения полей классов, возвращаемых значений и параметров методов.

Кроме того, локальным переменным, объявленным с помощью ключевого слова **var**, обязательно должно быть присвоено начальное значение в самом объявлении.

Все операции запроса LINQ состоят из трех различных действий:

1. Получение доступа к источнику данных. При этом в источнике должен быть реализован интерфейс **IEnumerable**.
2. Создание запроса, которое заключается в определении того, что именно следует извлечь из источника данных.
3. Выполнение запроса, в ходе которого выводятся результаты. Это может быть сделано в цикле **foreach**.

В общем виде запрос LINQ записывается следующим образом:

```
var переменная_запроса = выражение_запроса;
```

Запрос хранится в *переменной запроса* и инициализируется *выражением запроса*. Сама переменная запроса только хранит команды запроса. Фактическое выполнение запроса откладывается до выполнения итерации переменной запроса в операторе **foreach**. Выражение запроса состоит из набора предложений, включающих операции запросов LINQ и операнды.

Операции запросов LINQ.

В языке C# определены различные операции запросов LINQ. Наиболее часто используемыми операциями являются:

- **from ... in** – используется для определения основы любого выражения, позволяющей извлечь подмножество данных из нужного источника;
- **select** – используется для выбора последовательности из источника данных;
- **where** – используется для определения ограничений на извлекаемые из источника данные;
- **orderby** – позволяет упорядочить результирующий набор;
- **group** – позволяет группировать данные по указанному ключу.

В простейшем виде каждый запрос LINQ строится из операций **from**, **in** и **select**, которые вместе с операндами образуют соответствующие предложения. Синтаксис простейшего запроса имеет вид:

```
var перем_запроса = from перем_диапаз in источник_данных
                    select результат_запроса;
```

Переменная диапазона, следующая за операцией **from**, принимает данные из *источника данных*, записанного после операции **in**. Тип переменной диапазона выводится из источника данных. Имя переменной диапазона задается в самом выражении запроса.

Запрос может содержать несколько предложений **from**, что требуется при получении данных из нескольких источников.

Операция **select** определяет, что именно должно быть получено по запросу (*результат запроса*). Когда предложение с опе-

рацией **select** создает что-либо отличное от копии переменной диапазона, то операция **select** называется *проекцией*. Использование проекций для преобразования данных является мощной возможностью выражений запросов LINQ.

Чтобы получить определенное подмножество из источника можно использовать операцию **where**. При этом запрос будет иметь следующий синтаксис:

```
var перемен_запроса = from перемен_диапаз in источник_данных
                        where логич_выражение
                        select результат_запроса;
```

Логическое выражение, задаваемое после операции **where**, задает определенное условие и возвращает булевское значение. Запрос возвращает только те элементы, для которых логическое выражение является истинным.

Для построения выражения фильтра в предложении **where** можно использовать логические операции C# AND (&&) и OR (||).

Часто требуется отсортировать возвращенные запросом данные по одному или нескольким критериям. Для этого используется операция **orderby**. Запрос LINQ при задании сортировки будет иметь следующий вид:

```
var перемен_запроса = from перемен_диапаз in источник_данных
                        orderby выраж_сортировки [descending]
                        select результат_запроса;
```

Выражение сортировки содержит элемент, по которому проводится сортировка. По умолчанию принята сортировка по возрастанию, поэтому упорядочивание строк производится в алфавитном порядке, числовых значений – от меньшего к большему, и т.д.

Операция **descending** используется для сортировки в обратном порядке (по убыванию). По умолчанию используется операция **ascending**, которую можно не записывать при сортировке по возрастанию.

Операция **group** служит для группирования полученных данных по ключам. Данная операция, как и операция **select**, мо-

жет завершать выражение запроса. Запрос с операцией **group** в простейшем случае может быть записан следующим образом:

```
var перемен_запроса = from перемен_диапаз in источник_данных
                        group результат_запроса by ключ;
```

Когда запрос завершается предложением **group**, его результаты представляют группу списков. При итерации таких результатов запроса, необходимо использовать вложенный цикл **foreach**. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл – итерацию элементов каждой группы.

Для результата выполнения операции **group** определено доступное только для чтения свойство **Key**, которое возвращает ключ.

Операция **join** служит для объединения двух последовательностей данных в одну. При использовании операции **join** каждый источник должен содержать данные, которые можно сравнивать. Сравниваемые элементы данных указываются после ключевого слова **on**. При этом операция **join** отбирает только те элементы данных, которые имеют общее значение. Запрос LINQ при использовании операции **join** имеет следующий синтаксис:

```
var перемен_запроса = from перемен_диап_A in источник_данных_A
                        join перемен_диап_B in источник_данных_B
                        on перемен_диап_A.свойство equals перемен_диап_B.свойство
                        select результат_запроса;
```

1.2.2. Анонимные типы. Расширяющие методы

Анонимные типы. Расширяющие методы.

В языке C# одним из средств, непосредственно связанных с LINQ, являются **анонимные типы**. Как следует из названия, анонимный тип представляет собой класс, не имеющий имени. Его основное назначение состоит в создании объекта, возвращаемого оператором **select**.

Благодаря анонимным типам в ряде случаев отпадает необходимость объявлять класс, который предназначен только для хранения результата запроса.

Анонимный тип объявляется с помощью следующей общей формы:

```
new { имя_A = значение_A, имя_B = значение_B, ... }
```

Для того чтобы поменять порядок элементов в результирующем наборе на обратный используется расширяющий метод **Reverse<T>()** класса **Enumerable**.

Для вычисления одного значения из коллекции значений используются статистические операции. К основным статистическим операциям LINQ относятся:

- **Average** – вычисляет среднее арифметическое значение в коллекции;
- **Count** – подсчитывает число элементов в коллекции (при необходимости только те элементы, которые удовлетворяют заданному условию);
- **Max** – определяет максимальное значение в коллекции;
- **Min** – определяет минимальное значение в коллекции;
- **Sum** – вычисляет сумму значений в коллекции.

□ **Пример 1.1. Выполнение запросов LINQ к массиву объектов с помощью консольного приложения на языке C#.**

Требуется разработать набор запросов LINQ к массиву объектов, каждый из которых содержат данные о продуктовом товаре (наименование, производитель, количество единиц, вес одной единицы, цена, код стеллажа), хранимом на складе.

Исходный код на языке C# для класса **Product**, описывающего продуктовой товар, представлен на рис. 1.1, а код класса **Store** (стеллаж) – на рис. 1.2.

```

8 // Описание класса Product (товар)
9 class Product
10 {
11     // Автоматические свойства класса Product
12     public string Name { get; set; } // Наименование
13     public string ProducedBy { get; set; } // Производитель
14     public int NumberInStok { get; set; } // Число единиц
15     public int Weight { get; set; } // Вес, гр
16     public double Price { get; set; } // Цена, руб
17     public string StoreID { get; set; } // Код стеллажа
18
19     // Переопределение метода ToString для вывода информации об объектах
20     public override string ToString()
21     {
22         return string.Format("Наименование: {0}.\n" +
23             "Производитель: {1}\n" +
24             "Количество={2} шт.; Вес={3} гр.; Цена={4:f2} руб.\n" +
25             "Код стеллажа: {5}", Name, ProducedBy, NumberInStok,
26             Weight, Price, StoreID);
27     }
28 }

```

Рис. 1.1. Исходный код класса **Product**

```

30 // Описание класса Store (стеллаж)
31 class Store
32 {
33     // Автоматические свойства класса Store
34     public string StoreID { get; set; } // Код стеллажа
35     public int NumberOfFlour { get; set; } // Число ярусов
36     public int MaxWeight { get; set; } // Макс. грузонесущая способность, кг
37 }

```

Рис. 1.2. Исходный код класса **Store**

Запросы:

1. Данные о всех товарах.
2. Наименования всех товаров в алфавитном порядке.
3. Товары с количеством более 50 шт.
4. Товары фирмы «Алтайпродукт» с ценой менее 80 руб.
5. Число наименований товаров весом от 250 до 500 г.
6. Наименования товаров и их количество в порядке убывания количества.
7. Средняя, наибольшая и наименьшая цены товаров фирмы «Алтайпродукт».
8. Суммарный вес всех товаров на складе.
9. Общая стоимость товаров каждого наименования.
10. Наименования товаров, сгруппированные по производителям (используется операция **group**).
11. Наименование и количество товара с указанием данных о стеллаже, на котором он хранится (используется операция **join**).

```

11 static void Main(string[] args)
12 {
13     Console.Title = "Выполнение запросов LINQ к массиву объектов";
14
15     // Объявление массива объектов класса Product с инициализацией элементов
16     Product[] itemsInStock = new[] {
17         new Product { Name = "Крупа гречневая",
18             ProducedBy = "ООО Алтайпродукт",
19             NumberInStok = 42,
20             Weight = 350,
21             Price = 72.50,
22             StoreID = "024"
23         },
24         new Product { Name = "Кукуруза консервированная",
25             ProducedBy = "ОАО Балтпром",
26             NumberInStok = 67,
27             Weight = 340,
28             Price = 56.80,
29             StoreID = "017"
30         },
31         new Product { Name = "Крупа рисовая",
32             ProducedBy = "ЗАО Увелка",
33             NumberInStok = 53,
34             Weight = 350,
35             Price = 42.35,
36             StoreID = "024"
37         },
38         new Product { Name = "Говядина тушеная",
39             ProducedBy = "ОАО Балтпром",
40             NumberInStok = 30,
41             Weight = 330,
42             Price = 96.00,
43             StoreID = "017"
44         },
45         new Product { Name = "Макароны",
46             ProducedBy = "ООО Алтайпродукт",
47             NumberInStok = 48,
48             Weight = 500,
49             Price = 56.50,
50             StoreID = "028"
51         }
52     };
53
54     Console.WriteLine("***** Результаты запросов LINQ *****");
55     // Вызовы методов выполнения запросов LINQ
56     GetAllProds(itemsInStock);
57     GetAllNames(itemsInStock);
58     GetProdOver(itemsInStock);
59     GetAltPrNam(itemsInStock);
60     GetNumProds(itemsInStock);
61     GetNameNumb(itemsInStock);
62     GetAvgPrice(itemsInStock);
63     GetSumWeigt(itemsInStock);
64     GetSumPrice(itemsInStock);
65     GetNameMake(itemsInStock);
66     GetProdBalt(itemsInStock);
67     Console.ReadLine();
68 }

```

Рис. 1.3. Исходный код метода **Main()** консольного приложения

```

64 // --- Метод получения всех данных о товарах ---
65 static void GetAllProds(Product[] products)
66 {
67     Console.WriteLine("\n1. Все данные о товарах на складе:");
68     var all = from p in products select p;
69     foreach (var a in all)
70     { Console.WriteLine(a.ToString()); }
71 }
72
73 // --- Метод получения наименований всех товаров в алфавитном порядке ---
74 static void GetAllNames(Product[] products)
75 {
76     Console.WriteLine("\n2. Все наименования товаров на складе (по алфавиту):");
77     var names = from p in products orderby p.Name select p.Name;
78     foreach (var n in names)
79     { Console.WriteLine("- " + n); }
80 }
81
82 // --- Метод получения товаров с количеством более 50 ---
83 static void GetProdOver(Product[] products)
84 {
85     Console.WriteLine("\n3. Все товары с количеством более 50:");
86     var overStock = from p in products where p.NumberInStok > 25 select p;
87     foreach (var os in overStock)
88     { Console.WriteLine(os.ToString()); }
89 }
90
91 // --- Метод получения наименований товаров Алтайпродукт с ценой менее 80 руб.
92 static void GetAltPrNam(Product[] products)
93 {
94     Console.WriteLine("\n4. Все товары Алтайпродукт с ценой менее 80 руб.:");
95     var altNames = from p in products
96                     where p.ProducedBy == "ООО Алтайпродукт" && p.Price < 80
97                     select p.Name;
98     foreach (var an in altNames)
99     { Console.WriteLine("- {0}", an); }
100 }
101
102 // --- Метод получения числа наименований товаров весом от 250 до 500 гр.
103 static void GetNumProds(Product[] products)
104 {
105     Console.WriteLine("\n5. Число наименований товаров весом от 250 до 500 гр.:");
106     var nameProd = from p in products
107                     where p.Weight > 250 && p.Weight < 500
108                     select p.Name;
109     int numProds = nameProd.Count();
110     Console.WriteLine("Всего {0} наименований.", numProds);
111 }

```

Рис. 1.4. Исходный код методов, выполняющих запросы LINQ (часть 1)

```

102 // --- Метод получения числа наименований товаров весом от 250 до 500 гр.
103 static void GetNumProds(Product[] products)
104 {
105     Console.WriteLine("\n5. Число наименований товаров весом от 250 до 500 гр.:");
106     var nameProd = from p in products
107                     where p.Weight > 250 && p.Weight < 500
108                     select p.Name;
109     int numProds = nameProd.Count();
110     Console.WriteLine("Всего {0} наименований.", numProds);
111 }
112
113 // --- Метод получения наименования и количества товара (по убыванию) ---
114 static void GetNameNumb(Product[] products)
115 {
116     Console.WriteLine("\n6. Наименование и количество товара (по убыванию):");
117     var nameNumb = from p in products
118                     orderby p.NumberInStok descending
119                     select p;
120     foreach (var nn in nameNumb)
121     { Console.WriteLine("- {0}, {1} шт.", nn.Name, nn.NumberInStok); }
122 }
123
124 // --- Метод нахождения наибольшей, наименьшей и средней цены товаров Алтайпродукт ---
125 static void GetAvgPrice(Product[] products)
126 {
127     Console.WriteLine("\n7. Средняя, наибольшая и наименьшая цены товаров Алтайпродукт:");
128     var prAltPrice = from p in products
129                      where p.ProducedBy == "ООО Алтайпродукт"
130                      select p.Price;
131     Console.WriteLine("- средняя цена: {0:f2} руб.;\n" +
132                      "- наибольшая цена: {1:f2} руб.;\n" +
133                      "- наименьшая цена: {2:f2} руб.",
134                      prAltPrice.Average(), prAltPrice.Max(), prAltPrice.Min());
135 }
136
137 // --- Метод нахождения суммарного веса товаров на складе ---
138 static void GetSumWeigt(Product[] products)
139 {
140     Console.WriteLine("\n8. Суммарный вес всех товаров:");
141     var all = from p in products select p;
142     double sumWeight = 0;
143     foreach (var a in all)
144     { sumWeight += a.NumberInStok * a.Weight; }
145     Console.WriteLine("Всего {0:f2} кг.", sumWeight / 1000);
146 }
147
148 // --- Метод нахождения общей стоимости товаров каждого наименования ---
149 static void GetSumPrice(Product[] products)
150 {
151     Console.WriteLine("\n9. Общая стоимость товаров каждого наименования:");
152     var all = from p in products orderby p.Name select p;
153     foreach (var a in all)
154     {
155         Console.WriteLine("- {0}, {1:f2} руб.", a.Name,
156                           a.NumberInStok * a.Price);
157     }
158 }

```

Рис. 1.5. Исходный код методов, выполняющих запросы LINQ (часть 2)

```

167 // --- Метод нахождения наименований товаров, сгруппированных по производителям ---
168 static void GetNameMake(Product[] products)
169 {
170     Console.WriteLine("\n10. Наименования товаров, сгруппир. по производителям:");
171     var groups = from p in products
172                  group p by p.ProducedBy;
173     // Цикл для выбора каждой группы group из списка групп groups
174     foreach (var group in groups)
175     {
176         Console.WriteLine("Товары фирмы {0}:", group.Key);
177         // Цикл для выбора каждого элемента elem группы group
178         foreach (var elem in group)
179         { Console.WriteLine("- " + elem.Name); }
180     }
181 }
182
183 // --- Метод нахождения наименований и количества товаров фирм ОАО Балтпром или
184 // ЗАО Увелка с указанием данных по стеллажам, на которых они хранятся (исп. join) ---
185 static void GetProdBalt(Product[] products)
186 {
187     // Массив объектов класса Store
188     Store[] stores = new[] {
189         new Store { StoreID = "024",
190                   NumberOfFlour = 4,
191                   MaxWeight = 10000
192         },
193         new Store { StoreID = "028",
194                   NumberOfFlour = 4,
195                   MaxWeight = 10000
196         },
197         new Store { StoreID = "017",
198                   NumberOfFlour = 6,
199                   MaxWeight = 12000
200         }
201     };
202
203     Console.WriteLine("\n11. Наименования и количество товаров фирм ОАО Балтпром\n" +
204                       "или ЗАО Увелка с указанием данных по стеллажам, на которых они хранятся");
205     var result = from p in products
206                  where p.ProducedBy == "ОАО Балтпром" || p.ProducedBy == "ЗАО Увелка"
207                  join s in stores on p.StoreID equals s.StoreID
208                  select new
209                  {
210                      name = p.Name,
211                      numb = p.NumberInStok,
212                      stID = p.StoreID,
213                      numF = s.NumberOfFlour,
214                      maxW = s.MaxWeight
215                  };
216     Console.WriteLine("{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
217                       "Наименование", "Количество", "Код стел.", "Число ярусов",
218                       "Макс. груз, кг");
219     foreach (var r in result)
220     {
221         Console.WriteLine("{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
222                           r.name, r.numb, r.stID, r.numF, r.maxW);
223     }
224 }

```

Рис. 1.6. Исходный код методов, выполняющих запросы LINQ (часть 3)

```

Выполнение запросов LINQ к массиву объектов
- Крупа гречневая, 42 шт.;
- Говядина тушеная, 30 шт.;

7. Средняя, наибольшая и наименьшая цены товаров Алтайпродукт:
- средняя цена: 64,50 руб.;
- наибольшая цена: 72,50 руб.;
- наименьшая цена: 56,50 руб.

8. Суммарный вес всех товаров:
Всего 89,93 кг.

9. Общая стоимость товаров каждого наименования:
- Говядина тушеная, 2880,00 руб.;
- Крупа гречневая, 3045,00 руб.;
- Крупа рисовая, 2244,55 руб.;
- Кукуруза консервированная, 3805,60 руб.;
- Макароны, 2712,00 руб.;

10. Наименования товаров, сгруппир. по производителям:
Товары фирмы ООО Алтайпродукт:
- Крупа гречневая
- Макароны
Товары фирмы ОАО Балтпром:
- Кукуруза консервированная
- Говядина тушеная
Товары фирмы ЗАО Увелка:
- Крупа рисовая

11. Наименования и количество товаров фирм ОАО Балтпром
или ЗАО Увелка с указанием данных по стеллажам, на которых они хранятся
:
Наименование:Количество: Код стел.:Число ярусов: Макс. груз, кг:
:Кукуруза консервированная: 67: 017: 6: 12000:
: Крупа рисовая: 53: 024: 4: 10000:
: Говядина тушеная: 30: 017: 6: 12000:

```

Рис. 1.7. Результат работы консольного приложения □

1.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Создать консольное приложение на языке C# с заданным классом (табл. 1.1), а также массив объектов этого класса. Число элементов массива должно быть не менее пяти.
3. Выполнить заданные запросы LINQ к массиву объектов (табл. 1.1). Для запроса №6 следует использовать операцию **group**, а для запроса №7 – операцию **join**.
4. Оформить и защитить отчет по лабораторной работе.

