МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тульский государственный университет»

Институт прикладной математики и компьютерных наук

Кафедра вычислительной техники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ
по дисциплине

ВИЗУАЛЬНОЕ МОДЕЛИРОВАНИЕ И ДОКУМЕНТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Направление подготовки: 09.03.01«Информатика и вычислительная техника»
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

Форма обучения: очная

Тула 2017

СОДЕРЖАНИЕ

Методические указания к лабораторным работам составлены доцентом кафедры ВТ Г.Б. Берсеневым и обсуждены на заседании кафедры ВТ института прикладной математики и компьютерных наук
протокол №_1_ от "_30_"_августа__ 2017 г.
Зав. кафедрой_____А.Н. Ивутин

Методические указания к лабораторным работам пересмотрены и утверждены на заседании кафедры ВТ
института прикладной математики и компьютерных наук
протокол ___ от "_"_____ 201_ г.
Зав. кафедрой_____А.Н. Ивутин

Лабораторная работа № 1

Создание диаграмм последовательностей в интегрированной среде MS Visual Studio Ultimate 2010/12/13

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Ознакомление с унифицированным языком визуального моделирования UML и его применением при создании диаграмм последовательностей в среде MS Visual Studio Ultimate 2010/12/13.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Диаграммы последовательностей

Сценарии для варианта использования обычно представляются в форме *диаграмм взаимодействия* (interaction diagrams). Различают два типа таких диаграмм - *диаграммы последовательностей* (sequence diagrams) и *диаграммы сотрудничества* (collaboration diagrams).

Диаграмма последовательностей иллюстрирует очередность выполнения операций взаимодействия объектов во времени и отображает объекты и классы, вовлеченные в сценарий, наряду с цепочками сообщений, которыми объекты обмениваются в ходе осуществления функций, предусмотренных сценарием. Диаграммы последовательностей обычно ассоциируются с реализациями вариантов использования.

В соответствии с требованиями UML объект на диаграмме последовательностей изображается в виде прямоугольника, содержащего подчеркнутое наименование объекта. Как показано на рис. 1, объект можно именовать тремя способами: указать только его название (имя объекта), задать имена объекта и класса либо ограничиться наименованием класса (для анонимного объекта).

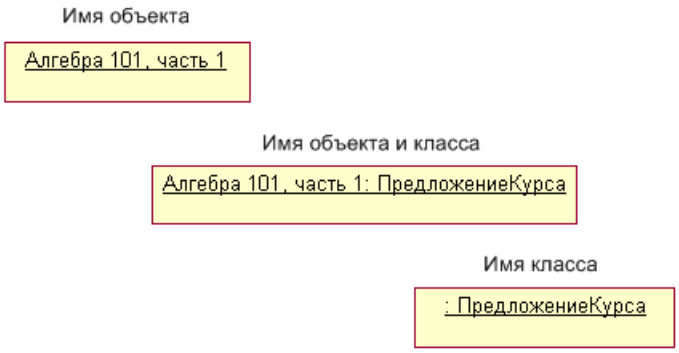

Рис. 1. Представление объектов на диаграммах

Названия объектов могут быть частными (например, "алгебра 101, часть 1") или общими (как "предложение курса"). Зачастую анонимный объект используется для представления произвольного объекта класса.

Каждому объекту на диаграмме последовательностей ставится в соответствие временная отметка, обозначаемая отрезком штриховой линии, а сообщения, которыми обмениваются два объекта, представляются стрелкой, соединяющей объект-источник с объектом-приемником.

На рис. 2 приведен фрагмент диаграммы последовательностей, содержащий описания объектов и соответствующих сообщений.

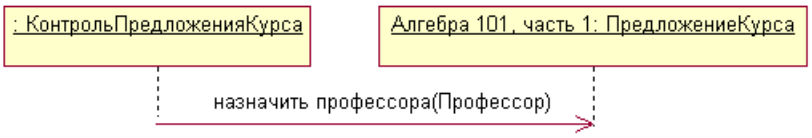

Рис. 2 . Обозначения объектов и сообщений, применяемые в диаграммах последовательностей

Диаграмма последовательностей для сценария "создание курса" варианта использования "сохранение информации о курсах", полученная в инструментальной среде Rational Rose, изображена на рис. 3.

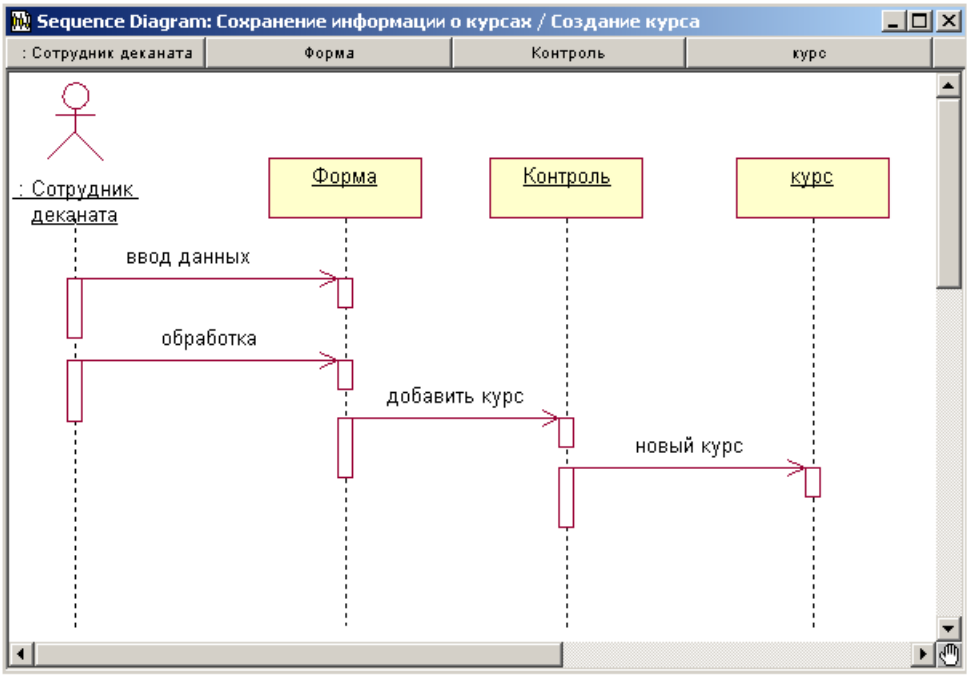

Рис. 3. Диаграмма последовательностей для сценария "создание курса"

2.2. Диаграммы сотрудничества

Диаграмма сотрудничества (collaboration diagram), представляющая альтернативный способ описания сценария, воспроизводит процесс взаимодействия объектов и содержит объекты-прямоугольники, связи между объектами и текстовые сообщения со стрелками, указывающими направление от объекта-источника к объекту-приемнику.

На рис. 4 приведен фрагмент диаграммы сотрудничества, содержащий описания объектов, связей и соответствующих сообщений.

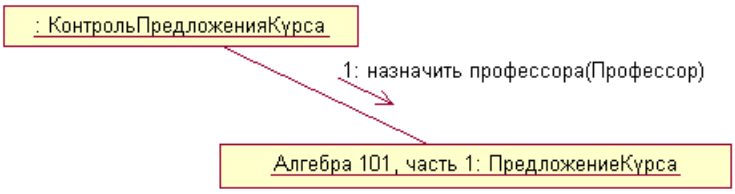

Рис. 4. Обозначения объектов, связей и сообщений, применяемые в диаграммах сотрудничества

Диаграмма сотрудничества для сценария "создание курса" варианта использования "сохранение информации о курсах", полученная в инструментальной среде Rational Rose, изображена на рис. 5. Она полностью соответстает диаграмме, приведенной на рис. 3.

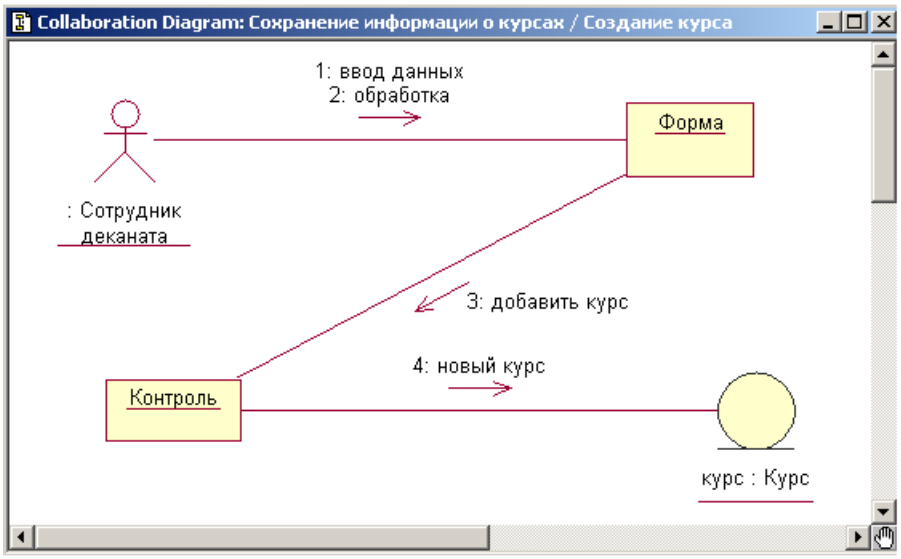

Рис. 5. Диаграмма сотрудничества

Диаграммы последовательностей позволяют взглянуть на сценарий с точки зрения упорядочения операций во времени: одни операции осуществляются вначале, что-то происходит позже и т.д. Подобные диаграммы весьма легко воспринимаются будущими пользователями системы и поэтому полезны на ранних стадиях анализа. Диаграммы сотрудничества, в свою очередь, предлагают более полную картину динамики развития сценария, отображая объекты в связи с другими объектами. Такие диаграммы, вероятно, более предпочтительны на этапе проектирования, когда исследуются варианты реализации связей.

