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

Методические указания к самостоятельной работе

Рекомендовано учебно-методической комиссией направления
подготовки бакалавра 09.03.02 (230400.62)
«Информационные системы и технологии»
в качестве электронного издания
для самостоятельной работы

Кемерово 2015

Рецензенты:

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

Турчин Денис Евгеньевич. Архитектура информационных систем: методические указания к самостоятельной работе [Электронный ресурс] для студентов направления подготовки бакалавра 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 самостоятельных практических работ, порядок и примеры их выполнения, а также контрольные вопросы к ним.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Пособие предназначено для студентов направления подготовки бакалавра 230400.62 «Информационные системы и технологии», изучающих дисциплину «Архитектура информационных систем».

Основной целью самостоятельной работы студентов является формирование умений, связанных с использованием технологий и средств разработки архитектуры информационных систем.

В качестве используемых средств и технологий выступают:

- интегрированная среда разработки программного обеспечения MS Visual Studio 2012 и язык программирования Visual C# 2012;
- языки написания XML-схем DTD и XSD;
- порождающие шаблоны объектно-ориентированного проектирования из каталога GoF;
- платформа разработки насыщенных клиентских приложений MS Windows Presentation Foundation (WPF) и язык XAML.

Выписка из ФГОС ВПО по направлению 230400.62

Код	**Компетенции, формируемые при освоении дисциплины**	**Результаты освоения**
Б2.Б.5	• **ПК-1** умение разрабатывать стратегии проектирования, определение целей проектирования, критериев эффективности, ограничений применимости; • **ПК-2** умение разрабатывать новые методы и средства проектирования информационных систем; • **ПК-9** умение проводить разработку и исследование методик анализа, синтеза, оптимизации и прогнозирования качества процессов функционирования информационных си-	***Знать***: • понятие архитектуры информационной системы и ее контекст (ПК-2); • основные предметные области архитектуры информационной системы (ПК-2); • методики описания архитектуры (ПК-9); • общую схему процесса разработки архитектуры информационной системы и методы управления этим процессом (ПК-1). ***Уметь***: • моделировать бизнес-процессы предприятия с использованием

	стем и технологий; • **ПК-10** умение осуществлять моделирование процессов и объектов на базе стандартных пакетов автоматизированного проектирования и исследований.	различных языков (ПК-10); • разрабатывать архитектурное описание информационной системы (ПК-6); • определять цели и задачи проектирования архитектуры информационной системы (ПК-1).

САМОСТОЯТЕЛЬНЫЕ ПРАКТИЧЕСКИЕ РАБОТЫ

1. ОСНОВЫ СОЗДАНИЯ XML-СХЕМ И ПРОВЕРКИ С ИХ ПОМОЩЬЮ ДОКУМЕНТОВ XML

1.1. Цель и задачи работы

Цель работы – прибрести умение создавать XML-схемы на языках DTD и XSD, а также осуществлять с их помощью проверку XML-документов.

Основные задачи работы:

• приобрести умение создавать определение типа документа DTD;

• освоить создание XML-схем с помощью языка XSD;

• научиться выполнять проверку XML-документов с помощью XML-схем на платформе .NET Framework.

Работа рассчитана на 6 часов.

1.2. Основные теоретические сведения

1.2.1. Общие сведения о XML-схемах. Определение типа документа (DTD). Инструкции DTD

Общие сведения о XML-схемах. Определение типа документа (DTD).

Программы, просматривающие XML-документ должны «понимать» вложенность и порядок следования элементов, число и тип атрибутов, а также другие характеристики. Для этого XML-документ должен быть снабжен формализованным описанием, понятным для программы-анализатора. Такое описание называется ***XML-схемой*** и содержит метаданные, то есть данные о данных.

XML-схемы предназначены для решения следующих задач:

• дать формальное описание (шаблон) XML-документа;

• предоставить список элементов и атрибутов в словаре;

• связать типы данных (целочисленный, вещественный, строковый и др.) со значениями в документе;

• предоставить документацию, понимаемую как человеком, так и машиной.

Роль XML-схемы аналогична роли схемы базы данных.

В настоящее время существует несколько широко применяемых языков описания схемы, главными из которых являются DTD и XSD. Другими известными схемами являются RELAX NG и Schematron.

DTD (англ. *Document Type Definition* – определение типа документа) – язык, который используется для записи метаданных в документах XML и SGML. Также под определением DTD понимают XML-схему, написанную на соответствующем языке.

Подход с использованием DTD унаследован от языка SGML, в котором для описания DTD используется грамматика EBNF (англ. *Extended Backus-Naur Form*).

Схема DTD задается в XML-документе с помощью ***объявления DOCTYPE***, которое записывается сразу после объявления XML и связывает DTD с XML-документом.

Самое простое объявление DOCTYPE определяет только корневой элемент документа:

```
<!DOCTYPE корн_элем>
```

где **корн_элем** – имя корневого элемента.

Чаще всего объявление DOCTYPE ссылается на ***внешнюю схему DTD***, расположенную в отдельном текстовом документе с расширением **.dtd**. В этом случае объявление DOCTYPE имеет следующий вид:

```
<!DOCTYPE корн_элем SYSTEM "URL">
```

где **URL** – указатель ресурса, задающий путь к внешней схеме DTD.

Объявление DOCTYPE также может содержать в себе ***внутреннюю схему DTD*** без ссылки на внешние файлы. При этом объявление DOCTYPE записывается следующим образом:

```
<!DOCTYPE корн_элем [
  разметка_DTD
```

```
]>
```

где **разметка_DTD** – объявления разметки внутренней схемы DTD.

Пример XML-документа с внутренней схемой DTD представлен в листинге 1.1.

Листинг 1.1. Код XML-документа с DTD

```
<?xml version="1.0" encoding="utf-8" standalone="yes">
<!DOCTYPE recipe [
  <!ELEMENT recipe (title, ingredients, preparation,
serving?)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT ingredients (item+)>
  <!ELEMENT item (#PCDATA)>
  <!ATTLIST item quantity CDATA #IMPLIED>
  <!ELEMENT preparation (#PCDATA)>
  <!ELEMENT serving (#PCDATA)>
]>
<recipe>
  <title>Каша пшенная</title>
  <ingredients>
    <item quantity="100 г">Крупа пшенная</item>
    <item quantity="650 мл">Молоко</item>
    <item quantity="20 г">Масло сливочное</item>
  </ingredients>
  <preparation>
  <![CDATA[Крупу промыть, засыпать в кастрюлю и залить молоком.
Добавить сливочное масло, сахар и соль по вкусу и перемешать.
Закрыть кастрюлю крышкой и варить на слабом огне около 40 минут]]>
  </preparation>
  <serving>
  <![CDATA[При подаче на стол можно полить сверху 1-2 ст. ложки
сгущенки]]>
</serving>
</recipe>
```

К основным недостаткам определений DTD относятся:

- использование синтаксиса, отличного от XML, что усложняет разработку и чтение DTD, а также их преобразование в другие языки разметки (например, в HTML);
- отсутствие поддержки типов данных, что затрудняет проверку на соответствие стандартам в приложениях, использующих

другие типы данных (финансовые транзакции, обмен научными данными и др.);

• отсутствие поддержки пространств имён, что может привести к конфликтам имён при обработке.

Объявления разметки DTD. Инструкции ELEMENT и ATTLIST.

Схема DTD состоит из ***объявлений разметки***, начинающихся парой символов «<!» и заканчивающихся символом «>». Между этими символами может быть записано одна из следующих инструкций:

• **ELEMENT** – объявляет элемент и его допустимые дочерние элементы;

• **ATTLIST** – объявляет список атрибутов; эти атрибуты определяются именем, типом данных, значениями по умолчанию;

• **ENTITY** – определяет сущность;

• **NOTATION** – определяет нотации, позволяющие XML-документу передавать внешним приложениям уведомляющие сведения.

Инструкция **ELEMENT** служит для объявления элементов и имеет следующий синтаксис:

```
<!ELEMENT имя_элем модель_содерж>
```

где **имя_элем** – имя объявляемого элемента;

модель_содерж – модель содержимого для элемента, которая выбирается из следующих вариантов:

• **ANY** – внутри элемента допускается любое содержимое (неограниченная модель содержимого);

• **EMPTY** – в элементе не допускается содержимое, т. е. элемент должен оставаться пустым;

• **(**правило**)** – специальное правило, заключаемое в круглые скобки.

При построении правил модели содержимого используют следующие зарезервированные ключевые слова и символы:

• **#PCDATA** – указывает, что содержимое элемента является символьными данными;

• «**,**» (запятая) – указывает на порядок отображения дочерних элементов. Например, элемент **book** должен содержать элемент **author**, за которым следует элемент **title**:

```
<!ELEMENT book (author, title)>
```

• «|» (вертикальная черта) – указывает, что любой из элементов, расположенных слева и справа от «|», может отображаться как дочерний элемент. Например, элемент **fruit** содержит элемент **apple** или элемент **orange**:

```
<!ELEMENT fruit (apple|orange)>
```

• «**?**» (вопросительный знак) – указывает, что элемент является необязательным и может использоваться только один раз в данном фрагменте XML-документа. Например, элемент **memo** должен содержать элемент **body**, за которым необязательно должен следовать элемент **postscript**:

```
<!ELEMENT memo (body, postscript?)>
```

• «+» (знак плюс) – указывает, что элемент является обязательным и может использоваться более одного раза в данном фрагменте XML-документа. Например, элемент **catalog** должен содержать один или более элемент **book**:

```
<!ELEMENT catalog (book+)>
```

• «*****» (звездочка) – указывает, что элемент является необязательным и может использоваться более одного раза в данном фрагменте XML-документа. Например, элемент **table** может быть пустым, либо содержать элементы **rowset**:

```
<!ELEMENT table (rowset*)>
```

Инструкция ATTLIST.

Инструкция **ATTLIST** служит для задания списка атрибутов определенного элемента и имеет следующий синтаксис:

```
<!ATTLIST имя_элем
  имя_атриб1 ТИП ЗНАЧ_УМОЛЧ
  имя_атриб2 ТИП ЗНАЧ_УМОЛЧ
        ...
  имя_атрибN ТИП ЗНАЧ_УМОЛЧ>
```

где **имя_элем** – имя элемента, к которому относится список атрибутов; **имя_атриб** – имя атрибута заданного элемента; **ТИП** – тип атрибута; **ЗНАЧ_УМОЛЧ** – значение атрибута по умолчанию.

Параметры **имя_атриб**, **ТИП** и **ЗНАЧ_ УМОЛЧ** определяют каждый атрибут в списке. Их можно повторять столько раз, сколько нужно, пока не будут перечислены и определены все атрибуты, доступные для данного элемента.

Тип атрибута может иметь одно из следующих значений:

- **CDATA** – атрибут содержит только символьные данные;
- **ID** – значение атрибута должно быть уникальным; оно не может повторяться в других элементах или атрибутах данного документа;
- **IDREF** – атрибут ссылается на значение другого атрибута типа **ID** из данного документа;
- **ENTITY** – значение атрибута должно соответствовать имени внешней сущности;
- (список) – значения атрибута ограничены перечисленными в списке, который заключен в круглые скобки; каждое значение отделяется вертикальной чертой «|».

Для атрибута возможными значения по умолчанию являются:

- **#REQUIRED** – атрибут обязательно должен присутствовать в XML-документе;
- **#IMPLIED** – атрибут может присутствовать в XML-документе, но его отсутствие не вызывает ошибки;
- **#FIXED** – значение атрибута зафиксировано в определении DTD; изменить его в XML-документе нельзя; при использовании этого ключевого слова непосредственно за ним нужно обязательно записать постоянное значение атрибута.

В данном примере для элемента **книга** объявляются следующие атрибуты:

- **ISBN** – обязательный атрибут, который должен содержать уникальное значение-идентификатор для каждого элемента **книга** в XML-документе;
- **издательство** – необязательный атрибут, который может содержать только символьные данные;

• **магазин** – фиксированный атрибут, для которого задано значение «Книжный Мир»;

• **в_печати** – обязательный атрибут, который должен содержать либо значение «да», либо значение «нет»; по умолчанию, если значение не задано явно в XML-документе, используется значение «да».

```
<!ATTLIST книга
    ISBN          ID        #REQUIRED
    издательство  CDATA     #IMPLIED
    магазин       CDATA     #FIXED     "Книжный Мир"
    в_печати      (да|нет)  "да"
>
```

❒ ***Пример* 1.1. *Разработка внешнего определения DTD.***

Требуется создать внешнее определение DTD для проверки документа XML, приведённого в приложении П.1.

Введём следующие требования к XML-документу, предъявляемые со стороны DTD:

• элемент **дом** является обязательным (ЖКК обслуживает хотя бы один дом) и может встречаться более одного раза; дочерние элементы должны быть записаны в следующем порядке: **адрес**, **описание**, **квартира**;

• элемент **квартира** является обязательным (в доме обслуживается хотя бы одна квартира); и может встречаться более одного раза; дочерние элементы должны идти в следующем порядке: **площадь**, **жилец**, **показ_приборов**, **плата**; обязательным атрибутом элемента **квартира** является **номер**; в элементе **площадь** присутствует обязательный атрибут **ед_изм**, который принимает фиксированное значение «м2»;

• элемент **жилец** не является обязательным в родительском элементе **квартира** (например, в квартире никто не прописан) и может встречаться любое число раз; в элементе **жилец** должны присутствовать дочерние элементы в следующем порядке: **фио**, **дата_рожд**;

• в элементе **показ_приборов** дочерние элементы располагаются в приведённом порядке: **хол_вода**, **гор_вода**, **эл_энерг**; эти дочерние элементы являются необязательными (например, в квартире не установлены счётчики или вовремя не переданы их

показания) и могут встречаться только один раз; необязательным атрибутом для элемента **показ_приборов** является **дата**; при наличии элементов **хол_вода** и **гор_вода** в них должен присутствовать атрибут **ед_изм**, который может принимать значение «м3» или «л»;

• в элементе **плата** используется следующий порядок расположения дочерних элементов: **всего**, **пеня**; обязательным атрибутом для элемента **плата** является **дата**;

• все атрибуты **код** являются обязательными и должны иметь уникальное значение.

Код полученного определения DTD представлен в листинге 1.2.

Листинг 1.2. Код определения DTD

```
<!ELEMENT коммун_услуги (дом+)>
<!ELEMENT дом (адрес, описание?, квартира+)>
<!ATTLIST дом
    код ID #REQUIRED>
<!ELEMENT адрес (улица, номер)>
<!ELEMENT улица (#PCDATA)>
<!ELEMENT номер (#PCDATA)>
<!ELEMENT описание (#PCDATA)>
<!ELEMENT квартира (площадь, жилец*, показ_приборов, плата)>
<!ATTLIST квартира
    код ID #REQUIRED
    номер CDATA #REQUIRED>
<!ELEMENT площадь (#PCDATA)>
<!ATTLIST площадь
    ед_изм CDATA #FIXED "м2">
<!ELEMENT жилец (фио, дата_рожд)>
<!ATTLIST жилец
    код ID #REQUIRED>
<!ELEMENT фио (#PCDATA)>
<!ELEMENT дата_рожд (#PCDATA)>
<!ELEMENT показ_приборов (хол_вода?, гор_вода?, эл_энерг?)>
<!ATTLIST показ_приборов
    дата CDATA #IMPLIED>
<!ELEMENT хол_вода (#PCDATA)>
<!ATTLIST хол_вода
    ед_изм (м3|л) "м3">
<!ELEMENT гор_вода (#PCDATA)>
<!ATTLIST гор_вода
    ед_изм (м3|л) "м3">
<!ELEMENT эл_энерг (#PCDATA)>
```

```
<!ATTLIST эл_энерг
    ед_изм CDATA #FIXED "квтч">
<!ELEMENT плата (всего, пеня)>
<!ATTLIST плата
    дата CDATA #REQUIRED>
<!ELEMENT всего (#PCDATA)>
<!ELEMENT пеня (#PCDATA)>
```

Для связывания XML-документа с внешним определением DTD необходимо добавить в код документа объявление **DOCTYPE**:

```
<?xml version="1.0" encoding="utf-8"? standalone="no"?>
<!DOCTYPE коммун_услуги SYSTEM "H:\DTD\CommService.dtd">
<коммун_услуги>
    ...
</коммун_услуги>
```

Внешнее определение типа документа размещено в файле «CommService.dtd». ❒

1.2.2. Общие сведения о схеме XSD. Встроенные типы XSD. Связывание документа XML с XSD-схемой

Общие сведения о схеме XSD.

В настоящее время описание структуры XML-документа, выполняемое средствами DTD, не удовлетворяет многих разработчиков. Для решения различных задач требуется более точное описание метаданных, учитывающее тип содержимого элемента, количество повторений и порядок следования вложенного элемента, а также другие подробности.

По указанным причинам в 2001 году консорциум W3C предложил рекомендацию ***XML Schema* 1.0**. В данном стандарте описывается язык ***XSD*** (англ. *XML Schema Definition Language*) – язык определения XML-схем.

Окончательная редакция версии 1.0 для XML Schema была опубликована 28 октября 2004 г. В настоящее время идёт работа над версией 1.1.

XML-схема на языке XSD называется ***XSD-схемой*** и представляет собой текстовый документ с расширением «**.xsd**». Синтаксис XSD-схемы соответствует языку XML.

Для того чтобы выделить элементы, которые принадлежат языку XSD, применяют пространство имён с идентификатором **http://www.w3.org/2001/XMLSchema**. Общепринятыми префиксами этого пространства имён являются **xs** и **xsd** (в среде Visual Studio по умолчанию используется префикс **xs**).

Корневой элемент схемы XSD называется **xs:schema**. Элемент **xs:schema** содержит элементы верхнего уровня, определяющие структуру XML-документа.

В корневом элементе **xs:schema** может быть указан атрибут **targetNamespace**, который задаёт целевое пространство имён. Под ***целевым пространством имён*** понимают пространство имён, которое показывает XML-процессору, какую схему использовать для проверки конкретного документа XML.

При создании XSD-схем в Visual Studio для целевого пространства имён по умолчанию используется идентификатор **http://tempuri.org/XMLSchema.xsd**.

Таким образом, запись корневого элемента схемы вместе с пространствами имён может иметь следующий вид:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- Здесь располагается содержимое XML-схемы -->
</xs:schema>
```

Язык XSD различает сложные и простые элементы XML. ***Простыми элементами*** считаются элементы, не содержащие атрибутов и вложенных элементов. Соответственно, ***сложные элементы*** содержат атрибуты и вложенные элементы.

Схема определяет ***простые типы*** – типы простых элементов, и ***сложные типы*** – типы сложных элементов.

Встроенные типы XSD.

Язык XSD содержит множество встроенных простых типов, которые позволяют записывать числа, дату и время, строки символов, логические значения, адреса URI. Кроме того, существует множество встроенных производных типов.

Наиболее часто используемые встроенные типы данных перечислены в табл. 1.1.

Таблица 1.1

Основные встроенные типы данных схемы XSD

Тип XSD	Описание	Пример
	Простые типы	
xs:string	Строки символов	Образец Строки
xs:normalizedString	Строки без пробелов	ОбразецСтроки
xs:boolean	Логические значения	true, false, 1, 0
xs:float	Вещественные 32-разрядные числа с плавающей точкой	12, -1.222, 12.34E-5
xs:double	Вещественные 64-разрядные числа с плавающей точкой	12, -1.222, 12.34E-5
xs:decimal	Вещественные числа с фиксированной точкой	12, -1.222, 7000.00
xs:duration	Интервалы времени	P1Y1M4DT10H50M11.7S
xs:dateTime	Моменты времени	2014-05-28T11:15:00.000-03:00
xs:time	Моменты времени, повторяющиеся ежедневно	11:15:00.000
xs:date	Даты	2014-05-28
xs:anyURI	Адреса URI	http://tempuri.org
xs:QName	Квалифицированное имя, которое состоит из префикса и локального имени, разделенных двоеточием	doc:message
	Производные типы	
xs:integer	Целые числа (является производным от типа **decimal**)	123456, -12, 10
xs:positiveInteger	Целые числа от 1 до 2147483647 (является производным от типа **integer**)	123456, 12, 10
xs:long	Целые числа от -9223372036854775808 до 9223372036854775807 (является производным от типа **integer**)	-1234, 12345678901234
xs:int	Целые числа от -	123456, -12, 10

Тип XSD	Описание	Пример
	2147483648 до 2147483647 (является производным от типа **long**)	
xs:short	Целые числа от -32768 до 32767 (является производным от типа **int**)	-12, 1234
xs:byte	Целые числа от -128 до 127 (является производным от типа **short**)	1, -10, 125

Схема XSD также поддерживает пользовательские типы данных, производные от встроенных типов. Для задания пользовательских типов данных используется элемент **xs:comlexType**.

Связывание документа XML с XSD-схемой.

Документ, проверяемый с помощью схемы, также должен содержать объявление пространства имен. Пространство имен всегда указывается в корневом элементе XML-документа с помощью атрибута **xmlns**. По общему соглашению для этого пространства имен используется префикс **xsi** (*XML-schema instance*):

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Схему документа можно указать прямо в документе. Это делается одним из двух способов:

• Если элементы документа не принадлежат никакому пространству имен и записаны без префикса, то в корневом элементе документа записывается атрибут **noNamespaceSchemaLocation**, указывающий расположение файла со схемой в форме URI:

```
xsi:noNamespaceSchemaLocation="URI"
```

• Если элементы документа относятся к некоторому пространству имен, то применяется атрибут **schemaLocation** корневого элемента. В этом атрибуте через пробел перечисляются пространство имён и расположение файла со схемой:

```
xsi:SchemaLocation="NameSpace URI"
```

Пример XML-схемы на языке XSD приведён в листинге 1.3. а XML-документ, соответствующий данной схеме, представлен в листинге 1.4.

Листинг 1.3. Код XML-схемы на языке XSD (**simple.xsd**)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="order">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="good" minOccurs="1"
    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" />
              <xs:element name="amount" type="xs:int" />
              <xs:element name="price">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:decimal">
                      <xs:attribute name="discount"
    type="xs:int" use="optional" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xs:integer"
    use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Листинг 1.4. Код документа XML, соответствующего схеме из листинга 1.3

```
<?xml version="1.0" encoding="utf-8"?>
<order
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="H:\XSD\simple.xsd">
```

```
  <good id="12503">
    <name>Коробка конфет</name>
    <amount>2</amount>
    <price discount="15">123.50</price>
  </good>
  <good id="14097">
    <name>Бутылка вина</name>
    <amount>1</amount>
    <price>210.00</price>
  </good>
</order>
```

1.2.3. Объявление элементов и их атрибутов в схеме XSD. Определение сложных типов XSD

Объявление элементов и атрибутов в схеме XSD.

Элементы, из которых состоит XML-документ, объявляются с помощью элемента **xs:element** следующим образом:

```
<xs:element name="имя_элемента" type="тип_элемента"
minOccurs="число_вхождений" maxOccurs="число_вхождений" />
```

Атрибуты **minOccurs** и **maxOccurs** задают наименьшее и наибольшее число вхождений элемента (по умолчанию имеют значение 1) и являются необязательными. Атрибут **maxOccurs** также может принимать значение **unbounded** (неограниченно).

Пример описания простого элемента **автор**, содержащего фамилию и инициалы автора книги:

```
<xs:element name="автор" type="xs:string"
    minOccurs="1" maxOccurs="unbounded" />
```

Указание типа элемента в атрибуте **type** удобно, если это встроенный тип или тип, определенный пользователем. В этом случае в атрибуте **type** можно записать только имя элемента.

Если же тип элемента определяется здесь же, то определение типа элемента лучше вынести в содержимое элемента **xs:element**:

```
<xs:element name="имя_элем">
    Определение типа элемента
</xs:element>
```

Объявление атрибута элемента производится с помощью элемента **xs:attribute**:

```
<xs:attribute name="имя_атрибута" type="тип_атрибута"
    use="обязательность" default="значение_по_умолч" />
```

Необязательный атрибут **use** принимает три значения:

• **optional** – описываемый атрибут необязателен (значение по умолчанию);

• **required** – описываемый атрибут обязателен.

Пример определения атрибута **id**, содержащего уникальные целочисленные значения:

```
<xs:attribute name="id" type="xs:long" use="required" />
```

Определение типа атрибута можно вынести в содержимое элемента **xs:attribute**:

```
<xs:attribute name="имя_атрибута">
    Определение типа атрибута
</xs:attribute>
```

Элемент **xs:annotation** определяет заметку по схеме XSD. Сами сведения записываются в содержимом элемента **xs:documentation**, вложенном в **xs:annotation**. Информация, размещаемая в **xs:documentation**, не используется при проверке.