Таблица 1.1

Варианты заданий для создания массивов объектов и запросов к ним

№ вар.	Данные для создания массивов объектов и запросов к ним
1	<i>Студент.</i> Код, ФИО, группа, пол, дата рождения, средний бал, код научного руководителя.

№ вар.	Данные для создания массивов объектов и запросов к ним
	<p>Научный руководитель. Код, ФИО, должность.</p> <ol style="list-style-type: none"> 1. Данные по студентам мужского пола. 2. ФИО студентов с датой рождения «дата». 3. Число студентов, у которых средний бал более «бал». 4. ФИО и даты рождения студентов группы «группа». 5. Общий средний бал для всех студентов группы «группа». 6. Все студенты, сгруппированные по студенческим группам (group). 7. ФИО студента и его группа с указанием ФИО и должности научного руководителя (join).
2	<p>Автомобиль. Код, фирма, модель, год выпуска, цена, расход топлива, код магазина.</p> <p>Магазин. Код, название, телефон.</p> <ol style="list-style-type: none"> 1. Данные по автомобилям фирмы «фирма». 2. Модели автомобилей с годом выпуска «год выпуска». 3. Число автомобилей с ценой от «цена» до «цена». 4. Фирмы и модели автомобилей с расходом топлива менее «...». 5. Средняя цена автомобиля фирмы «фирма». 6. Все автомобили, сгруппированные по коду магазина (group). 7. Фирмы и модели автомобилей с указанием названия и телефона магазина (join).
3	<p>Сотрудник. Код, ФИО, пол, дата рождения, должность, зарплата, код отдела.</p> <p>Отдел. Код, название, телефон.</p> <ol style="list-style-type: none"> 1. Данные по сотрудникам, занимающим должность «должность». 2. ФИО сотрудников женского пола. 3. Число сотрудников с датой рождения «дата». 4. ФИО и должности сотрудников с зарплатой от «...» до «...». 5. Суммарная зарплата сотрудников с должностью «должность». 6. Все сотрудники, сгруппированные по коду отдела (group). 7. ФИО и даты рождения сотрудников с указанием названия и телефона отдела (join).
4	<p>Книга. Код, название, автор, цена, число страниц, год издания, код стеллажа.</p> <p>Стеллаж. Код, число полок, название отдела.</p> <ol style="list-style-type: none"> 1. Данные по книгам автора «автор». 2. Названия книг, изданных в «год издания». 3. Число книг с ценой от «цена» до «цена». 4. Авторы и названия книг с числом страниц более «число страниц». 5. Средняя цена одной книги автора «автор». 6. Все книги, сгруппированные по коду стеллажа (group). 7. Названия и число страниц книг с указанием числа полок стеллажа

№ вар.	Данные для создания массивов объектов и запросов к ним
	и отдела, в котором он расположен (join).
5	<p>Учебная дисциплина. Код, название, ФИО преподавателя, форма контроля, семестр, число часов, код специальности.</p> <p>Специальность. Код, название, факультет.</p> <ol style="list-style-type: none"> 1. Данные по дисциплинам за семестр «семестр». 2. Названия дисциплин с формой итогового контроля «...». 3. Число дисциплин с количеством часов от «...» до «...». 4. ФИО преподавателей и названия дисциплин за семестр «...». 5. Общее число часов по дисциплинам преподавателя «фио». 6. Все дисциплины, сгруппированные по коду специальности (group). 7. Названия и семестры дисциплин с указанием специальности и факультета (join).
6	<p>Банковский вклад. Код, номер счета, ФИО вкладчика, дата вклада, сумма, процент, код банка.</p> <p>Банк. Код, название, адрес центрального офиса.</p> <ol style="list-style-type: none"> 1. Данные по всем вкладам, сделанным «дата». 2. Счета, на которых размещены вклады размером более «...». 3. Число вкладов, по которым процент составляет от «...» до «...». 4. Номер счета и ФИО вкладчика для вкладов с суммой менее «...». 5. Средний процент по вкладам вкладчика «фио». 6. Все вклады, сгруппированные по вкладчикам (group). 7. Номера счетов и суммы вкладов с указанием названия и адреса банка (join).
7	<p>Предмет обуви. Код, наименование, производитель, число пар, размер, цена, код отдела.</p> <p>Отдел. Код, название, число сотрудников.</p> <ol style="list-style-type: none"> 1. Данные по всей обуви фирмы «...». 2. Наименования обуви с ценой более «цена». 3. Число наименований обуви с размером от «...» до «...». 4. Производитель и наименование обуви с количеством менее «...». 5. Суммарная стоимость обуви по каждому наименованию. 6. Вся обувь, сгруппированная по кодам отделов (group). 7. Наименования и цены обуви с указанием названия отдела и числа сотрудников (join).
8	<p>Ноутбук. Код, фирма, модель, процессор, объем памяти, цена, код магазина.</p> <p>Магазин. Код, название, адрес.</p> <ol style="list-style-type: none"> 1. Данные по всем ноутбукам с процессором «...». 2. Модели ноутбуков фирмы «...». 3. Число моделей с ценой от «...» до «...».

№ вар.	Данные для создания массивов объектов и запросов к ним
	4. Модель и производитель ноутбуков с объемом памяти более «...». 5. Средняя цена ноутбуков фирмы « <i>фирма</i> ». 6. Все ноутбуки, сгруппированные по коду магазина (group). 7. Модель и цена ноутбуков с указанием названия и адреса магазина (join).
9	Билет на междугородный автобус. Код, рейс, пункт назначения, время отправления, длительность, номер места, код автобуса. Автобус. Код, модель, число посадочных мест. 1. Данные по всем билетам с пунктом назначения «...». 2. Рейсы с временем отправления «...». 3. Число билетов с номерами мест от «...» до «...». 4. Пункты назначения и рейсы с длительностью более «...». 5. Средняя длительность рейсов в пункт назначения «...». 6. Все билеты, сгруппированные по пункту назначения (group). 7. Рейсы и места с указанием модели автобуса и числа мест в нем (join).
10	Квартира. Код, дом, номер, этаж, площадь, цена, код агентства. Агентство недвижимости. Код, название, телефон. 1. Данные по всем продаваемым квартирам. 2. Номера квартир, продаваемых в доме «дом». 3. Число квартир с ценой не более «цена». 4. Дома и номера квартир, расположенных на этажах от «...» до «...». 5. Средняя стоимость квартир с площадью менее «площадь». 6. Все квартиры, сгруппированные по коду агентства (group). 7. Код и цена квартиры с указанием названия агентства и его телефона (join).
11	Заказ на перевозку груза. Код, номер, дата, адрес доставки, вес груза, стоимость перевозки, код водителя. Водитель. Код, ФИО, дата рождения. 1. Данные по всем заказам на перевозку. 2. Номера заказов, сделанных «дата». 3. Число заказов со стоимостью перевозки более «стоимость». 4. Даты и адреса доставки грузов весом от «вес» до «вес». 5. Общий вес груза, переведенный «дата». 6. Все заказы, сгруппированные по коду водителя (group). 7. Номера и стоимости заказов с указанием ФИО и даты рождения водителя (join).
12	Спортсмен. Код, ФИО, вид спорта, дата рождения, пол, рост, вес, код тренера. Тренер. Код, ФИО, звание. 1. Данные по всем спортсменам.

№ вар.	Данные для создания массивов объектов и запросов к ним
	2. ФИО спортсменов мужского пола. 3. Число спортсменов с весом от «вес» до «вес». 4. Даты рождения и ФИО спортсменов, имеющих рост более «...». 5. Средний рост спортсменов по «вид спорта». 6. Все спортсмены, сгруппированные по видам спорта (group). 7. ФИО и дата рождения спортсмена с указанием ФИО и звания тренера (join).

1.4. Контрольные вопросы

1. Для чего предназначена технология LINQ?
2. Какие выделяют разновидности LINQ по источнику данных?
3. Что такое неявно типизированные переменные и как они используются в LINQ?
4. Из каких этапов состоит выполнение запроса LINQ?
5. Каков синтаксис для простейшего выражения запроса LINQ?
6. Как задается фильтрация данных в выражении запроса LINQ?
7. Каким образом задается сортировка результатов запроса LINQ?
8. Как можно сгруппировать данные запроса LINQ по определенному ключу?
9. Для чего предназначена операция **join** в выражении запроса LINQ?
10. Что называют анонимным типом и как он записывается в выражении запроса LINQ?

2. ОСНОВЫ РАБОТЫ С XML-ДОКУМЕНТАМИ С ПОМОЩЬЮ LINQ TO XML

2.1. Цель работы

Цель работы – приобрести умение работать с документами XML с помощью технологии LINQ to XML.

Работа рассчитана на 4 часов.

2.2. Основные теоретические сведения

2.2.1. Основные классы LINQ to XML. Осевые методы LINQ to XML

Основные классы LINQ to XML.

Программная модель LINQ to XML позволяет выражать структуру XML-данных в коде на языках Visual Basic и C# и предлагает простой способ создания, манипулирования и сохранения XML-данных. Также с помощью выражений запросов LINQ можно легко извлекать информацию из XML-документов.

Пространство имен **System.Xml.Linq** содержит классы, представляющие различные аспекты XML-документа (элементы, атрибуты, пространства имен XML и др.) (рис. 2.1).

Основными классы пространства имен **System.Xml.Linq** являются:

- **XDocument** – представляет целиком весь XML-документ;
- **XElement** – представляет определенный элемент внутри XML-документа;
- **XAttribute** – представляет атрибут определенного элемента;
- **XDeclaration** – представляет открывающее объявление XML-документа;
- **XComment** – представляет комментарий XML;
- **XName** – представляет имя элемента или атрибута XML;
- **XNamespace** – представляет пространство имен XML;
- **XNode** – представляет узел на дереве XML-документа (элемент, атрибут, комментарий и др.).

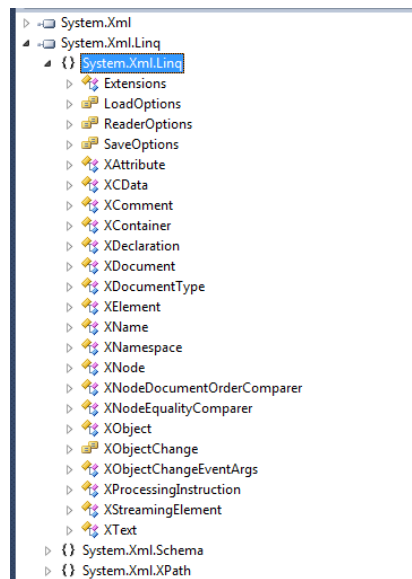


Рис. 2.1. Отображение классов пространства имен **System.Xml.Linq** в окне обозревателя объектов Visual Studio 2010

В качестве свойства класса **XDocument** можно отметить **Root**, которое получает корневой элемент данного документа.

Можно отметить следующие общие методы класса **XDocument**:

- **Load(string fileName)** – создает новый объект **XDocument** из файла с именем **fileName** (статический метод);
- **Parse(string text)** – создает новый объект **XDocument** из строки **text** (статический метод);
- **Save(string fileName)** – преобразует объект **XDocument** в XML-файл с именем **fileName**;
- **Element(XName name)** – возвращает первый (в порядке следования) дочерний элемент с именем **name**;
- **Elements(XName name)** – возвращает коллекцию дочерних элементов для данного документа; в состав коллекции входят только элементы с именем **name**.

Класс **XElement** обозначает элемент XML и является одним из основных классов в LINQ to XML. Этот класс можно использовать для создания элементов, изменения содержимого элемента, добавления, изменения или удаления дочерних элементов, добавления к элементам атрибутов или преобразования содержимого элемента в текстовую форму.

Некоторые свойства класса **XElement**:

- **Name** – получает или задает имя элемента;
- **Value** – получает или задает текстовое содержимое элемента;
- **Parent** – получает родительский элемент для данного элемента.

Некоторые методы класса **XElement**:

- **Element(XName name)** – возвращает первый дочерний элемент объекта **XElement**, имеющий указанное имя **name**;
- **Attribute(XName name)** – возвращает атрибут **XAttribute**, имеющий имя **name**;
- **Attribute()** – возвращает коллекцию **IEnumerable** всех атрибутов **XAttribute**.

□ *Пример 2.1. Создание XML-документа с помощью консольного приложения на языке C#.*

Требуется, используя классы LINQ to XML, создать с нуля одну ветвь XML-документа, полученного в приложении П.1. Для этого создадим консольное приложение на языке C#.

Исходный код полученного приложения представлен на рис. 2.2.

```

using System.Text;
using System.Xml.Linq;

namespace LinqSozdXmlDoc
{
    class Program
    {
        static void Main(string[] args)
        {
            XDocument xmlDoc =
                new XDocument(
                    new XDeclaration("1.0", "utf-8", "yes"),
                    new XComment("Данные о предоставляемых услугах и их оплате"),
                    new XElement("коммун_услуги",
                        new XElement("дом", new XAttribute("код", "h18"),
                            new XElement("адрес",
                                new XElement("улица", "Волгоградская"),
                                new XElement("номер", "8")
                            )
                        ),
                        new XElement("квартира", new XAttribute("код", "a238"),
                            new XAttribute("номер", "57"),
                            new XElement("площадь", "28"),
                            new XElement("жилец", new XAttribute("код", "c11568"),
                                new XElement("фio", "Костенко Игорь Сергеевич"),
                                new XElement("дата_рожд", "10.11.1978")
                            )
                        ),
                        new XElement("показ_приборов", new XAttribute("дата", "02.05.2013"),
                            new XElement("хол_вода", "19.04", new XAttribute("ед_изм", "м3")),
                            new XElement("гор_вода", "6.89", new XAttribute("ед_изм", "м3")),
                            new XElement("эл_энерг", "39.27", new XAttribute("ед_изм", "квтч"))
                        ),
                        new XElement("плата", new XAttribute("дата", "04.05.2013"),
                            new XElement("всего", "1896.45"),
                            new XElement("пеня", "0")
                        )
                    )
                );

            xmlDoc.Save("komUslug.xml");
        }
    }
}

```

Рис. 2.2. Исходный код консольного приложения

После выполнения метода **Main()** консольного приложения в файл «komUslug.xml» будут записаны данные, показанные на рис. 2.3.


```

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--Данные о предоставляемых услугах и их оплате-->
<коммун_услуги>
  <дом код="h18">
    <адрес>
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес>
    <квартира код="a238" номер="57">
      <площадь>28</площадь>
      <жилец код="c11568">
        <фio>Костенко Игорь Сергеевич</фio>
        <дата_рожд>10.11.1978</дата_рожд>
      </жилец>
      <показ_приборов дата="02.05.2013">
        <хол_вода ед_изм="м3">19.04</хол_вода>
        <гор_вода ед_изм="м3">6.89</гор_вода>
        <эл_энерг ед_изм="квтч">39.27</эл_энерг>
      </показ_приборов>
      <плата дата="04.05.2013">
        <всего>1896.45</всего>
        <пеня>0</пеня>
      </плата>
    </квартира>
  </дом>
</коммун_услуги>

```

Рис. 2.3. XML-код полученного документа □

□ **Пример 2.2. Создание XML-документа из массива объектов с помощью консольного приложения на языке C#.**

Требуется создать XML-документ на основе массива объектов, полученного в примере 1.1.