2.3. Создание диаграмм последовательностей из кода на языке C#

Ниже приведен код программы на языке C#, в которой объявлен класс A, имеющий приватное поле temperature и свойство Temperature, обеспечивающее управляемый доступ к этому полю.

```
namespace Prog1
{
    public class A
    {
        private int temperature;
        public int Temperature
        {
            get
            {
                System.Console.WriteLine("Извлечение значения temperature");
                return temperature;
            }
            set
            {
                System.Console.WriteLine("Установка значения temperature");
                temperature = value;
            }
        }
    }
    public class MainClass
    {
        static void Main()
        {
            A obj = new A();
            obj.Temperature = 1;
            System.Console.WriteLine("obj.Temperature = {0}",
            obj.Temperature);
            string str = Console.ReadLine();
```

```
        }
    }
}
```

Диаграмма последовательностей для метода Main(), полученная непосредственно из кода средствами инструментальной системы MS Visual Studio 2010, показана на рис. 6.

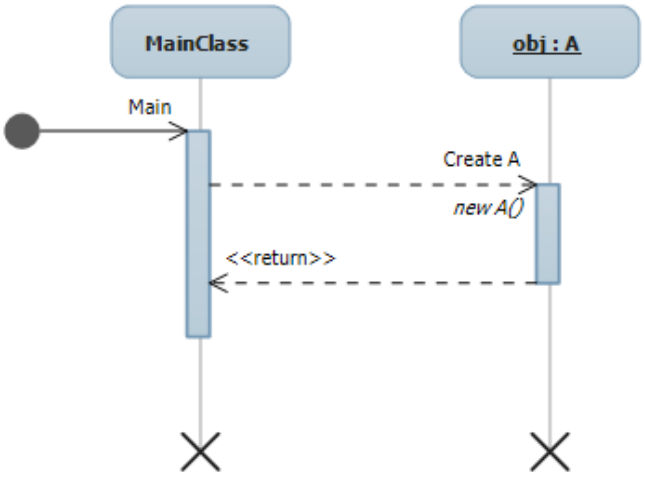

Рис. 6. Простейшая диаграмма последовательностей

Введем цикл do в метод Main() следующим образом:

```
static void Main()
{
    int i = 1;
    do
    {
        A obj = new A();
        obj.Temperature = i++;
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    } while (i < 4);
    Console.ReadKey();
}
```

Диаграмма последовательности для метода Main() также изменится – в нем появится объединенный фрагмент Loop (рис. 7).

Ниже приведен метод Main() с циклами while, for и foreach, а на рис. 8, 9 и 10 - соответствующие диаграммы последовательностей, расширенные возможностями отслеживания изменения свойств объектов:

```
static void Main()
{
    int i = 1;
    while (i < 4)
    {
        A obj = new A();
        obj.Temperature = i++;
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    } ;
    Console.ReadKey();
}

static void Main()
{
    int i;
```

```
    for (i = 1; i < 5; i++ )
    {
        A obj = new A();
        obj.Temperature = i;
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    }
    Console.ReadKey();
}

static void Main()
{
    int [] numbers = {1, 3, 5, 9};
    foreach (int i in numbers)
    {
        A obj = new A();
        obj.Temperature = i;
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    }
    Console.ReadKey();
}
```

