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

Кафедра информационных и автоматизированных производственных систем

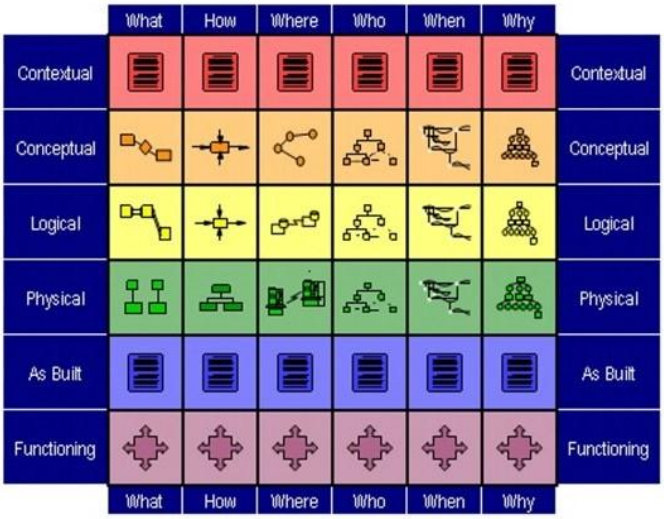
Составитель

Д. Е. Турчин

АРХИТЕКТУРА ИНФОРМАЦИОННЫХ СИСТЕМ

Лабораторный практикум

Рекомендовано учебно-методической комиссией направления
подготовки бакалавров 09.03.02. (230400.62) «Информационные
системы и технологии» в качестве электронного издания
для использования в учебном процессе



Кемерово 2015

Рецензенты:

Ванеев Олег Николаевич – доцент кафедры информационных и автоматизированных производственных систем

Чичерин Иван Владимирович – председатель учебно-методической комиссии направления 09.03.02. (230400.62) «Информационные системы и технологии»

Турчин Денис Евгеньевич. **Архитектура информационных систем:** лабораторный практикум [Электронный ресурс] для студентов направления 09.03.02. (230400.62) «Информационные системы и технологии», очной формы обучения / сост.: Д. Е. Турчин. – Электрон. дан. – Кемерово: КузГТУ, 2015. – Систем. требования: Pentium IV; ОЗУ 256 Мб; Windows XP; мышь. – Загл. с экрана.

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

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 4
ЛАБОРАТОРНЫЕ РАБОТЫ 6
1. Основы преобразования XML-документов с помощью XSLT 6
1.1. Цель и задачи работы 6
1.2. Основные теоретические сведения 6
1.3. Порядок выполнения работы 25
1.4. Контрольные вопросы 31
2. Основы использования полиморфизма в семействах классов на языке C# 33
2.1. Цель и задачи работы 33
2.2. Основные теоретические сведения 33
2.3. Порядок выполнения работы 50
2.4. Контрольные вопросы 52
3. Работа с интерфейсами в приложениях на языке C# 54
3.1. Цель и задачи работы 54
3.2. Основные теоретические сведения 54
3.3. Порядок выполнения работы 78
3.4. Контрольные вопросы 81
4. Основы работы с шаблонами GRASP в приложениях на языке C# 82
4.1. Цель и задачи работы 82
4.2. Основные теоретические сведения 82
4.3. Порядок выполнения работы 110
4.4. Контрольные вопросы 112
5. Работа со структурными шаблонами GoF на языке C# .. 113
5.1. Цель и задачи работы 113
5.2. Основные теоретические сведения 113
5.3. Порядок выполнения работы 133
5.4. Контрольные вопросы 138
6. Работа с поведенческими шаблонами GoF на языке C# 139
6.1. Цель и задачи работы 139
6.2. Основные теоретические сведения 139
6.3. Порядок выполнения работы 158
6.4. Контрольные вопросы 160

7. Основы создания запросов к коллекциям объектов с помощью LINQ 161
7.1. Цель и задачи работы 161
7.2. Основные теоретические сведения 161
7.3. Порядок выполнения работы 174
7.4. Контрольные вопросы 178
8. Основы работы с XML-документами при помощи LINQ to XML 179
8.1. Цель и задачи работы 179
8.2. Основные теоретические сведения 179
8.3. Порядок выполнения работы 193
8.4. Контрольные вопросы 196
9. Основы создания приложений WPF с использованием языка XAML 197
9.1. Цель и задачи работы 197
9.2. Основные теоретические сведения 197
9.3. Порядок выполнения работы 220
9.4. Контрольные вопросы 221
10. Основы привязки и форматирования данных в приложениях WPF 222
10.1. Цель и задачи работы 222
10.2. Основные теоретические сведения 222
10.3. Порядок выполнения работы 235
10.4. Контрольные вопросы 236
СПИСОК ЛИТЕРАТУРЫ 237
ПРИЛОЖЕНИЕ 241
П.1. Пример разработки XML-документа 241
П.2. Некоторые физические формулы и константы 243

ПРЕДИСЛОВИЕ

Пособие предназначено для студентов третьего курса направления подготовки бакалавра 09.03.02. «Информационные системы и технологии», изучающих дисциплину «Архитектура информационных систем».

Основной целью лабораторного практикума является формирование умений, связанных с использованием технологий и средств разработки архитектуры информационных систем.

В качестве используемых средств и технологий выступают:

- язык преобразования XML-документов XSLT;
- интегрированная среда разработки программного обеспечения MS Visual Studio 2012 и язык программирования Visual C# 2012;
- технология LINQ для работы с данными на платформе .NET Framework;
- технология разработки насыщенных клиентских приложений MS Windows Presentation Foundation (WPF) и язык разметки XAML.

Выписка из ФГОС ВПО по направлению 230400.68

Код	**Компетенции, формируемые при освоении дисциплины**	**Результаты освоения**
Б2.Б.5	• **ПК-1** способность проводить предпроектное обследование объекта проектирования, системный анализ предметной области, их взаимосвязей; • **ПК-4** способность проводить выбор исходных данных для проектирования; • **ПК-12** способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные); • **ПК-18** способность исполь-	***знать***: • классификацию информационных систем, структуры и конфигурации информационных систем; • общую характеристику процесса проектирования информационных систем; • основные слои и уровни приложений; • основные методы управления процессом разработки архитектуры информационной системы; ***уметь***: • использовать архитектурные и детализированные решения при проектировании систем;

	зовать технологии разработки объектов профессиональной деятельности, в областях: машиностроение, приборостроение, наука, техника, образование, медицина, административное управление и все виды деятельности в условиях экономики информационного общества	• использовать основные архитектурные шаблоны и стили при проектировании приложений; • использовать основные методики составления архитектурного описания информационной системы; ***владеть***: • моделями и средствами разработки архитектуры информационных систем.

ЛАБОРАТОРНЫЕ РАБОТЫ

1. ОСНОВЫ ПРЕОБРАЗОВАНИЯ XML-ДОКУМЕНТОВ С ПОМОЩЬЮ XSLT

1.1. Цель и задачи работы

Цель работы – прибрести умение преобразовывать XML-документы с помощью технологии XSLT.

Основные задачи:

- освоить создание таблиц стилей XSLT;
- научиться преобразовывать XML-документы с помощью XSLT;
- освоить работу с XSLT на платформе .NET Framework.

Работа рассчитана на 6 часов.

1.2. Основные теоретические сведения

1.2.1. Общие сведения о технологии XSLT

Общие сведения о XSLT. Спецификация XSL.

При работе с XML часто требуется преобразовывать XML-данные из одного представления в другое. Для решения подобной задачи применяют ***XSLT*** (*eXtensible Stylesheet Language Transformations*) – расширяемый язык преобразования XML-документов.

С помощью XSLT можно трансформировать XML-документ в любой формат, например HTML, PDF, RTF, FB2 и др. Кроме того, можно заданный XML-документ преобразовать в XML-документ с другой структурой.

XSLT позволяет работать с содержимым XML-документов без написания программного кода. XSLT – это декларативное описание преобразований XML-документа, то есть без какого-либо программирования описываем, что требуется получить.

Технология XSLT является частью более крупной спецификации – ***XSL*** (*eXtensible Stylesheet Language*) – расширяемый язык таблиц стилей, предложенный консорциумом W3C.

Спецификация XSL состоит из двух частей:

• ***XSL-FO*** (*XSL-Formatting Objects*) – язык для верстки XML-документов. На языке XSL можно описать, как будет оформлен итоговый документ, где и как должны располагаться данные. Язык XSL разрабатывался для тех же самых целей, что и каскадные таблицы стилей (CSS) для HTML, но из-за высокой сложности не получил широкого распространения.

• ***XSL-T*** (*XSL-Transformation*) – язык для преобразования XML-документов.

В настоящее время применяются следующие версии XSLT:

• версия XSLT 1.0 – одобрена в качестве рекомендации 16 ноября 1999 года;

• версия XSLT 2.0 – одобрена в качестве рекомендации 24 января 2007 года;

• версия XSLT 3.0 – находится в стадии начальной рекомендации.

XSLT имеет множество различных применений, в основном в области web-программирования и создания отчётов. В клиент-серверных приложениях XSLT используют для приведения структуры данных из внутреннего формата сервера к формату, понятному клиенту.

Можно отметить следующие основные ограничения XSLT:

• XSLT не подходит для описания преобразований с очень сложной логикой. Это связано с тем, что преобразования в XSLT представляют собой наборы элементарных правил, которых может быть недостаточно для решения сложных задач.

• XSLT не подходит для преобразований, которые требуют сложных вычислений. Данное ограничение вызвано тем, что язык XPath, применяемый в XSLT, поддерживает только простейшие вычислительные операции и функции.

По традиции, документ, записанный на языке XSLT, называется ***таблицей стилей*** (*stylesheet*), хотя его правильнее называть документом, содержащим правила преобразований.

Язык XSLT представляет собой диалект XML. Это означает, что обеспечивающая трансформацию XML-документа таблица стилей XSLT является правильно оформленным документом XML.

XSLT использует язык XPath для доступа к отдельным частям входного XML-документа и для организации вычислений.

XSLT-процессоры.

В XSLT-преобразовании участвуют три документа:

• входящий XML-документ, который подвергается преобразованию;

• документ XSLT, который описывает само преобразование;

• выходящий документ, который является результатом преобразования.

Выполнением преобразования над XML-документами занимаются специальные программы, которые называются ***XSLT-процессорами***.

Процессор оперирует не самими документами, а древовидными моделями их структур.

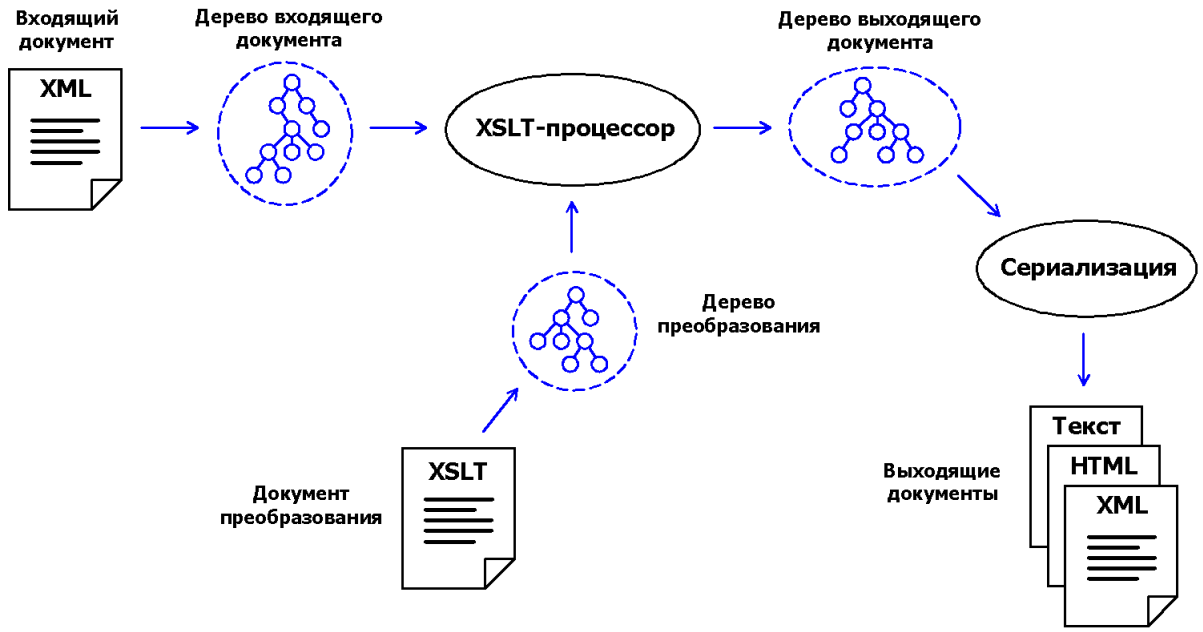


Рис. 1.1. Общая схема XSLT-преобразования

Процесс обработки делится на три этапа:

• ***Этап разбора*** (парсинг) документа, на котором процессор разбирает входящий XML-документ и документ XSLT, создавая для них древовидные структуры данных.

• ***Этап преобразования***, на котором к дереву входного документа применяются правила, описанные в XSLT-документе. В итоге процессор создаёт дерево выходного документа.

Таблица стилей XSLT состоит из шаблонов (*templates*), определяющих, каким образом каждый узел исходного дерева должен быть представлен в дереве результата. Преобразования в XSLT чаще всего это наборы правил вида «если обнаружен узел определённого типа, то выполнить следующие действия».

Процессор проходит по дереву источника, начиная с корня, и ищет соответствующие шаблоны в дереве таблицы стилей. Обнаружив шаблон, он с помощью содержащихся в нем правил записывает абстрактное представление результата в дерево выходного документа.

• ***Этап сериализации***, на котором созданное дерево выходного документа записывается в файл.

Пример преобразуемого XML-документа и простой таблицы стилей XSLT представлены в листингах 1.1 и 1.2. Выходным документом данного преобразования будет являться HTML-документ.

Листинг 1.1. Код XML-документа

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="transform.xslt"?>
<employees>
  <employee id="1">
    <firstname>Игорь</firstname>
    <lastname>Васильев</lastname>
    <phone>(206) 555-9857</phone>
    <notes>
    <![CDATA[Закончил КузГТУ в 2009 г. по специальности
"Автоматизация технологических процессов и производств".]]>
    </notes>
  </employee>
  <employee id="2">
    <firstname>Елена</firstname>
    <lastname>Крылова</lastname>
    <phone>(206) 555-9482</phone>
    <notes>
    <![CDATA[Закончила КузГТУ в 2011 г. по специальности
"Информационные системы и технологии".]]>
    </notes>
  </employee>
</employees>
```

Листинг 1.2. Код таблицы стилей XSLT (**transform.xslt**)

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <head>
        <title>Список сотрудников</title>
      </head>
      <body bgcolor="#DDDDFF">
        <h1>Список сотрудников</h1>
        <table border="1">
          <tr>
            <th>Код</th>
            <th>Фамилия</th>
            <th>Имя</th>
            <th>Телефон</th>
            <th>Примечания</th>
          </tr>
          <xsl:for-each select="employees/employee">
            <tr>
              <td>
                <xsl:value-of select="@id"/>
              </td>
              <td>
                <xsl:value-of select="lastname"/>
              </td>
              <td>
                <xsl:value-of select="firstname"/>
              </td>
              <td>
                <xsl:value-of select="phone"/>
              </td>
              <td>
                <xsl:value-of select="notes"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Результат преобразования XML-документа с помощью таблицы стилей XSLT показан на рис. 1.2.

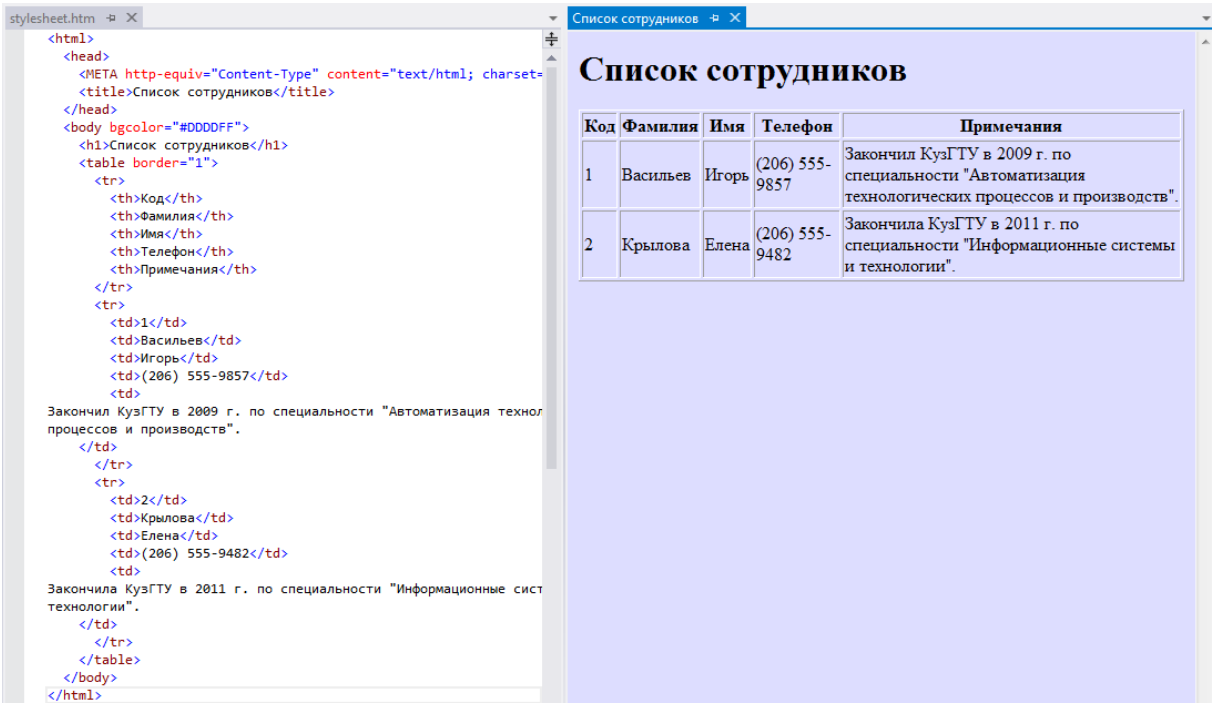


Рис. 1.2. Результат XSLT-преобразования (MS Visual Studio 2012)

1.2.2. Структура и основные элементы таблицы стилей XSLT. Редакторы XSLT

Структура таблицы стилей XSLT. Шаблоны преобразования.

Для того чтобы выделить элементы, которые принадлежат языку XSLT, применяют пространство имён с идентификатором **http://www.w3c.org/1999/XSL/Transform**. Общепринятым префиксом пространства имён языка XSLT является **xsl**.

Таким образом, объявление пространства имён XSLT в общем случае будет иметь вид:

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

Поскольку преобразования XSLT являются документами XML, то они должны начинаться с объявления:

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

После указанного объявления записывается корневой элемент **xsl:stylesheet** или его аналог **xsl:transform**. Обязательным

атрибутом данного элемента является **version**, который определяет версию XSLT (**1.0** или **2.0**).

Необязательный атрибут **id** задаёт уникальный идентификатор данного преобразования. Этот атрибут используется в тех случаях, когда преобразование включено в документ XML.

Прямыми дочерними элементами для **xsl:stylesheet** могут быть:

• **xsl:temlplate** – задаёт шаблон преобразования, позволяющий указать способ преобразования определённых узлов в исходном XML-документе;

• **xsl:output** – определяет тип выходного документа (по умолчанию методом вывода является HTML);

• **xsl:include** – позволяет включать в таблицу стилей XSLT внешние файлы преобразования.

Для указания способа выполнения преобразования используются ***шаблоны преобразования***, задаваемые с помощью элемента **xsl:temlplate**. Каждый элемент **xsl:temlplate** обеспечивает выбор одного узла или набора узлов в исходном XML-документе.

Необязательными атрибутами элемента **xsl:temlplate** являются:

• **match** – задаёт выражение XPath, позволяющее выбирать узлы в исходном XML-документе;

• **name** – содержит имя шаблона;

• **priority** – задаёт приоритет шаблона с помощью положительного или отрицательного целого числа.

Шаблоны строятся по строгим правилам и могут содержать элементы **xsl:param**, за которыми следует тело шаблона.

В теле шаблона может присутствовать ряд элементов XSLT, называемых ***инструкциями***. К наиболее часто применяемым инструкциям XSLT относятся:

• **xsl:apply-templates** – позволяет применить шаблон в зависимости от типа и контекста выбранного узла;

• **xsl:value-of** – обеспечивает доступ к значению выбранного узла;

• **xsl:attribute** – обеспечивает создание нового атрибута в выходном документе.

Элемент **xsl:apply-templates** позволяет применить соответствующий шаблон в зависимости от типа и контекста каждого

выбранного узла. Этот элемент имеет следующие необязательные атрибуты:

• **select** – задаёт с помощью XPath-выражения набор обрабатываемых узлов; если данный атрибут отсутствует, то будут обрабатываться только дочерние узлы текущего узла;

• **mode** – устанавливает режим обработки.

Элемент **xsl:apply-templates** по умолчанию применяет шаблоны только к дочерним узлам выбранного узла или набора узлов. При этом атрибуты не считаются дочерними узлами элементов.

Получить доступ к значению узла можно при помощи элемента **xsl:value-of**. Данный элемент имеет обязательный атрибут **select**, который позволяет выбрать узел с помощью выражения XPath. Элемент **xsl:value-of** является пустым.

Для создания новых атрибутов в выбранном элементе служит элемент **xsl:attribute**. Элемент **xsl:attribute** имеет обязательный атрибут **name** – имя нового атрибута. Кроме того, данный элемент может содержать атрибут **namespace** – пространство имён нового атрибута (присваивается URI).

Принятие решений и сортировка данных.

В XSLT существует возможность управления преобразованием в зависимости от значения данных. Эта возможность обеспечивается такими элементами, как **xsl:if**, **xsl:choose**, **xsl:for-each**, **xsl:sort** и др.

При помощи элемента **xsl:if** осуществляется проверка условия и принимаются действия на основе результата проверки. Элемент **xsl:if** имеет один обязательный атрибут **test**, который устанавливает логическое выражение.

Элемент **xsl:if** во многом похож на оператор **if-then**, применяемый во многих языках программирования. В отличие от **if-then** элемент **xsl:if** не имеет инструкции **else**, которая выполняется при ложности условия в **if**.

Внутри элемента **xsl:if** помещается тело шаблона. Если логическое выражение истинно, то тело шаблона используется, если ложно – игнорируется.

Для задания альтернативных ветвей выполнения XSLT служит элемент **xsl:choose**, который не имеет атрибутов. Элемент **xsl:choose** похож на оператор **switch**, используемый в языках Java и C#.

Элемент **xsl:choose** должен содержать один или более дочерних элементов **xsl:when**, имеющих обязательный атрибут **test** и может содержать единственный элемент **xsl:otherwise**, который в случае применения должен стоять последним.

Элемент **xsl:for-each** позволяет применять тело шаблона в цикле для всех выбранных узлов. Данный элемент имеет единственный обязательный атрибут **select**, который принимает выражение XPath.

При помощи элемента **xsl:sort** можно сортировать узлы. Этот элемент устанавливает порядок обработки узлов для **xsl:apply-templates** и **xsl:for-each** и должен располагаться внутри них. Элемент **xsl:sort** является пустым.

Необязательными атрибутами элемента **xsl:sort** являются:

• **select** – принимает выражение XPath, возвращающее набор узлов для сортировки;

• **order** – задаёт порядок сортировки; принимает значение «ascending» (по возрастанию – значение по умолчанию) или «descending» (по убыванию);

• **data-type** – определяет, будет ли сортировка вестись в алфавитном или числовом порядке; принимает значение «text» (текст – по умолчанию) или «number» (число).

Выбор методов вывода. Связывание XML-документа с таблицей стилей XSLT.

Тип выходного документа определяет элемент **xsl:output**, который является дочерним элементом для **xsl:stylesheet**. Элемент **xsl:output** имеет несколько необязательных атрибутов, основными из которых являются:

• **method** – задаёт формат вывода; принимает значения «xml», «html», «text»;

• **indent** – определяет будет ли выходной документ выровнен с отражением структуры вложенности; может принимать значения «no» (по умолчанию) или «yes»;

- **encoding** – задаёт кодировку символов;
- **version** – задаёт версию выходного документа.

Официально методом вывода по умолчанию является HTML, при условии, что выполняются следующие требования:

- корневой элемент документа имеет дочерний элемент;
- в названии корневого элемента присутствует «html» в любой комбинации верхнего и нижнего регистров.

Если указанные требования не выполняются, то методом вывода по умолчанию является XML.

Для связывания XML-документа и документа XSLT используется инструкция обработки **<?xsl:stylesheet?>**, которая записывается в прологе XML-документа и указывает процессору XSLT, какую таблицу применить к данному документу.

Инструкция **<?xsl:stylesheet?>** имеет следующие обязательные атрибуты:

- **href** – URI документа XSLT;
- **type** – тип MIME документа XSLT; обычно «text/xml» или «application/xml».

К необязательным атрибутам **<?xsl:stylesheet?>** инструкции относятся:

- **title** – устанавливает заголовок, позволяющий различать различные инструкции **<?xsl:stylesheet?>**;
- **charset** – устанавливает кодировку символов;
- **media** – описывает средства вывода (например, «print» или «aural»);
- **alternate** – принимает значение «yes», либо «no».

Связывание файла преобразования **transform.xslt** с документом XML может иметь следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="stylesheet.xslt"?>
<root>
  <!-- ... -->
</root>
```

Работа с XSLT в MS Visual Studio. Отладчик XSLT.

Для создания нового документа XSLT с помощью MS Visual Studio необходимо в меню **File** выбрать команду **New File**. В открывшемся окне шаблонов (рис. 1.3) следует выбрать шаблон **XSLT File**  (XSLT-файл) и нажать кнопку **Open**. Откроется ре-

дактор XML, в котором для нового документа автоматически добавлено объявление XML, корневой элемент **xsl:stylesheet**, а также ряд других элементов.

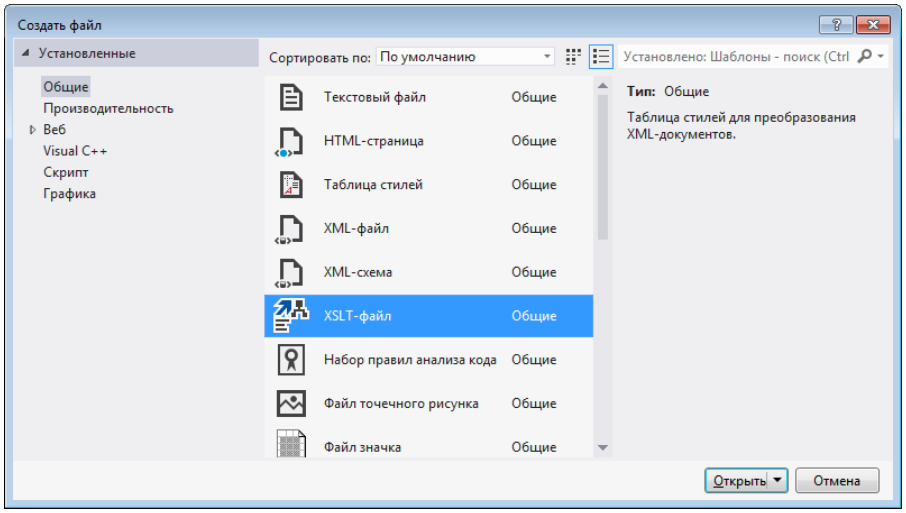


Рис. 1.3. Окно **Создать файл** с выбранным пунктом **XSLT-файл** (Visual Studio 2012)

Редактор XML, встроенный в MS Visual Studio, может использоваться для редактирования таблиц стилей XSLT. При этом являются доступными такие стандартные функции редактора, как технология IntelliSense, структурирование, XML-фрагменты и др. К функциям, доступным только при работе с XSLT, относятся:

- выделение ключевых слов XSLT, таких как **template**, **match** и других, специальным цветом (задаётся с помощью настройки **Шрифты и цвета**);
- редактор XML использует установленный файл **xslt.xsd** для проверки таблиц стилей XSLT; ошибки проверки подчеркиваются синей волнистой линией;
- возможность запуска отладчика XSLT из XSLT-файла; отладка XSLT доступна в Visual Studio Team System и в выпуске Professional Edition;
- возможность выполнять преобразование XSLT и просматривать выходные данные в редакторе XML.

Для отладки таблицы стилей XSLT используется ***отладчик XSLT***, который поддерживает задание точек останова, пошаговое выполнение кода, просмотр состояний выполнения XSLT и т. д.

При запуске отладки отладчик XSLT открывает окна, в которых отображается входной документ и выход XSLT (рис. 1.4).

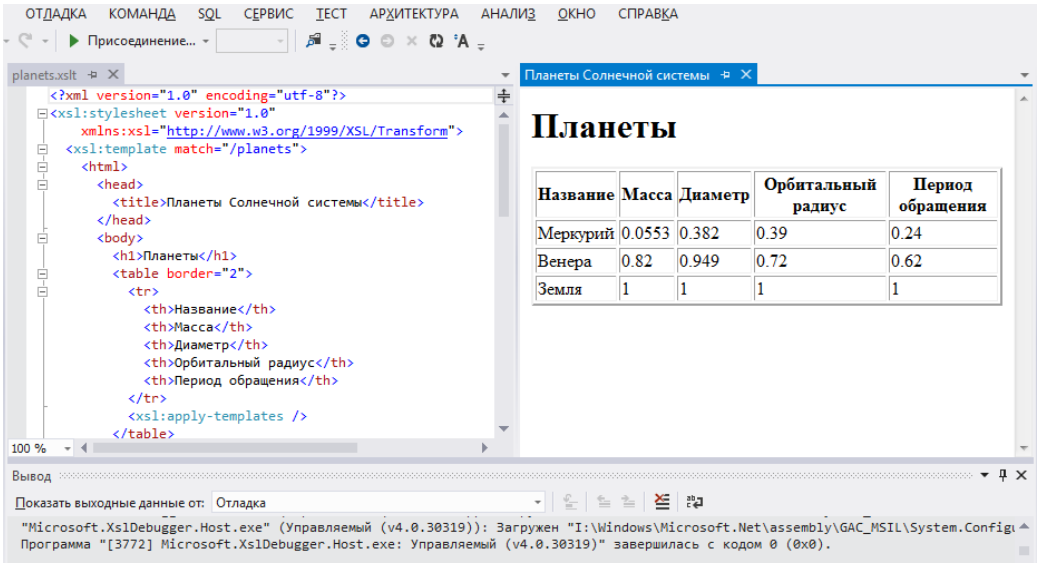


Рис. 1.4. Окна с таблицей стилей и выходом при отладке XSLT (Visual Studio 2012)

❒ ***Пример* 1.1. *Создание таблицы стилей XSLT для преобразования XML-документа в документ с другой структурой.***

Требуется преобразовать структуру XML-документа из приложения П.1 к виду, показанному на рис. 1.5. Для этого необходимо разработать таблицу стилей XSLT.

Полученная таблица стилей представлена в листинге 1.3.

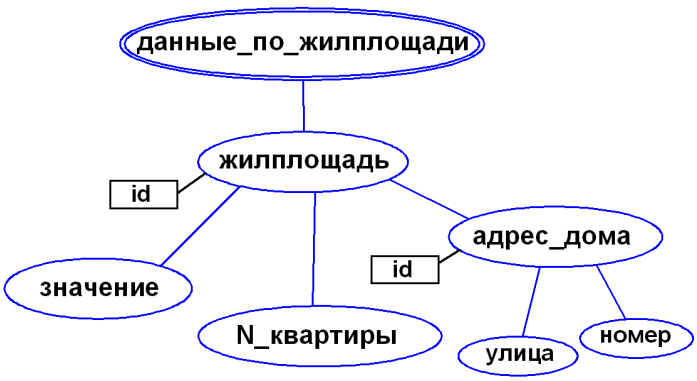


Рис. 1.5. Требуемое дерево выходного XML-документа

Листинг 1.3. Код таблицы стилей XSLT (**transfToXml.xslt**)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="коммун_услуги">
    <данные_по_жилплощади>
      <xsl:for-each select="//квартира">
        <жилплощадь>
          <!-- Добавление атрибута "id" в элемент "жилплощадь" -->
          <xsl:attribute name="id">
            <xsl:value-of select="@код"/>
          </xsl:attribute>
          <значение>
            <xsl:value-of select="площадь"/>
          </значение>
          <N_квартиры>
            <xsl:value-of select="@номер"/>
          </N_квартиры>
          <адрес_дома>
            <xsl:attribute name="id">
              <xsl:value-of select="parent::дом/@код"/>
            </xsl:attribute>
            <улица>
              <xsl:value-of select="parent::дом/адрес/улица"/>
            </улица>
            <номер>
              <xsl:value-of select="parent::дом/адрес/номер"/>
            </номер>
          </адрес_дома>
        </жилплощадь>
      </xsl:for-each>
    </данные_по_жилплощади>
  </xsl:template>

</xsl:stylesheet>
```

Код полученного XML-документа представлен в листинге 1.4. ❒

Листинг 1.4. Код полученного XML-документа (**transform.xml**)

```
<?xml version="1.0" encoding="utf-8"?>
<данные_по_жилплощади>
  <жилплощадь id="a234">
    <значение>28</значение>
    <N_квартиры>57</N_квартиры>
    <адрес_дома id="h18">
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес_дома>
  </жилплощадь>
  <жилплощадь id="a236">
    <значение>42</значение>
    <N_квартиры>59</N_квартиры>
    <адрес_дома id="h18">
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес_дома>
  </жилплощадь>
  <жилплощадь id="a358">
    <значение>36</значение>
    <N_квартиры>35</N_квартиры>
    <адрес_дома id="h72">
      <улица>Сибиряков-Гвардейцев</улица>
      <номер>112</номер>
    </адрес_дома>
  </жилплощадь>
</данные_по_жилплощади>
```

❒ ***Пример* 1.2. *Создание документа XSLT для преобразования XML-документа в формат HTML.***

Требуется разработать таблицу стилей XSLT, которая преобразует XML-документ в формат HTML. Часть данных из документа должна быть представлена в форме таблицы, а другая часть – в виде многоуровневого списка.

Отобразим в виде списка данные о жильцах обслуживаемых домов (адрес дома, ФИО и дата рождения жильца). В форме таблицы будут приведены данные о квартирах (код, номер, площадь, адрес дома).

Код полученной таблицы стилей XSLT представлен в листингах 1.5 и 1.6.

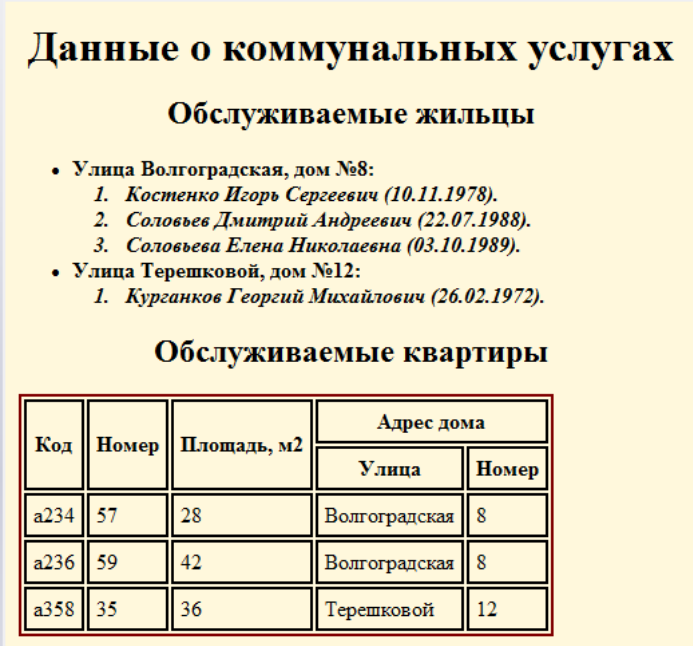
Листинг 1.5. Код таблицы стилей XSLT (часть 1)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">

  <xsl:output method="html" indent="yes"/>

  <xsl:template match="коммун_услуги">
    <html>
      <head>
        <style type="text/css">
          body {
          background-color: rgb(255, 248, 220);
          }
          h1, h2 {
          text-align: center;
          }
          table {
          border: thin solid rgb(128, 0, 0);
          }
          thead {
          font-weight: bold;
          text-align: center;
          }
          td {
          padding: 5 px;
          border: thin solid black;
          }
          ul {
          font-weight: bold;
          }
          ol {
          font-style: italic;
          forecolor:
          }
        </style>
        <title>Список жильцов</title>
      </head>
      <body>
        <h1>Данные о коммунальных услугах</h1>

        <h2>Обслуживаемые жильцы</h2>
        <!-- Размещение данных в многоуровневом списке -->
        <ul>
          <xsl:for-each select="дом">
            <li>
              <xsl:text>Улица </xsl:text>
              <xsl:value-of select="адрес/улица"/>
              <xsl:text>, дом №</xsl:text>
              <xsl:value-of select="адрес/номер"/>
              <xsl:text>:</xsl:text>
              <ol>
                <xsl:for-each select="квартира/жилец">
                  <li>
                    <xsl:value-of select="фио"/>
                    <xsl:text> (</xsl:text>
                    <xsl:value-of select="дата_рожд"/>
                    <xsl:text>).</xsl:text>
                  </li>
                </xsl:for-each>
              </ol>
            </li>
          </xsl:for-each>
        </ul>
```

Листинг 1.6. Код таблицы стилей XSLT (часть 2)

```
        <h2>Обслуживаемые квартиры</h2>
        <!-- Размещение данных в таблице -->
          <table>
            <thead>
              <tr>
                <td rowspan="2">Код</td>
                <td rowspan="2">Номер</td>
                <td rowspan="2">Площадь, м2</td>
                <td colspan="2">Адрес дома</td>
              </tr>
              <tr>
                <td>Улица</td>
                <td>Номер</td>
              </tr>
            </thead>
            <tbody>
              <xsl:for-each select="//квартира">
                <tr>
                  <td>
                    <xsl:value-of select="@код"/>
                  </td>
                  <td>
                    <xsl:value-of select="@номер"/>
                  </td>
                  <td>
                    <xsl:value-of select="площадь"/>
                  </td>
                  <td>
                    <xsl:value-of select="parent::дом/адрес/улица"/>
                  </td>
                  <td>
                    <xsl:value-of select="parent::дом/адрес/номер"/>
                  </td>
                </tr>
              </xsl:for-each>
            </tbody>
          </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

Результат преобразования исходного XML-документа показан на рис. 1.6. ❒



Данные о коммунальных услугах

Обслуживаемые жильцы

- Улица Волгоградская, дом №8:
  1. *Костенко Игорь Сергеевич (10.11.1978).*
  2. *Соловьев Дмитрий Андреевич (22.07.1988).*
  3. *Соловьева Елена Николаевна (03.10.1989).*
- Улица Терешковой, дом №12:
  1. *Курганков Георгий Михайлович (26.02.1972).*

Обслуживаемые квартиры

Код	Номер	Площадь, м2	Адрес дома	
			Улица	Номер
a234	57	28	Волгоградская	8
a236	59	42	Волгоградская	8
a358	35	36	Терешковой	12

Рис. 1.6. Результат преобразования XML-документа

1.2.3. Работа с XSLT на платформе .NET Framework. Класс XmlCompiledTransform

Работа с XSLT на платформе .NET Framework. Класс XmlCompiledTransform.

При разработке собственных приложений, обрабатывающих XML-данные, может потребоваться, чтобы эти приложения выполняли функцию XSLT-процессора. Такая возможность поддерживается при разработке приложений на платформе .NET Framework.

Классы для работы с XSLT размещены в пространстве имён **System.Xml.Xsl**. Основным классом данного пространства является класс **XslCompiledTransform**, который представляет XSLT-процессор, осуществляющий компиляцию и преобразование.

К основным методам класса **XslCompiledTransform** относятся:

- **void Load(string stylesheetUri)** – загружает и компилирует таблицу стилей XSLT, размещённую по заданному **stylesheetUri**;
- **void Transform(string inputUri, string outputUri)** – выполняет преобразование входного документа, заданного с помо-

щью **inputUri**, и записывает результат в выходной документ, расположенный по указанному **outputUri**.

❒ ***Пример* 1.3. *Разработка приложения Windows Forms на языке C# для выполнения XSLT-преобразования документов XML*.**

Требуется разработать Windows-приложение, которое обеспечивает преобразование заданного XML-документа с помощью указанной таблицы стилей XSLT. Выбор файлов, необходимых для XSLT-преобразования, требуется осуществлять через диалоговое окно. Результат преобразования должен отображаться в окне приложения.

Для реализации диалога открытия файлов добавим на форму элемент управления **openFileDialog1**. Отображение выходного документа будет выполняться через элемент управления **webBrowser1**.

Интерфейс пользователя для разрабатываемого приложения показан на рис. 1.7.

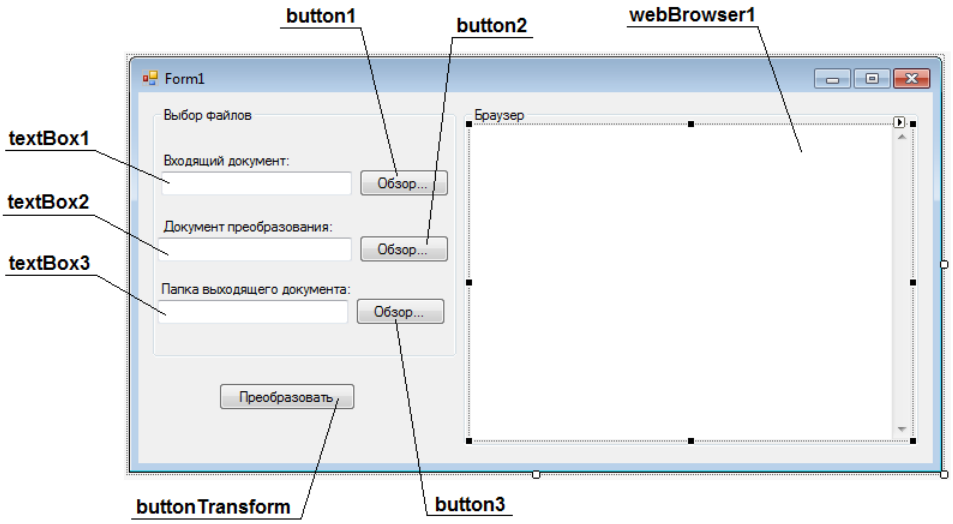


Рис. 1.7. Интерфейс пользователя приложения

Исходный код приложения представлен в листинге 1.7.

Листинг 1.7. Исходный код приложения

```
using System.IO;
using System.Xml.Xsl; // Импорт пространства имён для работы с XSLT

namespace WinFormsApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // Обработчик события "Загрузка формы Form1"
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "XSLT-процессор (Турчин Д.Е., каф. ИиАПС)";
            this.Font = new System.Drawing.Font("Arial", 10, FontStyle.Bold);
            textBox1.Text = @"H:\XML\document.xml";
            textBox2.Text = @"H:\XSLT\stylesheet.xslt";
            textBox3.Text = @"H:\output\out.htm";
        }

        // Обработчик события "Нажатие на кнопку buttonTransform"
        private void buttonTrasform_Click(object sender, EventArgs e)
        {
            XslCompiledTransform xslt = new XslCompiledTransform();
            xslt.Load(textBox2.Text);
            xslt.Transform(textBox1.Text, textBox3.Text);

            StreamReader sr = new StreamReader(textBox3.Text);
            webBrowser1.DocumentText = sr.ReadToEnd();
            sr.Close();
        }

        // Обработчик события "Нажатие на кнопку button1"
        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "XML-файлы (*.xml)|*.xml";
            openFileDialog1.ShowDialog();
            textBox1.Text = openFileDialog1.FileName;
        }

        // Обработчик события "Нажатие на кнопку button2"
        private void button2_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "XSLT-файлы (*.xslt)|*.xslt" +
                                     "|XSL-файлы (*.xsl)|*xsl";
            openFileDialog1.ShowDialog();
            textBox2.Text = openFileDialog1.FileName;
        }