```
<xs:annotation>
  <xs:documentation>
    Успеваемость студентов КузГТУ
    Copyright 2014. KuzSTU. All right reserved
  </xs:documentation>
</xs:annotation>
```

Определение сложных типов в схеме XSD.

Тип элемента называется ***сложным***, если в элемент вложены другие элементы и/или в открывающем теге элемента есть атрибуты.

Сложный тип определяется элементом **xs:complexType**, имеющим следующий синтаксис:

```
<xs:complexType name="имя_типа">
    Определение типа
</xs:complexType>
```

В содержимом элемента **xs:complexType** описываются элементы, входящие в сложный тип, и/или атрибуты открывающего тега.

Определения сложных типов можно разделить на три группы:

• определение ***типа пустого элемента***;
• определение ***типа элемента с простым телом***;
• определение ***типа элемента со сложным телом***.

Наиболее просто определяется ***тип пустого элемента*** – элемента, задаваемого только одним тегом с атрибутами в нем. В этом каждый атрибут определяется одним элементом **xs:attribute**.

Например, определим тип **imageType** пустого элемента **image** с одним атрибутом **href**:

```
<xs:complexType name="imageType">
  <xs:attribute name="href" type="xs:anyURI" />
</xs:complexType>
```

После этого определения можно в схеме XSD объявить элемент **image** типа **imageType**:

```
<xs:element name="image" type="imageType" />
```

В XML-документе данный элемент может выглядеть следующим образом:

```
<image href="http://www.kuzstu.ru/images/ITstudent.jpg" />
```

Более сложным является описание ***типа элемента с простым телом*** – элемента, имеющего символьное содержание и атрибуты в открывающем теге. Данный тип определяется элементом **xs:simpleContent**.

В теле элемента **xs:simpleContent** должен быть либо элемент **xs:restriction** (служит для указания ограничений на определение типа), либо элемент **xs:extension** (расширяет определение типа путем добавления атрибутов). Атрибут **base** элементов **xs:extension** и **xs:restriction** задает простой тип тела описываемого элемента.

Например, определим тип **calcResultType** для элемента **result** (результат вычислений), содержимое которого представляет значение встроенного простого типа **xs:decimal**. Простой тип

расширяется тем, что к нему добавляются атрибуты **unit** (единица) и **precision** (точность):

```
<xs:complexType name="calcResultType">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="unit" type="xs:string" />
      <xs:attribute name="precision"
       type="xs:nonNegativeInteger" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

В схеме XSD объявление элемента **result** запишется следующим образом:

```
<xs:element name="result" type="calcResultType"/>
```

В XML-документе элемент **result** может иметь следующий вид:

```
<result unit="см" precision="2">123.45</result>
```

Описание ***типа элемента со сложным телом*** требуется в том случае, когда элемент содержит вложенные элементы. В этом случае для описания сложного типа необходимо выбрать модель группы вложенных элементов.

Вложенные элементы, составляющие определяемый сложный тип, могут появляться или в определенном порядке, или в произвольном порядке, кроме того можно выбирать только один из перечисленных элементов. Эта возможность называется ***моделью группы*** (*model group*) вложенных элементов и определяется одним из трех элементов:

- **xs:sequence** – применяется в том случае, когда перечисляемые элементы должны записываться в документе в определенном порядке;
- **xs:all** – используется, если элементы можно перечислить в произвольном порядке; в данном компоненте не допускается использование элементов **xs:sequence** и **xs:choice**;
- **xs:choice** – применяется, когда необходимо выбрать один из нескольких элементов.

Например, требуется описать книгу (элемент **book**), сведения о которой должны записываться в определенном порядке: автор (**author**), заголовок (**title**), издательство (**publisher**), число

страниц (**pages**). Кроме того, вместо автора у книги может быть задан редактор (**redactor**):

```
<xs:complexType name="bookType">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="author" type="xs:normalizedString"
       minOccurs="0" />
      <xs:element name="redactor"
       type="xs:normalizedString" />
    </xs:choice>
    <xs:element name="title" type="xs:normalizedString" />
    <xs:element name="pages" type="xs:positiveInteger"
     minOccurs="0" />
    <xs:element name="publisher"
     type="xs:normalizedString" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

Работа с XML-схемами в Visual Studio. Конструктор XML-схем.

Для создания новой XML-схемы через Visual Studio необходимо в меню **File** выбрать **New File**. В открывшемся окне шаблонов (рис. 1.1) следует выбрать шаблон **XML Schema** (XML-схема) и нажать кнопку **Open** (Открыть).

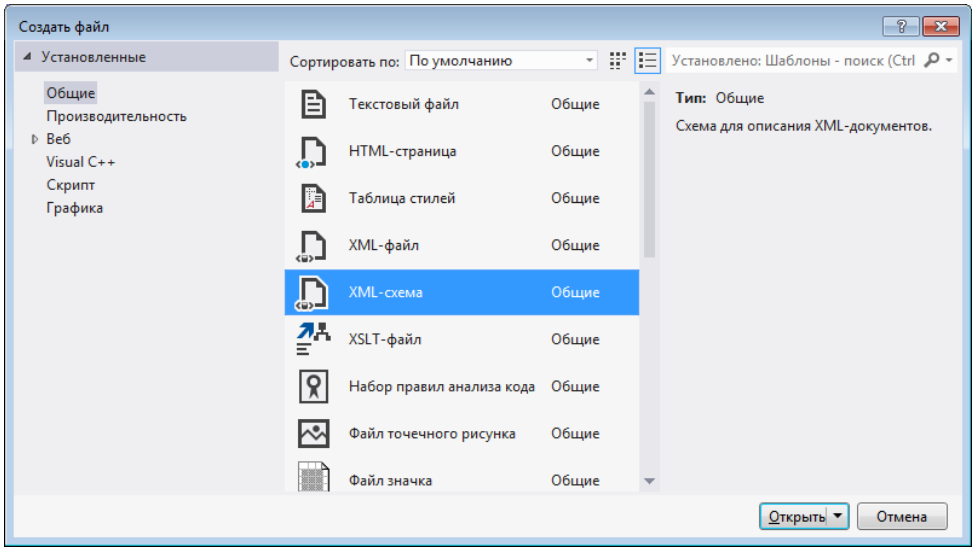

Рис. 1.1. Окно **Создать файл** с выбранным элементом XML-схема (Visual Studio 2012)

После создания нового файла откроется визуальный конструктор XML-схем (рис. 1.2), состоящий из рабочей области конструктора и обозревателя XML-схем.

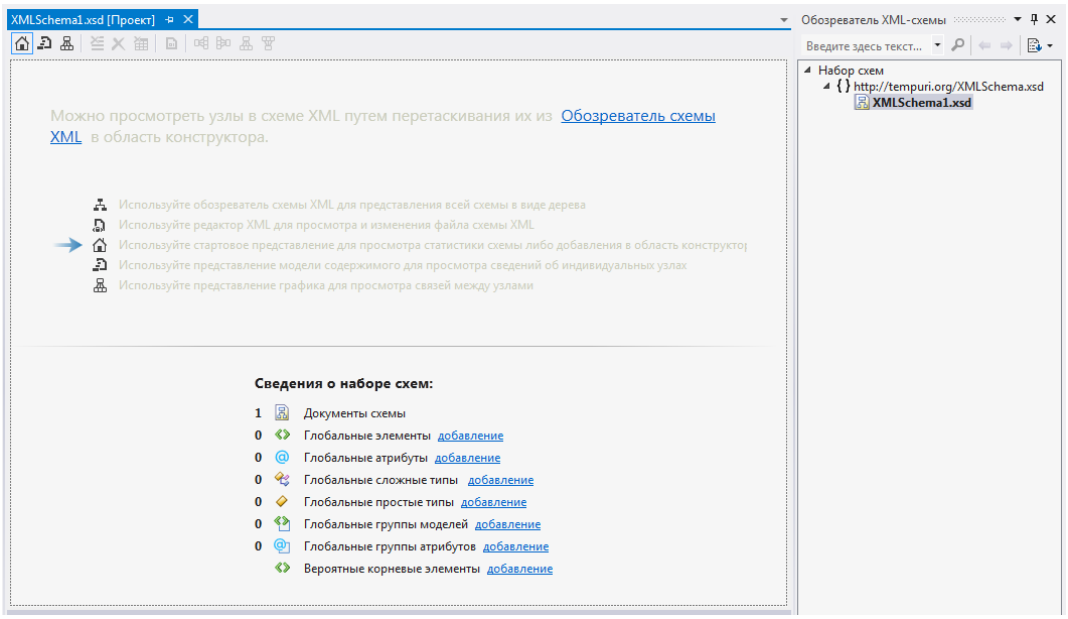

Рис. 1.2. Окно конструктора XML-схем при создании новой схемы (Visual Studio 2012)

Для перехода к коду XML-схемы следует выбрать в окне обозревателя узел схемы (**XMLSchema1.xsd** для новой схемы) и сделать по нему двойной клик левой кнопкой мыши. В результате откроется окно текстового редактора XML-схем.

Обозреватель XML-схем представляет иерархический вид схемы в виде дерева.

Конструктор XML-схем является графическим инструментом, позволяющим просматривать XML-схемы. Помимо обозревателя схем XML, позволяющего осуществлять поиск и навигацию по дереву схемы XML, конструктор предоставляет графическое представление подробных сведений о локальных и глобальных узлах схемы, включая простые и сложные типы, элементы, группы, атрибуты и группы атрибутов (рис. 1.3).

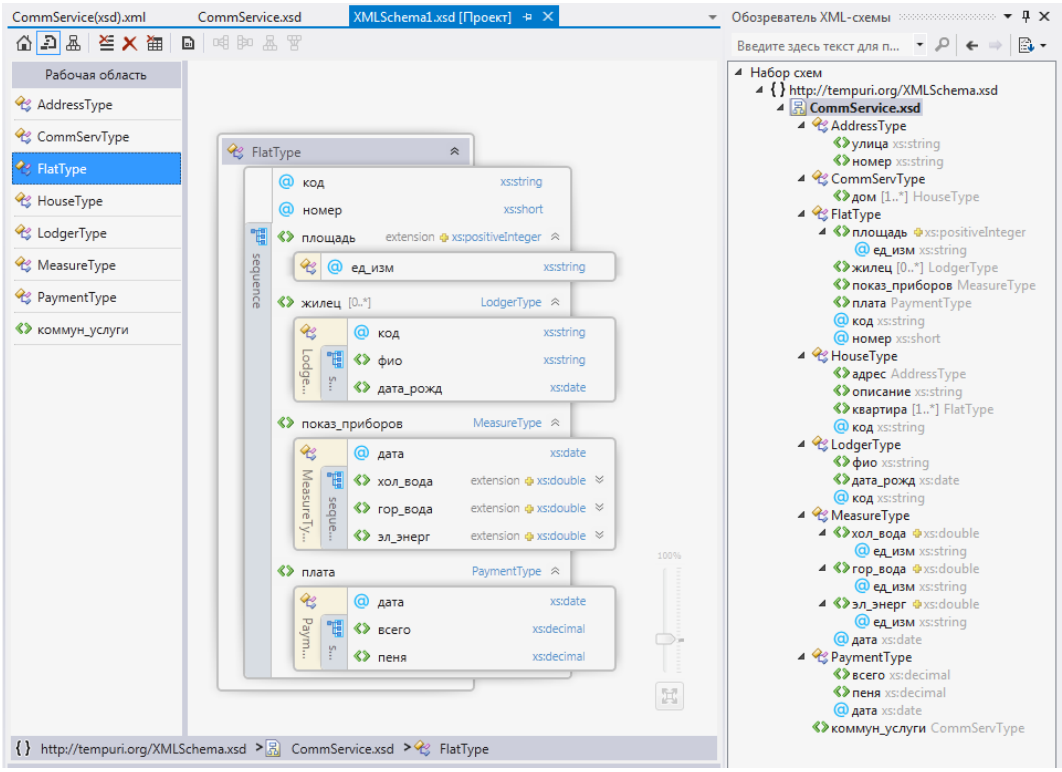

Рис. 1.3. Просмотр узлов XML-схемы в конструкторе (Visual Studio 2012)

❒ ***Пример* 1.2. *Разработка XML-схемы на языке XSD.***

Требуется разработать XML-схему на языке XSD для проверки документа XML, приведённого в приложении П.1.

Для разработки XML-схемы воспользуемся ограничениями, введёнными в примере 1.1 при создании DTD. Дополнительно зададим типы данных для содержимого простых элементов и значений атрибутов, используя встроенные типы XSD (табл. 1.1).

Примем для узлов XML-схемы следующие встроенные типы данных XSD:

- **xs:string** – элементы: **описание**, **улица**, **номер** (адрес дома), **фио**; атрибуты: **код** (для всех элементов), **ед_изм**;
- **xs:pstiveInteger** – элемент **площадь**;
- **xs:short** – атрибут **номер** (квартиры);
- **xs:date** – элемент **дата_рожд**; атрибуты **дата**;
- **xs:double** – элементы: **хол_вода**, **гор_вода**, **эл_энерг**;
- **xs:decimal** – элементы: **всего**, **пеня**.

Для остальных элементов зададим пользовательские сложные типы.

Код полученной XML-схемы представлен в листинге 1.5.

Листинг 1.5. Код XML-схемы (**CommService.xsd**)

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    targetNamespace="http://tempuri.org/XMLSchema.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/XMLSchema.xsd"
    xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
  <!-- Определение аннотации XML-схемы -->
  <xs:annotation>
    <xs:documentation xml:lang="ru-ru">
      XML-схема. Коммунальные услуги и их оплата.
      Copyright 2014. Turchin Denis. All rights reserved
    </xs:documentation>
  </xs:annotation>
  <!-- Определение корневого элемента коммун_услуги -->
  <xs:element name="коммун_услуги" type="CommServType" />
  <!-- Определение сложного типа CommServType -->
  <xs:complexType name="CommServType">
    <xs:sequence>
      <xs:element name="дом" type="HouseType"
                  minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <!-- Определение сложного типа HouseType -->
  <xs:complexType name="HouseType">
    <xs:sequence>
      <xs:element name="адрес" type="AddressType" />
      <xs:element name="описание" type="xs:string" />
      <xs:element name="квартира" type="FlatType"
                  minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="код" type="xs:string" use="required" />
  </xs:complexType>
  <!-- Определение сложного типа addressType -->
  <xs:complexType name="AddressType">
    <xs:sequence>
      <xs:element name="улица" type="xs:string" />
      <xs:element name="номер" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <!-- Определение сложного типа FlatType -->
```

```
  <xs:complexType name="FlatType">
    <xs:sequence>
      <xs:element name="площадь">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:positiveInteger">
              <xs:attribute name="ед_изм" type="xs:string"
fixed="м2" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="жилец" type="LodgerType"
                  minOccurs="0" maxOccurs="unbounded" />
      <xs:element name="показ_приборов" type="MeasureType" />
      <xs:element name="плата" type="PaymentType" />
    </xs:sequence>
    <xs:attribute name="код" type="xs:string" use="required" />
    <xs:attribute name="номер" type="xs:short" />
  </xs:complexType>
  <!-- Определение сложного типа LodgerType -->
  <xs:complexType name="LodgerType">
    <xs:sequence>
      <xs:element name="фио" type="xs:string" />
      <xs:element name="дата_рожд" type="xs:date" />
    </xs:sequence>
    <xs:attribute name="код" type="xs:string" use="required"  />
  </xs:complexType>
  <!-- Определение сложного типа MeasureType -->
  <xs:complexType name="MeasureType">
    <xs:sequence>
      <xs:element name="хол_вода">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:double">
              <xs:attribute name="ед_изм" type="xs:string" de-
fault="м3" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="гор_вода">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:double">
              <xs:attribute name="ед_изм" type="xs:string" de-
fault="м3" />
            </xs:extension>
          </xs:simpleContent>
```

```
        </xs:complexType>
      </xs:element>
      <xs:element name="эл_энерг">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:double">
              <xs:attribute name="ед_изм" type="xs:string"
fixed="квтч" />
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="дата" type="xs:date" />
  </xs:complexType>
  <!-- Определение сложного типа PaymentType -->
  <xs:complexType name="PaymentType">
    <xs:sequence>
      <xs:element name="всего" type="xs:decimal" />
      <xs:element name="пеня" type="xs:decimal" />
    </xs:sequence>
    <xs:attribute name="дата" type="xs:date" />
  </xs:complexType>
</xs:schema>
```

Для связывания документа и схемы добавим в корневой элемент документа стандартное пространство имен **xsi** и атрибут **noNamespaceSchemaLocation**:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Данные о коммунальных услугах и их оплате -->
<коммун_услуги
  xmlns="http://tempuri.org/XMLSchema.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation ="H:\XSD\CommService.xsd">
    ...
</коммун_услуги>
```

Полученная XML-схема на языке XSD сохранена в файле «**CommService.xsd**». ❐

1.2.4. Работа с XML-схемами на платформе .NET Framework

Работа с XML-схемами на платформе .NET Framework.

Платформа .NET Framework предоставляет набор классов для работы с XML-схемами на языках DTD и XSD и проверки XML-документов с помощью схем.

Для выполнения действий со схемой XSD используется набор классов в пространстве имен **System.Xml.Schema**, называемый ***объектной моделью схемы SOM*** (англ. *Schema Object Model*).

Модель SOM позволяет считывать схему XSD из файла или программно создавать схему в памяти. После этого схему XSD можно просматривать, изменять, проверять или записывать в файл. SOM играет ту же роль для схем XSD, что и DOM для XML-документов.

Для проверки XML-документа на соответствие DTD и XSD может использоваться класс **XmlReader**. При этом необходимо настроить свойства экземпляра класса **XmlReaderSettings**, который может передаваться как параметр в конструктор **XmlReader**.

Класс **XmlReaderSettings** используется для задания набора функций, которые нужно включить для объекта **XmlReader**. Основными свойствами и методами класса **XmlReaderSettings** являются:

- **DtdProcessing** – указывает, разрешать ли обработку определения DTD; по умолчанию обработка определения DTD не разрешается;
- **Shemas.Add(string uri)** – добавляет схему XSD, расположенную по указанному URL-адресу **uri**;
- **ValidationType** – указывает, выполняет ли экземпляр класса **XmlReader** проверку данных, и какой тип проверки выполнять (**DTD** или **XSD**); по умолчанию проверка данных не выполняется;
- **ValidationEventHandler** – указывает обработчик события для получения данных о событиях проверки;

❒ ***Пример* 1.3. *Разработка приложения Windows Forms для проверки XML-документов с помощью XML-схем.***

Требуется разработать на языке C# приложение Windows Forms, которое проверяет XML-документ на соответствие схеме XSD или определению DTD.

Интерфейс пользователя для разрабатываемого приложения показан на рис. 1.4.

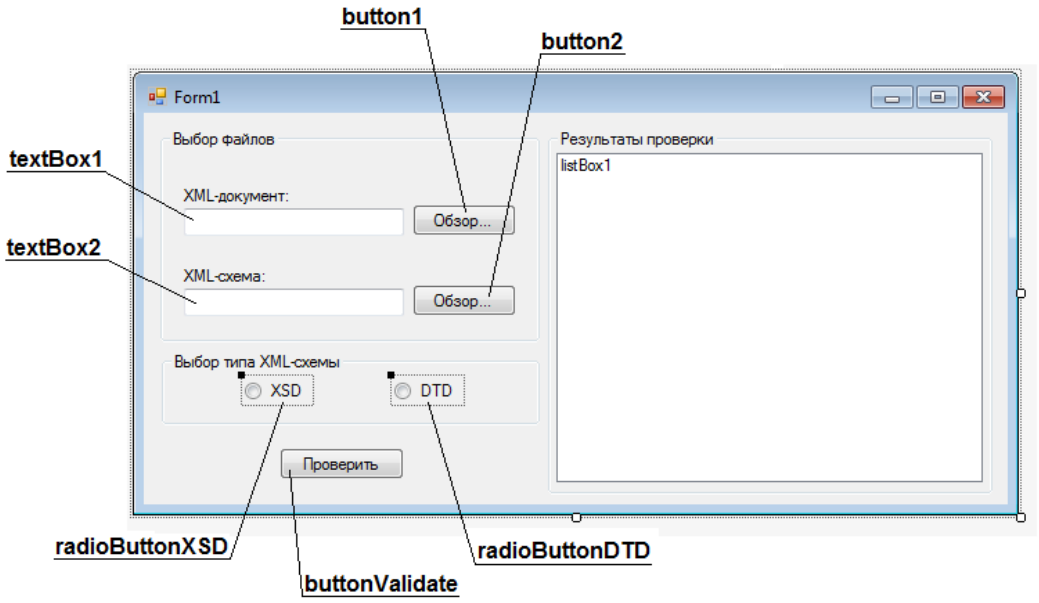

Рис. 1.4. Интерфейс пользователя

Исходный код приложения представлен в листингах 1.6 и 1.7.

Листинг 1.6. Исходный код приложения (часть 1)

```
10  using System.Xml;
11  using System.Xml.Schema; // Импорт пространства имён для работы с XML-схемами
12
13  namespace XmlSchemaValidate
14  {
15      public partial class Form1 : Form
16      {
17          bool xmlValid; // Признак правильности XML-документа
18
19          public Form1()
20          {
21              InitializeComponent();
22              xmlValid = true;
23          }
24
25          // Обработчик события "Загрузка формы Form1"
26          private void Form1_Load(object sender, EventArgs e)
27          {
28              this.Text = "Проверка документов XML с помощью XML-схем " +
29                          "(Турчин Д.Е., каф. ИиАПС)";
30              this.Font = new System.Drawing.Font("Arial", 10, FontStyle.Bold);
31              textBox1.Text = @"H:\XML\CommService(xsd).xml";
32              textBox2.Text = @"H:\XSD\CommService.xsd";
33              radioButtonXSD.Checked = true;
34          }
35
36          /// <summary>
37          /// Обработчик события, которое возникает в результате ошибок,
38          /// найденных при проверке XML-документа с помощью XML-схемы
39          /// </summary>
40          private void readerValidationEventHandler(object sender,
41              ValidationEventArgs e)
42          {
43              if (e.Severity == XmlSeverityType.Warning)
44              {
45                  listBox1.Items.Add("Предупреждение: " + e.Message);
46                  xmlValid = false;
47              }
48              else if (e.Severity == XmlSeverityType.Error)
49              {
50                  listBox1.Items.Add("Ошибка: " + e.Message);
51                  xmlValid = false;
52              }
53          }
```

Листинг 1.7. Исходный код приложения (часть 2)