Для этого добавим в код консольного приложения из примера 2.1 метод **GenerXML()**, приведенный на рис. 2.4. Кроме того, в процедуре **Main()** необходимо добавить вызов этого метода с передачей ему массива объектов:

```
GenerXML(itemsInStock);
```

В результате будет получен XML-документ, приведенный на рис. 2.5.

```

150
157 static void GenerXML(Product[] products)
158 {
159     // Случайная переменная для генерирования кодов
160     Random r = new Random();
161
162     // Генерирование XML-данных из массива products
163     var xmlData = from p in products
164                   select
165                       new XElement("product",
166                                   new XAttribute("id", r.Next()),
167                                   new XAttribute("numberInStock", p.NumberInStok),
168                                   new XElement("name", p.Name),
169                                   new XElement("make", p.ProducedBy),
170                                   new XElement("weight", p.Weight, new XAttribute("unit", "г.")),
171                                   new XElement("price", p.Price, new XAttribute("unit", "руб.)))
172                   );
173
174     // Корневой элемент
175     XElement rootElem = new XElement("productStock", xmlData);
176
177     // Документ XML
178     XDocument xmlDoc = new XDocument();
179     xmlDoc.Add(rootElem);
180     xmlDoc.Save("product.xml");
181 }
182

```

Рис. 2.4. Код метода для создания XML-документа из массива объектов

```

<?xml version="1.0" encoding="utf-8"?>
<productStock>
  <product id="1777605012" numberInStock="42">
    <name>Крупа гречневая</name>
    <make>ЗАО Увелка</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">72.5</price>
  </product>
  <product id="979461422" numberInStock="67">
    <name>Кукуруза консервированная</name>
    <make>ООО Славянский консервный комбинат</make>
    <weight unit="г.">340</weight>
    <price unit="руб.">56.35</price>
  </product>
  <product id="2079450253" numberInStock="53">
    <name>Крупа рисовая</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">42.35</price>
  </product>
  <product id="56812781" numberInStock="30">
    <name>Говядина тушеная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">330</weight>
    <price unit="руб.">96</price>
  </product>
  <product id="867299654" numberInStock="48">
    <name>Макароны</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">500</weight>
    <price unit="руб.">56.35</price>
  </product>
</productStock>

```

Рис. 2.5. Код XML-документа, полученного из массива объектов

2.2.2. Осевые методы LINQ to XML. Модификация XML-документов с помощью LINQ to XML

Осевые методы LINQ to XML. Использование XPath.

После загрузки XML-документа к нему можно применить запросы LINQ для поиска коллекций элементов и атрибутов, а также извлечения их значений.

Коллекции элементов и атрибутов XML получают с помощью методов, называемых *осевыми методами* или *осями*. Эти методы можно применять непосредственно к частям дерева XML или узлам либо использовать их для построения более сложных запросов LINQ.

Часть осей являются методами классов **XElement** и **XDocument**. Другие оси являются методами расширений в классе **Extensions**.

Некоторые осевые методы LINQ to XML:

- **Elements(XName name)** – возвращает коллекцию дочерних элементов объекта **XElement**, имеющих указанное имя **name**;
- **Descendants(XName name)** – возвращает коллекцию элементов-потомков объекта **XElement**, имеющих имя **name**;
- **Ancestors()** – возвращает коллекцию элементов-предков объекта **XElement**.

LINQ to XML обеспечивает поддержку **XPathNavigator** через методы расширения в пространстве имен **System.Xml.XPath**. К данным методам относятся:

- **XPathSelectElement(string expr)** – выбирает требуемый элемент с помощью выражения **expr** на языке XPath;
- **XPathSelectElements(string expr)** – выбирает коллекцию элементов с помощью выражения **expr** на языке XPath;
- **XPathValue(string expr)** – вычисляет выражение XPath **expr**.

□ *Пример 2.3. Выполнение запросов LINQ к документу XML с помощью консольного приложения на языке C#.*

Требуется разработать консольное приложение на языке C#, которое с помощью технологии LINQ to XML выполняет заданный набор запросов к XML-документу из приложения П.1.

Запросы к XML-документу:

1. Данные по всем жильцам обслуживаемых домов (в алфавитном порядке).
2. Суммарный расход холодной и горячей воды и электроэнергии.
3. Данные квартирам, расположенным в доме номер 8 по улице «Волгоградская».
4. Число квартир, в которых расход горячей воды менее 100 м³ или расход электроэнергии менее 120 квтч.
5. Квартиры с площадью более 30 м², сгруппированные по обслуживаемым домам.
6. ФИО жильцов дома с кодом «h18», сгруппированные по номерам квартир.
7. Работники ЖKK с указанием адресов домов, которые они обслуживают.
8. Квартиры, в которых размер квартплаты превышает 2000 руб. (с помощью XPath).

Для выполнения запроса №7 необходимо использовать дополнительные XML-данные о работниках ЖKK (код, код дома, ФИО, должность). Для этого в методе, реализующем запрос LINQ, опишем XML-элемент **работники_ЖKK**, в котором создадим три дочерних элемента **работник**. Выражение для этого запроса должно включать операцию **join**.

Для выполнения запроса №8 требуется импортировать в модуль пространство имен **System.Xml.XPath**.

Исходный код метода **Main()** консольного приложения показан на рис. 2.6. Код методов, выполняющих запросы к документу XML, показан на рис. 2.7 – 2.9.

```

4 using System.Linq;
5 using System.Xml.Linq;
6 using System.Xml.XPath;
7
8 namespace ZaprostoLINQtoXML
9 {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             Console.Title = "Выполнение запросов LINQ к XML-документу";
15
16             // Загрузить XML-документ
17             XDocument xmlDoc = XDocument.Load("komUslug.xml");
18
19             Console.WriteLine("***** Результаты запросов LINQ к XML-документу *****");
20
21             // Вызовы процедур, выполняющих запросы к XML-документу
22             GetPersData(xmlDoc);
23             GetWaterElc(xmlDoc);
24             GetAptsData(xmlDoc);
25             GetNumApts(xmlDoc);
26             GetApts30m(xmlDoc);
27             GetPersName(xmlDoc);
28             GetContData(xmlDoc);
29             GetServCost(xmlDoc);
30             GetAparCode(xmlDoc);
31
32             Console.ReadLine();
33         }
34     }
35 }

```

Рис. 2.6. Исходный код метода **Main()** консольного приложения

```

31 // Данные по всем жильцам (в алфавитном порядке)
32 static void GetPersData(XDocument doc)
33 {
34     var result = from elPers in doc.Descendants("желец")
35                  orderby elPers.Element("фио").Value
36                  select elPers;
37
38     Console.WriteLine("\n1. Данные по всем жильцам обслуживаемых домов:");
39     // "Шапка" текстовой таблицы
40     Console.WriteLine("|{0,15}|{1,15}|{2,14}|", "ФИО", "", "Дата рождения");
41     foreach (var r in result)
42     {
43         Console.WriteLine("|{0,30}|{1,14}|", r.Element("фио").Value,
44                             r.Element("дата_рожд").Value);
45     }
46 }
47
48 // Суммарный расход воды и электроэнергии
49 static void GetWaterElc(XDocument doc)
50 {
51     var result = from d in doc.Descendants("показ_приборов")
52                  select d;
53
54     Console.WriteLine("\n2. Суммарный расход воды и электроэнергии:");
55     double sumColWater = 0;
56     double sumHotWater = 0;
57     double sumElcPower = 0;
58     foreach (var r in result)
59     {
60         sumColWater += Convert.ToDouble(r.Element("хол_вода").Value);
61         sumHotWater += Convert.ToDouble(r.Element("гор_вода").Value);
62         sumElcPower += Convert.ToDouble(r.Element("эл_энерг").Value);
63     }
64     Console.WriteLine("- Холодная вода: {0:f2} м3\n" +
65                       "- Горячая вода: {1:f2} м3\n" +
66                       "- Электроэнергия: {2:f2} квтч",
67                       sumColWater, sumHotWater, sumElcPower);
68 }
69
70 // Данные по квартирам, расположенным в доме 8 по улице Волгоградская
71 static void GetAptsData(XDocument doc)
72 {
73     var result = from d in doc.Descendants("дом")
74                  from k in d.Elements("квартира")
75                  from a in d.Elements("адрес")
76                  where a.Element("улица").Value == "Волгоградская" &&
77                        a.Element("номер").Value == "8"
78                  select k;
79
80     Console.WriteLine("\n3. Данные по квартирам в доме 8, ул. Волгоградская:");
81     Console.WriteLine("|{0,6}|{1,12}|", "Номер", "Площадь, м2");
82     foreach (var r in result)
83     {
84         Console.WriteLine("|{0,6}|{1,12}|",
85                             r.Attribute("номер").Value, r.Element("площадь").Value);
86     }
87 }

```

Рис. 2.7. Исходный код процедур выполнения запросов к XML-документу (часть 1)

```

89 // Число квартир, в которых расход горячей воды < 100 м3
90 // или расход электроэнергии < 120 квтч
91 static void GetNumAppts(XDocument doc)
92 {
93     var result = from a in doc.Descendants("квартира")
94                   from p in a.Elements("показ_приборов")
95                   where Convert.ToDouble(p.Element("гор_вода").Value) < 100 ||
96                        Convert.ToDouble(p.Element("эл_энерг").Value) < 120
97                   select a;
98     int numAp = result.Count();
99     Console.WriteLine("\n4. Число квартир с расходом горячей воды < 100 м3" +
100                      " или расходом электроэнергии < 120 квтч: " + numAp);
101 }
102
103 // Квартиры с площадью более 30 м2, сгруппированные по домам
104 static void GetAppts30m(XDocument doc)
105 {
106     var groups = from d in doc.Descendants("дом")
107                   from a in d.Elements("квартира")
108                   where (int)a.Element("площадь") > 30
109                   group a by d.Attribute("код");
110
111     Console.WriteLine("\n5. Квартиры с площадью более 30 м2 (сгруп. по домам):");
112     // Цикл для выбора каждой группы group из списка групп groups
113     foreach (var group in groups)
114     {
115         Console.WriteLine("Квартиры в доме " + group.Key);
116         // Цикл для выбора каждого элемента elem группы group
117         foreach (var elem in group)
118         {
119             Console.WriteLine("- {0}, №{1}", elem.Attribute("код"),
120                               elem.Attribute("номер").Value);
121         }
122     }
123 }
124
125 // ФИО жильцов дома с кодом h18, сгруппированные по квартирам
126 static void GetPersName(XDocument doc)
127 {
128     var groups = from h in doc.Descendants("дом")
129                   from a in h.Elements("квартира")
130                   from p in a.Elements("жилец")
131                   where h.Attribute("код").Value == "h18"
132                   group p by a.Attribute("номер").Value;
133
134     Console.WriteLine("\n6. ФИО жильцов дома с кодом h18 (сгруп. по квартирам):");
135     //
136     foreach (var group in groups)
137     {
138         Console.WriteLine("Жильцы квартиры №{0}:", group.Key);
139         //
140         foreach (var elem in group)
141         {
142             Console.WriteLine("- {0}", elem.Element("фio").Value);
143         }
144     }
145 }

```

Рис. 2.8. Исходный код процедур выполнения запросов к XML-документу (часть 2)

```

173 // Работники ЖКК с указанием адресов домов, которые они обслуживают (join)
174 static void GetComWorks(XDocument doc)
175 {
176     XElement commWorkers = new XElement("работники_ЖКК",
177         new XElement("работник", new XAttribute("код", "e012"),
178             new XAttribute("код_дома", "h18"),
179             new XElement("фio", "Мишутин Дмитрий Олегович"),
180             new XElement("должность", "дворник")
181         ),
182         new XElement("работник", new XAttribute("код", "e067"),
183             new XAttribute("код_дома", "h18"),
184             new XElement("фio", "Сидоренко Татьяна Николаевна"),
185             new XElement("должность", "уборщик")
186         ),
187         new XElement("работник", new XAttribute("код", "e104"),
188             new XAttribute("код_дома", "h72"),
189             new XElement("фio", "Ермолова Галина Степановна"),
190             new XElement("должность", "уборщик")
191         )
192     );
193
194     var result = from a in doc.Descendants("дом")
195                  from s in a.Elements("адрес")
196                  join b in commWorkers.Elements("работник")
197                      on a.Attribute("код").Value equals b.Attribute("код_дома").Value
198                  select new
199                  {
200                      name = b.Element("фio").Value,
201                      disg = b.Element("должность").Value,
202                      strt = s.Element("улица").Value,
203                      numb = s.Element("номер").Value
204                  };
205
206     Console.WriteLine("\n7. Работники ЖКК и обслуживаемые ими дома:");
207     Console.WriteLine("{0,30}|{1,10}|{2,20}|{3,10}|",
208         "ФИО", "Должность", "Улица", "Номер дома");
209     foreach (var r in result)
210     {
211         Console.WriteLine("{0,30}|{1,10}|{2,20}|{3,10}|",
212             r.name, r.disg, r.strt, r.numb);
213     }
214 }
215
216 // Жильцы квартир, в которых пеня > 0 руб. (с помощью XPath)
217 static void GetAparCode(XDocument doc)
218 {
219     var result = doc.XPathSelectElements("//квартира[плата/пеня>0]/жилец/фio");
220
221     Console.WriteLine("\n9. Жильцы квартир, в которых пеня > 0 руб.");
222     foreach (var r in result)
223     {
224         Console.WriteLine("- " + r.Value);
225     }
226 }

```

Рис. 2.9. Исходный код процедур выполнения запросов к XML-документу (часть 3)


```

Выполнение запросов LINQ к XML-документу
***** Результаты запросов LINQ к XML-документу *****

1. Данные по всем жильцам обслуживаемых домов:
   :      ФИО      :      Дата рождения:
   : Костенко Игорь Сергеевич : 10.11.1978:
   : Курганков Георгий Михайлович : 26.02.1972:
   : Соловьев Дмитрий Андреевич : 22.07.1988:
   : Соловьева Елена Николаевна : 03.10.1989:

2. Суммарный расход воды и электроэнергии:
- Холодная вода: 596,49 м3
- Горячая вода: 231,31 м3
- Электроэнергия: 374,72 квтч

3. Данные по квартирам в доме 8, ул. Волгоградская:
   : Номер: Площадь, м2:
   : 57: 28:
   : 59: 42:

4. Число квартир с расходом горячей воды < 100 м3 или расходом электроэнергии < 120 квтч: 2

5. Квартиры с площадью более 30 м2 (сгруп. по домам):
Квартиры в доме код="h18"
- код="a236", №59
Квартиры в доме код="h72"
- код="a702", №23

6. ФИО жильцов дома с кодом h18 (сгруп. по квартирам):
Жильцы квартиры №57:
- Костенко Игорь Сергеевич
Жильцы квартиры №59:
- Соловьев Дмитрий Андреевич
- Соловьева Елена Николаевна

7. Работники ЖК и обслуживаемые ими дома:
   :      ФИО      :      Должность:      Улица:Номер дома:
   : Мишутин Дмитрий Олегович : дворник:      Волгоградская: 8:
   : Сидоренко Татьяна Николаевна : уборщик:      Волгоградская: 8:
   : Ермолова Галина Степановна : уборщик:Сибиряков-Гвардейцев: 59:

8. Жильцы квартир, в которых пеня > 0 руб.
- Курганков Георгий Михайлович

```

Рис. 2.10. Результат работы консольного приложения

2.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать консольное приложение на языке C#, в котором с помощью классов **System.Xml.Linq** «с нуля» создается одна ветвь XML-документа.
3. Добавить метод в приложение из работы №1 (запросы к массиву объектов), который с помощью запроса LINQ преобразовывает данные из массива объектов в XML.
4. Разработать консольное приложение на языке C#, которое обеспечивает выполнение заданных запросов (табл. 2.2) к XML-документу. Запрос №3 требуется сделать с использованием операции **group**, запрос №4 – с помощью **join**, а запрос №5 – с помощью выражения на языке XPath.

5. Оформить и защитить отчет по лабораторной работе.