        // Обработчик события "Нажатие на кнопку button3"
        private void button3_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Все файлы (*.*)|*.*";
            openFileDialog1.ShowDialog();
            textBox3.Text = openFileDialog1.FileName;
        }
    }
}
```

Результат работы приложения после выполнения XSLT-преобразования показан на рис. 1.8. ❒

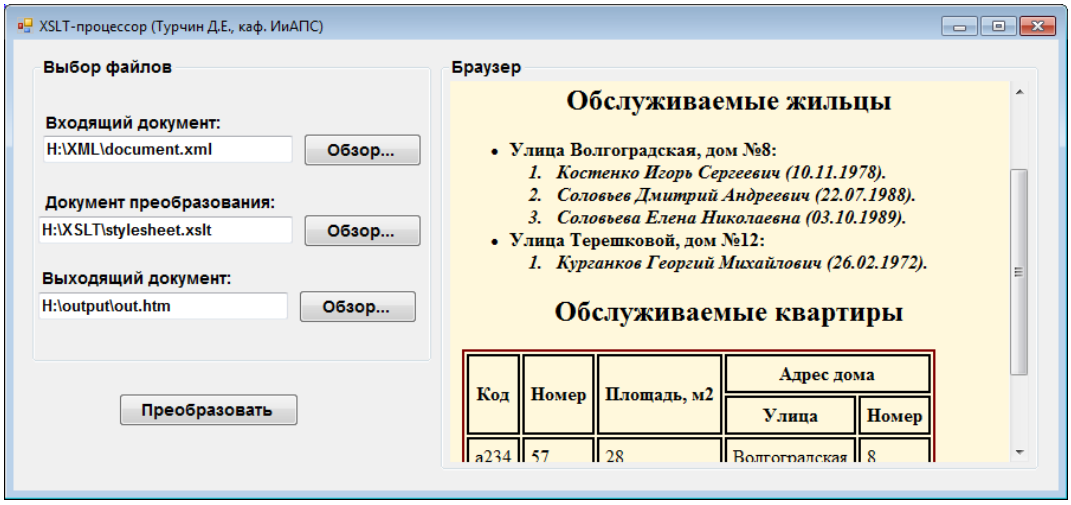


Рис. 1.8. Работа приложения

1.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать исходный XML-документ, содержащий указанные данные (табл. 1.1). Создать не менее двух экземпляров элемента первого уровня и от одного до трёх экземпляров элементов последующих уровней.
3. Разработать таблицу стилей XSLT для преобразования исходного XML-документа в XML-документ с заданной структурой (табл. 1.2).
4. Разработать таблицу стилей XSLT для преобразования XML-документа в формат HTML. Одну часть данных требуется отобразить в форме многоуровневого списка (табл. 1.3), другую часть – в виде таблицы.
5. Разработать на языке C# приложение Windows Forms, обеспечивающее преобразование указанного документа XML с помощью выбранной таблицы стилей XSLT.
6. Оформить и защитить отчет по лабораторной работе.

Таблица 1.1

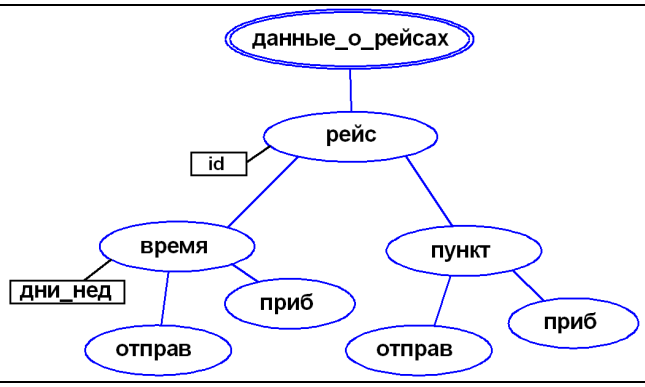
Варианты заданий для разработки XML-документов

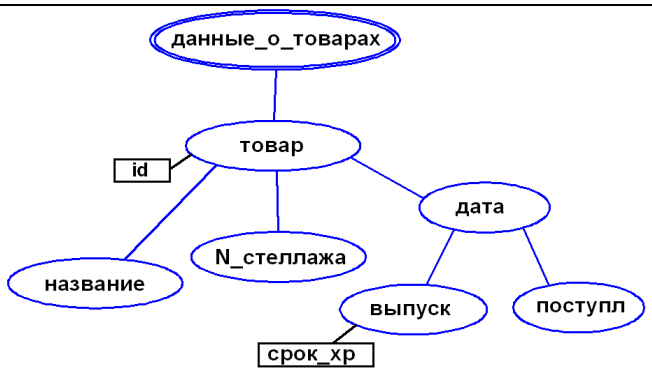
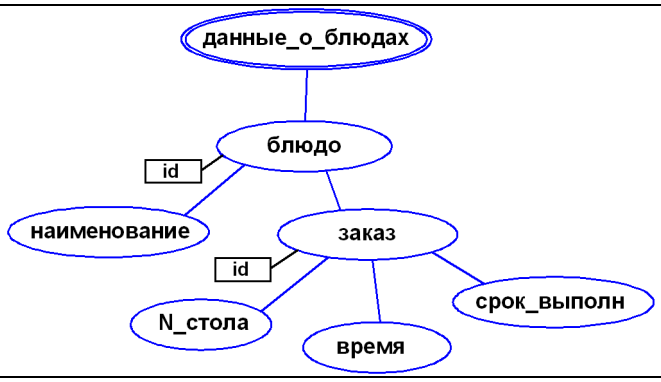
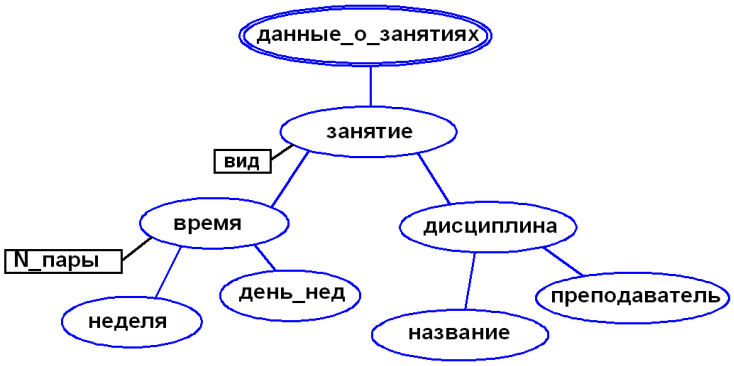
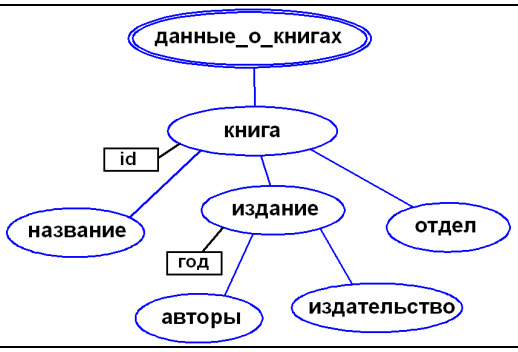
№ вар.	Данные для разработки XML-документа
1, 13	***Расписание рейсов междугородных автобусов.*** Документ должен содержать сведения о пунктах отправления (код, название (например, «Кемеровский АВ», «Прокопьевская АС»), пунктах прибытия (код, название), рейсах (код, дни недели (например, «ежедневно», «еженедельно: пн, чт»), время отправления, время прибытия, цена)
2, 14	***Товарный склад.*** Документ должен содержать сведения о стеллажах (код, тип, число ярусов), товарах (код, наименование, количество, вес, дата поступления, дата выпуска, срок хранения, цена и др.), фирме-производителе (название, контактная информация (телефон, адрес))
3, 15	***Ресторан.*** Документ должен содержать сведения о сделанных заказах (код, дата, время, стол, выполнение (да нет) и др.), блюдах (код, название, время приготовления, цена) и ингредиентах (название, объем)
4, 16	***Расписание занятий.*** Документ должен содержать сведения по дням недели о проводимых занятиях (номер пары, название учебной дисциплины, тип занятия (лекция, практика, лабораторная), аудитория (номер), ФИО и должность преподавателя). Расписание отличается на четных и нечетных неделях
5, 17	***Книжный магазин.*** Документ должен содержать сведения об отделах книжного магазина (код, название), книгах (код, название, авторы, издательство, год, число страниц, цена) и журналах (код, название, год, номер, цена)
6, 18	***Кинотеатр.*** Документ должен содержать сведения о зрительных залах (код, номер, число мест), фильмах (код, название, год выпуска, кинокомпания, режиссёр, актеры и др.), сеансах (код, дата, время начала, цена билета)
7, 19	***Проектно-строительная компания.*** Документ должен содержать сведения об отделах (код, название), сотрудниках (код, фамилия, имя, отчество, пол, дата рождения, дата устройства, должность) и выполняемых ими проектах (код, наименование, срок выдачи, стоимость)
8, 20	***Обувной магазин.*** Документ должен содержать сведения об отделах (код, название),

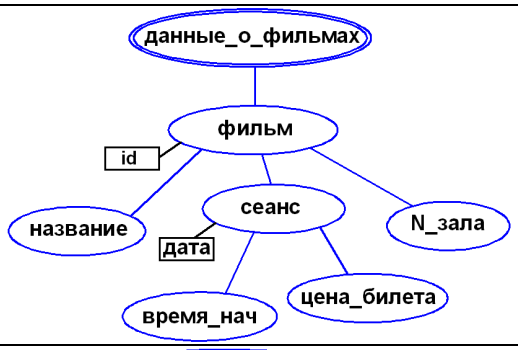
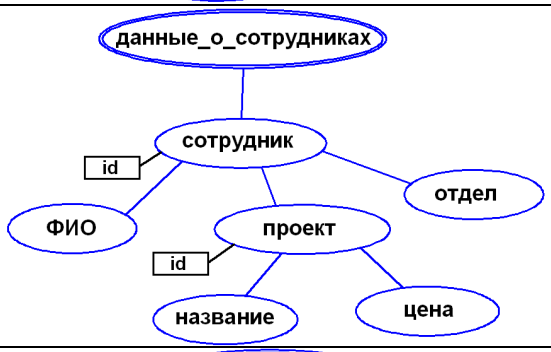
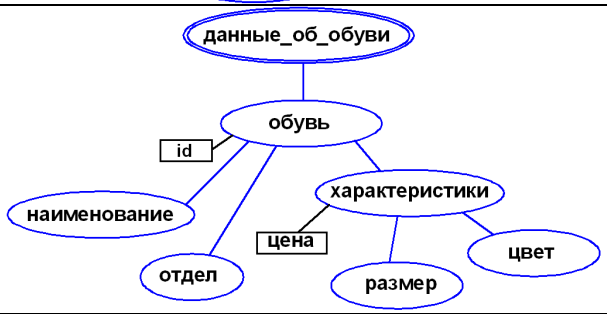
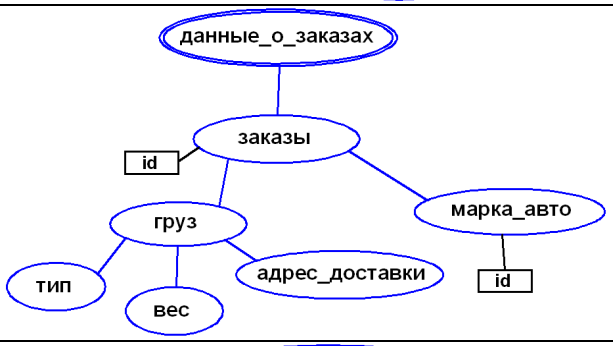
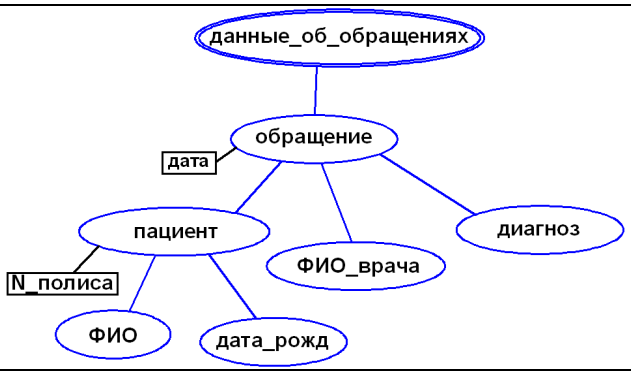
№ вар.	Данные для разработки XML-документа
	обуви (код, название, размер, материал, цвет, производитель, цена, скидка) и аксессуарах (код, название, цена)
9, 21	***Перевозка грузов.*** Документ должен содержать сведения о водителях (код, ФИО, дата рождения, телефон и др.), автомобилях (код, марка, тип кузова и др.) и заказах (код, дата/время, адрес доставки, тип груза, вес груза и др.)
10, 22	***Поликлиника.*** Документ должен содержать сведения о врачах (код, ФИО, специальность), пациентах (код, ФИО, дата рождения, пол, номер полиса) и обращениях (код, дата обращения, болезнь, продолжительность, результат лечения и др.)
11, 23	***Банковские услуги.*** Документ должен содержать сведения о клиентах (код, фамилия, имя, отчество, паспортные данные, телефон), кредитах (код, вид, сумма, дата, срок, процентная ставка) и вкладах (код, вид, номер счёта, сумма, процентная ставка, дата)
12, 24	***Прогноз погоды.*** Документ должен содержать сведения о днях (код, дата, день недели), времени суток (тип (утро, день, вечер, ночь), температура, давление, влажность), ветре (направление, скорость) и атмосферных явлениях (облачность, осадки)

Таблица 1.2

Варианты заданий для преобразования исходного XML-документа в XML-документ с другой структурой

№ вар.	Требуемая структура выходного XML-документа
1, 13	

№ вар.	Требуемая структура выходного XML-документа
2, 14	
3, 15	
4, 16	
5, 17	

№ вар.	Требуемая структура выходного XML-документа
6, 18	
7, 19	
8, 20	
9, 21	
10, 22	

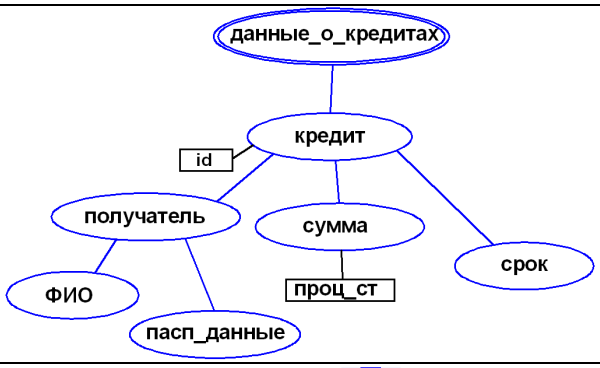
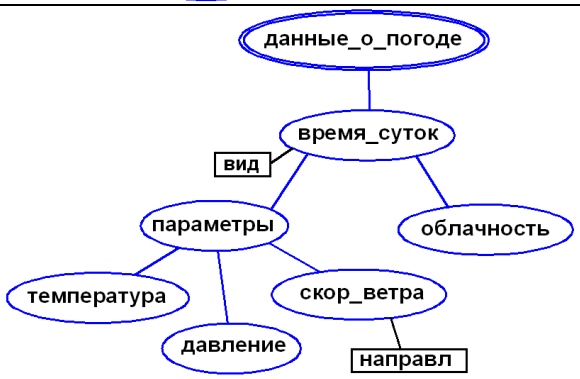
№ вар.	Требуемая структура выходного XML-документа
11, 23	
12, 24	

Таблица 1.3

Варианты заданий для преобразования исходного XML-документа в документ HTML

№ вар.	Данные, размещаемые в многоуровневом списке	Данные, размещаемые в таблице
1, 13	Пункт отправления (код, название): - Пункты прибытия (код, название).	Рейс (код, время отправления, цена), пункт отправления (название), пункт прибытия (название).
2, 14	Стеллаж (код, тип): - Товары (наименование, количество, производитель).	Товар (код, наименование, дата выпуска, цена), стеллаж (код).
3, 15	Заказ (код, дата, время): - Блюда (название, цена).	Блюдо (код, название, время приготовления), ингредиент (название, объём).
4, 16	День недели (название, чётность): - Занятия (номер, название дисциплины, тип).	Занятие (номер, название дисциплины, аудитория), день недели (название), неделя (чётность).
5, 17	Отдел (код, название): - Книга (автор, название, це-	Книга (код, название, издательство, год, число страниц), отдел (назва-

№ вар.	Данные, размещаемые в многоуровневом списке	Данные, размещаемые в таблице
	на).	ние).
6, 18	Зрительный зал (номер): - Фильмы (название, кинокомпания, режиссёр)	Фильм (код, название), сеанс (код, дата, время начала, цена билета).
7, 19	Отдел (код, название): - Сотрудники (ФИО, пол, дата рождения).	Сотрудник (ФИО, должность), проект (код, наименование, срок выдачи).
8, 20	Отдел (код, название): - Обувь (название, цена, скидка).	Обувь (код, название, размер, производитель), отдел (название).
9, 21	Водитель (ФИО, телефон). - Автомобили (марка, тип кузова)	Заказ (код, дата, адрес доставки), автомобиль (код, марка).
10, 22	Врач (ФИО, специальность): - Пациенты (ФИО, дата рождения, пол).	Обращение (код, дата, болезнь), пациент (ФИО, номер полиса).
11, 23	Клиент (код, ФИО): - Кредиты (сумма, процент, дата).	Вклад (код, вид, сумма, процент), клиент (ФИО, телефон).
12, 24	День (дата): - Времена суток (тип, температура, давление, влажность).	Время суток (тип), ветер (направление, скорость), день (дата, день недели).

1.4. Контрольные вопросы

1. Из каких частей состоит спецификация XSL?
2. Для какой цели предназначена технология XSLT?
3. В чём заключаются основные ограничения XSLT?
4. Что представляет собой таблица стилей XSLT?
5. Что понимают под XSLT-процессором?
6. Какие этапы включает в себя процесс XML-документа с помощью XSLT-процессора?
7. Какой корневой элемент используется для таблицы стилей XSLT?
8. Что понимают под шаблонами преобразования в таблице стилей XSLT?
9. Каково назначение элемента **xsl:for-each**?

10. Как задаётся тип выходного документа в таблице стилей XSLT?

11. Каким образом входной XML-документ связывается с таблицей стилей XSLT?

12. Для чего предназначен класс **XslCompiledTransform**?

2. ОСНОВЫ ИСПОЛЬЗОВАНИЯ ПОЛИМОРФИЗМА В СЕМЕЙСТВАХ КЛАССОВ НА ЯЗЫКЕ C#

2.1. Цель и задачи работы

Цель работы – прибрести умение использовать полиморфизм в семействах классов, связанных отношением наследования, путём использования в классах виртуальных и абстрактных методов.

Основные задачи:

• освоить создание виртуальных методов в базовых классах и их переопределение в производных классах;

• научиться использовать абстрактные классы для построения иерархии классов.

Работа рассчитана на 4 часа.

2.2. Основные теоретические сведения

2.2.1. Понятие полиморфизма. Виртуальные методы. Класс System.Object и его методы

Понятие полиморфизма. Виртуальные методы.

Открыто унаследованные методы (свойства, индексаторы и др.) работают идентично для всех базовых классов. Часто требуется, чтобы производные классы имели собственные версии метода, определённого в базовом классе. Для этого используется механизм, называемый полиморфизмом (*polymorphism*).

В общем случае под ***полиморфизмом*** понимают изменение поведения базового класса в производном классе. Концепция полиморфизма позволяет одним именем (например, одним объявлением переменной) обозначать объекты, относящиеся к разным классам, но связанные с общим базовым классом. Любой объект, обозначенный таким именем, может обладать общим набором операций.

Полиморфизм наряду с инкапсуляцией и наследованием представляет одну из важнейших концепций объектно-ориентированного программирования. Применение полиморфиз-

ма позволяет строить легко расширяемое и гибкое программное обеспечение.

В программировании с полиморфизмом тесно связаны такие понятия, как виртуальные методы, переопределение методов, абстрактные классы и абстрактные методы.

Виртуальными методами называют методы, у которых в заголовке присутствует ключевое слово **virtual**:

```
public virtual void GetTotalValue() ...
```

Виртуальные методы создаются в базовых классах и переопределяются в производных классах. Производные классы не должны в обязательном порядке переопределять виртуальные методы базового класса.

Объявление метода «виртуальным» означает, что все ссылки на этот метод будут разрешаться на стадии выполнения программы, а не на стадии компиляции. Такой механизм называется ***поздним связыванием*** (англ., *late binding*).

Для обеспечения позднего связывания необходимо, чтобы адреса виртуальных методов хранились там, где ими можно в любой момент воспользоваться. С этой целью компилятор формирует ***таблицу виртуальных методов*** (англ., *Virtual Method Table, VMT*), в которую записываются адреса виртуальных методов в порядке описания в классе.

Если в производном классе требуется переопределить виртуальный метод, то в заголовке соответствующего метода используется ключевое слово **override**:

```
public override void GetTotalValue()
{ ...
     base.GetTotalValue()
... }
```

Переопределенный виртуальный метод должен обладать таким же набором параметров, как и одноименный метод базового класса.

Виртуальные методы базового класса определяют интерфейс всей иерархии классов. Этот интерфейс может расширяться в потомках за счет добавления новых виртуальных методов.

При описании классов рекомендуется определять в качестве виртуальных те методы, которые в производных классах должны реализовываться по-другому.

Класс System.Object и его методы.

В .NET Framework базовым классом для всех остальных классов и типов является класс **System.Object**. В языке C# данный класс носит имя **object**. Таким образом, каждый значащий и ссылочный тип в C# неявно наследуют класс **object**. В частности это означает, что переменная ссылочного типа **object** может ссылаться на объект любого другого типа.

Класс **object** имеет следующие открытые виртуальные методы:

- **bool Equals(object obj)** – возвращает значение **true**, если параметр **obj** является таким же, как и вызывающий объект;
- **int GetHashCode()** – формирует хеш-код объекта и возвращает целое число, однозначно идентифицирующее объект; может использоваться в алгоритмах, где хеширование используется в качестве средства доступа к хранимым объектам;
- **srting ToString()** – по умолчанию возвращает для ссылочных типов полное имя класса в виде строки, а для значимых типов – значение величины, преобразованное в строку; данный метод переопределяют для того, чтобы можно было выводить информацию о состоянии объекта.

Переопределить метод **ToString()** можно при помощи следующего объявления метода:

```
public override string ToString()
{ ... }
```

В Visual Studio после набора слова **override** внутри класса и нажатии клавиши пробела IntelliSense автоматически отображает список всех переопределяемых элементов базового класса (рис. 2.1).

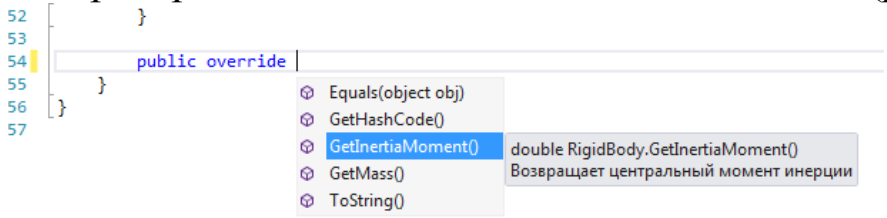


Рис. 2.1. Просмотр списка переопределяемых элементов в Visual Studio

❒ ***Пример* 2.1. *Разработка классов с виртуальными методами и их переопределением*.**

Требуется разработать класс **BaseCar** (Легковой автомобиль в базовой комплектации), который содержит виртуальный метод **int GetCost()** (Определить стоимость) и переопределяет методы базового класса **System.Object**. Кроме того, требуется разработать производный от него класс **ForcedCar** (Легковой автомобиль с гидроусилителем руля), который переопределять метод **int GetCost()**.

Исходный код указанных классов представлен в листингах 2.1, 2.2 и 2.3.

Исходный код класса **Program** консольного приложения приведён в листинге 2.4.

Листинг 2.1. Исходный код класса **BaseCar** (часть 1)

```
 9      /// <summary>
10      /// Представляет легковые автомобили в базовой комплектации
11      /// </summary>
12      class BaseCar
13      {
14          protected string mark; // Марка
15          protected int year; // Год выпуска
16          protected int cost; // Базовая стоимость, руб.
17
18          /// <summary>
19          /// Конструктор по умолчанию
20          /// </summary>
21          public BaseCar()
22          {
23              mark = "Daewoo Nexia";
24              year = 2014;
25              cost = 290000;
26          }
27
28          /// <summary>
29          /// Конструктор с параметрами
30          /// </summary>
31          /// <param name="mark">Марка</param>
32          /// <param name="year">Год выпуска</param>
33          /// <param name="cost">Стоимость, руб.</param>
34          public BaseCar(string mark, int year, int cost)
35          {
36              this.mark = mark;
37              this.year = year;
38              this.cost = cost;
39          }
40
41          /// <summary>
42          /// Возвращает результат сравнения объектов
43          /// </summary>
44          /// <param name="obj">объект</param>
45          /// <returns>Результат сравнения</returns>
46          public override bool Equals(object obj)
47          {
48              return base.Equals(obj);
49          }
50
51          /// <summary>
52          /// Возвращает хеш-код экземпляра класса BaseCar
53          /// </summary>
54          /// <returns>Хеш-код объекта</returns>
55          public override int GetHashCode()
56          {
57              return base.GetHashCode();
58          }
```

Листинг 2.2. Исходный код класса **BaseCar** (часть 2)

```
60          /// <summary>
61          /// Возвращает строку с данными об автомобиле
62          /// </summary>
63          /// <returns>Строка с данными об автомобиле</returns>
64          public override string ToString()
65          {
66              return string.Format("Автомобиль: {0}\n" +
67                                   "Год выпуска: {1}\n" +
68                                   "Базовая стоимость: {2} руб.\n",
69                                   mark, year, cost);
70          }
71
72          /// <summary>
73          /// Определяет стоимость в зависимости от года выпуска
74          /// </summary>
75          /// <returns>Стоимость, руб.</returns>
76          public virtual int GetCost()
77          {
78              // Степень уценки в зависимости от года выпуска
79              double dis = 1 - (DateTime.Today.Year - year) / 85;
80              return Convert.ToInt32(cost * dis);
81          }
82      }
```

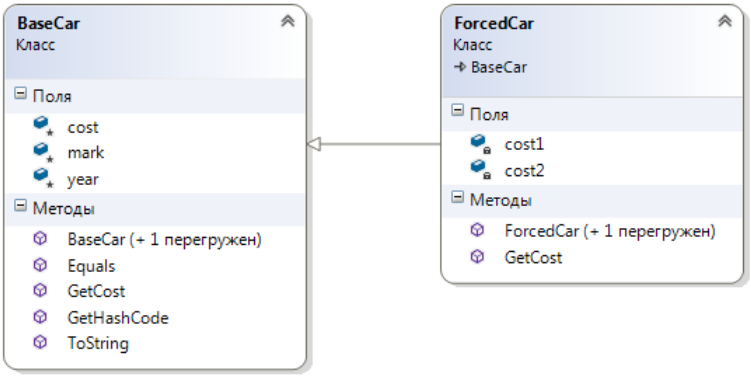


Рис. 2.2. Диаграмма классов проекта

Листинг 2.3. Исходный код класса **ForcedCar** (часть 2)

```
    /// <summary>
    /// Представляет легковые автомобили с гидроусилителем руля
    /// </summary>
    class ForcedCar : BaseCar
    {
        int cost1; // Стоимость гидроусилителя
        int cost2; // Стоимость монтажа

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public ForcedCar()
            : base()
        {
            cost1 = 7000;
            cost2 = 3000;
        }

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="mark">Марка</param>
        /// <param name="year">Год выпуска</param>
        /// <param name="cost">Базовая стоимость, руб.</param>
        /// <param name="cost1">Стоимость гидроусилителя, руб.</param>
        /// <param name="cost2">Стоимость монтажа, руб.</param>
        public ForcedCar(string mark, int year, int cost, int cost1, int cost2)
            : base(mark, year, cost)
        {
            this.cost1 = cost1;
            this.cost2 = cost2;
        }

        /// <summary>
        /// Возвращает полную стоимость
        /// </summary>
        /// <returns>Полную стоимость, руб.</returns>
        public override int GetCost()
        {
            return (base.GetCost() + cost1 + cost2);
        }
    }
```

Листинг 2.4. Исходный код класса **Program** консольного приложения

```
 9      class Program
10      {
11          static void Main(string[] args)
12          {
13              Console.Title = "Работа с виртуальными методами";
14
15              BaseCar car1 = new BaseCar();
16              ForcedCar car2 = new ForcedCar();
17              BaseCar car3 = new BaseCar("Daewoo Matiz", 2013, 230000);
18              ForcedCar car4 = new ForcedCar("Daewoo Matiz", 2014, 230000, 6000, 3000);
19
20              BaseCar[] cars = new BaseCar[] { car1, car2, car3, car4 };
21
22              foreach (BaseCar car in cars)
23              {
24                  Console.WriteLine("Идентификатор: {0}", car.GetHashCode());
25                  Console.WriteLine(car.ToString());
26                  Console.WriteLine("Полная стоимость: {0} руб.\n", car.GetCost());
27              }
28
29              Console.WriteLine("Результаты сравнения объектов:");
30              Console.WriteLine(car1.Equals(car1));
31              car1 = car3;
32              Console.WriteLine(car3.Equals(car1));
33              car3 = car2;
34              Console.WriteLine(car3.Equals(car1));
35
36              Console.Read();
37          }
38      }
39  }
```

Результат работы консольного приложения представлен на рис. 2.3. ❒

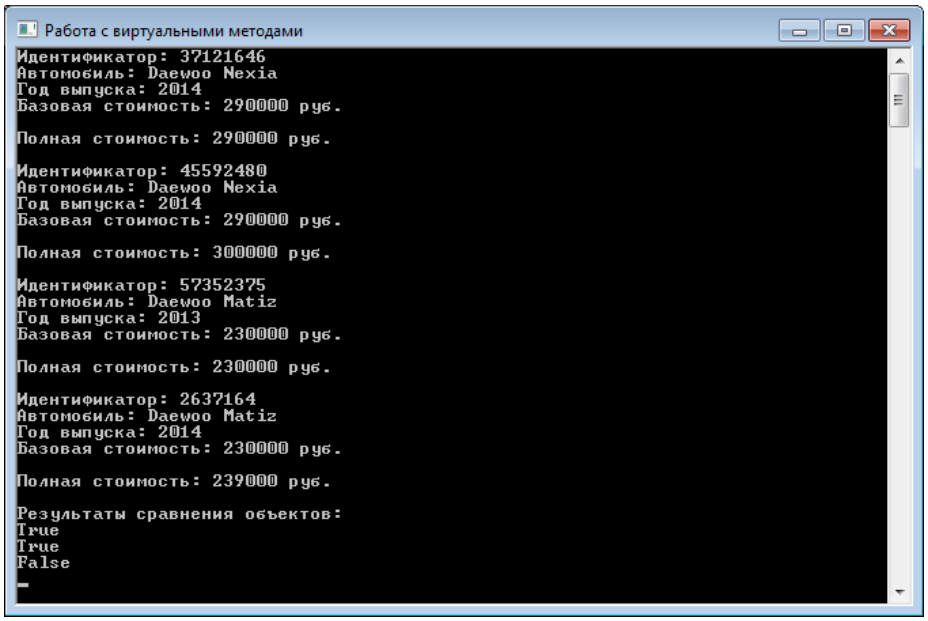


Рис. 2.3. Работа консольного приложения

2.2.2. Абстрактные классы и методы. Запечатанные классы

Абстрактные классы и методы. Запечатанные классы.

При разработке иерархии классов для исключения повторяющегося кода часто требуется выделить их общие свойства классов в один базовый класс. При этом полученный базовый класс может представлять очень общую сущность, и создание экземпляров такого класса не будет иметь смысла. В этом случае класс объявляют как абстрактный (*abstract*) и в нём присутствуют абстрактные методы.

Класс называется ***абстрактным***, если он имеет хотя бы один абстрактный метод. Метод называется ***абстрактным***, если при его определении задана его сигнатура, но не задана реализация метода. Абстрактный метод представляет собой виртуальный метод, переопределяемый производными классами.

Объявление абстрактных классов и абстрактных методов должно сопровождаться модификатором **abstract.**

```
public abstract class Shape
{ ...
     public abstract void Draw();
... }
```

Модификатор **abstract** может применяться только в методах экземпляра, но не в статических методах.

Поскольку абстрактные классы не являются полностью определенными классами, то нельзя создавать объекты абстрактных классов. Кроме того, абстрактные классы могут содержать и полностью определённые методы, в отличие от сходного с ним по назначению специального вида классов, называемых интерфейсом.

Абстрактные классы служат только для создания классов-потомков. Обычно в абстрактном классе задается набор методов, которые каждый из потомков будет реализовывать по-своему. Если класс, являющийся производным от абстрактного класса, не переопределяет все абстрактные методы, то он также является абстрактным и должен иметь модификатор **abstract.**

Полезным видом классов являются ***запечатанные*** (бесплодные) классы, для которых запрещается строить производные классы путём наследования. Примером запечатанного класса является класс **String** встроенной библиотеки .NET Framework.

Запечатанный класс задается с помощью ключевого слова **sealed**.

Если требуется использовать функциональность запечатанного класса, то используют не наследование, а включение.

Иногда требуется не запечатывать класс целиком, а просто предотвратить переопределение некоторых виртуальных методов в производных классах. Для этого в заголовке метода указывается ключевое слово **sealed,** и такой метод называется ***запечатанным виртуальным методом***.

❒ ***Пример* 2.2. *Разработка иерархии классов, начиная с абстрактного класса*.**

Требуется разработать семейство из трёх классов, связанных иерархией наследования:

• **Твёрдое тело**: плотность материала тела; Определить объём; Определить массу; Определить центральный момент инерции:

• **Сегмент параболоида вращения**: радиус основания, высота (рис. 2.4);

• **Усечённый параболоид вращения**: радиус малого основания.

Рис. 2.4. Параболоиды вращения

Создадим в Visual Studio решение **Polymorphism**, с проектом консольного приложения **ConsoleApplication**. Добавим в решение проект библиотеки классов **PolymorphClasses**.

Реализуем в библиотеке классов **PolymorphClasses** следующую иерархию классов:

• **RigidBody** (Твёрдое тело) – абстрактный класс;

• **Paraboloid** (Параболоид вращения) – наследник класса **RigidBody**:

• **TruncParab** (Усечённый параболоид вращения) – наследник класса **Paraboloid**.

Исходный код классов **RigidBody**, **Paraboloid** и **TruncParab** приведён в листингах 2.5 – 2.9.

Добавим в проект библиотеки классов схему классов (рис. 2.5).

Листинг 2.5. Исходный код абстрактного класса **RigidBody** (Твёрдое тело)

```
    /// <summary>
    /// Представляет твёрдые тела (абстрактный класс)
    /// </summary>
    public abstract class RigidBody
    {
        protected static int count = 0; // Общее число твёрдых тел
        protected double dencity; // Плотность материала тела, кг/м3

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public RigidBody()
        { dencity = 1000; count++; }

        /// <summary>
        /// Параметрический конструктор
        /// </summary>
        /// <param name="_d">Плотность материала тела, кг/м3</param>
        public RigidBody(double _d)
        { dencity = _d; count++; }

        /// <summary>
        /// Возвращает число твёрдых тел
        /// </summary>
        public static int Count
        {
            get { return count; }
        }

        /// <summary>
        /// Возвращает или устанавливает плотность тела, кг/м3
        /// </summary>
        public double Dencity
        {
            get { return dencity; }
            set
            {
                if (value <= 0)
                    throw new Exception("Плотность тела должна быть > 0!");
                dencity = value;
            }
        }

        /// <summary>
        /// Возвращает массу
        /// </summary>
        /// <returns>Масса, кг</returns>
        public abstract double GetMass();

        /// <summary>
        /// Возвращает центральный момент инерции
        /// </summary>
        /// <returns>Центральный момент инерции, кг*м2</returns>
        public abstract double GetInertiaMoment();
    }
```

Листинг 2.6. Исходный код класса **Paraboloid** (Параболоид вращения). Часть 1

```
    /// <summary>
    /// Представляет параболоиды вращения, являющиеся твёрдыми телами
    /// </summary>
    public class Paraboloid : RigidBody
    {
        protected double radius; // Радиус основания, м
        protected double height; // Высота, м

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public Paraboloid()
            : base()
        {
            radius = 1;
            height = 1;
        }

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="_d">Плотность, кг/м3</param>
        /// <param name="_r">Радиус основания, м</param>
        /// <param name="_h">Высота, м</param>
        public Paraboloid(double _d, double _r, double _h)
            : base(_d)
        {
            radius = _r;
            height = _h;
        }

        /// <summary>
        /// Возвращает или устанавливает радиус, м
        /// </summary>
        public virtual double Radius
        {
            get { return radius; }
            set
            {
                if (value <= 0)
                    throw new Exception("Радиус должен быть > 0!");
                radius = value;
            }
        }
```

Листинг 2.7. Исходный код класса **Paraboloid** (Параболоид вращения). Часть 2

```
54          /// <summary>
55          /// Возвращает или устанавливает высоту, м
56          /// </summary>
57          public double Height
58          {
59              get { return height; }
60              set
61              {
62                  if (value <= 0)
63                      throw new Exception("Высота должна быть > 0!");
64                  height = value;
65              }
66          }
67
68          /// <summary>
69          /// Возвращает строку с характеристиками объекта
70          /// </summary>
71          /// <returns>Характеристики параболоида вращения</returns>
72          public override string ToString()
73          {
74              return (string.Format("Характеристики параболоида вращения:\n" +
75                                    "- плотность: {0} кг/м3\n" +
76                                    "- радиус основания: {1} м\n" +
77                                    "- высота: {2} м", dencity, radius, height));
78          }
79
80          /// <summary>
81          /// Возвращает объём
82          /// </summary>
83          /// <returns>Объём, м3</returns>
84          public virtual double GetVolume()
85          {
86              return (Math.PI * radius * radius * height / 2);
87          }
88
89          /// <summary>
90          /// Возвращает массу
91          /// </summary>
92          /// <returns>Масса, кг</returns>
93          public override double GetMass()
94          {
95              return (dencity * GetVolume());
96          }
97
98          /// <summary>
99          /// Возвращает центральный момент инерции
100         /// </summary>
101         /// <returns>Центральный момент инерции, кг*м2</returns>
102         public override double GetInertiaMoment()
103         {
104             return (GetMass() * radius * radius / 5);
105         }
106     }
```

Листинг 2.8. Исходный код класса **TruncParab** (Усечённый параболоид вращения). Часть 1

```
    /// <summary>
    /// Представляет усечённые параболоиды вращения,
    /// являющиеся твёрдыми телами (запечатанный класс)
    /// </summary>
    public sealed class TruncParabol : Paraboloid
    {
        double radiusSmall; // Радиус малого основания, м

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public TruncParabol()
            : base()
        { radiusSmall = Radius / 2; }

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="_d">Плотность, кг/м3</param>
        /// <param name="_r">Радиус большого основания, м</param>
        /// <param name="_rs">Радиус малого основания, м</param>
        /// <param name="_h">Высота, м</param>
        public TruncParabol(double _d, double _r, double _rs, double _h)
            : base(_d, _r, _h)
        { radiusSmall = _rs; }

        /// <summary>
        /// Возвращает или устанавливает радиус большого основания, м
        /// </summary>
        public override double Radius
        {
            get { return base.Radius; }
            set
            {
                if (value > base.Radius)
                    throw new Exception("Радиус малого основания не должен быть" +
                                        " больше радиуса большого основания!");
                base.Radius = value;
            }
        }

        /// <summary>
        ///  Возвращает или устанавливает радиус малого основания, м
        /// </summary>
        public double RadiusSmall
        {
            get { return radiusSmall; }
            set
            {
                if (value <= 0)
                    throw new Exception("Радиус малого основания должен быть > 0!");
                radiusSmall = value;
            }
        }
```

Листинг 2.9. Исходный код класса **TruncParab** (Усечённый параболоид вращения). Часть 2

```
64          /// <summary>
65          /// Возвращает строку с характеристиками объекта
66          /// </summary>
67          /// <returns></returns>
68          public override string ToString()
69          {
70              return (base.ToString() +
71                  string.Format("\n- радиус малого основания: {0} м", radiusSmall));
72          }
73
74          /// <summary>
75          /// Возвращает объём
76          /// </summary>
77          /// <returns>Объём, м3</returns>
78          public override double GetVolume()
79          {
80              return (base.GetVolume() +
81                  Math.PI * height * radiusSmall * radiusSmall /2);
82          }
83
84          /// <summary>
85          /// Возвращает массу
86          /// </summary>
87          /// <returns>Масса, кг</returns>
88          public override double GetMass()
89          {
90              return (dencity * GetVolume());
91          }
92
93          /// <summary>
94          /// Возвращает центральный момент инерции
95          /// </summary>
96          /// <returns>Центральный момент инерции, кг*м2</returns>
97          public override double GetInertiaMoment()
98          {
99              return (base.GetInertiaMoment() +
100                 GetMass() * radiusSmall * radiusSmall / 2);
101         }
102     }
```

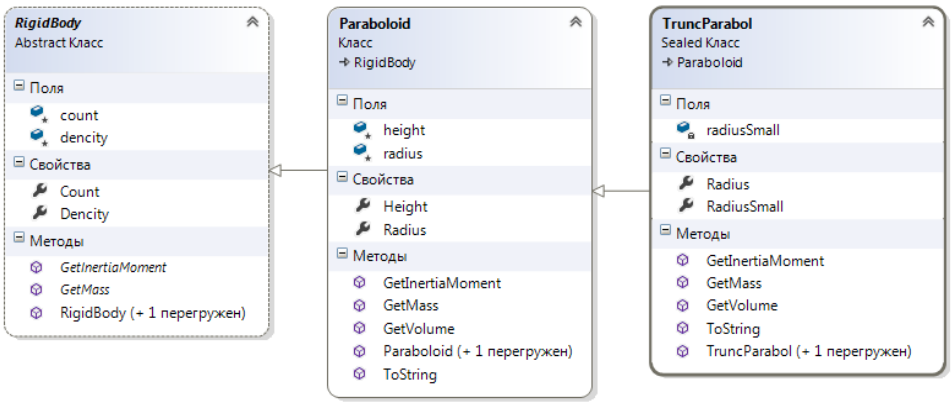


Рис. 2.5. Схема классов проекта

Исходный код консольного приложения представлен в листинге 2.10.

Листинг 2.10. Исходный код метода **Main()** консольного приложения

```
10      class Program
11      {
12          static void Main(string[] args)
13          {
14              Console.Title = "Работа с семейством полиморфных классов";
15              Console.WriteLine("*********** Твёрдые тела ***********\n");
16
17              Paraboloid pb1 = new Paraboloid();
18              Console.WriteLine(pb1.ToString());
19
20              string s = "- объём: {0:f2} м3\n" +
21                         "- масса: {1:f2} кг\n" +
22                         "- момент инерции: {2:f2} кг*м2\n";
23
24              Console.WriteLine(s, pb1.GetVolume(), pb1.GetMass(),
25                  pb1.GetInertiaMoment());
26
27              Paraboloid pb2 = new Paraboloid(800, 0.7, 1.2);
28              Console.WriteLine(pb2.ToString());
29              Console.WriteLine(s, pb2.GetVolume(), pb2.GetMass(),
30                  pb2.GetInertiaMoment());
31
32              TruncParabol pb3 = new TruncParabol();
33              Console.WriteLine(pb3.ToString());
34              Console.WriteLine(s, pb3.GetVolume(), pb3.GetMass(),
35                  pb3.GetInertiaMoment());
36
37              // Экземпляру базового класса присваивается ссылка на
38              // экземпляр производного класса
39              Paraboloid pb4 = new TruncParabol(750, 0.6, 0.25, 0.9);
40              // Вызов через динамическое связывание методов
41              // экземпляра производного класса
42              Console.WriteLine(pb4.ToString());
43              Console.WriteLine(s, pb4.GetVolume(), pb4.GetMass(),
44                  pb4.GetInertiaMoment());
45
46              Console.WriteLine("Всего твёрдых тел: {0} шт\n", RigidBody.Count);
47
48              // Массив параболоидов вращения
49              Paraboloid[] parabs = new Paraboloid[4];
50              parabs[0] = pb1;
51              parabs[1] = pb2;
52              parabs[2] = pb3; // Ссылка на объект производного класса
53              parabs[3] = pb4;
54
55              // Цикл для совместного вызова методов экземпляров
56              // базового и производного классов
57              foreach (Paraboloid p in parabs)
58              {
59                  Console.WriteLine("Вычисленные характеристики тела:\n" + s,
60                      p.GetVolume(), p.GetMass(), p.GetInertiaMoment());
61              }
62              Console.Read();
63          }
64      }
```

Результат работы консольного приложения показан на рис. 2.5. ❒

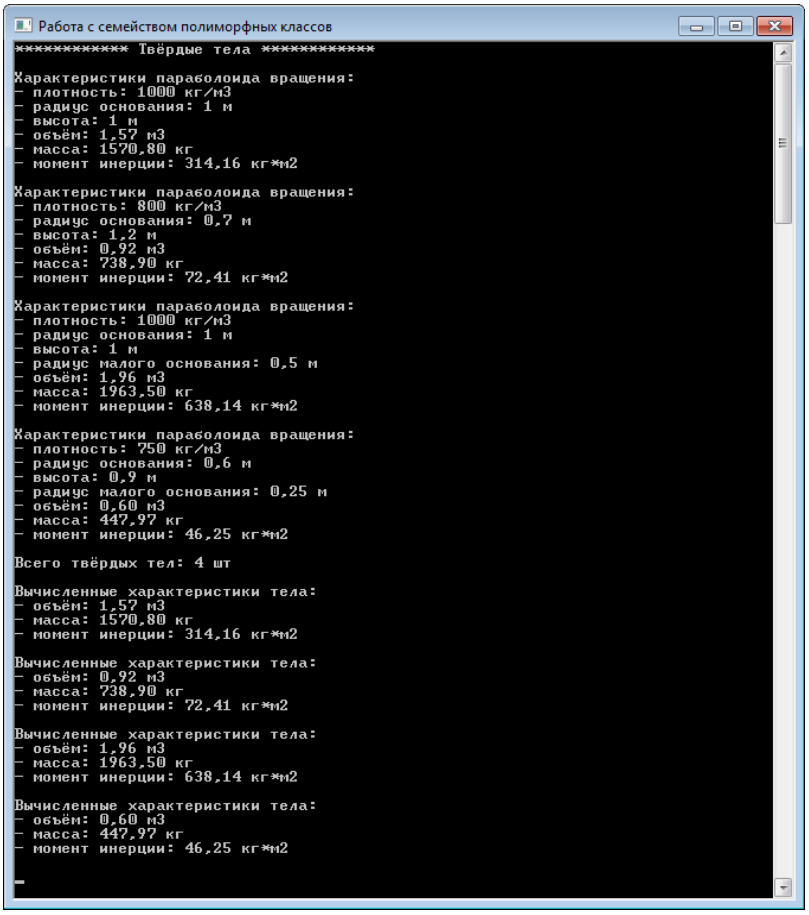


Рис. 2.6. Результат работы консольного приложения

2.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать базовый класс, содержащий виртуальные методы (табл. 2.1), и производный от него класс, в котором эти методы переопределены. Дополнительно требуется переопределить методы класса **System.Object**.
3. Построить иерархию классов, начиная с абстрактного класса (табл. 2.2). Предусмотреть виртуальные методы в проектируемых классах, а также переопределение этих методов в клас-

сах-потомках. Разместить полученные классы в DLL. Разработать консольное приложение, демонстрирующее полиморфизм построенного семейства классов.

4. Оформить и защитить отчет по лабораторной работе.

Таблица 2.1

Варианты заданий для разработки классов, содержащих виртуальные методы и их переопределение

№ вар.	Классы	Виртуальные методы
1, 5, 9	***Базовый***. Здание (площадь, высота, число этажей, износ) ***Производный***. Жилое здание (общая жилплощадь)	Определить стоимость ремонта
2, 6, 10	***Базовый***. Кредит (дата выдачи, сумма, срок, процент) ***Производный***. Кредит с залогом имущества (стоимость имущества).	Определить сумму возврата
3, 7, 11	***Базовый***. Квартира (адрес дома, номер, площадь, число комнат, этаж). ***Производный***. Квартира с лоджией (площадь лоджии)	Определить цену
4, 8, 12	***Базовый***. Перевозка грузов (дата, адрес доставки, тип груза, вес груза) ***Производный***. Перевозка грузов за пределы города (расстояние от города).	Определить стоимость перевозки

Таблица 2.2

Варианты заданий для разработки иерархии классов, начиная от абстрактного класса

№ вар.	Данные для разработки иерархии классов
1, 9, 17	***Класс* 1. Объёмная фигура** (Определить объём, Определить площадь поверхности). ***Класс* 2. Шар** (радиус). ***Класс* 3. Полый шар** (внутренний радиус).
2, 10, 18	***Класс* 1. Плоская фигура** (Определить периметр, Определить площадь). ***Класс* 2. Круг** (радиус). ***Класс* 3. Круглое кольцо** (внутренний радиус).
3,	***Класс* 1. Объёмная фигура** (Определить объём, Определить

№ вар.	Данные для разработки иерархии классов
11, 19	площадь поверхности). ***Класс* 2. Прямой круговой конус** (радиус основания, высота). ***Класс* 3. Усечённый прямой круговой конус** (радиус малого основания).
4, 12, 20	***Класс* 1. Плоская фигура** (Определить периметр, Определить площадь). ***Класс* 2. Трапеция** (верхнее основание, нижнее основание, высота). ***Класс* 3. Параллелограмм** (угол параллелограмма).
5, 13, 21	***Класс* 1. Объёмная фигура** (Определить объём, Определить площадь поверхности). ***Класс* 2. Прямой круговой цилиндр** (радиус основания, высота). ***Класс* 3. Полый цилиндр** (внутренний радиус).
6, 14, 22	***Класс* 1. Объёмная фигура** (Определить объём, Определить площадь поверхности). ***Класс* 2. Правильная пирамида** (площадь основания, высота, апофема). ***Класс* 3. Правильная усечённая пирамида** (площадь малого основания).
7, 15, 23	***Класс* 1. Объёмная фигура** (Определить объём, Определить площадь поверхности). ***Класс* 2. Шаровой сегмент** (высота, радиус шара, радиус основания). ***Класс* 3. Шаровой слой** (радиус малого основания).
8, 16, 24	***Класс* 1. Плоская фигура** (Определить периметр, Определить площадь). ***Класс* 2. Круговой сектор** (радиус, центральный угол). ***Класс* 3. Круговой сегмент** (высота).

2.4. Контрольные вопросы

1. Что понимают под полиморфизмом в объектно-ориентированном программировании?
2. Какие методы называют виртуальными?
3. Как в коде на C# осуществляется переопределение виртуальных методов?
4. Какие виртуальные методы содержит базовый класс **System.Object**?
5. Что понимают под абстрактным методом?
6. Для чего предназначены абстрактные классы?

7. Как объявляются абстрактные классы и методы в языке C#?
8. Какие классы называют запечатанными?

3. РАБОТА С ИНТЕРФЕЙСАМИ В ПРИЛОЖЕНИЯХ НА ЯЗЫКЕ C#

3.1. Цель и задачи работы

Цель работы – прибрести умение работать с интерфейсами при разработке приложений на языке C#.

Основные задачи:

• научиться описывать собственные интерфейсы на языке C#;

• освоить реализацию интерфейсов классами и структурами в программах на C#;

• научиться работать со стандартными интерфейсами .NET Framework (**IComparable**, **IEnumerable**, **IEnumerator**).

Работа рассчитана на 6 часов.

3.2. Основные теоретические сведения

3.2.1. Объявление и реализация интерфейсов

Объявление интерфейса в C#. Сравнение интерфейсов и абстрактных классов.

Одним из наиболее важных средств языка C# являются интерфейсы, которые позволяют определить, что именно должен делать класс (структура), но не как он должен это делать. Для этого интерфейс должен быть объявлен предком выбранного класса.

Благодаря поддержке интерфейсов в C# может быть в полной мере реализован главный принцип полиморфизма: один интерфейс – множество методов.

Под ***интерфейсом*** в языке C# понимают именованный набор сигнатур методов. То есть внутри интерфейса следует перечисление заголовков методов без их реализации.

Интерфейсы в языке C# объявляются с помощью ключевого слова **interface**. Синтаксис объявления интерфейса аналогичен синтаксису класса и имеет следующий вид:

```
    [атрибуты]
```

```
[модификаторы] interface Имя_интерфейса [: Предки]
{
    Возвр_тип Имя_метода1([список_параметров]);
    Возвр_тип Имя_метода2([список_параметров]);
    . . .
    Возвр_тип Имя_методаN([список_параметров]);
}
```

Для интерфейсов могут быть указаны модификаторы **new**, **public** (по умолчанию), **protected**, **internal** и **private**. Модификатор **new** применяется для вложенных интерфейсов.

По соглашению имена всех интерфейсов .NET снабжаются префиксом в виде заглавной буквы «I». При создании собственных специальных интерфейсов также рекомендуется следовать этому соглашению.

Предками интерфейса могут выступать другие интерфейсы.

Помимо методов, в интерфейсах также можно указывать свойства, индексаторы и события.

Модификатор доступа у элементов интерфейса не указывается, поскольку элементы интерфейса всегда являются открытыми. Кроме того, ни один из элементов интерфейса не может быть объявлен с модификаторами **virtual** и **static**.

С точки зрения синтаксиса интерфейсы подобны абстрактным классам. Можно отметить следующие отличия интерфейса от абстрактного класса:

• если класс наследует абстрактный класс, то он может реализовать лишь некоторые методы родительского абстрактного класса, оставаясь абстрактным классом; класс, наследующий интерфейс, обязан полностью реализовать все методы интерфейса;

• абстрактный класс представляет собой начальный этап проектирования иерархии классов, которые в будущем получат конкретную реализацию; интерфейсы задают дополнительные свойства разных иерархий классов;

• элементы интерфейса по умолчанию имеют модификатор доступа **public** и не могут иметь модификаторов, заданных явным образом;

• интерфейс не может иметь полей и обычных методов – все элементы интерфейса должны быть абстрактными;

• интерфейс не может иметь перегруженных операций, что вызвано возможной несовместимостью с другими языками .NET (например, с Visual Basic .NET, который не поддерживает перегрузку операций).

Интерфейсы чаще всего используются для задания общего поведения объектов различных иерархий. Если определённый набор действий имеет смысл только для какой-то конкретной иерархии классов, реализующих эти действия разными способами, то уместнее задать этот набор в виде виртуальных методов абстрактного базового класса иерархии.

В настоящее время интерфейсы являются незаменимым инструментом в различных шаблонах проектирования, которые позволяют создавать большие, но при этом очень гибкие и расширяемые приложения.

Реализация интерфейса. Интерфейсные свойства и индексаторы.

Как только интерфейс будет определён, он может быть указан в качестве предка одного или нескольких классов:

```
class Car : IControl
{ ... }
```

Принято говорить, что класс, построенный на базе интерфейса, ***реализует*** этот ***интерфейс***.

Класс, реализующий интерфейс, воплощает методы, описанные в интерфейсы. Класс не может не реализовать какой-либо из методов интерфейса.

В отличие от наследования класс может реализовывать более одного интерфейса. В этом случае все реализуемые в классе интерфейсы указываются списком через запятую. Кроме того, у класса может быть указан базовый класс и в тоже время реализован один или более интерфейс. В таком случае имя базового класса должно быть указано перед списком интерфейсов:

```
class CivilBuilding : Building, IConstructable, IComparable
{ ... }
```

Интерфейсы могут наследовать друг друга точно также как классы. При этом интерфейс может иметь сколько угодно интер-

фейсов-предков. Базовые интерфейсы должны быть доступны не в меньшей степени, чем их потомки. Например, нельзя использовать интерфейс, объявленный с модификатором **private** или **internal**, в качестве базового для открытого (**public**) интерфейса.

В интерфейсе-потомке можно указывать элементы, переопределяющие унаследованные элементы с той же сигнатурой. В этом случае перед элементом указывается ключевое слово **new**.

При реализации элемента интерфейса имеется возможность указать его имя полностью вместе с именем самого интерфейса. В этом случае получается ***явная реализация*** элемента интерфейса. При этом модификаторы доступа не указываются. К таким элементам можно обращаться в программе только через объект типа интерфейса. Простой пример явной реализации элемента интерфейса представлен в листинге 3.1.

Листинг 3.1. Пример явной реализации метода интерфейса

```
// Интерфейс
interface IMyInterface
{
    double MyMethodA(double x);
    double MyMethodB(double x);
}

// Класс, реализующий интерфейс
class MyClass : IMyInterface
{
    // Обычная реализация метода
    double MyMethodA(double x)
    {
        return (1 / x);
    }

    // Явная реализация метода
    double IMyInterface.MyMethodB(double x)
    {
        return (1 / (x * x));
    }
}

class Program
{
    double IMyInterface.MyMethodB(double x)
    {
```

```
            return (1 / (x * x));
        }
    }
```

Явная реализация элементов интерфейса может использоваться по двум причинам:

• При явной реализации элемент интерфейса не становится открытым элементом класса. Это позволяет закрыть в классе элементы интерфейса, не требуемые конечному пользователю.

• В одном классе могут быть реализованы два интерфейса с методами, имеющими одинаковые имена и сигнатуры. Неоднозначность в этом случае устраняется благодаря указанию в именах этих методов их соответствующих интерфейсов.

Аналогично методам, свойства в интерфейсе указываются без тела. Общая форма объявления интерфейсного свойства выглядит следующим образом:

```
Тип_возврата Имя_свойства { get; set; }
```

В определении интерфейсного свойства, доступного только для чтения или только для записи, должен присутствовать единственный аксессор: **get** или **set** соответственно.

Несмотря на то, что объявление интерфейсного свойства очень похоже на объявление автоматически реализуемого свойства в классе, между ними существует различие. При объявлении в интерфейсе свойство не становится автоматически реализуемым. В этом случае указывается только имя и тип свойства, а его реализация предоставляется классу. Кроме того, при объявлении свойства в интерфейсе не разрешается указывать модификаторы доступа для аксессоров.

В интерфейсе также можно указать индексаторы. Ниже приведена общая форма интерфейсного индексатора:

```
Тип_возврата this[int индекс] { get; set; }
```

В объявлении интерфейсных индексаторов, доступных только для чтения или только для записи, должен присутствовать единственный аксессор: **get** или **set** соответственно.

В среде Visual Studio добавление модулей интерфейсов в проект осуществляется аналогично модулям классов. Для этого выбираем в меню **Project** (**Проект**) команду **Add New Item** (**До-**

бавить новый элемент…). В открывшемся диалоговом окне (рис. 3.1) выбираем шаблон **Interface** (Интерфейс) и нажимаем кнопку **Add** (**Добавить**).

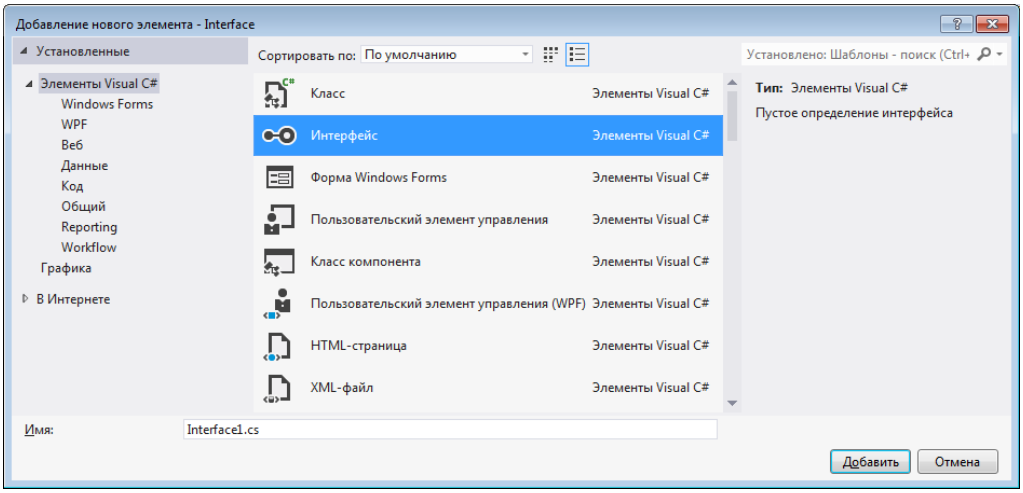


Рис. 3.1. Окно **Add New Item** (Добавление нового элемента) с выбранным шаблоном **Interface** (Visual Studio 2012)

Интерфейсы на диаграммах UML. Интерфейсы и отношение реализации.

В программном обеспечении важно строить системы с чётким разделением аспектов, чтобы по мере их развития изменения в одной части системы не затрагивали остальные части. Один из способов достижения этой цели – указание строго определённых соединений (интерфейсов) между частями, которые могут изменяться независимо друг от друга.

Под ***интерфейсом*** в UML понимают набор операций, используемых для определения сервиса класса. То есть интерфейс описывает контракт, который класс обязан исполнять.

Интерфейсы не могут содержать атрибутов. Они содержат только операции без указания особенностей их реализации.

На диаграмме классов интерфейс изображают как класс со стереотипом **«interface»** (рис. 3.2). Кроме того, интерфейс может быть изображён в виде маленького круга, рядом с которым записывается его имя. Такое обозначение интерфейса может использоваться на диаграммах компонентов и диаграммах развёртывания.

Предоставляемый интерфейс описывает сервисы, обеспечиваемые классом; изображается в виде маленького кружка, присоединённого к классу. Требуемый интерфейс описывает сервисы, требуемые одним классом от другого; изображается в форме маленького полукруга, соединённого с классом.

Многие языки программирования, включая C#, Java, Visual Basic, поддерживают концепцию интерфейса.

Реализацией в UML называют отношение между классификаторами, один из которых определяет некий контракт, а другой обязуется его выполнять. При этом один класс реализует поведение, заданное другим классом. Чаще всего реализация применяется для описания отношения между классом и интерфейсом.

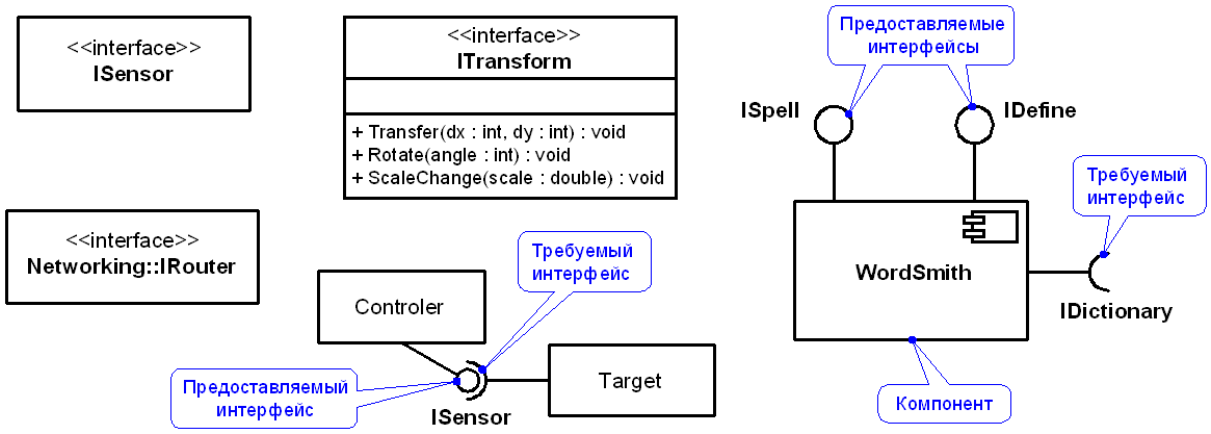


Рис. 3.2. Интерфейсы на диаграммах UML

По смыслу реализация представляет собой нечто среднее между зависимостью и обобщением. На диаграмме классов реализация изображается пунктирной линией с не закрашенной треугольной стрелкой, указывающей на класс, который определяет контракт (рис. 3.3, *а*). Кроме того, реализация может быть представлена в сокращенной форме с использованием нотации «леденец» (англ. *lollipop notation*) для предоставляемого интерфейса (рис. 3.3, *б*).

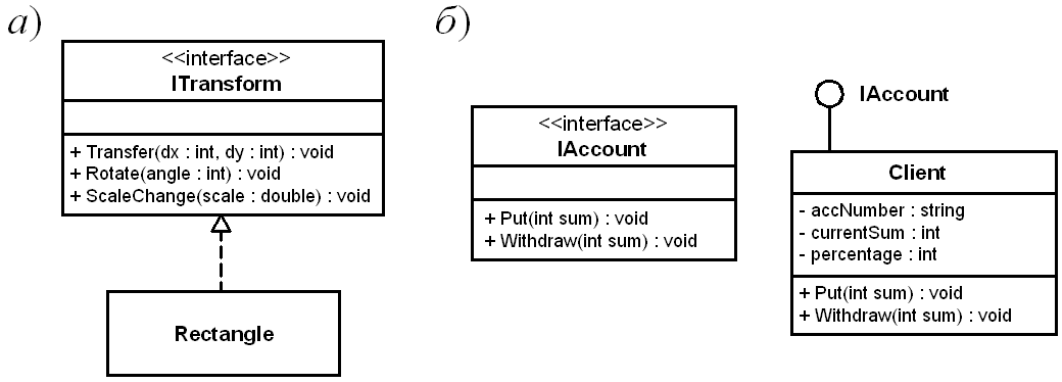


Рис. 3.3. Отношение реализации на диаграмме классов UML

❒ ***Пример* 3.1. *Создание и реализация интерфейсов на языке C#.***

Требуется разработать библиотеку классов, в которой заданы классы **Person** (человек), **Client** (клиент банка) и **Firm** (фирма). Класс **Client** является производным от класса **Person**.

Класс **Client** должен реализовывать интерфейс **IAccount** (счёт в банке), в котором должны быть определены следующие элементы:

- **string AccNumber** – свойство, возвращающее номер счёта;
- **int CurrentSum** – свойство, возвращающее текущую сумму;
- **int Percentage** – свойство, возвращающее процент начислений;
- **void Put(int sum)** – метод, обеспечивающий пополнение счёта на сумму **sum**;
- **void Withdraw(int sum)** – метод, позволяющий снять со счёта сумму **sum**.

Класс **Firm** должен реализовывать интерфейсы **ICommercial** (торговая организация) и **IAccount**. В интерфейсе **ICommercial** должны быть определены следующие элементы:

- **int Funds** – свойство, возвращающее размер денежного капитала фирмы;
- **void Buy(int sum)** – метод, позволяющий купить товары на сумму **sum**;
- **void Sell(int sum)** – метод, обеспечивающий продажу товаров на сумму **sum**.

Исходный код интерфейса **IAccount** приведён в листинге 3.2, а интерфейса **ICommercial** – в листинге 3.3.

Листинг 3.2. Исходный код интерфейса **IAccount**

```
 9      /// <summary>
10      /// Представляет денежные счета
11      /// </summary>
12      public interface IAccount
13      {
14          /// <summary>
15          /// Возвращает номер счёта
16          /// </summary>
17          string AccNumber { get; }
18
19          /// <summary>
20          /// Возвращает текущую сумму на счету
21          /// </summary>
22          int CurrentSum { get; }
23
24          /// <summary>
25          /// Возвращает процент начислений
26          /// </summary>
27          int Percentage { get; }
28
29          /// <summary>
30          /// Положить деньги на счёт
31          /// </summary>
32          /// <param name="sum">Сумма, руб</param>
33          void Put(int sum);
34
35          /// <summary>
36          /// Снять деньги со счёта
37          /// </summary>
38          /// <param name="sum">Сумма, руб</param>
39          void Withdraw(int sum);
40      }
```

Листинг 3.3. Исходный код интерфейса **ICommercial**

```
 9      /// <summary>
10      /// Представляет торговые организации
11      /// </summary>
12      interface ICommercial
13      {
14          /// <summary>
15          /// Возвращает денежный капитал, руб.
16          /// </summary>
17          int Funds { get; }
18
19          /// <summary>
20          /// Купить
21          /// </summary>
22          /// <param name="sum">Сумма покупки, руб</param>
23          void Buy(int sum);
24
25          /// <summary>
26          /// Продать
27          /// </summary>
28          /// <param name="sum">Сумма продажи, руб</param>
29          void Sell(int sum);
30      }
```

Исходные коды класса **Person**, а также производного от него класса **Client**, реализующего интерфейс **IAccount**, представлены в листингах 3.4, 3.5 и 3.6.

Листинг 3.4. Исходный код класса **Person**

```
    /// <summary>
    /// Представляет людей
    /// </summary>
    public class Person
    {
        /// <summary>
        /// Имя
        /// </summary>
        public string FirstName { get; set; }

        /// <summary>
        /// Фамилия
        /// </summary>
        public string LastName { get; set; }

        /// <summary>
        /// Пол
        /// </summary>
        public string Gender { get; set; }

        /// <summary>
        /// Дата рождения
        /// </summary>
        public DateTime BirthDate { get; set; }

        /// <summary>
        /// Параметрический конструктор
        /// </summary>
        /// <param name="firstName">Имя</param>
        /// <param name="lastName">Фамилия</param>
        /// <param name="gender">Пол: 'м', 'ж'</param>
        /// <param name="birthDate">Дата рождения</param>
        public Person(string firstName, string lastName, string gender,
                      DateTime birthDate)
        {
            FirstName = firstName;
            LastName = lastName;
            Gender = gender;
            BirthDate = birthDate;
        }

        /// <summary>
        /// Возвращает строку с данными о человеке
        /// </summary>
        /// <returns>Данные о человеке</returns>
        public override string ToString()
        {
            return (string.Format("* Имя: {0}\n" +
                                  "* Фамилия: {1}\n" +
                                  "* Пол: {2}\n" +
                                  "* Дата рождения: {3}\n",
                                  FirstName, LastName, Gender, BirthDate));
        }
    }
```

Листинг 3.5. Исходный код класса **Client** (часть 1)

```
    /// <summary>
    /// Представляет клиентов банка
    /// </summary>
    public class Client : Person, IAccount
    {
        string accNumber; // Номер счёта
        int currentSum;   // Текущая сумма, руб
        int percentage;   // Процент начислений, руб

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="fName">Имя</param>
        /// <param name="lName">Фамилия</param>
        /// <param name="gender">Пол</param>
        /// <param name="birthDate">Дата рождения</param>
        /// <param name="accNumber">Номер счёта</param>
        /// <param name="currentSum">Текущая сумма, руб</param>
        /// <param name="percentage">Процент начислений, руб</param>
        public Client(string fName, string lName, string gender, DateTime birthDate,
                string accNumber, int currentSum, int percentage)
            : base(fName, lName, gender, birthDate)
        {
            this.accNumber = accNumber;
            this.currentSum = currentSum;
            this.percentage = percentage;
        }

        /// <summary>
        /// Возвращает номер счёта (реализация свойства IAccount)
        /// </summary>
        public string AccNumber
        {
            get { return accNumber; }
        }

        /// <summary>
        /// Возвращает текущую сумму на счету (реализация свойства IAccount)
        /// </summary>
        public int CurrentSum
        {
            get { return currentSum; }
        }

        /// <summary>
        /// Возвращает процент начислений (реализация свойства IAccount)
        /// </summary>
        public int Percentage
        {
            get { return percentage; }
        }
```

Листинг 3.6. Исходный код класса **Client** (часть 2)

```
61          /// <summary>
62          /// Положить деньги на счёт (реализация метода IAccount)
63          /// </summary>
64          /// <param name="sum">Сумма, руб</param>
65          public void Put(int sum)
66          {
67              currentSum += sum;
68          }
69
70          /// <summary>
71          /// Снять деньги со счёта (реализация метода IAccount)
72          /// </summary>
73          /// <param name="sum">Сумма, руб</param>
74          public void Withdraw(int sum)
75          {
76              if (sum <= currentSum) currentSum -= sum;
77          }
78
79          public override string ToString()
80          {
81              return ("Данные о клиенте:\n" + base.ToString() +
82                  string.Format("* Номер счёта: {0}\n" +
83                                "* Текущая сумма: {1} руб.\n" +
84                                "* Процент начислений: {2} руб.\n",
85                                accNumber, currentSum, percentage));
86          }
87      }
```

Исходный код класса **Firm**, реализующего интерфейсы **ICommercial** и **IAccount**, представлен в листингах 3.7 и 3.8.

Листинг 3.7. Исходный код класса **Firm** (часть 1)

```
    /// <summary>
    /// Представляет фирмы (реализует интерфейсы ICommercial и IAccount)
    /// </summary>
    class Firm : ICommercial, IAccount
    {
        string name;      // Название
        Person director;  // Директор
        int funds;        // Денежный капитал, руб.
        string accNumber; // Номер счёта
        int currentSum;   // Сумма на счету, руб.
        int percentage;   // Процент начислений, руб.

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="name">Название</param>
        /// <param name="director">Директор</param>
        /// <param name="funds">Денежный капитал, руб.</param>
        /// <param name="accNumber">Номер счёта</param>
        /// <param name="currentSum">Сумма на счету, руб.</param>
        /// <param name="percentage">Процент начислений, руб.</param>
        public Firm(string name, Person director, int funds, string accNumber,
            int currentSum, int percentage)
        {
            this.name = name;
            this.director = director;
            this.funds = funds;
            this.accNumber = accNumber;
            this.currentSum = currentSum;
            this.percentage = percentage;
        }

        /// <summary>
        /// Возвращает капитал фирмы, руб.
        /// </summary>
        public int Funds
        {
            get { return funds; }
        }

        /// <summary>
        /// Купить
        /// </summary>
        /// <param name="sum">Сумма покупки, руб</param>
        public void Buy(int sum)
        {
            if (sum <= funds) funds -= sum;
        }

        /// <summary>
        /// Продать
        /// </summary>
        /// <param name="sum">Сумма продажи, руб</param>
        public void Sell(int sum)
        {
            funds += sum;
        }
```

Листинг 3.8. Исходный код класса **Firm** (часть 2)

```
        /// <summary>
        /// Возвращает номер счёта (реализация свойства IAccount)
        /// </summary>
        public string AccNumber
        {
            get { return accNumber; }
        }

        /// <summary>
        /// Возвращает текущую сумму на счету (реализация свойства IAccount)
        /// </summary>
        public int CurrentSum
        {
            get { return currentSum; }
        }

        /// <summary>
        /// Возвращает процент начислений (реализация свойства IAccount)
        /// </summary>
        public int Percentage
        {
            get { return percentage; }
        }

        /// <summary>
        /// Положить деньги на счёт (реализация метода IAccount)
        /// </summary>
        /// <param name="sum">Сумма, руб</param>
        public void Put(int sum)
        {
            if (sum <= funds)
            {
                funds -= sum;
                currentSum += sum;
            }
        }

        /// <summary>
        /// Снять деньги со счёта (реализация метода IAccount)
        /// </summary>
        /// <param name="sum">Сумма, руб</param>
        public void Withdraw(int sum)
        {
            if (sum <= currentSum)
            {
                currentSum -= sum;
                funds += sum;
            }
        }

        /// <summary>
        /// Возвращает данные о фирме
        /// </summary>
        /// <returns>Строка с данными о фирме</returns>
        public override string ToString()
        {
            return ("Данные о фирме:\n" +
                string.Format("* Название: {0}\n" +
                              "* Директор: {1}\n" +
                              "* Капитал: {2} руб.\n" +
                              "* Номер счёта: {3}\n" +
                              "* Сумма на счету: {4} руб.\n",
                              name, director.LastName, funds, accNumber, currentSum));
        }
    }
```

Схема классов, созданная средствами Visual Studio для полученной библиотеки классов, показана на рис. 3.4.

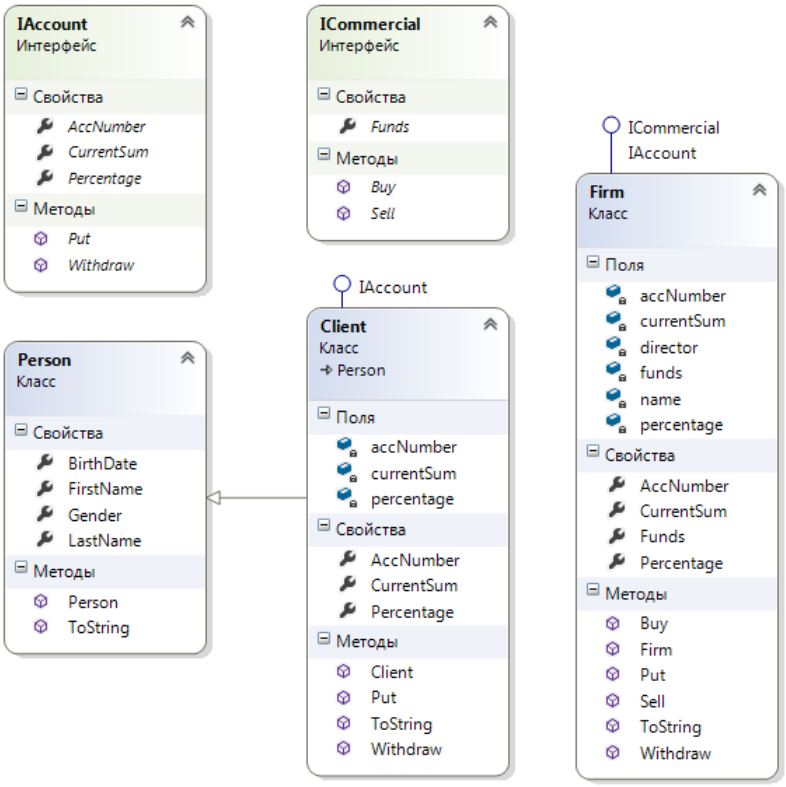


Рис. 3.4. Схема классов (Visual Studio 2012)

Для демонстрации работы классов, реализующих интерфейсы, добавим в решение проект консольного приложения.

Исходный код класса **Program** консольного приложения представлен в листинге 3.9.

Листинг 3.9. Исходный код класса **Program** консольного приложения

```
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Реализация интерфейсов";

        Client client1 = new Client("Дмитрий", "Егоров", "М",
            new DateTime(1988, 11, 25), "0011305", 30000, 2500);

        Client client2 = new Client("Анастасия", "Подольская", "Ж",
            new DateTime(1983, 7, 3), "1207358", 240000, 12000);

        Client client3 = new Client("Константин", "Марченко", "М",
            new DateTime(1976, 4, 11), "0003407", 135000, 4500);

        Client[] clients = new Client[] { client1, client2, client3 };

        Console.WriteLine("Клиенты банка:\n");
        foreach (Client c in clients)
            Console.WriteLine(client1.ToString());

        string s1 = "На счету {0} сумма: {1} руб.\n";

        client1.Put(50000);
        Console.WriteLine(s1, client1.AccNumber, client1.CurrentSum);

        client2.Withdraw(16500);
        Console.WriteLine(s1, client2.AccNumber, client2.CurrentSum);

        Firm firm1 = new Firm("ООО Бином", client3, 2500000,
            "1230567", 1550000, 152050);

        Console.WriteLine(firm1.ToString());

        string s2 = "Капитал фирмы {0} составляет {1} руб.\n";

        firm1.Buy(255000);
        Console.WriteLine(s2, firm1.Name, firm1.Funds);

        firm1.Sell(345000);
        Console.WriteLine(s2, firm1.Name, firm1.Funds);

        firm1.Put(300000);
        Console.WriteLine(s1, firm1.AccNumber, firm1.CurrentSum);

        client3.Put(45000);
        Console.WriteLine(s1, client3.AccNumber, client3.CurrentSum);

        Console.Read();
    }
```

Результат работы консольного приложения показан на рис. 3.5. ❒

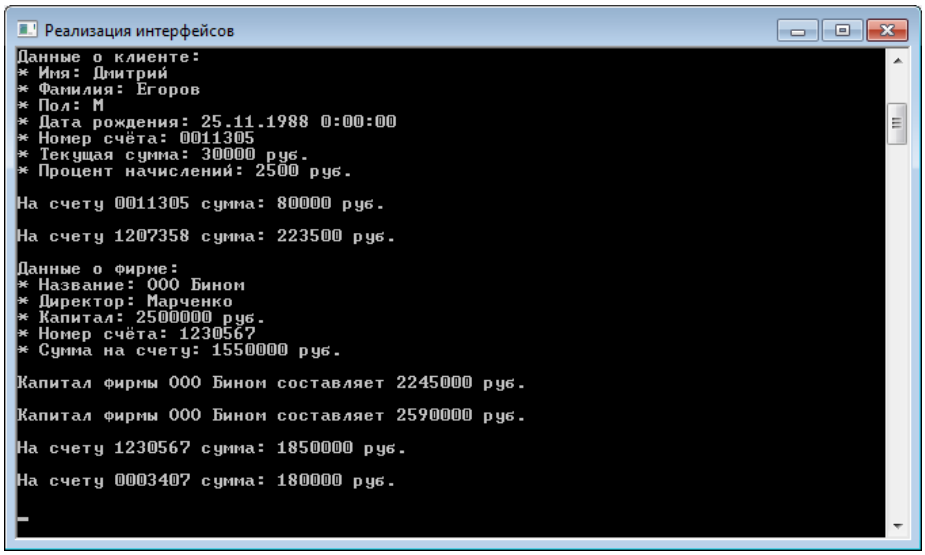


Рис. 3.5. Работа консольного приложения

3.2.2. Стандартные интерфейсы .NET Framework. Интерфейсы IComparable, IEnumerable и IEnumertor

Стандартные интерфейсы .NET Framework. Интерфейс IComparable.

В библиотеке классов .NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов. Например, интерфейс **IComparable** задаёт метод сравнения объектов по критерию больше или меньше, что позволяет выполнять их сортировку. Интерфейсы **IEnumerable** и **IEnumerator** дают возможность просматривать содержимое объекта с помощью цикла **foreach**. Реализация интерфейса **ICloneable** позволяет клонировать объекты.

В интерфейсе **IComparable** определен единственный метод **CompareTo(object obj)**, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра **obj**.

Метод должен возвращать:

• 0, если текущий объект и параметр равны;

• отрицательное число, если текущий объект меньше параметра;

• положительное число, если текущий объект больше параметра.

Интерфейсы IEnumerable и IEnumerator. Итераторы

Интерфейс **IEnumerable** определяет метод **GetEnumerator**, возвращающий объект типа **IEnumerator**, который можно использовать для просмотра элементов объекта.

Интерфейс **IEnumerator** задаёт три функциональных элемента:

• свойство **Current**, возвращающее текущий элемент объекта;

• метод **MoveNext**, передвигающий к следующему элементу объекта;

• метод **Reset**, устанавливающий в начало просмотра.

Для упрощения перебора в объекте используются средства, называемые итераторами.

Итератор представляет собой блок кода, задающий последовательность перебора элементов объекта.

Преимущество использования итераторов заключается в том, что для одного и того же класса можно задать различный порядок перебора элементов.

В теле итератора должен присутствовать оператор **yield**, имеющий следующий синтаксис:

```
return yield выражение;
```

При выполнении итератора автоматически создаётся контейнер, в который добавляется элемент при каждом выполнении оператора **yield**. Порядок выполнения оператора **yield** определяет порядок перечислимости элементов контейнера.

Операции is и as.

При работе с объектом часто требуется проверить, поддерживает ли объект определённый интерфейс. Проверка может быть выполнена с помощью бинарной операции **is.**

Операция **is** определяет, совместим ли тип объекта, находящегося слева от ключевого слова **is**, с типом, заданным справа. Результат операции равен **true**, если объект можно преобразовать к заданному типу, и **false** в противном случае.

Недостатком использования операции **is** является то, что преобразование фактически выполняется дважды: при проверке и при собственно преобразовании.

Операция **as** выполняет преобразование к заданному типу, а если это невозможно, то формирует результат **null**.

❒ ***Пример* 3.2. *Работа со стандартными интерфейсами .NET Framework.***

Требуется разработать проект библиотеки классов, содержащий следующие классы:

• **Car** – представляет автомобили; реализует интерфейс **IComparable**;

• **Garage** – представляет автомобильные гаражи; реализует интерфейс **INumerable**.

Сравнение экземпляров класса **Car** с помощью метода **CompareTo()** будем производить по значению свойства **Probeg** (пробег автомобиля, км).

В классе **Garage** должен быть реализован метод **GetEnumerator()**, возвращающий интерфейсный объект типа **IEnumerator**, который позволяет просматривать элементы массива **carLot** (состоит из экземпляров класса **Car**).

Дополнительно добавим в класс **Garage** метод **CarsIterator(bool reverse)**, который возвращает элементы массива в прямом или обратном порядке в зависимости от значения параметра **reverse**.

Исходный код класса **Car** представлен в листингах 3.8 и 3.9, а код класса **Garage** – в листинге 3.10.

Листинг 3.8. Исходный код класса **Car** (часть 1)

```
    /// <summary>
    /// Представляет автомобили (реализует интерфейс IComparable)
    /// </summary>
    public class Car : IComparable
    {
        /// <summary>
        /// Марка автомобиля
        /// </summary>
        public string Mark { get; set; }

        /// <summary>
        /// Регистрационный номер
        /// </summary>
        public string Number { get; set; }

        /// <summary>
        /// Пробег, км
        /// </summary>
        public int Probeg { get; set; }

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="mark">Марка</param>
        /// <param name="number">Номер</param>
        /// <param name="probeg">Пробег, км</param>
        public Car(string mark, string number, int probeg)
        {
            Mark = mark;
            Number = number;
            Probeg = probeg;
        }

        /// <summary>
        /// Возвращает строку с данными об автомобиле
        /// </summary>
        /// <returns>Данные об автобмобиле</returns>
        public override string ToString()
        {
            return string.Format("Данные об автомобиле:\n" +
                                 "- марка: {0}\n" +
                                 "- номер: {1}\n" +
                                 "- пробег: {2} км\n",
                                 Mark, Number, Probeg);
        }
```

Листинг 3.9. Исходный код класса **Car** (часть 2)

```
55          /// <summary>
56          /// Возвращает результат сравнения объектов в форме целого числа
57          /// (явная реализация IComparable)
58          /// </summary>
59          /// <param name="obj">Сравниваемый объект</param>
60          /// <returns>Результат сравнения</returns>
61          int IComparable.CompareTo(object obj)
62          {
63              const string s = "Сравниваемый объект не принадлежит классу Car";
64
65              // Присвоить объекту car объект obj, приведённый к типу Car,
66              // если такое преобразование невозможно, то присвоить null
67              Car car = obj as Car;
68
69              if (car != null)
70              {
71                  if (this.Probeg > car.Probeg) return 1;
72                  if (this.Probeg < car.Probeg) return -1;
73                  else return 0;
74              }
75              // Более короткий вариант кода (используется интерфейс IComparable
76              // типа данных int):
77              // if (car != null) return this.Probeg.CompareTo(car.Probeg);
78
79              // Сгенерировать исключение, если параметр obj не является
80              // экземпляром класса Car
81              else throw new ArgumentException(s);
82          }
83      }
```

Листинг 3.10. Исходный код класса **Garage**

```
using System.Collections;

namespace ClassLibrary
{
    /// <summary>
    /// Представляет гаражи (реализует интерфейс IEnumerable)
    /// </summary>
    public class Garage : IEnumerable
    {
        Car[] carArray; // Массив автомобилей

        /// <summary>
        /// Параметрический конструктор
        /// </summary>
        /// <param name="carArray">Массив автомобилей</param>
        public Garage(Car[] carArray)
        {
            this.carArray = carArray;
        }

        /// <summary>
        /// Возвращает IEnumerator (реализация IEnumerable)
        /// </summary>
        /// <returns>Интерфейсный объект типа IEnumerator</returns>
        public IEnumerator GetEnumerator()
        {
            return carArray.GetEnumerator();
        }

        /// <summary>
        /// Итератор, возвращающий элементы массива carArray
        /// в прямом или обратном порядке
        /// </summary>
        /// <param name="reverse">Обратный порядок элементов</param>
        /// <returns>Интерфейсный объект типа IEnumerable</returns>
        public IEnumerable CarsIterator(bool reverse)
        {
            // Возвратить элементы в обратном порядке
            if (reverse)
            {
                for (int i = carArray.Length; i != 0; i--)
                {
                    yield return carArray[i - 1];
                }
            }
            else
            {
                foreach (Car c in carArray)
                {
                    yield return c;
                }
            }
        }
    }
}
```

Схема классов для полученной библиотеки классов показана на рис. 3.6.

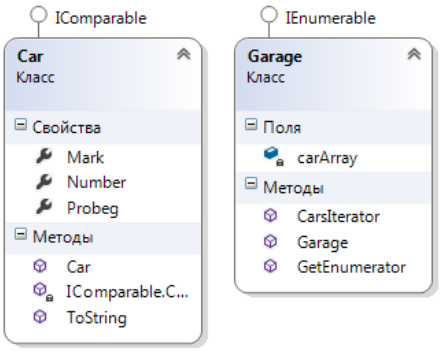


Рис. 3.6. Схема классов для библиотеки классов

Для демонстрации работы с классами, реализующими стандартные интерфейсы, добавим в решение проект консольного приложения.

Исходный код класса **Program** консольного приложения показан в листинге 3.11.

Листинг 3.11. Исходный код класса **Program**