```
55          // Обработчик события "Нажатие на кнопку buttonValidate"
56          private void buttonValidate_Click(object sender, EventArgs e)
57          {
58              xmlValid = true;
59              XmlReaderSettings rdSets = new XmlReaderSettings();
60              // Выбор типа XML-схемы в зависимости от установки
61              // переключателей RadioButton
62              if (radioButtonXSD.Checked == true)
63              {
64                  rdSets.ValidationType = ValidationType.Schema;
65                  rdSets.Schemas.Add("http://tempuri.org/XMLSchema.xsd",
66                                     textBox2.Text);
67              }
68              else
69              {
70                  rdSets.DtdProcessing = DtdProcessing.Parse;
71                  rdSets.ValidationType = ValidationType.DTD;
72              }
73
74              // Добавление события
75              rdSets.ValidationEventHandler += new
76                  ValidationEventHandler(readerValidationEventHandler);
77
78              XmlReader reader = XmlReader.Create(textBox1.Text, rdSets);
79
80              listBox1.Items.Clear();
81
82              // Чтение узлов XML-документа с выполенением проверки
83              while (reader.Read());
84              if (xmlValid == true)
85              {
86                  listBox1.Items.Add("Ошибок не обнаружено! " +
87                      "XML-документ соответствует схеме");
88              }
89          }
90
91          // Обработчик события "Нажатие на кнопку button1"
92          private void button1_Click(object sender, EventArgs e)
93          {
94              openFileDialog1.Filter = "XML-файлы (*.xml)|*.xml";
95              openFileDialog1.ShowDialog();
96              textBox1.Text = openFileDialog1.FileName;
97          }
98
99          // Обработчик события "Нажатие на кнопку button2"
100         private void button2_Click(object sender, EventArgs e)
101         {
102             openFileDialog1.Filter = "XSD-файлы (*.xsd)|*.xsd" +
103                                      "|DTD-файлы (*.dtd)|*dtd";
104             openFileDialog1.ShowDialog();
105             textBox2.Text = openFileDialog1.FileName;
106         }
107     }
```

Результаты работы полученного приложения при проверке документов XML, соответствующих XSD и DTD, показаны на рис. 1.5.

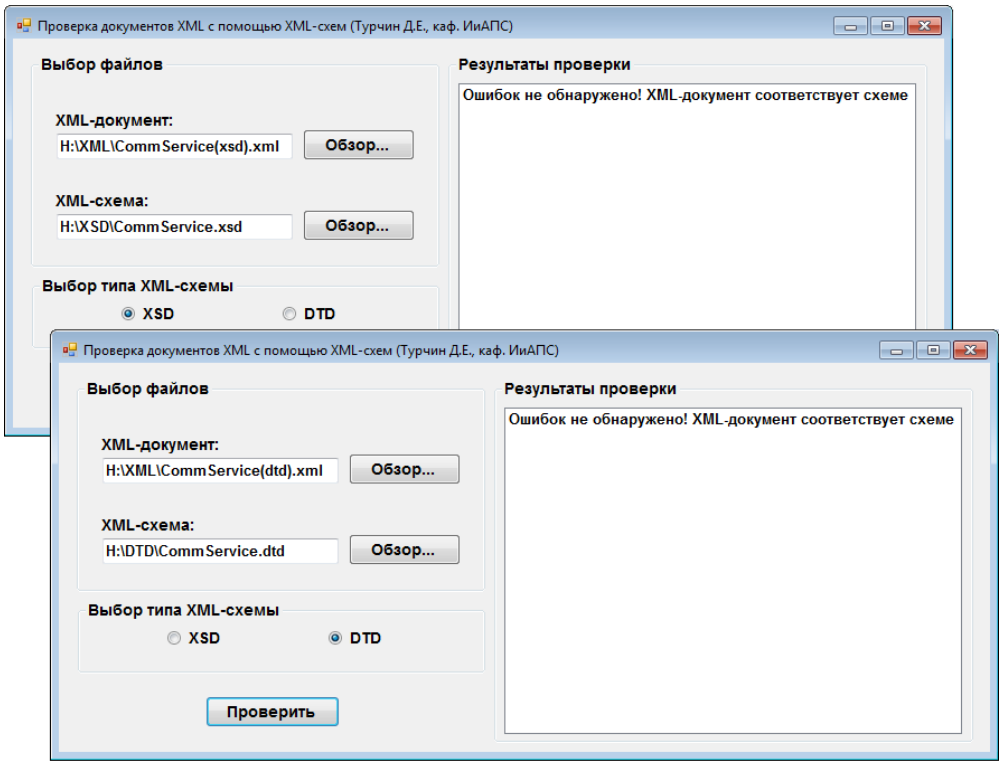

Рис. 1.5. Работа приложения при документах XML, соответствующих своим XML-схемам

Для проверки работоспособности программы изменим в документе XML имя атрибута **код** одного из элементов **дом** на имя **id**. Результат работы приложения с описанием найденных ошибок представлен на рис. 1.6. ❒

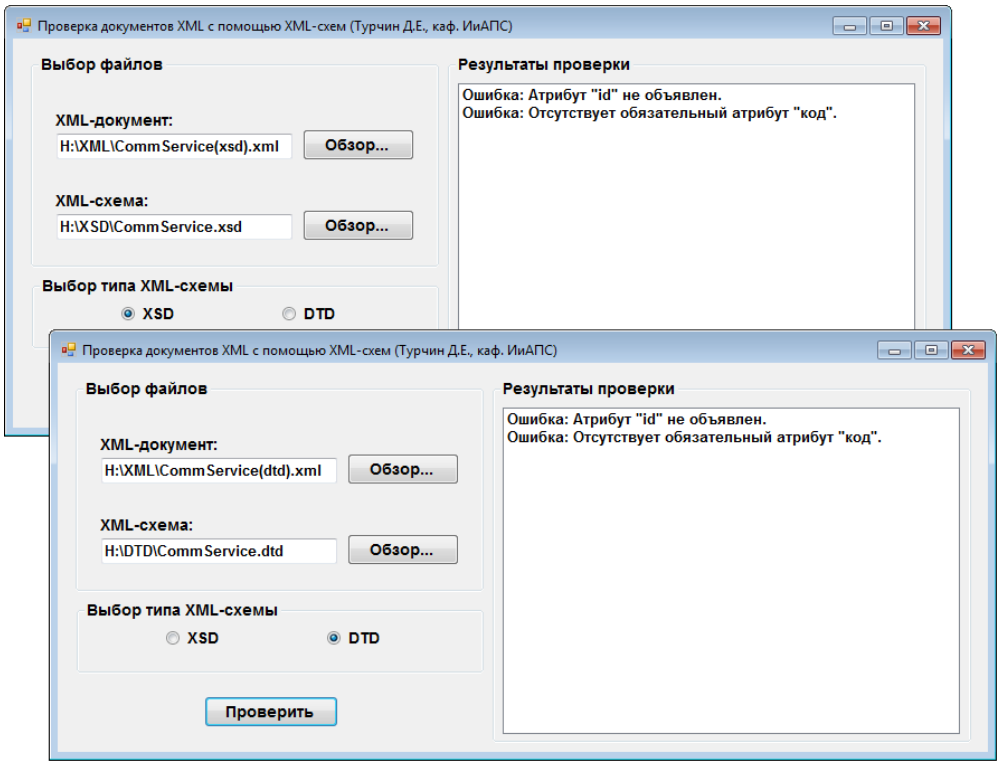

Рис. 1.6. Работа приложения при несоответствующих XML-схемам документах XML

1.3. Порядок выполнения работы

Данная практическая работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Для XML-документа из лабораторной работы №1, разработать внешнее определение DTD. Связать XML-документ с DTD.
3. Разработать XML-схему на языке XSD, в которой заданы требуемые метаданные для XML-документа из работы №1. Связать XML-документ со схемой XSD. При необходимости доработать XML-документ для обеспечения соответствия между ним и схемой.
4. Разработать приложение Windows Forms на языке C# для проверки выбранного документа XML на соответствие заданной XML-схеме (XSD или DTD).
5. Оформить и защитить отчет по лабораторной работе.

1.4. Контрольные вопросы

1. Что называют метаданными?
2. Для чего предназначены схемы XML?
3. В чем заключаются основные недостатки схемы DTD?
4. Как записывается объявление DOCTYPE при задании внутренней и внешней схемы DTD?
5. Каково назначение и синтаксис инструкции ELEMENT схемы DTD?
6. Для чего предназначены ключевые слова ANY и EMPTY в инструкции ELEMENT?
7. Как записываются правила модели содержимого в инструкции ELEMENT?
8. Каково назначение и синтаксис инструкции ATTLIST схемы DTD?
9. Каковы возможные типы атрибута в инструкции ATTLIST?
10. Что можно задавать в качестве значения атрибута по умолчанию в инструкции ATTLIST?
11. Для чего предназначены пространства имен в XML?
12. Как объявляется пространство имен в XML-документе?
13. Что называют пространством имен по умолчанию?
14. Как можно указать схему XSD в XML-документе?

2. РАБОТА С ДЕЛЕГАТАМИ И СОБЫТИЯМИ НА ЯЗЫКЕ C#

2.1. Цель и задачи работы

Цель работы – приобрести умение работать с делегатами и событиями в приложениях на языке C#.

Основные задачи:

• освоить создание универсальных методов с помощью делегатов;

• приобрести умение создавать классы, поддерживающие события;

• научиться использовать стандартные обработчики событий .NET Framework.

Работа рассчитана на 6 часов.

2.2. Основные теоретические сведения

2.2.1. Понятие делегата. Передача делегатов в методы

Понятие делегата. Объявление делегатов.

В языке C# для организации эффективного взаимодействия объектов существуют такие средства, как делегаты (*delegates*) и события (*events*). Делегаты используются для поддержки событий, а также как самостоятельная конструкция языка.

Делегаты C# задают функциональный тип данных. Экземплярами делегата являются функции (методы).

Введение делегатов в C# вызвано необходимостью отказа от указателей, характерных для языка C++. Указатель C++ просто задаёт адрес функции в памяти. В нём не содержится информация ни о количестве параметров функции, ни об их типе, ни о типе возвращаемого значения. То есть указатели C++ не обеспечивают безопасность типов, хотя их и отличает высокая скорость выполнения.

Делегаты в C# применяются главным образом для следующих целей:

• определение вызываемого метода не при компиляции, а динамически во время выполнения программы;

• создание универсальных методов, в которые можно передавать другие методы;

• использование обратных вызовов;

• поддержка событий.

Важным понятием при рассмотрении делегатов является понятие сигнатуры метода.

Под ***сигнатурой метода*** понимают тип и порядок следования выходных и входных параметров метода. К сигнатуре не относятся ни имя метода, ни имена его параметров.

Делегатом называют тип, предназначенный для хранения ссылок на методы с определённой сигнатурой. Описание делегата задаёт сигнатуру методов, которые могут быть вызваны с его помощью.

Для определения делегата в C# используется ключевое слово **delegate**. Синтаксис объявления делегата имеет следующий вид:

```
[модификаторы] delegate Тип Имя_делегата([параметры]);
```

Модификаторы делегата имеют тот же смысл, что и для класса, причём допускаются только модификаторы **new**, **public**, **protected**, **internal** и **private**. ***Тип*** описывает возвращаемое значение методов, вызываемых с помощью делегата. ***Параметрами*** делегата являются параметры вызываемых им методов.

Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса.

Базовым классом для любого делегата является класс **System.Delegate**, который имеет следующие основные статические методы:

• **Combine()** – добавляет метод в список, поддерживаемый делегатом; данный метод может быть вызван неявно с помощью перегруженной операции «+=»;

• **Remove()** – удаляет метод из списка вызовов делегата; может быть вызван посредством перегруженной операции «-=»;

• **RemoveAll()** – удаляет все методы из списка вызовов делегата.

В отличие от класса делегат не может являться предком.

Делегат может хранить ссылки на несколько методов с одинаковой сигнатурой. В этом случае методы вызываются последовательно в том порядке, в котором были добавлены в делегат.

Добавление метода в список выполняется с помощью метода **Combine** класса **System.Delegate**. Также для этой цели можно использовать перегруженную операцию сложения.

При вызове последовательности методов с помощью делегата необходимо учитывать следующие особенности:

• сигнатура методов должна с точности соответствовать делегату;

• каждому методу в списке передается один и тот же набор параметров;

• попытка вызвать делегат, в списке которого нет ни одного метода, вызывает исключение **System.NullReferenceException**.

Передача делегатов в методы. Обратный вызов.

Поскольку делегат является типом, его можно передавать в методы в качестве параметра. Такой приём обеспечивает реализацию универсальных методов и возможность обратного вызова.

Универсальным методом (функцией высшего порядка) называется метод (функция), у которого один или несколько аргументов принадлежат к функциональному типу (делегату). Классическим примером универсального метода является метод вычисления интеграла, у которого один из аргументов задаёт подынтегральную функцию.

Обратным вызовом (англ. *callback*) называется вызов метода (функции), передаваемого в другой метод (функцию) в качестве параметра. Например, в библиотеке описана функция **F**, аргументом которой является другая функция (рис. 2.1). В вызывающем коде описывается функция **Q** с требуемой сигнатурой и передаётся в функцию **F**. Выполнение функции **F** приводит к вызову функции **Q**, то есть управление передаётся из библиотечной функции обратной в вызывающий код.

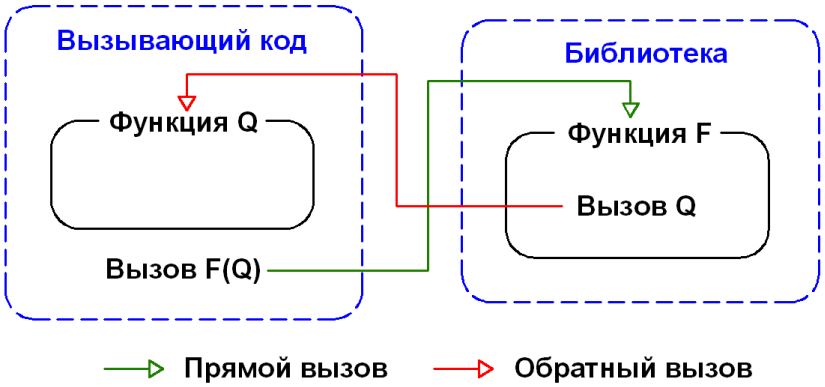

Рис. 2.1. Механизм обратного вызова

❒ ***Пример* 2.1. *Разработка консольного приложения для реализации универсального метода с помощью делегата.***

Требуется разработать консольное приложение, в котором с помощью делегата будет реализован универсальный метод. В качестве универсального метода создадим функцию вычисления определённого интеграла вида:

$$\int_b^a f(x)dx; \qquad (2.1)$$

где a, b – верхний и нижний пределы интеграла; $f(x)$ – подынтегральная функция.

Программа должна выполнять численное интегрирование выбранной функции с заданной точностью по методу трапеций.

Формула метода трапеций:

$$\int_b^a f(x)dx \approx \frac{b-a}{n}\left(\frac{y_0}{2} + y_1 + y_2 + \ldots + y_{n-1} + \frac{y_n}{2}\right). \qquad (2.3)$$

Для решения указанной задачи создадим делегат **double DToDMethod(double x)**, определяющий сигнатуру подынтегральной функции $f(x)$. Также создадим класс **Integral** с методом **Calculate**, вычисляющим интеграл. Этот метод является универсальным методом, который в качестве параметра принимает экземпляр делегата **DToDMethod**.

Исходный код класса **Integral** представлен в листингах 2.1 и 2.2.

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

```
/// <summary>
/// Делегат, указывающий на любой метод, который принимает
/// вещественное число и возращает вещественное число
/// </summary>
/// <param name="x">Вещественное число</param>
/// <returns>Вещественное число</returns>
public delegate double DToDmethod(double x);

/// <summary>
/// Представляет интеграл (сигнатура подинтегральной
/// функции определяется делегатом DToDmethod)
/// </summary>
public class Integral
{
    /// <summary>
    /// Вычисляет интеграл по методу трапеций
    /// </summary>
    /// <param name="a">Верхний предел интеграла</param>
    /// <param name="b">Нижний предел интеграла</param>
    /// <param name="eps">Точность вычисления</param>
    /// <param name="subIntegFunc">Подинтегральная функция</param>
    /// <returns>Значение интеграла</returns>
    public double Calculate(double a, double b, double eps,
        DToDmethod subIntegFunc)
    {
        const int initPoints = 4;  // Начальное число точек на
        // интервале интегрирования
        const int maxPonts = 1024; // Максимальное число точек
        int n = initPoints; // Текущее число точек

        // Значения интегральных сумм при разном числе n
        double s0 = 0, sn = IntegSum(a, b, n, subIntegFunc);
        double epsf = Math.Abs(s0 - sn); // Достигнутая точность

        // Цикл для нахождения n, при котором достигается треб. точность
        for (n *= 2; n < maxPonts; n *= 2)
        {
            s0 = sn;
            sn = IntegSum(a, b, n, subIntegFunc);
            epsf = Math.Abs(s0 - sn);
            if (epsf < eps) break;
        }

        if (epsf < eps)
        {
            Console.WriteLine("Требуемая точность достигнута!\n" +
                "- Требуемая точность: {0}\n" +
                "- Достигнутая точность: {1}\n" +
                "- Число точек на интервале интегр.: {2}\n",
                eps, epsf, n);
        }
        else
        {
            Console.WriteLine("Требуемая точность не достигнута!\n" +
                "- Требуемая точность: {0}\n" +
                "- Достигнутая точность: {1}\n", eps, epsf);
        }

        return (sn);
    }
```

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

```
70          /// <summary>
71          /// Вычисляет интегральную сумму по методу трапеций
72          /// </summary>
73          /// <param name="a">Верхний предел интеграла</param>
74          /// <param name="b">Нижний предел интеграла</param>
75          /// <param name="n">Число точек на интервале интегрирования</param>
76          /// <param name="subIntegFunc">Подинтегральная функция</param>
77          /// <returns>Значение частной интегральной суммы</returns>
78          private double IntegSum(double a, double b, int n,
79              DToDmethod subIntegFunc)
80          {
81              double x = a; // Значение аргумента подинтегральной функции
82              double sum = subIntegFunc(x)/2; // Значение суммы в формуле трапеций
83              double dx = (b - a) / n; // Ширина одной трапеции
84
85              for (int i = 2; i < n; i++)
86              {
87                  x += dx;
88                  sum += subIntegFunc(x);
89              }
90
91              x = b;
92              sum += subIntegFunc(x);
93
94              return (sum * dx);
95          }
96      } // class Integral
97  }
```

Для задания подынтегральных функций создадим класс **SubIntegFunc**, в котором будет два статических метода, реализующих линейную и квадратичную функции (листинг 2.3).

Листинг 2.3. Исходный код класса **SubIntegFunc**

```
 9      /// <summary>
10      /// Представляет подинтегральные функции
11      /// </summary>
12      class SubIntegFunc
13      {
14          // Линейная функция
15          static public double linearFunc(double x)
16          {
17              double k = 0.5, b = 1.0;
18              return (k * x + b);
19          }
20
21          // Квадратичная функция
22          static public double parabolFunc(double x)
23          {
24              double a = 1.0, b = 2.0, c = 3.0;
25              return (a * x * x + b * x + c);
26          }
27      } // class SubIntegFunc
```

Исходный код метода **Main()** консольного приложения показан в листинге 2.4.

Листинг 2.4. Исходный код метода **Main()**

```
11          static void Main(string[] args)
12          {
13              Console.Title = "Работа с делегатами";
14
15              Integral integ = new Integral();
16
17              DToDmethod sif1 = new DToDmethod(SubIntegFunc.linearFunc);
18              double iv = integ.Calculate(2, 3, 1e-3, sif1);
19              Console.WriteLine("Значение интеграла линейной функции: {0}\n",  iv);
20              iv = integ.Calculate(1, 2, 1e-4, sif1);
21              Console.WriteLine("Значение интеграла линейной функции: {0}\n",  iv);
22
23              DToDmethod sif2 = new DToDmethod(SubIntegFunc.parabolFunc);
24              iv = integ.Calculate(2, 3, 2e-2, sif2);
25              Console.WriteLine("Значение интеграла квадратичной функции: {0}\n",  iv);
26              iv = integ.Calculate(1, 2, 2e-3, sif2);
27              Console.WriteLine("Значение интеграла квадратичной функции: {0}\n",  iv);
28
29              DToDmethod sif3 = new DToDmethod(Math.Exp);
30              iv = integ.Calculate(0, 2, 5e-2, sif3);
31              Console.WriteLine("Значение интеграла экспоненты: {0}\n",  iv);
32              iv = integ.Calculate(1, 3, 2e-2, sif3);
33              Console.WriteLine("Значение интеграла экспоненты: {0}\n",  iv);
34
35              Console.Read();
36          }
```

Результат работы консольного приложения показан на рис. 2.2. ❒

2.2.2. Понятие события. Объявление и обработка событий. Стандартные обработчики событий

Понятие события. Объявление и обработка событий.

Важным средством языка C# являются события.

Событие представляет собой автоматическое уведомление от объекта о том, что произошло некоторое действие. Содержательно событием является некоторое специальное состояние, в котором может оказаться объект.

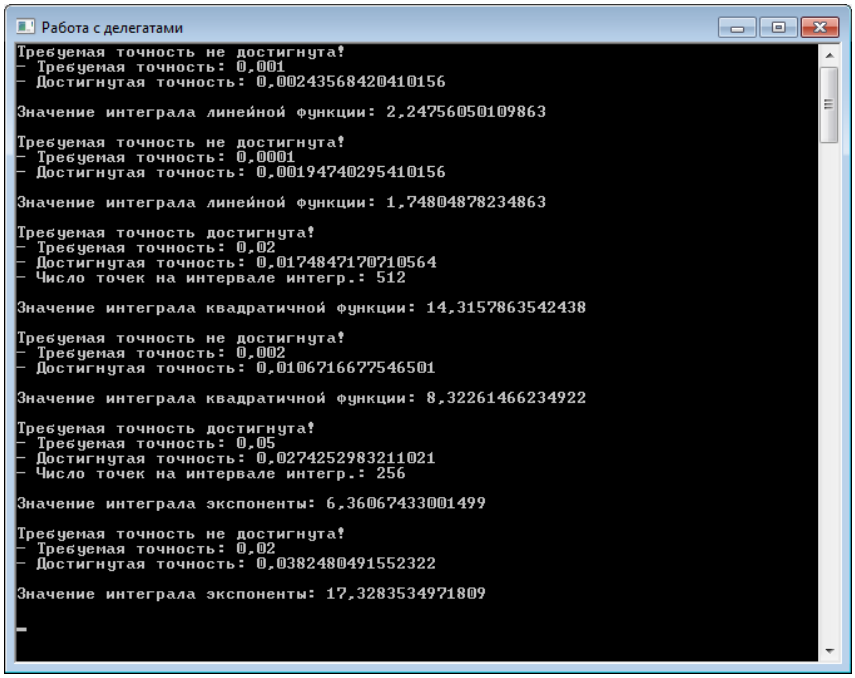

Рис. 2.2. Результат работы программы

Класс, экземпляры которого включают событие, называют ***классом-отправителем*** (издателем) ***сообщения*** (**sender**). Класс, чьи экземпляры получают сообщения, называют ***классом-получателем*** (подписчиком) ***сообщения*** (**receiver**).

Определённый объект должен вызывать событие. Остальные объекты будут подписываться на это событие и реагировать на его возникновение. После возникновения события все подписанные на него объекты получают уведомления и могут совершать различные действия.

Результатом оповещения объектов о событии должен быть запуск определённого кода, который называют ***обработчиком событий*** (англ. *event handler*).

События работают по следующему принципу: объекты, которые должны реагировать на событие, регистрирует обработчик этого события. Когда событие происходит, то вызываются все зарегистрированные обработчики этого события.

В языке C# каждое событие определяется делегатом, описывающим сигнатуру сообщения.

События являются элементами класса и объявляются с помощью ключевого слова **event**. Синтаксис объявления события имеет следующий вид:

```
[модификаторы] event [тип_события] имя_события;
```

Тип события – это тип делегата, на котором основано событие, а ***имя события*** – конкретный экземпляр объявляемого события.

Создание события состоит из следующих этапов:

1. Объявление делегата – функционального класса, задающего сигнатуру. Для некоторых событий можно использовать стандартные делегаты. В этом случае достаточно только знать их имена.
2. Объявление события в классе **sender** как экземпляра соответствующего делегата. При этом добавляется ключевое слово **event**.
3. Объявление методов, инициирующих событие.

Пример объявления делегата и события, соответствующего типу этого делегата, а также обработчика события представлен в листинге 2.5.

Листинг 2.5. Пример объявления делегата, события и обработчика

```
// Тип делегата для события
public delegate void MyDelegate();

// Класс, содержащий событие
public class ClassWithEvent
{
    // Событие
    public event MyDelegate MyEvent;
    // Метод, запускающий событие
    public void OnMyEvent()
    {
        if (MyEvent != null) MyEvent();
    }
}

// Класс, подписанный на событие
class Program
{
    // Обработчик события
    static void Handler()
    {
        Console.WriteLine("Произошло событие!");
    }
```

```
        static void Main(string[] args)
        {
            ClassWithEvent evt = new ClassWithEvent();
            // Добавить обработчик Handler() в список MyEvent
            evt.MyEvent += Handler;
            // Запустить событие
            evt.OnMyEvent();
        }
    }
```

❒ ***Пример* 2.2. *Разработка консольного приложения, содержащего класс, который поддерживает событие.***

В консольном приложении на языке C# требуется разработать класс «телефон» (**Phone**), в котором задано событие «звонок по телефону» (**Phone_Call**). В классе **Program** консольного приложения должен присутствовать метод-обработчик события (**Handler**).

Событие **Phone_Call** должно наступать случайным образом с заданной вероятностью **p**. Кроме того, должна учитываться вероятность **q** того, что на телефонный звонок не будет ответа.

Для создания события **CallEvent** объявим тип делегата **void CallDelegate(int a, int b)**, который задаёт сигнатуру метода-обработчика с двумя целочисленными параметрами. При вызове и обработке события **CallEvent** такими параметрами будут являться время вызова и длительность разговора по телефону.

Объявление делегата **CallDelegate**, а также исходный код класса **Phone** представлены в листингах 2.6 и 2.7.

Листинг 2.6. Объявление делегата **CallDelegate** и исходный код класса **Phone** (часть 1)

```
    /// <summary>
    /// Делегат, задающий тип для события "Звонок на телефон"
    /// </summary>
    /// <param name="a">Целочисленный параметр</param>
    /// <param name="b">Целочисленный параметр</param>
    public delegate void CallDelegate(int a, int b);

    /// <summary>
    /// Представляет телефоны (класс с событием)
    /// </summary>
    class Phone
    {
        int tMax;   // Максимальное время моделирования, мин
        int dtMax;  // Максимальная длительность разговора, мин
        double p;   // Вероятность звонка на телефон
        double q;   // Вероятность отсутствия ответа на звонок
        Random rnd; // Случайное поле

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public Phone()
        {
            tMax = 10000;
            dtMax = 15;
            p = 0.005;
            q = 0.2;
            rnd = new Random();
        }

        /// <summary>
        /// Событие "Звонок на телефон" на основе делегата CallDelegate
        /// </summary>
        public event CallDelegate CallEvent;

        /// <summary>
        /// Метод, запускающий событие CallEvent
        /// <param name="tc">Время звонка, мин</param>
        /// </summary>
        public void OnCall(int tc, int dt)
        {
            if (CallEvent != null) CallEvent(tc, dt);
        }
```

Листинг 2.7. Исходный код класса **Phone** (часть 2)

```
        /// <summary>
        /// Метод, моделирующий работу телефона
        /// </summary>
        public void PhoneWork()
        {
            int dt = 0; // Время разговора
            bool wasCall = false; // Признак наличия звонка

            for (int t = 1; t < tMax; t++)
            {
                // Проверка наличия звонка
                if (rnd.NextDouble() < p)
                {
                    // Проверка ответа на звонок
                    if (rnd.NextDouble() > q)
                    {
                        dt = rnd.Next(1, dtMax);
                    }
                    else { dt = 0; }

                    t += dt;
                    OnCall(t, dt);
                    wasCall = true;
                }
            }

            if (wasCall == false) Console.WriteLine("Звонков не было");
        }
    }
```

Обработчик события **CallEvent** создадим в классе **Program** консольного приложения.

Исходный код класса **Program** представлен в листинге 2.8.

Листинг 2.8. Исходный код класса **Program** консольного приложения

```
class Program
{
    /// <summary>
    /// Обработчик события CallEvent класса Phone
    /// </summary>
    /// <param name="t">Время звонка, мин</param>
    /// <param name="dt">Длительность разговора, мин</param>
    static void Phone_CallEvent(int t, int dt)
    {
        string message = "Сделан звонок! Время: {0} мин. ";

        if (dt > 0) message += "Длительность разговора: {1} мин.";
        else message += "Ответа нет.";

        Console.WriteLine(message, t, dt);
    }

    static void Main(string[] args)
    {
        Console.Title = "Моделирование работы телефона";

        Phone phone = new Phone();
        // Добавление обработчика Phone_CallEvent в список события
        // CallEvent (подписывание обработчика на событие CallEvent)
        phone.CallEvent += Phone_CallEvent;
        phone.PhoneWork();

        Console.Read();
    }
}
```

Результат работы консольного приложения показан на рис. 2.3. ❒

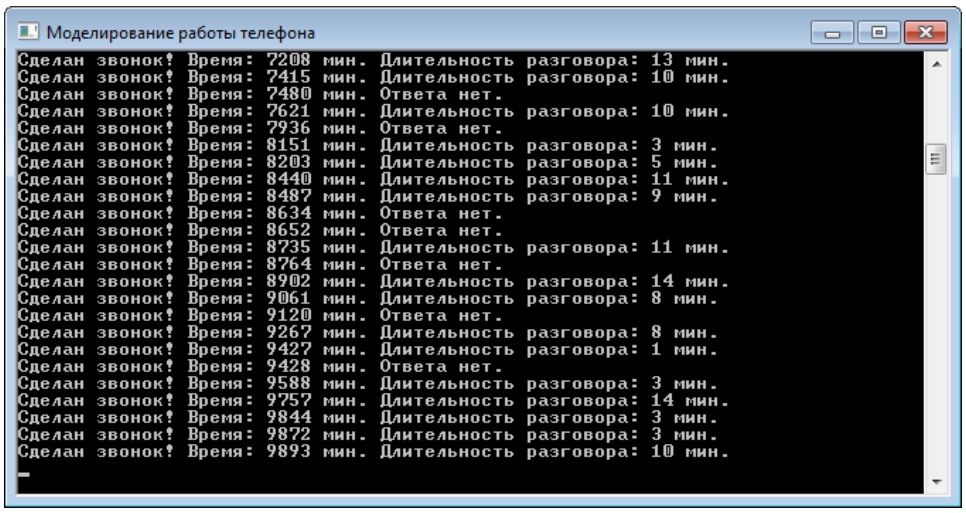

Рис. 2.3. Результат работы консольного приложения

Стандартные обработчики событий в .NET Framework.

В приложениях на языке C# разрешается формировать события с любым списком параметров, но для совместимости программных компонентов с .NET Framework следует придерживаться определённых требований.

Стандартные обработчики событий в .NET Framework должны объявляться следующим образом:

```
[модификаторы] void Имя_Обработчика(object sender, EventArgs e)
{ ... }
```

Стандартный обработчик событий должен иметь два параметра:

• **sender** – ссылка на объект, формирующий событие; параметр, передаваемый вызывающим кодом с помощью ключевого слова **this**;

• **e** – параметр типа **EventArgs**, содержащий любую дополнительную информацию, которая требуется обработчику.

Класс **EventArgs** является базовым классом для классов, содержащих данные о событии. Он предназначен исключительно для передачи аргументов объекта обработчикам событий.

Обычно имена стандартных обработчиков событий формируются следующим образом: имя класса, нижнее подчёркивание, имя события (например, **Phone_CallEvent**).

Для объявления делегатов событий в .NET Framework предоставляется встроенный обобщённый делегат под названием **EventHandler<TEventArgs>**, где тип **TEventArgs** обозначает тип аргумента **e**.

❒ ***Пример* 2.3. *Разработка приложения Windows Forms, содержащего класс с событием и стандартный обработчик.***

Требуется разработать приложение Window Forms, которое обеспечивает моделирование объектов класса «телефон» (**Phone**), содержащего событие «звонок по телефону» (**Phone_Call**). Для обработки события в классе формы (**Form1**) должен быть реализовать стандартный обработчик **void CallEventHandler(object sender, EventArgs e)**.

Вызов события **Phone_Call** будет производиться с помощью элемента управления «таймер» (**timer1**). Скорость работы таймера, задаваемая свойством **Interval**, должна иметь возможность регулироваться в процессе работы приложения.

Интерфейс пользователя приложения показан на рис. 2.4.

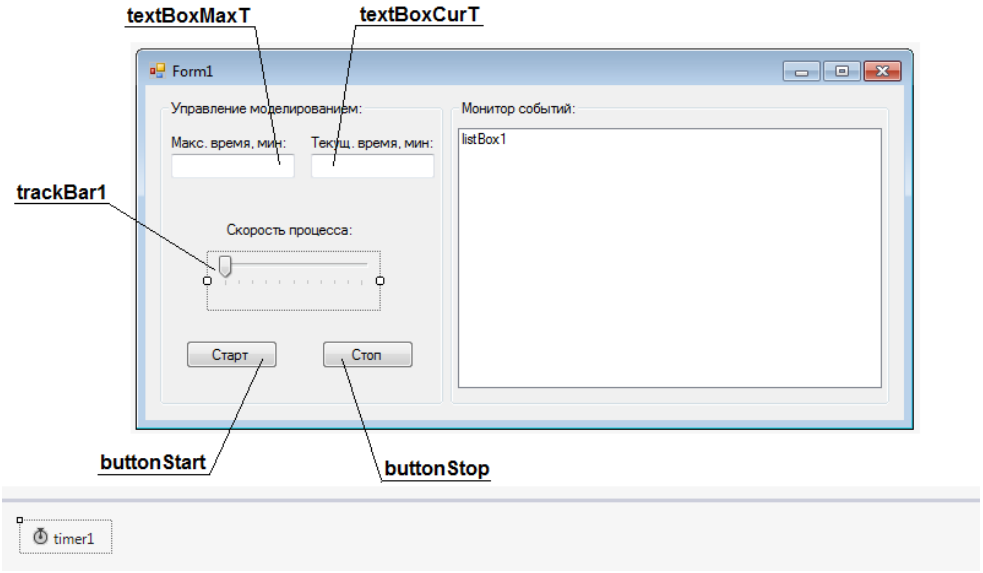

Рис. 2.4. Интерфейс пользователя приложения

Для передачи аргументов события (длительность разговора) в стандартный обработчик создадим класс **CallEventArgs**, производный от класса **EventArgs** (листинг 2.9).

Листинг 2.9. Исходный код класса **CallEventArgs**

```
 9    /// <summary>
10    /// Класс, представляющий аргументы события CallEvent
11    /// </summary>
12    public class CallEventArgs : EventArgs
13    {
14        /// <summary>
15        /// Длительность разговора, мин
16        /// </summary>
17        public int Duration { get; set; }
18
19    } // class CallEventArgs
```

Листинг 2.10. Исходный код класса **Phone**

```
    /// <summary>
    /// Представляет телефоны (класс с событием)
    /// </summary>
    public class Phone
    {
        int dtMax;  // Максимальная длительность разговора, мин
        double p;   // Вероятность звонка
        double q;   // Вероятность отсутствия ответа на звонок
        Random rnd; // Случайное поле

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public Phone()
        {
            dtMax = 15;
            p = 0.01;
            q = 0.2;
            rnd = new Random();
        }

        /// <summary>
        /// Событие "Звонок на телефон" на основе встроенного
        /// обобщённого делегата EventHandler
        /// </summary>
        public event EventHandler<CallEventArgs> CallEvent;

        /// <summary>
        /// Метод, запускающий событие CallEvent
        /// <param name="e">Аргументы события</param>
        /// </summary>
        public void OnCall(CallEventArgs e)
        {
            if (CallEvent != null) CallEvent(this, e);
        }

        /// <summary>
        /// Метод, моделирующий работу телефона
        /// </summary>
        public void PhoneWork()
        {
            int dt = 0; // Время разговора
            CallEventArgs e = new CallEventArgs();

            // Проверка наличия звонка
            if (rnd.NextDouble() < p)
            {
                // Проверка ответа на звонок
                if (rnd.NextDouble() > q)
                {
                    dt = rnd.Next(1, dtMax);
                }
                else { dt = 0; }
                e.Duration = dt;
                OnCall(e);
            }
        }
    }
```

Листинг 2.11. Исходный код класса **Form1**

```
14      public partial class Form1 : Form
15      {
16          Phone phone; // Телефон
17          int t;       // Текущее время моделирования, мин
18
19          public Form1()
20          {
21              InitializeComponent();
22              phone = new Phone();
23              t = 0;
24          }
25
26          /// <summary>
27          /// Стандартный обработчик события CallEvent
28          /// </summary>
29          /// <param name="sender">Объект-отправитель события</param>
30          /// <param name="e">Аргументы события</param>
31          private void Phone_CallEvent(object sender, CallEventArgs e)
32          {
33              string message = "- Сделан звонок! Время: {0} мин. ";
34              if (e.Duration > 0) message += "Длительность разговора: {1} мин.";
35              else message += "Ответа нет.";
36              listBox1.Items.Add(string.Format(message, t, e.Duration));
37          }
38
39          private void Form1_Load(object sender, EventArgs e)
40          {
41              this.Text = "Моделирование работы телефона (Турчин Д.Е., каф. ИиАПС)";
42              this.Font = new Font("Arial", 10, FontStyle.Bold);
43              textBoxMaxT.Text = "10000";
44              textBoxCurT.Text = "0";
45              trackBar1.Maximum = 10;
46              trackBar1.Minimum = 1;
47              trackBar1.Value = 5;
48              timer1.Interval = Convert.ToInt32(100 / trackBar1.Value);
49              // Добавить обработчик Phone_CallEvent в список события CallEvent
50              phone.CallEvent += Phone_CallEvent;
51          }
52
53          // Обработчик события, происходящего по истечении заданного интервала
54          // таймера timer1 при условии, что таймер включен
55          private void timer1_Tick(object sender, EventArgs e)
56          {
57              phone.PhoneWork(); // Запуск моделирования
58              timer1.Interval = Convert.ToInt32(100 / trackBar1.Value);
59              t += 1;
60              if (t >= Convert.ToInt32(textBoxMaxT.Text)) timer1.Stop();
61              textBoxCurT.Text = t.ToString();
62          }
63
64          private void buttonStart_Click(object sender, EventArgs e)
65          {
66              timer1.Start();
67          }
68
69          private void buttonStop_Click(object sender, EventArgs e)
70          {
71              timer1.Stop();
72          }
73      }
```

Работа приложения при заданном времени моделирования показана на рис. 2.5.

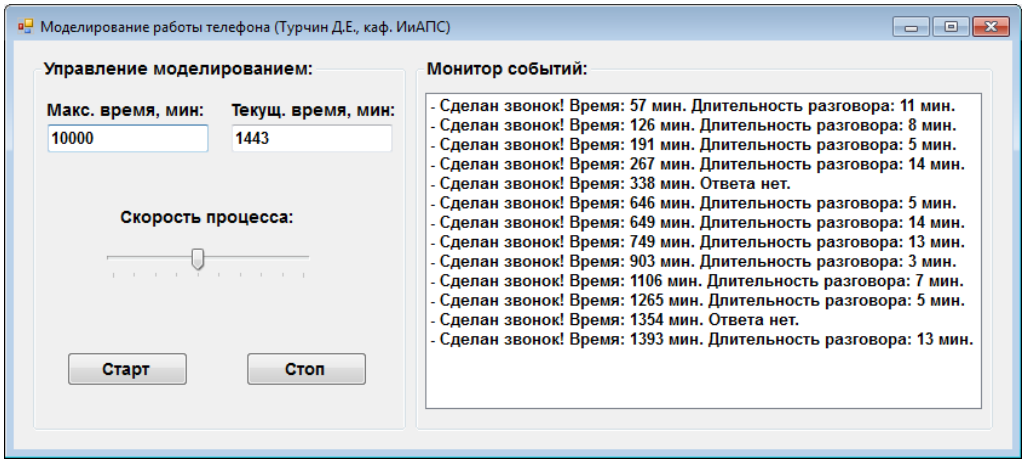

Рис. 2.5. Работа приложения

2.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать проект консольного приложения, в котором с помощью делегата реализуется универсальный метод, реализующий численное интегрирование функции одной переменной (табл. 2.1) по методу прямоугольников.

Формула метода прямоугольников:

$$\int_b^a f(x)dx \approx \frac{b-a}{n}\left(y_0 + y_1 + y_2 + ... + y_{n-1}\right). \qquad (2.3)$$

3. Разработать проект консольного приложения, в котором присутствует заданный класс с событием (табл. 2.2). Вызов события осуществляется случайным образом с передачей указанных параметров. Обработчик события задаётся в классе **Program** консольного приложения.
4. Разработать проект приложения Windows Forms, в котором имеется класс с событием (табл. 2.2). Вызов события должен производиться с помощью таймера. Для обработки события

необходимо использовать стандартные обработчики .NET Framework.

5. Оформить и защитить отчет по лабораторной работе.

Таблица 2.1

Варианты заданий для реализации универсального метода с помощью делегата

№ вар.	Функции	Параметры	№ вар.	Функции	Параметры
1	$\frac{1}{\sqrt{5+4x-x^2}}$	$a = 2$ $b = 3{,}5$ $\varepsilon = 0{,}001$	**13**	$\frac{x^3}{x^8+1}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}005$
2	$\frac{x}{x^2+3x+2}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}002$	**14**	$\frac{1}{x^2+4}$	$a = 0$ $b = 2$ $\varepsilon = 0{,}001$
3	$\frac{1+\sqrt{x}}{x^2}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}005$	**15**	$\sqrt{1+x^3}$	$a = 0$ $b = 2$ $\varepsilon = 0{,}002$
4	$\sqrt{1+x^5}$	$a = 0$ $b = 2$ $\varepsilon = 0{,}001$	**16**	$\frac{1}{\sqrt{1+x^4}}$	$a = 0$ $b = 0{,}5$ $\varepsilon = 0{,}005$
5	$\frac{1}{\sqrt{1-x^4}}$	$a = 0$ $b = 0{,}6$ $\varepsilon = 0{,}002$	**17**	$\frac{1}{\sqrt[3]{1+x^2}}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}001$
6	$\sqrt{x(1-x)}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}005$	**18**	$x\ln(1+x)$	$a = 0$ $b = 1$ $\varepsilon = 0{,}001$
7	e^{x^2}	$a = 0$ $b = 1$ $\varepsilon = 0{,}001$	**19**	e^{x^3}	$a = 0$ $b = 1$ $\varepsilon = 0{,}002$
8	$e^{\sqrt{x}}$	$a = 0$ $b = 0{,}5$ $\varepsilon = 0{,}002$	**20**	$\frac{1}{\ln x}$	$a = 2$ $b = 3$ $\varepsilon = 0{,}005$
9	$\frac{\ln(1+x^2)}{1+x^2}$	$a = 0$ $b = 1$ $\varepsilon = 0{,}005$	**21**	$x\ln(5+4\cos x)$	$a = 0$ $b = 3$ $\varepsilon = 0{,}001$
10	$\frac{\sin x}{\sqrt{x}}$	$a = 0{,}2$ $b = 1$ $\varepsilon = 0{,}001$	**22**	$\sqrt[3]{x}\cos x$	$a = 0$ $b = 1$ $\varepsilon = 0{,}002$

№ вар.	Функции	Параметры	№ вар.	Функции	Параметры
11	$\sqrt{\sin x}\sin\frac{x}{2}$	$a = 0$ $b = 3$ $\varepsilon = 0{,}001$	**23**	$\frac{\operatorname{arctg} x}{x}$	$a = 0{,}2$ $b = 1$ $\varepsilon = 0{,}005$
12	$\frac{e^x}{x}$	$a = 0{,}4$ $b = 0{,}6$ $\varepsilon = 0{,}002$	**24**	$\frac{\sin x}{x}$	$a = 0$ $b = 0{,}8$ $\varepsilon = 0{,}002$

Таблица 2.2

Варианты заданий для создания класса с событием

№ вар.	Данные для разработки класса с событием
1, 9, 17	***Банкомат*** (время, код карты, снимаемая сумма)
2, 10, 18	***Сейсмостанция*** (время, координаты эпицентра (x, y), сила землетрясения по шкале Рихтера)
3, 11, 19	***Интернет-магазин*** (время покупки, № товара, количество)
4, 12, 20	***Пожарная сигнализация*** (время, номер дома, номер квартиры)
5, 13, 21	***Оплата покупок в супермаркете*** (время, № кассы, сумма)
6, 14, 22	***Станция РЛС*** (время, координаты объекта (x, y), высота объекта)
7, 15, 23	***Домофон*** (время вызова, номер квартиры, открытие двери (да, нет))
8, 16, 24	***Ремонтная служба цеха*** (время, номер отказавшего станка, категория сложности ремонта)

2.4. Контрольные вопросы

1. Что понимают под делегатом в языке C#?
2. Для решения каких задач применяют делегаты?
3. Как объявляется делегат в программе на языке C#?
4. Какие методы называют универсальными?
5. Что такое обратный вызов?
6. Что понимают под событием в языке C#?
7. Какие методы называют обработчиками событий?
8. Каков синтаксис объявления события на языке C#?
9. Как объявляются стандартные обработчики событий на языке C#?

3. ОСНОВЫ ИСПОЛЬЗОВАНИЯ ПОРОЖДАЮЩИХ ШАБЛОНОВ GOF В ПРИЛОЖЕНИЯХ НА ЯЗЫКЕ C#

3.1. Цель и задачи работы

Цель работы – прибрести умение разрабатывать приложения на языке C# с использованием таких порождающих шаблонов проектирования, как Фабричный метод и Абстрактная фабрика.

Основные задачи:

• Освоить использование таких порождающих шаблонов из каталога GoF, как Фабричный метод и Абстрактная фабрика.

• Научиться применять порождающие шаблоны GoF при разработке приложений на языке C#.

Работа рассчитана на 6 часов.

3.2. Основные теоретические сведения

3.2.1. Общие сведения о порождающих шаблонах GoF. Шаблон Фабричный метод

Общие сведения о порождающих шаблонах GoF. Шаблон Фабричный метод.

Порождающие шаблоны проектирования предназначены для создания объектов в программной системе. Данный вид шаблонов помогает сделать систему независимой от способа создания объектов.

Порождающие шаблоны из каталога GoF:

- Фабричный метод (*Factory Method*).
- Абстрактная фабрика (*Abstract Factory*).
- Одиночка (*Singleton*).
- Строитель (*Builder*).
- Прототип (*Prototype*).

Фабричный метод – порождающий шаблон проектирования, предоставляющий производным классам интерфейс для создания экземпляров некоторого класса. В момент создания про-

изводные классы могут определить, какой класс создавать. Фабричный метод позволяет базовому классу делегировать создание производных классов.

Проблема. Как определить интерфейс для создания экземпляров иерархии классов, но оставить производным классам решение о том, экземпляр какого класса создавать?

Решение. Возложить обязанность по созданию других объектов на базовый класс с помощью специального метода (фабричного).

Структура. Диаграмма классов шаблона Фабричный метод показана на рис. 3.1.

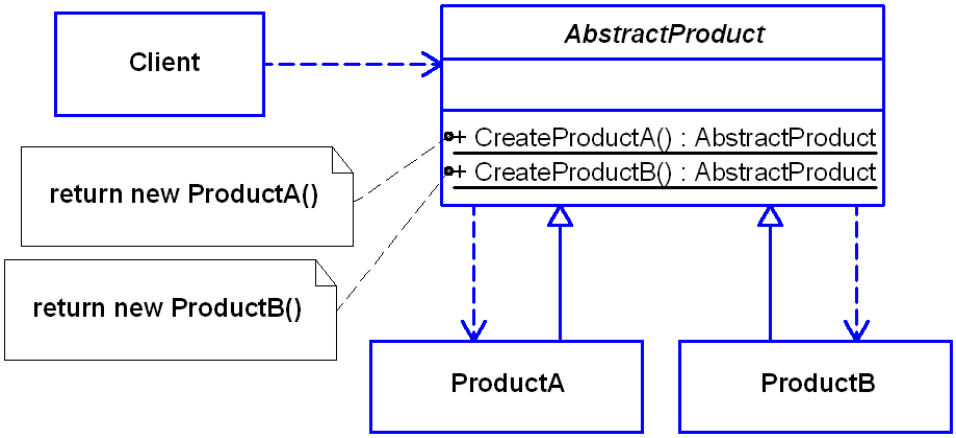

Рис. 3.1. Диаграмма классов шаблона Фабричный метод

Основные участники шаблона:

• **AbsrtactProduct** – абстрактный класс, определяющий интерфейс создания объектов с помощью статических фабричных методов;

• **ProductA**, **ProductB** – классы, представляющие экземпляры конкретных продуктов.

Реализация шаблона Фабричный метод на языке C#.

Простой пример реализации шаблона Фабричный метод на языке C# приведён в листинге 3.1.

Листинг 3.1. Пример реализации шаблона Фабричный метод на языке C#