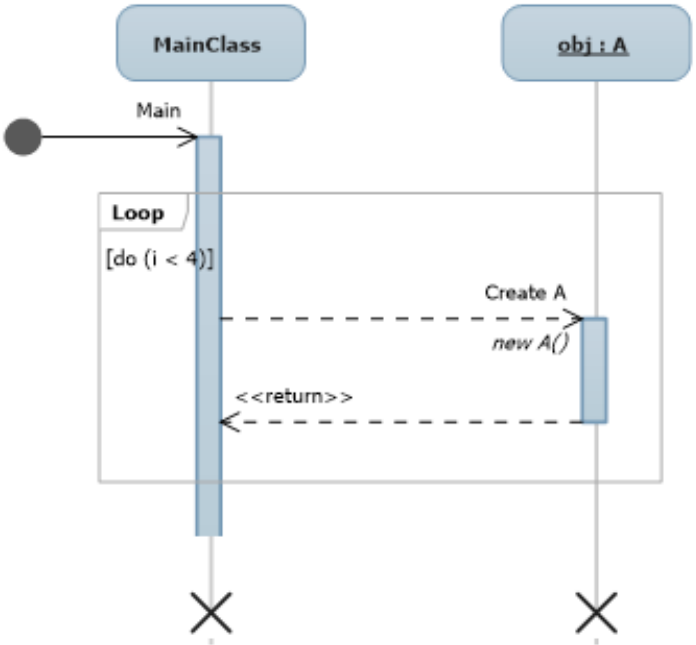

Рис. 7. Диаграмма последовательностей с циклом do

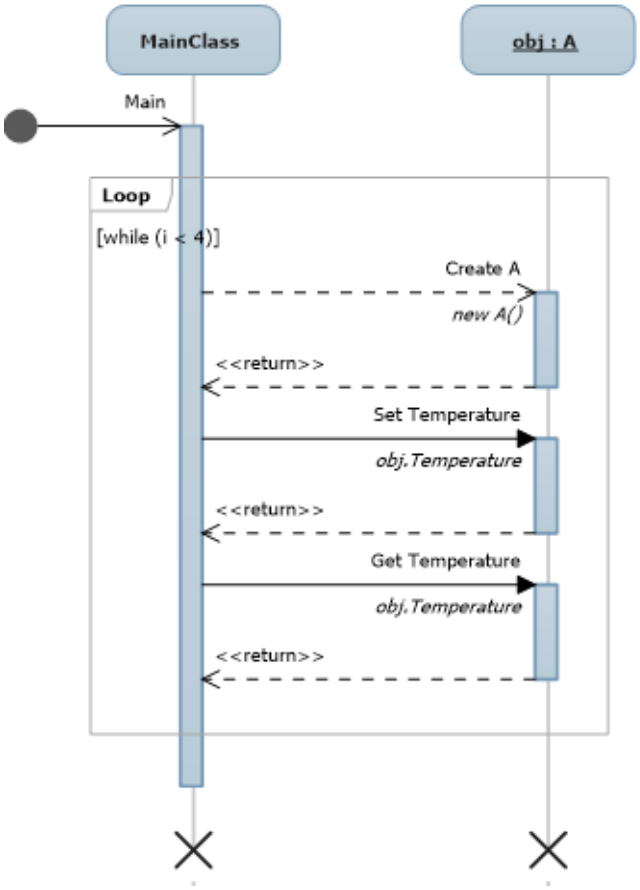

Рис. 8. Диаграмма последовательностей с циклом while

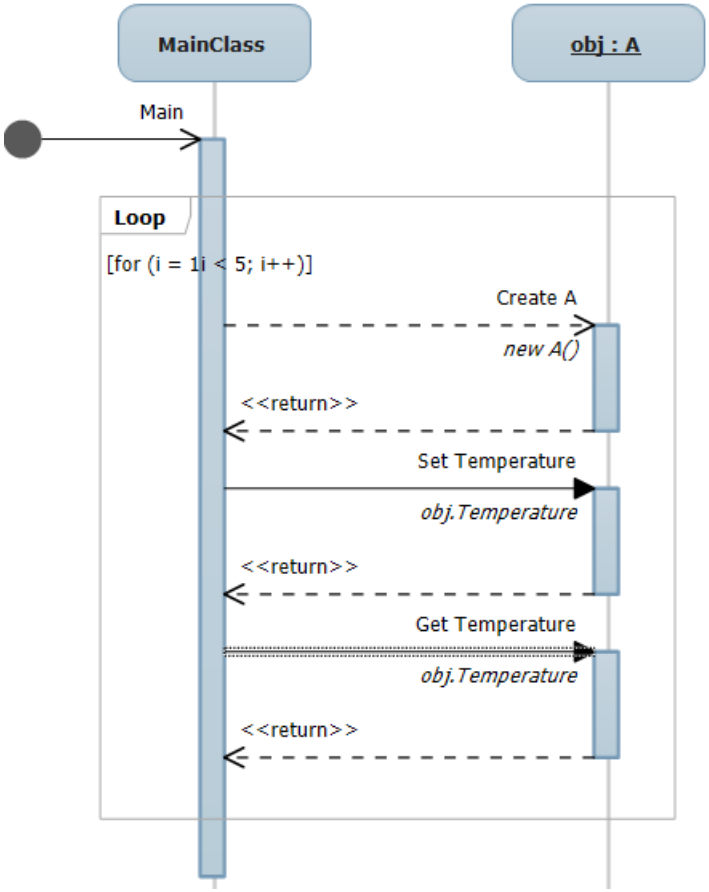

Рис. 9. Диаграмма последовательностей с циклом for

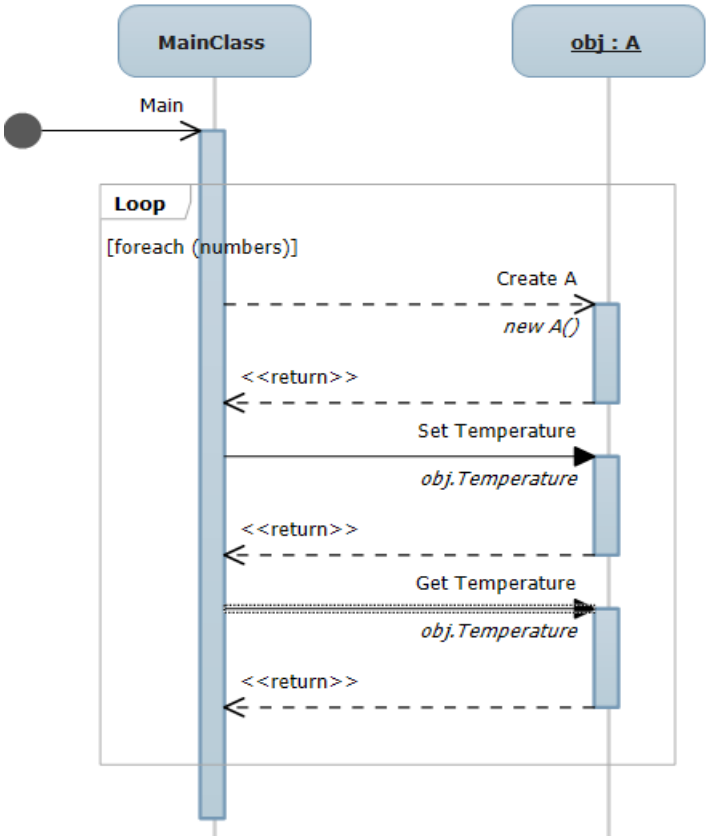

Рис. 10. Диаграмма последовательностей с циклом foreach

На UML-схеме последовательностей в Visual Studio Ultimate с помощью объединенных фрагментов можно показывать циклы, ветви и другие альтернативные элементы. Объединенный фрагмент 1 состоит из одного или нескольких операндов взаимодействия 2 с именами 4 (3 – выделенный фрагмент), каждый из которых включает одно или несколько сообщений, использований взаимодействия или объединенных фрагментов (рис. 11).

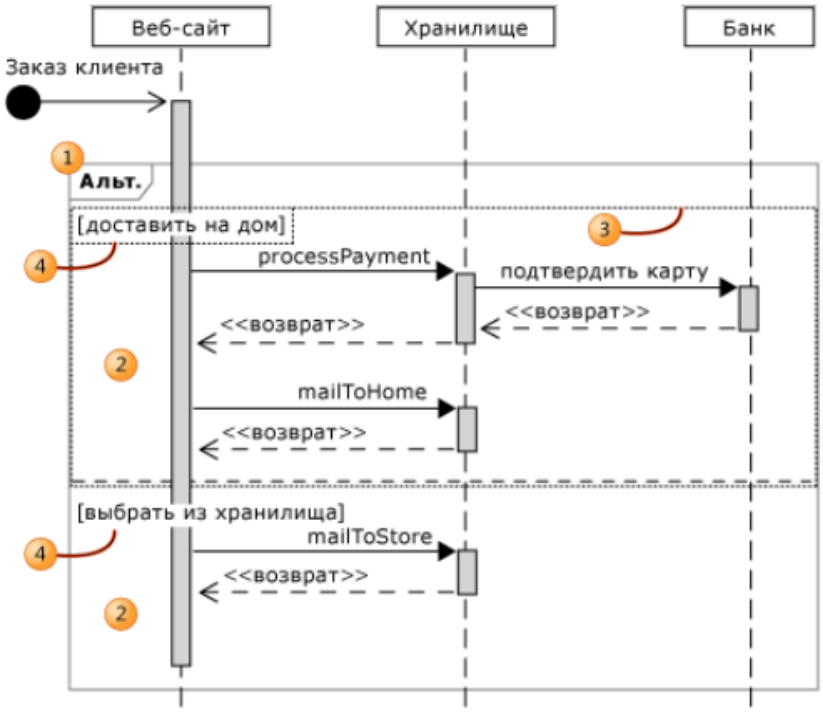

Рис. 11. Диаграмма последовательностей с альтернативными ветвями

Введем в цикле метода Main предыдущей программы оператор if так, чтобы в зависимости от индекса цикла устанавливались различные значения свойства Temperature (диаграмма последовательностей показана на рис. 12):

```
static void Main()
{
    int i = 1;
    do
    {
        A obj = new A();
        if (i < 5)
            obj.Temperature = i++;
        else
            obj.Temperature = 10 + i++;
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    } while (i < 10);
    Console.ReadKey();
}
```

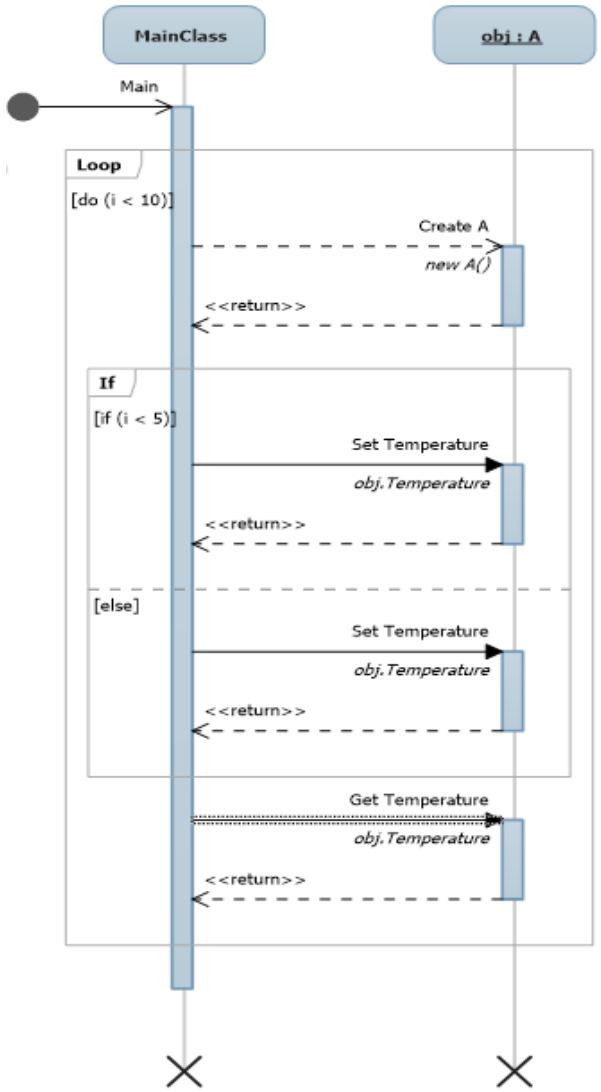

Рис. 12. Диаграмма последовательности при наличии цикла и ветвлениий

Ниже представлена программа, в которой вместо операторов ветвления if используется оператор выбора switch, а на рис. 13 – соответствующая диаграмма последовательностей:

```
static void Main()
```

```
{
    int i = 1, j = 1;
    do
    {
        A obj = new A();
        switch (i)
        {
            case 4:
                j++;
                obj.Temperature = j * i++;
                break;
            case 8:
                j++;
                obj.Temperature = j * i++;
                break;
            default:
                obj.Temperature = j * i++;
                break;
        }
        System.Console.WriteLine("obj.Temperature = {0}",
        obj.Temperature);
    } while (i < 10);
    Console.ReadKey();
}
```

Далее рассмотрим визуальное моделирование более сложной объектно-ориентированной программы, содержащей вложенные классы.

Вложенные классы определяются внутри области определения другого класса. Вложенные классы обладают некоторыми специальными возможностями, которые удобны, когда нужен вспомогательный класс, работающий внутри содержащего его класса.

Например, контейнерный класс может содержать коллекцию объектов. Предположим, что требуется некоторое средство для выполнения итерации по всем содержащимся объектам, чтобы позволить внешним пользователям, выполняющим итерацию, поддерживать маркер, или некую разновидность курсора, который запоминает свое текущее место во время итерации. Это распространенный подход в проектировании. Избавление пользователей от необходимости хранить прямые ссылки на содержащиеся в коллекции объекты обеспечивает большую гибкость в отношении изменения внутреннего поведения контейнерного класса без разрушения кода, использующего этот контейнерный класс. Вложенные классы по нескольким причинам предоставляют отличное решение такой проблемы.

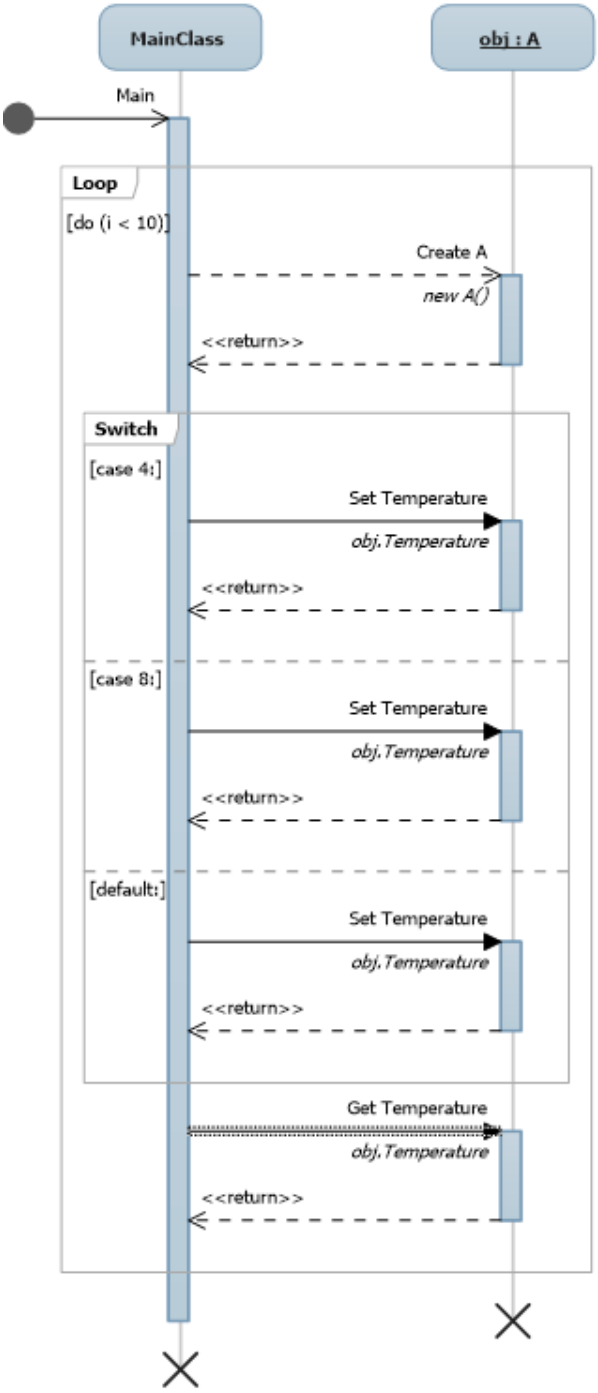

Рис. 13. Диаграмма последовательности при наличии цикла и выбора

Вложенные классы имеют доступ ко всем членам, видимым содержащему их классу, даже если эти члены являются приватными. Рассмотрим следующий код, который представляет контейнерный класс, включающий экземпляры GeometricShape:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
public abstract class GeometricShape
{
    public abstract void Draw();
}
public class Rectangle : GeometricShape
{
```