```
11  class Program
12  {
13      static void Main(string[] args)
14      {
15          Console.Title = "Работа со стандартными интерфейсами";
16          Car car1 = new Car("ВАЗ", "11111", 15000);
17          Car car2 = new Car("УАЗ", "22222", 25000);
18          Car car3 = new Car("ГАЗ", "33333", 20000);
19
20          Car[] carLot = new Car[] { car1, car2, car3 };
21
22          // Сортировать массив carLot, используя IComparable
23          Array.Sort(carLot);
24
25          Console.WriteLine("Автомобили, отсортированные по пробегу:\n");
26          // Отобразить содержимое отсортированного массива
27          foreach (Car car in carLot)
28              Console.WriteLine(car.ToString());
29
30          Garage garage = new Garage(carLot);
31
32          Console.WriteLine("Все автомобили в гараже:\n");
33          // Получить элементы, используя IEnumerator
34          foreach (Car car in garage)
35              Console.WriteLine(car.ToString());
36
37          IEnumerator ien = garage.GetEnumerator();
38
39          Console.WriteLine("Все автомобили в обратном порядке:\n");
40          // Получить элементы (в обр. порядке), используя итератор
41          foreach (Car car in garage.CarsIterator(true))
42          {
43              Console.WriteLine(car.ToString());
44          }
45
46          Console.Read();
47      }
48  }
```

Результат работы консольного приложения показан на рис. 3.7. ❒

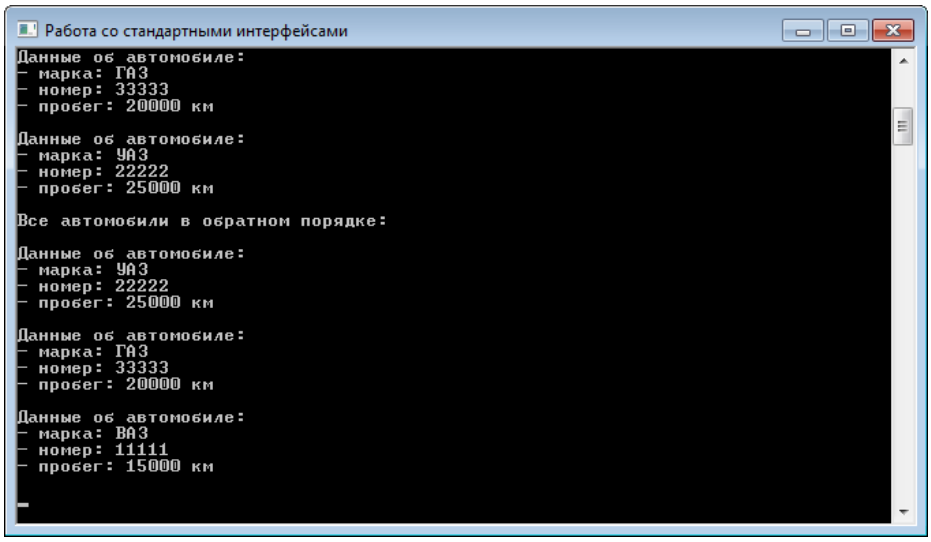


Рис. 3.7. Работа консольного приложения

3.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

5. Изучить методические указания к лабораторной работе.
6. Реализовать заданный интерфейс в указанных классах. Создать библиотеку с указанными классами и интерфейсами (табл. 3.1). Реализовать интерфейсы в подходящих для них классах. Разработать проект консольного приложения для работы с полученными классами.
7. Создать заданные классы (табл. 3.2), реализующие стандартные интерфейсы **IComparable** и **IEnumerable**. Разработать проект консольного приложения для работы с экземплярами созданных классов.
8. Оформить и защитить отчет по лабораторной работе.

Таблица 3.1

Варианты заданий на реализацию интерфейсов

№ вар.	Классы	Интерфейсы
1, 9, 17	***Здание*** (адрес, площадь, число этажей, дата постройки, получить данные). ***Автомобиль*** (регистр номер, марка, дата выпуска, пробег, получить данные).	***Ремонтируемый*** (износ, определить стоимость ремонта, ремонтировать). ***Заправляемый топливом*** (текущий уровень топлива, объём топливного бака, заправить топливом).
2, 10, 18	***Книга*** (название, автор, издательство, год выпуска, число страниц, получить данные). ***Электрочайник*** (модель, цвет, объём, мощность, получить данные).	***Товар*** (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки). ***Потребляющий электроэнергию*** (напряжение питания, сила тока, подключить к сети, отключить от сети, определить затраты энергии).
3, 11, 19	***Квартира*** (адрес дома, номер, этаж, площадь, число комнат, получить данные). ***Локомотив*** (модель, год выпуска, мощность, максимальная скорость, получить данные).	***Ремонтируемый*** (износ, определить стоимость ремонта, ремонтировать). ***Перемещаемый в пространстве*** (текущие координаты, заданные координаты, двигаться к цели).
4, 12, 20	***Легковой автомобиль*** (марка, тип кузова, цвет, расход топлива, получить данные). ***Обувь*** (название, сезон, материал, цвет, размер, получить данные).	***Товар*** (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки). ***Очищаемый*** (степень загрязнения, очистить, определить время очистки).
5, 13, 21	***Посылка*** (код, отправитель, адрес получателя, вес, получить данные). ***Станок*** (тип, модель, мощность привода, точность, получить данные).	***Перемещаемый в пространстве*** (текущие координаты, заданные координаты, перемещать к цели). ***Потребляющий электроэнергию*** (напряжение питания, сила тока, подключить к сети, отключить от сети, определить затраты энергии).
6, 14,	***Дорога*** (длина, ширина, мате-	***Ремонтируемый*** (износ, опре-

№ вар.	Классы	Интерфейсы
22	риал полотна, получить данные). ***Обувь*** (название, сезон, материал, цвет, размер, получить данные).	делить стоимость ремонта, ремонтировать). ***Товар*** (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки).
7, 15, 23	***Принтер*** (модель, скорость печати, объём картриджа, максимальное число листов в лотке, получить данные). ***Квартира*** (адрес дома, номер, этаж, площадь, число комнат, получить данные).	***Товар*** (цена, скидка, производитель, дата выпуска, определить цену с учётом скидки). ***Очищаемый*** (степень загрязнения, очистить, определить время очистки).
8, 16, 24	***Грузовой контейнер*** (код, перевозчик, получатель груза, вес груза, получить данные). ***Пассажирский самолёт*** (модель, авиакомпания, макс. число пассажиров, крейсерская скорость).	***Перемещаемый в пространстве*** (текущие координаты, заданные координаты, перемещать к цели). ***Заправляемый топливом*** (текущий уровень топлива, объём топливного бака, заправить топливом).

Таблица 3.2

Варианты заданий на создание классов, реализующих стандартные интерфейсы **IComparable** и **IEnumerable**

№ вар.	IComparable	IEnumerable
1, 13	***Квартира.*** Номер, этаж, площадь, число комнат.	***Многоквартирный дом.*** Улица, номер, квартиры.
2, 14	***Заказ на перевозку груза.*** Номер, дата, адрес доставки, вес груза, стоимость перевозки.	***Транспортная компания.*** Название, телефон, заказы.
3, 15	***Спортсмен.*** ФИО, вид спорта, дата рождения, пол, рост, вес.	***Спортивная команда.*** Название, тренер, спортсмены.
4, 16	***Ноутбук.*** Модель, процессор, размер экрана, вес, цена.	***Компьютерный магазин.*** Название, адрес, ноутбуки.
5, 17	***Студент.*** ФИО, группа, пол, дата рождения, средний бал.	***Студенческая группа.*** Название, куратор, студенты.

6, 18	***Книга.*** Название, автор, цена, число страниц, год издания.	***Книжный магазин.*** Название, адрес, книги.
7, 19	***Сотрудник.*** ФИО, пол, дата рождения, должность, зарплата.	***Организация.*** Название, адрес, сотрудники.
8, 20	***Учебная дисциплина.*** Название, ФИО преп-ля, форма контроля, семестр, число часов.	***Учебный план.*** Направление подготовки, профиль, дисциплины.
9, 21	***Кредит.*** Получатель, сумма, процент, дата получения, срок.	***Банк.*** Название, адрес центрального офиса, кредиты.
10, 22	***Предмет обуви.*** Наименование, производитель, число пар, размер, цена.	***Обувной магазин.*** Название, адрес, обувь.
11, 23	***Телевизор.*** Фирма, модель, размер экрана, вес, цена.	***Магазин бытовой электроники.*** Название, адрес, телевизоры.
12, 24	***Билет на междугородный транспорт.*** Рейс, пункт назнач., время отправления, длительность, номер места.	***Автовокзал.*** Город, число касс, билеты.

3.4. Контрольные вопросы

9. Какой тип называют интерфейсом в языке C#?
10. Для чего применяют интерфейсы?
11. Как объявляется интерфейс в программе на C#?
12. Чем интерфейс отличается от абстрактного класса?
13. Что такое реализация интерфейса?
14. Как интерфейсы обозначают на диаграммах UML?
15. Что понимают под отношением реализации в UML и как это отношение показывают на диаграммах классов?
16. Для чего предназначен стандартный интерфейс **IComparable**?
17. Каково назначение интерфейса **INumerable**?

4. ОСНОВЫ РАБОТЫ С ШАБЛОНАМИ GRASP В ПРИЛОЖЕНИЯХ НА ЯЗЫКЕ C#

4.1. Цель и задачи работы

Цель работы – прибрести умение разрабатывать приложения на языке C# с использованием таких шаблонов проектирования GRASP, как Information Expert, Creator, Loosely Coupling и High Cohesion.

Основные задачи:

• научиться применять основные шаблоны GRASP (Information Expert, Creator, Loosely Coupling, High Cohesion) для объектно-ориентированного проектирования;

• научиться создавать проекты моделирования с помощью MS Visual Studio, а также выполнять преобразование модели в программный код;

Работа рассчитана на 6 часов.

4.2. Основные теоретические сведения

4.2.1. Проектирование классов на основе обязанностей. Основные шаблоны GRASP

Проектирование классов на основе обязанностей.

В разрабатываемой программной системе могут быть определены десятки и сотни различных классов, на которые могут быть возложены сотни и тысячи обязанностей.

Во время объектно-ориентированного проектирования при определении принципов взаимодействия объектов необходимо распределить обязанности между классами. При правильном выполнении этой задачи система становится гораздо проще для понимания, поддержки и расширения. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

Проектирование на основе обязанностей (англ. *Responsibility-Driven Design, RDD*) – это общий подход к проектированию классов, в котором считается, что классы программной системы

имеют определённые обязанности и должны взаимодействовать с другими классам для их выполнения.

В общем случае выделяют два типа обязанностей: знание (*knowing*) и действие (*doing*).

Обязанности, относящиеся к знаниям объекта (вытекают из модели предметной области):

• наличие информации о закрытых инкапсулированных данных;

• наличие информации о связанных объектах;

• наличие информации о следствиях или вычисляемых величинах.

Обязанности, относящиеся к действиям объекта:

• выполнение некоторых действий самим объектом (например, создание экземпляра или выполнение вычислений);

• инициирование действий других объектов;

• управление действиями других объектов и их координирование.

Выявить и описать основные принципы распределения обязанностей, положенные в основу подхода RDD, позволяют ***общие шаблоны распределения обязанностей*** (англ. *General Responsibility Assignment Software Patterns, GRASP*).

К основным шаблонам GRASP относятся:

• **Information Expert** – информационный эксперт;

• **Creator** – создатель экземпляров класса;

• **Low Coupling** – слабая связанность;

• **High Cohesion** – сильное сцепление.

Шаблоны Information Expert и Creator.

Шаблон ***Information Expert*** является наиболее общим шаблоном GRASP и при распределении обязанностей используется гораздо чаще других шаблонов.

Проблема.

Каков наиболее общий принцип распределения обязанностей между классами при объектно-ориентированном проектировании?

Решение.

Назначить обязанность ***информационному эксперту*** – классу, который обладает достаточной информацией для выполнения этой обязанности.

Результаты.

• Поддержка инкапсуляции; для выполнения требуемых задач объекты используют собственные данные.

• Более простое понимание и поддержка системы классов, моделирующих заданную систему.

Создание объектов в объектно-ориентированной системе является одним из наиболее стандартных видов деятельности. Поэтому при распределении обязанностей, связанных с созданием объектов, следует руководствоваться шаблоном ***Creator***.

Проблема.

Какой класс должен отвечать за создание нового экземпляра выбранного класса **A**?

Решение.

Назначить классу **B** обязанность создавать объекты другого класса **A**, если выполняется одно из следующих условий:

• класс **B** агрегирует или содержит объекты **A**;

• класс **B** активно использует объекты **A**;

• класс **B** обладает данными инициализации для объектов **A**.

Результаты.

• Использование шаблона не увеличивает число связей между классами, поскольку созданный класс, как правило, виден только для класса-создателя.

• Если процедура создания объекта достаточно сложная, то предпочтительно использовать такой шаблон, как Фабрика.

Шаблоны Low Coupling и High Cohesion.

Шаблон ***Слабая связанность*** позволяет снизить влияние изменений в одном классе системы на другие связанные с ним классы.

Степень связанности (англ. *coupling*) – это мера, определяющая, насколько жестко один элемент программной системы связан с другими элементами, либо каким количеством данных о других элементах он обладает.

Класс с низкой степенью связанности (слабо связанный) зависит от не большого числа других классов.

Класс с высокой степенью связанности (жестко связанный) зависит от множества других классов. Наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем:

• изменения в связанных классах приводят к изменениям в данном классе;

• затрудняется понимание каждого класса в отдельности;

• усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.

Проблема.

Как уменьшить влияние изменений, вносимых в данный класс, на другие классы?

Решение.

Распределить обязанности между классами таким образом, чтобы степень связанности системы оставалась низкой.

Результаты.

• Улучшаются возможности для повторного использования классов системы.

• Появляется возможность поручить разработку отдельных частей системы разным разработчикам.

• При злоупотреблении шаблоном система будет состоять из набора изолированных сложных классов, самостоятельно выполняющих все операции, и набора классов, хранящих данные.

Ещё одним важным шаблоном GRASP является шаблон ***Сильное сцепление***, который позволяет создавать простые для понимания классы, обладающие возможностью повторного использования

В терминах объектно-ориентированного проектирования ***функциональное сцепление*** (англ., *cohesion*) – это мера связанности или сфокусированности обязанностей класса (подсистемы).

Класс обладает высокой степенью сцепления (сильным сцеплением), если его обязанности тесно связанны между собой и он не выполняет непомерных объёмов работы.

Класс с низкой степенью сцепления (слабым сцеплением) выполняет много несвязанных между собой обязанностей. Такие классы приводят к возникновению следующих проблем:

• сложность понимания системы;
• сложность при повторном использовании;
• сложность поддержки;
• ненадёжность, постоянная подверженность изменениям.

Проблема.

Как обеспечить сфокусированность обязанностей классов, их управляемость и ясность для понимания?

Решение.

Обеспечить высокую степень сцепления при распределении обязанностей.

Результаты.

Классы с высокой степенью сцепления просты в поддержке и повторном использовании;

В некоторых случаях неоправданно использовать высокое сцепление для распределённых серверных объектов.

❒ ***Пример* 4.1. *Проектирование диаграммы классов.***

Требуется разработать диаграмму классов для предметной области «Железнодорожные перевозки».

В рассматриваемой предметной области выделены следующие классы с указанными атрибутами:

- **Груз** (код, наименование, вес).
- **Вагон** (код, модель, грузы, грузоподъёмность, вес).
- **Локомотив** (код, модель, максимальный вес поезда).
- **Поезд** (код, номер, вагоны, локомотив).

Между отмеченными классами требуется распределить следующие обязанности:

- Определить текущую координату.
- Добавить груз в вагон.
- Убрать груз из вагона.
- Прицепить вагон к поезду.
- Отцепить вагон от поезда.
- Определить суммарный вес груза.
- Определить общий вес поезда.

• Переместить поезд на заданное расстояние.

Для реализации шаблонов Information Expert и Creator распределим обязанности между классами следующим образом:

Вагон.

• Добавить груз в вагон.
• Убрать груз из вагона.
• Определить общий вес вагона с грузом.

Поезд.

• Определить текущую координату.
• Прицепить вагон к поезду.
• Отцепить вагон от поезда.
• Определить общий вес поезда.
• Переместить поезд на заданное расстояние.

Диаграмма классов, созданная с учётом распределения обязанностей, показана на рис. 4.1. ❒

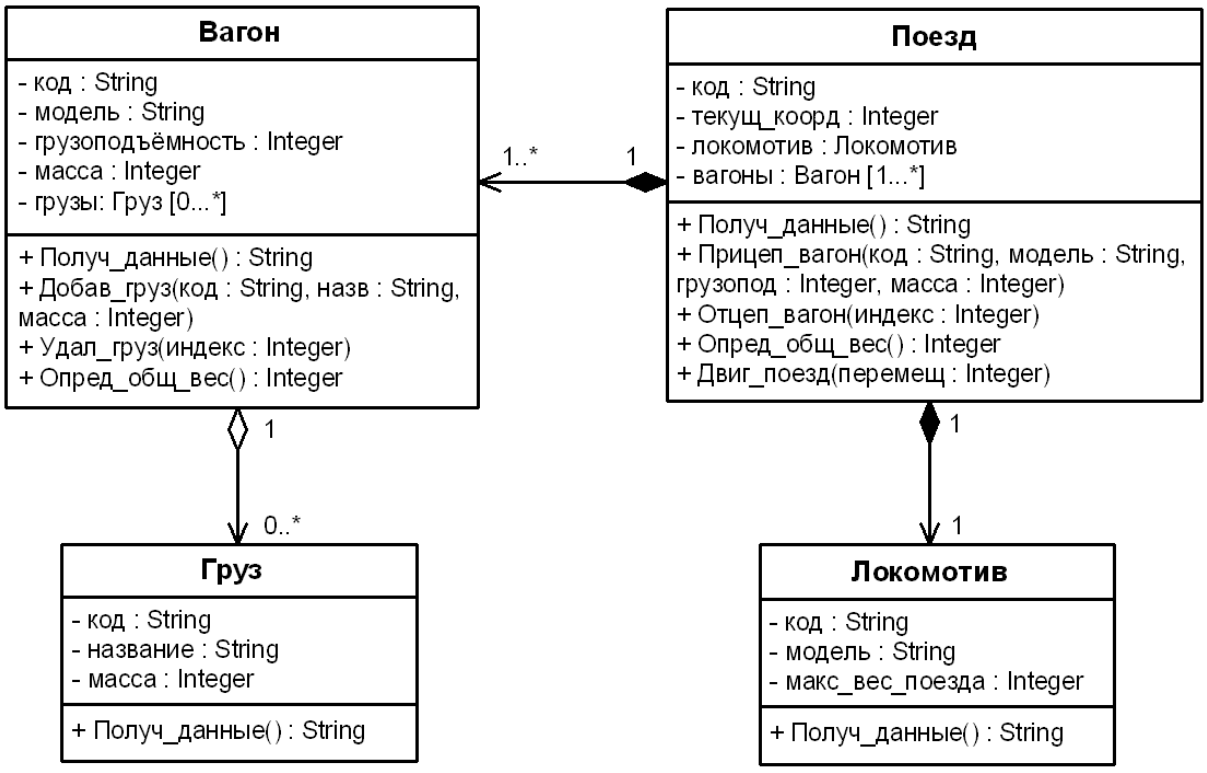


Рис. 4.1. Диаграмма классов

4.2.2. Работа с UML-моделями в Visual Studio

Создание проектов моделирования в Visual Studio.

С помощью MS Visual Studio можно разрабатывать модели на языке UML. Для создания проектов моделирования требуется Visual Studio Ultimate. В Visual Studio Professional можно только просматривать проекты моделирования.

Для создания схемы и добавления её в проект требуется в меню **Архитектура** (**Architecture**) выбрать пункт **Создать схему** (**New Diagram**). В открывшемся диалоговом окне **Добавление новой схемы** (рис. 4.2) производится выбор требуемого типа схемы моделирования и вводится имя новой схемы.

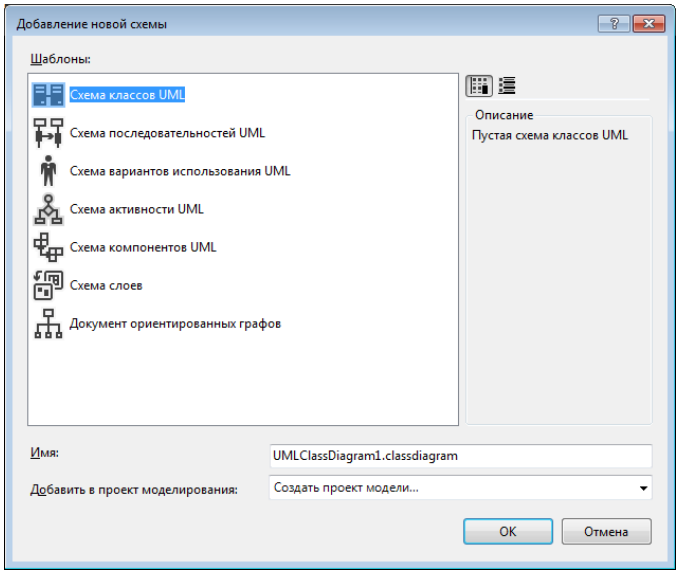


Рис. 4.2. Окно **Добавление новой схемы**

После нажатия кнопки **OK** откроется окно **Создание проекта модели** (рис. 4.3), в котором следует ввести имя и расположение нового проекта, а затем нажать кнопку **Создать**.

Главное окно Visual Studio с открытым проектом моделирования показано на рис. 4.4.

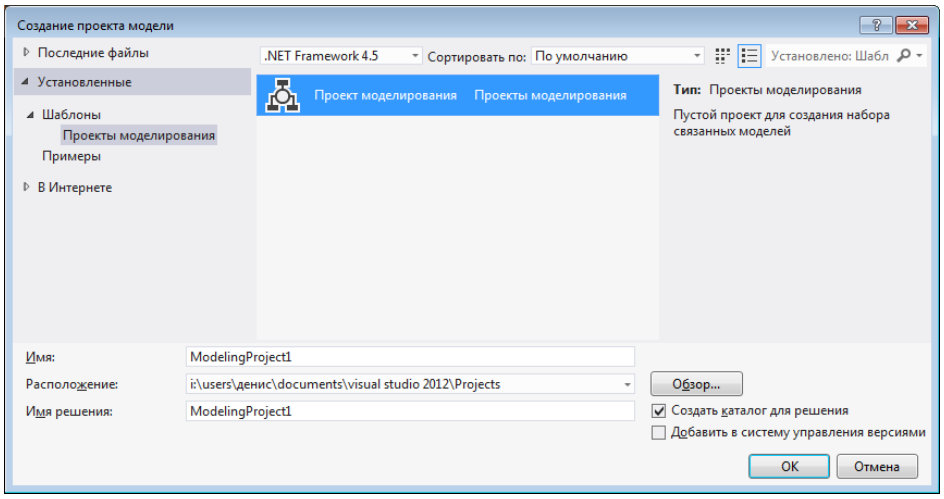


Рис. 4.3. Окно **Создание проекта модели**

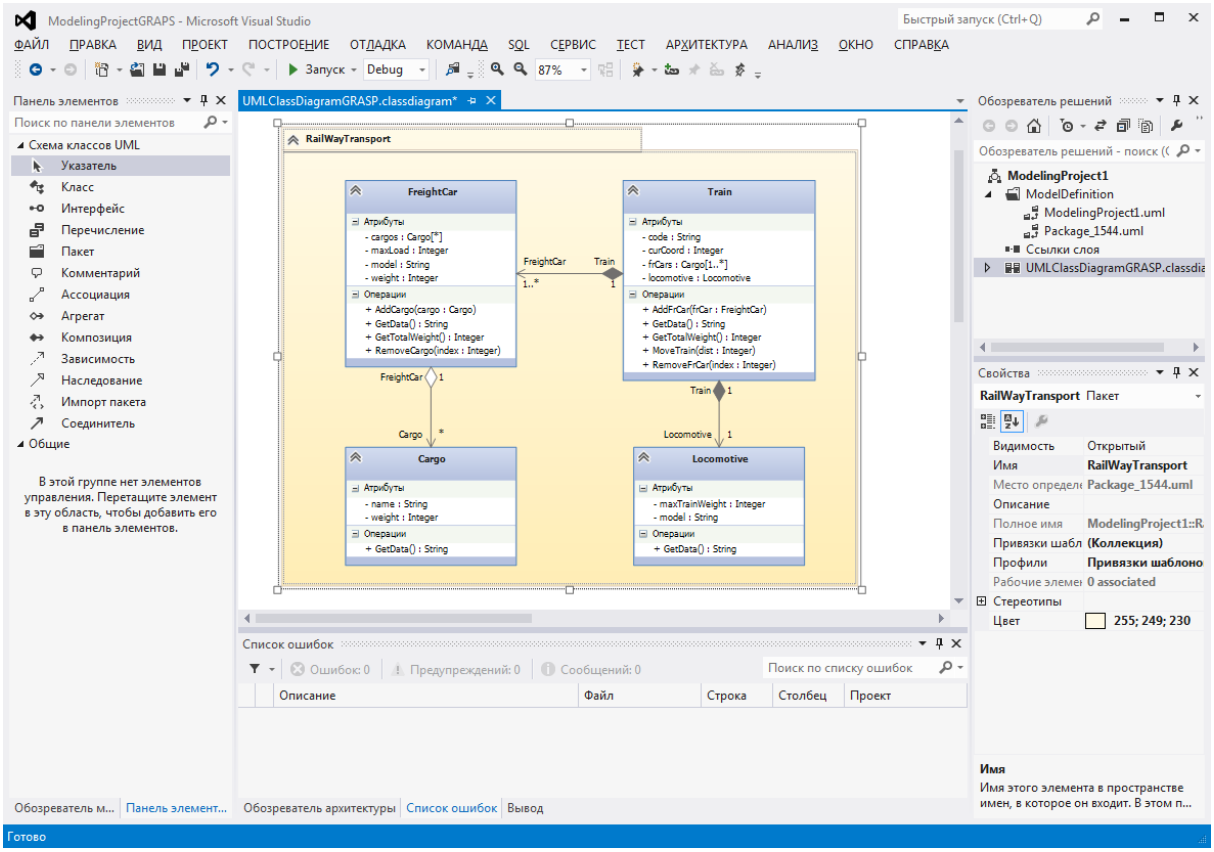


Рис. 4.4. Главное окно проекта моделирования

Разработка UML-схем классов.

UML-схема классов описывает структуры объектов, используемые для внутренней организации приложения и для взаимодействия с пользователями. Её классы и отношения могут реали-

зовываться различными способами, например в таблицах реляционной базы данных или узлах XML-документа.

Чтобы создать тип, требуется на панели элементов (рис. 4.5) выбрать Класс, Интерфейс или Перечисление и затем щелкнуть пустую область схемы.

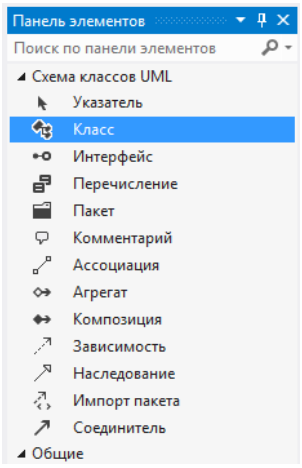


Рис. 4.5. Панель элементов

Для добавления атрибута (операции) требуется щелкнуть правой кнопкой мыши на класс или интерфейс, выбрать команду **Добавить** и щелкнуть по пункту **Атрибут** (**Операция**).

После добавления атрибута или операции их сигнатуру можно настроить с помощью окна **Свойства** (рис. 4.6).

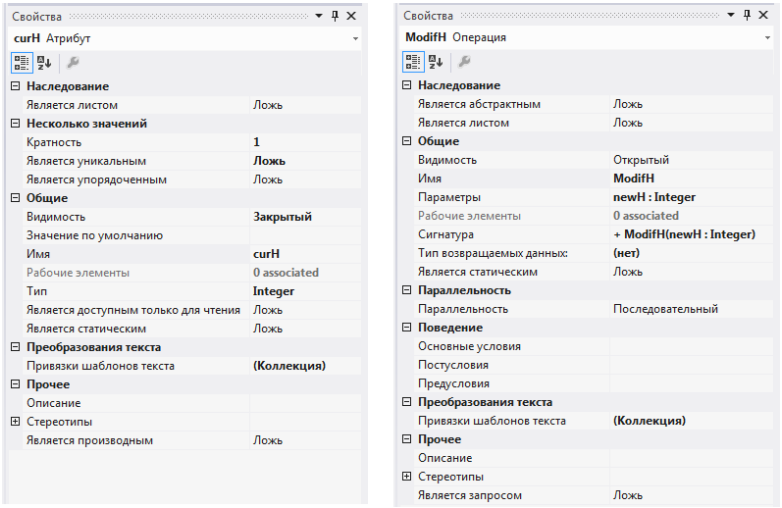


Рис. 4.6. Окно **Свойства** для выбранных атрибута и операции

Для определения параметров операции требуется в окне **Свойства** выбрать пункт **Параметры** и нажать кнопку ... В результате откроется окно **Редактор параметров операции** (4.7).

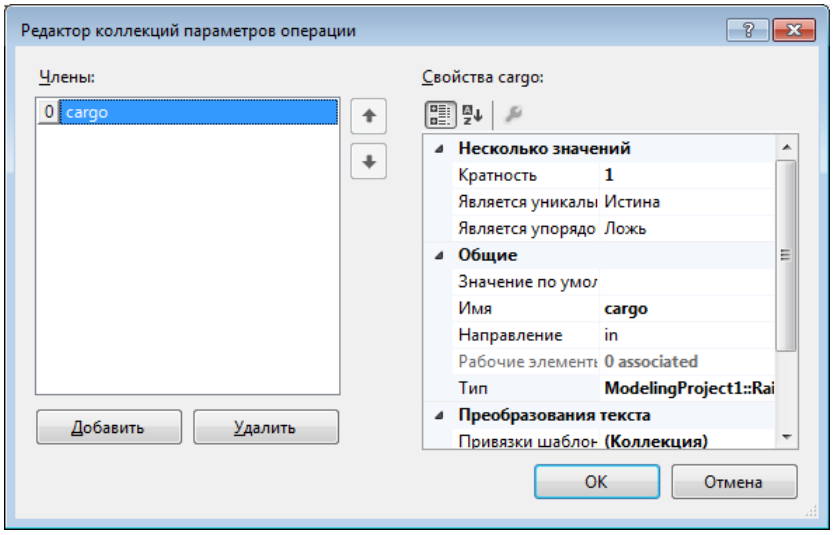


Рис. 4.7. Окно **Редактор параметров операции**

Пример схемы классов, отображаемой в окне **Обозревателя моделей UML**, показан на рис. 4.8.

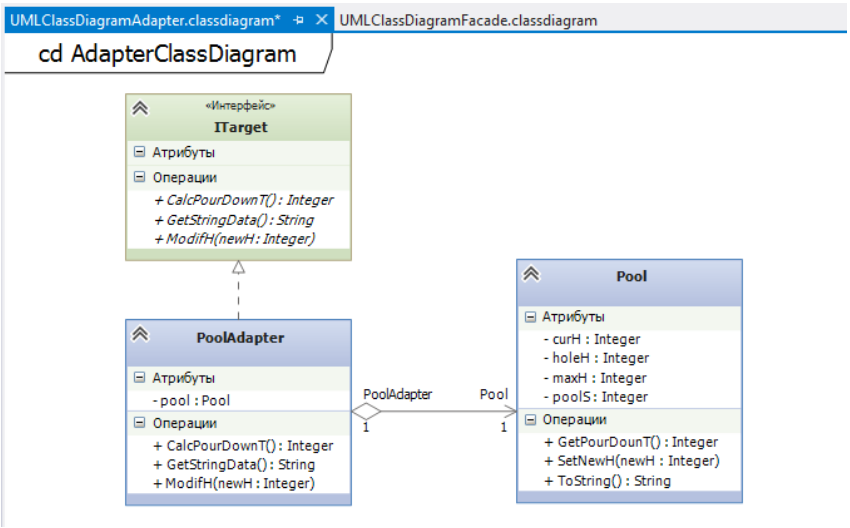


Рис. 4.8. Окно **Обозреватель модели UML**

Создание кода из UML-схем классов.

В Visual Studio Ultimate 2012 можно создать код из схем классов UML с помощью команды **Создать код**. По умолчанию

команда создает тип языка C# для каждого выбранного типа UML.

Для создания кода по умолчанию из схемы классов UML используется следующая процедура. На схеме классов или выделяются элементы, из которых требуется создать код. Открывается контекстное меню выделенного элемента и выбирается команда **Создать код**. При первом использовании команды **Создать код** в определенной модели отобразится диалоговое окно **Привязки шаблонов текста** (рис. 4.9), позволяющее изменять параметры создания кода модели.

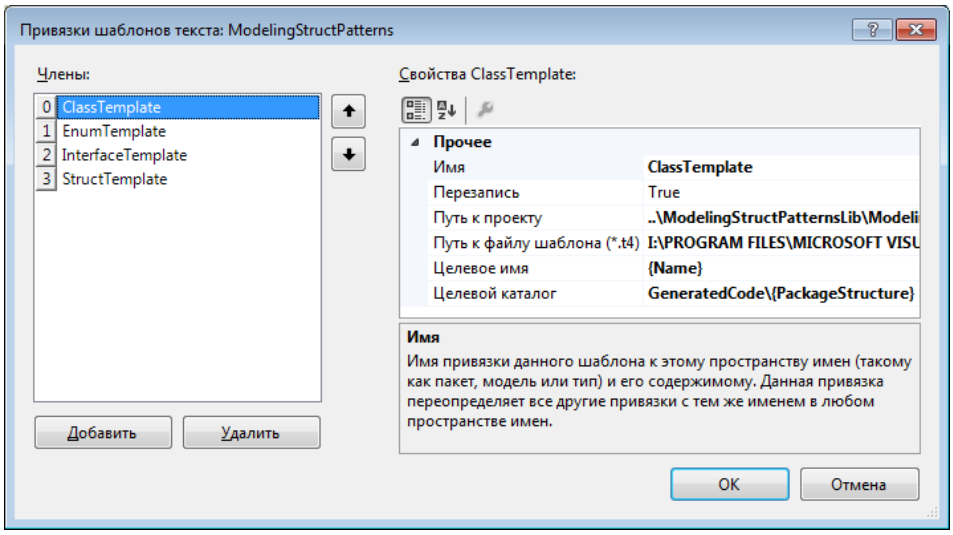


Рис. 4.9. Окно **Привязки шаблонов текста**

❒ ***Пример* 4.2. *Разработка приложения, использующего основные шаблоны GRASP.***

Требуется разработать приложение, в котором используются основные шаблоны GRASP. Приложение должно обеспечивать работу с экземплярами классов, описанных в примере 4.1.

Добавим в решение следующие проекты:

- **GraspModelingProject** – проект моделирования, содержащий диаграмму классов;
- **RailWayClassLibrary** – библиотека классов, которая содержит классы, определённые в примере 4.1;
- **ConsoleApp** – проект консольного приложения.

Схема классов, полученная с помощью Visual Studio, показана на рис. 4.10.

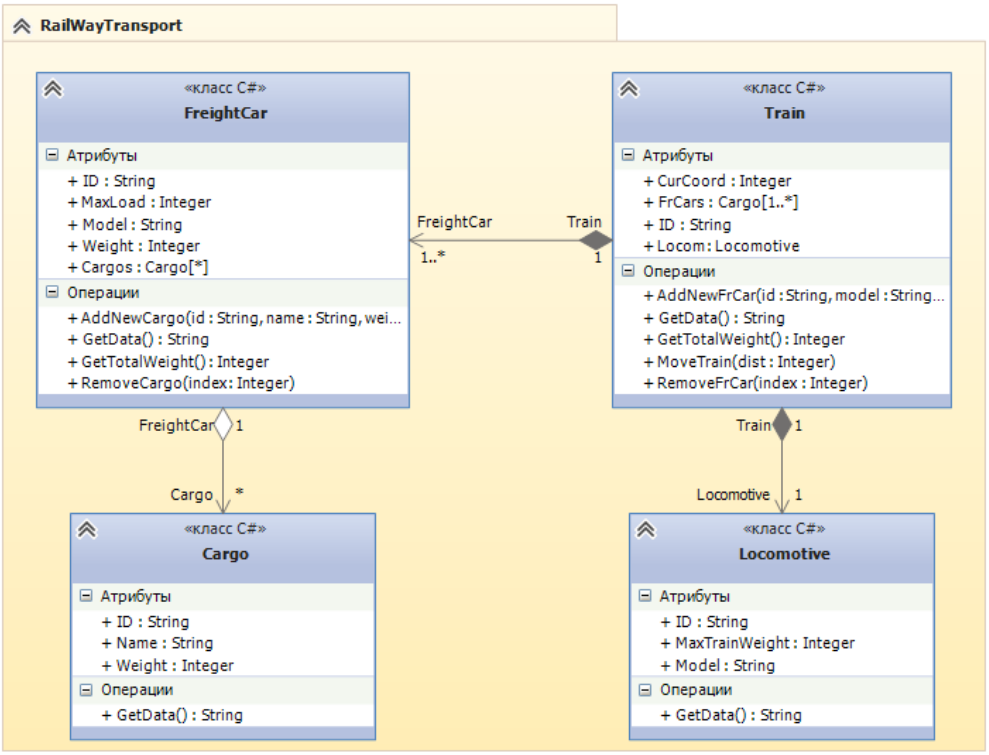


Рис. 4.10. Схема классов

Используя полученную схему классов UML, сгенерируем программный код на языке C#, который разместим в библиотеке классов **RailWayClassLibrary**.

Исходный код классов **Cargo**, **FreightCar**, **Locomotive** и **Train** представлен в листингах 4.1 – 4.4.

Работа с экземплярами указанных классов производится в методе **Main()** консольного приложения, исходный код которого показан в листинге 4.5.

Листинг 4.1. Исходный код класса **Cargo**

```
14      /// <summary>
15      /// Представляет перевозимые по Ж/Д грузы
16      /// </summary>
17      public class Cargo
18      {
19          /// <summary>
20          /// Наименование
21          /// </summary>
22          public virtual string Name { get; set; }
23
24          /// <summary>
25          /// Вес, т
26          /// </summary>
27          public virtual int Weight { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public Cargo()
33          {
34              Name = "Груз";
35              Weight = 1;
36          }
37
38          /// <summary>
39          /// Конструктор с параметрами
40          /// </summary>
41          /// <param name="name">Наименование</param>
42          /// <param name="weight">Вес, т</param>
43          public Cargo(string name, int weight)
44          {
45              Name = name;
46              Weight = weight;
47          }
48
49          /// <summary>
50          /// Возвращает строку с данными о грузе
51          /// </summary>
52          /// <returns>Строка с данными о грузе</returns>
53          public virtual string GetData()
54          {
55              return string.Format("Данные о грузе:\n" +
56                                   "- наименов.: {0}\n" +
57                                   "- вес: {1} т\n", Name, Weight);
58          }
59      }
```

Листинг 4.2. Исходный код класса **FreightCar** (часть 1)

```
14      /// <summary>
15      /// Представляет вагоны
16      /// </summary>
17      public class FreightCar
18      {
19          /// <summary>
20          /// Идентификационный номер
21          /// </summary>
22          public virtual string ID { get; set; }
23
24          /// <summary>
25          /// Модель
26          /// </summary>
27          public virtual string Model { get; set; }
28
29          /// <summary>
30          /// Грузоподъёмность, т
31          /// </summary>
32          public virtual int MaxLoad { get; set; }
33
34          /// <summary>
35          /// Вес, т
36          /// </summary>
37          public virtual int Weight { get; set; }
38
39          /// <summary>
40          /// Список грузов
41          /// </summary>
42          public virtual List<Cargo> Cargos { get; set; }
43
44          /// <summary>
45          /// Конструктор по умолчанию
46          /// </summary>
47          public FreightCar()
48          {
49              ID = "000-00000";
50              Model = "Вагон 00-000";
51              MaxLoad = 50;
52              Weight = 20;
53              Cargos = new List<Cargo>();
54          }
55
56          /// <summary>
57          /// Конструктор с параметрами
58          /// </summary>
59          /// <param name="model">Модель</param>
60          /// <param name="maxLoad">Грузоподъёмность, т</param>
61          /// <param name="weight">Вес, т</param>
62          public FreightCar(string id, string model, int maxLoad, int weight)
63          {
64              ID = id;
65              Model = model;
66              MaxLoad = maxLoad;
67              Weight = weight;
68              Cargos = new List<Cargo>();
69          {
```

Листинг 4.2. Исходный код класса **FreightCar** (часть 2)

```
71          /// <summary>
72          /// Возвращает строку с данными о вагоне
73          /// </summary>
74          /// <returns>Строка с данными о вагоне</returns>
75          public virtual string GetData()
76          {
77              return string.Format("Данные о вагоне:\n" +
78                                   "- идентификатор: {0}\n" +
79                                   "- модель: {1}\n" +
80                                   "- грузоподъёмность: {2} т\n" +
81                                   "- вес: {3} т\n", ID, Model, MaxLoad, Weight);
82          }
83
84          /// <summary>
85          /// Возвращает общий вагона с грузом
86          /// </summary>
87          /// <returns>Общий вес вагона с грузом, т</returns>
88          public virtual int GetTotalWeight()
89          {
90              int totalWeight = Weight;
91
92              foreach (Cargo c in Cargos) totalWeight += c.Weight;
93
94              return totalWeight;
95          }
96
97          /// <summary>
98          /// Добавить новый груз в вагон
99          /// </summary>
100         /// <param name="name">Наименование груза</param>
101         /// <param name="weight">Вес груза, кг</param>
102         public virtual void AddNewCargo(string name, int weight)
103         {
104             if (GetTotalWeight() - Weight + weight <= MaxLoad)
105                 Cargos.Add(new Cargo(name, weight));
106             else
107                 throw new Exception("Общий вес груза больше допустимого!");
108         }
109
110         /// <summary>
111         /// Удаляет груз с заданным номером из вагона
112         /// </summary>
113         /// <param name="index">Номер груза</param>
114         public virtual void RemoveCargo(int index)
115         {
116             if (index >= 0 && index < Cargos.Count)
117                 Cargos.RemoveAt(index);
118             else
119                 throw new Exception("Груза с указанным номером не существует!");
120         }
121     }
```

Листинг 4.3. Исходный код класса **Locomotive**

```
14      /// <summary>
15      /// Представляет локомотивы
16      /// </summary>
17      public class Locomotive
18      {
19          /// <summary>
20          /// Модель
21          /// </summary>
22          public virtual string Model { get; set; }
23
24          /// <summary>
25          /// Максимальный вес поезда, т
26          /// </summary>
27          public virtual int MaxTrainWeight { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public Locomotive()
33          {
34              Model = "Электровоз 0000";
35              MaxTrainWeight = 5000;
36          }
37
38          /// <summary>
39          /// Конструктор с параметрами
40          /// </summary>
41          /// <param name="model">Модель</param>
42          /// <param name="maxTrainWeight">Максимальный вес поезда, т</param>
43          public Locomotive(string model, int maxTrainWeight)
44          {
45              Model = model;
46              MaxTrainWeight = maxTrainWeight;
47          }
48
49          /// <summary>
50          /// Возвращает строку с данными о локомотиве
51          /// </summary>
52          /// <returns>Строка с данными о локомотиве</returns>
53          public virtual string GetData()
54          {
55              return string.Format("Данные о локомотиве:\n" +
56                                   "- модель: {0}\n" +
57                                   "- макс. вес поезда: {1} т\n",
58                                   Model, MaxTrainWeight);
59          }
60      }
```

Листинг 4.4. Исходный код класса **Train** (часть 1)

```
14      /// <summary>
15      /// Представляет поезда
16      /// </summary>
17      public class Train
18      {
19          /// <summary>
20          /// Идентификационный номер
21          /// </summary>
22          public virtual string ID { get; set; }
23
24          /// <summary>
25          /// Текущая координата, км
26          /// </summary>
27          public virtual int CurCoord { get; set; }
28
29          /// <summary>
30          /// Список прицепленных вагонов
31          /// </summary>
32          public virtual List<FreightCar> FrCars { get; set; }
33
34          /// <summary>
35          /// Локомотив
36          /// </summary>
37          public virtual Locomotive Locom { get; set; }
38
39          /// <summary>
40          /// Конструктор по умолчанию
41          /// </summary>
42          public Train()
43          {
44              ID = "00000";
45              CurCoord = 0;
46              FrCars = new List<FreightCar>();
47              Locom = new Locomotive("Электровоз 2ЭС10 Гранит", 9000);
48          }
49
50          /// <summary>
51          /// Конструктор с параметрами
52          /// </summary>
53          /// <param name="id">Идентификац. номер</param>
54          /// <param name="coor">Текущая координата, км</param>
55          public Train(string id, int coor)
56          {
57              ID = id;
58              CurCoord = coor;
59              FrCars = new List<FreightCar>();
60              Locom = new Locomotive("Электровоз 2ЭС10 Гранит", 9000);
61          }
62
63          /// <summary>
64          /// Возвращает строку с данными о поезде
65          /// </summary>
66          /// <returns>Строка с данными о поезде</returns>
67          public virtual string GetData()
68          {
69              return string.Format("Данные о поезде:\n" +
70                                   "- идентификатор: {0}\n" +
71                                   "- текущая координата: {1} км\n" +
72                                   "- общее число вагонов: {2} шт\n",
73                                   ID, CurCoord, FrCars.Count);
74          }
```

Листинг 4.4. Исходный код класса **Train** (часть 2)

```
76          /// <summary>
77          /// Возвращает общий вес поезда без локомотива
78          /// </summary>
79          /// <returns>Общий вес поезда без локомотива, т</returns>
80          public virtual int GetTotalWeight()
81          {
82              int totalWeight = 0;
83
84              foreach (FreightCar fc in FrCars)
85                  totalWeight += fc.GetTotalWeight();
86
87              return totalWeight;
88          }
89
90          /// <summary>
91          /// Прицепить новый вагон к поезду
92          /// </summary>
93          /// <param name="model">Модель вагона</param>
94          /// <param name="maxLoad">Грузоподъёмность, т</param>
95          /// <param name="weight">Вес вагона, т</param>
96          public virtual void AddNewFrCar(string id, string model,
97              int maxLoad, int weight)
98          {
99              FreightCar fc = new FreightCar(id, model, maxLoad, weight);
100             FrCars.Add(fc);
101         }
102
103         /// <summary>
104         /// Двигать поезд на заданное расстояние
105         /// </summary>
106         /// <param name="dist">Перемещение, км</param>
107         public virtual void MoveTrain(int dist)
108         {
109             if (GetTotalWeight() <= Locom.MaxTrainWeight)
110                 CurCoord += dist;
111             else
112                 throw new Exception("Слишком большой вес! " +
113                                     "Локомотив не может тянуть поезд!");
114         }
115
116         /// <summary>
117         /// Отцепить от поезда вагон с заданным номером
118         /// </summary>
119         /// <param name="index">Номер вагона</param>
120         public virtual void RemoveFrCar(int index)
121         {
122             if (index >= 0 && index < FrCars.Count)
123                 FrCars.RemoveAt(index);
124             else
125                 throw new Exception("Вагона с указанным номером не существует!");
126         }
127     }
```

Листинг 4.5. Исходный код класса **Program**

```
12      class Program
13      {
14          static void Main(string[] args)
15          {
16              Console.Title = "Работа с шаблонами GRASP";
17
18              Train train1 = new Train("018053", 123);
19
20              train1.AddNewFrCar("120-04118", "Крытый вагон 11-217", 68, 23);
21              train1.AddNewFrCar("120-04145", "Крытый вагон 11-286", 67, 27);
22              train1.AddNewFrCar("125-05603", "Платформа 13-401", 69, 21);
23              train1.AddNewFrCar("129-05710", "Цистерна 16-327", 62, 24);
24
25              train1.FrCars[0].AddNewCargo("Электрооборудов.", 21);
26              train1.FrCars[1].AddNewCargo("Ящики деревянные, 47 шт", 18);
27              train1.FrCars[2].AddNewCargo("ЖБ Опора, 2 шт", 4);
28              train1.FrCars[2].AddNewCargo("ЖБ Балка, 5 шт", 12);
29              train1.FrCars[2].AddNewCargo("ЖБ Плита, 1 шт", 3);
30              train1.FrCars[3].AddNewCargo("Мазут", 38);
31
32              Console.WriteLine(train1.GetData());
33              Console.WriteLine("Общий вес поезда {0}: {1} т\n",
34                  train1.ID, train1.GetTotalWeight());
35
36              Console.WriteLine(train1.Locom.GetData());
37
38              Console.WriteLine("Вагоны поезда {0}:\n", train1.ID);
39              foreach (FreightCar fc in train1.FrCars)
40              {
41                  Console.WriteLine("* " + fc.GetData());
42              }
43
44              Console.WriteLine("Грузы в вагоне {0}:\n", train1.FrCars[2].ID);
45              foreach (Cargo c in train1.FrCars[2].Cargos)
46              {
47                  Console.WriteLine("* " + c.GetData());
48              }
49              Console.WriteLine("Общий вес вагона {0} с грузом: {1} т\n",
50                  train1.FrCars[2].ID, train1.FrCars[2].GetTotalWeight());
51
52              train1.MoveTrain(-75);
53              train1.RemoveFrCar(3);
54              Console.WriteLine(train1.GetData());
55
56              train1.FrCars[2].RemoveCargo(0);
57              train1.FrCars[2].RemoveCargo(0);
58              train1.FrCars[2].AddNewCargo("Металлопрокат", 16);
59
60              Console.WriteLine("Грузы в вагоне {0}:\n", train1.FrCars[2].ID);
61              foreach (Cargo c in train1.FrCars[2].Cargos)
62              {
63                  Console.WriteLine(c.GetData());
64              }
65
66              Console.Read();
67          }
```

Результат работы консольного приложения показан на рис. 4.11. ❒

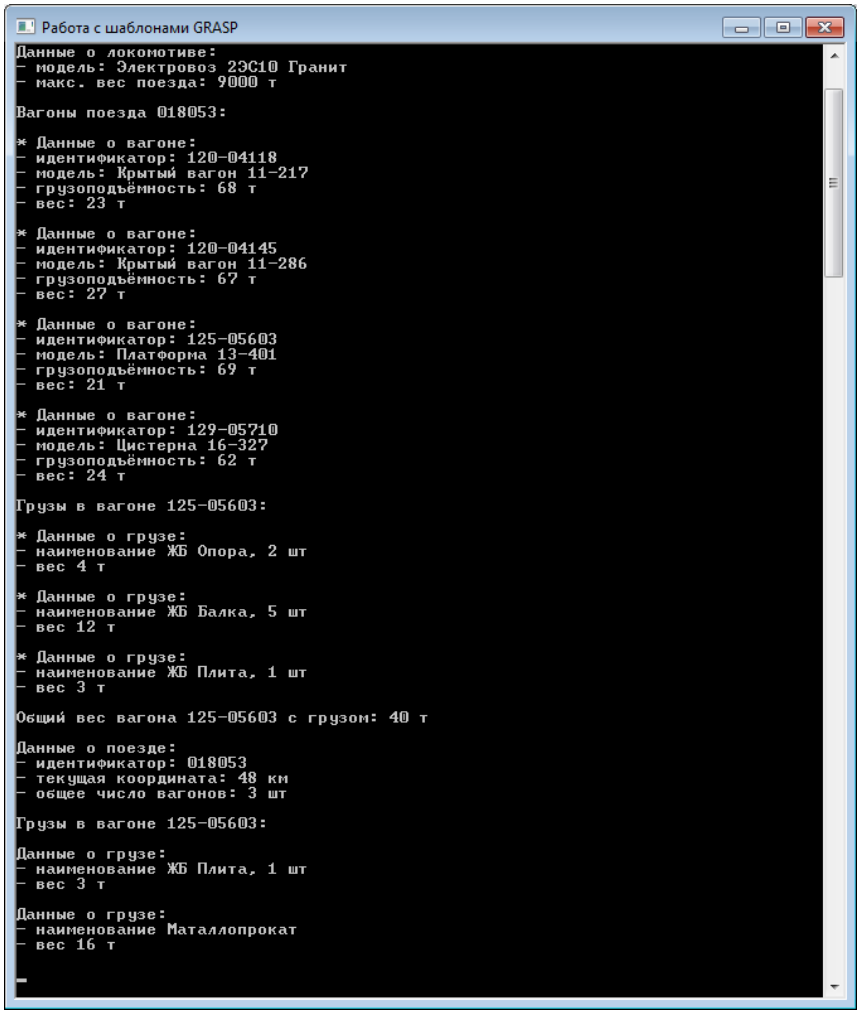


Рис. 4.11. Результат работы консольного приложения

❒ ***Пример* 4.3. *Разработка приложения Windows Forms.***

Требуется разработать приложение Windows Forms, которое позволяет работать с библиотекой классов **RailWayClassLibrary**, разработанной в примере 4.2.

Приложение должно обеспечивать создание и удаление экземпляров классов, а также возможность просмотра и редактирования данных об объектах. Кроме того, должна быть обеспечена возможность сохранения данных в XML-документе (через сериализацию), а также их загрузка (через десериализацию) при запуске приложения.

Созданные экземпляры классов **Train**, **FreightCar**, **Cargo** будут представлены элементами трёх списков **ListBox**. Данные объектов будут отображаться с помощью элемента управления **DataGridView** при выделении любого элемента одного из списков.

Интерфейс пользователя разрабатываемого приложения показан на рис. 4.12.

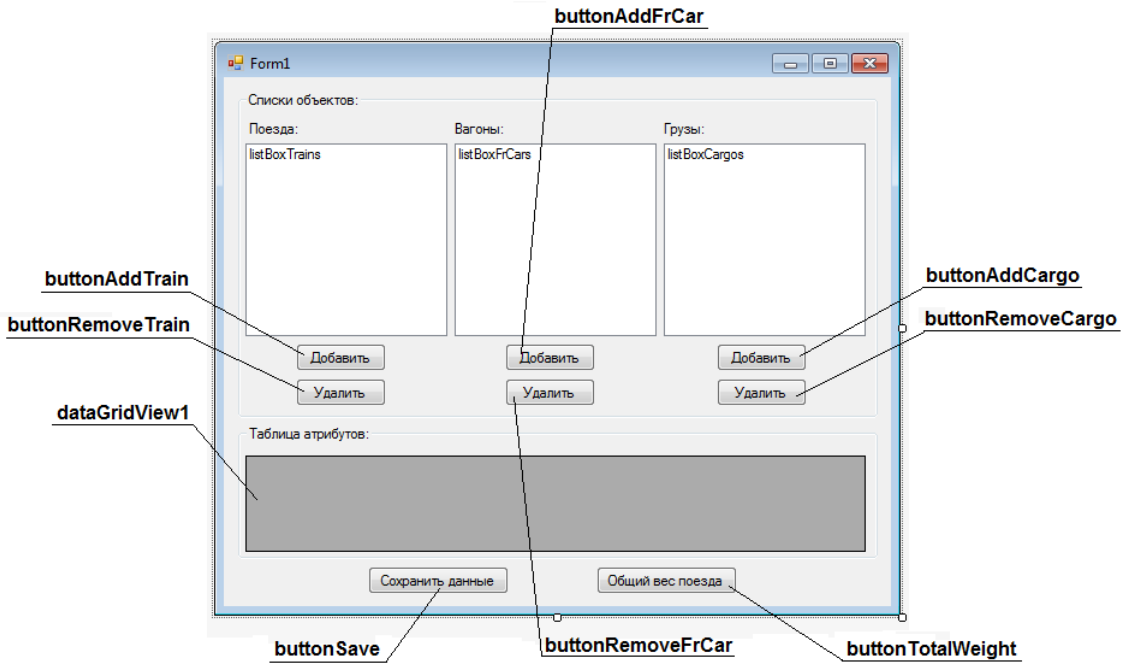


Рис. 4.12. Интерфейс пользователя

Исходный код класса **Form1** представлен в листинге 4.6.

Главное окно работающего приложения показано на рис. 4.13, а окна сообщений – на рис. 4.14.

В листинге 4.7 приведён код XML-документа, полученного путём сериализации объектов.

Листинг 4.6. Исходный код класса **Form1** (часть 1)