```
// Представляет абстрактные продукты
abstract class AbstractProduct
{
    // Фабричный метод, возвращающий экземпляры ProductA
    public static AbstractProduct CreateProductA()
    {
        return new ProductA();
    }

    // Фабричный метод, возвращающий экземпляры ProductB
    public static AbstractProduct CreateProductB()
    {
        return new ProductB();
    }

    public abstract string GetData();
}

// Представляет конкретные продукты A
class ProductA : AbstractProduct
{
    public override string GetData()
    {
        return "Экземпляр класса ProductA";
    }
}

// Представляет конкретные продукты B
class ProductB : AbstractProduct
{
    public override string GetData()
    {
        return "Экземпляр класса ProductB";
    }
}

// Представляет клиентов
class Client
{
    static void Main(string[] args)
    {
        Product pA = Product.CreateProductA();
        Console.WriteLine(pA.GetData());
        Product pB = Product.CreateProductB();
        Console.WriteLine(pB.GetData());
    }
}
```

❒ ***Пример* 3.1. *Разработка решения на языке C# с использованием шаблона Фабричный метод.***

Требуется разработать на языке C# приложение, которое позволяет создавать объекты определённой иерархии классов с помощью шаблона Фабричный метод.

Иерархия классов должна включать в себя следующие классы:

• **AstroObject** – астрономический объект (название, масса), абстрактный класс, содержащий

- **GetStringData() : string** – абстрактный метод, возвращающий строку с данными об объекте;

- **CreateStar() : AstroObject** , **CreatePlanet() : AstroObject** – статические фабричные методы, возвращающие экземпляры классов **Star** и **Planet**.

• **Star** – звезда (радиус, спектральный класс);

• **Planet** – планета (экваториальный диаметр, орбитальный радиус).

Указанные классы разместим в проекте библиотеки классов **AstroObjectsLib**. Диаграмма классов данного проекта показана на рис. 3.2.

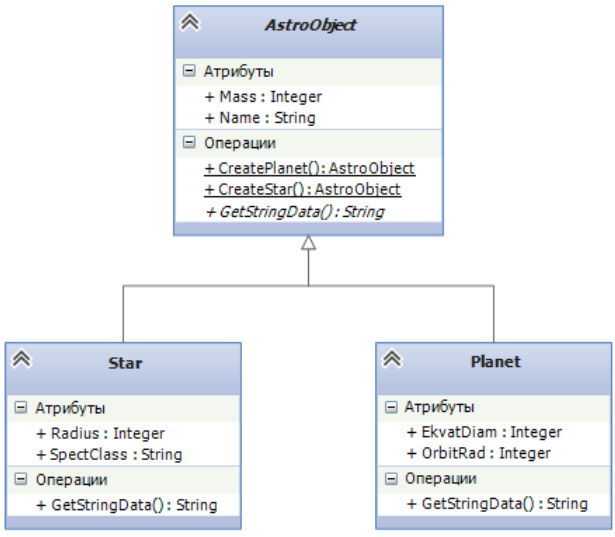

Рис. 3.2. Диаграмма классов проекта **AstroObjectsLib**

Исходный код классов **AstroObject**, **Star** и **Planet** представлен в листингах 3.2, 3.3 и 3.4.

Для демонстрации работы фабричных методов добавим в решение проект консольного приложения **ConsoleClient**. Исходный код класса **Program** консольного приложения показан в листинге 3.5.

Листинг 3.2. Исходный код класса **AstroObjects** (часть 1)

```
 9      /// <summary>
10      /// Представляет астрономические объекты
11      /// </summary>
12      public abstract class AstroObject
13      {
14          /// <summary>
15          /// 1 астрономическая единица (а.е), м
16          /// </summary>
17          public const double astroUnit = 1.496e+11;
18
19          /// <summary>
20          /// Название
21          /// </summary>
22          public string Name { get; set; }
23
24          /// <summary>
25          /// Масса
26          /// </summary>
27          public double Mass { get; set; }
28
29          /// <summary>
30          /// Конструктор с параметрами
31          /// </summary>
32          /// <param name="name">Название</param>
33          /// <param name="mass">Масса</param>
34          public AstroObject(string name, double mass)
35          {
36              Name = name;
37              Mass = mass;
38          }
39
40          /// <summary>
41          /// Возвращает строку с данными об объекте
42          /// </summary>
43          /// <returns>Строка с данными об объекте</returns>
44          public abstract string GetStringData();
```

Листинг 3.2. Исходный код класса **AstroObjects** (часть 2)

```
46          /// <summary>
47          /// Возвращает экземпляр класса Star (Фабричный метод)
48          /// </summary>
49          /// <param name="name">Название</param>
50          /// <param name="mass">Масса (относительно Солнца)</param>
51          /// <param name="radius">Радиус (относительно Солнца)</param>
52          /// <param name="spect">Спектральный класс</param>
53          /// <returns>Звезда</returns>
54          public static AstroObject CreateStar(string name, double mass,
55              double radius, string spect)
56          {
57              return new Star(name, mass, radius, spect);
58          }
59
60          /// <summary>
61          /// Возвращает экземпляр класса Star (Фабричный метод)
62          /// </summary>
63          /// <returns>Солнце</returns>
64          public static AstroObject CreateStar()
65          {
66              return new Star();
67          }
68
69          /// <summary>
70          /// Возвращает экземпляр класса Planet (Фабричный метод)
71          /// </summary>
72          /// <param name="name">Название</param>
73          /// <param name="mass">Масса (относительно Земли)</param>
74          /// <param name="ekvDiam">Экватор. диаметр (относительно Земли)</param>
75          /// <param name="orbitRad">Радиус орбиты (относительно Земли)</param>
76          /// <returns>Планета</returns>
77          public static AstroObject CreatePlanet(string name, double mass,
78              double ekvDiam, double orbitRad)
79          {
80              return new Planet(name, mass, ekvDiam, orbitRad);
81          }
82
83          /// <summary>
84          /// Возвращает экземпляр класса Planet (Фабричный метод)
85          /// </summary>
86          /// <returns>Земля</returns>
87          public static AstroObject CreatePlanet()
88          {
89              return new Planet();
90          }
91      }
```

Листинг 3.3. Исходный код класса **Star**

```
/// <summary>
/// Представляет звёзды
/// </summary>
internal class Star : AstroObject
{
    /// <summary>
    /// Масса Солнца, кг
    /// </summary>
    public const double sunMass = 1.99e+30;

    /// <summary>
    /// Радиус Солнца, м
    /// </summary>
    public const double sunRadius = 6.96e+8;

    /// <summary>
    /// Радиус (относительно Солнца)
    /// </summary>
    public double Radius { get; set; }

    /// <summary>
    /// Спектральный класс
    /// </summary>
    public string SpectClass { get; set; }

    /// <summary>
    /// Конструктор с параметрами
    /// </summary>
    /// <param name="name">Название</param>
    /// <param name="mass">Масса (относительно Солнца)</param>
    /// <param name="radius">Радиус (относительно Солнца)</param>
    /// <param name="spect">Спектральный класс</param>
    public Star(string name, double mass, double radius, string spect)
        : base(name, mass)
    {
        Radius = radius;
        SpectClass = spect;
    }

    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public Star()
        : this("Солнце", 1.0, 1.0, "G2V") { }

    /// <summary>
    /// Возвращает строку с данными о звезде
    /// </summary>
    /// <returns>Строка с данными о звезде</returns>
    public override string GetStringData()
    {
        return string.Format("Данные о звезде:\n" +
                             "- название: {0}\n" +
                             "- масса: {1} кг\n" +
                             "- радиус: {2} м\n" +
                             "- спектр. класс: {3}\n",
        Name, Mass * sunMass, Radius * sunRadius, SpectClass);
    }
}
```

Листинг 3.4. Исходный код класса **Planet**

```
/// <summary>
/// Представляет планеты
/// </summary>
internal class Planet : AstroObject
{
    /// <summary>
    /// Масса земли, кг
    /// </summary>
    public const double earthMass = 5.95e+24;

    /// <summary>
    /// Экваториальный диаметр Земли, м
    /// </summary>
    public const double earthEkvDiam = 1.27e+7;

    /// <summary>
    /// Экваториальный диаметр (относительно Земли)
    /// </summary>
    public double EkvatDiam { get; set; }

    /// <summary>
    /// Радиус орбиты (относительно Земли)
    /// </summary>
    public double OrbitRadius { get; set; }

    /// <summary>
    /// Конструктор с параметрами
    /// </summary>
    /// <param name="name">Название</param>
    /// <param name="mass">Масса (относительно Земли)</param>
    /// <param name="ekvDiam">Экватор. диаметр (относительно Земли)</param>
    /// <param name="orbitRad">Радиус орбиты, а.е</param>
    public Planet(string name, double mass, double ekvDiam, double orbitRad)
        : base(name, mass)
    {
        EkvatDiam = ekvDiam;
        OrbitRadius = orbitRad;
    }

    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public Planet()
        : this("Земля", 1.0, 1.0, 1.0) { }

    /// <summary>
    /// Возвращает строку с данными о планете
    /// </summary>
    /// <returns>Строка с данными о планете</returns>
    public override string GetStringData()
    {
        return string.Format("Данные о пленете:\n" +
                             "- название: {0}\n" +
                             "- масса: {1} кг\n" +
                             "- экват. диаметр: {2} м\n" +
                             "- орбит. радиус: {3} м\n",
        Name, Mass * earthMass, EkvatDiam * earthEkvDiam,
        OrbitRadius * astroUnit);
    }
}
```

Листинг 3.5. Исходный код класса **Program** консольного приложения

```
11      class Program
12      {
13          static void Main(string[] args)
14          {
15              Console.Title = "Работа с шаблоном Фабричный метод";
16
17              // Создание объектов с помощью фабричных методов
18              AstroObject planet1 = AstroObject.CreatePlanet();
19              AstroObject planet2 = AstroObject.CreatePlanet("Марс", 0.11, 0.53, 1.52);
20              AstroObject star1 = AstroObject.CreateStar("Проксима Центавра",
21                  0.12, 0.15, "M5.5Ve");
22              AstroObject star2 = AstroObject.CreateStar();
23
24              List<AstroObject> astroObjects = new List<AstroObject> {
25                  planet1, planet2, star1, star2 };
26
27              foreach (AstroObject ao in astroObjects)
28              {
29                  Console.WriteLine(ao.GetStringData());
30              }
31
32              Console.Read();
33          }
```

Результат работы консольного приложения показан на рис. 3.3. ❒

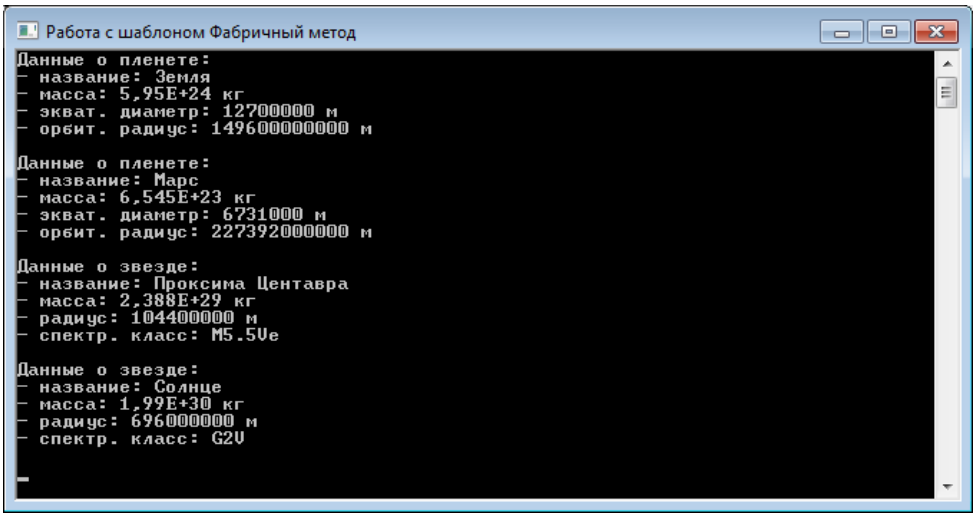

Рис. 3.3. Результат работы консольного приложения

3.2.2. Шаблон Абстрактная фабрика

Шаблон Абстрактная фабрика.

Шаблон ***абстрактная фабрика*** (Abstract Factory) обеспечивает создание семейств взаимосвязанных объектов без указания их конкретных классов.

Проблема. Как создать семейство взаимосвязанных объектов, не указывая их конкретных классов?

Решение. Создать абстрактный класс, в котором объявлен интерфейс для создания семейств конкретных классов.

Структура. Диаграмма классов шаблона Абстрактная фабрика представлена на рис. 3.4.

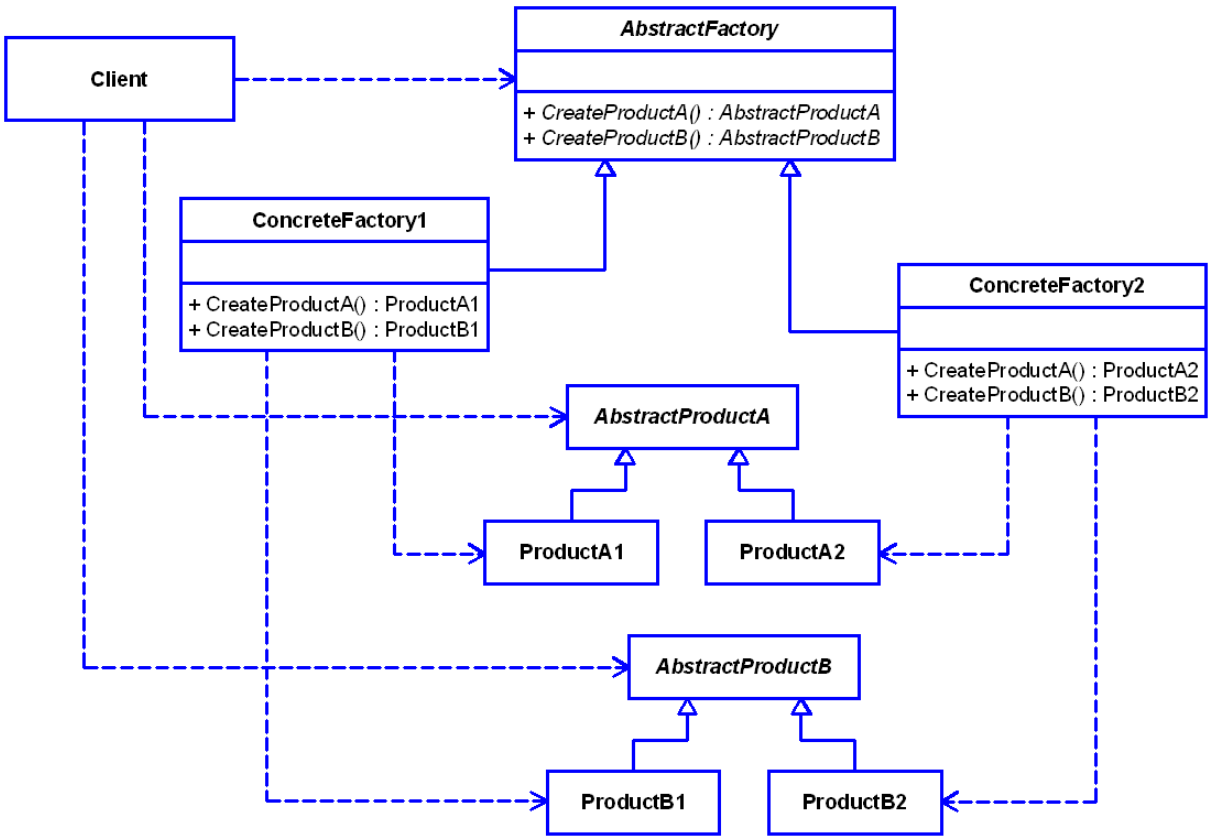

Рис. 3.4. Диаграмма классов для шаблона Абстрактная фабрика

Основными участниками шаблона Абстрактная фабрика являются:

• **AbstractFactory** – абстрактная фабрика, представляющая собой абстрактный класс; объявляет интерфейс для операций, создающих абстрактные объекты-продукты;

• **AbstractProductA**, **AbstractProductB** – абстрактные продукты; объявляют интерфейс для объектов-продуктов;

• **ProductA1**, **ProductA2**, **ProductB1**, **ProductB2** – конкретные продукты; определяют объекты-продукты, создаваемые соответствующей конкретной фабрикой;

• **ConcreteFactory1**, **ConcreteFactory2** – конкретные фабрики, которые реализуют операции, создающие конкретные объекты-продукты.

В клиентском коде вызывается абстрактный класс (интерфейс) **AbstractFactory** и производится вызов абстрактных классов **AbstractProductA** и **AbstractProductB**.

Реализация шаблона Абстрактная фабрика на языке C#.

Простой пример реализации шаблона Фабричный метод на языке C# приведён в листинге 3.6.

Листинг 3.6. Пример реализации шаблона Абстрактная фабрика на языке C#

```
// Представляет абстрактнные продукты A
public abstract class AbstractProductA
{
    public abstract string GetData();
}

// Представляет конкретные продукты A1
public class ProductA1 : AbstractProductA
{
    public override string GetData()
    { return "Экземпляр класса ProductA1"; }
}

// Представляет конкретные продукты A2
public class ProductA2 : AbstractProductA
{
    public override string GetData()
    { return "Экземпляр класса ProductA2"; }
}

// Представляет абстрактнные продукты B
```

```
public abstract class AbstractProductB
{
    public abstract string GetData();
}

// Представляет конкретные продукты B1
public class ProductB1 : AbstractProductB
{
    public override string GetData()
    { return "Экземпляр класса ProductB1"; }
}

// Представляет конкретные продукты B2
public class ProductB2 : AbstractProductB
{
    public override string GetData()
    { return "Экземпляр класса ProductB2"; }
}

// Представляет абстрактные фабрики
public abstract class AbstractFactory
{
    public abstract AbstractProductA CreateProductA();
    public abstract AbstractProductB CreateProductB();
}

// Представляет конкретные фабрики 1
public class ConcreteFactory1 : AbstractFactory
{
    public override AbstractProductA CreateProductA()
    { return new ProductA1(); }

    public override AbstractProductB CreateProductB()
    { return new ProductB1(); }
}

// Представляет конкретные фабрики 2
public class ConcreteFactory2 : AbstractFactory
{
    public override AbstractProductA CreateProductA()
    { return new ProductA2(); }

    public override AbstractProductB CreateProductB()
    { return new ProductB2(); }
}

// Представляет клиентов
class Client
{
```

```
        static void Main(string[] args)
        {
            AbstractFactory factory1 = new ConcreteFactory1();
            AbstractProductA productA = factory1.CreateProductA();
            Console.WriteLine(productA.GetData());
            AbstractProductB productB = factory1.CreateProductB();
            Console.WriteLine(productB.GetData());
            AbstractFactory factory2 = new ConcreteFactory2();
            productA = factory2.CreateProductA();
            Console.WriteLine(productA.GetData());
            productB = factory2.CreateProductB();
            Console.WriteLine(productB.GetData());
        }
    }
```

❒ ***Пример* 3.2. *Разработка решения на языке C# с использованием шаблона Абстрактная фабрика.***

Требуется разработать на языке C# решение, в котором используется шаблон Абстрактная фабрика для создания объектов из двух разных семейств классов (книги и журналы).

Первое семейство классов:

- **Book** – книга (абстрактный класс);
- **LibraryBook** – библиотечная книга;
- **ShopBook** – магазинная книга.

Второе семейство классов:

- **Magazine** – журнал (абстрактный класс);
- **LibraryMagazine** – библиотечный журнал;
- **ShopMagazine** – магазинный журнал.

Классы фабрик:

- **IFactory** – абстрактная фабрика (интерфейс);
- **LibraryFactory** – фабрика, содержащая методы для создания экземпляров классов **LibraryBook** и **LibraryMagazine**;
- **ShopFactory** – фабрика, содержащая методы для создания экземпляров классов **ShopBook** и **ShopMagazine**.

Модули указанных классов разместим в проекте библиотеки классов **FactoryLib**. Диаграмма классов данного проекта показана на рис. 2.5.

Исходный код интерфейса **IFactory** представлен в листинге 3.6, а в листингах 3.7 и 3.8 приведены исходные коды фабрик **LibraryFactory** и **ShopFactory**.

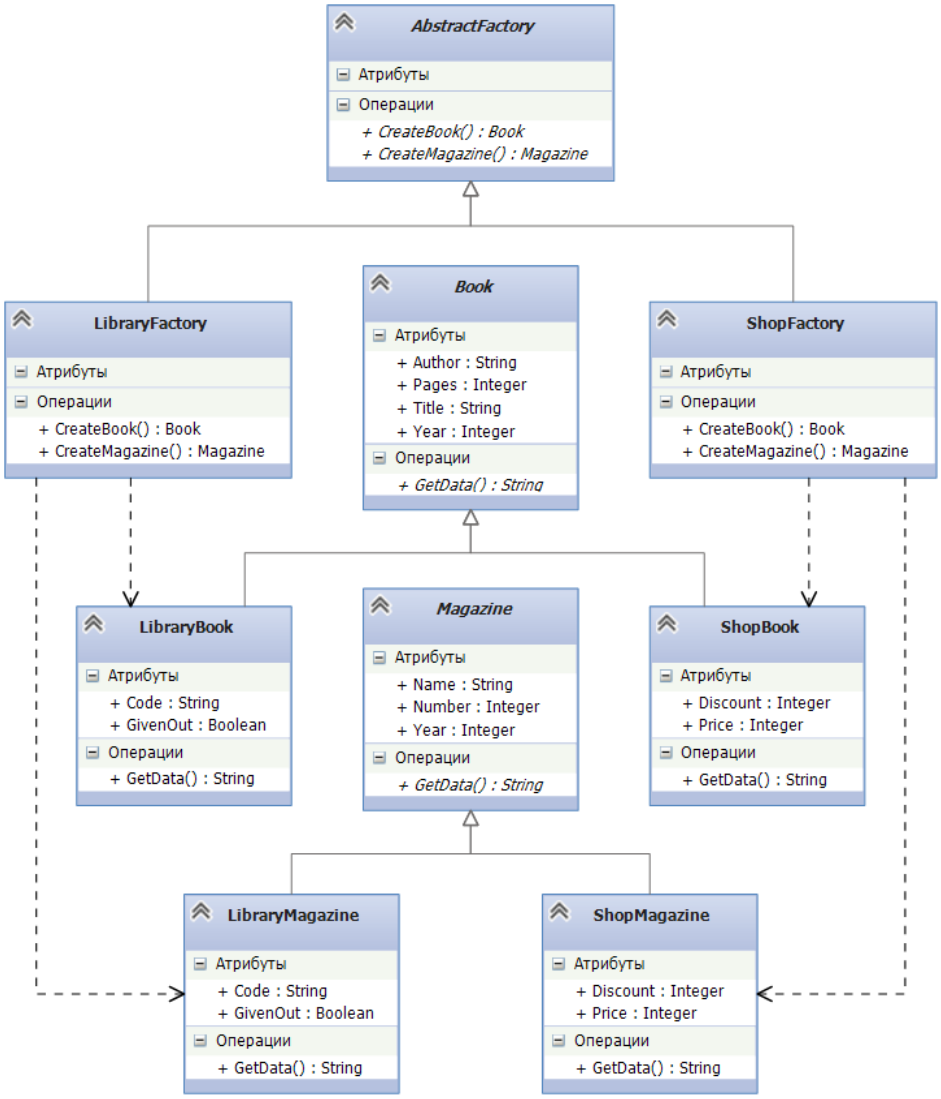

Рис. 3.5. Диаграмма классов для проекта **FactoryLib**

Листинг 3.6. Исходный код интерфейса **IFactory**