Таблица 2.2

Варианты заданий для разработки запросов LINQ к XML-документу

№ вар.	Данные для разработки запросов LINQ к XML-документу
1	<i>Ресторан.</i> 1. Дата, время и стол для заказов, которые не были выполнены. 2. Число заказов, сделанных «Дата» со стола «Стол». 3. Данные по ингредиентам, сгруп. по блюдам с ценой более «...». 4. Заказанные блюда с указанием кода заказа (join). 5. Ингредиенты блюда «Название», с объемом более «...» (XPath).
2	<i>Успеваемость студентов.</i> 1. Названия и число часов для дисциплин с формой контроля «...». 2. Среднее число часов по всем дисциплинам. 3. Данные по студентам пола «Пол», сгруппированные по группам. 4. Оценки студентов с указанием группы по дисц. «...» (join). 5. Группы, в которых студентов муж. пола <, чем женского (XPath).
3	<i>Книжный магазин.</i> 1. Названия и авторы книг ценой менее «Цена». 2. Суммарная цена книг, изданных в «Издательство». 3. Названия книг автора «Автор», сгруппиров. по годам издания. 4. ФИО сотрудника с указанием названия отдела (join). 5. Книги с числом страниц менее «...» и ценой более «...» (XPath).
4	<i>Кинотеатр.</i> 1. Дата и время начала сеансов с ценой билета более «Цена». 2. Число показанных фильмов, снятых режиссером «Режиссер». 3. Данные о занятых местах, сгруппированные по сеансам. 4. Список сеансов с указанием названия фильма (join). 5. Сеансы, на которые число занятых мест было более «...» (XPath).
5	<i>Организационная структура.</i> 1. Фамилия, имя и отчество сотрудников с датой рождения «Дата». 2. Число сотрудников пола «Пол», занимающих должность «...». 3. Данные по сотрудникам с окладом > «...», сгруппир. по отделам. 4. Список должностей с указанием названия отдела (join). 5. Все сотрудники муж. пола, имеющие оклад менее «...» (XPath).
6	<i>Обувной магазин.</i> 1. Наименование, размер и цена обуви, выпущенной фирмой «...». 2. Суммарная стоимость обуви с размером от «...» до «...». 3. Данные по аксессуарам ценой < «Цена», сгруппиров. по отделам.

№ вар.	Данные для разработки запросов LINQ к XML-документу
	4. Список обуви с указанием названия отдела (join). 5. Отделы, в кот. скидка на обувь составляет менее «...» (XPath).
7	Перевозка грузов. 1. Даты рейсов и адреса доставки грузов с весом более «Вес». 2. Число рейсов, в которых вес груза составил от «...» до «...». 3. ФИО водителей, сгруппир. по автомобилям с «Тип кузова». 4. Список водителей с указанием рейса (join). 5. Рейсы, в которых участвовали два и более водителя (XPath).
8	Поликлиника. 1. Даты обращений и болезни с результатом лечения «Результат». 2. Число обращений с продолжит. болезни от «...» до «...». 3. ФИО пациентов пола «Пол», сгруппированные по врачам. 4. Список обращений с указанием ФИО врача (join). 5. Пациенты, которые сделали более «...» обращений (XPath).
9	Банк. 1. Вид и сумма вкладов с процентной ставкой более «Процент». 2. Число вкладов, по которым процент составляет от «...» до «...». 3. Данные по вкладам размером более «...», сгруппир. по клиентам. 4. Список вкладов с указанием данных по вкладчику (join). 5. Клиенты, сделавшие более одного вклада (XPath).
10	Аэропорт. 1. Коды и модели самолетов авиакомпании «...». 2. Число рейсов, в которых длителън. составляет от «...» до «...». 3. Данные по пассажирам, сгруппированные по самолетам. 4. Список пассажиров с указанием рейса (join). 5. Самолеты, в которых число мест класса «...» больше «...» (XPath).
11	Гостиница. 1. Коды и названия гостиничных номеров вместимостью более «...». 2. Число клиентов, с датой заселения «...» и датой выезда «...». 3. ФИО клиентов, сгруппированные по номерам. 4. Список клиентов с указанием обслужив. сотрудников (join). 5. Номера, в которые заселено более одного клиента (XPath).
12	Продажа легковых автомобилей. 1. Модели и цены автомобилей фирмы «Фирма». 2. Средняя цена автомобиля с кузовом «Тип». 3. Данные по автомобилям мощностью более «», сгрупп по фирмам. 4. Список автомобилей с указанием ответств. сотрудника (join). 5. Все автомобили с мощностью двигателя более «...» (XPath).

2.4. Контрольные вопросы

1. Для чего предназначена технология LINQ to XML?
2. Каковы основные классы пространства имен **System.Xml.Linq**?
3. Какие методы класса **XDocument** используются для загрузки XML-документа в память и для его сохранения?
4. Каким образом с помощью классов LINQ to XML строится дерево XML?
5. Что понимают под осевыми методами LINQ to XML?
6. Как с помощью осевых методов получить доступ к наборам дочерних элементов и элементов-потомков?
7. Как в LINQ to XML можно использовать выражения XPath для получения данных из XML-документа?
8. Какие методы класса **XElement** позволяют редактировать XML-данные, добавлять и удалять элементы и атрибуты?

3. ОСНОВЫ СОЗДАНИЯ ПРИЛОЖЕНИЙ WPF С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА XAML

3.1. Цель работы

Цель работы – приобрести умение разрабатывать приложения на платформе MS Windows Presentation Foundation (WPF) с использованием языка разметки XAML.

Работа рассчитана на 4 часа.

3.2. Основные теоретические сведения

3.2.1. Назначение и возможности WPF. Создание приложений WPF в Visual Studio

Назначение и возможности WPF.

Windows Presentation Foundation (WPF) – это набор средств для построения пользовательских интерфейсов, появившийся в .NET Framework, начиная с версии 3.0. Основная цель WPF состоит в интеграции и унификации множества разрозненных технологий в единую программную модель.

Платформа WPF спроектирована для .NET под влиянием таких современных технологий отображения, как HTML5 и Flash. В настоящее время WPF является альтернативой таким традиционными графическим интерфейсам на платформе .NET, как Windows Forms и GDI+.

Возможности разработки приложений WPF включают модель приложения, элементы управления, язык разметки XAML, стили и шаблоны, двухмерную и трехмерную графику, привязки данных, анимацию, использование звука и видео.

С помощью WPF можно создавать широкий спектр клиентских приложений. На рис. 3.1. показан пример одного из таких приложений – Contoso Healthcare Sample Application, которое предназначено для просмотра медицинских карт пациентов в учреждениях здравоохранения.

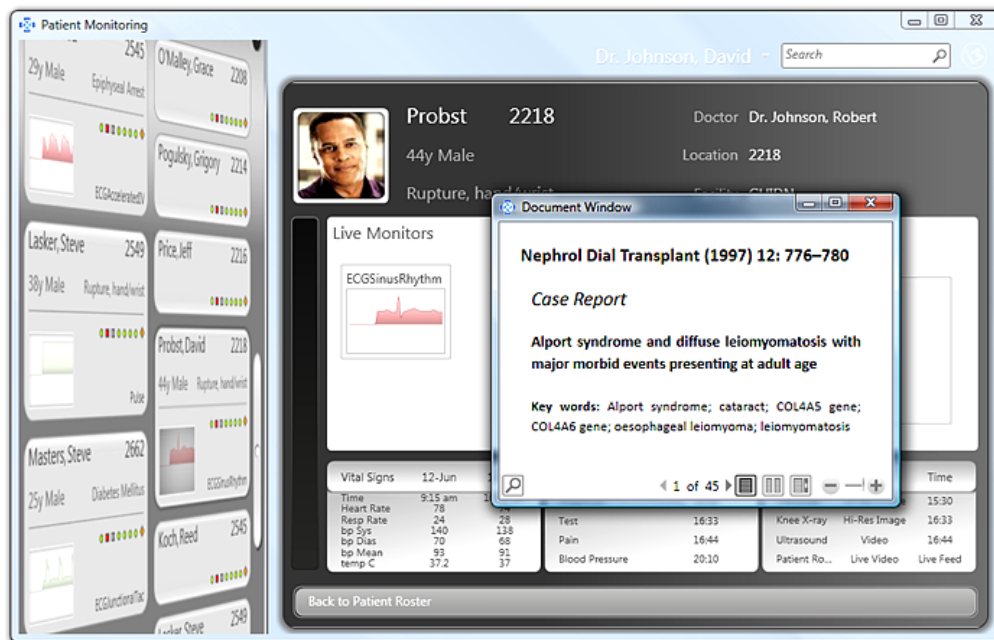


Рис. 3.1. Пример окна приложения WPF (Contoso Healthcare Sample Application)

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

WPF позволяет строить приложения XBAP, которые работают внутри Web-браузера. Кроме того, WPF является основой для технологии Silverlight, предназначенной для разработки многофункциональных Web-приложений.

Основными классами для любого приложения WPF являются классы **Application** и **Window**.

Класс **System.Windows.Application** представляет экземпляр работающего приложения WPF. В этом классе предусмотрен метод **Run()** для запуска приложения, а также событие **Exit** для выхода из приложения. Внутри класса **Application** определяется точка входа программы (метод **Main()**).

Одним из свойств класса **Application** является свойство **Windows**, предоставляющего доступ к коллекции **WindowCollection**, в которой представлены все загруженные в память окна для данного приложения WPF.

Класс **System.Windows.Window** представляет одиночное окно, включая все диалоговые окна.

Построение приложений WPF с помощью Visual Studio 2010.

Интегрированная среда MS Visual Studio предлагает большой набор средств для создания приложений WPF. Возможность разработки приложений WPF в Visual Studio поддерживается, начиная с версии 2008.

Альтернативой Visual Studio для построения WPF-приложений является MS Expression Blend.

В диалоговом окне **New Project** (Создать проект) среды Visual Studio определен набор шаблонов для проектов WPF, которые размещены в узле Windows (рис. 3.2).

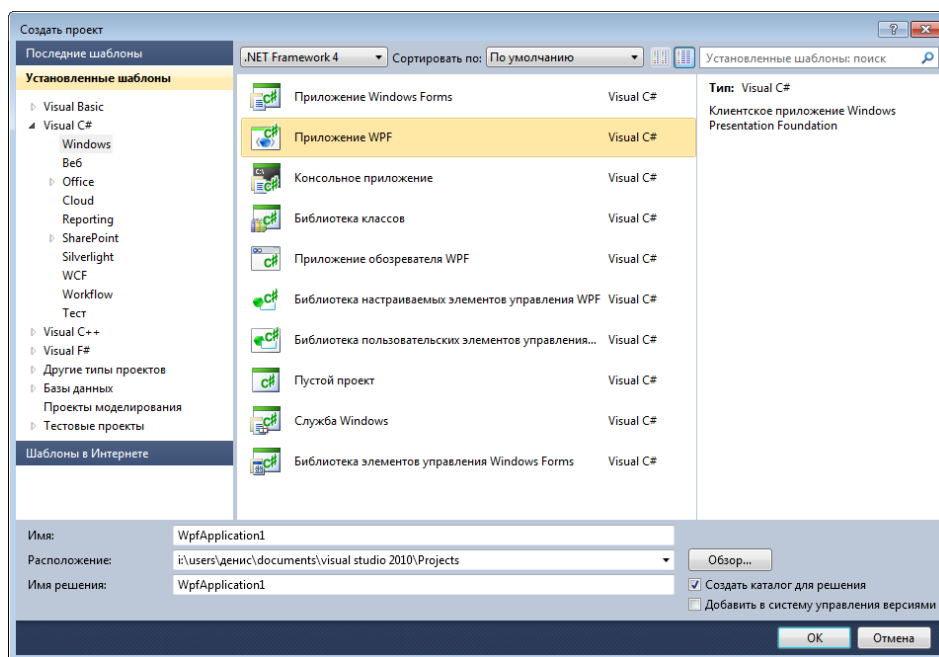


Рис. 3.2. Окно создания нового проекта Visual Studio

В Visual Studio 2010 доступны следующие варианты шаблонов для WPF:

- **WPF Application** – настольное приложение WPF, которое запускается на локальном компьютере в виде одного или нескольких окон; данный вид приложений может использовать Web-подобную модель работы, при которой в каждое из окон мо-

гут загружаться страницы с элементами графического интерфейса;

- **WPF Browser Application** – приложение обозревателя WPF (браузерное приложение XAML – XBAP), которое открывается через Web-браузер при переходе пользователя по заданному URL-адресу; в настоящее время поддержка приложений XBAP осуществляется только браузерами Internet и Explorer Firefox;

- **WPF User Control Library** – библиотека пользовательских элементов управления WPF;

- **WPF Custom Control Library** – библиотека настраиваемых элементов управления WPF.

На рис. 3.3 показан общий вид окна Visual Studio 2010 для разработки проекта приложения WPF. Основную часть окна проекта занимают визуальный конструктор и текстовый редактор XAML.

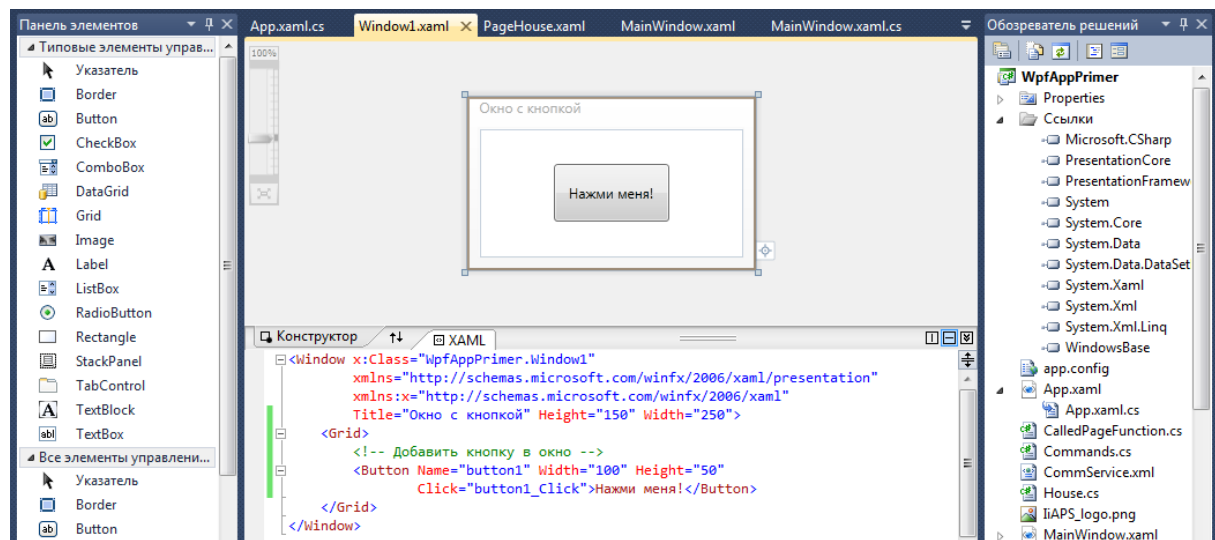


Рис. 3.3. Окно Visual Studio 2010 с открытым проектом приложения WPF

Кроме стандартных панели элементов (**ToolBox**), обозревателя решений (**Solution Explorer**) и окна свойств (**Properties**), в окне проекта приложения WPF может быть открыто окно структура документа (**Document Outline**) (рис. 3.3). Данное окно служит для быстрого выбора элементов с целью редактирования в визуальном конструкторе Visual Studio.

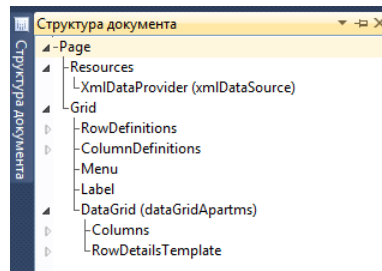


Рис. 3.3. Окно структура документа (**Document Outline**)

3.2.2. Особенности языка разметки XAML.

Основные элементы XAML. Свойства и события в XAML

Особенности языка разметки XAML.

Одним из наиболее значительных преимуществ WPF перед Windows Forms является строгое отделение внешнего вида приложения с графическим интерфейсом от программной логики, которая им управляет. Данная задача решается с помощью языка XAML (произносится как «Замл» или «Замль»).

Язык XAML (*Extensible Application Markup Language* – расширяемый язык разметки приложений) – язык разметки, являющийся диалектом языка XML, который используется для создания экземпляров объектов на платформе .NET.

На этапе разработки приложения WPF файлы XAML являются текстовыми XML-документами, которые имеют расширение **.xaml**. Данные файлы можно сохранять в любой кодировке, поддерживаемой XML, но обычно используется кодировка UTF-8.

В процессе компиляции приложения WPF файлы XAML преобразуются в файлы **языка двоичной разметки приложений BAML** (*Binary Application Markup Language*), которые затем встраиваются в виде ресурсов в сборку проекта.

Основное назначение XAML – конструирование пользовательских интерфейсов WPF. То есть документы XAML определяют расположение и внешний вид элементов управления в приложении WPF.