```
    public override void Draw()
    {
        System.Console.WriteLine ( "Rectangle.Draw" );
    }
}
public class Circle : GeometricShape
{
    public override void Draw()
    {
        System.Console.WriteLine( "Circle.Draw" );
    }
}
public class Drawing : IEnumerable
{
    private ArrayList shapes;
    private class Iterator : IEnumerator
    {
        public Iterator(Drawing drawing)
        {
            this.drawing = drawing;
            this.current = -1;
        }
        public void Reset()
        {
            current = -1;
        }
        public bool MoveNext()
        {
            ++current;
            if (current < drawing.shapes.Count)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        public object Current
        {
            get
            {
                return drawing.shapes[current];
            }
        }
        private Drawing drawing;
        private int current;
    }
    public Drawing()
    {
        shapes = new ArrayList();
    }
    public IEnumerator GetEnumerator()
    {
        return new Iterator(this);
    }
    public void Add(GeometricShape shape)
    {
        shapes.Add(shape);
    }
}
public class EntryPoint
{
    static void Main()
    {
```

```
        Rectangle rectangle = new Rectangle ();
        Circle circle = new Circle ();
        Drawing drawing = new Drawing();
        drawing.Add( rectangle );
        drawing.Add( circle );
        foreach( GeometricShape shape in drawing ) {
        shape.Draw();
        }
        Console.ReadKey();
    }
}
```

Диаграмма классов данного приложения, содержащего итератор и классы геометрических фигур, приведена на рис. 14.

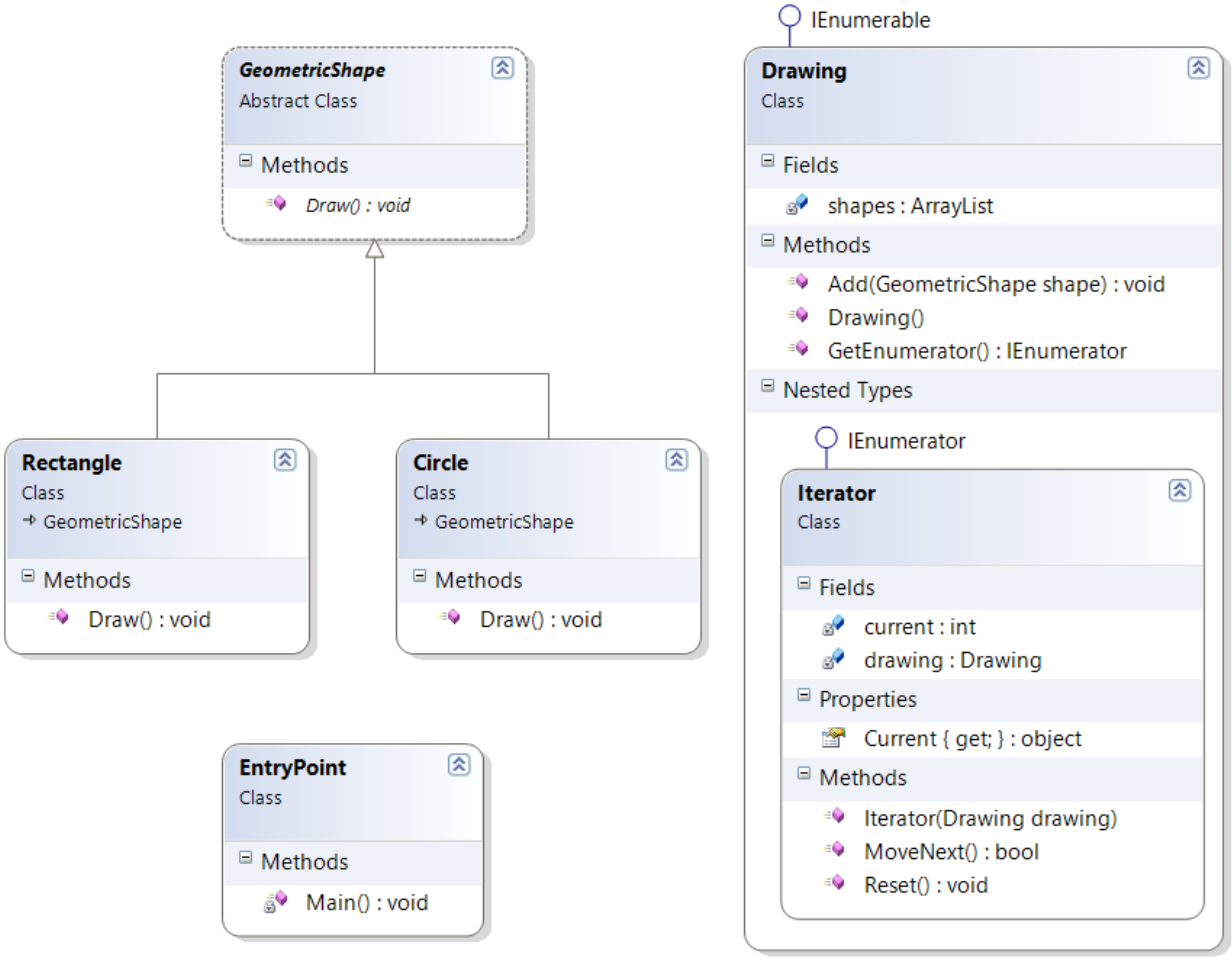

Рис. 14. Диаграмма классов приложения

В этом примере демонстрируется ряд концепций, в том числе интерфейсы IEnumerable и IEnumerator Отметим, что класс Drawing поддерживает метод GetEnumerator, являющийся частью реализации интерфейса IEnumerable. Он создает и возвращает экземпляр вложенного класса Iterator.

Класс Iterator принимает ссылку на экземпляр содержащего его класса Drawing в виде параметра конструктора. Затем он сохраняет этот экземпляр для последующего использования, чтобы можно было добраться до коллекции shapes внутри объекта drawing. Обратите внимание, что коллекция shapes в классе Drawing объявлена как private. Это не имеет значения, потому что вложенные классы имеют доступ к приватным членам охватывающего их класса.

Класс Iterator сам по себе объявлен как private. Невложенные классы могут объявляться только как public или internal и по умолчанию являются internal. К вложенным классам можно применять те же модификаторы доступа, что и к любым

другим членам класса. В данном случае класс Iterator объявлен как private, так что внешний код вроде процедуры Main не может создавать экземпляры Iterator непосредственно. Это может делать только сам класс Drawing. Возможность создания экземпляров Iterator не имеет смысла ни для чего другого, кроме Drawing.GetEnumerator.

Диаграмма последовательностей для метода Main класса EntryPoint представлена на рис. 15.

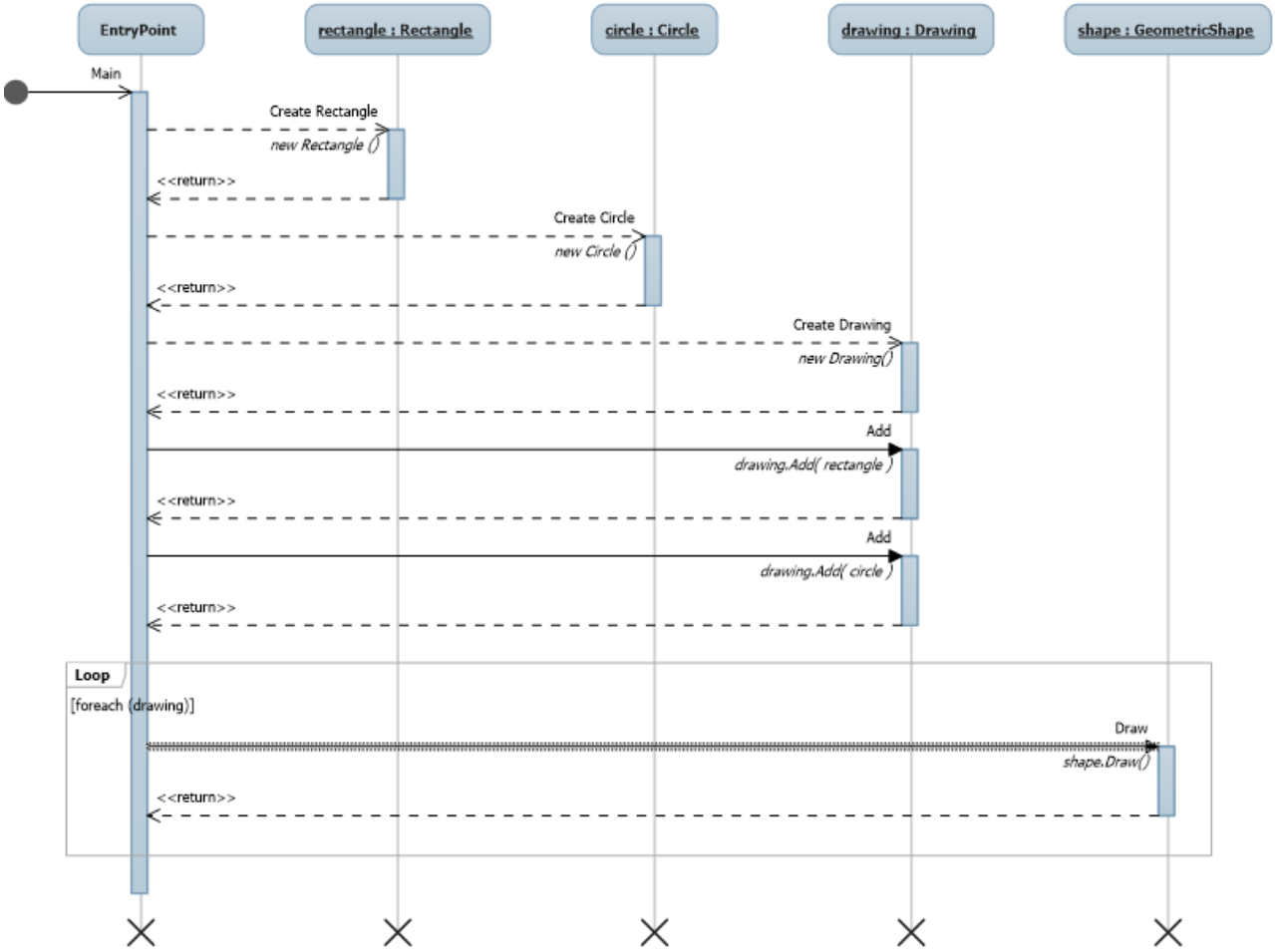

Рис. 15. Диаграмма последовательностей приложения

В приведенных выше примерах диаграмма последовательностей создавалась автоматически средой Visual Studio на основе кода программы на языке C#. Для других языков Visual Studio автоматически такие диаграммы не создает. Необходимо также отметить, что разработка диаграмм последовательностей может предшествовать разработке кода (например, такие диаграммы могут создаваться при разработке технического задания). Для ручного создания диаграмм последовательностей в MS Visual Studio необходимо выбрать последовательность опций: «Архитектура» / «Создать схему» / «Схема последовательностей UML».

3. ОБОРУДОВАНИЕ

Персональный компьютер типа IBM PC Pentium, операционная система MS Windows 7/8, инструментальная система Microsoft Visual Studio Ultimate 2010/12/13, файл МУ_ЛР_ВМиДПО.doc (методические указания к лабораторным работам), 200 Мб свободной памяти на логическом диске, содержащем каталог Vmd\lbr1.

4. ЗАДАНИЕ НА РАБОТУ

4.1. Ознакомиться с интегрированной инструментальной системой Microsoft Visual Studio Ultimate 2010/12/13 для разработки и визуального моделирования программных систем. Создать проект консольного приложения на языке C#.

4.2. С помощью средств автоматического создания диаграмм системы Microsoft Visual Studio Ultimate 2010/12/13 создать в виде файлов проекта диаграммы классов и последовательностей, представленные на рис. 6 - 15.

4.3. Создать одну из диаграмм последовательностей, представленных на рис. 6 – 15, вручную с использованием инстументальной системы MS Visual Studio.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

5.1. Запустить выполнение интегрированной инструментальной системы Microsoft Visual Studio Ultimate 2010/12/13.

5.2. С помощью средств среды Microsoft Visual Studio Ultimate 2010/12/13 создать консольный проект на языке C#. Для этого выберем в меню File следующую последовательность опций: New->Project-> Visual C# -> Console Application. Далее определяем имя проекта в поле Name и местонахождение каталога с файлами проекта в поле Location. Отметим, что имя проекта является одновременно и именем каталога, который будет создан средой разработки.

5.3. Заменить код программы Program.cs в окне редактирования на код программы из методички, а затем нажать зеленый треугольник на панели инструментов, чтобы откомпилировать и выполнить программу. Ознакомиться с полученными результатами. Сохранить файлы проекта.

5.4. Установить курсор на метод Main в окне проекта и в контекстном меню выбрать опцию «Создать диаграмму последовательностей» и поставить галочку для опции «Добавить диаграмму в проект». Имя для диаграммы указывать не требуется – оно формируется автоматически. Ознакомиться с полученной диаграммой, проверить соответствие диаграммы загруженному коду. Сохранить файлы проекта.

5.5. Повторить выполнение п.п. 5.3 и 5.4, получить и добавить в состав проекта в виде файлов проекта диаграммы классов и последовательностей, представленные на рис. 6 - 15.

5.6. Создать одну из диаграмм последовательностей, представленных на рис. 6 – 15, вручную с использованием инстументальной системы MS Visual Studio.

6. ОФОРМЛЕНИЕ ОТЧЕТА

Отчет должен содержать:

- цель и задачи работы;
- листинги программ и автоматически полученные для них диаграммы классов и последовательностей;
- диаграмму последовательностей, полученную вручную;
- краткое описание полученных результатов.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

7.1. Дайте определение класса и объектов.

7.2. Приведите пример диаграммы сотрудничества.

7.3. Сравните диаграммы последовательностей и сотрудничества.

7.4. Что такое свойство объекта? Приведите пример.

7.5. Как получить диаграмму классов с полной сигнатурой и вложенными классами?

7.6. Что используется для моделирования циклов в диаграммах последовательностей?

7.7. Какие управляющие операторы имеются в языке C#?

7.8. Чем отличаются интерфейсы и абстрактные классы от обычных классов?

7.9. Объясните работу приложения, представленного на диаграммах рис. 14 и 15.

7.10. Что такое итератор?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Леоненков А.В. Самоучитель UML 2. — СПб.: БХВ-Петербург, 2007. — 576 с.

2. Орлов С. Технологии разработки программного обеспечения: Учебник. – СПб.: Питер, 2002. – 464 с.

3. Буч Г. Язык UML : Руководство пользователя / Г.Буч, Д.Рамбо, И.Якобсон; пер.с англ.Мухин Н. — 2-е изд. — М. : ДМК Пресс:Академия Айти, 2007 .— 496с.

4. INTUIT.ru . Курс «Нотация и семантика языка UML» . Автор А.В.Леоненков .— Интернет университет информационных технологий, 2009 .

5. INTUIT.ru. Курс «Введение в UML». Автор: А.В. Бабич. – Интернет университет информационных технологий, 2008.

6. INTUIT.ru . Курс «Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose» . Автор А.В.Леоненков .— Интернет университет информационных технологий, 2006 .

7. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов. Учебное пособие / Д. Розенберг, К. Скотт. – М. : ДМК Пресс, 2007. – 160. (ЭБС IPRbooks)