```
using System.Xml;
using System.Xml.Serialization;
using RailWayTransport;

namespace WindowsFormsApp
{
    public partial class Form1 : Form
    {
        List<Train> trains; // Список поездов
        string uri; // URI XML-документа

        public Form1()
        {
            InitializeComponent();
            trains = new List<Train>();
            uri = "RailWayTransport.xml";
        }

        /// <summary>
        /// Заполнить элементами список listBoxTrains
        /// </summary>
        private void GenerListBoxTrains()
        {
            listBoxTrains.Items.Clear();

            foreach (Train tr in trains)
            {
                listBoxTrains.Items.Add(tr.ID);
            }
        }

        /// <summary>
        /// Заполнить данными о поезде элемент dataGridView1
        /// </summary>
        private void GenerDataGridTrain()
        {
            dataGridView1.Columns.Clear();
            dataGridView1.Columns.Add("c1", "ID");
            dataGridView1.Columns.Add("c2", "Координата, км");

            int indexTr = listBoxTrains.SelectedIndex;
            if (indexTr < 0) return;

            string[] trData = new string[] { trains[indexTr].ID,
                trains[indexTr].CurCoord.ToString() };

            dataGridView1.Rows.Add(trData);
        }
```

Листинг 4.6. Исходный код класса **Form1** (часть 2)

```
59          /// <summary>
60          /// Заполнить элементами список listBoxFrCars
61          /// </summary>
62          private void GenerListBoxFrCars()
63          {
64              listBoxFrCars.Items.Clear();
65              int indexTr = listBoxTrains.SelectedIndex;
66              if (indexTr < 0) return;
67
68              foreach (FreightCar fc in trains[indexTr].FrCars)
69              {
70                  listBoxFrCars.Items.Add(fc.ID);
71              }
72          }
73
74          /// <summary>
75          /// Заполнить данными о вагоне элемент dataGridView1
76          /// </summary>
77          private void GenerFrCarsDataGrid()
78          {
79              dataGridView1.Columns.Clear();
80              dataGridView1.Columns.Add("c1", "ID");
81              dataGridView1.Columns.Add("c2", "Модель");
82              dataGridView1.Columns.Add("c3", "Грузопод., т");
83              dataGridView1.Columns.Add("c4", "Вес, т");
84
85              int indexTr = listBoxTrains.SelectedIndex;
86              int indexFc = listBoxFrCars.SelectedIndex;
87              if (indexTr < 0 || indexFc < 0) return;
88
89              string[] fcData = new string[] {
90                  trains[indexTr].FrCars[indexFc].ID,
91                  trains[indexTr].FrCars[indexFc].Model,
92                  trains[indexTr].FrCars[indexFc].MaxLoad.ToString(),
93                  trains[indexTr].FrCars[indexFc].Weight.ToString()};
94
95              dataGridView1.Rows.Add(fcData);
96          }
97
98          /// <summary>
99          /// Заполнить элементами список listBoxCargos
100         /// </summary>
101         private void GenerListBoxCargos()
102         {
103             listBoxCargos.Items.Clear();
104
105             int indexTr = listBoxTrains.SelectedIndex;
106             int indexFc = listBoxFrCars.SelectedIndex;
107             if (indexTr < 0 || indexFc < 0) return;
108
109             foreach (Cargo cg in trains[indexTr].FrCars[indexFc].Cargos)
110             {
111                 listBoxCargos.Items.Add(cg.Name);
112             }
113         }
```

Листинг 4.6. Исходный код класса **Form1** (часть 3)

```
        /// <summary>
        /// Заполнить данными о грузе элемент dataGridView1
        /// </summary>
        private void GenerCargosDataGrid()
        {
            dataGridView1.Columns.Clear();
            dataGridView1.Columns.Add("c1", "Наименование");
            dataGridView1.Columns.Add("c2", "Вес, т");

            int indexTr = listBoxTrains.SelectedIndex;
            int indexFc = listBoxFrCars.SelectedIndex;
            int indexCg = listBoxCargos.SelectedIndex;
            if (indexTr < 0 || indexFc < 0 || indexCg < 0) return;

            string[] cgData = new string[] {
                trains[indexTr].FrCars[indexFc].Cargos[indexCg].Name,
                trains[indexTr].FrCars[indexFc].Cargos[indexCg].Weight.ToString()};

            dataGridView1.Rows.Add(cgData);
        }

        /// <summary>
        /// Возвращает строковый массив с данными из первой строки
        /// элемента dataGridView1
        /// </summary>
        /// <returns>Строковый массив с данными</returns>
        private string[] GetDataFromGrid()
        {
            // Число столбцов элемента dataGridView1
            int n = dataGridView1.ColumnCount;
            string[] dataArray = new string[n];
            // Цикл для заполнения массива dataArray из ячеек dataGridView1
            for (int i = 0; i < n; i++)
                dataArray[i] = dataGridView1.Rows[0].Cells[i].Value.ToString();
            return dataArray;
        }

        /// <summary>
        /// Десериализовать XML-данные о объектах
        /// </summary>
        private void DeserializeObjects()
        {
            try
            {
                XmlSerializer xmlSzr = new XmlSerializer(typeof(List<Train>));
                XmlReader xmlRd = XmlReader.Create(uri);
                trains = xmlSzr.Deserialize(xmlRd) as List<Train>;
                xmlRd.Close();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message, "Ошибка!", MessageBoxButtons.OK,
                    MessageBoxIcon.Exclamation);
            }
        }
```

Листинг 4.6. Исходный код класса **Form1** (часть 4)

```
171         // Обработчик события "Загрузить форму Form1"
172         private void Form1_Load(object sender, EventArgs e)
173         {
174             this.Text = "Железнодорожный транспорт (Турчин Д.Е., каф. ИиАПС)";
175             buttonAddTrain.Enabled = false;
176             buttonRemoveTrain.Enabled = false;
177             buttonAddFrCar.Enabled = false;
178             buttonRemoveFrCar.Enabled = false;
179             buttonAddCargo.Enabled = false;
180             buttonRemoveCargo.Enabled = false;
181             DeserializeObjects();
182             GenerListBoxTrains();
183 
184         }
185 
186         // Обработчик события "Щелчёк элемента управления listBoxTrains"
187         private void listBoxTrains_Click(object sender, EventArgs e)
188         {
189             GenerDataGridTrain();
190             GenerListBoxFrCars();
191             buttonAddTrain.Enabled = true;
192             buttonRemoveTrain.Enabled = true;
193             buttonAddFrCar.Enabled = false;
194             buttonRemoveFrCar.Enabled = false;
195             buttonAddCargo.Enabled = false;
196             buttonRemoveCargo.Enabled = false;
197         }
198 
199         // Обработчик события "Щелчёк элемента управления listBoxFrCars"
200         private void listBoxFrCars_Click(object sender, EventArgs e)
201         {
202             GenerFrCarsDataGrid();
203             GenerListBoxCargos();
204             buttonAddFrCar.Enabled = true;
205             buttonRemoveFrCar.Enabled = true;
206             buttonAddCargo.Enabled = false;
207             buttonRemoveCargo.Enabled = false;
208         }
209 
210         // Обработчик события "Щелчёк элемента управления listBoxCargos"
211         private void listBoxCargos_Click(object sender, EventArgs e)
212         {
213             GenerCargosDataGrid();
214             buttonAddCargo.Enabled = true;
215             buttonRemoveCargo.Enabled = true;
216         }
217 
218         private void buttonAddTrain_Click(object sender, EventArgs e)
219         {
220             string[] trData = GetDataFromGrid();
221 
222             trains.Add(new Train(trData[0], Convert.ToInt32(trData[1])));
223             listBoxFrCars.Items.Clear();
224             listBoxCargos.Items.Clear();
225             GenerListBoxTrains();
226         }
```

Листинг 4.6. Исходный код класса **Form1** (часть 5)

```
228         private void buttonRemoveTrain_Click(object sender, EventArgs e)
229         {
230             int indexTr = listBoxTrains.SelectedIndex;
231             if (indexTr < 0) return;
232 
233             trains.RemoveAt(indexTr);
234             listBoxFrCars.Items.Clear();
235             listBoxCargos.Items.Clear();
236             GenerListBoxTrains();
237         }
238 
239         private void buttonAddFrCar_Click(object sender, EventArgs e)
240         {
241             int indexTr = listBoxTrains.SelectedIndex;
242             if (indexTr < 0) return;
243 
244             string[] frData = GetDataFromGrid();
245 
246             trains[indexTr].AddNewFrCar(frData[0], frData[1],
247                 Convert.ToInt32(frData[2]), Convert.ToInt32(frData[3]));
248             listBoxCargos.Items.Clear();
249             GenerListBoxFrCars();
250         }
251 
252         private void buttonRemoveFrCar_Click(object sender, EventArgs e)
253         {
254             int indexTr = listBoxTrains.SelectedIndex;
255             int indexFc = listBoxFrCars.SelectedIndex;
256             if (indexTr < 0 || indexFc < 0) return;
257 
258             trains[indexTr].RemoveFrCar(indexFc);
259             listBoxCargos.Items.Clear();
260             GenerListBoxFrCars();
261         }
262 
263         private void buttonAddCargo_Click(object sender, EventArgs e)
264         {
265             try
266             {
267                 int indexTr = listBoxTrains.SelectedIndex;
268                 int indexFc = listBoxFrCars.SelectedIndex;
269                 if (indexTr < 0 || indexFc < 0) return;
270 
271                 string[] cgData = GetDataFromGrid();
272 
273                 trains[indexTr].FrCars[indexFc].AddNewCargo(cgData[0],
274                     Convert.ToInt32(cgData[1]));
275                 GenerListBoxCargos();
276             }
277             catch (Exception exc)
278             {
279                 MessageBox.Show(exc.Message, "Внимание!");
280             }
281         }
```

Листинг 4.6. Исходный код класса **Form1** (часть 6)

```
283         private void buttonRemoveCargo_Click(object sender, EventArgs e)
284         {
285             int indexTr = listBoxTrains.SelectedIndex;
286             int indexFc = listBoxFrCars.SelectedIndex;
287             int indexCg = listBoxCargos.SelectedIndex;
288             if (indexTr < 0 || indexFc < 0 || indexCg < 0) return;
289 
290             trains[indexTr].FrCars[indexFc].RemoveCargo(indexCg);
291             GenerListBoxCargos();
292         }
293 
294         private void buttonSave_Click(object sender, EventArgs e)
295         {
296             XmlSerializer xmlSzr = new XmlSerializer(typeof(List<Train>));
297             XmlWriterSettings xmlWrS = new XmlWriterSettings();
298             xmlWrS.Indent = true;
299             XmlWriter xmlWr = XmlWriter.Create(uri, xmlWrS);
300             xmlSzr.Serialize(xmlWr, trains);
301             xmlWr.Close();
302         }
303 
304         private void buttonTotalWeight_Click(object sender, EventArgs e)
305         {
306             int indexTr = listBoxTrains.SelectedIndex;
307             if (indexTr < 0) return;
308 
309             MessageBox.Show(string.Format("Общий вес поезда {0}: {1} т",
310                 trains[indexTr].ID, trains[indexTr].GetTotalWeight()),
311                 "Результаты расчёта");
312         }
313     }
```

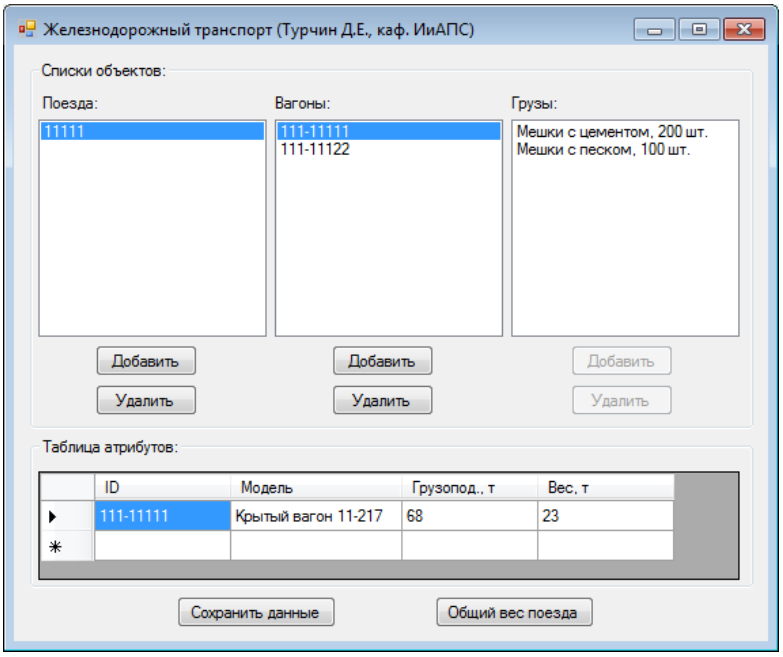


Рис. 4.13. Главное окно приложения

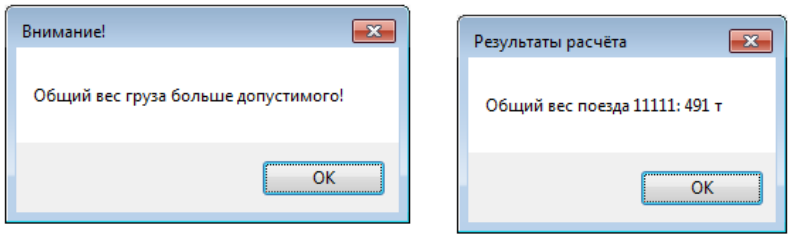


Рис. 4.14. Окна сообщений

Листинг 4.7. Код XML-документа, полученного путём сериализации объектов

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfTrain xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2
  <Train>
    <ID>11111</ID>
    <CurCoord>0</CurCoord>
    <FrCars>
      <FreightCar>
        <ID>111-11111</ID>
        <Model>Крытый вагон 11-217</Model>
        <MaxLoad>68</MaxLoad>
        <Weight>23</Weight>
        <Cargos>
          <Cargo>
            <Name>Мешки с цементом, 200 шт.</Name>
            <Weight>10</Weight>
          </Cargo>
          <Cargo>
            <Name>Мешки с песком, 100 шт.</Name>
            <Weight>5</Weight>
          </Cargo>
        </Cargos>
      </FreightCar>
      <FreightCar>
        <ID>111-11122</ID>
        <Model>Крытый вагон 11-286</Model>
        <MaxLoad>67</MaxLoad>
        <Weight>27</Weight>
        <Cargos>
          <Cargo>
            <Name>Металлопрокат</Name>
            <Weight>8</Weight>
          </Cargo>
        </Cargos>
      </FreightCar>
    </FrCars>
    <Locom>
      <Model>Электровоз 2ЭС10 Гранит</Model>
      <MaxTrainWeight>9000</MaxTrainWeight>
    </Locom>
  </Train>
</ArrayOfTrain>
```

4.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. На основе заданных классов предметной области, а также их обязанностей (табл. 4.1) разработать диаграмму классов UML, в которой применены такие шаблоны GRASP, как Information Expert и Creator.
3. Создать проект библиотеки классов, в которой должны присутствовать модули классов, отмеченных на диаграмме классов. Реализовать атрибуты и операции классов, а также отношения между ними с помощью средств языка C#.
4. Добавить в решение проект консольного приложения и связать его с полученной библиотекой классов. Продемонстрировать в консольном приложении выполнение классами требуемых обязанностей.
5. Дополнительно добавить в решение проект приложения Windows Forms. Реализовать в приложении представление данных об объектах и возможность добавления новых объектов. Обеспечить возможность сохранения данных путём сериализации объектов в XML-документ.
6. Оформить и защитить отчет по лабораторной работе.

Таблица 4.1

Классы и их обязанности

№ вар.	Классы	Обязанности
1, 9, 17	**Гостиница**. **Гостиничный номер**. **Клиент**.	***Знать.*** Стоимость одного дня проживания. Число дней проживания. ***Делать.*** Добавить клиента в номер. Удалить клиента из номера. Определить общее число клиентов. Определить суммарную плату за проживание.
2, 10, 18	**Цех**. **Станок**.	***Знать.*** Какие станки, обслуживаются рабочим.

	Рабочий (может обслуживать несколько станков).	Надбавка за обслуживание более чем одного станка. ***Делать.*** Добавить рабочего в цех. Добавить рабочему станок. Определить зарплату (зависит от числа обслуживаемых станков). Определить суммарные расходы на зарплату.
3, 11, 19	**Магазин**. **Товар**. **Продавец-консультант**.	***Знать.*** Какие товары, проданы сотрудником. Бонус за продажу единицы товара. ***Делать.*** Добавить товар в магазин. Продать товар. Определить зарплату (зависит от числа проданных товаров). Определить общую выручку.
4, 12, 20	**Кинотеатр**. **Фильм**. **Сеанс**.	***Знать.*** Какие сеансы проводятся на фильм. Число проданных билетов на сеанс. ***Делать.*** Добавить фильм. Добавить сеанс. Определить общее число зрителей. Определить общую выручку.
5, 13, 21	**Заказ в ресторане**. **Блюдо**. **Ингредиент**.	***Знать.*** Какие блюда входят в состав заказа. Какие ингредиенты входят в состав блюда. ***Делать.*** Добавить в блюдо ингредиент. Добавить в заказ блюдо. Определить стоимость блюда. Определить стоимость заказа.
6, 14, 22	**Институт**. **Студент**. **Студ. группа**.	***Знать.*** Какие студенты принадлежат к группе. Результаты сессии. ***Делать.*** Добавить студента в группу. Удалить студента из группы. Определить размер стипендии (зависит от результатов сессии). Определить общий размер выплат стипен-

		дии.
7, 15, 23	**Склад**. **Стеллаж.** **Товар.**	***Знать.*** Какие товары расположены на стеллаже. Грузонесущая способность стеллажа. ***Делать.*** Добавить товар. Удалить товар. Определить общую стоимость товаров. Определить общий вес товаров.
8, 16, 24	**Парикмахерская**. **Сотрудник**. **Клиент**.	***Знать.*** Какие клиенты, обслужены сотрудником. Плата за услуги. ***Делать.*** Добавить клиента. Добавить сотрудника. Определить зарплату сотрудника (зависит от числа обслуженных клиентов). Определить общие затраты на оплату труда.

4.4. Контрольные вопросы

1. В чём заключается подход проектирования на основе обязанностей (RDD)?
2. Какие выделяют виды обязанностей объекта?
3. Какие выделяют основные шаблоны GRASP?
4. Какую проблему решает шаблон Expert и в чём заключается это решение?
5. Какие классы должны отвечать за создание объектов согласно шаблону Creator?
6. К каким проблемам приводит использование в системе классов с высокой степенью связывания?
7. Что понимают под функциональным сцеплением в объектно-ориентированным проектировании?
8. Чем плохо использование классов с низкой степенью сцепления?

5. РАБОТА СО СТРУКТУРНЫМИ ШАБЛОНАМИ GOF НА ЯЗЫКЕ C#

5.1. Цель и задачи работы

Цель работы – прибрести умение разрабатывать приложения на языке C# с использованием таких структурных шаблонов проектирования из каталога GoF, как Адаптер, Фасад и Декоратор.

Основные задачи:

• Освоить такие структурные шаблоны проектирования, как Адаптер, Фасад и Декоратор.

• Научиться использовать структурные шаблоны GoF в приложениях на языке C#.

Работа рассчитана на 6 часов.

5.2. Основные теоретические сведения

5.2.1. Понятие и виды структурных шаблонов GoF. Шаблон Адаптер

Понятие и виды структурных шаблонов GoF. Шаблон Адаптер.

Важной группой шаблонов проектирования из каталога GoF[1] являются структурные шаблоны.

Структурные шаблоны проектирования (англ. *structural design patterns*) предназначены для образования из классов и объектов более крупных структур. Структурные шаблоны могут применяться как для объединения, так и для разделения элементов приложения.

К структурным шаблонам из каталога GoF относятся:

- **Adapter** – Адаптер.
- **Facade** – Фасад.

[1] Каталогом GoF (Gang of Four – «Банда четырёх») называется основополагающая работа по шаблонам проектирования «Design Patterns – Elements of Reusable Object-Oriented Software», написанная такими авторами, как Эрих Гамма, Ральф Джонсон, Ричард Хелм и Джон Власидес. В данной работе содержится описание 23 шаблонов объектно-ориентированного проектирования, которые разделены на структурные, поведенческие и порождающие шаблоны.

- **Proxy** – Заместитель.
- **Decorator** – Декоратор.
- **Composite** – Компоновщик.
- **Bridge** – Мост.
- **Flyweight** – Приспособленец.

Простейшим из структурных шаблонов является Адаптер (*англ. Adapter*).

Адаптер – структурный шаблон проектирования, предназначенный для преобразования интерфейса класса к другому интерфейсу, на который рассчитан клиент.

Проблема.

Как обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких классов с разными интерфейсами?

Решение.

Преобразовать исходный интерфейс класса к другому виду с помощью промежуточного объекта-адаптера.

Основными участниками решения являются:

- **Client** – клиент, использующий целевой интерфейс;
- **ITarget** – целевой интерфейс;
- **Adapter** – адаптер, реализующий интерфейс **ITarget**;
- **Adapted** – адаптируемый класс, имеющий интерфейс несовместимый с интерфейсом клиента.

Работа клиента с адаптируемым объектом происходит следующим образом:

1. Клиент обращается с запросом к объекту-адаптеру **Adapter**, вызывая его метод **Request()** через целевой интерфейс **ITarget**.

2. Адаптер **Adapter** преобразует запрос в один или несколько вызовов к адаптируемому объекту **Adapted**.

3. Клиент получает результаты вызова, не зная ничего о преобразованиях, выполненных адаптером.

Шаблон Адаптер имеет следующие разновидности:

- ***Адаптер объекта*** применяет для адаптации одного интерфейса к другому композицию объектов адаптируемого класса (рис. 5.1).
- ***Адаптер класса*** использует для адаптации наследование адаптируемому классу (рис. 5.2).

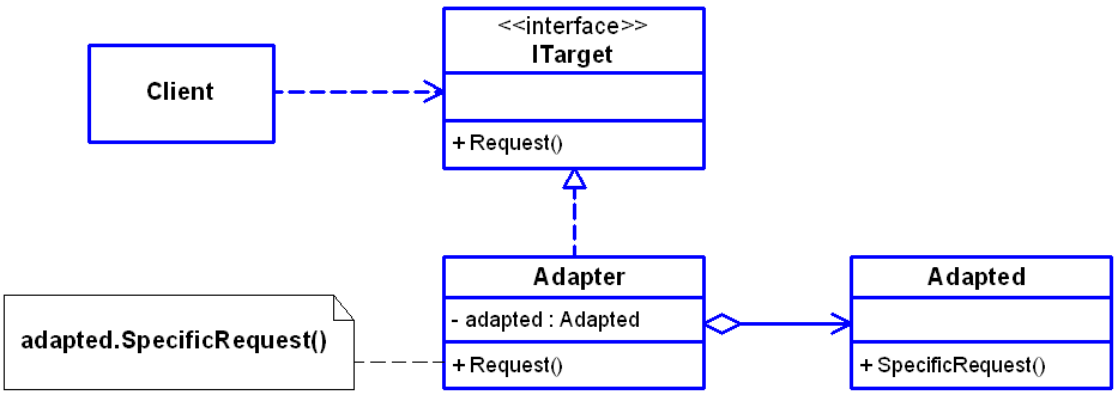


Рис. 5.1. Диаграмма классов шаблона Адаптер объекта

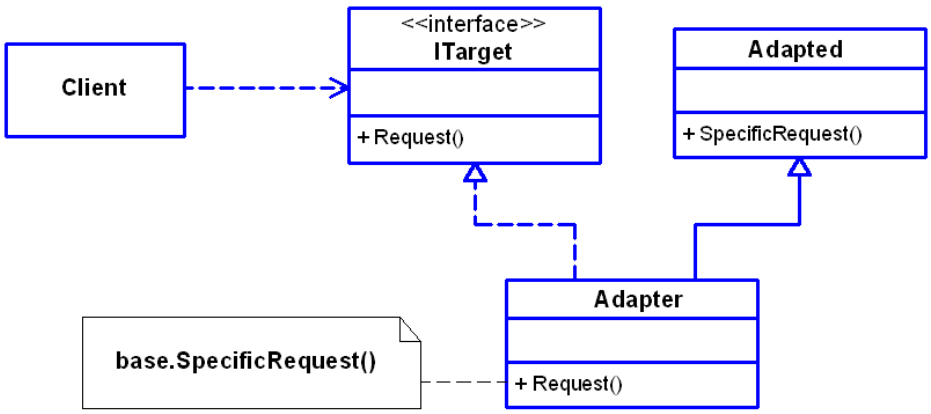


Рис. 5.2. Диаграмма классов шаблона Адаптер класса

Результаты.

Система становится независимой от интерфейса внешних классов (компонентов, библиотек). При переходе на использование внешних классов не требуется переделывать всю систему, достаточно переделать один класс **Adapter**.

Результаты применения адаптеров классов и объектов различаются.

Адаптер объекта:

• позволяет работать объекту **Adapter** со многими адаптируемыми объектами (например, с объектами класса **Adapted** и его производных классов);

• отличается сложностью при замещении операций класса **Adapted** (для этого может потребоваться создать класс производ-

ный от **Adapted** и добавить в класс **Adapter** ссылку на этот производный класс).

Адаптер классов:

• обеспечивает простой доступ к элементам адаптируемого класса, поскольку **Adapter** является производными классом от **Adapted**.

• характеризуется легкостью изменения адаптером операций адаптируемого класса **Adapted**;

• обладает возможностью работы только с одним адаптируемым классом (возможность адаптировать классы, производные от **Adaptee** отсутствует).

Реализация шаблона Адаптер на языке C#.

Простейший пример реализации шаблона Адаптер объекта на языке C# представлен в листинге 5.1.

Листинг 5.1. Пример реализации шаблона Адаптер объекта на языке C#

```
// Представляет целевой интерфейс
public interface ITarget
{
    void Request();
}

// Представляет адаптируемые объекты
class Adapted
{
    public void SpecificRequest()
    {
        Console.WriteLine("Вызван SpecificRequest()");
    }
}

// Представляет объекты-адаптеры
public class Adapter : ITarget
{
    Adapted adapted = new Adapted();

    public void Request()
    {
        adapted.SpecificRequest();
    }
```

```
    }

    // Клиент
    class Client
    {
        static void Main(string[] args)
        {
            ITarget target = new Adapter();
            target.Request();
        }
    }
```

Отличие реализации шаблона Адаптер класса будет заключаться только в коде класса **Adapter** (листинг 5.2).

Листинг 5.2. Исходный код класса **Adapter** для шаблона Адаптер класса

```
    // Представляет объекты-адаптеры
    public class Adapter : Adapted, ITarget
    {
        public void Request()
        {
            base.SpecificRequest();
        }
    }
```

В результате выполнения представленного кода в окне консоли будет выведен текст «Вызван SpecificRequest()», означающий, что был вызван соответствующий метод класса **Adapted**.

❒ ***Пример* 5.1. *Реализация шаблона Адаптер в приложении на языке C#.***

Требуется разработать приложение на языке C#, в котором для обеспечения совместимости интерфейсов используется шаблон Адаптер. Приложение должно состоять из библиотеки классов, содержащей адаптируемый класс, консольного приложения, выступающего в роли клиента, а также из библиотеки, содержащей целевой интерфейс и класс-адаптер.

Адаптируемый класс: **Pool** (представляет бассейны с водой).

Атрибуты:

 # maxH : double – максимальный уровень воды, см;

 # curH : double – текущий уровень воды, см;

 # poolS : double – площадь дна бассейна, м^2;

 # holeS : double – площадь отверстия для слива воды, см^2.

Операции:

+ CurrentH : int – возвращает и устанавливает текущий уровень воды в бассейне (свойство);

+ GetPourDownT(h : double) : int – определить время, за которое уровень воды опустится до h;

+ ToString() : string – возвращает строку с данными об объекте.

Требуемый интерфейс:

+ CurrentHeight : double – возвращает текущий уровень воды;

+ ModifH(dH : double) : void – изменить уровень воды в бассейне на величину **dH**, м;

+ CalcPourDownT() : int – определить время, за которое из бассейна выльется вся вода, с;

+ GetStringData() : string – возвращает строку с данными об объекте.

Время вытекания жидкости из сосуда можно найти следующим образом:

$$t = \frac{S}{\sigma}\sqrt{\frac{2}{g}}\left(\sqrt{H} - \sqrt{h}\right);$$

где S – площадь дна сосуда; σ – площадь отверстия в сосуде; H – уровень жидкости относительно дна сосуда; h – уровень, до которого опустится жидкость.

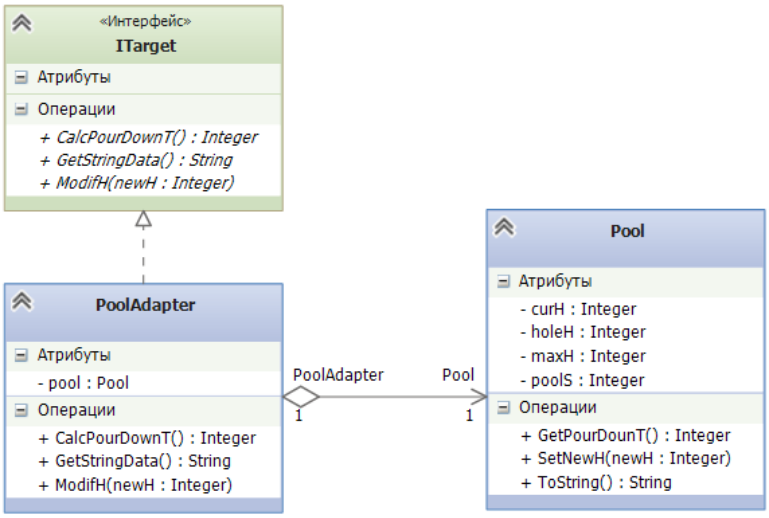


Рис. 5.3. Диаграмма классов проекта

Листинг 5.3. Исходный код класса **Pool** (адаптируемый класс, Часть 1)

```
/// <summary>
/// Представляет бассейны с водой (Адаптируемый класс)
/// </summary>
class Pool
{
    protected int maxH;     // Максимальный уровень воды, см
    protected int curH;     // Текущий уровень воды, см
    protected double poolS; // Площадь дна бассейна, м2
    protected int holeS;    // Площадь отверстия для слива воды, см2

    /// <summary>
    /// Конструктор с параметрами
    /// </summary>
    /// <param name="maxH">Максимальный уровень воды, см</param>
    /// <param name="curH">Текущий уровень воды, см</param>
    /// <param name="poolS">Площадь дна, м2</param>
    /// <param name="holeS">Площадь отверстия, см2</param>
    public Pool(int maxH, int curH, double poolS, int holeS)
    {
        this.maxH = maxH;
        this.curH = curH;
        this.poolS = poolS;
        this.holeS = holeS;
    }

    /// <summary>
    /// Конструктор по умолчанию (на основе парам. конструктора)
    /// </summary>
    public Pool()
        : this(200, 100, 20, 10) { }
```

Листинг 5.3. Исходный код класса **Pool** (адаптируемый класс, Часть 2)

```
        /// <summary>
        /// Возвращает и устанавливает текущий уровень воды
        /// </summary>
        public int CurrentH
        {
            get
            { return curH; }
            set
            { if ((value >= 0) && (value <= maxH)) curH = value; }
        }

        /// <summary>
        /// Определить время, за которое вода опустится до уровня h
        /// </summary>
        /// <param name="h">Уровень опускания воды, см</param>
        /// <returns>Время опускания воды, с</returns>
        public int GetPourDownT(int h)
        {
            double t = 0;   // Время опускания воды до уровня h, с
            double g = 9.8; // Ускорение свободного падаения, м/с2
            if ((h >= 0) && (h < curH))
            {
                t = (poolS * 10000 / holeS) * Math.Sqrt(2 / g) *
                (Math.Sqrt(curH / 100) - Math.Sqrt(h / 100));
            }
            return Convert.ToInt32(t);
        }

        /// <summary>
        /// Возвращает строку с данными о бассейне
        /// </summary>
        /// <returns>Строка с данными о бассейне</returns>
        public override string ToString()
        {
            return string.Format("Данные о бассейне:\n" +
                                 "- макс. уровень воды: {0} см\n" +
                                 "- текущ. уровень воды: {1} см\n" +
                                 "- площадь дна бассейна: {2} м2\n" +
                                 "- площадь отверстия для слива: {3} см2\n",
                                 maxH, curH, poolS, holeS);
        }
    }
```

Листинг 5.4. Исходный код интерфейса **ITarget** (целевой интерфейс)

```
/// <summary>
/// Целевой интерфейс
/// </summary>
public interface ITarget
{
    /// <summary>
    /// Возвращает текущий уровень воды, м
    /// </summary>
    double CurrentH { get; }

    /// <summary>
    /// Изменить уровень воды на величину dH
    /// </summary>
    /// <param name="h">Изменение уровня воды, м</param>
    void ModifH(double dH);

    /// <summary>
    /// Определить время, за которое выльется вся вода
    /// </summary>
    /// <returns>Время вытекания всей воды, с</returns>
    int CalcPourDownT();

    /// <summary>
    /// Возвращает строку с данными о объекте
    /// </summary>
    /// <returns>Строка с данными об объекте</returns>
    string GetStringData();
}
```

Листинг 5.5. Исходный код класса **PoolObjAdapter**

```
/// <summary>
/// Представляет Адаптеры объектов для класса Pool
/// </summary>
public class PoolObjAdapter : ITarget
{
    Pool pool;

    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public PoolObjAdapter()
    {
        pool = new Pool();
    }

    /// <summary>
    /// Конструктор с параметрами
    /// </summary>
    /// <param name="maxH">Максимальный уровень воды, см</param>
    /// <param name="curH">Текущий уровень воды, см</param>
    /// <param name="poolS">Площадь дна, м2</param>
    /// <param name="holeS">Площадь отверстия, см2</param>
    public PoolObjAdapter(int maxH, int curH, double poolS, int holeS)
    {
        pool = new Pool(maxH, curH, poolS, holeS);
    }

    /// <summary>
    /// Возвращает текущий уровень воды, м
    /// </summary>
    public double CurrentH
    {
        get
        { return (Convert.ToDouble(pool.CurrentH) / 100); }
    }

    /// <summary>
    /// Изменить уровень воды на величину dH
    /// </summary>
    /// <param name="h">Изменение уровня воды, м</param>
    public void ModifH(double dH)
    {
        pool.CurrentH += Convert.ToInt32(dH * 100);
    }

    /// <summary>
    /// Определить время, за которое выльется вся вода
    /// </summary>
    /// <returns>Время вытекания всей воды, с</returns>
    public int CalcPourDownT()
    {
        return pool.GetPourDownT(0);
    }

    /// <summary>
    /// Возвращает строку с данными о объекте
    /// </summary>
    /// <returns>Строка с данными об объекте</returns>
    public string GetStringData()
    {
        return pool.ToString();
    }
}
```

Листинг 5.6. Исходный код класса **Program** консольного приложения (клиент)

```
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Работа с шаблоном Адаптер";

        ITarget adapter = new PoolObjAdapter();

        Console.WriteLine(adapter.GetStringData());

        Console.WriteLine("Время слива всей воды из бассейна: {0} с",
            adapter.CalcPourDownT());

        adapter.ModifH(-0.25);

        Console.WriteLine("Текущий уровень воды: {0} м", adapter.CurrentH);

        Console.Read();
    }
```

Результат работы консольного приложения показан на рис. 5.4. ❒

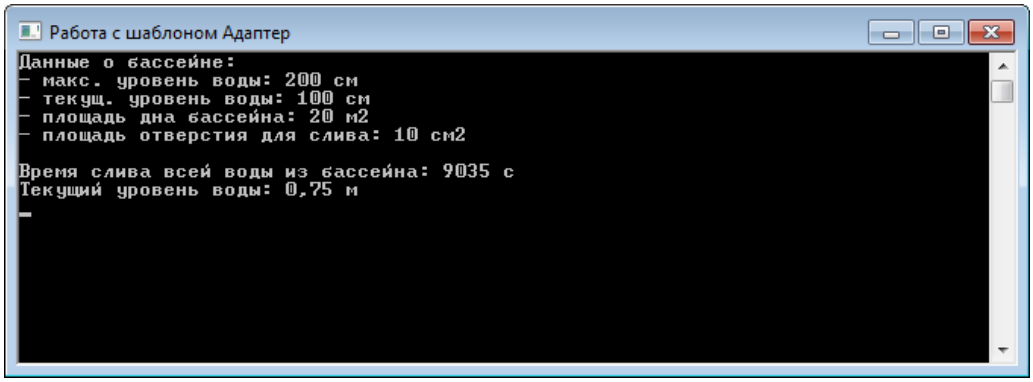


Рис. 5.4. Результат работы приложения

5.2.2. Шаблон Фасад

Шаблон Фасад.

Важным структурным шаблоном GoF является Фасад (*англ. Facade*), который упрощает интерфейс отдельной подсистемы – группы взаимосвязанных классов или компонентов.

Фасад – структурный шаблон проектирования, позволяющий скрыть сложность подсистемы путем сведения всех возмож-

ных вызовов к одному объекту (фасадному объекту), делегирующему их объектам подсистемы.

Проблема.

Как обеспечить унифицированный интерфейс с подсистемой, если нежелательна высокая степень связанности с этой подсистемой или реализация подсистемы может измениться?

Решение.

Определить одну точку взаимодействия с подсистемой – фасадный объект, обеспечивающий единый упрощенный интерфейс с подсистемой и возложить на него обязанность по взаимодействию с классами подсистемы.

Участники решения:

• **Client** – взаимодействует с фасадом и не имеет доступа к классам подсистемы;

• **Facade** – перенаправляет запросы клиентов к классам подсистемы;

• **Классы подсистемы** – выполняют работу, порученную объектом **Facade**, ничего не зная о существовании фасада, то есть не хранят ссылок на него.

Результаты.

• Клиенты изолируются от классов (компонентов) подсистемы, что уменьшает число объектов, с которыми клиенты взаимодействуют, и упрощает работу с подсистемой.

• Снижается степень связанности между клиентами и подсистемой, что позволяет изменять классы подсистемы, не затрагивая при этом клиентов.

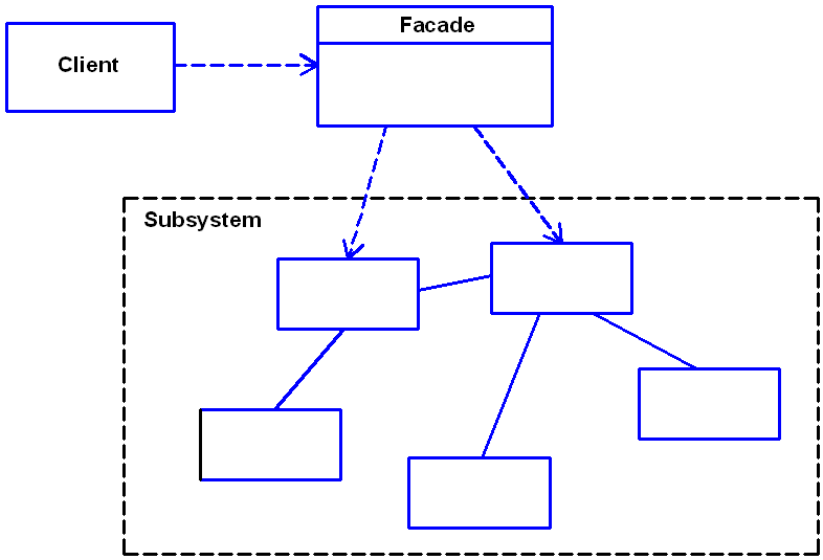


Рис. 5.5. Диаграмма классов шаблона Фасад

Реализация шаблона Фасад на языке C#.

Простой пример реализации шаблона Фасад на языке C# представлен в листинге 5.1.

Листинг 5.7. Пример реализации шаблона Фасад на языке C#

```
namespace Subsystem // Содержит классы подсистемы
{
    internal class ClassA
    {
        public string A1()
        {
            return "Метод A1() класса ClassA";
        }
        public string A2()
        {
            return "Метод A2() класса ClassA";
        }
    }

    internal class ClassB
    {
        public string B1()
        {
            return "Метод B1() класса ClassB";
        }
        public string B2()
        {
```

```
            return "Метод B2() класса ClassB";
        }
    }
}

    // Представляет фасадные объекты
    public class Facade
    {
        Subsystem.ClassA a;
        Subsystem.ClassB b;

        public Facade()
        {
            a = new Subsystem.ClassA();
            b = new Subsystem.ClassB();
        }
        public void F1()
        {
            Console.WriteLine("Метод F1() класса Facade:\n" +
                    "* {0}\n" +
                    "* {1}\n", a.A1(), b.B2());
        }
        public void F2()
        {
            Console.WriteLine("Метод F2() класса Facade:\n" +
                    "* {0}\n" +
                    "* {1}\n" +
                    "* {2}\n", a.A2(), b.B1(), b.B2());
        }
    }

    // Клиент
    class Client
    {
        static void Main(string[] args)
        {
            Facade facade = new Facade();
            facade.F1();
            facade.F2();
        }
    }
```

❒ ***Пример* 5.2. *Реализация шаблона Фасад в приложении на языке C#.***

Требуется разработать приложение по расчету максимального размера кредита. Приложение должно осуществлять расчет в зависимости от введенных пользователем данных и отображать

результат расчета. Расчет может производиться различным образом для различных типов кредита (потребительский кредит, ипотечный кредит, автокредит).

В качестве параметров для выполнения расчета будут выступать:

- средний ежемесячный доход;
- срок кредитования;
- наличие карты банка;
- вид собственности в качестве залога (автомобиль, квартира, земельный участок);
- размер непогашенных кредитов.

Предполагается, что процедура расчета взносов может меняться чаще, чем остальные элементы приложения. В этой ситуации может оказаться полезным использование шаблона Фасад, который должен обеспечивать доступ к различным процедурам расчета.

Для создания решения будем использовать следующие классы, размещаемые в отдельной библиотеке:

- **Credit** – представляет банковские кредиты (абстрактный класс);
- **ConsumeCredit** – представляет потребительские кредиты (производный от **Credit**);
- **EstateCredit** – представляет ипотечные кредиты (производный от **Credit**);
- **CarCredit** – представляет автокредиты (производный от **Credit**);
- **CreditFacade** – представляет фасадные объекты (статический класс).

Для классов **Credit**, **ConsumeCredit**, **EstateCredit**, **CarCredit** зададим пакетный уровень доступа (модификатор **internal** в C#) и разместим их в отдельном пакете (пространстве имён) **Credits**. Для класса **CreditFacade** и его операций будем использовать открытый доступ.

Диаграмма классов проектируемого решения, созданная средствами Visual Studio, показана на рис. 5.6.

В качестве клиента для фасада будем использовать консольное приложение.

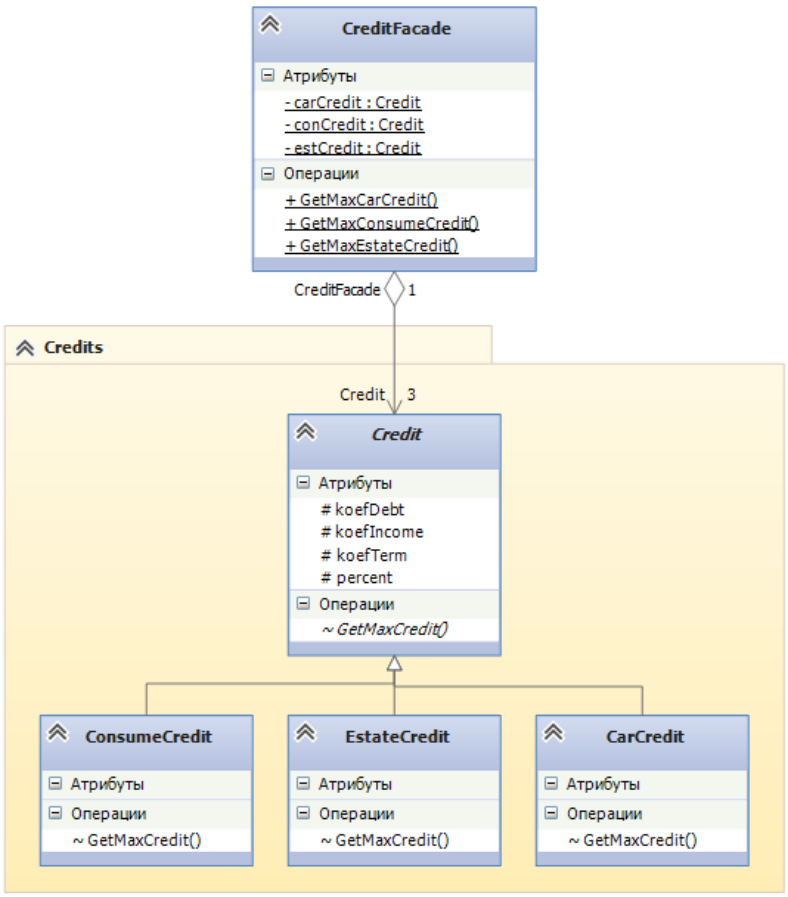


Рис. 5.6. Диаграмма классов решения

Исходный код абстрактного класса **Credit** представлен в листинге 5.8, а код производного от него класса **ConsumeCredit** – в листинге 5.9.