Элементы XAML отображаются на типы классов заданного пространства имен .NET. Атрибуты в открывающем теге элемента отображаются на свойства и события конкретного типа.

В следующем примере с помощью XAML реализуется внешний вид окна, содержащего одну кнопку:

```
<Window x:Class="WpfAppPrimer.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Окно с кнопкой" Height="150" Width="250">
    <Grid>
        <!-- Добавить кнопку в окно -->
        <Button Name="button1" Width="100" Height="50"
            Click="button1_Click">Нажми меня!</Button>
    </Grid>
</Window>
```

Представление указанного кода XAML показано на рис. 3.4.

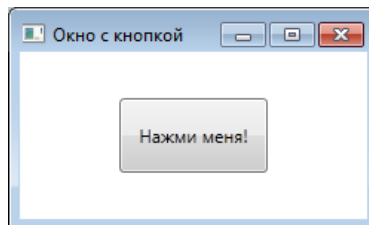


Рис. 3.4. Окно с кнопкой работающего приложения WPF

В большинстве случаев при разработке WPF-приложений используют комбинацию приемов. Часть пользовательского интерфейса разрабатывается с помощью инструмента проектирования (Visual Studio или Expression Blend), а затем производится тонкая настройка за счет ручного редактирования кода разметки.

Существует ряд задач, решение которых возможно только с помощью вручную написанного кода XAML. К таким задачам относятся:

- привязка обработчиков событий к элементам XAML;
- написание выражений привязки данных;
- определение ресурсов (например, стилей и шаблонов элементов управления).

Основные элементы и пространства имен XAML.

Каждый элемент в XAML-документе соответствует определенному экземпляру класса WPF. При этом имя элемента в точ-

ности должно соответствовать имени класса. Например, элемент **Button** сообщает WPF, что должен быть создан объект **Button**.

В коде XAML допускается вложение одного элемента в другой. При этом дочерний элемент обычно размещается в границах родительского элемента. Например, если элемент **Button** вложен в элемент **Grid**, то это может означать, что кнопка **Button** располагается на панели **Grid**.

Свойства каждого элемента XAML можно устанавливать через атрибуты. Кроме атрибутов в XAML для установления свойств элементов используются дескрипторы со специальным синтаксисом.

Документ XAML, как и любой XML-документ, должен иметь только один корневой элемент. Обычно при работе с WPF используются такие корневые элементы, как **Window** (для определения окна), **Page** (для определения страницы) или **Application** (для определения приложения).

Дочерними элементами для корневого элемента являются определения ресурсов и контейнеры.

Можно выделить следующие основные группы элементов XAML, используемые в приложениях WPF:

- *панели компоновки*, которые позволяют требуемым образом расположить содержащиеся внутри них элементы (**Grid**, **StackPanel**, **DockPanel**, **Canvas** и др.);
- *элементы управления* (**Button**, **Label**, **TextBox**, **ListBox**, **DataGrid** и др.);
- *графические примитивы* (**Ellipse**, **Line**, **Rectangle**, **Poligon** и др.);
- *службы документов*, которые позволяют разбивать содержимое на страницы, масштабировать содержимое, адаптировать документы под особенности дисплея пользователя и т.д.

Анализатору XAML требуется знать не только имя класса, но и пространство имен XML, к которому относится этот класс. Пространствами имен, которые присутствуют в каждом документе XAML, являются:

- *Пространство имен WPF:*

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
```

Это пространство является пространством имен по умолчанию и охватывает все классы WPF, включая элементы управления.

• **Пространство имен XAML:**

```
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

Данное пространство включает различные служебные свойства XAML. Этому пространству имен соответствует префикс **x**.

Атрибут **x:Class** может присутствовать только в корневом элементе документа XAML и только в коде, компилируемом в составе проекта. Он не может использоваться в автономном XAML-документе.

Для программного управления элементами управления, описанными в XAML-документе, необходимо для элемента управления задать атрибут **Name**.

Некоторые ключевые слова XAML:

- **x:Key** – позволяет устанавливать значение ключа (уникального имени) для элемента XAML;
- **x:type** – XAML-эквивалент операции **typeof** языка C#, которая выдает **System.Type** на основе указанного имени;
- **x:name** – позволяет указывать имя заданного элемента в документе XAML;
- **x:null** – представляет ссылку **null**.

Свойства и события в XAML.

Язык XAML имеет ряд особенностей при задании свойств (атрибутов) элементов.

Простые свойства элементов задаются в XAML-документе в соответствии с синтаксисом «свойство-значение»:

```
ИмяСвойства="значение"
```

При необходимости задать свойство, которое является полноценным объектом, используются **сложные свойства** в соответствии с синтаксисом «свойство-элемент», который имеет следующий вид:

```
<ИмяЭлемента>
```

```
<ИмяЭлемента.ИмяСвойства>
```

```

<!-- Значение для свойства класса -->
</ИмяЭлемента.ИмяСвойства>
</ИмяЭлемента>

```

В дополнение к синтаксису «свойство-элемент» в XAML поддерживается специальный синтаксис присоединяемых свойств. *Присоединяемые свойства* позволяют дочернему элементу устанавливать значение свойства, определенного в родительском элементе:

```

<РодитЭлемент>
  <ДочерЭлемент
    РодитЭлемент.СвойствоРодитЭлемента="значение" />
</РодитЭлемент>

```

Наиболее часто присоединяемые свойства применяются для позиционирования элементов пользовательского интерфейса внутри одного из диспетчеров компоновки (например, **Grid**).

Для задания свойства может использоваться *расширение разметки*, которое позволяет анализатору XAML получать значение свойства из выделенного внешнего класса.

Расширение разметки заключается в фигурные скобки и использует следующий синтаксис:

```

<Элемент Свойство="{КлассРасширРазметки Аргумент}" />

```

Расширения разметки реализуются классами, дочерними от класса **System.Windows.Markup.MarkupExtention**.

Атрибуты могут быть использованы для прикрепления обработчиков событий. Синтаксис XAML при этом выглядит следующим образом:

```

ИмяСобытия="ИмяМетода_Обработчика"

```

Если в редакторе XAML среды разработки Visual Studio ввести имя события, а за ним — знак равенства, то будут отображены все совместимые обработчики (если они существуют), а также опция **<New Event Handler>** (новый обработчик событий) (рис. 3.5).

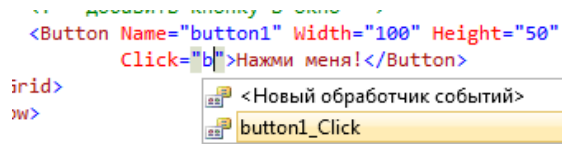


Рис. 3.5. Задание имени обработчика события в редакторе XAML

Двойной щелчок на пункте **<New Event Handler>** приводит к тому, что Visual Studio сгенерирует соответствующий обработчик в файле кода C#.

3.2.3. Основные элементы управления WPF. Размещение элементов управления. Панель Grid

Обзор основных элементов управления WPF.

WPF предоставляет полный набор элементов управления для создания удобных и многофункциональных интерфейсов пользователя.

При построении приложения WPF в Visual Studio большинство элементов управления находятся в панели инструментов (**ToolBox**), когда в окне открыт визуальный конструктор WPF. Как и при создании приложений Windows Forms элементы можно перетаскивать в конструктор и настраивать в окне свойств (**Properties**). При этом Visual Studio автоматически генерирует код XAML.

Элементы управления WPF можно сгруппировать в следующие категории:

- элементы для вывода информации пользователю: **Label**, **TextBlock**, **StatusBar** и др.;
- элементы для ввода: **TextBox**, **RichTextBox**, **PasswordBox**;
- кнопки: **Button**, **RepeatButton**;
- элементы для отображения данных: **DataGrid**, **ListView**, **TreeView**;
- меню: **Menu**, **ContextMenu**, **ToolBar**;
- элементы навигации: **Hyperlink**, **Frame**;
- элементы выбора одного или нескольких пунктов: **ListBox**, **ComboBox**, **CheckBox**;
- элементы отображения и выбора даты: **DatePicker**, **Calendar**;

- мультимедиа: **Image**, **MediaElement** и др.

Все элементы управления WPF являются потомками класса **System.Windows.Control**, в котором определены их базовые характеристики: расположение, фон, передний план, рамка, шрифт текстового содержимого и др.

Элементы управления имеют *фон* (задается свойством **Background**) и *передний план* (свойство **Foreground**). Фоном, как правило, является поверхность элемента управления, а передним планом – текст.

Класс **Control** определяет следующий набор свойств, связанных со шрифтами:

- **FontFamily** – имя шрифта;
- **FontSize** – размер шрифта в единицах, не зависящих от устройства (каждая из них представляет собой 1/96 дюйма);
- **FontStyle** – наклон текста; может принимать значения **Normal** (по умолчанию), **Italic**, **Oblique**;
- **FontWeight** – жирность текста; значения: **Normal** (по умолчанию), **Bold** и др.;
- **FontStretch** – коэффициент растяжения или сжатия текста; некоторые возможные значения: **UltraExpanded** (растягивает текст до 200% от обычной ширины), **UltraCondensed** (сжимает текст до 50%).

Панели компоновки.

При проектировании пользовательского интерфейса приложения необходимо сформировать в окне или странице требуемые элементы управления и задать нужные свойства. Этот процесс называется *компоновкой*.

В WPF компоновка осуществляется с использованием различных *панелей* (элементов-контейнеров) производных от класса **Panel**.

Каждая панель обладает своей собственной логикой компоновки – некоторые укладывают элементы последовательно в строки, другие организуют их в сетку невидимых ячеек.

Окно и страница в WPF может содержать только одну панель компоновки. В панель можно поместить различные элементы пользовательского интерфейса и другие панели.

Для компоновки в приложениях WPF используются следующие основные панели:

- **Grid** – размещает элементы в строки и колонки в соответствии с невидимой таблицей; используется по умолчанию для каждого нового элемента **Window**, созданного с помощью Visual Studio;
- **StackPanel** – размещает элементы в горизонтальные и вертикальные стопки; этот контейнер часто используется для организации небольших участков более крупного и сложного окна;
- **DockPanel** – размещает элементы управления относительно одного из своих внешних краев: **Top** (верхний), **Bottom** (нижний), **Left** (левый), **Right** (правый);
- **Frame** – аналогичен **StackPanel**, но является наиболее предпочтительным для переходов на страницы;
- **Canvas** – размещает элементы управления по заданным координатам; является идеальным контейнером для построения рисунков с помощью фигур.

Панели компоновки размещают свои дочерние элементы с помощью следующих свойств:

- **HorizontalAlignment** и **VerticalAlignment** – определяет, как дочерний элемент позиционируется внутри компоновки, когда имеется дополнительное пространство по горизонтали/вертикали;
- **Margin** – добавляет пустое пространство вокруг элемента;
- **MinWidth** и **MaxWidth** – устанавливает максимальные размеры для элемента;
- **Width** и **Height** – явно устанавливает размеры элемента.

Панель **StackPanel** является одним из простейших контейнеров компоновки. Данная панель укладывает свои дочерние элементы в одну строку или колонку.

Важным свойством панели **StackPanel** является свойство **Orientation**, которое может принимать два значения:

- **Vertical** – компоновать вложенные элементы по вертикали (значение по умолчанию);
- **Horizontal** – компоновать вложенные элементы по горизонтали.

Панель компоновки **Grid** является наиболее гибким и мощным из всех панелей WPF. Панель **Grid** позволяет разбить окно

на меньшие области, которыми можно управлять с помощью других панелей. Элемент **Grid** распределяет дочерние элементы по сетке строк и столбцов.

Создание компоновки на основе **Grid** состоит из двух этапов:

- задание необходимого количества строк и столбцов с указанием их высоты (**Height**) и ширины (**Width**);
- назначение каждому элементу соответствующей строки и столбца.

Столбцы и строки создаются путем заполнения объектами коллекции **Grid.ColumnDefinition** и **Grid.RowDefinition**. Например, если требуется задать три столбца и две строки для **Grid**, то XAML-код может выглядеть следующим образом:

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

Отображение данного кода в окне визуального проекта показано на рис. 3.7.

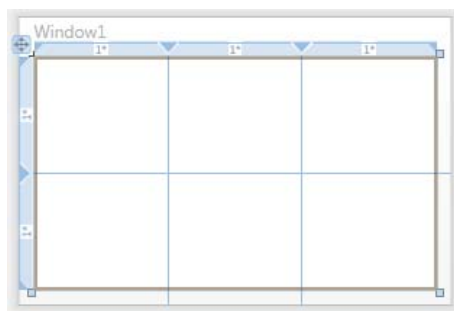


Рис. 3.7. Отображение элемента **Grid** в окне визуального конструктора

Для помещения индивидуальных элементов в ячейку используются присоединенные свойства **Grid.Row** и **Grid.Column**.

Эти свойства принимают числовое значение индекса, начинающееся с 0.

Чтобы растянуть элемент на несколько ячеек можно использовать присоединенные свойства **RowSpan** и **ColumnSpan**. Данные свойства принимают количество строк или столбцов, которые должен занять элемент.

□ **Пример 3.1. Разработка приложения WPF для выполнения параметрических запросов к массиву объектов.**

Требуется разработать приложение WPF, которое позволяет производить запросы LINQ к массиву из примера 1.1. При этом условие отбора результатов запроса должно задаваться через элементы управления (текстовые поля, комбинированные списки и др.). Результаты запроса будут выводиться через окна сообщений.

В создаваемое приложение необходимо добавить класс **Product** из примера 1.1, а также массив объектов данного класса.

Выполним следующие запросы к массиву объектов:

1. Данные о товарах с ценой больше, чем введенная цена.
2. Наименования товаров заданного производителя.
3. Число наименований товаров с весом больше, чем указанный вес.
4. Суммарная стоимость товаров с ценой меньше, чем заданная цена.
5. Наименования товаров с датой выпуска раньше, чем указанная дата.

Код документа XAML для главного окна приложения (**MainWindow.xaml**) представлен на рис. 3.7. Связанный с данным документом код на C# (модуль **MainWindow.xaml.cs**), содержащий обработчики событий, показан на рис. 3.8 и 3.9.

```

<Window x:Class="WpfAppSimple.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Параметрические запросы (Турчин Д.Е., каф. ИиАПС)" Height="330" Width="550"
        Loaded="Window_Loaded">
    <Grid Background="#998BFF">
        <!-- Описание строк и столбцов панели Grid -->
        <Grid.RowDefinitions>
            <RowDefinition Height="140" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="250" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <!-- Панель со списком товаров -->
        <StackPanel Grid.Row="0" Grid.Column="0" Orientation="Vertical">
            <Label>Список товаров:</Label>
            <ListBox Name="listBoxProducts" Height="100" Margin="5"
                SelectionChanged="listBoxProducts_SelectionChanged"></ListBox>
        </StackPanel>
        <!-- Панель с текстовым описанием выбранного товара -->
        <StackPanel Grid.Row="1" Grid.Column="0" Orientation="Vertical"
            Background="#99DD99">
            <Label>Информация о товаре:</Label>
            <TextBlock Name="textBlockInfo" Text="" Margin="5" />
        </StackPanel>
        <!-- Панель с элементами для выполнения параметрических запросов -->
        <StackPanel Grid.Row="0" Grid.Column="1" Grid.RowSpan="2"
            Orientation="Vertical" Background="#99DD99">
            <TextBlock Padding="5">1. Данные о товарах с количеством более N</TextBlock>
            <StackPanel Orientation="Horizontal" Background="#998BFF">
                <Label Width="95">Введите N:</Label>
                <TextBox Name="textBox1" Width="70" Margin="5, 5, 40, 5"></TextBox>
                <Button Name="button1" Width="50" Margin="5" Click="button1_Click">Найти</Button>
            </StackPanel>
            <TextBlock Padding="5">2. Наименования товаров производителя M</TextBlock>
            <StackPanel Orientation="Horizontal" Background="#998BFF">
                <Label Width="80">Выберите M:</Label>
                <ComboBox Name="comboBoxM" Width="120" Margin="5"></ComboBox>
                <Button Name="button2" Width="50" Margin="5" Click="button2_Click">Найти</Button>
            </StackPanel>
            <TextBlock Padding="5">3. Число наименов. товаров весом от W1 до W2</TextBlock>
            <StackPanel Orientation="Horizontal" Background="#998BFF">
                <Label Width="45">W1, r:</Label>
                <TextBox Name="textBoxW1" Width="50" Margin="5"></TextBox>
                <Label Width="45">W2, r:</Label>
                <TextBox Name="textBoxW2" Width="50" Margin="5"></TextBox>
                <Button Name="button3" Width="50" Margin="5" Click="button3_Click">Найти</Button>
            </StackPanel>
            <TextBlock Padding="5">4. Суммар. стоимость товаров с ценой менее P</TextBlock>
            <StackPanel Orientation="Horizontal" Background="#998BFF">
                <Label Width="95">Введите P, руб:</Label>
                <TextBox Name="textBoxP" Width="70" Margin="5, 5, 40, 5"></TextBox>
                <Button Name="button4" Width="50" Margin="5" Click="button4_Click">Найти</Button>
            </StackPanel>
            <TextBlock Padding="5">5. Товары с датой выпуска раньше выбран. даты</TextBlock>
            <StackPanel Orientation="Horizontal" Background="#998BFF">
                <Label Width="95">Выберите дату:</Label>
                <DatePicker Name="datePicker1" Width="105" Margin="5" />
                <Button Name="button5" Width="50" Margin="5" Click="button5_Click">Найти</Button>
            </StackPanel>
        </StackPanel>
    </Grid>
</Window>