8. Буч Г. Язык UML : Руководство пользователя / Г.Буч,Д.Рамбо,И.Якобсон;пер.с англ.Мухин Н. — 2-е изд. — М. : ДМК Пресс:Академия Айти, 2008 .— 496с. (ЭБС IPRbooks)

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

Лабораторная работа № 2

Создание XML-схем для XML-документов

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Ознакомление с основными встроенными типами языка XSD и способами определения новых простых и сложных типов элементов и атрибутов, с использованием именованных и неименованных типов при объявлении элементов и атрибутов, а также получение практических навыков создания XML-схем для документов XML.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1. Встроенные простые типы XSD

2.1.1. Введение. В мае 2001 года консорциум W3C рекомендовал описывать структуру документов XML на языке описания схем XSD (XML Schema Definition Language). На этом языке записывается схема XML (XML Schema), описывающая конструкции, использованные в документе XML.

Язык XSD создан как реализация XML. Это значит, что схема XML сама записывается в виде документа XML. Ее элементы называют *компонентами* (components), чтобы отличить их от элементов описываемого документа XML. Корневой компонент схемы носит имя schema. Компоненты схемы описывают элементы XML и определяют различные типы элементов. Рекомендация схемы XML, которую можно посмотреть по адресу http:// www.w3.org/XML/Schema/, перечисляет 13 типов компонентов, но наиболее важны компоненты, определяющие простые и сложные типы элементов, сами элементы и их атрибуты.

Язык XSD различает простые и сложные элементы XML. *Простыми* (simple) элементами описываемого документа XML считаются элементы, не содержащие атрибутов и вложенных элементов. Соответственно, *сложные* (complex) элементы содержат атрибуты и/или вложенные элементы. Схема XML определяет *простые типы* — типы простых элементов, и *сложные типы* — типы сложных элементов.

Встроенные типы языка описания схем XSD позволяют записывать двоичные и десятичные целые числа, вещественные числа, дату и время, строки символов, логические значения, адреса URI. Рассмотрим их по порядку.

2.1.2.Вещественные числа. Вещественные числа в языке XSD разделены на три типа: decimal, float и double.

Тип decimal составляют вещественные числа, записанные с фиксированной точкой: 123.45, -0.1234567689345 и т.д. Фактически хранятся два целых числа. Одно число представляет мантиссу, другое — порядок вещественного числа. Спецификация языка XSD не ограничивает количество цифр в мантиссе, но требует, чтобы можно было записать не менее 18 цифр. При обработке документа средствами технологии Java этот тип легко реализуется, например, классом java.math.BigDecimal, входящим в стандарт Java API.

Типы float и double соответствуют стандарту IEEE754-85 и одноименным типам Java. Они записываются с фиксированной или с плавающей десятичной точкой. Например, 34.567, -45.67, 1e-5, 34.58e14.

2.1.3. Целые числа. Основной целый тип integer понимается как подтип типа decimal, содержащий числа с нулевым порядком. Это целые числа с любым количеством десятичных цифр: -34567, 123456789012345 и т.д. При использовании средств Java для обработки документа этот тип легко реализуется классом Java.math.BigInteger.

Типы long, int, short и byte полностью соответствуют одноименным типам Java. Они понимаются как подтипы типа integer, типы более коротких чисел считаются подтипами более длинных чисел, например тип byte — это подтип типа short, оба они подтипы типа int и т. д.

Значения типа byte, как следует из его названия, занимают один байт и изменяются от -128 до 127. Тип short занимает два байта, его значения лежат в диапазоне от -32768 до +32767. Числа типа int хранятся в четырех байтах и меняются от -2147483648 до +2147483647. Наконец, тип long располагается в восьми байтах, его значения от -9223372036854775808 до +9223372036854775807.

Типы `nonPositivelnteger` и `negativelnteger` — подтипы типа integer — составлены из неположительных и отрицательных чисел соответственно с любым количеством цифр.

Типы nonNegativelnteger и positivelnteger— подтипы типа integer— со-ставлены из неотрицательных и положительных чисел соответственно с любым количеством цифр.

У типа nonNegativelnteger есть подтипы целых чисел без знака unsignedLong, unsignedlnt, unsignedShort и unsignedByte.

2.1.4. Строки символов. Основной символьный тип string описывает произвольную строку символов Unicode. Его можно реализовать средствами Java, используя класс Java.lang.String.

Тип normalizedString — подтип типа string — это строки, не содержащие символов перевода строки '\n', возврата каретки '\r' и горизонтальной табуляции '\t'.

В строках типа token — подтипа типа normalizedString — нет, кроме того, начальных и завершающих пробелов и нескольких подряд идущих пробелов.

В типе token выделены три подтипа. Подтип language определен для записи названия языка согласно рекомендации RFC 1766, например, ru, en, de, fr. Подтип NMTOKEN используется только в атрибутах для записи их перечисляемых значений. Подтип name составляют *имена XML* — последовательности букв, цифр, дефисов, точек, двоеточий, знаков подчеркивания, начинающиеся с буквы (кроме зарезервированной последовательности букв X, x, M, m, L, 1 в любом сочетании регистров) или знака подчеркивания. Мы видели в предыдущих главах, что имена, начинающиеся со строки xml, используются самой спецификацией XML, например, имя атрибута xmlns. Двоеточие в значениях типа name применяется для выделения префикса в уточненных именах при использовании пространства имен.

Из типа name выделен подтип NCName (Non-Colonized Name) имен, не содер-жащих двоеточия, в котором, в свою очередь, определены три подтипа: ID, ENTITY, IDREF, — описывающие идентификаторы XML, сущности и перекрестные ссылки на идентификаторы.

2.1.5. Дата и время. Тип duration описывает промежуток времени, например, запись P1Y2M3DT10H30M45S означает один год (1Y), два месяца (2м), три дня (3D), де-сять часов (10H), тридцать минут (30M) и сорок пять секунд (45S). Запись может быть сокращенной, например, P120M означает 120 месяцев, а T120M — 120 минут.

Тип dateTime содержит дату и время в формате CCYY-MM-DDThh:mm:ss, например, 2003-04-25T09:30:05. Остальные типы выделяют какую-либо часть даты или времени.

Тип time содержит время в обычном формате hh:mm:ss.

Тип date содержит дату в формате CCYY-MM-DD.

Тип gYearMonth выделяет год и месяц в формате CCYY-MM.

Тип gMonthDay содержит месяц и день месяца в формате -MM-DD.

Тип gYear означает год в формате CCYY, тип gMonth - месяц в формате -мм-, тип gDay — день месяца в формате -DD.

2.1.6. Двоичные типы. Двоичные целые числа записываются либо в шестнадцатеричной форме без всяких дополнительных символов: 0B2F, 356C0A и т. д., это тип hexBinary, либо в кодировке Base64, это тип base64Binary.

2.1.7.Прочие встроенные простые типы. Еще три встроенных простых типа описывают значения, часто используемые в документах XML.

Адреса URI относятся к типу anyURI.

Расширенное имя тега или атрибута (qualified name), т. е. имя вместе с префиксом, отделенным от имени двоеточием, — это тип QName.

Обозначение NOTATION описания DTD выделено как отдельный простой тип схемы XML. Его используют для записи математических, химических и других символов, нот, азбуки Бройля и прочих обозначений.

2.2. Определение простых типов

2.2.1. Введение. В схемах XML с помощью встроенных типов можно тремя способами определить новые типы простых элементов. Они вводятся как *сужение* (restriction) встроенного или ранее определенного простого типа, *список* (list) или *объединение* (union) простых типов.

Простой тип определяется компонентом схемы simpleType, имеющим вид

<xsd:simpleType name="имя типа">Определение типа</xsd:simpleType>

2.2.2. Сужение. Сужение простого типа определяется компонентом restriction, в котором атрибут base указывает сужаемый простой тип, а в содержимом задаются ограничения, выделяющие определяемый простой тип. Например, почтовый индекс zip можно определить как шесть арабских цифр следующим образом:

```
<xsd:simpleType name="zip">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{6}" />
  </xsd:restriction>
</xsd:simpleType>
```

Можно дать другое определение простого типа zip как целого положительного числа, находящегося в диапазоне от 100000 до 999999:

```
<xsd:simpleType name="zip">
  <xsd:restriction base="xsd:positivelnteger">
      <xsd:minlnclusive value="100000" />
      <xsd:maxlnclusive value="999999" />
   </xsd:restriction>
</xsd:simpleType>
```

Теги <pattern>, <maxInclusive> и др., задающие ограничения, называются *фасетками* (facets). Вот их список:

- <maxExclusive> — наибольшее значение, которое уже не входит в определяемый тип;
- <maxInclusive> — наибольшее значение определяемого типа;
- <minExclusive> — наименьшее значение, уже не входящее в определяемый тип;
- <minInclusive> — наименьшее значение определяемого типа;
- <totalDigits> — общее количество цифр в определяемом числовом типе — сужении типа decimal;
- <fractionDigits> — количество цифр в дробной части числа;
- <length> — длина значений определяемого типа;
- <maxLength> — наибольшая длина значений определяемого типа;
- <minLength> — наименьшая длина значений определяемого типа;
- <enumeration> — одно из перечислимых значений;

- <pattern> — регулярное выражение;
- <whiteSpace> — применяется при сужении типа string и определяет способ преобразования пробельных символов '\n', '\r', '\t'. Атрибут value этого тега принимает одно из трех значений:

    - preserve — не убирать пробельные символы;
    - replace — заменить пробельные символы пробелами;
    - collapse — после замены пробельных символов пробелами убрать начальные и конечные пробелы, а из нескольких подряд идущих пробелов оставить только один.

В тегах-фасетках можно записывать следующие атрибуты, называемые *базисными фасетками* (fundamental facets):

□ ordered — задает упорядоченность определяемого типа, принимает одно из трех значений:

- false — тип неупорядочен;
- partial — тип частично упорядочен;
- total — тип полностью упорядочен;

□ bounded — задает ограниченность или неограниченность типа значением true false;

□ cardinality — задает конечность или бесконечность типа значением finite или countably infinite;

□ numeric — показывает, числовой этот тип или нет, значением true или false.

Как видно из приведенных выше и ниже примеров, в одном сужении может быть несколько ограничений-фасеток. При этом фасетки <pattern> и <enumeration> задают независимые друг от друга ограничения, их можно мысленно объединить союзом "или". Остальные фасетки задают общие, совместно накладываемые ограничения, их можно мысленно объединить союзом "и".

2.2.3. Список. Простой тип-список — это тип элементов, в теле которых записываются через пробел несколько значений одного и того же простого типа. Например, в документе XML может встретиться такой элемент, содержащий список целых чисел:

```
<days>21 34 55 46</days>
```

Список определяется компонентом list, в котором атрибутом itemType указывается тип элементов определяемого списка. Тип элементов списка можно определить и в содержимом элемента list. Например, показанный выше элемент документа XML days можно определить в схеме так:

```
<xsd:element name="days" type="listOfInteger" />
```

а использованный при его определении тип listOfInteger задать как список не более чем из пяти целых чисел следующим образом:

```
<xsd:simpleType name="listOfInteger">
    <xsd:restriction>
      <xsd:simpleType>
         <xsd:list itemType="xsd:integer" />
      </xsd:simpleType>
   <xsd:maxLength value="5" />
</xsd:restriction>
 </xsd:simpleType>
```

При определении списка можно применять фасетки <iength>, <minLength>, <maxLength>, <enumeration>, <pattern>. В приведенном выше примере список — тело элемента days — не может содержать более пяти чисел.

2.2.4. Объединение. Простой тип-объединение определяется компонентом union, в котором атрибутом memberTypes можно указать имена объединяемых типов. Например:

```
<xsd:union memberTypes="xsd:string xsd:integer listOfInteger" />
```

Другой способ — записать в содержимом компонента union определения простых типов, входящих в объединение. Например:

```
<xsd:attribute name="size">
   <xsd:simpleType>
      <xsd:union>
         <xsd:simpleType>
            <xsd:restriction base="xsd:positivelnteger">
                <xsd:minlnclusive value="8"/>
                 <xsd:maxlnclusive value="72"/>
            </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
               <xsd:enumeration value="small"/>
               <xsd:enumeration value="medium"/>
               <xsd:enumeration value="large"/>
            . </xsd:restriction>
        </xsd:simpleType>
     </xsd:union>
  </xsd:simpleType>
</xsd:attribute>
```

После этого атрибут size можно использовать, например, так:

```
<font  size="large">Глава  1/font>
<font  size='12'>Простой текст</font>
```

2.3. Объявление элементов и их атрибутов

Элементы, из которых будет состоять документ XML, объявляются в схеме компонентом element:

```
<xsd:element name="имя элемента" type="Тип элемента"
    minOccurs="наименьшее число появлений элемента в документе"
    maxOccurs="наибольшее число появлений"  />
```

Значение по умолчанию необязательных атрибутов minOccurs и maxoccurs равно 1. Это означает, что если эти атрибуты отсутствуют, то элемент должен появиться в документе XML ровно один раз. Например:

```
<xsd:element name="degree" type="xsd:nonPositiveInteger" />
```

Указание типа элемента в атрибуте type удобно, если это встроенный простой тип или тип, определенный заранее. Тогда в атрибуте type можно записать только имя типа.

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

```
<xsd:element name ="имя элемента" >
      Определение типа элемента
</xsd:element>
```

Объявление атрибута элемента тоже несложно:

```
<xsd:attribute name="имя атрибута" type="тип атрибута"
  use="обязательность атрибута" default="значение по умолчанию" />
```

Необязательный атрибут use принимает три значения:

- optional — описываемый атрибут необязателен (это значение по умолчанию);
- required — описываемый атрибут обязателен;
- prohibited — описываемый атрибут неприменим. Это значение полезно при определении подтипа, чтобы отменить некоторые атрибуты базового типа.

Пример:

```
<xsd:attribute name="id" type="positivelnteger" use="required" />
```

Если описываемый атрибут необязателен, то атрибутом default можно задать его значение по умолчанию:

```
<xsd:attribute name="name" type="NCName" default="anonymous" />
```

Определение типа атрибута, — а это должен быть простой тип, — можно вынести в содержимое элемента attribute:

```
<xsd:attribute пате="имя атрибута">
      Тип атрибута
</xsd:attribute>
```

2.4. Определение сложных типов

2.4.1. Введение. Напомним, что тип элемента называется *сложным,* если в элемент вложены другие элементы и/или в открывающем теге элемента есть атрибуты.

Сложный тип определяется компонентом complexType, имеющим вид:

```
<xsd:complexType пате="имя типа" >
      Определение типа
</xsd:complexType>
```

Необязательный атрибут name задает имя типа, а в содержимом компонента complexType описываются элементы, входящие в сложный тип, и/или атрибуты открывающего тега.

Определение сложного типа можно разделить на три группы:

- □ определение типа пустого элемента;
- □ определение типа элемента с простым телом;
- □ определение типа элемента, содержащего вложенные элементы.

Рассмотрим эти определения подробнее.

2.4.2. Определение типа пустого элемента. Проще всего определяется тип пустого элемента — элемента, не содержащего тела, а содержащего только атрибуты в открывающем теге. Таков, например, элемент name листинга 1.2. Каждый атрибут объявляется одним компонентом attribute, как в предыдущем разделе, например:

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

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

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

а в документе XML использовать это объявление:

```
<image href="http://some.com/images/myface.gif" />
```

2.4.3. Определение типа элемента с простым телом. Немного сложнее описание элемента, содержащего тело простого типа и атрибуты в открывающем теге. Этот тип отличается от простого типа только наличием атрибутов и определяется компонентом simpiecontent. В теле этого компонента должен быть либо компонент restriction, либо компонент extension, атрибутом base задающий тип (простой) тела описываемого элемента.

В компоненте extension указываются атрибуты открывающего тега описываемого элемент

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

Эту конструкцию можно описать словами так: "Определяется тип calcResuitType элемента, тело которого содержит значения встроенного простого типа xsd: decimal. Простой тип расширяется тем, что к нему добавляются атрибуты unit и precision".

Если в схеме объявить элемент result этого типа следующим образом:

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

то в документе XML можно написать:

```
<result unit="cM" precision="2">123.25</result>
```

В компоненте restriction, кроме атрибутов, описывается простой тип содержимого элемента и/или фасетки, ограничивающие тип, заданный атрибутом base. Например:

```
   <xsd:complexType name="calcResultType">
    <xsd:simpleContent>
        <xsd:restriction base="xsd:decimal">
         <xsd:totalDigits value="8" />
        <xsd:attribute name="unit" type="xsd:string" />
         <xsd:attribute name="precision"
            type="xsd:nonNegativeInteger" />
      </xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
```

2.4.4. Определение типа вложенных элементов. Если значениями определяемого сложного типа будут элементы, содержащие вложенные элементы, как, например, элементы address, phone-list , то перед тем, как перечислять их описания, надо выбрать *модель группы* (model group) вложенных элементов. Дело в том, что вложенные элементы, составляющие определяемый тип, могут появляться или в определенном порядке, или в произвольном порядке, кроме того, можно выбирать только один из перечисленных элементов. Эта возможность и называется *моделью группы* элементов. Она определяется одним из трех компонентов: sequence, all или choice.

Компонент sequence применяется в том случае, когда перечисляемые элементы должны записываться в документе "в определенном порядке. Пусть, например, мы описываем книгу. Сначала определяем тип:

```
<xsd:complexType name="bookType">
  <xsd:sequence maxOccurs="unbounded">
     <xsd:element name="author" type="xsd:normalizedString"
                                                  minOccurs="0" />
      <xsd:element name="title" type="xsd:normalizedString" />
     <xsd:element name="pages" type="xsd:positivelnteger"
                 minOccurs="0" />
     <xsd:element name="publisher" type="xsd:normalizedString"
                 minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

Потом описываем элемент:

```
<xsd:element name="book" type="bookType" />
```

Элементы author, title, pages И publisher должны входить в элемент book именно в таком порядке. В документе XML надо писать:

```
<book>
  <author>М. Ильф, Е., Петров В.</author>
  <title>Золотой теленок< /title >
<publisher>Ильф и Петров</publisher> </book>
```

Если же вместо компонента xsd: sequence записать компонент xsd:all, то элементы author, title, pages и publisher можно перечислять в любом порядке.

Компонент choice применяется в том случае, когда надо выбрать один из нескольких элементов. Например, при описании журнала вместо издательства, описываемого элементом publisher, надо записать название журнала. Это можно определить так:

```
<xsd:complexType name="bookType">
  <xsd:sequence maxOccurs="unbounded">
     <xsd:element name="author" type="xsd:normalizedString"
minOccurs="0" />
    <xsd:element name="title" type="xsd:normalizedString" />
     <xsd:element name="pages" type="xsd:positivelnteger"
minOccurs="0" />
    <xsd:choice>
        <xsd:element name="publisher"
   type="xsd:normalizedString" minOccurs="0" />
        <xsd:element name="magazine"
   type="xsd:normalizedString" minOccurs="0" />
     </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

Как видно из этого примера, компонент choice можно вложить в компонент sequence или, наоборот, вложить компонент sequence в компонент choice. Такие вложения можно проделать сколько угодно раз. Кроме того, каждая группа в этих моделях может появиться сколько угодно раз, т. е.

В компоненте choice тоже можно записать атрибут maxOccurs="unbounded".

Модель группы all отличается в этом от моделей sequence и choice. В компоненте all не допускается использование компонентов sequence И choice. Аналогично, в компонентах sequence и choice нельзя применять компонент all. Каждый элемент, входящий в группу модели all, может появиться не более одного раза, т. е. атрибут maxoccurs этого элемента может равняться только единице.

2.4.5. Определение типа со сложным телом. При определении сложного типа можно воспользоваться уже определенным, *базовым,* сложным типом, расширив его

дополнительными элементами, или, наоборот, удалив из него некоторые элементы. Для этого необходимо применить компонент compiexcontent. В этом компоненте, так же как и в компоненте simpieContent, записывается либо компонент extension, если надо расширить базовый тип, либо компонент restriction, если нужно его сузить. Базовый тип указывается атрибутом base, так же как и при записи компонента simpieContent, но теперь это должен быть сложный, а не простой тип!

Расширим, например, определенный выше тип Ьооктуре, добавив год издания — элемент year:

```
<xsd:complexType name="newBookType">
        <xsd:complexContent>
          <xsd:extension base="bookType">
            <xsd:sequence>
                  <xsd:element name="year" type="xsd:gYear">
             </xsd:sequence>
          </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
```

При сужении базового типа компонентом restriction надо перечислить те элементы, которые останутся после сужения. Например, оставим в типе newbookType только автора и название книги из типа Booktype:

```
  <xsd:complexType name="newBookType">
   <xsd:complexContent>
    <xsd:restriction base="bookType">
      <xsd:sequence>
         <xsd:element name="author" type="xsd:normalizedString"
            minOccurs="0" />
      <xsd:element name="title" type="xsd:normalizedString" />
       </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
```

Это описание выглядит странно. Почему надо заново объявлять все элементы, остающиеся после сужения? Не проще ли определить новый тип?

Дело в том, что в язык XSD внесены элементы объектно-ориентированного программирования, которых мы не будем касаться. Расширенный и суженный типы связаны со своим базовым типом отношением наследования, и к ним можно применить операцию подстановки. У всех типов языка XSD есть общий предок — базовый тип апуТуре. От него наследуются все сложные типы. Это подобно тому, как у всех классов Java есть общий предок — класс object, а все массивы наследуются от него. От базового типа апуТуре наследуется и тип anySimpieType — общий предок всех простых типов.

Таким образом, сложные типы определяются как сужение типа апуТуре. Если строго подходить к определению сложного типа, то определение типа bookType, сделанное в начале предыдущего раздела, надо записать так:

```
  <xsd:complexType name="bookType">
   <xsd:complexContent>
     <xsd:restriction base="xsd:anyType">
      <xsd:sequence maxOccurs="unbounded">
         <xsd:element name="author" type="xsd:normalizedString"
   minOccurs="0" />
       <xsd:element name="title" type="xsd:normalizedString" />
         <xsd:element name="pages"
   type="xsd:positivelnteger" minOccurs="0" />
        <xsd:element name="publisher" type="xsd:normalizedString"
  minOccurs="0" />
</xsd:sequence>
```