```
14      /// <summary>
15      /// Представляет абстрактные фабрики
16      /// </summary>
17      public interface IFactory
18      {
19          /// <summary>
20          /// Создать книгу (фабричный метод)
21          /// </summary>
22          /// <returns>Книга</returns>
23          Book CreateBook();
24
25          /// <summary>
26          /// Создать журнал (фабричный метод)
27          /// </summary>
28          /// <returns>Журнал</returns>
29          Magazine CreateMagazine();
30      }
```

Листинг 3.7. Исходный код класса **LibraryFactory**

```
14      /// <summary>
15      /// Представляет фабрики для создания элементов
16      /// библиотечного фонда (конкретная фабрика 1)
17      /// </summary>
18      public class LibraryFactory : IFactory
19      {
20          /// <summary>
21          /// Создать библиотечную книгу
22          /// </summary>
23          /// <returns>Книга</returns>
24          public Book CreateBook()
25          {
26              return new LibraryBook();
27          }
28
29          /// <summary>
30          /// Создать библиотечный журнал
31          /// </summary>
32          /// <returns>Журнал</returns>
33          public Magazine CreateMagazine()
34          {
35              return new LibraryMagazine();
36          }
37      }
```

Листинг 3.8. Исходный код класса **ShopFactory**

```
14      /// <summary>
15      /// Представляет фабрики для создания товаров
16      /// книжного магазина (конкретная фабрика 2)
17      /// </summary>
18      public class ShopFactory : IFactory
19      {
20          /// <summary>
21          /// Создать магазинную книгу
22          /// </summary>
23          /// <returns>Книга</returns>
24          public Book CreateBook()
25          {
26              return new ShopBook();
27          }
28
29          /// <summary>
30          /// Создать магазинный журнал
31          /// </summary>
32          /// <returns>Журнал</returns>
33          public Magazine CreateMagazine()
34          {
35              return new ShopMagazine();
36          }
37      }
```

Исходные коды классов, представляющих продукты (книги и журналы), приведены в листингах 3.9 – 3.14.

Листинг 3.9. Исходный код класса **Book**

```
14      /// <summary>
15      /// Представляет книги (абстрактный продукт A)
16      /// </summary>
17      public abstract class Book
18      {
19          /// <summary>
20          /// Заголовок
21          /// </summary>
22          public virtual string Title { get; set; }
23
24          /// <summary>
25          /// Автор
26          /// </summary>
27          public virtual string Author { get; set; }
28
29          /// <summary>
30          /// Год издания
31          /// </summary>
32          public virtual int Year { get; set; }
33
34          /// <summary>
35          /// Число страниц
36          /// </summary>
37          public virtual int Pages { get; set; }
38
39          /// <summary>
40          /// Конструктор по умолчанию
41          /// </summary>
42          public Book()
43          {
44              Title = "Применение UML 2.0 и шаблонов проектирования";
45              Author = "Ларман Крэг";
46              Year = 2013;
47              Pages = 736;
48          }
49
50          /// <summary>
51          /// Возвращает строку с данными о книге
52          /// </summary>
53          /// <returns>Строка с данными о книге</returns>
54          public abstract string GetData();
55      }
```

Листинг 3.10. Исходный код класса **LibraryBook**

```
14      /// <summary>
15      /// Представляет библиотечные книги (конкретный продукт A1)
16      /// </summary>
17      public class LibraryBook : Book
18      {
19          /// <summary>
20          /// Код
21          /// </summary>
22          public virtual string Code { get; set; }
23
24          /// <summary>
25          /// Наличие в библиотеке
26          /// </summary>
27          public virtual bool IsInLib { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public LibraryBook()
33              : base()
34          {
35              Code = "12.345.67";
36              IsInLib = true;
37          }
38
39          /// <summary>
40          /// Возвращает строку с данными о книге
41          /// </summary>
42          /// <returns>Строка с данными о книге</returns>
43          public override string GetData()
44          {
45              return string.Format("Библ. книга. Код: {0}", Code);
46          }
47      }
```

Листинг 3.11. Исходный код класса **ShopBook**

```
14      /// <summary>
15      /// Представляет магазинные книги (конкретный продукт A2)
16      /// </summary>
17      public class ShopBook : Book
18      {
19          /// <summary>
20          /// Цена, руб.
21          /// </summary>
22          public virtual int Price { get; set; }
23
24          /// <summary>
25          /// Скидка, %
26          /// </summary>
27          public virtual int Discount { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public ShopBook()
33              : base()
34          {
35              Price = 300;
36              Discount = 0;
37          }
38
39          /// <summary>
40          /// Возвращает строку с данными о книге
41          /// </summary>
42          /// <returns>Строка с данными о книге</returns>
43          public override string GetData()
44          {
45              return string.Format("Магаз. книга. Цена: {0} руб.", Price);
46          }
47      }
```

Листинг 3.12. Исходный код класса **Magazine**

```
14      /// <summary>
15      /// Представляет журналы (абстрактный продукт B)
16      /// </summary>
17      public abstract class Magazine
18      {
19          /// <summary>
20          /// Название
21          /// </summary>
22          public virtual string Name { get; set; }
23
24          /// <summary>
25          /// Год издания
26          /// </summary>
27          public virtual int Year { get; set; }
28
29          /// <summary>
30          /// Номер
31          /// </summary>
32          public virtual int Number { get; set; }
33
34          /// <summary>
35          /// Конструктор по умолчанию
36          /// </summary>
37          public Magazine()
38          {
39              Name = "Открытые системы";
40              Year = 2014;
41              Number = 1;
42          }
43
44          /// <summary>
45          /// Возвращает строку с данными о журнале
46          /// </summary>
47          /// <returns>Строка с данными о журнале</returns>
48          public abstract string GetData();
49      }
```

Листинг 3.13. Исходный код класса **LibraryMagazine**

```
14      /// <summary>
15      /// Представляет библиотечные журналы (конкретный продукт B2)
16      /// </summary>
17      public class LibraryMagazine : Magazine
18      {
19          /// <summary>
20          /// Код
21          /// </summary>
22          public virtual string Code { get; set; }
23
24          /// <summary>
25          /// Наличие в библиотеке
26          /// </summary>
27          public virtual bool IsInLib { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public LibraryMagazine()
33              : base()
34          {
35              Code = "12.345.67";
36              IsInLib = true;
37          }
38
39          /// <summary>
40          /// Возвращает строку с данными о журнале
41          /// </summary>
42          /// <returns>Строка с данными о журнале</returns>
43          public override string GetData()
44          {
45              return string.Format("Библ. журнал. Код: {0}", Code);
46          }
47      }
```

Для работы с полученными классами добавим в решение проект приложения Windows Forms. Интерфейс пользователя для экранной формы **Form1** показан на рис. 3.6.

Исходный код класса **Form1** представлен в листинге 3.15.

Листинг 3.14. Исходный код класса **ShopMagazine**

```
14      /// <summary>
15      /// Представляет магазинные журналы (конкретный продукт B2)
16      /// </summary>
17      public class ShopMagazine : Magazine
18      {
19          /// <summary>
20          /// Цена, руб.
21          /// </summary>
22          public virtual int Price { get; set; }
23
24          /// <summary>
25          /// Скидка, %
26          /// </summary>
27          public virtual int Discount { get; set; }
28
29          /// <summary>
30          /// Конструктор по умолчанию
31          /// </summary>
32          public ShopMagazine()
33              : base()
34          {
35              Price = 100;
36              Discount = 0;
37          }
38
39          /// <summary>
40          /// Возвращает строку с данными о журнале
41          /// </summary>
42          /// <returns>Строка с данными о журнале</returns>
43          public override string GetData()
44          {
45              return string.Format("Магаз. журнал. Цена: {0} руб.", Price);
46          }
47      }
```

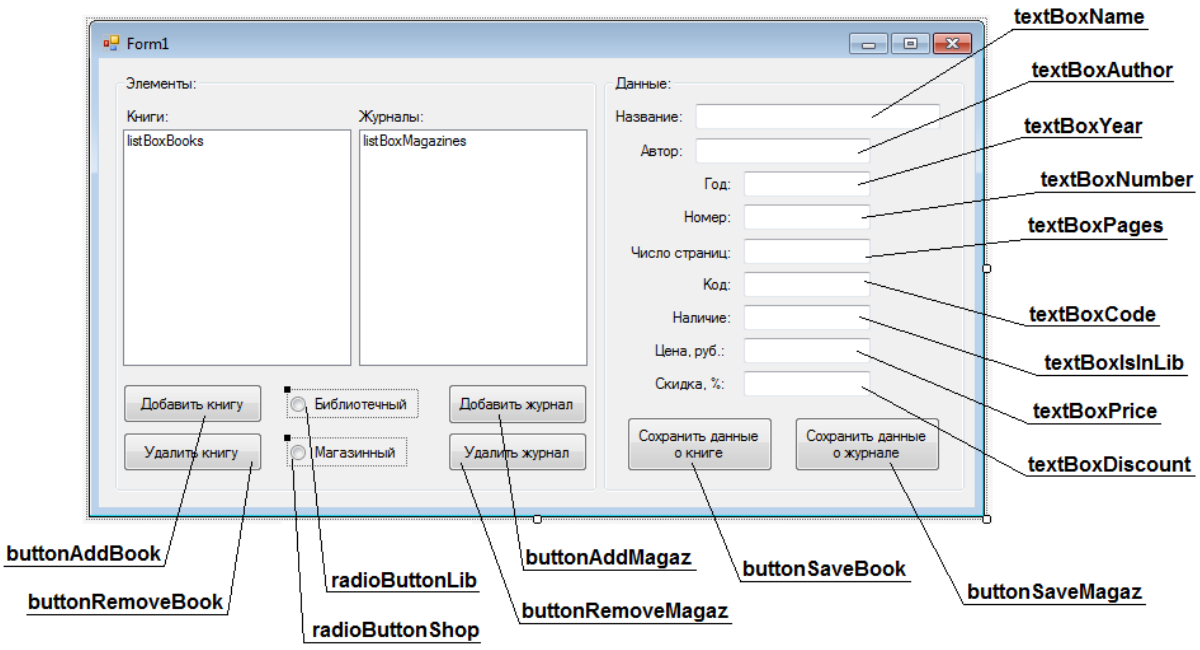

Рис. 3.6. Интерфейс пользователя приложения

Листинг 3.15. Исходный код класса **Form1** (часть 1)

```
14  public partial class Form1 : Form
15  {
16      List<Book> books; // Список книг
17      List<Magazine> magazines; // Список журналов
18      IFactory factory; // Фабрика
19
20      public Form1()
21      {
22          InitializeComponent();
23          books = new List<Book>();
24          magazines = new List<Magazine>();
25          factory = new LibraryFactory();
26      }
27
28      /// <summary>
29      /// Заполнить элементами список listBoxBooks
30      /// </summary>
31      private void GenerBooksList()
32      {
33          listBoxBooks.Items.Clear();
34          int i = 0;
35          foreach (Book b in books)
36          {
37              listBoxBooks.Items.Add(string.Format("{0} - {1}",
38                  ++i, b.GetData()));
39          }
40      }
41
42      /// <summary>
43      /// Заполнить элементами список listBoxMagazines
44      /// </summary>
45      private void GenerMagazinesList()
46      {
47          listBoxMagazines.Items.Clear();
48          int i = 0;
49          foreach (Magazine m in magazines)
50          {
51              listBoxMagazines.Items.Add(string.Format("{0} - {1}",
52                  ++i, m.GetData()));
53          }
54      }
55
56      /// <summary>
57      /// Выбрать конкретный объект-фабрику
58      /// </summary>
59      private void SelectFactory()
60      {
61          if (radioButtonLib.Checked) factory = new LibraryFactory();
62          else if (radioButtonShop.Checked) factory = new ShopFactory();
63      }
```

Листинг 3.15. Исходный код класса **Form1** (часть 2)

```
65          private void Form1_Load(object sender, EventArgs e)
66          {
67              this.Text = "Работа с шаблоном Абстрактная фабрика. " +
68                          "Турчин Д.Е. (каф. ИиАПС)";
69              radioButtonLib.Checked = true;
70              GenerBooksList();
71              GenerMagazinesList();
72          }
73
74          private void buttonAddBook_Click(object sender, EventArgs e)
75          {
76              SelectFactory();
77              books.Add(factory.CreateBook());
78              GenerBooksList();
79          }
80
81          private void buttonAddMagaz_Click(object sender, EventArgs e)
82          {
83              SelectFactory();
84              magazines.Add(factory.CreateMagazine());
85              GenerMagazinesList();
86          }
87
88          private void listBoxBooks_SelectedIndexChanged(object sender, EventArgs e)
89          {
90              int index = listBoxBooks.SelectedIndex;
91              if (index < 0) return;
92              textBoxName.Text = books[index].Title;
93              textBoxAuthor.Text = books[index].Author;
94              textBoxYear.Text = Convert.ToString(books[index].Year);
95              textBoxNumber.Text = "";
96              textBoxPages.Text = Convert.ToString(books[index].Pages);
97              if (books[index] is LibraryBook)
98              {
99                  LibraryBook librBook = books[index] as LibraryBook;
100                 textBoxCode.Text = librBook.Code;
101                 textBoxIsInLib.Text = Convert.ToString(librBook.IsInLib);
102                 textBoxPrice.Text = "";
103                 textBoxDiscount.Text = "";
104             }
105             else if (books[index] is ShopBook)
106             {
107                 ShopBook shopBook = books[index] as ShopBook;
108                 textBoxPrice.Text = Convert.ToString(shopBook.Price);
109                 textBoxDiscount.Text = Convert.ToString(shopBook.Discount);
110                 textBoxCode.Text = "";
111                 textBoxIsInLib.Text = "";
112             }
113             else { MessageBox.Show("Неизвестный тип продукта!"); }
114         }
```

Листинг 3.15. Исходный код класса **Form1** (часть 3)

```
116         private void listBoxMagazines_SelectedIndexChanged(object sender, EventArgs e)
117         {
118             int index = listBoxMagazines.SelectedIndex;
119             if (index < 0) return;
120             textBoxName.Text = magazines[index].Name;
121             textBoxAuthor.Text = "";
122             textBoxYear.Text = Convert.ToString(magazines[index].Year);
123             textBoxNumber.Text = Convert.ToString(magazines[index].Number);
124             textBoxPages.Text = "";
125             if (magazines[index] is LibraryMagazine)
126             {
127                 LibraryMagazine librMagaz = magazines[index] as LibraryMagazine;
128                 textBoxCode.Text = librMagaz.Code;
129                 textBoxIsInLib.Text = Convert.ToString(librMagaz.IsInLib);
130                 textBoxPrice.Text = "";
131                 textBoxDiscount.Text = "";
132             }
133             else if (magazines[index] is ShopMagazine)
134             {
135                 ShopMagazine shopMagaz = magazines[index] as ShopMagazine;
136                 textBoxPrice.Text = Convert.ToString(shopMagaz.Price);
137                 textBoxDiscount.Text = Convert.ToString(shopMagaz.Discount);
138                 textBoxCode.Text = "";
139                 textBoxIsInLib.Text = "";
140             }
141             else { MessageBox.Show("Неизвестный тип продукта!"); }
142         }
143 
144         private void buttonSaveBook_Click(object sender, EventArgs e)
145         {
146             int index = listBoxBooks.SelectedIndex;
147             if (index < 0) return;
148             books[index].Title = textBoxName.Text;
149             books[index].Author = textBoxAuthor.Text;
150             books[index].Year = Convert.ToInt32(textBoxYear.Text);
151             books[index].Pages = Convert.ToInt32(textBoxPages.Text);
152             if (books[index] is LibraryBook)
153             {
154                 LibraryBook librBook = books[index] as LibraryBook;
155                 librBook.Code = textBoxCode.Text;
156                 librBook.IsInLib = Convert.ToBoolean(textBoxIsInLib.Text);
157             }
158             else if (books[index] is ShopBook)
159             {
160                 ShopBook shopBook = books[index] as ShopBook;
161                 shopBook.Price = Convert.ToInt32(textBoxPrice.Text);
162                 shopBook.Discount = Convert.ToInt32(textBoxDiscount.Text);
163             }
164             else
165             {
166                 MessageBox.Show("Неизвестный тип продукта!");
167                 return;
168             }
169             GenerBooksList();
170         }
```

Листинг 3.15. Исходный код класса **Form1** (часть 4)

```
172         private void buttonSaveMagaz_Click(object sender, EventArgs e)
173         {
174             int index = listBoxMagazines.SelectedIndex;
175             if (index < 0) return;
176             magazines[index].Name = textBoxName.Text;
177             magazines[index].Year = Convert.ToInt32(textBoxYear.Text);
178             magazines[index].Number = Convert.ToInt32(textBoxNumber.Text);
179             if (magazines[index] is LibraryMagazine)
180             {
181                 LibraryMagazine librMagaz = magazines[index] as LibraryMagazine;
182                 librMagaz.Code = textBoxCode.Text;
183                 librMagaz.IsInLib = Convert.ToBoolean(textBoxIsInLib.Text);
184             }
185             else if (magazines[index] is ShopMagazine)
186             {
187                 ShopMagazine shopMagaz = magazines[index] as ShopMagazine;
188                 shopMagaz.Price = Convert.ToInt32(textBoxPrice.Text);
189                 shopMagaz.Discount = Convert.ToInt32(textBoxDiscount.Text);
190             }
191             else
192             {
193                 MessageBox.Show("Неизвестный тип продукта!");
194                 return;
195             }
196             GenerMagazinesList();
197         }
198
199         private void buttonRemoveBook_Click(object sender, EventArgs e)
200         {
201             int index = listBoxBooks.SelectedIndex;
202             if (index < 0) return;
203             books.RemoveAt(index);
204             GenerBooksList();
205         }
206
207         private void buttonRemoveMagaz_Click(object sender, EventArgs e)
208         {
209             int index = listBoxMagazines.SelectedIndex;
210             if (index < 0) return;
211             magazines.RemoveAt(index);
212             GenerMagazinesList();
213         }
214     }
```

Главное окно работающего приложения Windows Forms показано на рис. 3.7. ❒

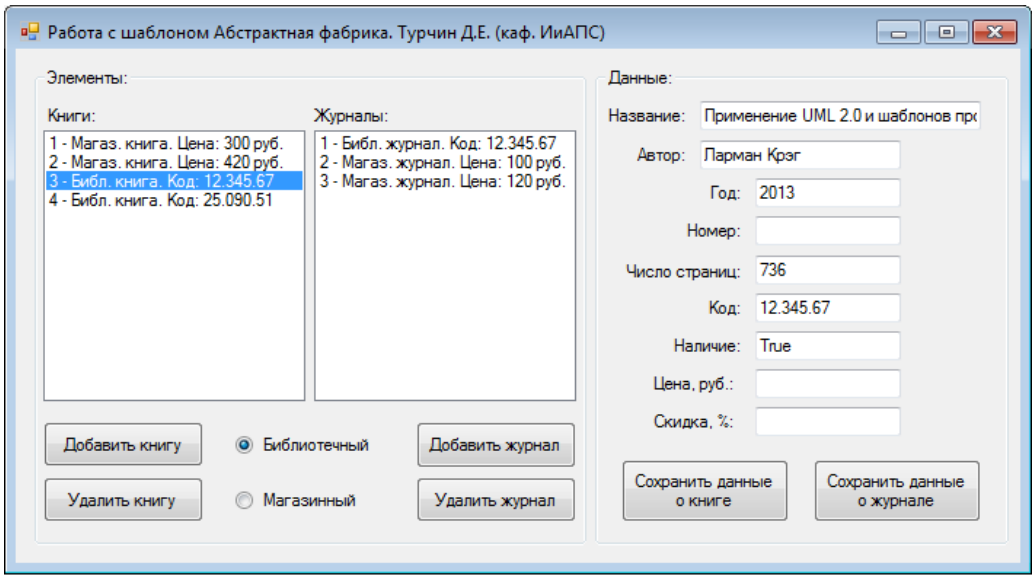

Рис. 3.7. Главное окно приложения Windows Forms

3.3. 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. 3.1), участвующие в шаблоне Фабричный метод. Добавить в решение проект консольного приложения, в котором создаются экземпляры заданных классов с использованием фабричных методов.
3. Разработать библиотеку классов, содержащую заданные классы (табл. 3.2), которые используются для реализации шаблона Абстрактная фабрика. Добавить в решение проект приложения Windows Forms для демонстрации процесса создания экземпляров классов.
4. Оформить и защитить отчет по лабораторной работе.

Таблица 3.1

Варианты заданий для разработки приложения, использующего шаблон Фабричный метод

№ вар.	Абстрактный класс	Конкретные классы
1, 7, 13, 19	**Плоская фигура**. ***Атрибуты***: • Координата X. • Координата Y. • Угол поворота. ***Операции***: • Определить площадь. • Получить данные.	**Прямоугольник** (Длина, Ширина). **Эллипс** (Полуось A, Полуось B). **Равнобедренный треугольник** (Длина основания, Угол между основанием и бедром).
2, 8, 14, 20	**Твёрдое тело** (погружено в жидкость). • Плотность материала тела. • Плотность жидкости. • Глубина погружения тела. ***Операции***: • Определить архимедову силу. • Получить данные.	**Шар** (Радиус). **Прямая круглая труба** (Длина, Наружный диаметр, Внутренний диаметр) **Круглая пластина** (Диаметр, Толщина).
3, 9, 15, 21	**Объёмная фигура**. ***Атрибуты***: • Координата X. • Координата Y. • Координата Z. ***Операции***: • Определить объём. • Получить данные.	**Прямой круговой цилиндр** (Диаметр, Высота). **Прямой круговой конус** (Радиус основания, Высота). **Куб** (Длина стороны).
4, 10, 16, 22	**Плоская фигура**. ***Атрибуты***: • Координата X. • Координата Y. • Масштабный коэффициент. ***Операции***: • Определить периметр. • Получить данные.	**Ромб** (Длина стороны, Меньший угол между сторонами). **Эллипс** (Полуось A, Полуось B). **Прямоугольный треугольник** (Длина первого катета, Длина второго катета).
5, 11, 17, 23	**Твёрдое тело** (погружено в жидкость). • Плотность материала тела. • Плотность жидкости.	**Стержень круглого сечения** (Длина, Диаметр). **Кольцо круглого сечения** (Наружный диаметр, Внутрен-

№ вар.	Абстрактный класс	Конкретные классы
	• Глубина погружения тела. ***Операции***: • Определить вес тела. • Получить данные.	ний диаметр) **Квадратная пластина** (Длина стороны, Толщина).
6, 12, 18, 24	**Объёмная фигура**. ***Атрибуты***: • Координата X. • Координата Y. • Координата Z. ***Операции***: • Определить площадь поверхности. • Получить данные.	**Параллелепипед** (Длина, Ширина, Высота). **Тор** (Радиус поперечного сечения, расстояние от центра поперечного сечения до оси вращения) **Шар** (Радиус).

Таблица 3.2

Варианты заданий для разработки приложения, использующего шаблон Абстрактная фабрика

№ вар.	Семейство классов 1	Семейство классов 2
1, 5, 9, 13, 17, 21	**Кредит** (сумма, дата получения, срок, процент) **Кредит для физических лиц** (ФИО заёмщика, серия и номер паспорта) **Кредит для юридических лиц** (Название организации, адрес регистрации)	**Вклад** (номер счёта, сумма, процент). **Вклад для физических лиц** (ФИО вкладчика, серия и номер паспорта). **Вклад для юридических лиц** (Название организации, адрес регистрации)
2, 6, 10, 14, 20, 22	**Клавиатура** (цвет, наличие подставки для рук, цена) **Проводная клавиатура** (длина провода, интерфейс (PS/2, USB)) **Беспроводная клавиатура** (радиус действия, батарея)	**Мышь** (цвет, число кнопок, цена) **Проводная мышь** (длина провода, интерфейс (PS/2, USB)) **Беспроводная мышь** (радиус действия, батарея)
3, 7, 11, 15, 19,	**Одежда** (наименование, размер, цвет, цена). **Летняя одежда** (материал, влагостойкость). **Зимняя одежда** (утеплитель,	**Обувь** (наименование, размер, материал верха, цена). **Летняя обувь** (цвет,). **Зимняя обувь** (утеплитель, подкладка).

№ вар.	Семейство классов 1	Семейство классов 2
23	нижняя температура).	
4, 8, 12, 16, 20, 24	**Студент** (номер зачётной книжки, ФИО, пол, группа). **Студент-бюджетник** (средний бал за семестр, размер стипендии). **Студент-контрактник** (наличие оплаты за текущий семестр, дата оплаты).	**Аспирант** (ФИО, пол, научный руководитель, тема диссертации). **Аспирант-бюджетник** (число статей, размер стипендии). **Аспирант-контрактник** (наличие оплаты за текущий год, дата оплаты).

3.4. Контрольные вопросы

1. Для чего предназначены порождающие шаблоны проектирования?
2. Какие из шаблонов проектирования отнесены к порождающим в каталоге GoF?
3. Что понимают под Фабричным методом?
4. Какую проблему позволяет решить шаблон Фабричный метод?
5. Каково назначение шаблона Абстрактная фабрика?
6. Какое решение проблемы предлагается в шаблоне Абстрактная фабрика?
7. За что отвечают классы конкретных фабрик в шаблоне Абстрактная фабрика?

4. ОСНОВЫ РАБОТЫ СО СТИЛЯМИ И ШАБЛОНАМИ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ В ПРИЛОЖЕНИЯХ WPF

4.1. Цель и задачи работы

Цель работы – прибрести умение работать со стилями и шаблонами элементов управления в приложениях WPF.

Основные задачи:

• ознакомиться с использованием кистей, стилей и шаблонов элементов управления WPF;

• научиться работать со стилями в приложениях WPF;

• освоить создание шаблонов элементов управления WPF.

Работа рассчитана на 4 часа.

4.2. Основные теоретические сведения

4.2.1. Ресурсы WPF. Цвета и кисти. Задание градиентных кистей

Общие сведения о ресурсах WPF.

Важной составной частью WPF является система ресурсов, представляющая собой способ поддержания набора полезных объектов.

Под ***ресурсом*** в WPF понимают объект, который можно многократно использовать в разных местах приложения WPF. Использование ресурсов позволяет сократить число повторяющихся фрагментов кода и упростить модификацию приложения.

К ресурсам в приложениях WPF могут относиться такие объекты, таких как кисти, стили, анимации, шаблоны элементов управления и данных.

Можно определить ресурсы в любом элементе XAML, с помощью свойство **Resources**. Однако наиболее часто ресурсы определяются в корневом элементе XAML-документа (**Application**, **Window**, **Page** и др.).

Когда требуется определить ресурс, для установки свойства применяют синтаксис «свойство-элемент». Кроме того, элементу ресурса задается значение **x:Key**, которое может быть использовано другими частями элемента, когда потребуется обратиться к ресурсу.

Синтакисис объявления ресурса имеет следующий вид:

```
<ИмяЭлемента>
  <ИмяЭлемента.Resources>
    <ТипРесурса x:Key="ИмяРесурса">
      <!-- Описание свойств ресурса -->
    </ТипРесурса>
  </ИмяЭлемента.Resources>
</ИмяЭлемента>
```

Когда требуется применить ресурс с указанными ключевым именем к выбранному элементу XAML, используют расширение разметки **{ … }**. При этом указывают ключевое слово **StaticResource** (статический ресурс) или **DinamicResource** (динамический ресурс), а также ключевое имя ресурса:

```
<ИмяЭлемента Свойство="{ StaticResource ИмяРесурса}">
```

Статические ресурсы устанавливаются только один раз и остаются подключенными к объекту в течении всего времени работы приложения. Статические ресурсы должны быть предварительно определены в коде XAML перед ссылкой на них.

В процессе работы приложения ресурс, объявленный как ***динамический***, для использующего его элемента может быть заменен ресурсом с другим ключевым именем.

В том случае, когда ресурсы должны многократно использоваться во многих проектах WPF, определяют ***словари ресурсов***. Для задания словаря ресурса создают отдельный документ XAML, в котором не содержится ничего кроме коллекции ресурсов. Корневым элементом такого документа является **ResourceDictionary**.

Цвета и кисти. Настройка кистей в Visual Studio.

Цвет в WPF задается структурой **Color**, определенной в пространстве имен **System.Windows.Media**. Структура **Color**

может работать с цветовым пространством RGB, в котором цвет задается уровнями интенсивности красной, зеленой и синей составляющих (свойства **R**, **G**, и **B** соответственно).

Структура **Color** также поддерживает цветовое пространство ARGB, в котором к трем свойствам RGB добавлено байтовое свойство **A**, называемое ***альфа-каналом*** и предназначенное для управления прозрачностью цвета. При **A**=0 цвет полностью прозрачен, а при **A**=255 цвет полностью непрозрачен. Промежуточные значения **A** соответствуют разным степеням прозрачности.

Кроме того, структура **Color** позволяет работать с цветовым пространством scRGB, в котором цветовые компоненты задаются в виде вещественных свойств с одинарной точностью **ScA**, **ScR**, **ScG** и **ScB**.

Пространство имен **System.Windows.Media** также содержит класс **Colors**, в котором присутствует 141 статическое свойство, доступное только для чтения. Имена свойств идут в алфавитном порядке от **AliceBlue** до **YellowGreen**. Все названия цветов, кроме одного, совпадают с теми, которые поддерживают Web-браузеры. Единственное исключение составляет свойство **Transparent**, которое возвращает значение цвета с нулевым альфа-каналом.

В приложениях WPF для управления эффектами при заполнении фона используются ***кисти***, задаваемые классами, производными от класса **Brush**. Класс **Brush** определен в пространстве имен **System.Windows.Media**.

Основными классами кистей являются:

• **SolidColorBrush** – рисует область с использованием одного сплошного цвета;

• **LinearGradientBrush** – рисует область с использованием линейного градиентного заполнения, при котором наблюдается плавный переход от одного цвета к другому вдоль прямой линии;

• **RadialGradientBrush** – рисует область с использованием радиального градиентного заполнения, представляющего собой плавный переход от одного цвета к другому в радиальном направлении от центральной точки;

• **ImageBrush** – рисует область, используя графическое изображение, которое может растягиваться, масштабироваться или многократно повторяться.

Для работы с кистями в Visual Studio встроен редактор кистей (рис. 4.1), позволяющий вручную настраивать свойство **Fill** различных объектов. Открыть редактор кистей можно, раскрыв список **Кисти** в окне **Properties (Свойства)** и щелкнув мышью на свойстве **Fill.**

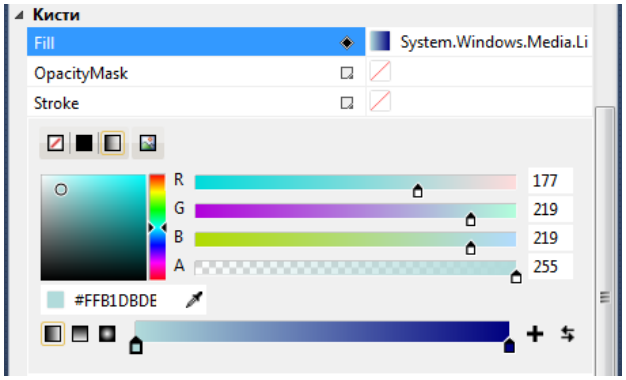

Рис. 4.1. Редактор кистей в окне свойств Visual Studio

Редактор кистей содержит четыре ползунка, которые позволяют устанавливать значение ARGB (цветовые компоненты Alpha, Red, Green, Blue) для текущей кисти. Используя эти ползунки можно установить любой сплошной цвет. Кроме того, этот редактор позволяет настраивать градиентные кисти.

Задание градиентных кистей.

Кисти **LinearGradientBrush** и **RadialGradientBrush** позволяют осуществлять градиентное закрашивание заданной области. При этом **LinearGradientBrush** применяет алгоритм смещения цветов вдоль вектора, а **RadialGradientBrush** смещает цвета радиально, начиная с указанной точки.

Кисть **LinearGradientBrush** задается с помощью таких свойств, как **StartPoint** и **EndPoint**, а также коллекции объектов **GradientStop**, имеющих свойства **Color** и **Offset.** (рис. 4.2).

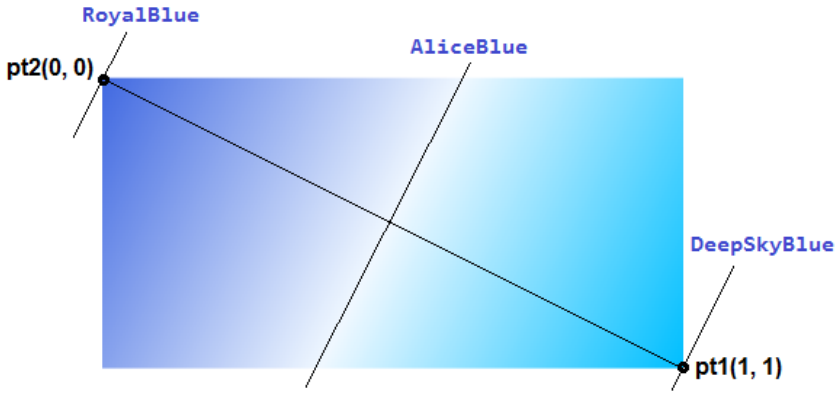

Рис. 4.2. Заполнение с помощью линейного градиента

Направление распространения линейного градиента управляется следующими свойствами (рис. 4.3):

• **StartPoint** – координаты первой точки, через которую проходит линия распространения градиента;

• **EndPoint** – координаты второй точки линии распространения градиента.

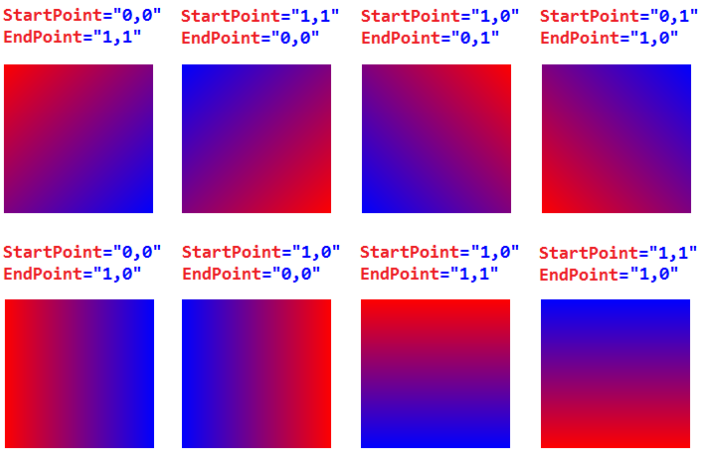

Рис. 4.3. Примеры использования кисти **LinearGradientBrush** с разными значениями свойств **StartPoint** и **EndPoint**

Элементы **GradientStop** определяют два свойства:

• **Color** – задает цвет области линейного градиента, начиная от **StartPoint**;

• **Offset** – относительное расстояние между **StartPoint** и **EndPoint**; значение свойства **Offset** лежит в интервале от 0 до 1.

Пример задания линейной градиентной кисти для элемента прямоугольник (**Rectangle**):

```
<Rectangle Width="150" Height="150">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
      <GradientStop Color="Red" Offset="0" />
      <GradientStop Color="Blue" Offset="1" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

Кисть **RadialGradientBrush** задается с помощью свойств **GradientOrigin**, **Center**, **RadiusX**, **RadiusY** а также коллекции объектов **GradientStop**.

Радиальный градиент распространяется из начальной точки в радиальном направлении. В конечном итоге он достигает границы внутренней области градиента (рис. 4.4). За пределами этой области производится заполнение цветом, определенным последним в коллекции **RadialGradientBrush.GradientStops**.

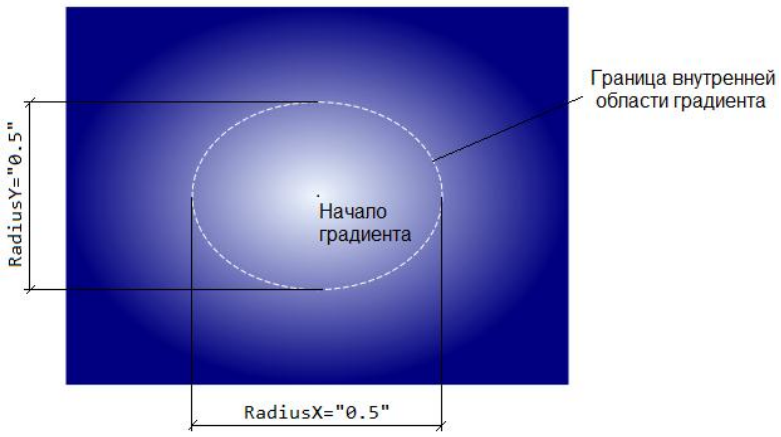

Рис. 4.4. Заполнение с помощью радиального градиента

Для идентификации точки, с которой начинается первый цвет градиента, служит свойство **GradientOrigin**. По умолчанию оно хранит координату «0.5, 0.5», представляющую середину заполняемой области.

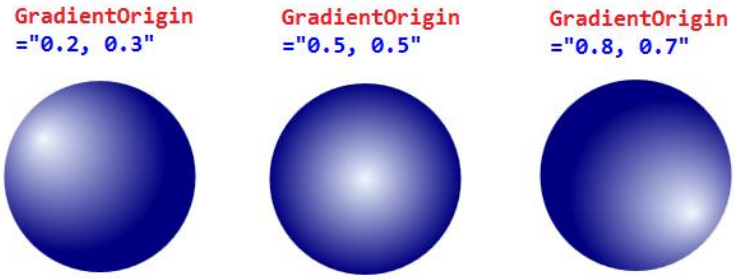

Рис. 4.5. Примеры использования кисти **RadialGradientBrush** с разными значениями свойства **GradientOrigin**

Граница внутренней области задается тремя параметрами:

• **Center** – задает положение центральной точки закрашиваемой области (значение по умолчанию – «0.5, 0.5»);

• **RadiusX** – радиус внутренней области вдоль горизонтальной оси;

• **RadiusY** – радиус внутренней области вдоль вертикальной оси.

Пример задания радиальной градиентной кисти для элемента элипс (**Ellipse**):

```
<Ellipse Width="150" Height="150">
  <Ellipse.Fill>
    <RadialGradientBrush Center="0.5, 0.5"
                         RadiusX="1" RadiusY="1"
                         GradientOrigin="0.2, 0.3">
      <GradientStop Color="AliceBlue" Offset="0" />
      <GradientStop Color="Navy" Offset="0.5" />
    </RadialGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```

4.2.2. Стили и шаблоны элементов управления WPF

Общие сведения о стилях и шаблонах элементов управления WPF.

Под ***стилем*** в WPF понимают совокупность значений свойств, которые можно применить к нужному элементу. То есть стиль WPF – это объект, который поддерживает коллекцию пар «свойство-значение».

Система стилей WPF играет ту же роль, что и стандарт каскадных таблиц стилей (CSS) при разработке Web-страниц. Использование стилей обеспечивает повторное использование вариантов форматирования.

Каждый стиль представлен классом **System.Windows.Style**. Этот класс имеет свойство **Setters**, которое является коллекцией объектов **Setter**. Именно объект **Setter** позволяет определять пары «свойство-значение». Другими важными элементами класса **Style** являются:

Стили WPF поддерживают ***триггеры***, которые позволяют изменять стиль элемента при изменении определенного свойства.

В WPF отсутствует возможность применить один стиль к элементам разных типов.

В WPF каждый элемент управления имеет встроенное средство, определяющее способ его визуализации с помощью блока XAML-разметки и называемое ***шаблоном элемента управления*** (*control template*). С помощью шаблонов можно создавать новые элементы управления на основе существующих.

Шаблон элемента управления задается как ресурс WPF с помощью элемента **ControlTemplate**.

❒ ***Пример* 4.1. *Разработка стилей и шаблонов элементов управления для приложения WPF*.**

Требуется добавить кисти, стили и шаблоны для элементов управления из приложения WPF со страничной навигацией (пример 3.2).

Определим в документе **App.xaml** следующие стили и шаблоны элементов управления:

• стиль по умолчанию для всех элементов **Hyperlink**;

• **LabelStyle** – стиль меток, задающих заголовки окон; дополнительно зададим в данном стиле линейную градиентную кисть;

• **LabelStyle2** – стиль меток, задающих заголовки страниц; данный стиль определим на основе стиля **LabelStyle**;

• **DataGridStyle** – стиль для таблиц **DataGrid**;

• **MenuTemplate** – шаблон для главного меню, в котором должны быть определены такие пункты, как «Редактировать», «Отменить», «Создать», «Удалить», «Выход», «Отчет»;

• **StatusBarTemplate** – шаблон для строки состояния (элемент **StatusBar**).

Для окна **MainWindow**, а также для всех загружаемых в него страниц определим линейную градиентную кисть **WindowBackground**.

Дополнительно добавим в приложение окно **DiagramWindow** для построения диаграмм и формирования отчетов. Данное окно должно открываться при выборе пункта **Отчет** главного меню.

В окне **DiagramWindow** поместим элемент **Canvas**, а также пару кнопок (элементы **Button**): «Построить» и «Очистить». Кроме того, в ресурсах данного окна добавим радиальную градиентную кисть **BackgrDiagramWindow**. Для кнопок создадим стиль **buttonStyle**, который будет определять их фон и некоторые другие свойства.

Код XAML-документа **App.xaml** с определенными в нем стилями и шаблонами приведен в листингах 4.1 и 4.2. Код XAML-документа **MainWindow.xaml** для главного окна с определенной в нем кистью показан в листинге 4.3.

Листинг 4.1. Код XAML-документа **App.xaml** (часть 1)

```
<Application x:Class="WpfAppCommData.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <!-- Определение стиля ссылок -->
        <Style TargetType="{x:Type Hyperlink}">
            <Setter Property="FontFamily" Value="Calibri" />
            <Setter Property="FontSize" Value="14" />
            <Setter Property="FontStyle" Value="Italic" />
            <Setter Property="Foreground" Value="Blue" />
        </Style>
        <!-- Определение стиля текстовых меток -->
        <Style x:Key="LabelStyle" TargetType="{x:Type Label}">
            <Setter Property="FontFamily" Value="Impact" />
            <Setter Property="FontSize" Value="22" />
            <Setter Property="Margin" Value="10" />
            <Setter Property="HorizontalAlignment" Value="Center" />
            <Setter Property="Foreground" Value="Navy" />
            <!-- Задание линейной градиентной кисти для текста -->
            <Setter Property="Label.Foreground">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.2" Color="Navy" />
                        <GradientStop Offset="0.5" Color="#5555BB" />
                        <GradientStop Offset="0.8" Color="Black" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Style>

        <!-- Определение стиля 2 текстовых меток -->
        <Style x:Key="LabelStyle2" BasedOn="{StaticResource LabelStyle}"
               TargetType="{x:Type Label}">
            <Setter Property="FontFamily" Value="Arial" />
            <Setter Property="FontSize" Value="16" />
            <Setter Property="FontWeight" Value="Bold" />
            <Setter Property="Margin" Value="10" />
        </Style>
```

Листинг 4.2. Код XAML-документа **App.xaml** (часть 2)

```
        <!-- Определение стиля для элементов DataGrid -->
        <Style x:Key="DataGridStyle" TargetType="{x:Type DataGrid}">
            <!-- Отключение автоматического создания столбцов -->
            <Setter Property="AutoGenerateColumns" Value="False" />
            <!-- Разрешение редактирования, добавления и удаления строк -->
            <Setter Property="IsReadOnly" Value="True" />
            <Setter Property="CanUserAddRows" Value="True" />
            <Setter Property="CanUserDeleteRows" Value="True" />
            <!-- Задание чередования цветов фона для строк -->
            <Setter Property="RowBackground" Value="plum" />
            <Setter Property="AlternatingRowBackground" Value="palegreen" />
            <!-- Задание цвета и толщины рамки таблицы -->
            <Setter Property="BorderBrush" Value="#5555BB" />
            <Setter Property="BorderThickness" Value="3" />
            <!-- Задание фона и курсора для таблицы -->
            <Setter Property="Background" Value="Transparent" />
            <Setter Property="Cursor" Value="Hand" />
            <!-- Задание отображения деталей строк при их выделении -->
            <Setter Property="RowDetailsVisibilityMode" Value="VisibleWhenSelected" />
        </Style>

        <!-- Определение шаблона меню -->
        <ControlTemplate x:Key="MenuTemplate" TargetType="{x:Type Menu}">
            <Menu>
                <MenuItem Header="Действие" BorderThickness="3">
                    <MenuItem Header="Отменить" Command="Undo" />
                    <Separator />
                    <MenuItem Header="Создать" Command="New" />
                    <MenuItem Header="Редактировать" />
                    <MenuItem Header="Открыть" Command="Open" />
                    <MenuItem Header="Сохранить" Command="Save" />
                    <Separator />
                    <MenuItem Header="Удалить" Command="Delete" />
                </MenuItem>
                <MenuItem Header="Отчет" />
            </Menu>
        </ControlTemplate>

        <!-- Определение шаблона строки состояния -->
        <ControlTemplate x:Key="StatusBarTemplate" TargetType="{x:Type StatusBar}">
            <StatusBar Background="Gainsboro">
                <StatusBarItem>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="Готово" />
                        <TextBlock Text="" />
                    </StackPanel>
                </StatusBarItem>
            </StatusBar>
        </ControlTemplate>