```

Рис. 3.7. Код XAML-документа **MainWindow.xaml**

```

12 using System.Windows.Navigation;
13 using System.Windows.Shapes;
14
15 namespace WpfAppSimple
16 {
17     /// <summary>
18     /// Логика взаимодействия для MainWindow.xaml
19     /// </summary>
20     public partial class MainWindow : Window
21     {
22         // Объявление массива объектов класса Product с инициализацией элементов
23         private Product[] products = new[] {
24             new Product { ProductID = "p02356",
25                           Name = "Крупа гречневая",
26                           ProducedBy = "ООО Алтайпродукт".
27
28                           NumberInStock = 48,
29                           Weight = 500,
30                           Price = 56.50,
31                           MakeData = new DateTime(2013, 08, 04)
32             };
33
34         public MainWindow()
35         {
36             InitializeComponent();
37         }
38
39         private void Window_Loaded(object sender, RoutedEventArgs e)
40         {
41             textBox1.Text = "50";
42             textBox3.Text = "330";
43
44             // Заполнить список listBoxProducts элементами
45             foreach (Product p in products)
46             { listBoxProducts.Items.Add(p.ProductID + " - " + p.Name); }
47
48             // Заполнить список comboBox2 производителями
49             foreach (Product p in products) comboBox2.Items.Add(p.ProducedBy);
50             comboBox2.SelectedIndex = 0;
51
52             // --- Вывести данные о товаре по выделенному пункту списка listBoxProducts ---
53             private void listBoxProducts_SelectionChanged(object sender, SelectionChangedEventArgs e)
54             {
55                 textBlockInfo.Text = products[listBoxProducts.SelectedIndex].ToString();
56             }
57         }
58     }
59 }

```

Рис. 3.8. Код на C# для модуля **MainWindow.xaml.cs** (часть 1)

```

94 // --- Получить товары с количеством более n при нажатии button1 ---
95 private void button1_Click(object sender, RoutedEventArgs e)
96 {
97     int n = int.Parse(textBox1.Text);
98     var result = from p in products where p.NumberInStok > n select p;
99     string s = "";
100     foreach (var r in result) s += r.ToString();
101     MessageBox.Show(s, "Все товары с количеством более N");
102 }
103
104 // --- Получить наименования товаров выбр. производителя при нажатии button2 ---
105 private void button2_Click(object sender, RoutedEventArgs e)
106 {
107     var result = from p in products
108                 where p.ProducedBy == comboBoxM.Text
109                 select p;
110     string s = "";
111     foreach (var r in result) s += string.Format("- {0}\n", r.Name);
112     MessageBox.Show(s, "Наименования товаров выбранного производителя");
113 }
114
115 //--- Получить число наименований товаров весом более W при нажатии button3 ---
116 private void button3_Click(object sender, RoutedEventArgs e)
117 {
118     int w1 = int.Parse(textBoxW1.Text);
119     int w2 = int.Parse(textBoxW2.Text);
120     var result = from p in products
121                 where p.Weight > w1 && p.Weight < w2
122                 select p;
123     string s = string.Format("{0} шт.", result.Count());
124     MessageBox.Show(s, "Число товаров весом от W1 до W2");
125 }
126
127 //--- Получить суммарную стоимость товаров с ценой менее P при нажатии button4 ---
128 private void button4_Click(object sender, RoutedEventArgs e)
129 {
130     double price = double.Parse(textBoxP.Text);
131     var result = from p in products where p.Price < price select p;
132     double sumCost = 0;
133     foreach (var r in result)
134     {
135         sumCost += r.Price * r.NumberInStok;
136     }
137     string s = string.Format("{0:f2} руб", sumCost);
138     MessageBox.Show(s, "Суммарная стоимость товаров с ценой меньше P");
139 }
140
141 //--- Получить наименования товаров с датой выпуска позже D при нажатии button5 ---
142 private void button5_Click(object sender, RoutedEventArgs e)
143 {
144     DateTime dtM = Convert.ToDateTime(datePicker1.Text);
145     var result = from p in products
146                 where dtM.CompareTo(p.MakeData) > 0
147                 select p;
148     string s = "";
149     foreach (var r in result) s += string.Format("- {0}\n", r.Name);
150     MessageBox.Show(s, "Товары с датой выпуска позже заданной");
151 }

```

Рис. 3.9. Код на C# для модуля **MainWindow.xaml.cs** (часть 1)

Окно запущенного приложения WPF показано на рис. 3.10. Результаты одного из параметрических запросов, выводимые в окне сообщения, представлены на рис. 3.11. □

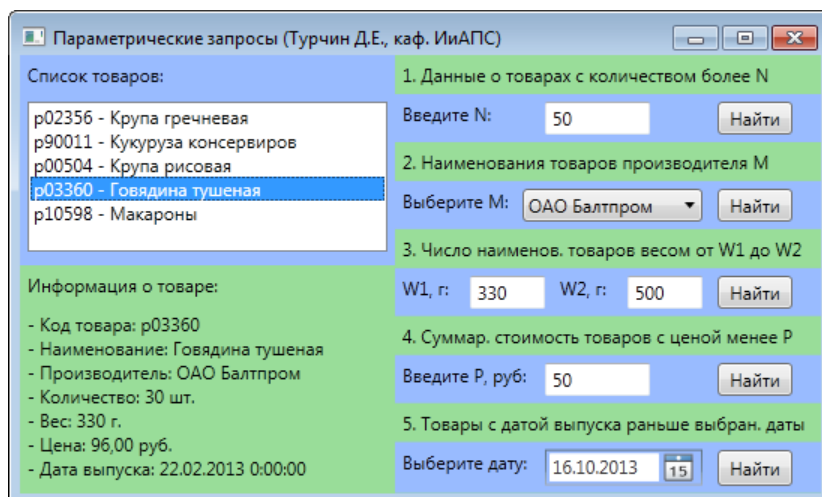


Рис. 3.10. Окно работающего приложения WPF

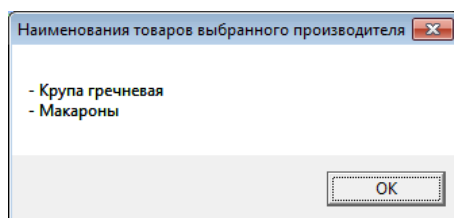


Рис. 3.11. Окна сообщения при выполнении запроса №2

3.2.4. Страничная навигация в приложениях WPF

Страничная навигация в приложениях WPF.

Серьезным достоинством Web-приложений является простая и удобная модель приложения, связанная с размещением информации и элементов управления на Web-страницах и перемещением с одной страницы на другую. Чтобы предоставить разработчикам возможность создавать приложения Windows в стиле Web-приложений, в состав WPF была включена модель страниц со средствами страничной навигации.

В приложениях WPF каждая страница представлена экземпляром класса **System.Windows.Control.Page**.

Класс **Page** имеет следующие основные свойства:

- **Background** – принимает кисть, которая устанавливает заливку для фона;

- **Content** – принимает один элемент, который отображается на странице. Обычно в роли такого элемента выступает панель компоновки (например, **Grid** или **StackPanel**);

- **Foreground**, **FontFamile**, **FontSize** – определяет используемый по умолчанию внешний вид для текста внутри страницы; значение этих свойств наследуется элементами внутри страницы;

- **NavigationService** – возвращает ссылку на объект **NavigationService**, которую можно использовать для отправки пользователя на другую страницу программным путем;

- **KeepAlive** – определяет, должен ли объект страницы оставаться действующим после перехода пользователя на другую страницу;

- **Title** – устанавливает имя, которое должно применяться для страницы в хронологии навигации.

Для добавления страницы в проект WPF в Visual Studio необходимо выбрать в меню **Project** (Проект) пункт **Add Page** (Добавить страницу).

Для отображения страницы (объект **Page**) в работающем приложении WPF ее необходимо разместить в одном из следующих контейнеров:

- **NavigationWindow** – представляет собой видоизмененную версию класса **Window**;

- **Frame** – панель компоновки, которая может быть расположена внутри окна **Window** или внутри другой страницы.

Для перехода между страницами вручную используется элемент **Hyperlink**, который является аналогом гиперссылки в Web-приложениях.

В WPF элементы **Hyperlink** являются *потокowymi элементами* и должны обязательно размещаться внутри другого поддерживающего их элемента (например, внутри **TextBlock**). Это вызвано тем, что гиперссылки часто располагаются внутри текста.

Основным свойством элемента **Hyperlink** является **NavigateUri** – имя страницы, которую производится переход:

```
<Hyperlink NavigateUri="Page1.xaml">Страница 1</Hyperlink>
```

Свойство **NavigateUri** работает только в том случае, когда элемент **Hyperlink** располагается на странице. В противном случае необходимо использовать событие **Click**.

□ *Пример 3.2. Разработка приложения WPF со страничной навигацией.*

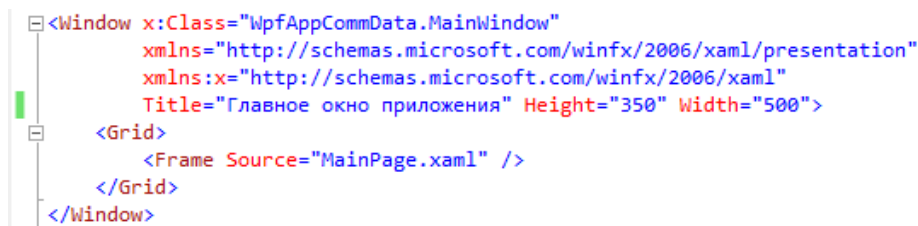
Требуется разработать WPF-приложение, которое предоставляет интерфейс пользователя для работы с данными из XML-документа, полученного в приложении П.1.

Приложение будет состоять из одного окна (**MainWindow**) и четырех страниц, загружаемых в это окно при переходах по ссылкам:

- **MainPage** – главная страница, содержащая ссылки для перехода на другие страницы;
- **HousesPage** – страница «Дома», содержащая табличные данные по обслуживаемым домам;
- **ApartmsPage** – страница «Квартиры», содержащая сведения по обслуживаемым квартирам;
- **PersonsPage** – страница «Жильцы», содержащая данные в табличной форме по обслуживаемым жильцам.

Приложение должно содержать главное меню (**Menu**) с командами по управлению приложением и строку состояния (**StatusBar**), в котором будет отображаться информация по текущему состоянию приложения. В качестве таблиц используется элемент **DataGrid**.

Код документов XAML, соответствующих окну и двум страницам приложения, приведен на рис. 3.12 – 3.14.



```
<?xml version="1.0" encoding="utf-8" ?>
<Window x:Class="WpfAppCommData.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Главное окно приложения" Height="350" Width="500">
    <Grid>
        <Frame Source="MainPage.xaml" />
    </Grid>
</Window>
```

Рис. 3.12. Код XAML-документа **MainWindow.xaml** (главное окно)


```

<Page x:Class="WpfAppCommData.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="400" d:DesignWidth="500"
      Title="Главная страница">
    <Grid>
        <!-- Описание строк и столбцов панели Grid -->
        <Grid.RowDefinitions>
            <RowDefinition Height="20" />
            <RowDefinition Height="50" />
            <RowDefinition Height="*" />
            <RowDefinition Height="80" />
            <RowDefinition Height="20" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="300" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <!-- Панель с главным меню приложения -->
        <StackPanel Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2">
            <Menu />
        </StackPanel>
        <!-- Панель с заголовком страницы -->
        <StackPanel Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2">
            <Label>Оплата коммунальных услуг</Label>
        </StackPanel>
        <!-- Панель со ссылками на другие страницы -->
        <StackPanel Grid.Row="2" Grid.Column="0">
            <TextBlock Margin="25, 15, 0, 0">
                <Hyperlink NavigateUri="HousesPage.xaml">Обслуживаемые дома</Hyperlink>
            </TextBlock>
            <TextBlock Margin="25, 15, 0, 0">
                <Hyperlink NavigateUri="ApartmsPage.xaml">Обслуживаемые квартиры</Hyperlink>
            </TextBlock>
            <TextBlock Margin="25, 15, 0, 0">
                <Hyperlink NavigateUri="PersonsPage.xaml">Обслуживаемые жильцы</Hyperlink>
            </TextBlock>
        </StackPanel>
        <!-- Панель с изображением -->
        <StackPanel Grid.Row="2" Grid.Column="1">
            <Image Name="imgLogo" Source="IiAPS_logo.png" Width="100" Height="100"
                Margin="15" />
        </StackPanel>
        <!-- Панель с Copyright -->
        <StackPanel Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2"
            HorizontalAlignment="Center">
            <TextBlock>© 2013, Turchin Denis. All rights reserved</TextBlock>
        </StackPanel>
        <!-- Панель со строкой состояния -->
        <StackPanel Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="2">
            <StatusBar />
        </StackPanel>
    </Grid>
</Page>

```

Рис. 3.13. Код XAML-документа **MainPage.xaml** (главная страница)

```

<Page x:Class="WpfAppCommData.ApartmsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="350" d:DesignWidth="500"
      Title="Страница Квартиры">
  <Grid>
    <!-- Описание строк и столбцов панели Grid -->
    <Grid.RowDefinitions>
      <RowDefinition Height="20" />
      <RowDefinition Height="50" />
      <RowDefinition Height="*" />
      <RowDefinition Height="20" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <!-- Панель с главным меню приложения -->
    <StackPanel Grid.Row="0" Grid.Column="0">
      <Menu />
    </StackPanel>
    <!-- Панель с заголовком страницы -->
    <StackPanel Grid.Row="1" Grid.Column="0">
      <Label>Обслуживаемые квартиры</Label>
    </StackPanel>
    <!-- Панель с таблицей -->
    <StackPanel Grid.Row="2" Grid.Column="0">
      <DataGrid Name="ApartmsDataGrid">
        <DataGrid.Columns>
          <DataGridTextColumn Header="Код" />
          <DataGridTextColumn Header="Номер" />
          <DataGridTextColumn Header="Площадь, м2" />
          <DataGridTextColumn Header="Плата всего, руб" />
          <DataGridTextColumn Header="Плата пеня, руб" />
          <DataGridTextColumn Header="Дата оплаты" />
        </DataGrid.Columns>
      </DataGrid>
    </StackPanel>
    <!-- Панель со строкой состояния -->
    <StackPanel Grid.Row="3" Grid.Column="0">
      <StatusBar />
    </StackPanel>
  </Grid>
</Page>

```

Рис. 3.14. Код XAML-документа **ApartmsPage.xaml** (страница Квартиры)

Окно работающего приложения WPF с загруженными в него страницами показано на рис. 3.15 и 3.16. □

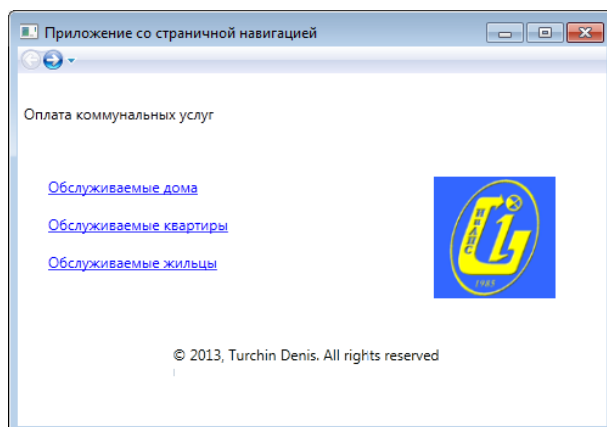


Рис. 3.15. Окно приложения с загруженной главной страницей

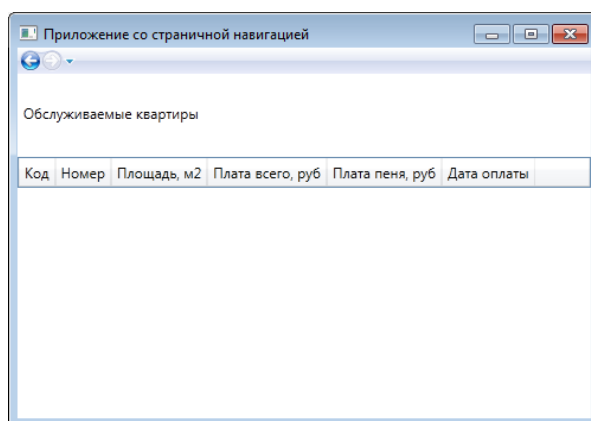


Рис. 3.16. Окно приложения с загруженной страницей «Квартиры»

3.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать приложение WPF, которое выполняет параметрические запросы к массиву объектов из примера 1.1. Параметры запроса вводятся через различные элементы управления (**TextBox**, **ComboBox**, **DatePicker** и др.) и подставляются в предложения **where** запросов LINQ. Результаты запросов выводятся через окно сообщений (**MessageBox**).