Исходный код статического класса **CreditFasade** приведён в листинге 5.10.

Листинг 5.8. Исходный код класса **Credit**

```
    /// <summary>
    /// Представляет кредиты
    /// </summary>
    internal abstract class Credit
    {
        // Коэффициент, учитывающий размер ежемесячного дохода
        protected double koefIncome;
        // Коэффициент, учитывающий срок кредитования
        protected double koefTerm;
        // Коэффициент, учитывающий общий размер долга по другим кредитам
        protected double koefDebt;
        protected double percent; // Процент

        /// <summary>
        /// Конструктор с параметрами
        /// </summary>
        /// <param name="koefIncome">Коэффициент, учитывающий доход</param>
        /// <param name="koefTerm">Коэффициент, учитывающий срок кредитования</param>
        /// <param name="koefDebt">Коэффициент, учитывающий общий размер долга</param>
        /// <param name="percent">Процент</param>
        internal Credit(double koefIncome, double koefTerm, double koefDebt, double percent)
        {
            this.koefIncome = koefIncome;
            this.koefTerm = koefTerm;
            this.koefDebt = koefDebt;
            this.percent = percent;
        }

        /// <summary>
        /// Рассчитать максимальный размер кредита
        /// </summary>
        /// <param name="income">Доход, руб.</param>
        /// <param name="term">Срок, мес.</param>
        /// <param name="card">Наличие банковской карты</param>
        /// <param name="pawnKoef">Коэф., учитывающий вид залога</param>
        /// <param name="debt">Задолженность по другим кредитам, руб.</param>
        /// <returns>Максимальный размер кредита</returns>
        internal abstract double GetMaxCredit(int income, int term, bool card,
            double pawnKoef, int debt);
    }
```

Листинг 5.9. Исходный код класса **ConsumeCredit**

```
    /// <summary>
    /// Представляет потребительские кредиты
    /// </summary>
    internal class ConsumeCredit : Credit
    {
        /// <summary>
        /// Параметрический конструктор
        /// </summary>
        /// <param name="percent">Процент</param>
        internal ConsumeCredit(double percent)
            : base(28.5, 9.5, 0.1, percent) {}

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        internal ConsumeCredit() :
            this(16.5) { }

        /// <summary>
        /// Рассчитать максимальный размер кредита
        /// </summary>
        /// <param name="income">Доход, руб.</param>
        /// <param name="term">Срок, мес.</param>
        /// <param name="card">Наличие банковской карты</param>
        /// <param name="pawnKoef">Коэф., учитывающий вид залога</param>
        /// <param name="debt">Задолженность по другим кредитам, руб.</param>
        /// <returns>Максимальный размер кредита</returns>
        internal override double GetMaxCredit(int income, int term, bool card,
            double pawnKoef, int debt)
        {
            // Процент при отсутствии банковской карты
            const double noCardPercent = 19.5;
            if (card == false) percent = noCardPercent;

            double maxCredit = (1 + pawnKoef - percent / 100) * (koefIncome * income +
                koefTerm * term - koefDebt * debt);

            return maxCredit;
        }
    }
```

Листинг 5.10. Исходный код класса **CreditFacade** (часть 1)

```
10      /// <summary>
11      /// Представляет фасадные объекты
12      /// </summary>
13      public static class CreditFacade
14      {
15          static ConsumeCredit conCredit = new ConsumeCredit();
16          static EstateCredit estCredit = new EstateCredit();
17          static CarCredit carCredit = new CarCredit();
18
19          /// <summary>
20          /// Получить максимальный размер потребительского кредита
21          /// </summary>
22          /// <param name="income">Ежемесячный доход, руб.</param>
23          /// <param name="term">Срок кредитования, мес.</param>
24          /// <param name="card">Наличие банковской карты</param>
25          /// <param name="pawn">Вид имущества под залог</param>
26          /// <param name="debt">Задолженность по другим кредитам, руб.</param>
27          /// <returns>Максимальный размер потребительского кредита, руб.</returns>
28          public static double GetMaxConsumeCredit(int income, int term,
29              bool card, string pawn, int debt)
30          {
31              double pawnKoef = 0;
32              switch (pawn)
33              {
34                  case "квартира":   pawnKoef = 0.03; break;
35                  case "участок":    pawnKoef = 0.02; break;
36                  case "автомобиль": pawnKoef = 0.01; break;
37                  default:           pawnKoef = 0; break;
38              }
39              return conCredit.GetMaxCredit(income, term, card, pawnKoef, debt);
40          }
41
42          /// <summary>
43          /// Получить максимальный размер ипотечного кредита
44          /// </summary>
45          /// <param name="income">Ежемесячный доход, руб.</param>
46          /// <param name="term">Срок кредитования, мес.</param>
47          /// <param name="card">Наличие банковской карты</param>
48          /// <param name="pawn">Вид имущества под залог</param>
49          /// <param name="debt">Задолженность по другим кредитам, руб.</param>
50          /// <returns>Максимальный размер ипотечного кредита, руб.</returns>
51          public static double GetMaxEstateCredit(int income, int term,
52              bool card, string pawn, int debt)
53          {
54              double pawnKoef = 0;
55              switch (pawn)
56              {
57                  case "участок":    pawnKoef = 0.02; break;
58                  case "автомобиль": pawnKoef = 0.01; break;
59                  default:           pawnKoef = 0; break;
60              }
61              return estCredit.GetMaxCredit(income, term, card, pawnKoef, debt);
62          }
```

Листинг 5.10. Исходный код класса **CreditFacade** (часть 2)

```
        /// <summary>
        /// Получить максимальный размер автокредита
        /// </summary>
        /// <param name="income">Ежемесячный доход, руб.</param>
        /// <param name="term">Срок кредитования, мес.</param>
        /// <param name="card">Наличие банковской карты</param>
        /// <param name="pawn">Вид имущества под залог</param>
        /// <param name="debt">Задолженность по другим кредитам, руб.</param>
        /// <returns>Максимальный размер автокредита, руб.</returns>
        public static double GetMaxCarCredit(int income, int term,
            bool card, string pawn, int debt)
        {
            double pawnKoef = 0;
            switch (pawn)
            {
                case "квартира": pawnKoef = 0.03; break;
                case "участок":  pawnKoef = 0.02; break;
                default:         pawnKoef = 0; break;
            }
            return carCredit.GetMaxCredit(income, term, card, pawnKoef, debt);
        }
    }
```

Исходный код класса **Program** консольного приложения представлен в листинге 5.11.

Листинг 5.11. Исходный код класса **Program** консольного приложения (клиент)

```
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Работа с шаблоном Фасад";

            Console.WriteLine("Вычисление максимального размера кредита\n");

            Console.WriteLine("* Потребительский кредит: {0:f2} руб.\n",
                CreditFacade.GetMaxConsumeCredit(35000, 12, true, "квартира", 0));

            Console.WriteLine("* Ипотечный кредит: {0:f2} руб.\n",
                CreditFacade.GetMaxEstateCredit(40000, 60, true, "автомобиль", 0));

            Console.WriteLine("* Автокредит: {0:f2} руб.\n",
                CreditFacade.GetMaxEstateCredit(25000, 24, true, "участок", 0));

            Console.Read();
        }

```

Результат работы консольного приложения показан на рис. 5.6. ❒

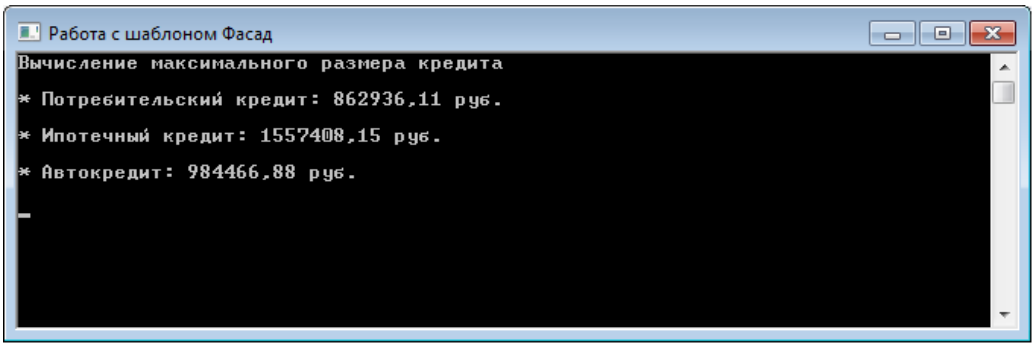


Рис. 5.6. Результат работы приложения

5.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать библиотеку классов, которая содержит указанные классы (табл. 5.1), задействованные в шаблоне Адаптер. Для адаптера объектов атрибуты класса должны быть реализованы как автоматические свойства, а для шаблона Адаптер классов – как защищённые поля.
3. Разработать библиотеку классов, которая должна содержать класс-фасад и заданный набор классов (табл. 5.2). В фасаде необходимо задать ссылки на другие классы библиотеки.
4. Добавить в решение консольное приложение, которое для реализованных шаблонов играет роль клиента. Продемонстрировать работу в консольном приложении работу шаблонов проектирования.
5. Оформить и защитить отчет по лабораторной работе.

Таблица 5.1

Варианты заданий для разработки приложения с использованием шаблона Адаптер

№ вар.	**Тип адаптера**	**Требуемый интерфейс**	**Адаптируемый класс**
1, 13	Объектов	**+ CalculateDp(T0 : int, dT : int) : double** – Определить изменение давления при заданной	**Баллон с газом.** ***Атрибуты***: • **Volume : double** –

№ вар.	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
		начальной температуре **T0** и изменении температуры **dT**; **+ ModifMass(dm : double) : void** – изменить массу газа в баллоне на величину **dm**; **+ GetData() : string** – возвращает строку с данными об объекте	объём баллона, м^3; • **Mass : double** – масса газа, кг; • **Molar : double** – молярная масса газа, кг/моль. ***Операции***: **+ GetPressure(T : int) : double** – определить давление в баллоне при заданной температуре газа **T**; **+ AmountOfMatter() : double** – определить количество вещества; **+ ToString() : string** – возвращает строку с данными об объекте.
2, 14	Классов	**+ ModifVolume(dV : double) : void** – изменить объём баллона на величину **dV**; **+ GetDp(T0 : int, T1 : int) : double** – Определить изменение давления при изменении температуры с **T0** до **T1**; **+ Passport() : string** – возвращает строку с данными об объекте.	
3, 15	Объектов	**+ ModifLength(dl : double) : void** – изменить длину подвеса маятника на величину **dl**; **+ GetFreq() : double** – Определить частоту колебаний; **+ Passport() : string** – возвращает строку с данными об объекте	**Математический маятник** ***Атрибуты***: • **a : double** – амплитуда колебаний; • **d : double** – длина нерастяжимой нити; • **m : int** – масса маятника; ***Операции***: **+ X(t : int): double** – определить смещение маятника в момент времени **t**; **+ CalculateT() : double** – Определить период колебаний; **+ ToString() : string** – возвращает строку с данными об объекте
4, 16	Классов	**+ ModifMass(dm : int) : void** – изменить массу маятника на величину **dm**; **+ CalculateW() : double** – определить циклическую частоту колебаний; **+ GetData() : string** – возвращает строку с данными об объекте	
5, 17	Объектов	**+ Translate(double dx, double dy) : void** – переместить объект на величину **dx** по оси X и на **dy**	**Астероид** ***Атрибуты***: • **m : double** – масса;

№ вар.	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
		по оси Y; **+ GetGravForce(int m, double r, double a) : double** – определить силу гравитационного взаимодействия с объектом массы **m** и полярными координатами **r** и **a**; **+ Passport() : string** – возвращает строку с данными об объекте.	• **x : double** – координата X; • **y : double** – координата Y. ***Операции***: **+ Distance(x : double, y : double) : double** – определить расстояние от астероида до объекта с координатами **x** и **y**; **+ Fg(m : int, r : double) : double** – определить силу взаимодействия с объектом массы **m**, расположенного на расстоянии **r** от данного объекта; **+ ToString() : string** – возвращает строку с данными об объекте.
6, 18	Классов	**+ Move(double dr, double a) : void** – переместить объект на расстояние **dr** в направлении, задаваемом углом **a**; **+ CalculateFgrav(int m, double x, double y) : double** – определить силу гравитационного взаимодействия с объектом массы **m** и координатами **x** и **y**; **+ GetData() : string** – возвращает строку с данными об объекте.	
7, 19	Объектов	**+ CalculateW(u : int) : double** – определить электрическую энергию конденсатора при известном напряжении **u** между обкладками; **+ ModifS(dS : double) : void** – изменить площадь обкладок конденсатора на величину **dS**; **+ GetData() : string** – возвращает строку с данными об объекте.	**Плоский конденсатор** ***Атрибуты***: • **s : double** – площадь обкладки конденсатора; • **d : double** – расстояние между обкладками; • **eps : double** – диэлектрическая проницаемость среды между обкладками. ***Операции***: **+ Capacity() : double** – определить ёмкость конденсатора; **+ GetCharge(u : int) : double** – определить
8, 20	Классов	**+ GetW(u : int) : double** – определить электрическую энергию конденсатора при известном напряжении **u** между обкладками; **+ ModifD(dD : double) : void** – изменить расстояние между обкладками конденсатора на величину **dD**;	

№ вар.	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
		+ Passport() : string – возвращает строку с данными об объекте.	заряд на обкладках конденсатора при известном напряжении **u**; **+ ToString() : string** – возвращает строку с данными об объекте.
9, 21	Объектов	**+ CalculateW() : double** – определить циклическую частоту колебаний; **+ ModifA(dA : double) : void** – изменить амплитуду колебаний на величину **dA**; **+ Passport() : string** – возвращает строку с данными об объекте.	**Пружинный маятник** ***Атрибуты***: • **a : double** – амплитуда колебаний; • **k : double** – жёсткость пружины; • **m : int** – масса маятника. ***Операции***: **+ ElasticForce(t : int) : double** – определить силу упругости в момент времени **t**; **+ GetFreq() : double** – определить частоту колебаний; **+ ToString() : string** – возвращает строку с данными об объекте. смещение от положения равновесия
10, 22	Классов	**+ ModifMass(dM : double) : void** – изменить массу маятника на величину **dM**; **+ CalculateT() : double** – Определить период колебаний; **+ GetData() : string** – возвращает строку с данными об объекте.	
11, 23	Объектов	**+ Move(dx : double, dy : double) : void** – переместить объект на величину **dx** по оси X и на **dy** по оси Y; **+ CalculateForce(q : double, r : double, a : double) : double** – определить силу электростатического взаимодействия с точечным зарядом **q**, имеющим полярные координаты **r** и **a**; **+ GetData() : string** – возвращает строку с данными об объекте.	**Точечный электрический заряд** ***Атрибуты***: • **q : double** – заряд; • **x : double** – координата X; • **y : double** – координата Y. ***Операции***: **+ F(m : int, r : double) : double** – определить силу взаимодействия

№ вар.	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
12, 24	Классов	**+ Translate(double dr, double a) : void** – переместить объект на расстояние **dr** в направлении, задаваемом углом **a**; **+ GetElectrForce(int q, double x, double y) : double** – определить силу электростатического взаимодействия с точечным зарядом **q**, расположенном в координатах **x** и **y**; **+ Passport() : string** – возвращает строку с данными об объекте.	с объектом массы **m**, расположенного на расстоянии **r** от данного объекта; **+ R(double x, double y) : double** – определить расстояние от данного заряда до точки с координатами **x** и **y**; **+ ToString() : string** – возвращает строку с данными об объекте.

Таблица 5.2

Варианты заданий для разработки приложения с использованием шаблона Фасад

№ вар.	Данные для разработки приложения на основе шаблона Фасад
1, 7, 13, 19	***Расчёт страхового взноса за недвижимость.*** Классы (типы недвижимости): квартира, таун-хаус, коттедж. Параметры: срок страхования, жилплощадь ($м^2$), число проживающих, год постройки здания, износ здания (%).
2, 8, 14, 20	***Расчёт ежедневной нормы потребления килокалорий.*** Классы (Тип телосложения): Астеник, Нормостеник, Гиперстеник. Параметры: Рост, Вес, Возраст, Пол, Группа физической активности (низкая, средняя и высокая активность).
3, 9, 15, 21	***Расчёт стоимости туристической путевки:*** Классы (виды путевок): пляжный отдых, экскурсия, горные лыжи. Параметры: длительность, страна, гостиница (число звезд), рацион питания (двухразовый, трехразовый, всё включено).
4, 10, 16, 22	***Расчёт ОСАГО.*** Классы (типы автотранспортных средств): легковой автомобиль, грузовой автомобиль, автобус. Параметры: срок договора, лицо (физическое, юридическое), стаж водителя, регион.
5, 11, 17, 23	***Расчёт стоимости покупаемого легкового автомобиля***. Классы (марки автомобилей): марка1, марка2, марка3 (придумать самостоятельно).

	Параметры: год выпуска, объем двигателя, цвет, комплектация (минимальная, средняя, максимальная).
6, 12, 18, 24	***Расчёт срока строительства здания*.** Классы (типы зданий): жилое здание, общественное здание, промышленное здание. Параметры: строительный объем здания ($м^3$), занимаемая площадь ($м^2$), период строительства (лето, зима, весна-осень), геологические условия (простые, средние, сложные).

5.4. Контрольные вопросы

1. Какие шаблоны проектирования называют структурными?
2. Каково назначение структурного шаблона Адаптер?
3. Каким образом осуществляется взаимодействие клиента с адаптируемым классом в шаблоне Адаптер?
4. В чём заключается различие между Адаптером объектов и Адаптером классов?
5. Какая проблема решается с помощью шаблона Фасад?
6. В чём заключается решение, предлагаемое в шаблоне Фасад?
7. К каким последствиям приводит использование шаблона Фасад?

6. РАБОТА С ПОВЕДЕНЧЕСКИМИ ШАБЛОНАМИ GOF НА ЯЗЫКЕ C#

6.1. Цель и задачи работы

Цель работы – прибрести умение разрабатывать приложения на языке C# с использованием таких поведенческих шаблонов проектирования, как Стратегия, Состояние и Шаблонный метод.

Основные задачи:

• Освоить такие поведенческие шаблоны проектирования, как Стратегия, Состояние и Шаблонный метод.

• Научиться использовать поведенческие шаблоны GoF в приложениях на языке C#.

Работа рассчитана на 6 часов.

6.2. Основные теоретические сведения

6.2.2. Поведенческие шаблоны проектирования. Диаграммы конечных автоматов. Шаблон Состояние

Поведенческие шаблоны проектирования. Диаграммы конечных автоматов UML.

Поведенческие шаблоны (*англ. Behavioral Patterns*) предназначены для определения алгоритмов и способов реализации взаимодействия различных объектов и классов.

К поведенческим шаблонам проектирования из каталога GoF относятся:

- **Observer** – Наблюдатель.
- **Iterator** – Итератор.
- **State** – Состояние.
- **Strategy** – Стратегия.
- **Template method** – Шаблонный метод.
- **Command** – Команда.
- **Chain of responsibility** – цепочка обязанностей и др.

Одним из средств описания поведения программных объектов в языке UML являются диаграммы конечных автоматов (диаграммы состояний).

Диаграммы конечных автоматов (англ. *state machine diagrams*) UML отображают жизненный цикл объекта с помощью состояний, событий и переходов.

Основными элементами диаграмм конечных автоматов являются:

• ***Состояние*** – это ситуация во время жизни объекта, в которой он удовлетворяет определённым условиям, выполняет определённую деятельность или находится в ожидании событий. Состояния изображаются на диаграмме в виде прямоугольников со скруглёнными углами. Специальными видами состояний являются начальное и конечное состояния, изображаемые соответственно символами ● и ◉.

• ***Переход*** – это отношение между двумя состояниями, указывающее на то, что объект из первого состояния перейдёт во второе, при выполнении определённого условия. Переходы на диаграммах конечных автоматов изображают стрелками, ведущими от одного состояния к другому.

• ***Событие*** – это значимое или заслуживающее внимания происшествие, которое может инициализировать переход объекта от одного состояния к другому. Событием может являться поступление сигнала, выполнение какого-либо условия, истечение определённого периода времени. События разделяют на внешние, внутренние и временные.

Пример простой диаграммы конечных автоматов для объекта «входная дверь» показан на рис. 6.1.

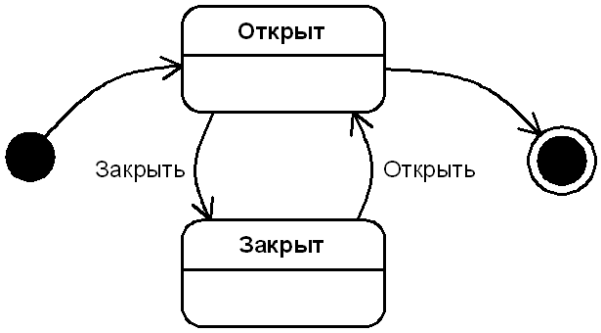


Рис. 6.1. Простая диаграмма конечных автоматов UML

Шаблон Состояние.

Шаблон ***Состояние*** (англ. *State*) управляет изменением поведения объекта в зависимости от его внутреннего состояния.

Проблема. Как изменять поведение объекта в зависимости от его внутреннего состояния?

Решение. Определить для каждого состояния отдельный класс со стандартным интерфейсом.

Структура. Диаграмма классов шаблона Состояние представлена на рис. 6.2.

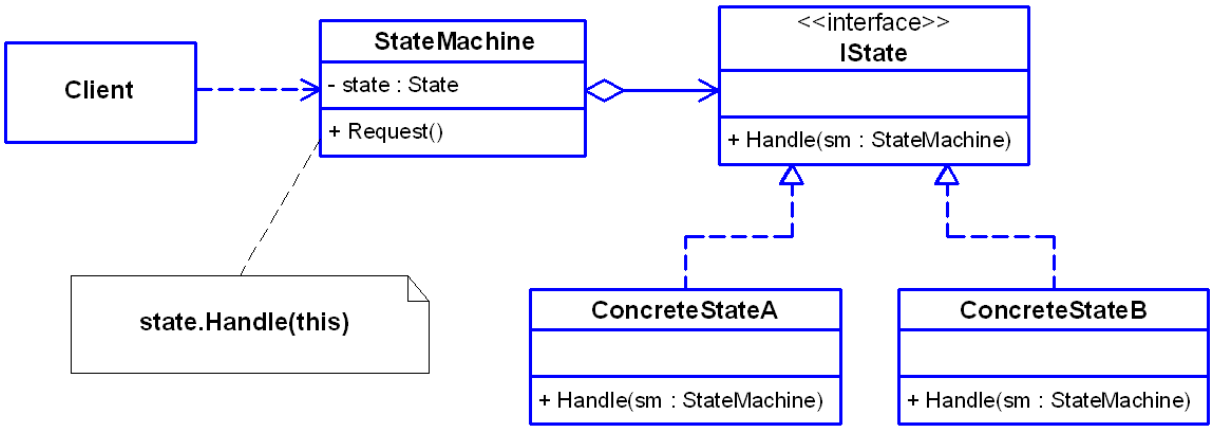


Рис. 6.2. Диаграмма классов шаблона Состояние

Участники шаблона:

- **State** – абстрактный класс, определяющий общий интерфейс для всех конкретных состояний.
- **StateMachine** – класс с несколькими внутренними состояниями, хранит экземпляр класса **State.**
- **ConcreteStateA**, **ConcreteStateB** – классы конкретных состояний, обрабатывающие запросы от класса **StateMachine**; каждый класс предоставляет собственную реализацию обработки запроса.

Реализация. Простой пример реализации шаблона Состояние на языке C# приведён в листинге 6.1.

Листинг 6.1. Пример реализации шаблона Состояние на языке C#

```
    // Представляет состояния
```

```
interface IState
{
    void Handle(StateMachine sm);
}

// Представляет конкретные состояния A
class ConcreteStateA : IState
{
    public void Handle(StateMachine sm)
    {
        Console.Write("Объект {0} в сост. A", sm.ToString());
    }
}

// Представляет конкретные состояния B
class ConcreteStateB : IState
{
    public void Handle(StateMachine sm)
    {
        Console.Write("Объект {0} в сост. B", sm.ToString());
    }
}

// Представляет конечные автоматы
class StateMachine
{
    public IState State { get; set; }

    public void Request()
    {
        State.Handle(this);
    }
}

// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        StateMachine sm = new StateMachine();
        sm.State = new ConcreteStateA();
        sm.Request();
        sm.State = new ConcreteStateB();
        sm.Request();
    }
}
```

❒ ***Пример* 6.1. *Реализация шаблона Состояние в приложении на языке C#.***

Требуется разработать приложение, в котором моделируется работа конечного автомата, в качестве которого выступает навесной замок. Примем, что объект «навесной замок» может находиться в следующих состояниях: Открыт, Закрыт, Неисправен.

Основными операциями объекта «навесной замок» являются:

Закрыть – переводит объект из состояния «Закрыт» в состояние «Открыт»;

Открыть – переводит объект из состояния «Закрыт» в состояние «Открыт»;

Ремонтировать – переводит объект из состояния «Неисправен» в состояние «Открыт».

Переход объекта в состояние «Неисправен» может происходить при вызове операций **Открыть** и **Закрыть** случайным образом с вероятностью **p**.

Диаграмма конечных автоматов указанного объекта показана на рис. 6.3.

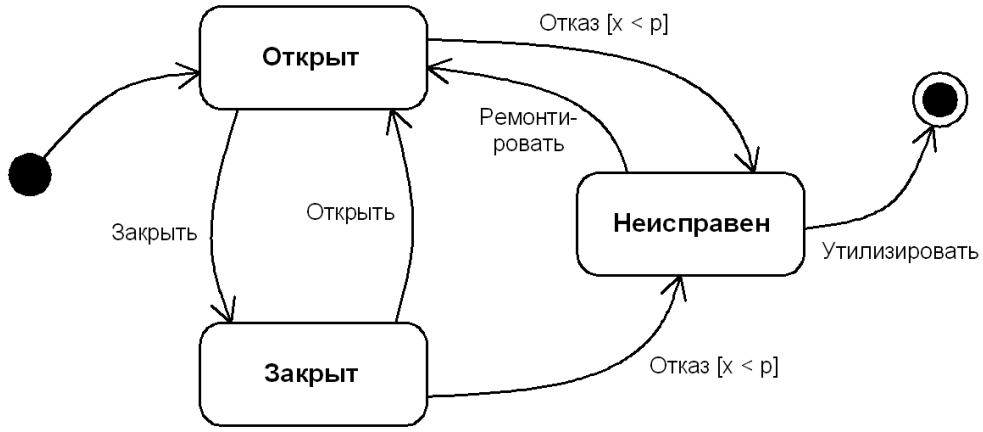


Рис. 6.3. Диаграмма конечных автоматов для объекта Навесной замок

Возможные реакции объекта на выполнение операций в различных состояниях представлены в табл. 6.1.

Таблица 6.1.

Реакции объекта «навесной замок» на вызов операций в различных состояниях

Операции	**Состояния**		
	Открыт	**Закрыт**	**Неисправен**
Закрыть	Переход в состояние «Закрыт»	Исключение «Замок уже закрыт»	Исключение «Замок неисправен»
Открыть	Исключение «Замок уже открыт»	Переход в состояние «Открыт»	Исключение «Замок неисправен»
Ремонтировать	Исключение «Замок не требует ремонта»	Исключение «Замок не требует ремонта»	Переход в состояние «Открыт»

Класс **Padlock** (навесной замок) разместим в проекте библиотеки классов **StatePatternLib** вместе с интерфейсом **IState** и классами состояний (**LockedState** – закрыт, **UnlockedState** – открыт, **UnfixedState** – неисправен). Диаграмма классов для указанного проекта показана на рис. 6.4.

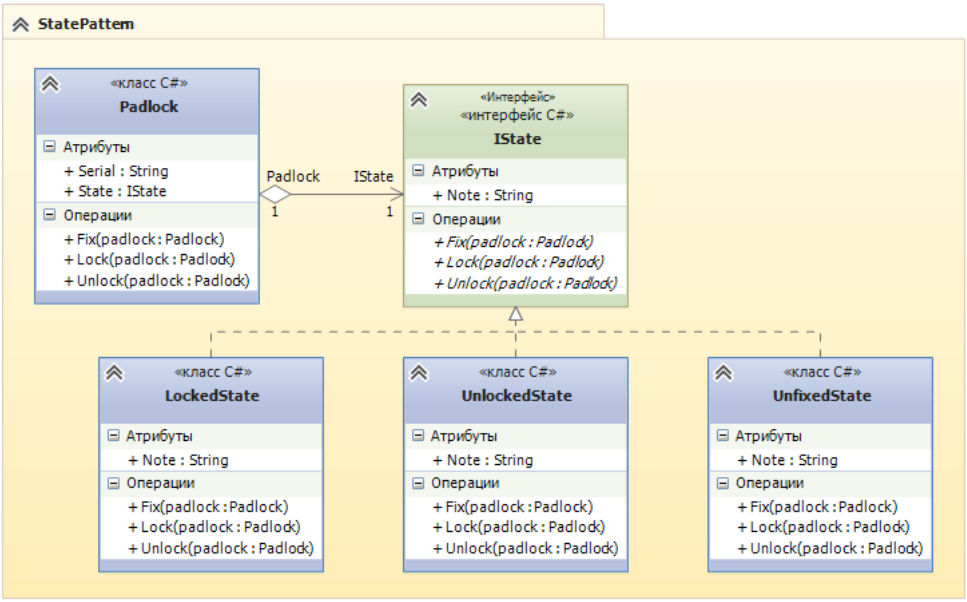


Рис. 6.4. Диаграмма классов проекта **StatePatternLib**

Исходный код класса **Padlock** представлен в листинге 6.2.

Листинг 6.2. Исходный код класса **Padlock** (часть 1)

```
 9      /// <summary>
10      /// Представляет навесные замки (Конечный автомат)
11      /// </summary>
12      public class Padlock
13      {
14          /// <summary>
15          /// Возвращает или задаёт серийный номер
16          /// </summary>
17          public string Serial { get; set; }
18
19          /// <summary>
20          /// Возвращает или задаёт вероятность поломки
21          /// </summary>
22          public double UnfixProb { get; set; }
23
24          /// <summary>
25          /// Текущее состояние
26          /// </summary>
27          public IState State { get; set; }
28
29          Random rnd;
30
31          /// <summary>
32          /// Параметрический конструктор
33          /// </summary>
34          /// <param name="serial">Серийный номер</param>
35          /// <param name="p">Вероятность поломки</param>
36          public Padlock(string serial, double p)
37          {
38              Serial = serial;
39              UnfixProb = p;
40              State = new UnlockedState();
41              rnd = new Random();
42          }
43
44          /// <summary>
45          /// Моделирует случайную поломку при открывании/закрывании
46          /// </summary>
47          public void ModelUnfix()
48          {
49              if (rnd.NextDouble() < UnfixProb)
50              {
51                  State = new UnfixedState();
52                  throw new Exception(string.Format("Замок {0} сломался", Serial));
53              }
54          }
55
```

Листинг 6.2. Исходный код класса **Padlock** (часть 2)

```
56          /// <summary>
57          /// Закрыть замок
58          /// </summary>
59          public void Lock()
60          {
61              ModelUnfix();
62              State.Lock(this);
63          }
64
65          /// <summary>
66          /// Открыть замок
67          /// </summary>
68          public void Unlock()
69          {
70              ModelUnfix();
71              State.Unlock(this);
72          }
73
74          /// <summary>
75          /// Отремонтировать замок
76          /// </summary>
77          public void Fix()
78          {
79              State.Fix(this);
80          }
81
82          /// <summary>
83          /// Возвращает строку с данными об объекте
84          /// </summary>
85          /// <returns>Строка с данными об объекте</returns>
86          public override string ToString()
87          {
88              return string.Format("Данные о замке:\n" +
89                                   "Серийный номер: {0}\n" +
90                                   "Вероятность поломки: {1}\n" +
91                                   "Текущее состояние: {2}\n",
92                                   Serial, UnfixProb, State.Note);
93          }
94      }
```

Исходный код интерфейса **IState** представлен в листинге 6.3. Для классов конкретных состояний, реализующих интерфейс **IState**, исходные коды приведены в листингах 6.4, 6.5 и 6.6.

Листинг 6.3. Исходный код интерфейса **IState**

```
9   /// <summary>
10  /// Представляет состояния объекта Навесной замок
11  /// </summary>
12  public interface IState
13  {
14      /// <summary>
15      /// Возвращает строку с описанием состояния
16      /// </summary>
17      string Note { get; }
18
19      /// <summary>
20      /// Закрыть замок
21      /// </summary>
22      /// <param name="padlock">Навесной замок</param>
23      void Lock(Padlock padlock);
24
25      /// <summary>
26      /// Открыть замок
27      /// </summary>
28      /// <param name="padlock">Навесной замок</param>
29      void Unlock(Padlock padlock);
30
31      /// <summary>
32      /// Отремонтировать замок
33      /// </summary>
34      /// <param name="padlock">Навесной замок</param>
35      void Fix(Padlock padlock);
36  }
```

Листинг 6.4. Исходный код класса **LockedState**

```
    /// <summary>
    /// Представляет состояние "Закрыт" объекта Навесной замок
    /// </summary>
    public class LockedState : IState
    {
        string note; // Строка с описанием состояния

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public LockedState()
        {
            note = "Закрыт";
        }

        /// <summary>
        /// Возвращает строку с описанием состояния
        /// </summary>
        public string Note
        {
            get { return note; }
        }

        /// <summary>
        /// Закрыть замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Lock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} уже закрыт!",
                padlock.Serial));
        }

        /// <summary>
        /// Открыть замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Unlock(Padlock padlock)
        {
            padlock.State = new UnlockedState();
        }

        /// <summary>
        /// Отемонтировать замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Fix(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} не нуждается в ремонте!",
                padlock.Serial));
        }
    }
```

Листинг 6.5. Исходный код класса **UnlockedState**

```
9   /// <summary>
10  /// Представляет состояние "Открыт" объекта Навесной замок
11  /// </summary>
12  public class UnlockedState : IState
13  {
14      string note; // Строка с описанием состояния
15
16      /// <summary>
17      /// Конструктор по умолчанию
18      /// </summary>
19      public UnlockedState()
20      {
21          note = "Открыт";
22      }
23
24      /// <summary>
25      /// Возвращает строку с описанием состояния
26      /// </summary>
27      public string Note
28      {
29          get { return note; }
30      }
31
32      /// <summary>
33      /// Закрыть замок
34      /// </summary>
35      /// <param name="padlock">Навесной замок</param>
36      public void Lock(Padlock padlock)
37      {
38          padlock.State = new LockedState();
39      }
40
41      /// <summary>
42      /// Открыть замок
43      /// </summary>
44      /// <param name="padlock">Навесной замок</param>
45      public void Unlock(Padlock padlock)
46      {
47          throw new Exception(string.Format("Замок {0} уже открыт!",
48              padlock.Serial));
49      }
50
51      /// <summary>
52      /// Отремонтировать замок
53      /// </summary>
54      /// <param name="padlock">Навесной замок</param>
55      public void Fix(Padlock padlock)
56      {
57          throw new Exception(string.Format("Замок {0} не нуждается в ремонте!",
58              padlock.Serial));
59      }
60  }
```

Листинг 6.6. Исходный код класса **UnfixedState**

```
    /// <summary>
    /// Представляет состояние "Неисправен" объекта Навесной замок
    /// </summary>
    class UnfixedState : IState
    {
        string note; // Строка с описанием состояния

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public UnfixedState()
        {
            note = "Неисправен";
        }

        /// <summary>
        /// Возвращает описание состояния
        /// </summary>
        public string Note
        {
            get { return note; }
        }

        //// <summary>
        /// Закрыть замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Lock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} не исправен!",
                padlock.Serial));
        }

        /// <summary>
        /// Открыть замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Unlock(Padlock padlock)
        {
            throw new Exception(string.Format("Замок {0} не исправен!",
                padlock.Serial));
        }

        /// <summary>
        /// Отремонтировать замок
        /// </summary>
        /// <param name="padlock">Навесной замок</param>
        public void Fix(Padlock padlock)
        {
            padlock.State = new UnlockedState();
        }
    }
```

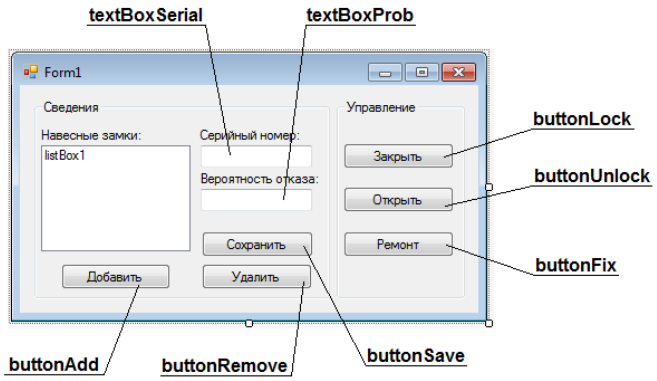


Рис. 6.5. Интерфейс пользователя приложения

Листинг 6.7. Исходный код класса **Form1** (часть 1)

```
14  public partial class Form1 : Form
15  {
16      List<Padlock> padlocks; // Список навесных замков
17
18      public Form1()
19      {
20          InitializeComponent();
21          padlocks = new List<Padlock>();
22          padlocks.Add(new Padlock("001234", 0.1));
23      }
24
25      /// <summary>
26      /// Заполнить список listBox1
27      /// </summary>
28      private void GenerPadlocksList()
29      {
30          listBox1.Items.Clear();
31          foreach (Padlock p in padlocks)
32          {
33              listBox1.Items.Add(string.Format("{0} - {1}",
34                  p.Serial, p.State.Note));
35          }
36      }
37
38      private void Form1_Load(object sender, EventArgs e)
39      {
40          this.Text = "Управление состоянием объектов " +
41              "(Турчин Д.Е., каф. ИиАПС)";
42          this.Font = new Font("Arial", 10, FontStyle.Bold);
43          textBoxSerial.Text = "001234";
44          textBoxProb.Text = "0,1";
45          GenerPadlocksList();
46      }
47
48      //
49      private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
50      {
51          int index = listBox1.SelectedIndex;
52          if (index < 0) return;
53          textBoxSerial.Text = padlocks[index].Serial;
54          textBoxProb.Text = padlocks[index].UnfixProb.ToString();
55      }
56
```

Листинг 6.7. Исходный код класса **Form1** (часть 2)

```
57          private void buttonLock_Click(object sender, EventArgs e)
58          {
59              int index = listBox1.SelectedIndex;
60              if (index < 0) return;
61              try
62              {
63                  padlocks[index].Lock();
64              }
65              catch (Exception exc)
66              {
67                  MessageBox.Show(exc.Message);
68              }
69              GenerPadlocksList();
70          }
71
72          private void buttonUnlock_Click(object sender, EventArgs e)
73          {
74              int index = listBox1.SelectedIndex;
75              if (index < 0) return;
76              try
77              {
78                  padlocks[index].Unlock();
79              }
80              catch (Exception exc)
81              {
82                  MessageBox.Show(exc.Message);
83              }
84              GenerPadlocksList();
85          }
86
87          private void buttonFix_Click(object sender, EventArgs e)
88          {
89              int index = listBox1.SelectedIndex;
90              if (index < 0) return;
91              try
92              {
93                  padlocks[index].Fix();
94              }
95              catch (Exception exc)
96              {
97                  MessageBox.Show(exc.Message);
98              }
99              GenerPadlocksList();
100         }
```

Листинг 6.7. Исходный код класса **Form1** (часть 3)

```
102         private void buttonAdd_Click(object sender, EventArgs e)
103         {
104             double p = 0;
105             try
106             {
107                 p = Convert.ToDouble(textBoxProb.Text);
108             }
109             catch (FormatException exc)
110             {
111                 MessageBox.Show(exc.Message);
112                 return;
113             }
114             padlocks.Add(new Padlock(textBoxSerial.Text, p));
115             GenerPadlocksList();
116         }
117 
118         private void buttonRemove_Click(object sender, EventArgs e)
119         {
120             int index = listBox1.SelectedIndex;
121             if (index < 0) return;
122             padlocks.RemoveAt(index);
123             GenerPadlocksList();
124         }
125 
126         private void buttonSave_Click(object sender, EventArgs e)
127         {
128             int index = listBox1.SelectedIndex;
129             if (index < 0) return;
130             double p = 0;
131             try
132             {
133                 p = Convert.ToDouble(textBoxProb.Text);
134             }
135             catch (FormatException exc)
136             {
137                 MessageBox.Show(exc.Message);
138                 return;
139             }
140             padlocks[index].Serial = textBoxSerial.Text;
141             padlocks[index].UnfixProb = p;
142         }
143     }
```

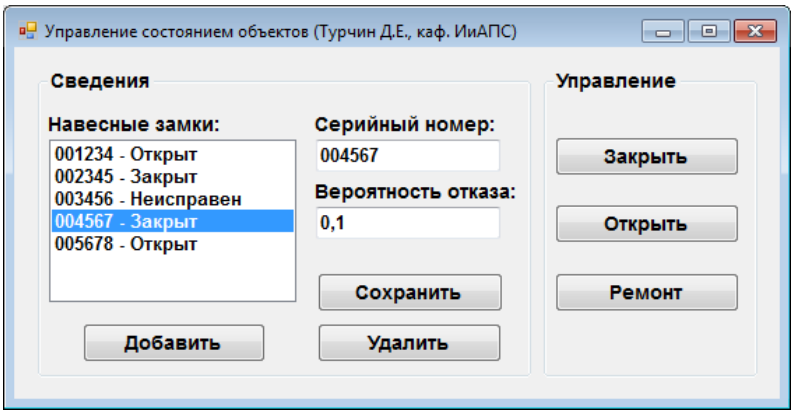


Рис. 6.6. Главное окно приложения

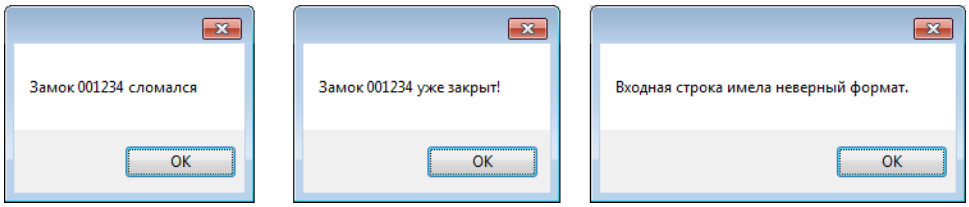


Рис. 6.7. Окна сообщений

6.2.1. Шаблоны Стратегия и Шаблонный метод

Шаблон Стратегия.

Шаблон ***Стратегия*** (англ., *Strategy* или *Policy*) позволяет менять алгоритм независимо от клиентов, которые его используют.

Проблема.

Как спроектировать изменяемые, но надёжные алгоритмы (стратегии)?

Решение.

Определить для каждого алгоритма (стратегии) отдельный класс со стандартным интерфейсом.

Структура.

Диаграмма классов шаблона Стратегия показана на рис. 6.8.

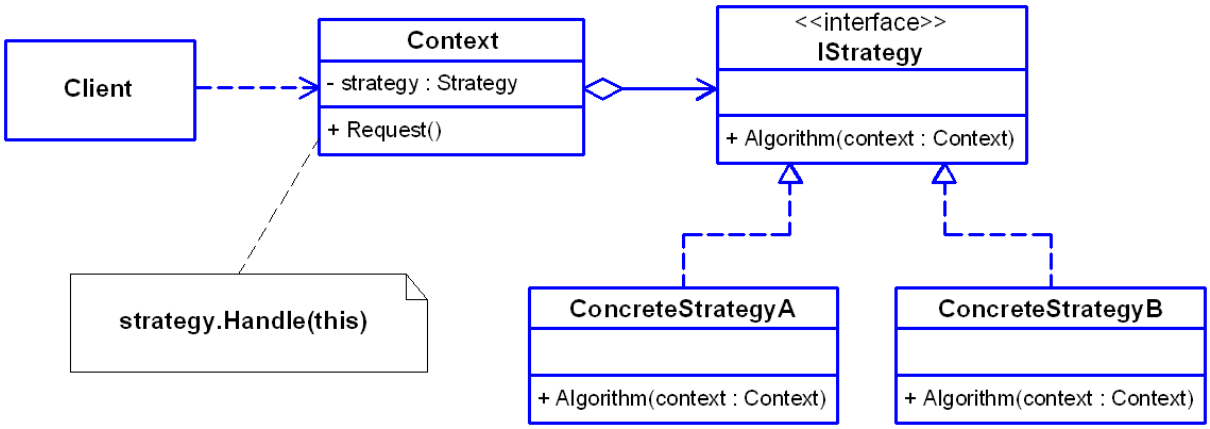


Рис. 6.8. Диаграмма классов шаблона Стратегия

Участники шаблона:

• **Strategy** – класс, определяющий общий для всех поддерживаемых алгоритмов интерфейс;

• **Context** – класс, хранящий ссылку на экземпляр класса **Стратегия**;

• **ConcreteStrategyA**, **ConcreteStrategyB** – классы, представляющие конкретные стратегии, использующие интерфейс класса **Strategy** для реализации алгоритма.

Реализация. Простой пример реализации шаблона Стратегия на языке C# приведён в листинге 6.8.

Листинг 6.8. Пример реализации шаблона Стратегия на языке C#

```
// Представляет стратегии, реализуемые классами
public interface IStrategy
{
    void Algorithm();
}

// Представляет конкретные стратегии A
public class ConcreteStrategyA : IStrategy
{
    public void Algorithm()
    {
        Console.WriteLine("Алгоритм стратегии A");
    }
}

// Представляет конкретные стратегии B
public class ConcreteStrategyB : IStrategy
{
    public void Algorithm()
    {
        Console.WriteLine("Алгоритм стратегии B");
    }
}

// Представляет контексты, использующие стратегии
public class Context
{
    public IStrategy Strategy { get; set; }

    public void Request()
    {
        Strategy.Algorithm();
    }
}
```

```
// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        Context context = new Context();
        context.Strategy = new ConcreteStrategyA();
        context.Request();
        context.Strategy = new ConcreteStrategyB();
        context.Request();
    }
}
```

Шаблонный метод.

Шаблон ***Шаблонный метод*** (англ., *Template Method*) определяет основу алгоритма (шаблонный метод) в базовом классе и позволяет производным классам переопределять отдельные шаги алгоритма, не изменяя его структуру в целом.

Проблема. Как определить алгоритм и реализовать возможность переопределения некоторых шагов алгоритма в производных классах без изменения общей структуры алгоритма?