    </Application.Resources>
</Application>
```

Листинг 4.3. Код XAML-документа **MainWindow.xaml**

```
<Window x:Class="WpfAppCommData.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Приложение со страничной навигацией" Height="350" Width="500">
    <Window.Resources>
        <!-- Определение кисти для главного окна -->
        <LinearGradientBrush x:Key="WindowBackground" StartPoint="0,0" EndPoint="1,1">
            <GradientStop Offset="0.0" Color="DeepSkyBlue" />
            <GradientStop Offset="0.5" Color="#DDFFDD" />
            <GradientStop Offset="1.0" Color="RoyalBlue" />
        </LinearGradientBrush>
    </Window.Resources>
    <Grid Background="{StaticResource WindowBackground}">
        <Frame Source="MainPage.xaml" />
    </Grid>
</Window>
```

Листинг 4.4. Код XAML-документа **ApartmsPage.xaml**

```
<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...>
        <Grid.ColumnDefinitions...>
        <!-- Панель с главным меню приложения -->
        <StackPanel Grid.Row="0" Grid.Column="0">
            <Menu Template="{StaticResource MenuTemplate}" />
        </StackPanel>
        <!-- Панель с заголовком страницы -->
        <StackPanel Grid.Row="1" Grid.Column="0">
            <Label Style="{StaticResource LabelStyle2}">Обслуживаемые квартиры</Label>
        </StackPanel>
        <!-- Панель с таблицей -->
        <StackPanel Grid.Row="2" Grid.Column="0">
            <DataGrid Name="ApartmsDataGrid" Style="{StaticResource DataGridStyle}">
                <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 Template="{StaticResource StatusBarTemplate}" />
        </StackPanel>
    </Grid>
</Page>
```

Листинг 4.5. Код XAML-документа **DiagramWindow.xaml**

```
<Window x:Class="WpfAppCommData.DiagramWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Окно отчетов" Height="450" Width="500" Loaded="Window_Loaded">
    <Window.Resources>
        <!-- Определение кисти для окна отчетов -->
        <RadialGradientBrush x:Key="BackgrDiagramWindow"
                             RadiusX="1" RadiusY="1"
                             GradientOrigin="0.3, 0.3">
            <GradientStop Offset="0.0" Color="#77DDFFDD" />
            <GradientStop Color="DeepSkyBlue" Offset="0.4" />
            <GradientStop Color="RoyalBlue" Offset="0.8" />
        </RadialGradientBrush>
        <!-- Определение стиля кнопок -->
        <Style TargetType="{x:Type Button}">
            <Setter Property="Margin" Value="10" />
            <Setter Property="Width" Value="80" />
            <Setter Property="Button.Background">
                <Setter.Value>
                    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
                        <GradientStop Offset="0.0" Color="#FFFFDD" />
                        <GradientStop Offset="1.0" Color="#DD8822" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Style>
    </Window.Resources>
    <Grid Background="{StaticResource BackgrDiagramWindow}">
        <Grid.RowDefinitions>
            <RowDefinition Height="20" />
            <RowDefinition Height="50" />
            <RowDefinition Height="*" />
            <RowDefinition Height="20" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Row="0">
            <Menu Template="{StaticResource MenuTemplate}" />
        </StackPanel>
        <StackPanel Grid.Row="1" Grid.Column="0">
            <Label Style="{StaticResource LabelStyle}">Построение диаграмм и отчетов</Label>
        </StackPanel>
        <StackPanel Grid.Row="2" Grid.Column="0">
            <Canvas Name="canv" Height="250" Background="#77FFFFFF" Margin="10" />
            <StackPanel Grid.Row="3" Grid.Column="0" Orientation="Horizontal"
                        HorizontalAlignment="Center">
                <Button Name="buttonDraw" Click="buttonDraw_Click">Построить</Button>
                <Button Name="buttonClear" Click="buttonClear_Click">Очистить</Button>
            </StackPanel>
        </StackPanel>
        <StackPanel Grid.Row="3" Grid.Column="0">
            <StatusBar Template="{StaticResource StatusBarTemplate}" />
        </StackPanel>
    </Grid>
</Window>
```

Листинг 4.6. Обработчик события «нажать пункт Отчет главного меню» (модуль **App.xaml.cs**)

```
13      public partial class App : Application
14      {
15          // --- Открыть окно отчетов при нажатии пункта Отчет в главном меню ---
16          private void MenuItem_Click(object sender, RoutedEventArgs e)
17          {
18              DiagramWindow dw = new DiagramWindow();
19              dw.Show();
20          }
21      }
22  }
```

Отображение отдельных страниц в главном окне приложения показано на рис. 4.6 и 4.7. Общий вид окна отчетов приведен на рис 4.8. ❒

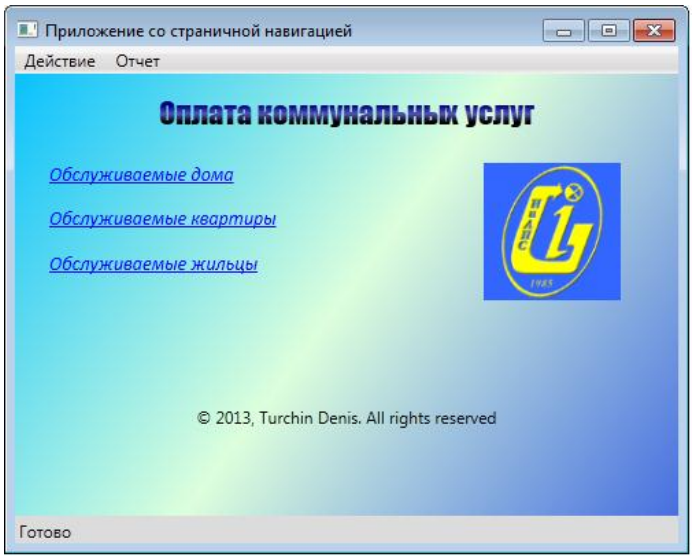

Рис. 4.6. Отображение главной страницы в окне приложения

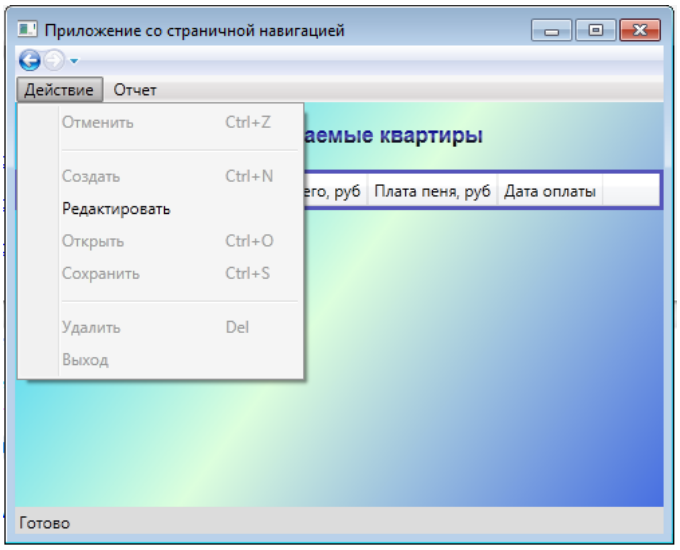

Рис. 4.7. Отображение страницы «Квартиры» в окне приложения

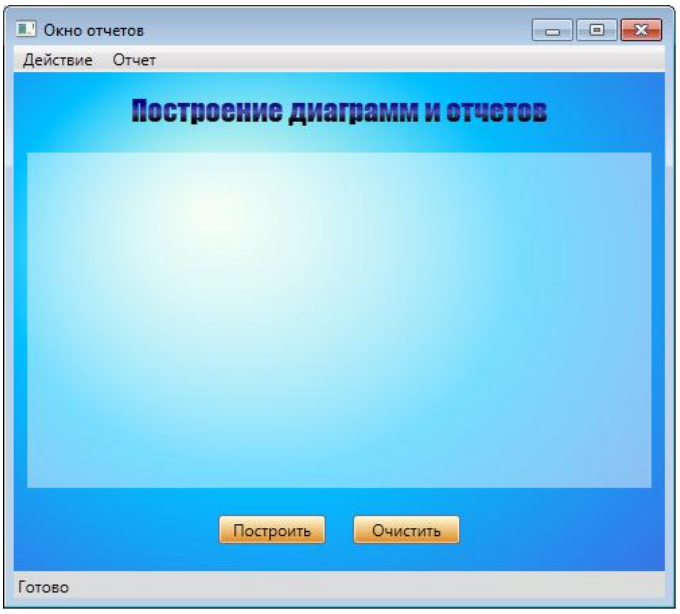

Рис. 4.8. Окно отчетов

4.3. Порядок выполнения работы

Данная лабораторная работа предполагает выполнение следующих этапов:

5. Изучить методические указания к лабораторной работе.

6. Для проекта приложения WPF со страничной навигацией (лабораторная работа №3) определить стили и шаблоны элементов управления в документе **App.xaml**. Задать фоновую кисть для главного окна и загружаемых в него страниц.

7. Привязать стили и шаблоны к соответствующим элементам управления по заданному значению свойства **x:Key**.

8. Добавить в приложение дополнительное окно отчетов, которое должно открываться через пункт «Отчет» главного меню. В это окно необходимо поместить элемент **Canvas** и пару кнопок. Определить для окна отчетов фоновую кисть и стили элементов управления.

9. Оформить и защитить отчет по лабораторной работе.

4.4. Контрольные вопросы

1. Что называют ресурсом в приложении WPF, и какие выделяют виды ресурсов?
2. В чем заключается различие между статическими и динамическими ресурсами?
3. Каким образом может быть задан цвет элемента в приложении WPF?
4. Что понимают под кистью, и какие выделяют виды кистей в WPF?
5. Какие параметры используются для задания линейной градиентной кисти?
6. С помощью каких параметров задают радиальную градиентную кисть?
7. Что понимают под стилями и шаблонами элементов управления в WPF?
8. Как осуществляется объявление стиля в приложении WPF?
9. Каким образом задают шаблон элемента управления в приложении WPF?

ПОДГОТОВКА К ЛАБОРАТОРНЫМ РАБОТАМ И ОФОРМЛЕНИЕ ОТЧЁТОВ ПО НИМ

1. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №1

1.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

- основные составные части XML-документа;
- синтаксис описания элементов, атрибутов и пространств имён языка XML;
- основные средства для работы с XML-документами на платформе .NET Framework;

уметь:

- разрабатывать собственные XML-документы;
- работать в среде MS Visual Studio;
- разрабатывать приложения Windows Forms на языке C#.

1.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код таблицы стилей XSLT для преобразования исходного XML-документа в XML-документ с заданной структурой.
4. Исходный код таблицы стилей XSLT для преобразования XML-документа в формат HTML.
5. Исходный код на языке C# приложения Windows Forms, обеспечивающего преобразование указанного документа XML с помощью выбранной таблицы стилей XSLT.
6. Выводы по лабораторной работе.

2. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №2

2.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

• основные принципы объектно-ориентированного программирования;

• синтаксис описания классов и их элементов на языке C#;

уметь:

• разрабатывать семейства классов, связанных отношением наследования;

• работать в среде MS Visual Studio;

• разрабатывать консольные приложения и приложения Windows Forms на языке C#.

2.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код базового класса с заданными виртуальными методами, а также код производного класса, в котором эти методы переопределены.
4. Исходный код иерархии классов, начиная с абстрактного класса.
5. Исходный код консольного приложения, демонстрирующего полиморфизм построенного семейства классов.
6. Выводы по лабораторной работе.

3. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №3

3.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

- особенности отношения реализации в UML;
- синтаксис описания классов и их элементов на языке C#;

уметь:

- разрабатывать семейства классов, связанных отношением наследования;
- работать в среде MS Visual Studio;
- разрабатывать консольные приложения и приложения Windows Forms на языке C#.

3.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код заданных интерфейсов и классов. Исходный код консольного приложения для работы с полученными классами.
4. Исходный код заданных классов, реализующие стандартные интерфейсы **IComparable** и **IEnumerable**. Исходный консольного приложения для работы с экземплярами созданных классов.
5. Выводы по лабораторной работе.

4. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №4

4.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен
знать:
- диаграммы классов языка UML;
- синтаксис описания классов и их элементов на языке C#;

уметь:
- строить диаграммы классов в среде MS Visual Studio;
- разрабатывать консольные приложения и приложения Windows Forms на языке C#.

4.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Диаграмма классов, построенная с помощью проекта моделирования в Visual Studio.
4. Исходный код классов, реализующих шаблоны GRASP. Исходный консольного приложения для работы с экземплярами созданных классов.
5. Интерфейс пользователя, а также исходный код приложения Windows Forms.
6. Выводы по лабораторной работе.

5. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №5

5.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

• особенности структурных шаблонов проектирования из каталога GoF;

• диаграммы классов языка UML;

• синтаксис описания классов и их элементов на языке C#;

уметь:

• разрабатывать классы и интерфейсы на языке C#;

• работать в среде MS Visual Studio;

• разрабатывать консольные приложения на языке C#.

5.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код классов, используемых для реализации шаблона Адаптер.
4. Исходный код классов, реализующих шаблон Фасад.
5. Выводы по лабораторной работе.

6. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №6

6.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

- особенности поведенческих шаблонов проектирования из каталога GoF;
- диаграммы классов языка UML;
- синтаксис описания классов и интерфейсов на языке C#;

уметь:

- разрабатывать классы и интерфейсы на языке C#;
- работать в среде MS Visual Studio;
- разрабатывать приложения Windows Forms на языке C#.

6.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Диаграмма классов UML.
4. Исходный код классов и интерфейсов, используемых для реализации шаблона Состояние.
5. Выводы по лабораторной работе.

7. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №7

7.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен
знать:
- основные операции языка SQL;
- синтаксис описания классов и их элементов на языке C#;

уметь:
- разрабатывать классы на языке C#;
- работать с коллекциями объектов на языке C#;
- работать в среде MS Visual Studio;
- разрабатывать консольные приложения и приложения Windows Forms на языке C#.

7.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код классов, на основе которых создаются коллекции объектов.
4. Исходный код консольного приложения с определённой коллекцией объектов и запросами LINQ к этой коллекции.
5. Выводы по лабораторной работе.

8. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №8

8.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен

знать:

- основные особенности языка XML;
- основные операции LINQ;

уметь:

- использовать основные операции языка LINQ для выполнения запросов;
- работать в среде MS Visual Studio;
- разрабатывать консольные приложения и приложения на языке C#.

8.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код XML-документа, к которому выполняются запросы.
4. Исходный код консольного приложения с запросами LINQ к XML-документу.
5. Выводы по лабораторной работе.

9. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №9

9.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен
знать:

- основные особенности языка XML;
- основные операции LINQ;
- синтаксис описания классов и их элементов на языке C#;

уметь:

- разрабатывать запросы к коллекциям объектов, использу LINQ;
- работать в среде MS Visual Studio.

9.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код на языке XAML главного окна приложения WPF.
4. Исходный код LINQ-запросов на языке C#
5. Выводы по лабораторной работе.

10. ПОДГОТОВКА К ЛАБОРАТОРНОЙ РАБОТЕ №10

10.1. Требования к знаниям и умениям

Для выполнения лабораторной работы студент должен
знать:
- основные особенности языка XAML;
- основные операции LINQ;;
- синтаксис описания классов и их элементов на языке C#;

уметь:
- работать с элементами управления приложений WPF;
- работать в среде MS Visual Studio;
- разрабатывать консольные приложения WPF на языке C#.

10.2. Требования к отчёту

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код глав.
4. Исходный код на языке XAML главного окна приложения WPF.
5. Исходный код приложения WPF на языке C#
6. Выводы по лабораторной работе.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

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. 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.

4. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч [и др.]. – 3-е изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 720 с.

5. Ватсон, Б. C# 4.0 на примерах. – СПб.: БХВ-Петербург, 2011. – 608 с.

6. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р., Джонсон, Дж. Влиссидес. – СПб.: Питер, 2013. – 368 с.

7. C#: пер. с англ. / Х. Дейтел, П. Дейтел, Дж. Листфорд, Т. Нието, Ш. Йегер, М. Златкина. – СПб.: БХВ-Петербург, 2006. – 1056 с.

8. Ларман, К. Применение UML и шаблонов проектирования. Практическое руководство: пер с англ. – 3-е изд. – М.: ООО «И.Д. Вильямс», 2013. – 736 с.

9. 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.

10. Макки, А. Введение в .NET и Visual Studio 2010 для профессионалов: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 416 с.

11. Мартин, Р. Принципы, паттерны и методики гибкой разработки на языке C# / Р. Мартин, М. Мартин. – пер. с англ. – СПб.: Символ-Плюс, 2011. – 768 с.

12. 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.

13. Нэш, Т. C# 2010: ускоренный курс для профессионалов: пер. с англ. – М.: ООО «И.Д. Вильямс», 2010 – 592 с.

14. Павловская, Т. А. C#. Программирование на языке высокого уровня: учеб. для вузов. – СПб.: Питер, 2007. – 432 с.

15. Петцольд, Ч. Microsoft Windows Presentation Foundation: пер. с англ. – СПб.: Питер, 2008. – 944 с.

16. Подбельский, В. В. Язык C# Базовый курс: учеб. пособие / В. В. Подбельский. – М.: Финансы и статистика; ИНФРА-М, 2011. – 384 с.

17. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 C#. – 4-е изд. – СПб.: Питер, 2013. – 896 с.

18. Стилмен, Э. Изучаем C#. / Э. Стилмен, Дж. Грин. – 2-е изд. – СПб.: Питер, 2012. – 696 с.

19. 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.

20. Фленов, М. Е. Библия C#. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 560 с.

21. Фримен Э. Паттерны проектирования / Э. Фримен, К. Сьера, Б. Бейтс. – СПб.: Питер, 2011. – 656 с.

22. Шилдт, Г. C# 4.0: полное руководство: пер. с англ. – М.: ООО «И. Д. Вильямс», 2011 – 1056 с.

Интернет-ресурсы.

23. http://www.intuit.ru/studies/courses/1176/186/info – учебный курс «Языки информационного обмена», автор Кищенко О.

24. http://www.realcoding.net/teach/xml/ – иллюстрированный самоучитель по XML.

25. http://professorweb.ru/index.php – сайт по .NET и Web-программированию.

26. http://citforum.ru/SE/project/pattern/ – Обзор паттернов проектирования, Ольга Дубина.

27. http://msdn.microsoft.com/ru-ru/library/ms123401.aspx – библиотека Microsoft Developers Network (MSDN) на русском языке.

28. http://metanit.com/index.php – сайт о программировании и IT-технологиях.

29. http://www.w3.org/standards/xml/ – описание стандартов W3C для технологий XML (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.

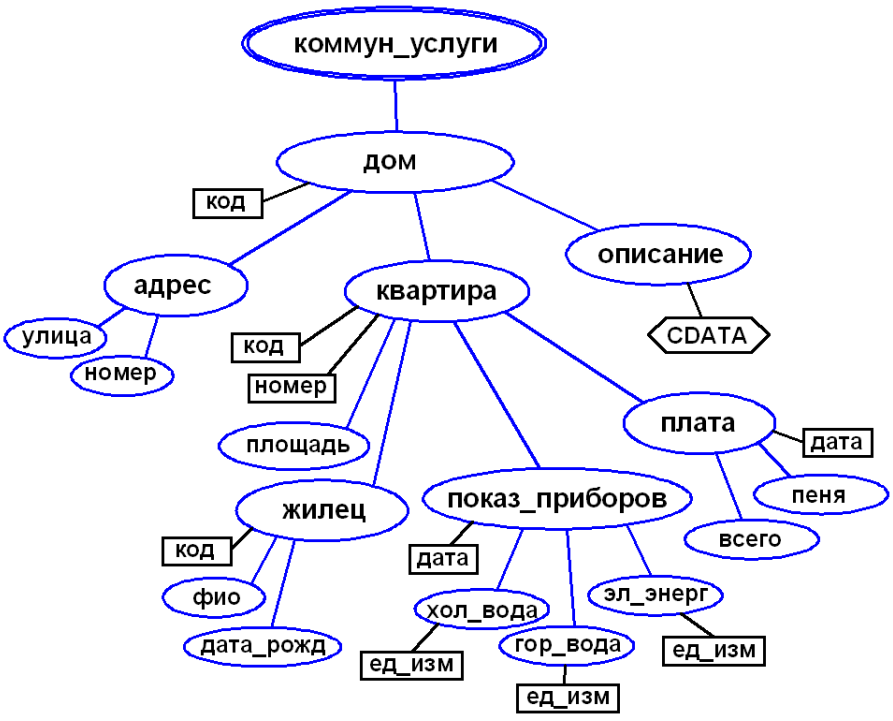

Рис. П.1. Дерево XML-документа

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. Вопросы и задачи к экзамену

Теоретические вопросы.

1. Понятие информационной системы (ИС). Основные составляющие и контекст ИС
2. Понятие архитектуры ИС (АИС). Предметная область АИС
3. Классификация информационных систем
4. Информационные системы и уровни управления организацией
5. Понятие архитектуры приложений. Основные слои корпоративных приложений

6. Локальная и распределённая архитектура приложений. Архитектурные стили приложений

7. Понятие архитектуры данных. Виды корпоративных данных

8. Архитектура хранилищ данных. Оперативный анализ данных

9. Понятие и особенности шаблонов проектирования

10. Составные части и классификация шаблонов проектирования

11. Гибкое проектирование приложений и его принципы (SOLID)

12. Обязанности объектов и подход RDD. Назначение и виды шаблонов GRASP

13. Шаблоны Информационный эксперт и Создатель экземпляров класса.

14. Шаблоны Слабое связывание и Сильное сцепление

15. Структурные шаблоны проектирования. Шаблон Адаптер

16. Особенности шаблона Фасад

17. Шаблон Декоратор

18. Поведенческие шаблоны проектирования. Шаблон Стратегия

19. Особенности шаблона Состояние

20. Особенности шаблона Шаблонный метод

21. Порождающие шаблоны проектирования. Фабричный метод

22. Особенности шаблона Абстрактная фабрика

23. Особенности шаблона Одиночка

24. Понятие и виды антипаттернов. Общие антипаттерны программирования

25. Антипаттерны объектно-ориентированного программирования

26. Антипаттерны проектирования. Методологические антипаттерны

27. Особенности архитектуры клиент-сервер

28. Особенности многоуровневой архитектуры

29. Понятие программных компонент и сервисов

30. Особенности сервис-ориентированной архитектуры (SOA)
31. Сервисная шина предприятия (ESB)
32. Основные особенности Web-сервисов
33. Протоколы взаимодействия Web-сервисов (SOAP и REST)
34. Спецификации WSDL, UDDI и WS-*
35. Язык описания бизнес-процессов BPEL
36. Понятие и характеристики облачных вычислений
37. Модели развёртывания и обслуживания в облачных вычислениях
38. Программное обеспечение как сервис (SaaS). Архитектурные уровни SaaS
39. Шаблоны проектирования облачных приложений
40. Понятие архитектуры предприятия и её контекст
41. Основные домены и уровни абстракции архитектуры предприятия
42. Архитектура и стратегия информационных технологий
43. Процесс разработки архитектуры предприятия. Архитектурные фреймворки
44. Модель Захмана
45. Методология TOGAF
46. Архитектура федеральной организации (FEA)

Практические задачи.

1. Разработать таблицу стилей XSLT
2. Разработать XML-схему на языке XSD
3. Разработать приложение на языке C#, в котором присутствует заданное событие
4. Разработать приложение на языке C#, в котором реализован требуемый интерфейс
5. Разработать приложение на языке C#, использующее заданный шаблон GoF
6. Разработать запросы LINQ к коллекции объектов
7. Разработать запросы LINQ к XML-документу