3. Разработать приложение WPF, использующее страничную навигацию. В создаваемое приложение необходимо добавить четыре страницы, каждая из которых должна содержать заголовок (элемент **Label**), главное меню (элемент **Menu**) и строку состояния (элемент **StatusBar**). Одна из страниц (главная) должна содержать ссылки (элементы **Hyperlink**) для переходов на остальные страницы. Три другие страницы должны соответствовать сущностям, определенным в XML-документе из работы №2. В каждой из этих страниц должна присутствовать таблица (элемент **DataGrid**) с определенным набором столбцов (на основе данных из XML-документа).

4. Оформить и защитить отчет по лабораторной работе.

3.4. Контрольные вопросы

1. Каково назначение платформы WPF?
2. Для чего предназначен язык XAML?
3. Каковы основные пространства имен, используемые в документах XAML?
4. Что понимают по присоединённым свойствам в XAML и как они записываются в XAML-документе?
5. Как в коде XAML событие для элемента привязывается к обработчику?
6. Каким образом панель **StackPanel** размещает вложенные элементы?
7. С помощью каких свойств осуществляется привязка элемента к определенной строке и столбцу панели **Grid**?
8. Внутри каких элементов XAML может быть размещен элемент **Page**?
9. Как в элементе **Hyperlink** задается адрес для перехода на заданную страницу?

4. ОСНОВЫ ПРИВЯЗКИ И ФОРМАТИРОВАНИЯ ДАННЫХ В ПРИЛОЖЕНИЯХ WPF

4.1. Цель работы

Цель работы – приобрести умение выполнять привязку данных к элементам управления, а также форматировать данные с помощью шаблонов в приложениях WPF.

Работа рассчитана на 5 часов.

4.2. Основные теоретические сведения

4.2.1. Привязка данных WPF

Общие сведения о привязке данных WPF.

Многие Windows-приложения связаны с данными. Задача извлечения информации из источника и отображения ее в пользовательском интерфейсе без использования специальных средств может потребовать написания большого объема кода. Для приложений WPF данная задача решается с помощью привязки данных.

Привязкой данных WPF (*WPF data binding*) называется метод связывания элементов управления с данными, применяемый в приложениях WPF. Привязка данных WPF обеспечивает простой и согласованный способ представления данных и взаимодействия с ними в приложениях.

Для привязки данных в WPF используется класс **System.Windows.Data.Binding**. Объекты класса **Binding** выступают посредниками, отвечающими за связь элементов управления с источниками данных.

Вне зависимости от того, какие элементы привязываются и какой источник данных используется, каждая привязка всегда соответствует модели, показанной на рис. 4.1.



Рис. 4.1. Основные части привязки данных

Привязка обычно состоит из следующих четырех компонентов:

- цель привязки – обычно определенный элемент управления (например, **TextBox**);
- свойство цели – выбранное свойство элемента управления (например, **Text**)
- источник привязки (**Source**);
- путь к используемому свойству источника (**Path**).

При задании привязки поток данных может идти от цели привязки к источнику и/или от источника привязки к цели. Направление потока данных для объекта **Binding** задается с помощью свойства **Mode**, которое может принимать следующие значения:

- **OneWay** – односторонняя привязка, при которой изменение в свойстве источника автоматически приводит к обновлению свойства цели, но изменения свойства цели не передаются обратно источнику;
- **TwoWay** – двусторонняя привязка, при которой изменение в свойстве цели автоматически ведет к обновлению свойства источника и наоборот;
- **OneWayToSource** – односторонняя привязка к источнику, является обратным по отношению к связыванию **OneWay**, то есть обновление свойства цели ведет к обновлению свойства источника.

Для явного задания источника привязки используется свойство **Source**.

Источник привязки также можно определить с помощью свойства **DataContext** родительского элемента, которое позволяет привязывать несколько свойств дочерних элементов к одному источнику.

В WPF все элементы управления, производные от **ItemsControl**, способны отображать список элементов. К таким элементам относятся **ListBox**, **ComboBox**, **ListView**, **TreeView** и **DataGrid**. Для привязки источника к указанным элементам используется свойство **ItemSource**. При этом тип данных, помещаемых в свойство должен поддерживать интерфейс **IEnumerable**.

Если источник привязки является объектом, то для указания пути к значению источника используется свойство **Path**. При свя-

звании данных XML для указания значения используют свойство **XPath**, которому присваивается выражение на языке XPath.

□ **Пример 4.1. Работа с XML-документом с помощью привязки данных в приложении WPF.**

Требуется разработать приложение WPF, которое позволяет производить модификацию XML-документа (добавление новых и удаление существующих элементов, изменение содержимого элементов и значений атрибутов).

В качестве XML-документа воспользуемся документом, полученным в примере 2.2. В доработанном виде этот документ показан на рис. 4.1.

```
<?xml version="1.0" encoding="utf-8"?>
<productStock>
  <product id="p1777" numberInStock="42">
    <name>Крупа гречневая</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">72,50</price>
  </product>
  <product id="p9794" numberInStock="67">
    <name>Кукуруза консервированная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">340</weight>
    <price unit="руб.">56</price>
  </product>
  <product id="p2079" numberInStock="53">
    <name>Крупа рисовая</name>
    <make>ЗАО Увелка</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">42,50</price>
  </product>
  <product id="p5681" numberInStock="30">
    <name>Говядина тушеная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">330</weight>
    <price unit="руб.">95</price>
  </product>
  <product id="p7678" numberInStock="68">
    <name>Макароны</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">500</weight>
    <price unit="руб.">56,35</price>
  </product>
</productStock>
```

Рис. 4.1. Код XML-документа

Для вывода списка товаров добавим шаблон данных с ключевым именем **ProductDataTemplate** в ресурсы главного окна. В

этом шаблоне определим три текстовых блока, содержащих различные данные (код товара, наименование товара и производитель).

Исходный код XAML приложения WPF приведен на рис. 4.2 – 4.4, код на C# показан на рис. 4.5 и 4.6.

```
<Window x:Class="WpfAppProducts.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Работа с XML-данными в приложении WPF (Турчин Д.Е., каф. ИиАПС)"
    Height="550" Width="700" Loaded="Window_Loaded">
    <Window.Resources>
        <!-- Определение шаблона данных для вывода списка товаров -->
        <DataTemplate x:Key="ProductDataTemplate"
            DataType="product">
            <Border Margin="5" BorderThickness="2" BorderBrush="#8B5555"
                CornerRadius="5">
                <StackPanel>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Margin="3" Text="{Binding Path=Attribute[id].Value}"
                            FontWeight="Bold" />
                        <TextBlock Margin="3" Text=" - " />
                        <TextBlock Margin="3" Text="{Binding Path=Element[name].Value}" />
                    </StackPanel>
                    <TextBlock Margin="3" Text="{Binding Path=Element[make].Value}" />
                </StackPanel>
            </Border>
        </DataTemplate>
        <!-- Определение стиля для текстовых полей -->
        <Style TargetType="{x:Type TextBox}">
            <Setter Property="FontFamily" Value="Consolas" />
            <Setter Property="Margin" Value="3" />
            <Setter Property="Width" Value="200" />
            <Setter Property="BorderThickness" Value="2" />
        </Style>
    </Window.Resources>
```

Рис. 4.2. Код XAML-документа **MainWindow.xaml** с описанием ресурсов (часть 1)


```

<Grid Background="#9988FF">
  <Grid.RowDefinitions>
    <RowDefinition Height="255" />
    <RowDefinition Height="50" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="310" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <!-- Панель со списком элементов -->
  <StackPanel Grid.Row="0" Grid.Column="0">
    <Label>Список товаров:</Label>
    <ListBox Name="listBoxProducts" Height="220" Margin="5" BorderThickness="2"
      ItemTemplate="{StaticResource ProductDataTemplate}"
      ItemsSource="{Binding}" />
  </StackPanel>

  <!-- Панель с кнопками -->
  <StackPanel Grid.Row="1" Grid.Column="0" Background="#99DD99">
    <Label>Управление списком товаров:</Label>
    <StackPanel Orientation="Horizontal">
      <Button Name="addButton" Margin="5, 0, 0, 0"
        Click="addButton_Click">Добавить товар</Button>
      <Button Name="remButton" Width="88" Margin="5, 0, 5, 0"
        Click="remButton_Click">Удалить товар</Button>
      <Button Name="saveXData" Click="saveXData_Click">Сохранить данные</Button>
    </StackPanel>
  </StackPanel>

  <!-- Панель с кодом XML-документа -->
  <StackPanel Grid.Row="0" Grid.Column="1" Grid.RowSpan="3">
    <Label>Код XML-документа:</Label>
    <TextBox Name="textBoxShowXML" Width="350" Height="470"
      IsReadOnly="True" VerticalScrollBarVisibility="Auto"/>
  </StackPanel>

```

Рис. 4.3. Код XAML-документа **MainWindow.xaml** (часть 2)

```

<!-- Панель с текстовыми полями -->
<StackPanel Grid.Row="2" Grid.Column="0" Background="#99DD99"
    DataContext="{Binding ElementName=listBoxProducts, Path=SelectedItem}">
    <Label>Редактировать выбранный товар:</Label>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">ID:</Label>
        <TextBox Text="{Binding Path=Attribute[id].Value}" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">Название:</Label>
        <TextBox Width="200" Text="{Binding Path=Element[name].Value}" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">Количество, шт:</Label>
        <TextBox Text="{Binding Path=Attribute[numberInStock].Value}" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">Производитель:</Label>
        <TextBox Text="{Binding Path=Element[make].Value}" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">Вес, г:</Label>
        <TextBox Text="{Binding Path=Element[weight].Value}" />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Label Width="100">Цена, руб:</Label>
        <TextBox Text="{Binding Path=Element[price].Value}" />
    </StackPanel>
</StackPanel>
</Grid>
</Window>

```

Рис. 4.4. Код XAML-документа **MainWindow.xaml** (часть 3)

```

20  /// </summary>
21  public partial class MainWindow : Window
22  {
23      // XML-документ с данными о товарах
24      private XDocument xDoc;
25
26      public MainWindow()
27      {
28          InitializeComponent();
29      }
30
31      // --- Выполнение действий при загрузке окна MainWindow ---
32      private void Window_Loaded(object sender, RoutedEventArgs e)
33      {
34          try
35          {
36              xDoc = XDocument.Load("XMLDocProducts.xml");
37              // Привязка XML-данных к списку listBoxProducts
38              listBoxProducts.DataContext = xDoc.Descendants("productStock").Elements();
39              textBoxShowXML.Text = xDoc.ToString();
40          }
41          catch (System.IO.FileNotFoundException ex)
42          {
43              // Вывод сообщения об ошибке при отсутствии XML-файла
44              MessageBox.Show(ex.Message);
45              return;
46          }
47      }
48
49      // --- Добавление нового элемента product в XML-документ при нажатии addButton ---
50      private void addButton_Click(object sender, RoutedEventArgs e)
51      {
52          // Элемент newProduct со значениями по умолчанию
53          XElement newProduct = new XElement("product",
54              new XAttribute("id", "pXXXX"),
55              new XAttribute("numberInStock", "0"),
56              new XElement("name", "Товар"),
57              new XElement("make", "Производитель"),
58              new XElement("weight", "0",
59                  new XAttribute("unit", "г")),
60              new XElement("price", "0",
61                  new XAttribute("unit", "руб.")),
62              );
63
64          // Добавление элемента newProduct в начала списка элементов product
65          xDoc.Element("productStock").Element("product").AddBeforeSelf(newProduct);
66          // Обновление списка listBoxProducts и текстового поля textBoxShowXML
67          listBoxProducts.DataContext = xDoc.Descendants("productStock").Elements();
68          textBoxShowXML.Text = xDoc.ToString();
69          // Сохранение XML-документа
70          xDoc.Save("XMLDocProducts.xml");
71      }

```

Рис. 4.5. Исходный код на C# (модуль **MainWindow.xaml.cs**, часть 1)

```

73 // --- Удаление выделенного элемента product из XML-документа при нажатии remButton ---
74 private void remButton_Click(object sender, RoutedEventArgs e)
75 {
76     int index = listBoxProducts.SelectedIndex;
77     // Проверка индекса выделенного элемента в списке listBoxProducts
78     if (index < 0) return;
79
80     // Определение элемента на основе выделенного пункта в списке listBoxProducts
81     XElement selProduct = (XElement)listBoxProducts.SelectedItem;
82
83     selProduct.Remove();
84
85     listBoxProducts.DataContext = xDoc.Descendants("productStock").Elements();
86     textBoxShowXML.Text = xDoc.ToString();
87     xDoc.Save("XMLDocProducts.xml");
88 }
89
90 // --- Принудительное сохранение данных с обновлением отображаемого кода XML ---
91 private void saveXData_Click(object sender, RoutedEventArgs e)
92 {
93     textBoxShowXML.Text = xDoc.ToString();
94     xDoc.Save("XMLDocProducts.xml");
95 }
96
97 }
98

```

Рис. 4.6. Исходный код на C# (модуль **MainWindow.xaml.cs**, часть 2)

Окно работающего приложения WPF с привязкой данных показано на рис. 4.7. □

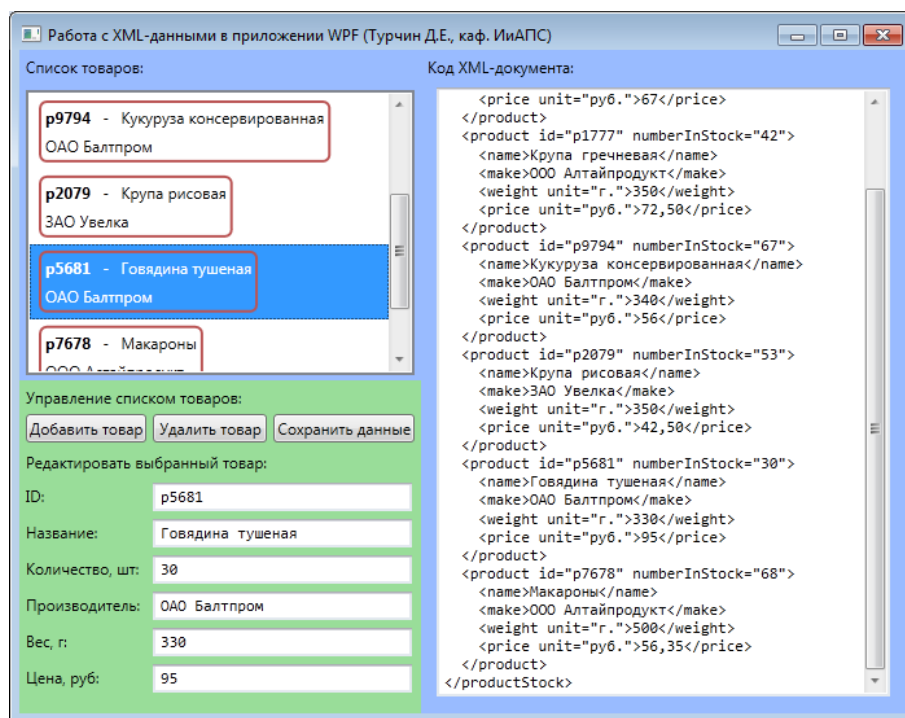


Рис. 4.7. Работа приложения WPF

4.2.2. Шаблоны данных

Шаблоны данных.

В WPF существует возможность использовать комбинацию свойств источника привязки для представления данных. Такая возможность обеспечивается с помощью шаблонов данных.

Шаблон данных (англ. *Data Template*) – это фрагмент кода XAML, задаваемый с помощью элемента **DataTemplate**, который определяет, как привязанный объект данных должен быть отображен.

Как и любой другой блок разметки XAML, шаблон данных может включать любую комбинацию элементов. Шаблон данных также должен включать одно или более выражений привязки, которые извлекают информацию для отображения.

Шаблоны данных поддерживают два типа элементов управления:

- Элементы управления содержимым (**Label**, **TextBox** и др.), поддерживающие шаблон данных через свойство **ContentTemplate**;
- Списочные элементы управления (**ListBox**, **ComboBox**, **ListView** и др.), которые поддерживают шаблоны данных с помощью свойства **ItemTemplate**.