Решение. Создать абстрактный класс, определяющий следующие операции:

- шаблонный метод, который задаёт основу алгоритма;
- абстрактные операции, которые замещаются в производных классах для реализации шагов алгоритма.

Структура. Диаграмма классов шаблона Шаблонный метод показана на рис. 6.9.

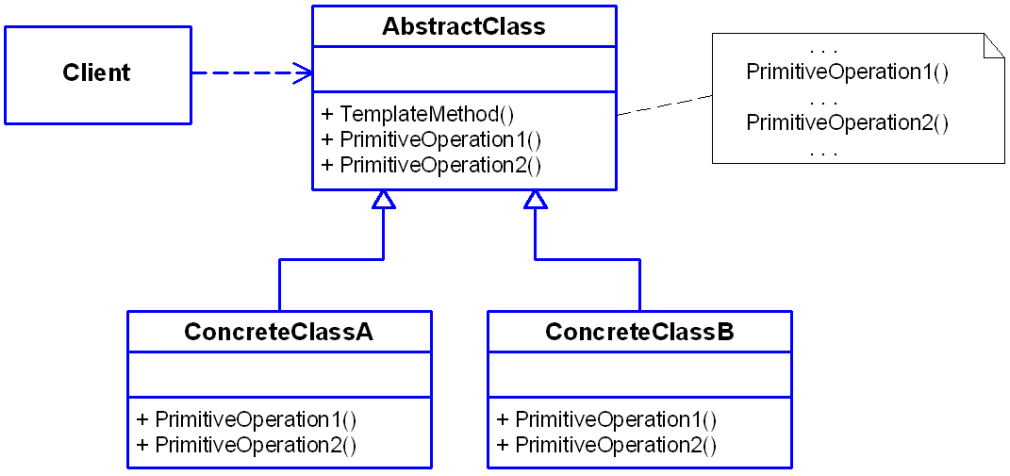


Рис. 6.9. Диаграмма классов шаблона Шаблонный метод

Участники шаблона:

• **AbstractClass** определяет абстрактные частные операции и содержит шаблонный метод, который вызывает эти операции;

• **ConcreteClassA**, **ConcreteClassB** реализуют частные операции, вызываемые шаблонным методом.

Реализация. Простой пример реализации шаблона Шаблонный метод на языке C# приведён в листинге 6.9.

Листинг 6.9. Пример реализации шаблона Шаблонный метод на языке C#

```
// Представляет Абстрактный класс
public abstract class AbstractClass
{
    int a = 10;

    // Шаблонный метод
    public int TemplateMethod(int x)
    {
        int y = 3;
        if (x > a) y += PartMethod1(x);
        else y += PartMethod2(x);
        y -= a;
        return y;
    }

    public abstract int PartMethod1(int x);

    public abstract int PartMethod2(int x);
}

// Представляет Конкретный класс A
public class ConcreteClassA : AbstractClass
{
    int k = 5;

    public override int PartMethod1(int x)
    { return k + x; }

    public override int PartMethod2(int x)
    { return k * x; }
}

// Представляет Конкретный класс B
```

```
    public class ConcreteClassB : AbstractClass
    {
        int k = 3;

        public override int PartMethod1(int x)
        { return x * x - k; }

        public override int PartMethod2(int x)
        { return x + x * k; }
    }

    // Представляет клиентов
    class Client
    {
        static void Main(string[] args)
        {
            AbstractClass a = new ConcreteClassA();
            AbstractClass b = new ConcreteClassB();
            int x = 20;
            Console.WriteLine(a.TemplateMethod(x));
            Console.WriteLine(b.TemplateMethod(x));
        }
    }
```

6.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать диаграмму конечных автоматов для заданного класса (табл. 6.1). Описать в форме таблицы варианты реакции экземпляра класса на операции, вызываемые в указанных состояниях.
3. Разработать библиотеку классов, включающую необходимые классы для реализации шаблона Состояние (класс Конечный автомат, интерфейс Состояние, классы Конкретные состояния).
4. Разработать приложение Windows Forms для управления состояниями экземпляров класса Конечный автомат.
5. Оформить и защитить отчет по лабораторной работе.

Таблица 6.1

Варианты заданий для разработки приложения с использованием шаблона Состояние

№ вар.	Классы, их атрибуты и операции	Состояния
1, 7, 13, 19	**Телефон.** ***Атрибуты***: номер, баланс, вероятность поступления звонка. ***Операции***: Позвонить, Ответить на звонок, Завершить разговор, Пополнить баланс.	Ожидание, Звонок, Разговор, Заблокирован (баланс отрицательный)
2, 8, 14, 20	**Банкомат.** ***Атрибуты***: ID, общая сумма денег в банкомате, вероятность отсутствия связи с банком. ***Операции***: Ввести PIN-код, Снять заданную сумму, Завершить работу, Загрузить деньги в банкомат.	Ожидание, Аутентификация пользователя, Выполнение операций, Заблокирован (денег нет)
3, 9, 15, 21	**Грузовой лифт.** ***Атрибуты***: текущий этаж, грузоподъёмность, вероятность отключения электроэнергии. ***Операции***: Вызвать на заданный этаж, Загрузить, Разгрузить, Восстановить подачу энергии.	Покой, Движение, Перегружен, Нет питания
4, 10, 16, 22	**Пулемёт.** ***Атрибуты***: скорострельность, число патронов в магазине, вероятность осечки. ***Операции***: Нажать курок, Отпустить курок, Перезарядить, Сменить ствол.	Готовность, Стрельба, Перегрев, Отсутствие патронов
5, 11, 17, 23	**Смеситель.** ***Атрибуты***: максимальный напор холодной (горячей) воды, вероятность отключения воды. ***Операции***: Повернуть кран с холодной (горячей) водой на заданный угол, Переключить воду на излив, переключить воду на лейку, Восстановить по-	Закрыт, Выливание воды через излив (носик), Выливание воды через душевую лейку, Нет воды.

	дачу воды.	
6, 12, 18, 24	**Принтер**. ***Атрибуты***: модель, число листов в лотке, количество краски в картридже, вероятность зажатия бумаги. ***Операции***: Печатать, Загрузить бумагу, Извлечь зажатую бумагу, Заправить картридж.	Ожидание, Печать документа, Зажатие бумаги, Отказ (отсутствует бумага или краска)

6.4. Контрольные вопросы

1. Каково назначение поведенческих шаблонов проектирования?
2. Какие основные элементы используются на диаграммах конечных автоматов UML?
3. Для чего предназначен поведенческий шаблон Состояние?
4. Какое решение предлагается в шаблоне Состояние?
5. Что понимают под стратегией в шаблоне Стратегия?
6. Какое решение предлагается в шаблоне Стратегия?
7. Какую проблему позволяет решить шаблон Шаблонный метод?

7. ОСНОВЫ СОЗДАНИЯ ЗАПРОСОВ К КОЛЛЕКЦИЯМ ОБЪЕКТОВ С ПОМОЩЬЮ LINQ

7.1. Цель и задачи работы

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

Основные задачи работы:

• ознакомиться с технологией LINQ;

• научиться создавать запросы LINQ к коллекциям объектов.

Работа рассчитана на 6 часов.

7.2. Основные теоретические сведения

7.2.1. 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 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 операцией **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 перем_запроса = from перем_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 зпросом данные по одному или нескольким критериям. Для этого используется операция **orderby**. Запрос LINQ при задании сортировки будет иметь следующий вид:

```
var перем_запроса = from перем_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 опера-

ция **ancesting**, которую можно не записывать при сортировке по возрастанию.

Операция **group** служит для группирования полученных данных по ключам. Данная операция, как и операция **select**, может завершать выражение запроса. Запрос с операцией **group** в простейшем случае может быть записан следующим образом:

```
var перем_запроса = from перем_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 перем_запроса = from перем_диап_A in источник_данных_A
                    join перем_диап_B in источник_данных_B
     on перем_диап_A.свойство equals перем_диап_B.свойство
                    select результат_запроса;
```

7.2.2. 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, ... }
```

Расширяющий метод представляет собой статический метод, который может быть связан с классом так, что он может быть вызван, как метод экземпляра этого класса. Расширяющие методы дают возможность использовать метод с классом, который не предоставляет его изначально.

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* 7.1. *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. Группы товаров размещаются на стеллажах (код, число ярусов, максимальная грузонесущая способность).

Исходный код на языке C# для класса **Product**, описывающего продуктовый товар, представлен в листинге 7.1, а код класса **Store** (стеллаж) – в листинге 7.2.

Листинг 7.1. Исходный код класса **Product**

```
    /// <summary>
    /// Представляет продуктовые товары
    /// </summary>
    public class Product
    {
        /// <summary>
        /// Наименование
        /// </summary>
        public string Name { get; set; }

        /// <summary>
        /// Производитель
        /// </summary>
        public string ProducedBy { get; set; }

        /// <summary>
        /// Число единиц
        /// </summary>
        public int NumberInStok { get; set; }

        /// <summary>
        /// Вес, г
        /// </summary>
        public int Weight { get; set; }

        /// <summary>
        /// Цена, руб
        /// </summary>
        public int Price { get; set; }

        /// <summary>
        /// Код стеллажа
        /// </summary>
        public string StoreID { get; set; }

        /// <summary>
        /// Возвращает строку с данными о продукте
        /// </summary>
        /// <returns>Строка с данными о продукте</returns>
        public override string ToString()
        {
            return string.Format("Наименование: {0}\n" +
                "- Производитель: {1}\n" +
                "- Количество: {2} шт\n" +
                "- Вес: {3} г\n" +
                "- Цена: {4} руб\n" +
                "- Код стеллажа: {5}", Name, ProducedBy, NumberInStok,
                Weight, Price, StoreID);
        }
    }
```

Листинг 7.2. Исходный код класса **Store**

```
/// <summary>
/// Представляет стеллажи
/// </summary>
public class Store
{
    /// <summary>
    /// Код стеллажа
    /// </summary>
    public string StoreID { get; set; }

    /// <summary>
    /// Число ярусов
    /// </summary>
    public int NumberOfFlour { get; set; }

    /// <summary>
    /// Макс. грузонесущая способность, кг
    /// </summary>
    public int MaxWeight { get; set; }
}
```

Требуется разработать следующие 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**).

Данные о товарах организуем в виде массива **itemsInStock**, состоящего из объектов класса **Product**. Массив **itemsInStock** будет объявлен в методе **Main()** (листинг 7.3) консольного приложения. Также в методе **Main()** производится вызов методов, выполняющих запросы LINQ.

Исходный код методов, которые содержат выражения запроса LINQ и выполняют их обработку, представлен в листингах 7.4, 7.5 и 7.6.

Листинг 7.3. Исходный код класса **Program** консольного приложения

```
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Выполнение запросов LINQ к массиву объектов";

        // Коллекция объектов класса Product с инициализацией элементов
        List<Product> products = new List<Product> {
            new Product {   Name = "Крупа гречневая",
                            ProducedBy = "ООО Алтайпродукт",
                            NumberInStok = 42,
                            Weight = 350,
                            Price = 72,
                            StoreID = "024" },
            new Product {   Name = "Кукуруза консервированная",
                            ProducedBy = "ОАО Балтпром",
                            NumberInStok = 67,
                            Weight = 340,
                            Price = 56,
                            StoreID = "017" },
            new Product {   Name = "Крупа рисовая",
                            ProducedBy = "ЗАО Увелка",
                            NumberInStok = 53,
                            Weight = 350,
                            Price = 42,
                            StoreID = "024" },
            new Product {   Name = "Говядина тушеная",
                            ProducedBy = "ОАО Балтпром",
                            NumberInStok = 30,
                            Weight = 330,
                            Price = 96,
                            StoreID = "017" },
            new Product {   Name = "Макароны",
                            ProducedBy = "ООО Алтайпродукт",
                            NumberInStok = 48,
                            Weight = 500,
                            Price = 56,
                            StoreID = "028" }};

        // Коллекция объектов класса Store
        List<Store> stores = new List<Store> {
                new Store { StoreID = "024",
                            NumberOfFlour = 4,
                            MaxWeight = 10000 },
                new Store { StoreID = "028",
                            NumberOfFlour = 4,
                            MaxWeight = 10000 },
                new Store { StoreID = "017",
                            NumberOfFlour = 6,
                            MaxWeight = 12000 }};
```

Листинг 7.4. Исходный код методов, выполняющих запросы LINQ (часть 1)

```
            Console.WriteLine("******* Результаты запросов LINQ *******");
            // Вызовы методов выполнения запросов LINQ
            GetAllProds(products);
            GetAllNames(products);
            GetProdOver(products, 25);
            GetAltPrNam(products, "ООО Алтайпродукт", 80);
            GetNumProds(products, 250, 500);
            GetNameNumb(products);
            GetAvgPrice(products, "ООО Алтайпродукт");
            GetSumWeigt(products);
            GetSumPrice(products);
            GetNameMake(products);
            GetProdBalt(products, stores, "ОАО Балтпром", "ЗАО Увелка");
            Console.ReadLine();
        }

        /// Получить все данные о товарах
        /// </summary>
        /// <param name="products">Список продуктов</param>
        static void GetAllProds(List<Product> products)
        {
            Console.WriteLine("\n1. Все данные о товарах на складе:");
            var all = from p in products select p;
            foreach (var a in all) Console.WriteLine(a.ToString());
        }

        /// <summary>
        /// Получить наименования всех товаров в алфавитном порядке
        /// </summary>
        /// <param name="products">Список продуктов</param>
        static void GetAllNames(List<Product> products)
        {
            Console.WriteLine("\n2. Все наименования товаров на складе (по алфавиту):");
            var names = from p in products orderby p.Name select p.Name;
            foreach (var n in names) Console.WriteLine("- " + n);
        }

        /// <summary>
        /// Получить данные о товарах с количеством больше заданного
        /// </summary>
        /// <param name="products">Список продуктов</param>
        /// <param name="amount">Число единиц</param>
        static void GetProdOver(List<Product> products, int amount)
        {
            Console.WriteLine("\n3. Все товары с количеством более {0}:", amount);
            var overStock = from p in products where p.NumberInStok > amount select p;
            foreach (var os in overStock) Console.WriteLine(os.ToString());
        }
```

Листинг 7.5. Исходный код методов, выполняющих запросы LINQ (часть 2)

```
        /// <summary>
        /// Получить наименования товаров выбранного производителя с ценой
        /// меньше заданной
        /// </summary>
        /// <param name="products">Список продуктов</param>
        /// <param name="make">Производитель</param>
        /// <param name="price">Цена, руб</param>
        static void GetAltPrNam(List<Product> products, string make, int price)
        {
            Console.WriteLine("\n4. Все товары фирмы {0} с ценой менее {1} руб.:",
                make, price);
            var altNames = from p in products
                           where p.ProducedBy == make && p.Price < price
                           select p.Name;
            foreach (var an in altNames) Console.WriteLine("- {0}", an);
        }

        /// <summary>
        /// Найти число наименований товаров с весом в указанном диапазоне
        /// </summary>
        /// <param name="products">Список продуктов</param>
        /// <param name="minW">Наибольший вес, г</param>
        /// <param name="maxW">Наименьший вес, г</param>
        static void GetNumProds(List<Product> products, int minW, int maxW)
        {
            Console.WriteLine("\n5. Число наименований товаров весом от {0} до {1} г:",
                minW, maxW);
            var nameProd = from p in products
                           where p.Weight > minW && p.Weight < maxW
                           select p.Name;
            Console.WriteLine("Всего {0} наименов.", nameProd.Count());
        }

        /// <summary>
        /// Получить наименования и количество продуктов (по убыванию)
        /// </summary>
        /// <param name="products">Список продуктов</param>
        static void GetNameNumb(List<Product> products)
        {
            Console.WriteLine("\n6. Наименование и количество товара (по убыванию):");
            var nameNumb = from p in products
                           orderby p.NumberInStok descending
                           select p;
            foreach (var nn in nameNumb)
            { Console.WriteLine("- {0}, {1} шт.", nn.Name, nn.NumberInStok); }
        }
```

Листинг 7.6. Исходный код методов, выполняющих запросы LINQ (часть 3)

```
157         /// <summary>
158         /// Найти наибольшую, наименьшую и среднюю цены товаров выбранной фирмы
159         /// </summary>
160         /// <param name="products">Список продуктов</param>
161         /// <param name="make">Производитель</param>
162         static void GetAvgPrice(List<Product> products, string make)
163         {
164             Console.WriteLine("\n7. Средняя, наибольшая и наименьшая цены товаров фирмы {0}",
165                 make);
166             var prAltPrice = from p in products
167                              where p.ProducedBy == make
168                              select p.Price;
169             Console.WriteLine("- средняя цена: {0:f2} руб.\n" +
170                               "- наибольшая цена: {1} руб.\n" +
171                               "- наименьшая цена: {2} руб.",
172                 prAltPrice.Average(), prAltPrice.Max(), prAltPrice.Min());
173         }
174 
175         /// <summary>
176         /// Найти суммарный вес товаров на складе
177         /// </summary>
178         /// <param name="products">Список продуктов</param>
179         static void GetSumWeigt(List<Product> products)
180         {
181             Console.WriteLine("\n8. Суммарный вес всех товаров:");
182             var all = from p in products select p;
183             double sumWeight = 0;
184             foreach (var a in all) sumWeight += a.NumberInStok * a.Weight;
185             Console.WriteLine("Всего {0:f2} кг.", sumWeight / 1000);
186         }
187 
188         /// <summary>
189         /// Найти общую стоимость товаров каждого наименования
190         /// </summary>
191         /// <param name="products">Список продуктов</param>
192         static void GetSumPrice(List<Product> products)
193         {
194             Console.WriteLine("\n9. Общая стоимость товаров каждого наименования:");
195             var all = from p in products orderby p.Name select p;
196             foreach (var a in all)
197             {
198                 Console.WriteLine("- {0}, {1} руб.", a.Name,
199                     a.NumberInStok * a.Price);
200             }
201         }
```

Листинг 7.7. Исходный код методов, выполняющих запросы LINQ (часть 4)

```
        /// <summary>
        /// Получить наименования товаров, сгруппированные по производителям
        /// </summary>
        /// <param name="products">Список продуктов</param>
        static void GetNameMake(List<Product> products)
        {
            Console.WriteLine("\n10. Наименования товаров, сгруппир. по производителям:");
            var groups = from p in products
                         group p by p.ProducedBy;
            // Цикл для выбора каждой группы group из списка групп groups
            foreach (var group in groups)
            {
                Console.WriteLine("Товары фирмы {0}:", group.Key);
                // Цикл для выбора каждого элемента elem группы group
                foreach (var elem in group)
                { Console.WriteLine("- " + elem.Name); }
            }
        }


        /// <summary>
        /// Получить наименования и количество товаров выбранных фирм с указанием
        /// данных по стеллажам, на которых они хранятся (исп. join)
        /// </summary>
        /// <param name="products">Список продуктов</param>
        /// <param name="stores">Список стеллажей</param>
        /// <param name="make1">Производитель 1</param>
        /// <param name="make2">Производитель 2</param>
        static void GetProdBalt(List<Product> products, List<Store> stores,
            string make1, string make2)
        {
            Console.WriteLine("\n11. Наименования и количество товаров фирм {0}\n" +
                "или {1} с указанием данных по стеллажам, на которых они хранятся",
                make1, make2);
            var rezult = from p in products
                         where p.ProducedBy == make1 || p.ProducedBy == make2
                         join s in stores on p.StoreID equals s.StoreID
                         select new
                         {
                             name = p.Name,
                             numb = p.NumberInStok,
                             stID = p.StoreID,
                             numF = s.NumberOfFlour,
                             maxW = s.MaxWeight
                         };
            Console.WriteLine("|{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
                     "Наименование", "Количество", "Код стел.", "Число ярусов",
                     "Макс. груз, кг");
            foreach (var r in rezult)
            {
                Console.WriteLine("|{0,25}|{1,10}|{2,10}|{3,12}|{4,15}|",
                    r.name, r.numb, r.stID, r.numF, r.maxW);
            }
        }
```

Результат работы полученного консольного приложения показан на рис. 7.1. ❒

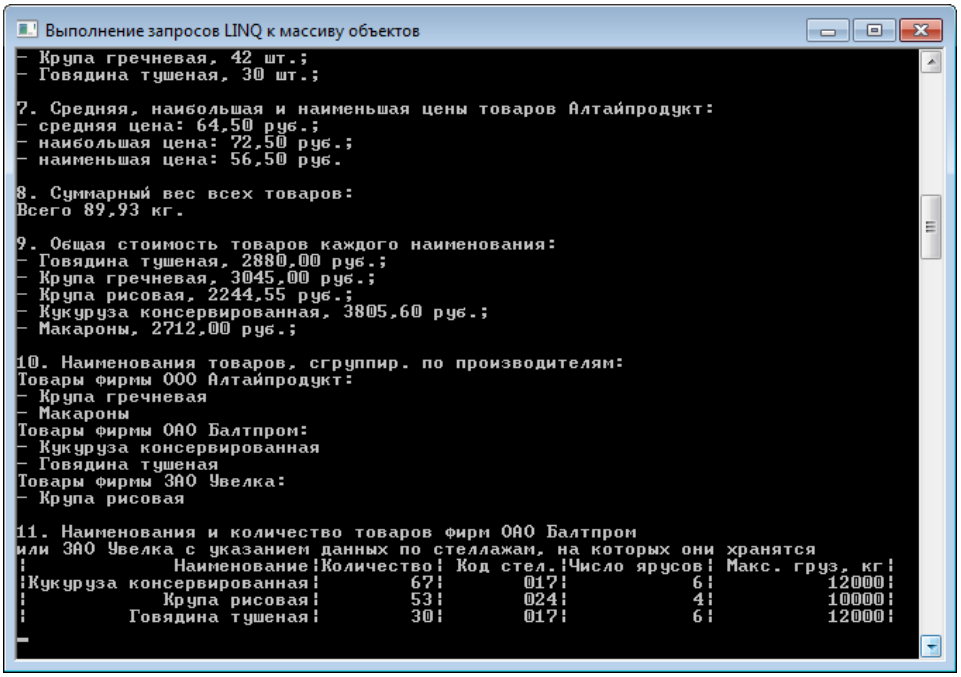


Рис. 7.1. Результат работы консольного приложения

7.3. 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. 7.1), а также коллекцией (массивом, списком и др.) объектов этого класса. Число элементов коллекции должно быть не менее пяти.
3. Выполнить заданные запросы LINQ к коллекции объектов (табл. 7.1). Для запроса №6 следует использовать операцию **group**, а для запроса №7 – операцию **join**.
4. Оформить и защитить отчет по лабораторной работе.

Таблица 7.1

Варианты заданий для создания массивов объектов и запросов к ним

№ вар.	Данные для создания массивов объектов и запросов к ним
1, 13	***Студент.*** Код, ФИО, группа, пол, дата рождения, средний бал, код научного руководителя. ***Научный руководитель.*** Код, ФИО, должность. 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, 14	***Автомобиль.*** Код, фирма, модель, год выпуска, цена, расход топлива, код магазина. ***Магазин.*** Код, название, телефон. 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, 15	***Сотрудник.*** Код, ФИО, пол, дата рождения, должность, зарплата, код отдела. ***Отдел.*** Код, название, телефон. 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, 16	***Книга.*** Код, название, автор, цена, число страниц, год издания, код стеллажа. ***Стеллаж***. Код, число полок, название отдела.

№ вар.	Данные для создания массивов объектов и запросов к ним
	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 одой книги автора «*автор*». 6. Все книги, сгруппированные по коду стеллажа (group). 7. Названия и число страниц книг с указанием числа полок стеллажа и отдела, в котором он расположен (join).
5, 17	***Учебная дисциплина.*** Код, название, ФИО преподавателя, форма контроля, семестр, число часов, код специальности. ***Специальность.*** Код, название, факультет. 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, 18	***Банковский вклад.*** Код, номер счета, ФИО вкладчика, дата вклада, сумма, процент, код банка. ***Банк.*** Код, название, адрес центрального офиса. 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, 19	***Предмет обуви.*** Код, наименование, производитель, число пар, размер, цена, код отдела. ***Отдел.*** Код, название, число сотрудников. 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

№ вар.	Данные для создания массивов объектов и запросов к ним
	сотрудников (join).
8, 20	***Ноутбук.*** Код, фирма, модель, процессор, объем памяти, цена, код магазина. ***Магазин.*** Код, название, адрес. 1. Данные по всем ноутбукам с процессором «…». 2. Модели ноутбуков фирмы «…». 3. Число моделей с ценой от «…» до «…». 4. Модель и производитель ноутбуков с объемом памяти более «…». 5. Средняя цена ноутбуков фирмы «*фирма*». 6. Все ноутбуки, сгруппированные по коду магазина (group). 7. Модель и цена ноутбуков с указанием названия и адреса магазина (join).
9, 21	***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, 22	***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, 23	***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*».

№ вар.	Данные для создания массивов объектов и запросов к ним
	6. Все заказы, сгруппированные по коду водителя (group). 7. Номера и стоимости заказов с указанием ФИО и даты рождения водителя (join).
12, 24	***Спортсмен***. Код, ФИО, вид спорта, дата рождения, пол, рост, вес, код тренера. ***Тренер.*** Код, ФИО, звание. 1. Данные по всем спортсменам. 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).

7.4. 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?

8. ОСНОВЫ РАБОТЫ С XML-ДОКУМЕНТАМИ ПРИ ПОМОЩИ LINQ TO XML

8.1. Цель и задачи работы

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

Основные задачи работы:

- ознакомиться с технологией LINQ to XML;
- научиться создавать и модифицировать XML-документы с помощью классов LINQ to XML;
- освоить создание запросов LINQ к XML-документам.

Работа рассчитана на 6 часов.

8.2. Основные теоретические сведения

8.2.1. 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. 8.1).

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.).

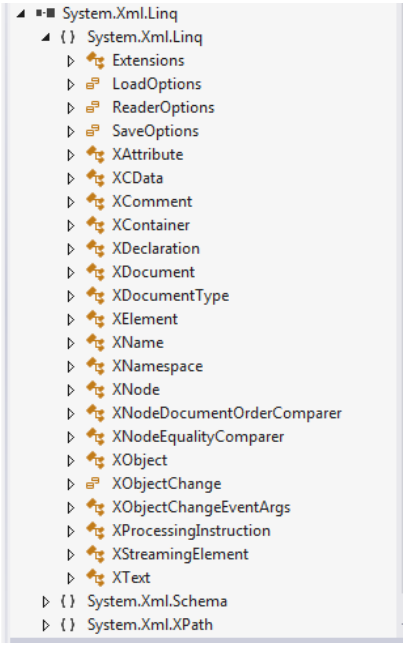


Рис. 2.1. Классы пространства имен **System.Xml.Linq** в окне обозревателя объектов (Visual Studio 2012)

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 **XElemen**t, имеющий указанное имя **name**;

• **Attribute(XName name)** – возвращает атрибут **XAttribute**, имеющий имя **name**;

• **Attribute()** – возвращает коллекцию **IEnumerable** всех атрибутов **XAttribute**.

❒ ***Пример* 8.1. *Создание XML-документа с помощью консольного приложения на языке C#.***

Требуется разработать консольное приложение на языке C#, которое, используя классы LINQ to XML, создаёт с нуля одну ветвь XML-документа, приведённого в примере 1.1. Кроме того, приложение должно загружать полученный документ и отображать его код в окне консоли.

Исходный код полученного приложения представлен в листинге 8.1.

Листинг 8.1. Исходный код консольного приложения

```
using System.Xml.Linq; // Импорт пространства имён для работы с LINQ to XML

namespace LinqToXmlTree
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Создание и чтение XML-документа с помощью LINQ to XML";
            // URI, задающий путь к XML-документу
            string uri = @"H:\XML\FragmCommService.xml";
            XmlCreate(uri);
            XmlLoad(uri);
            Console.Read();
        }

        /// <summary>
        /// Создать XML-документ "с нуля", используя LINQ to XML
        /// </summary>
        /// <param name="url">Путь к XML-документу</param>
        static void XmlCreate(string uri)
        {
            XDocument xDoc =
                new XDocument(
                    new XDeclaration("1.0", "utf-8", "yes"),
                    new XComment("Данные о коммунальных 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("описание",
                                new XCData("Крупнопанельный 9-этажный дом. Построен в 1978 г. " +
                                "Физический износ 14% (дата обслед. 15.06.2008 г.)")),
                            new XElement("квартира", new XAttribute("код", "a238"),
                                                     new XAttribute("номер", "57"),
                                new XElement("площадь", "28"),
                                new XElement("жилец", new XAttribute("код", "c11568"),
                                    new XElement("фио", "Костенко Игорь Сергеевич"),
                                    new XElement("дата_рожд", "1978-11-10")
                                ),
                                new XElement("показ_приборов", new XAttribute("дата", "2014-05-02"),
                                    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", "2014-05-04"),
                                    new XElement("всего", "1896.45"),
                                    new XElement("пеня", "0")
                                )
                            )
                        )
                    )
                );
            xDoc.Save(uri);
            Console.WriteLine("XML-документ сохранён по адресу : {0}\n", uri);
        }

        /// <summary>
        /// Загрузить XML-документ, используя LINQ to XML
        /// </summary>
        /// <param name="url">Путь к XML-документу</param>
        static void XmlLoad(string uri)
        {
            XDocument xDoc = XDocument.Load(uri);
            Console.WriteLine("Код загруженного XML-документа:\n" + xDoc.ToString());
        }
```

Результат работы консольного приложения показан на рис. 8.2.

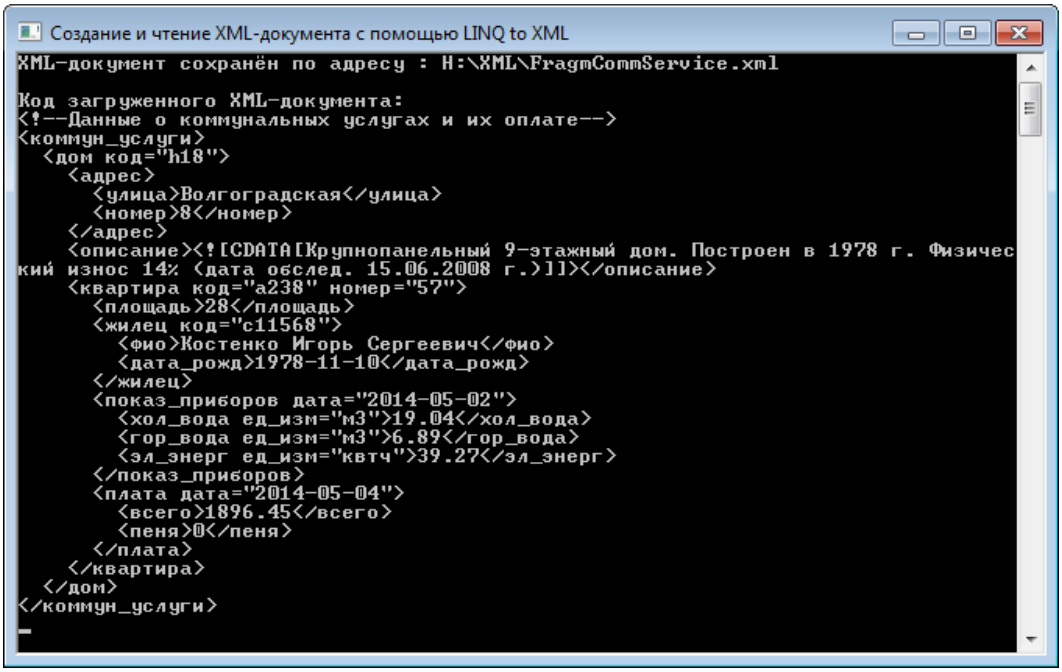


Рис. 8.2. Работа консольного приложения

После выполнения метода **Main()** консольного приложения в файл «FragmCommService.xml» будут записаны данные, представленные в листинге 8.2. ❐

Листинг 8.2. XML-код полученного документа

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--Данные о коммунальных услугах и их оплате-->
<коммун_услуги>
  <дом код="h18">
    <адрес>
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес>
    <описание><![CDATA[Крупнопанельный 9-этажный дом. Построен в 1978 г. Физический износ 14% (дат
    <квартира код="a238" номер="57">
      <площадь>28</площадь>
      <жилец код="c11568">
        <фио>Костенко Игорь Сергеевич</фио>
        <дата_рожд>1978-11-10</дата_рожд>
      </жилец>
      <показ_приборов дата="2014-05-02">
        <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="2014-05-04">
        <всего>1896.45</всего>
        <пеня>0</пеня>
      </плата>
    </квартира>
  </дом>
</коммун_услуги>
```

❒ ***Пример* 8.2. *Создание XML-документа из массива объектов с помощью консольного приложения на языке C#.***

Требуется получить XML-документ на основе массива объектов, созданного в примере 7.1.

Для этого добавим в код консольного приложения из примера 7.1 метод **GenerXML()**, приведенный в листинге. 8.3. Кроме того, в процедуре **Main()** необходимо добавить вызов этого метода с передачей ему массива объектов:

```
GenerXML(products, @"H:\XML\products.xml");
```

В результате будет получен XML-документ, приведенный в листинге 8.4.

Листинг 8.3. Код метода для создания XML-документа из массива объектов

```
        /// <summary>
        /// Создать XML-документ на основе данных из коллекции products
        /// </summary>
        /// <param name="products">Список продуктов</param>
        /// <param name="uri">URI XML-документа</param>
        static void GenerXML(List<Product> products, string uri)
        {
            // Случайная переменная для генерирования кодов
            Random r = new Random();

            // Генерирование XML-данных из products
            var xmlData = from p in products
                          select
                            new XElement("product",
                                new XAttribute("id", r.Next()),
                                new XAttribute("numberInStock", p.NumberInStok),
                                new XElement("name", p.Name),
                                new XElement("make", p.ProducedBy),
                                new XElement("weight", p.Weight, new XAttribute("unit", "г.")
                                new XElement("price", p.Price, new XAttribute("unit", "руб.")
                            );

            // Корневой элемент
            XElement rootElem = new XElement("productStock", xmlData);

            // Документ XML
            XDocument xmlDoc = new XDocument();
            xmlDoc.Add(rootElem);
            xmlDoc.Save(uri);
        }
    }
```

Листинг 8.4. Код полученного XML-документа

```
<?xml version="1.0" encoding="utf-8"?>
<productStock>
  <product id="1167223024" numberInStock="42">
    <name>Крупа гречневая</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">72</price>
  </product>
  <product id="152463349" numberInStock="67">
    <name>Кукуруза консервированная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">340</weight>
    <price unit="руб.">56</price>
  </product>
  <product id="1627966045" numberInStock="53">
    <name>Крупа рисовая</name>
    <make>ЗАО Увелка</make>
    <weight unit="г.">350</weight>
    <price unit="руб.">42</price>
  </product>
  <product id="1129762239" numberInStock="30">
    <name>Говядина тушеная</name>
    <make>ОАО Балтпром</make>
    <weight unit="г.">330</weight>
    <price unit="руб.">96</price>
  </product>
  <product id="356943331" numberInStock="48">
    <name>Макароны</name>
    <make>ООО Алтайпродукт</make>
    <weight unit="г.">500</weight>
    <price unit="руб.">56</price>
  </product>
</productStock>
```

8.2.2. 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* 8.3. *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.

Создадим следующие 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 $м^3$ или расход электроэнергии менее 120 квтч.
5. Квартиры с площадью более 30 $м^2$, сгруппированные по обслуживаемым домам.
6. ФИО жильцов дома с кодом «h18», сгруппированные по номерам квартир.
7. Работники ЖКК 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 ЖКК (код, код дома, ФИО, должность). Для этого в методе, реализующем запрос LINQ, опишем XML-элемент **работники_ЖКК**, 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 в листинге 8.5.

Листинг 8.5. Исходный код метода **Main()** консольного приложения

```
using System.Xml.Linq;
using System.Xml.XPath;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Выполнение запросов LINQ к XML-документу";

            string uri = @"H:\XML\CommService.xml";

            XDocument xDoc = XDocument.Load(uri);

            Console.WriteLine("***** Результаты запросов LINQ к XML-документу *****");

            XElement commWorkers = new XElement("работники_ЖКК",
            new XElement("работник", new XAttribute("код", "e012"),
                new XAttribute("код_дома", "h18"),
                new XElement("фио", "Мишутин Дмитрий Олегович"),
                new XElement("должность", "дворник")
                ),
                new XElement("работник", new XAttribute("код", "e067"),
                new XAttribute("код_дома", "h18"),
                new XElement("фио", "Сидоренко Татьяна Николаевна"),
                new XElement("должность", "уборщик")
                ),
                new XElement("работник", new XAttribute("код", "e104"),
                new XAttribute("код_дома", "h72"),
                new XElement("фио", "Ермолова Галина Степановна"),
                new XElement("должность", "уборщик")
                ));

            // Вызовы методов, выполняющих запросы к XML-документу
            GetPersData(xDoc);
            GetWaterElc(xDoc);
            GetAptsData(xDoc, "Волгоградская", "8");
            GetNumAppts(xDoc, 100.0, 120.0);
            GetAppts30m(xDoc, 30);
            GetPersName(xDoc, "h18");
            GetComWorks(xDoc, commWorkers);
            GetAparCode(xDoc, 0);
            Console.Read();
        }
```

Исходный код методов, выполняющих запросы к документу XML, представлен в листинге 8.7.

Листинг 8.6. Исходный код методов выполнения запросов к XML-документу (часть 1)

```
52          /// <summary>
53          /// Получить данные по всем жильцам (в алфавитном порядке)
54          /// </summary>
55          /// <param name="doc">XML-документ</param>
56          static void GetPersData(XDocument doc)
57          {
58              var rezult = from elPers in doc.Descendants("жилец")
59                           orderby elPers.Element("фио").Value
60                           select elPers;
61
62              Console.WriteLine("\n1. Данные по всем жильцам обслуживаемых домов:");
63              // "Шапка" текстовой таблицы
64              Console.WriteLine("|{0,15}{1,15}|{2,14}|", "ФИО", "", "Дата рождения");
65              foreach (var r in rezult)
66              {
67                  Console.WriteLine("|{0,30}|{1,14}|", r.Element("фио").Value,
68                      r.Element("дата_рожд").Value);
69              }
70          }
71
72          /// <summary>
73          /// Получить суммарный расход воды и электроэнергии
74          /// </summary>
75          /// <param name="doc">XML-документ</param>
76          static void GetWaterElc(XDocument doc)
77          {
78              var rezult = from d in doc.Descendants("показ_приборов")
79                           select d;
80
81              Console.WriteLine("\n2. Суммарный расход воды и электроэнергии:");
82              double sumColWater = 0;
83              double sumHotWater = 0;
84              double sumElcPower = 0;
85              foreach (var r in rezult)
86              {
87                  sumColWater += Convert.ToDouble(r.Element("хол_вода").Value);
88                  sumHotWater += Convert.ToDouble(r.Element("гор_вода").Value);
89                  sumElcPower += Convert.ToDouble(r.Element("эл_энерг").Value);
90              }
91              Console.WriteLine("- Холодная вода: {0:f2} м3\n" +
92                                "- Горячая вода: {1:f2} м3\n" +
93                                "- Электроэнергия: {2:f2} квтч",
94                                sumColWater, sumHotWater, sumElcPower);
95          }
```

Листинг 8.7. Исходный код методов выполнения запросов к XML-документу (часть 2)

```
97          /// <summary>
98          /// Получить данные по квартирам, расположенным в указанном доме
99          /// </summary>
100         /// <param name="doc">XML-документ</param>
101         /// <param name="street">Улица</param>
102         /// <param name="number">Номер дома</param>
103         static void GetAptsData(XDocument doc, string street, string number)
104         {
105             var rezult = from d in doc.Descendants("дом")
106                          from k in d.Elements("квартира")
107                          from a in d.Elements("адрес")
108                          where a.Element("улица").Value == street &&
109                                a.Element("номер").Value == number
110                          select k;
111
112             Console.WriteLine("\n3. Данные по квартирам в доме {0}, ул. {1}:",
113                 street, number);
114             Console.WriteLine("|{0,6}|{1,12}|", "Номер", "Площадь, м2");
115             foreach (var r in rezult)
116             {
117                 Console.WriteLine("|{0,6}|{1,12}|",
118                     r.Attribute("номер").Value, r.Element("площадь").Value);
119             }
120         }
121
122         /// <summary>
123         /// Найти число квартир, в которых расход горячей воды менее X м3
124         /// или расход электроэнергии менее Y квтч
125         /// </summary>
126         /// <param name="doc">XML-документ</param>
127         /// <param name="hotWater">Расход горячей воды, м3</param>
128         /// <param name="electr">Расход электроэнергии, квтч</param>
129         static void GetNumAppts(XDocument doc, double hotWater, double electr)
130         {
131             var rezult = from a in doc.Descendants("квартира")
132                          from p in a.Elements("показ_приборов")
133                          where Convert.ToDouble(p.Element("гор_вода").Value) < hotWater ||
134                                Convert.ToDouble(p.Element("эл_энерг").Value) < electr
135                          select a;
136             int numAp = rezult.Count();
137             Console.WriteLine("\n4. Число квартир с расходом горячей воды < {0} м3" +
138                               " или расходом электроэнергии < {1} квтч: {2}",
139                               hotWater, electr, numAp);
140         }
```

Листинг 8.7. Исходный код методов выполнения запросов к XML-документу (часть 3)

```
142         /// <summary>
143         /// Получить коды и номера квартир с площадью более S м2,
144         /// сгруппированные по домам
145         /// </summary>
146         /// <param name="doc">XML-документ</param>
147         /// <param name="square">Площадь квартиры, м2</param>
148         static void GetAppts30m(XDocument doc, int square)
149         {
150             var groups = from d in doc.Descendants("дом")
151                          from a in d.Elements("квартира")
152                          where (int)a.Element("площадь") > square
153                          group a by d.Attribute("код");
154 
155             Console.WriteLine("\n5. Квартиры с площадью более {0} м2 " +
156                 "(сгруп. по домам):", square);
157             // Цикл для выбора каждой группы group из списка групп groups
158             foreach (var group in groups)
159             {
160                 Console.WriteLine("Квартиры в доме " + group.Key);
161                 // Цикл для выбора каждого элемента elem группы group
162                 foreach (var elem in group)
163                 {
164                     Console.WriteLine("- {0}, №{1}", elem.Attribute("код"),
165                     elem.Attribute("номер").Value);
166                 }
167             }
168         }
169 
170         /// <summary>
171         /// Получить ФИО жильцов дома с указанным кодом, сгруппированные по квартирам
172         /// </summary>
173         /// <param name="doc">XML-документ</param>
174         /// <param name="houseID">Код дома</param>
175         static void GetPersName(XDocument doc, string houseID)
176         {
177             var groups = from h in doc.Descendants("дом")
178                          from a in h.Elements("квартира")
179                          from p in a.Elements("жилец")
180                          where h.Attribute("код").Value == houseID
181                          group p by a.Attribute("номер").Value;
182 
183             Console.WriteLine("\n6. ФИО жильцов дома с кодом {0} " +
184                 "(сгруп. по квартирам):", houseID);
185 
186             foreach (var group in groups)
187             {
188                 Console.WriteLine("Жильцы квартиры №{0}:", group.Key);
189                 foreach (var elem in group)
190                 {
191                     Console.WriteLine("- {0}", elem.Element("фио").Value);
192                 }
193             }
194         }
```

Листинг 8.7. Исходный код методов выполнения запросов к XML-документу (часть 4)