Для элемента **ListBox** простейший шаблон данных может выглядеть следующим образом:

```
<ListBox Name="PersonsList">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Path=PersonName}" />
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Подобно стилям, шаблоны данных часто объявляются как ресурс окна или приложения. Это позволяет повторно использовать шаблон данных в более чем одном элементе управления.

□ **Пример 4.2. Привязка и форматирование данных из документа XML в страничном приложении WPF.**

Требуется доработать приложение со страничной навигацией, полученное в примере 3.2. Для этого добавим привязку таблиц (элементы **DataGrid**) и их отдельных столбцов к данным из XML-документа (приложение П.1). Данные по домам, квартирам и жильцам должны выводиться в табличной форме.

Код XAML-документа для страницы «квартиры» показан на рис. 4.8.

Добавим в приложение класс **XMLObjectModel**, который будет содержать набор статических методов для выполнения действий с XML-документом. Исходный код данного класса приведен на рис. 4.9.

Код присоединенного модуля на C# для страницы «квартиры» показан на рис. 4.10.

```

<Page x:Class="WpfAppCommData.ApartmsPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      mc:Ignorable="d"
      d:DesignHeight="400" d:DesignWidth="500"
      Title="Страница Квартиры" Loaded="Page_Loaded">
    <Grid>
        <!-- Описание строк и столбцов панели Grid -->
        <Grid.RowDefinitions...>
        <Grid.ColumnDefinitions...>
        <!-- Панель с главным меню приложения -->
        <StackPanel...>
        <!-- Панель с заголовком страницы -->
        <StackPanel...>
        <!-- Панель с таблицей -->
        <StackPanel Grid.Row="2" Grid.Column="0">
            <DataGrid Name="ApartmsDataGrid" Style="{StaticResource DataGridStyle}">
                <DataGrid.Columns>
                    <DataGridTextColumn Header="Код"
                        Binding="{Binding Path=Attribute[код].Value}" />
                    <DataGridTextColumn Header="Номер"
                        Binding="{Binding Path=Attribute[номер].Value}" />
                    <DataGridTextColumn Header="Площадь, м2"
                        Binding="{Binding Path=Element[площадь].Value}" />
                    <DataGridTextColumn Header="Расход хол. воды, м3"
                        Binding="{Binding Path=Element[показ_приборов].Element[хол_вода].Value}" />
                    <DataGridTextColumn Header="Расход гор. воды, м3"
                        Binding="{Binding Path=Element[показ_приборов].Element[гор_вода].Value}" />
                    <DataGridTextColumn Header="Расход электр., квтч"
                        Binding="{Binding Path=Element[показ_приборов].Element[эл_энерг].Value}" />
                    <DataGridTextColumn Header="Плата всего, руб"
                        Binding="{Binding Path=Element[плата].Element[всего].Value}" />
                    <DataGridTextColumn Header="Плата пеня, руб"
                        Binding="{Binding Path=Element[плата].Element[пеня].Value}" />
                    <DataGridTextColumn Header="Дата оплаты"
                        Binding="{Binding Path=Element[плата].Attribute[дата].Value}" />
                </DataGrid.Columns>
                <!-- Описание шаблона данных для деталей строки -->
                <DataGrid.RowDetailsTemplate>
                    <DataTemplate>
                        <Border BorderThickness="4" Background="khaki" Padding="10">
                            <TextBlock>
                                <Hyperlink
                                    NavigateUri="PersonsPage.xaml">Перейти к списку жильцов</Hyperlink>
                            </TextBlock>
                        </Border>
                    </DataTemplate>
                </DataGrid.RowDetailsTemplate>
            </DataGrid>
        </StackPanel>
        <!-- Панель со строкой состояния -->
        <StackPanel...>
    </Grid>
</Page>

```

Рис. 4.8. Код страницы **ApartmsPage.xaml** (квартиры)

```

5 | using System.Windows;
6 | using System.Xml.Linq;
7 |
8 | namespace WpfAppCommData
9 | {
10 |     // Класс для определения набора статических методов, реализующих
11 |     // логику LINQ to XML
12 |     class XMLObjectModel
13 |     {
14 |         // Метод, который возвращает XDocument, заполненный на основе
15 |         // содержимого файла "CommService.xml"
16 |         public static XDocument GetXDocument()
17 |         {
18 |             try
19 |             {
20 |                 XDocument xDoc = XDocument.Load("CommService.xml");
21 |                 return xDoc;
22 |             }
23 |             catch (System.IO.FileNotFoundException ex)
24 |             {
25 |                 // Вывод сообщения об ошибке при отсутствии XML-файла
26 |                 MessageBox.Show(ex.Message);
27 |                 return null;
28 |             }
29 |         }
30 |     }

```

Рис. 4.9. Исходный код класса XMLObjectModel

```

13 | using System.Windows.Controls;
14 | using System.Xml.Linq;
15 |
16 | namespace WpfAppCommData
17 | {
18 |     /// <summary>
19 |     /// Логика взаимодействия для ApartmsPage.xaml
20 |     /// </summary>
21 |     public partial class ApartmsPage : Page
22 |     {
23 |         public ApartmsPage()
24 |         {
25 |             InitializeComponent();
26 |         }
27 |
28 |         // --- Выполнение различных действий при загрузке страницы ---
29 |         private void Page_Loaded(object sender, RoutedEventArgs e)
30 |         {
31 |             XDocument xDoc = XMLObjectModel.GetXDocument();
32 |
33 |             // Запрос LINQ с получением данных о квартирах
34 |             var apartms = from a in xDoc.Descendants("квартира") select a;
35 |
36 |             // Привязка результатов запроса LINQ к элементу ApartmsDataGrid
37 |             ApartmsDataGrid.ItemsSource = apartms.ToList();
38 |         }
39 |     }
40 | }

```

Рис. 4.10. Исходный код на C# (модуль ApartmsPage.xaml.cs)

Результат работы приложения WPF для загруженных страниц «квартиры» и «жильцы» показан на рис. 4.11 и 4.12. □

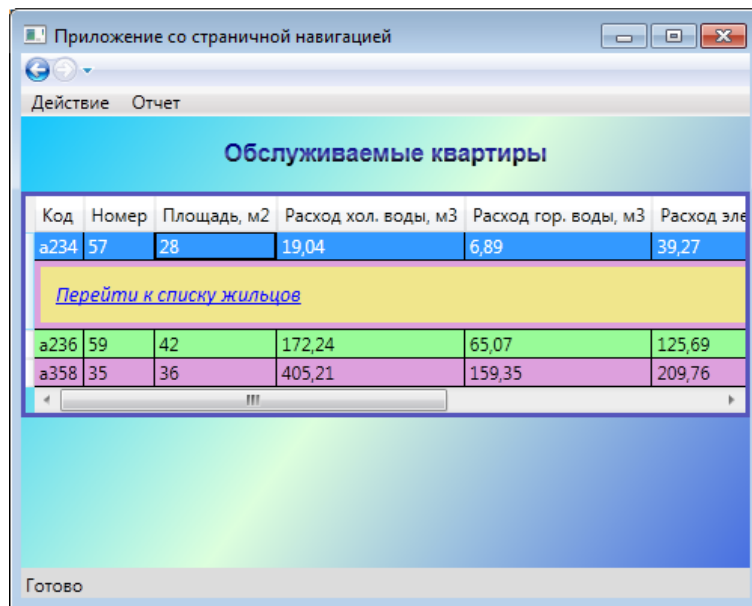


Рис. 4.11. Работа приложения WPF (страница «квартиры»)

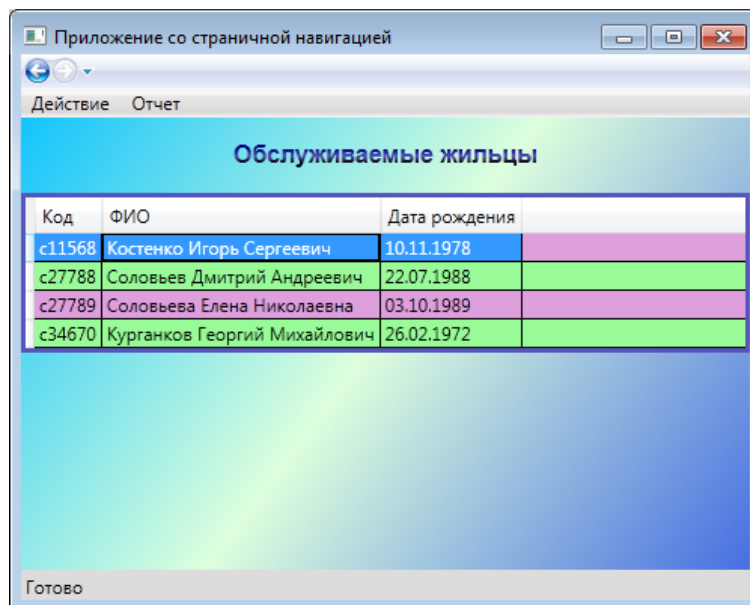


Рис. 4.12. Работа приложения WPF (страница «жильцы»)

4.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать приложение, которое позволяет модифицировать XML-документ, полученный в работе №2 из массива объектов. Данное приложение должно выводить данные по всем элементам, обеспечивать возможность редактирования данных, а также добавления и удаления элементов.
3. Доработать приложение со страничной навигацией из работы №3, добавив привязку таблиц (элементы **DataGrid**) и их отдельных столбцов к данным из XML-документа. При этом данные должны выводиться в табличной форме при загрузке соответствующих страниц приложения.
4. Оформить и защитить отчет по лабораторной работе.

4.4. Контрольные вопросы

1. Что понимают под привязкой данных WPF?
2. Из каких основных компонентов состоит привязка данных?
3. Какие выделяют виды привязки по направлению потока данных?
4. Какие свойства в элементах управления служат для привязки к ним источника данных?
5. Что называют шаблоном данных WPF?
6. Какие свойства элементов управления служат для задания шаблона данных?

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Андерсон К. Основы Windows Presentation Foundation: Пер. с англ. – М.: ДМК Пресс, 2008. – 428 с.
2. Архитектура информационных систем: учебник для студ. учреждений высш. проф. образования / Б.Я. Советов, А.И. Водяхо, В.А. Дубеницкий, В.В. Цехановский. – М.: Издательский центр «Академия», 2012. – 288 с.
3. Басс Л., Клементс Э., Кацман Д. Архитектура программного обеспечения на практике. – СПб.: Питер, 2006. – 240 с.
4. Ватсон Б. С# 4.0 на примерах. – СПб.: БХВ-Петербург, 2011. – 608 с.
5. Гвоздева Т.В. Проектирование информационных систем: учеб. пособие / Т.В. Гвоздева, Т.А. Баллод. – Ростов н/Д: Феникс, 2009. – 508 с.
6. Зиборов В.В. Visual C# 2010 на примерах. – СПб.: БХВ-Петербург, 2011. – 432 с.
7. Мак-Дональд М. Windows Presentation Foundation в .NET 4.0 с примерами на С# для профессионалов: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 1024 с.
8. Макки А. Введение в .NET и Visual Studio 2010 для профессионалов: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 416 с.
9. Пауэрс Л. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл: Пер. с англ. – СПб.: БХВ-Петербург, 2009. – 1200 с.
10. Петцольд Ч. Microsoft Windows Presentation Foundation: Пер. с англ. – СПб.: Питер, 2008. – 944 с.
11. Троелсен Э. Язык программирования С# 5.0 и платформа .NET 4.5, 6-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2013. – 1312 с.
12. Шилдт Г. С# 4.0: полное руководство.: Пер. с англ. – М.:ООО «И.Д. Вильямс», 2011 – 1056 с.

ПРИЛОЖЕНИЕ

П.1. Пример разработки XML-документа

Требуется разработать XML-документ, содержащий сведения об оплате за коммунальные услуги. В качестве сведений выступает информация о жилых домах (код, улица, номер), квартирах (код, номер, площадь), жильцах (код, ФИО, дата рождения), показаниях счетчиков (дата, расход холодной и горячей воды в м3, расход электроэнергии в кВт·ч) и по квартплате (дата, всего, пеня).

Дерево документа представлено на рис. П.1.

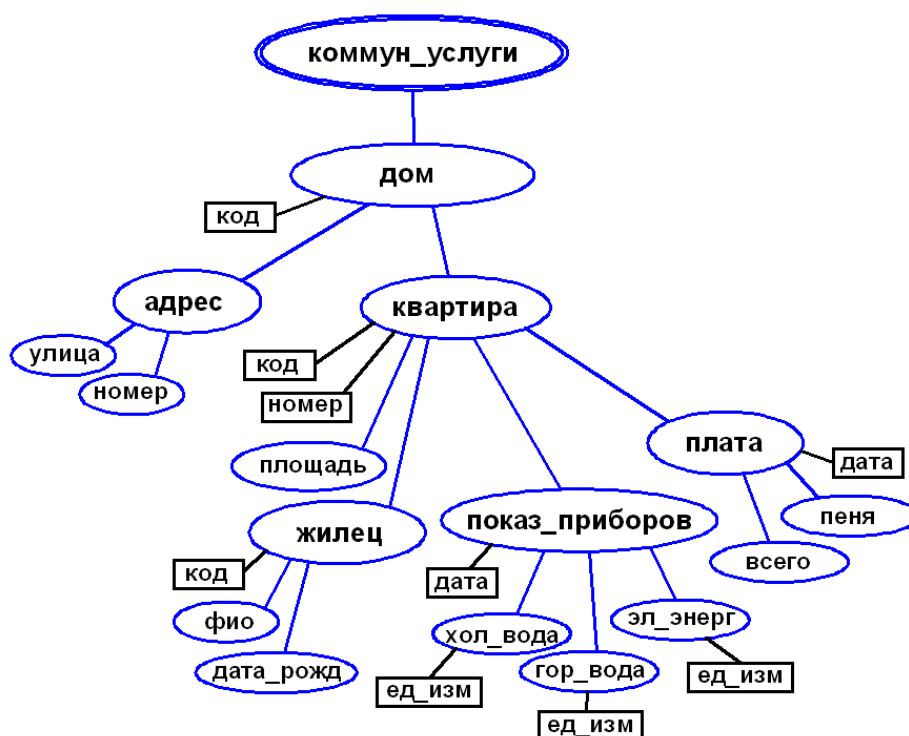


Рис. П.1. Дерево XML-документа

Код XML-документа может иметь следующий вид:

```

<?xml version="1.0" encoding="utf-8"?>
<коммун_услуги
  xmlns="http://www.g42.kommunal.ru/uslugi/oplata">
  <дом код="h18">
    <адрес>

```

```

<улица>Волгоградская</улица>
<номер>8</номер>
</адрес>
<квартира код="a234" номер="57">
  <площадь>28</площадь>
  <жилец код="c11568">
    <фιο>Костенко Игорь Сергеевич</фιο>
    <дата_рожд>10.11.1978</дата_рожд>
  </жилец>
  <показ_приборов дата="02.05.2013">
    <хол_вода ед_изм="м3">19,04</хол_вода>
    <гор_вода ед_изм="м3">6,89</гор_вода>
    <эл_энерг ед_изм="квтч">39,27</эл_энерг>
  </показ_приборов>
  <плата дата="04.05.2013">
    <всего>1896,45</всего>
    <пеня>0</пеня>
  </плата>
</квартира>
<квартира код="a236" номер="59">
  <площадь>42</площадь>
  <жилец код="c27788">
    <фιο>Соловьев Дмитрий Андреевич</фιο>
    <дата_рожд>22.07.1988</дата_рожд>
  </жилец>
  <жилец код="c27789">
    <фιο>Соловьева Елена Николаевна</фιο>
    <дата_рожд>03.10.1989</дата_рожд>
  </жилец>
  <показ_приборов дата="04.05.2013">
    <хол_вода ед_изм="м3">172,24</хол_вода>
    <гор_вода ед_изм="м3">65,07</гор_вода>
    <эл_энерг ед_изм="квтч">125,69</эл_энерг>
  </показ_приборов>
  <плата дата="05.05.2013">
    <всего>2396,45</всего>
    <пеня>0</пеня>
  </плата>
</квартира>
</дом>
<дом код="h72">
  <адрес>

```

```

<улица>Сибиряков-Гвардейцев</улица>
<номер>112</номер>
</адрес>
<квартира код="а358" номер="35">
  <площадь>36</площадь>
  <жилец код="с34670">
    <фио>Курганков Георгий Михайлович</фио>
    <дата_рожд>26.02.1972</дата_рожд>
  </жилец>
  <показ_приборов дата="08.05.2013">
    <хол_вода ед_изм="м3">405,21</хол_вода>
    <гор_вода ед_изм="м3">159,35</гор_вода>
    <эл_энерг ед_изм="квтч">209,76</эл_энерг>
  </показ_приборов>
  <плата дата="12.05.2013">
    <всего>50896,56</всего>
    <пеня>4507,34</пеня>
  </плата>
</квартира>
</дом>
</коммун_услуги>

```

В начальном теге корневого элемента **коммун_услуги** задано пространство имен по умолчанию, которое имеет идентификатор «<http://www.g42.komunal.ru/uslugi/oplata>».