```
196         /// <summary>
197         /// Получить данные о работниках ЖКК с указанием адресов домов,
198         /// которые они обслуживают (join)
199         /// </summary>
200         /// <param name="doc">XML-документ</param>
201         /// <param name="commWorkers">XML-данные о работниках ЖКК</param>
202         static void GetComWorks(XDocument doc, XElement commWorkers)
203         {
204             var rezult = from a in doc.Descendants("дом")
205                          from s in a.Elements("адрес")
206                          join b in commWorkers.Elements("работник")
207                            on a.Attribute("код").Value equals b.Attribute("код_дома").Value
208                          select new
209                          {
210                              name = b.Element("фио").Value,
211                              disg = b.Element("должность").Value,
212                              strt = s.Element("улица").Value,
213                              numb = s.Element("номер").Value
214                          };
215 
216             Console.WriteLine("\n7. Работники ЖКК и обслуживаемые ими дома:");
217             Console.WriteLine("|{0,30}|{1,10}|{2,15}|{3,10}|",
218                 "ФИО", "Должность", "Улица", "Номер дома");
219             foreach (var r in rezult)
220             {
221                 Console.WriteLine("|{0,30}|{1,10}|{2,15}|{3,10}|",
222                     r.name, r.disg, r.strt, r.numb);
223             }
224         }
225 
226         /// <summary>
227         /// Получить данные о жильцах квартир,
228         /// в которых пеня более X руб. (с помощью XPath)
229         /// </summary>
230         /// <param name="doc">XML-документ</param>
231         /// <param name="penalty">Пеня, руб</param>
232         static void GetAparCode(XDocument doc, int penalty)
233         {
234             var rezult = doc.XPathSelectElements(string.Format(
235                 "//квартира[плата/пеня>{0}]/жилец/фио", penalty));
236 
237             Console.WriteLine("\n8. Жильцы квартир, в которых пеня > {0} руб.",
238                 penalty);
239             foreach (var r in rezult) Console.WriteLine("- " + r.Value);
240         }
241     }
```

Результат работы полученного консольного приложения показан на рис. 8.2. ❒

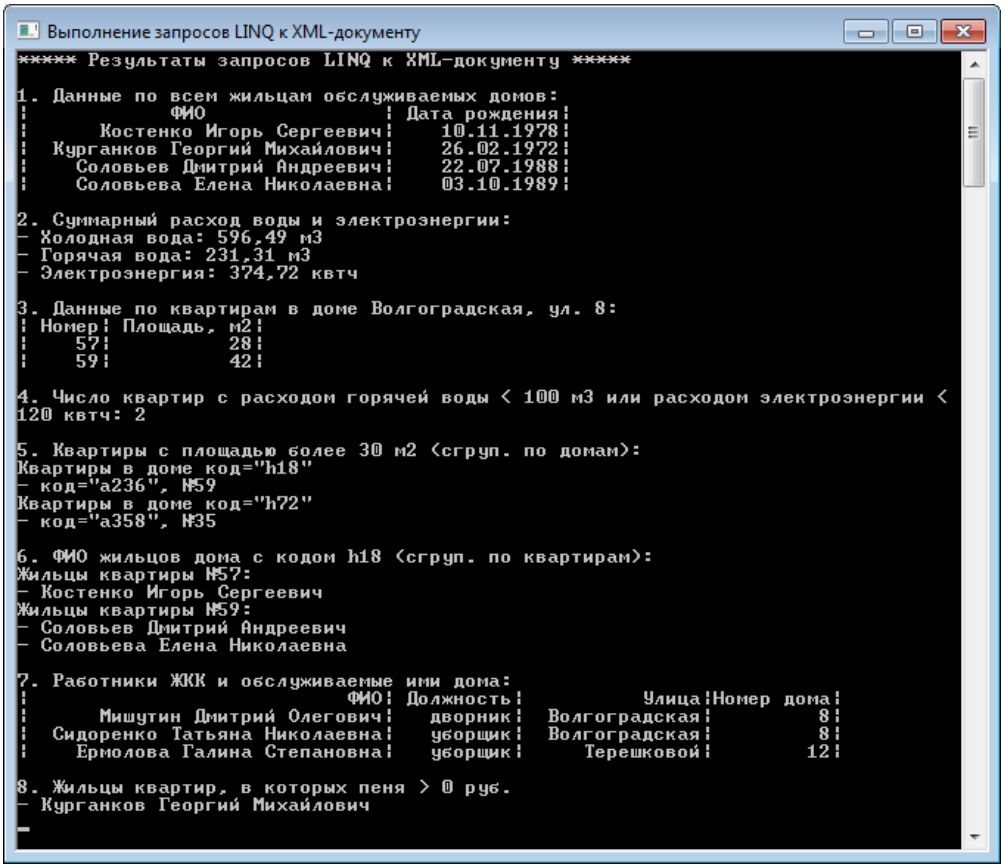


Рис. 8.2. Результат работы консольного приложения

8.3. 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 №7 (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. 8.1) к

XML-документу из работы №1. 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 8.1

Варианты заданий для разработки запросов LINQ к XML-документу

№ вар.	Данные для разработки запросов LINQ к XML-документу
1, 13	***Расписание рейсов междугородных автобусов.*** 1. Время и цена билетов из пункта отправления «*Название*». 2. Число рейсов, из пункта «...» в пункт «...». 3. Данные о рейсах с ценой > «…», сгруп. по пунктам прибытия. 4. Список рейсов с указанием данных об автобусах (join). ***Автобусы.*** Автобус (код, марка, число мест, код рейса).
2, 14	***Товарный склад.*** 1. Наименование и цена товаров производителя «*Название*». 2. Суммарный вес всех товаров на стеллаже с кодом «*Код*». 3. Данные о товарах с количеством > «…», сгруп. по стеллажам. 4. Список товаров с указанием данных о заказе (join). ***Заказ*** (код, дата, ФИО заказчика, код товара).
3, 15	***Ресторан.*** 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. Заказы с указанием официанта (join). ***Официанты.*** Официант (код, ФИО, код заказа).
4, 16	***Расписание занятий.*** 1. Названия дисциплин и типы занятий, провод. в аудитории «…». 2. Число дней недели, по которым проводится более «…» занятий. 3. Данные о занятиях с типом «*Тип*», сгрупп. по дням недели. 4. Список консультаций с указанием дня недели (join). ***Консультации***. Консультация (код, ФИО преп., время, код дня).
5, 17	***Книжный магазин.*** 1. Названия и авторы книг, выпущенных в издательстве «...». 2. Число журналов с ценой от «…» до «…». 3. Данные по книгам с ценой > «…», сгруппир. по отделам. 4. Список отделов с указанием сотрудников (join). ***Сотрудники магазина.*** Сотрудник (код, ФИО, код отдела)

№ вар.	Данные для разработки запросов LINQ к XML-документу
6, 18	***Кинотеатр.*** 1. Дата и время начала сеансов с ценой билета более «*Цена*». 2. Число показанных фильмов, снятых режиссером «*Режиссер*». 3. Данные о фильмах, сгруппированные по залам. 4. Список сеансов с указанием данных о билетах (join). ***Билеты***. Билет (код, номер места, ряд, код сеанса).
7, 19	***Проектно-строительная компания.*** 1. Фамилия, имя и отчество сотрудников с датой рождения «*Дата*». 2. Суммарная стоимость проектов, выполняемых в отделе «…». 3. Данные по сотрудникам пола «*Пол*», сгруппиров. по отделам. 4. Список отделов с указанием задания (join). ***Задания.*** Задание (код, название, дата выполнения, код отдела)
8, 20	***Обувной магазин.*** 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. 4. Список отделов с указанием сотрудников (join). ***Сотрудники магазина***. Сотрудник (код, ФИО, код отдела).
9, 21	***Перевозка грузов.*** 1. Даты заказов и адреса доставки грузов с весом более «*Вес*». 2. Число заказов, в которых вес груза составил от «…» до «…». 3. ФИО водителей, сгруппир. по автомобилям с «*Тип кузова*». 4. Список заказов с указанием заказчиков (join). ***Заказчики.*** Заказчик (код, ФИО, телефон, код заказа)
10, 22	***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 врачей с указанием кабинетов (join). ***Кабинеты.*** Кабинет (код, номер, код врача).
11, 23	***Банковские услуги.*** 1. Вид и сумма вкладов с процентной ставкой более «*Процент*». 2. Число кредитов, по которым процент составляет от «…» до «…».. 3. Данные о кредитах, сгруппированные по клиентам. 4. Список клиентов с указанием данных о картах (join). ***Банковские карты***. Карта (код, дата выдачи, срок действия, код клиента).
12, 24	***Прогноз погоды.*** 1. Направления и скорости ветра для времени суток «*Тип*». 2. Число дней, в которые температура днём была ниже «…». 3. Данные об атмосферных явлениях, сгруппир. по дням недели.

№ вар.	Данные для разработки запросов LINQ к XML-документу
	4. Список населённых пунктов с указанием температуры и давления. ***Населённые пункты***. Населённый пункт (код, название, область, код прогноза).

8.4. 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?

9. ОСНОВЫ СОЗДАНИЯ ПРИЛОЖЕНИЙ WPF С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА XAML

9.1. Цель и задачи работы

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

Основные задачи:

• ознакомиться с технологией WPF и языком XAML;

• научиться создавать простые приложения WPF с использованием языка XAML;

• приобрести умение создавать приложения WPF со страничной навигацией.

Работа рассчитана на 4 часа.

9.2. Основные теоретические сведения

9.2.1. 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. 9.1. показан пример одного из таких приложений – Contoso Healthcare Sample Application, которое предназначено для просмотра медицинских карт пациентов в учреждениях здравоохранения.

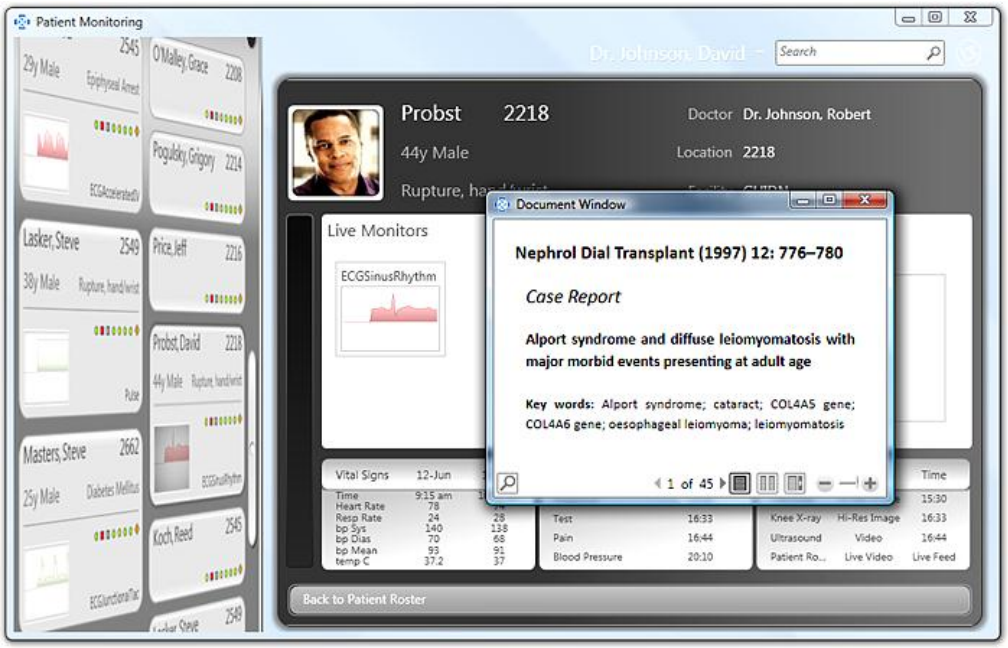


Рис. 9.1. Пример окна приложения WPF (Contoso Healthcare Sample Application)

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, предназначной 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 вы-

хода 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 2012.

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 приложений WPF 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. 9.2).

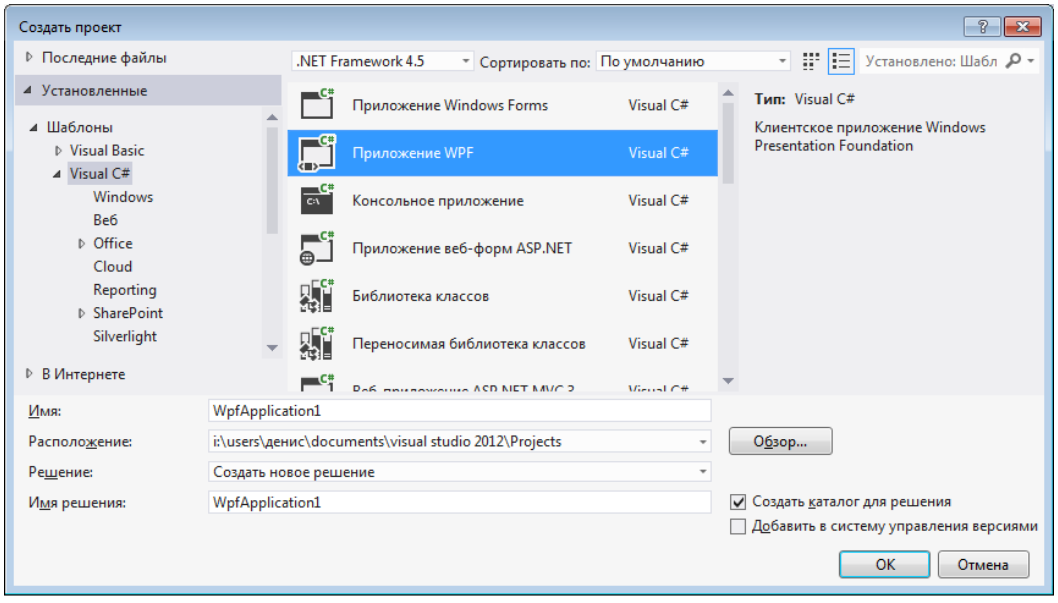


Рис. 9.2. Окно создания нового проекта с выбранным шаблоном «Приложение WPF» (Visual Studio 2012)

В Visual Studio 2012 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 могут 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. 9.3 показан общий вид окна Visual Studio 2012 для разработки проекта приложения WPF. Основную часть окна проекта занимают визуальный конструктор и текстовый редактор XAML.

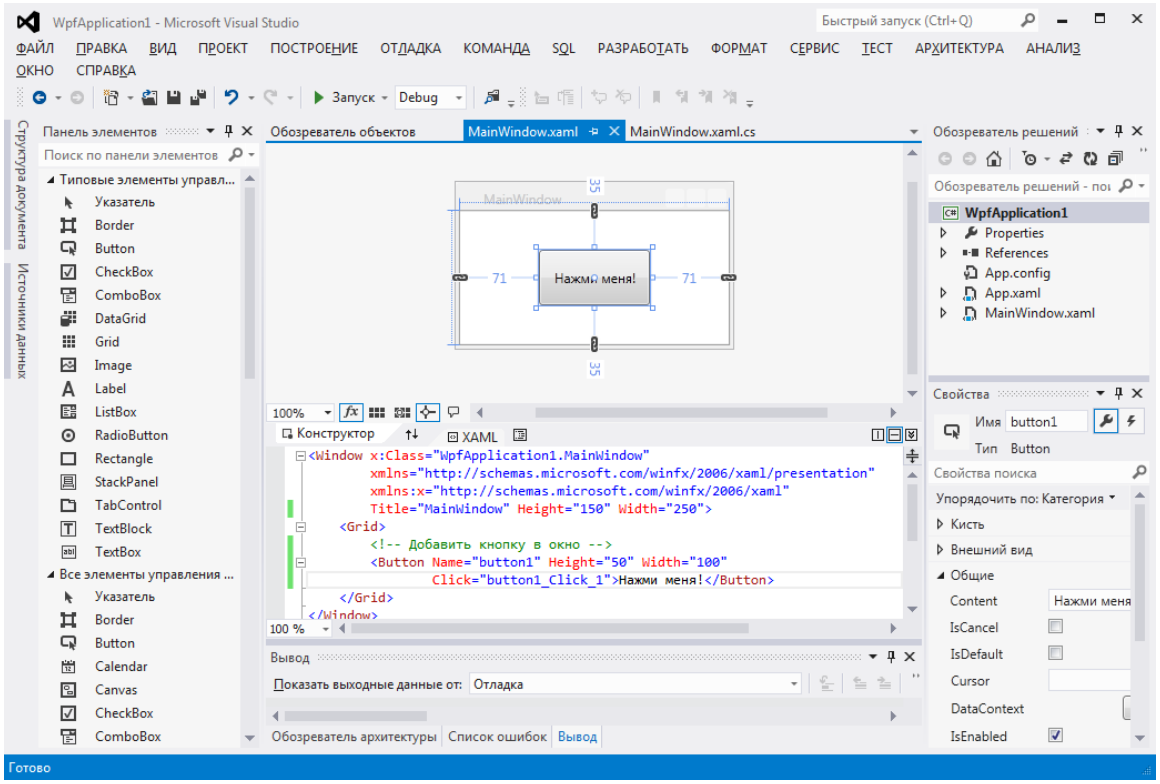


Рис. 9.3. Окно Visual Studio 2012 с открытым проектом приложения WPF

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. 9.4).

Окно «Структура документа» служит для быстрого выбора элементов с целью редактирования в визуальном конструкторе Visual Studio. Открыть данное окно можно через меню **Вид | Другие окна** (**View | Other Windows**) и команду **Структура документа** или с помощью комбинации клавиш **Ctrl + Alt + T**.

9.2.2. 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»).

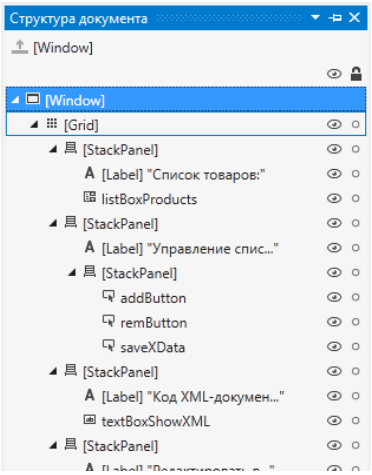


Рис. 9.4. Окно «Структура документа» (**Document Outline**)

Язык XAML (*Extensible Application Markup Language* – расширяемый язык разметки приложений) – язык разметки, являющийся диалектом языка XML, который используемый 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="WpfApplication.MainWindow"
    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. 9.5.

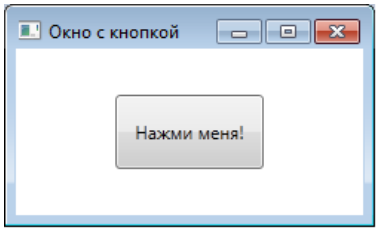


Рис. 9.5. Окно с кнопкой работающего приложения WPF

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 точности 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. 9.6).

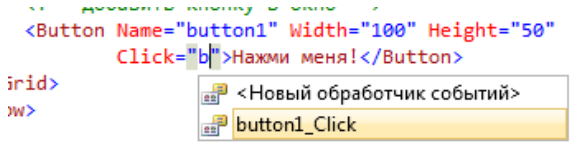


Рис. 9.6. Задание имени обработчика события в редакторе XAML

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

9.2.3. 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: **UltraExpandeded** (растяги-

вает 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 **Grig.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. 9.7.

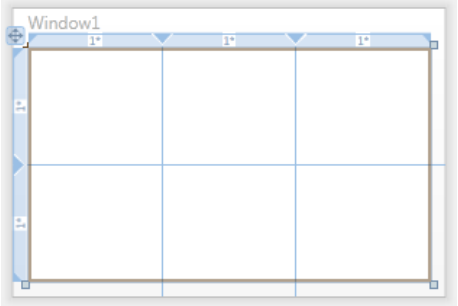


Рис. 9.7. Отображение элемента **Grid** в окне визуального конструктора

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* 9.1. *Разработка приложения WPF для выполнения параметрических запросов к массиву объектов*.**

Требуется разработать приложение WPF, которое позволяет производить запросы LINQ к массиву из примера 7.1. 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 7.1, 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 в листинге 9.1. Связанный с данным документом код на C# (модуль **MainWindow.xaml.cs**), содержащий обработчики событий, показан в листинга 9.2 и 9.3.

Листинг 9.1. Код XAML-документа **MainWindow.xaml**

```
<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="#99BBFF">
        <!-- 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="#99BBFF">
                <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="#99BBFF">
                <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="#99BBFF">
                <Label Width="45">W1, г:</Label>
                <TextBox Name="textBoxW1" Width="50" Margin="5"></TextBox>
                <Label Width="45">W2, г:</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="#99BBFF">
                <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="#99BBFF">
                <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>
```

Листинг 9.2. Код на C# для модуля **MainWindow.xaml.cs** (часть 1)

```
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppSimple
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        // Объявление массива объектов класса Product с инициализацией элементов
        private Product[] products = new[] {
                new Product {   ProductID = "p02356",
                                Name = "Крупа гречневая",
                                ProducedBy = "ООО Алтайпродукт",
                                NumberInStok = 48,
                                Weight = 500,
                                Price = 56.50,
                                MakeData = new DateTime(2013, 08, 04)
                }
            };

        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            textBox1.Text = "50";
            textBox3.Text = "330";

            // Заполнить список listBoxProducts элементами
            foreach (Product p in products)
            { listBoxProducts.Items.Add(p.ProductID + " - " + p.Name); }

            // Заполнить список comboBox2 производителями
            foreach (Product p in products) comboBox2.Items.Add(p.ProducedBy);
            comboBox2.SelectedIndex = 0;
        }

        // --- Вывести данные о товаре по веделенному пункту списка listBoxProducts ---
        private void listBoxProducts_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            textBlockInfo.Text = products[listBoxProducts.SelectedIndex].ToString();
        }
```

Листинг 9.3. Код на C# для модуля **MainWindow.xaml.cs** (часть 2)

```
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 rezult = from p in products where p.NumberInStok > n select p;
99              string s = "";
100             foreach (var r in rezult) 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 rezult = from p in products
108                          where p.ProducedBy == comboBoxM.Text
109                          select p;
110             string s = "";
111             foreach (var r in rezult) 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 rezult = from p in products
121                          where p.Weight > w1 && p.Weight < w2
122                          select p;
123             string s = string.Format("{0} шт.", rezult.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 rezult = from p in products where p.Price < price select p;
132             double sumCost = 0;
133             foreach (var r in rezult)
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 меньшее P");
139         }
140
141         //--- Получить наимнования 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 rezult = from p in products
146                          where dtM.CompareTo(p.MakeData) > 0
147                          select p;
148             string s = "";
149             foreach (var r in rezult) s += string.Format("- {0}\n", r.Name);
150             MessageBox.Show(s, "Товары с датой выпуска позже заданной");
151         }
```

Окно запущенного приложения WPF показано на рис. 9.8. Результаты одного из параметрических запросов, выводимые в окне сообщения, представлены на рис. 9.9. ❐

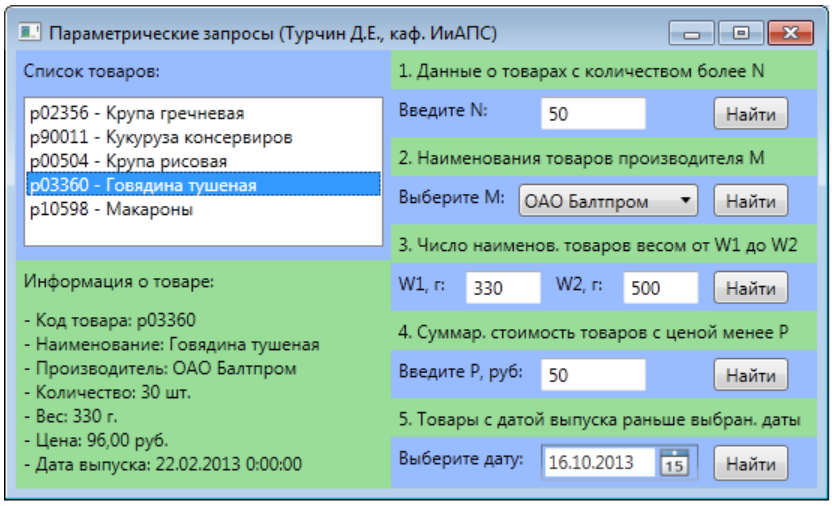


Рис. 9.8. Окно работающего приложения WPF

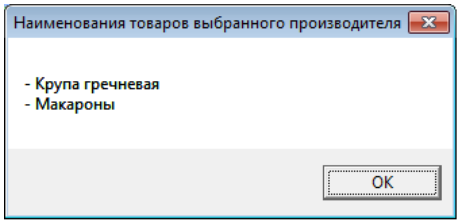


Рис. 9.9. Окно сообщения при выполнении запроса №2

9.2.4. 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 ***потоковыми 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* 9.2. *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, приведён в листингах 9.4, 9.5 и 9.6.

Листинг 9.4. Код XAML-документа **MainWindow.xaml** (главное окно)

```
<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>
```

Листинг 9.5. Код XAML-документа **MainPage.xaml** (главная страница)

```
<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>
```

Листинг 9.6. Код XAML-документа **ApartmsPage.xaml** (страница «Квартиры»)

```
<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>
```

Окно работающего приложения WPF с загруженными в него страницами показано на рис. 9.10 и 9.11. ❒

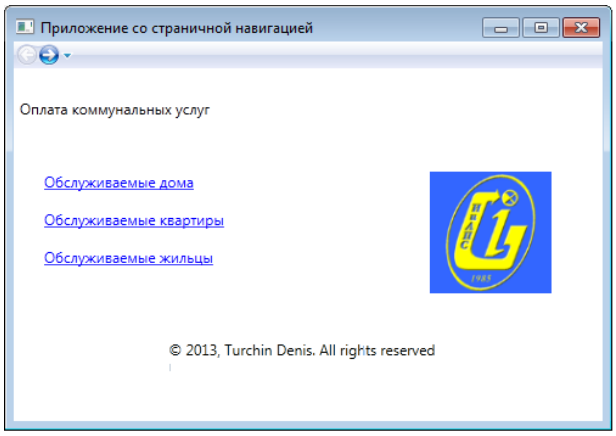


Рис. 9.10. Окно приложения с загруженной главной страницей

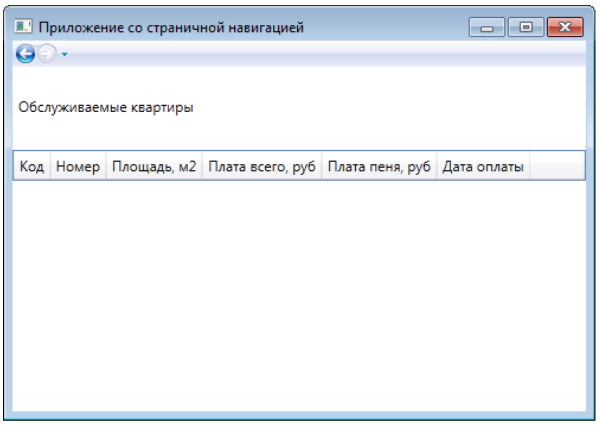


Рис. 9.11. Окно приложения с загруженной страницей «Квартиры»

9.3. 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 работы №7. 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.

9.4. 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 присоединёнными свойствами 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?

10. ОСНОВЫ ПРИВЯЗКИ И ФОРМАТИРОВАНИЯ ДАННЫХ В ПРИЛОЖЕНИЯХ WPF

10.1. Цель и задачи работы

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

Основные задачи:

• ознакомиться с возможностями привязки и форматирования данных в приложениях WPF;

• научиться привязывать данные к элементам управления WPF;

• освоить работу с шаблонами данных в приложениях WPF.

Работа рассчитана на 4 часа.

10.2. Основные теоретические сведения

10.2.1. Привязка данных WPF. Основные поставщики данных

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. 5.1.

Рис. 10.1. 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 связывании данных XML для указания значения используют свойство **XPath**, которому присваивается XPath-выражение.

❒ ***Пример* 10.1. *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 8.2. В доработанном виде этот документ показан в листинге 10.1.

Листинг 10.1. Код XML-документа

```xml
<?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>
```

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 в листингах 10.2, 10.3 и 10.4, код на C# представлен в листингах 10.5 и 10.6.

Листинг 10.2. Код XAML-документа **MainWindow.xaml** с описанием ресурсов (часть 1)

```
<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="#BB5555"
                    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>
```

Листинг 10.3. Код XAML-документа **MainWindow.xaml** с описанием ресурсов (часть 2)

```
<Grid Background="#99BBFF">
    <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>
```

Листинг 10.4. Код XAML-документа **MainWindow.xaml** с описанием ресурсов (часть 3)

```
        <!-- 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>
```

Листинг 10.5. Исходный код на C# (модуль **MainWindow.xaml.cs**, часть 1)

```
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          }
```

Листинг 10.6. Исходный код на C# (модуль **MainWindow.xaml.cs**, часть 2)

```
        // --- Удаление выделенного элемента product из XML-документа при нажатии remButton ---
        private void remButton_Click(object sender, RoutedEventArgs e)
        {
            int index = listBoxProducts.SelectedIndex;
            // Проверка индекса выделенного элемента в списке listBoxProducts
            if (index < 0) return;

            // Определение элемента на основе выделенного пункта в списке listBoxProducts
            XElement selProduct = (XElement)listBoxProducts.SelectedItem;

            selProduct.Remove();

            listBoxProducts.DataContext = xDoc.Descendants("productStock").Elements();
            textBoxShowXML.Text = xDoc.ToString();
            xDoc.Save("XMLDocProducts.xml");
        }

        // --- Принудительное сохранение данных с обновлением отображаемого кода XML ---
        private void saveXData_Click(object sender, RoutedEventArgs e)
        {
            textBoxShowXML.Text = xDoc.ToString();
            xDoc.Save("XMLDocProducts.xml");
        }

    }
}
```

Окно работающего приложения WPF с привязкой данных показано на рис. 10.2. ❒

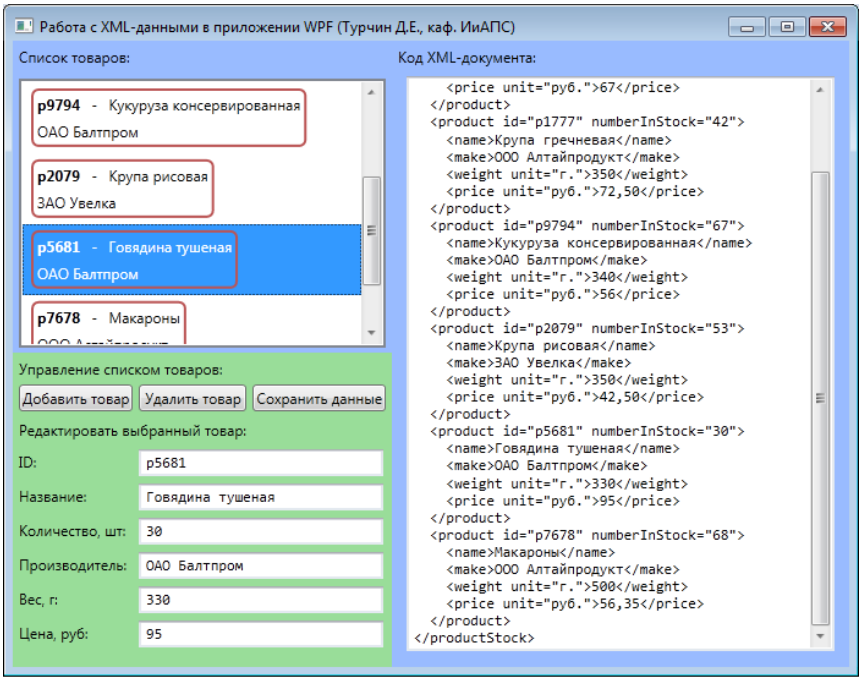


Рис. 10.2. Работа приложения WPF

10.2.2. 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* 10.2. *Привязка и форматирование данных из документа XML в страничном приложении WPF*.**

Требуется доработать приложение со страничной навигацией, полученное в примере 9.1. Для этого добавим привязку таб-

лиц (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 в листинге 10.7.

Добавим в приложение класс **XMLObjectModel**, который будет содержать набор статических методов для выполнения действий с XML-документом. Исходный код данного класса приведен в листинге 10.8.

Код присоединенного модуля на C# для страницы «квартиры» представлен в листинге 10.9.

Листинг 10.7. Код страницы **ApartmsPage.xaml** (квартиры)

```
<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>
```

Листинг 10.8. Исходный код класса **XMLObjectModel**

```
using System.Windows;
using System.Xml.Linq;

namespace WpfAppCommData
{
    // Класс для определения набора статических методов, реализующих
    // логику LINQ to XML
    class XMLObjectModel
    {
        // Метод, который возвращает XDocument, заполненный на основе
        // содержимого файла "CommService.xml"
        public static XDocument GetXDocument()
        {
            try
            {
                XDocument xDoc = XDocument.Load("CommService.xml");
                return xDoc;
            }
            catch (System.IO.FileNotFoundException ex)
            {
                // Вывод сообщения об ошибке при отсутствии XML-файла
                MessageBox.Show(ex.Message);
                return null;
            }
        }
```

Листинг 10.9. Исходный код на C# (модуль **ApartmsPage.xaml. cs**)

```
using System.Windows.Shapes;
using System.Xml.Linq;

namespace WpfAppCommData
{
    /// <summary>
    /// Логика взаимодействия для ApartmsPage.xaml
    /// </summary>
    public partial class ApartmsPage : Page
    {
        public ApartmsPage()
        {
            InitializeComponent();
        }

        // --- Выполнение различных действий при загрузке страницы ---
        private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            XDocument xDoc = XMLObjectModel.GetXDocument();

            // Запрос LINQ с получением данных о квартирах
            var apartms = from a in xDoc.Descendants("квартира") select a;

            // Привязка результатов запроса LINQ к элементу ApartmsDataGrid
            ApartmsDataGrid.ItemsSource = apartms.ToList();
        }
    }
}
```

Результат работы приложения WPF для загруженных страниц «квартиры» и «жильцы» показан на рис. 10.3 и 10.4. ❒

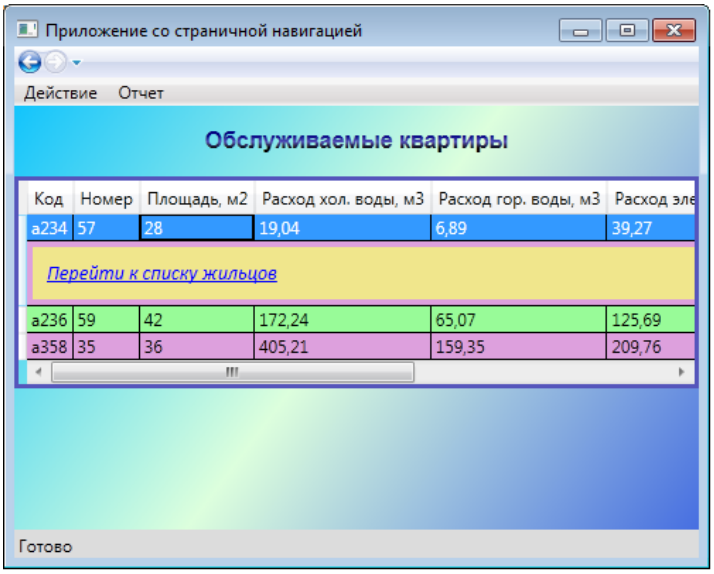


Рис. 10.3. Работа приложения WPF (страница «квартиры»)

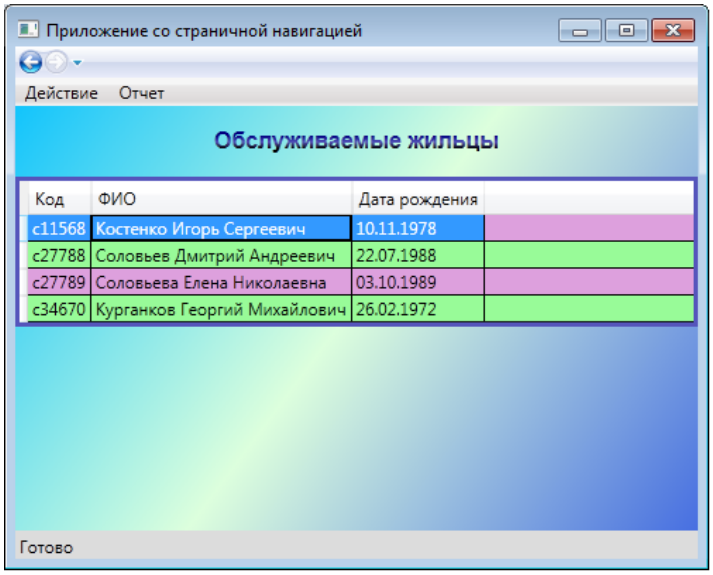


Рис. 10.4. Работа приложения WPF (страница «жильцы»)

10.3. 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 №8 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.

10.4. 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?

СПИСОК ЛИТЕРАТУРЫ

Печатные издания.

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. Бадд, Т. Объектно-ориентированное программирование в действии / Т. Бадд. – СПб.: Питер, 2002. – 796 с.

4. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс, Э. Клементс, Д. Кацман. – СПб.: Питер, 2006. – 240 с.

5. Бек, К. Шаблоны реализации корпоративных приложений : пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 176 с.

6. Белов, В. В. Проектирование информационных систем: учебник для студ. учреждений высш. проф. образования / В. В. Белов, В. И. Чистякова; под ред. В. В. Белова. – М.: Издательский центр «Академия», 2013. – 352 с.

7. Биллиг, В. А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) / В. А. Биллиг. – М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. – 582 с.

8. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч [и др.]. – 3-е изд.: gep. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 720 с.

9. Буч Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – 2-е изд.: пер. с англ. – М.: ДМК Пресс, 2006. – 496 с.

10. Ватсон Б. C# 4.0 на примерах. – СПб.: БХВ-Петербург, 2011. – 608 с.

11. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2013. – 368 с.

12. Гвоздева, Т. В. Проектирование информационных систем: учеб. пособие / Т. В. Гвоздева, Т. А. Баллод. – Ростов н/Д: Феникс, 2009. – 508 с.

13. Данилин, А. Архитектура и стратегия. «Инь» и «Янь» информационных технологий предприятия / А. Данилин, А. Слюсаренко – М.: ИНТУИТ.ру, 2011 – 504 с.

14. C#: Пер. с англ. / Х. Дейтел, П. Дейтел, Дж. Листфорд, Т. Нието, Ш. Йегер, М. Златкина. – СПб.: БХВ-Петербург, 2006. – 1056 с.

15. Киммел П. UML. Основы визуального анализа и проектирования : пер. с англ. – М.: НТ Пресс, 2008. – 272 с.

16. Ларман, К. Применение UML и шаблонов проектирования. Практическое руководство. – 3-е изд.: пер с англ. – М.: ООО «И.Д Вильямс», 2013. – 736 с.

17. Мак-Дональд, М. Windows Presentation Foundation в .NET 4.0 с примерами на C# для профессионалов: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 1024 с.

18. Макки, А. Введение в .NET и Visual Studio 2010 для профессионалов: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 416 с.

19. Макконнелл, С. Профессиональная разработка программного обеспечения: пер. с англ. – СПб.: Символ-Плюс, 2006. – 240 с.

20. Мангано, С. XSLT. Сборник рецептов. – М.: ДМК Пресс, 2008. – 864 с.

21. Мартин, Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2010. – 464 с.

22. Мартин, Р. Принципы, паттерны и методики гибкой разработки на языке C# / Р. Мартин, М. Мартин. – пер. с англ. – СПб.: Символ-Плюс, 2011. – 768 с.

23. Разработка распределенных приложений на платформе Microsoft .NET Framework: Учебный курс Microsoft : пер. с англ. / С. Морган, Б. Райн, Ш. Хорн, М. Бломсма. – М.: Русская редакция; СПб.: Питер, 2008. – 608 с.

24. C# 4.0 и платформа .NET 4 для профессионалов: пер. с англ. / К. Нейгел, Б. Ивьен, Д. Глинн, К. Уотсон.– М.: ООО «И.Д. Вильямс», 2011. – 1440 с.

25. Нэш Т. C# 2010: ускоренный курс для профессионалов: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 592 с.

26. Орлов С. А. Технологии разработки программного обеспечения: учебник для вузов / С. А. Орлов, Б. Я. Цилькер. 4-е изд. – СПб.: Питер, 2012. – 608 с.

27. Павловская Т. А. C#. Программирование на языке высокого уровня: учебник для вузов. – СПб.: Питер, 2007. – 432 с.

28. Пауэрс Л. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл: пер. с англ. – СПб.: БХВ-Петербург, 2009. – 1200 с.

29. Петцольд Ч. Microsoft Windows Presentation Foundation: пер. с англ. – СПб.: Питер, 2008. – 944 с.

30. Подбельский В. В. Язык C# Базовый курс: учеб. пособие / В. В. Подбельский. – М.: Финансы и статистика; ИНФРА-М, 2011. – 384 с.

31. Ратц-мл. Д. С. LINQ: язык интегрированных запросов в C# 2008 для профессионалов : пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 560 с.

32. Резник С. Основы Windows Communication Foundation для .NET Framework 3.5: пер. с англ. / С. Резник, Р. Крейн, К. Боуэн. – М.: ДМК Пресс, 2008. – 480 с.

33. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 C#. 4-е изд. – СПб.: Питер, 2013. – 896 с.

34. Стилмен Э. Изучаем C# / Э. Стилмен, Дж. Грин. – 2-е изд. – СПб.: Питер, 2012. – 696 с.

35. Троелсен, Э. Язык программирования C# 5.0 и платформа.NET 4.5, 6-е изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2013. – 1312 с.

36. Фаулер, М. Архитектура корпоративных программных приложений: пер. с англ. – М.: ООО «И.Д. Вильямс», 2006. – 544 с.

37. Фленов, М. Е. Библия C#. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 560 с.

38. Фримен, Э. Паттерны проектирования / Э. Фримен, К. Сьера, Б. Бейтс. – СПб.: Питер, 2011. – 656 с.

39. Шилдт, Г. C# 4.0: полное руководство: пер. с англ. – М.:ООО «И.Д. Вильямс», 2011. – 1056 с.

40. Шаллоуей, А. Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию / А. Шаллоуей, Дж. Р. Тротт: пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 288 с.

41. Эванс, Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем: пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 448 с.

Интернет-ресурсы.

42. www.intuit.ru/department/itmngt/entarc/ – Курс «Архитектура предприятия»: авторы: А.В. Данилин, А.И. Слюсаренко; ИнТУИТ.

43. http://www.intuit.ru/studies/courses/1176/186/info – учебный курс «Языки информационного обмена», автор Кищенко О.

44. http://www.realcoding.net/teach/xml/ – иллюстрированный самоучитель по XML.

45. http://professorweb.ru/index.php – сайт по .NET и Web-программированию.

46. mxsmirnov.wordpress.com/ – Архитектура информационных систем. Максим Смирнов.

47. http://citforum.ru/SE/project/pattern/ – Обзор паттернов проектирования, Ольга Дубина.

48. http://www.fostas.ru/about/arch.php – Об архитектуре программных и информационных систем (работы фонда ФОСТАС, Зиндер Е.А.).

49. http://msdn.microsoft.com/ru-ru/library/ms123401.aspx – библиотека Microsoft Developers Network (MSDN) на русском языке.

50. http://metanit.com/index.php – сайт о программировании и IT-технологиях.

51. msdn.microsoft.com/ru-ru/library/ee895049/ – статьи по архитектуре информационных систем на MSDN.

52. http://www.w3.org/standards/xml/ – описание стандартов W3C для технологий XML (на английском языке).

53. http://c2.com/cgi/wiki?AntiPatternsCatalog – Большой иллюстрированный каталог антипатернов (на английском языке).

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, описание), 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).

Идентификаторы (коды домов, квартир и жильцов), единицы измерения и даты разместим в атрибутах. Описания домов расположим в секциях CDATA. Остальные данные будут являться содержимым элементов.

Дерево XML-документа представлено на рис. П.1. Код XML-документа, построенного в соответствии с деревом, приведён в листинге П.1.

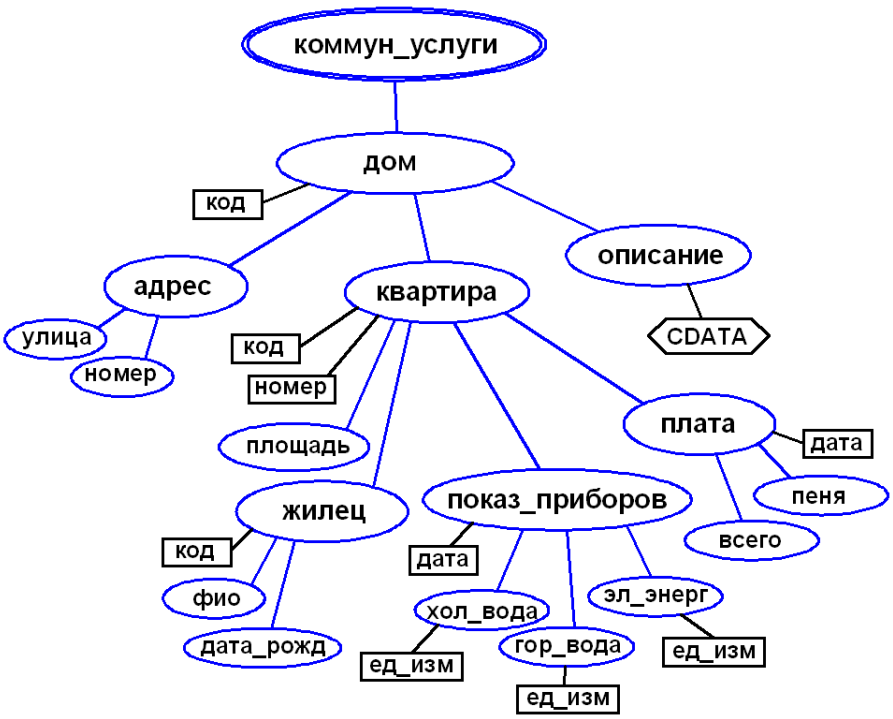


Рис. П.1. Дерево XML-документа

Листинг П.1. Код XML-документа «Коммунальные услуги»

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Данные о коммунальных услугах и их оплате -->
<коммун_услуги>
  <дом код="h18">
    <адрес>
      <улица>Волгоградская</улица>
      <номер>8</номер>
    </адрес>
    <описание>
      <![CDATA[Крупнопанельный 9-этажный дом. Построен в 1978 г.
Физический износ 14% (дата обслед. 15.06.2008 г.)]]>
    </описание>
    <квартира код="a234" номер="57">
      <площадь ед_изм="м2">28</площадь>
      <жилец код="c11568">
        <фио>Костенко Игорь Сергеевич</фио>
        <дата_рожд>1978-11-10</дата_рожд>
      </жилец>
      <показ_приборов дата="2014-05-02">
        <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="2013-05-04">
        <всего>1896.45</всего>
        <пеня>0</пеня>
      </плата>
    </квартира>
    <квартира код="a236" номер="59">
      <площадь ед_изм="м2">42</площадь>
      <жилец код="c27788">
        <фио>Соловьев Дмитрий Андреевич</фио>
        <дата_рожд>1988-07-22</дата_рожд>
      </жилец>
      <жилец код="c27789">
        <фио>Соловьева Елена Николаевна</фио>
        <дата_рожд>1989-10-03</дата_рожд>
      </жилец>
      <показ_приборов дата="2014-05-04">
        <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="2014-05-05">
        <всего>2396.45</всего>
        <пеня>0</пеня>
      </плата>
```

```
    </квартира>
  </дом>
  <дом код="h72">
    <адрес>
      <улица>Терешковой</улица>
      <номер>12</номер>
    </адрес>
    <описание>
      <![CDATA[Кирпичный 5-этажный дом. Построен в 1972 г. Физиче-
ский износ 18% (дата обслед. 24.08.2003 г.)]]>
    </описание>
    <квартира код="a358" номер="35">
      <площадь ед_изм="м2">36</площадь>
      <жилец код="c34670">
        <фио>Курганков Георгий Михайлович</фио>
        <дата_рожд>1972-02-26</дата_рожд>
      </жилец>
      <показ_приборов дата="2014-08-05">
        <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="2014-05-12">
        <всего>50896.56</всего>
        <пеня>4507.34</пеня>
      </плата>
    </квартира>
  </дом>
</коммун_услуги>
```

П.2. Некоторые физические формулы и константы

Уравнение состояния идеального газа (уравнение Менделеева-Клапейрона):

$$pV = \nu RT = \frac{m}{M} RT;$$

где p – давление газа; V – объём сосуда; ν – количество вещества; m – масса газа; M – молярная масса газа (например, для водорода M = 2,016 г/моль, для азота M = 28 г/моль, для кислорода M = 32 г/моль); T – температура газа; $R \approx 8{,}31$ Дж/(моль·К) – молярная газовая постоянная.

Гармонические колебания описываются уравнением:

$$x = A_x \cos(\omega t + \varphi_0);$$

где x – смещение тела от положения равновесия; A_x – амплитуда колебаний; ω – циклическая частота; φ_0 – начальная фаза.

Период колебаний T, частота ν и циклическая частота ω связаны соотношениями:

$$\nu = \frac{1}{T}; \qquad \omega = 2\pi\nu = \frac{2\pi}{T}.$$

Для гармонических колебаний тела под действием силы упругости F_x используются формулы:

$$F_x = -kx; \qquad \omega = \sqrt{\frac{k}{m}}.$$

где k – жёсткость деформируемого твёрдого тела (модуль Юнга).

Для математического маятника период колебаний определяется формулой:

$$T = 2\pi\sqrt{\frac{l}{g}};$$

где l – длина нерастяжимой нити; g – ускорение свободного падения.

Закон всемирного тяготения:

$$F = G\frac{m_1 \cdot m_2}{r^2};$$

где $G \approx 6{,}67 \cdot 10^{-11}$ Н·м^2·кг$^{-2}$ – гравитационная постоянная.

Закон Кулона:

$$F = \frac{1}{4\pi\varepsilon_0}\frac{q_1 \cdot q_2}{r^2};$$

где $\varepsilon_0 \approx 8{,}85 \cdot 10^{-12}$ Кл^2/Н·м^2 – электрическая постоянная в системе СИ.

Заряд на обкладках конденсатора q связан с напряжением между обкладками U следующим соотношением:

$$q = C \cdot U;$$

где C – электрическая ёмкость конденсатора.

Электроёмкость плоского конденсатора определяется по формуле:

$$C = \frac{\varepsilon_0 \varepsilon S}{d};$$

где S – площадь обкладок; d – расстояние между обкладками; ε – диэлектрическая проницаемость вещества между обкладками.