```
    </xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
```

2.5. Создание XSD с использованием именованных типов

В листинге приведена схема документа – записной книжки:

```
      <?xml version-"!.0"?>
      <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="notebook" type="notebookType" />
<xsd:complexType name="notebookType">
            <xsd:element name="person" type="personType"
      minOccurs="0" maxOccurs="unbounded" />
      </xsd:complexType>
      <xsd:complexType name="personType">
<xsd:sequence>
       <xsd:element name="name">
         <xsd:complexType>
           <xsd:attribute name="first"  type="xsd:string"
           use="optional" /> <xsd:attribute name="second"
     type="xsd:string"
           use="optional" /> <xsd:attribute name="surname"
     type="xsd:string"
          use="required" />
   </xsd:complexType>
       </xsd:element>
       <xsd:element name="birthday" type="ruDate" minOccurs="0" />
         <xsd:element name="address" type="addressType"
    minOccurs="0" maxOccurs="unbounded" />
         <xsd:element name="phone-list" type="phone-listType"
    minOccurs="0" />
      </xsd:sequence> </xsd:complexType>
      <xsd:complexType name="addressType" > <xsd:sequence>
       <xsd:element name="street" type="xsd:string" /> <xsd:element
  name="city"  type="cityType" /> <xsd:element name="zip"
  type="xsd:positivelnteger"
      </xsd:sequence> </xsd:complexType>
      <xsd:complexType name='cityType'>
       <xsd:simpleContent>
          <xsd:extension base='xsd:string' >
           <xsd:attribute name='type' type='placeType'
       default='город' /> </xsd:extension>
  </xsd:simpleContent>
      </xsd:complexType>
      <xsd:simpleType name="placeType">
       <xsd:restriction base = "xsd:string">
         <xsd:enumeration value^'^opofl" />
         <xsd:enumeration value="поселок" />
       <xsd:enumeration уа1ие="деревня" /> </xsd:restriction>
      </xsd:simpleType>
      <xsd:complexType name="phone-listType">
            <xsd:element  name="work-phone"  type="xsd:string"  minOccurs="0"
     maxOccurs="unbounded" />
            <xsd:element  name="home-phone"  type="xsd:string"  minOccurs="0"
     maxOccurs="unbounded" />
      </xsd:complexType>
      <xsd:simpleType name="ruDate">
       <xsd:restriction base="xsd:string">
       <xsd:pattern value="[0-9]{2}. [0-9]{2}. [0-9]{4}" />
  </xsd:restriction>
      </xsd:simpleType> </xsd:schema>
```

Листинг, как обычный документ XML, начинается с пролога, показывающего версию XML и определяющего стандартное пространство имен схемы XML с идентификатором http://www.w3.org/2001/XMLSchema. С этим идентификатором связан префикс xsd. Конечно, префикс может быть другим, часто пишут префикс xa.

Все описание схемы нашей адресной книжки заключено в третьей строке, в которой указано, что адресная книга состоит из одного элемента с именем notebook, имеющего сложный тип notebookType. Этот элемент должен появиться в документе ровно один раз. Остальная часть листинга посвящена описанию типа этого элемента и других типов.

Определение сложного типа notebookType занимает три строки листинга, не считая открывающего и закрывающего тега, и просто говорит о том, что данный тип составляют несколько элементов person типа personType.

Определение типа personType немногим сложнее. Оно означает, что этот тип составляют четыре элемента: name, birthday, address и phone-list. Для элемента name сразу же объявлены необязательные атрибуты first и second простого типа string, определенного в пространстве имен xsd. Тип обязательного атрибута surname — тоже string.

Далее в листинге определяются оставшиеся типы addressType, phone-listType и ruDate. Необходимость определения простого типа ruDate возникает потому, что встроенный в схему XML тип date предписывает записывать дату в виде 2003-02-22, а в России принят формат 22.02.2003. Тип ruDate определяется как сужение (restriction) типа string с помощью шаблона. Шаблон (pattern) для записи даты в виде ДД.ММ.ГГГГ задается регулярным выражением.

2.6. Создание XSD с использованием неименованных типов

Почти все описанные в листинге типы используются только один раз. Поэтому необязательно давать типу имя. Схема XML, как говорилось выше, позволяет определять безымянные типы. Такое определение дается внутри описания элемента. Именно так в листинге описаны атрибуты элемента name. В следующем листинге показано упрощенное описание схемы адресной книги.

```
      <?xml version='1.0'?>
      <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema'>
      <xsd:element name='notebook'>
<xsd:complexType> <xsd:sequence>
      <xsd:element name='person' maxOccurs='unbounded'>
<xsd:complexType> <xsd:sequence>
        <xsd:element name='name'> <xsd:complexType>
          <xsd:attribute name='first'
             type='xsd:string' use='optional' />
          <xsd:attribute name='second'
             type='xsd:string' use='optional' />
           <xsd:attribute name='surname'
              type='xsd:string' use='required' />
        </xsd:complexType> </xsd:element>
        <xsd:element name='birthday'> <xsd:simpleType>
           <xsd:restriction base='xsd:string'>
           <xsd:pattern value=' [0-9] {2}.. [0-9] {2} . [0-9] (4} ' />
    </xsd:restriction>
        </xsd:simpleType> </xsd:element>
        <xsd:element name='address' maxOccurs='unbounded'>
 <xsd:complexType> <xsd:sequence>
           <xsd:element name='street' type='xsd:string' /> <xsd:element
    name='city'> <xsd:complexType> <xsd:simpleContent>
             <xsd:extension base='xsd:string'>
                <xsd:attribute name='type' type='xsd:string'
             use='optional' default='gorod' /> </xsd:extension>
```

```
          </xsd:simpleContent> </xsd:complexType>
     </xsd:element>
            <xsd:element name='zip' type='xsd:positivelnteger' />
          </xsd:sequence>
    </xsd:complexType> </xsd:element>
          <xsd:element name='phone-list'>
    <xsd:complexType> <xsd:sequence>
              <xsd:element name='work-phone' type='xsd:string'
          minOccurs='0' maxOccurs='unbounded' />
              <xsd:element name='home-phone' type='xsd:string'
          minOccurs='0' maxOccurs='unbounded' />
          </xsd:sequence>
    </xsd:complexType> </xsd:element>
        </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
        </xsd:sequence>
  </xsd:complexType>
      </xsd:element>
</xsd:schema>
```

3. ОБОРУДОВАНИЕ

Персональный компьютер типа IBM PC Pentium, операционная система MS Windows 7/8, интегрированная среда разработки приложений MS Visual Studio 2010/12/13 с комплектом документации MSDN, редактор XML Notepad и инструментальная система Altova XMLSpy, файл МУ_ЛР_ВМиДПО.doc (методические указания к лабораторным работам), 200 Мб свободной памяти на логическом диске, содержащем каталог Vmd\lbr4.

4. ЗАДАНИЕ НА РАБОТУ

4.1. Ознакомиться с основными встроенными типами языка XSD и способами определения новых простых и сложных типов элементов и атрибутов, с использованием именованных и неименованных типов при объявлении элементов и атрибутов.

4.2. Для XML-документа, заданного преподавателем, разработать две XML-схемы на языке XSD: с именованными и неименованными типами элементов и атрибутов. Предлагаемые варианты XML-документов приведены ниже.

Вариант 1, 6:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<Bands>
  <Band ID="1" Name=”Beatles”>
    <NumberOfMembers>4</NumberOfMembers>
    <HomeTown>Liverpool England</HomeTown>
    <CD Name="Yellow Submarine" />
  </Band>
  <Band ID="2" Name=”Citizen King”>
    <NumberOfMembers>5</NumberOfMembers>
    <HomeTown>Milwaukee, WI</HomeTown>
    <CD Name="Brown Paper Bag"/>
  </Band>
</Bands>
```

Вариант 2, 7:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
  <dogs>
    <dogsCaption>Собаки</dogsCaption>
    <dogsCaptionName>Кличка</dogsCaptionName>
    <dogsCaptionWeight>Вес</dogsCaptionWeight>
    <dogsCaptionColor>Цвет</dogsCaptionColor>
    <dog Name="Шарик" >
```

```
      <dogWeight unit="кг">24</dogWeight>
      <dogColor>рыжий с черными подпалинами</dogColor>
    </dog>
    <dog Name="Тузик">
      <dogWeight unit="кг">10</dogWeight>
      <dogColor>белый с черными пятнами</dogColor>
    </dog>
    <dog Name="Трезор">
      <dogWeight unit="кг">18</dogWeight>
      <dogColor>черный</dogColor>
    </dog>
  </dogs>
```

Вариант 3, 8:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<jornal title="Мир ПК">
    <fone-list>
         <home-fone>333311</home-fone>
         <work-fone>334455</work-fone>
         <work-fone>334455</work-fone>
    </fone-list>
    <email>Ivanov@tula.ru</email>
    <email>Vanja@tula.ru</email>
    <url>www.Ivanov.tula.ru</url>
    <article id="1">
      <title>Технология ASP.NET 2.0</title>
      <author>Смит Дж. </author>
      <author>Петзолд Дж. </author>
      <price name="Рубли">320.99</price>
    </article>
    <article id="2">
      <title>Веб-сервисы ASP.NET 2.0</title>
      <author>Смит Дж. </author>
      <price name="Рубли">420.99</price>
    </article>
</jornal>
```

Вариант 4, 9:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<notebook>
  <person ID="1">
    <name first="Иван" second="Петрович" surname="Сидоров" />
    <address  city="Тула street="Садовая, 12 – 34"  zip="300001" />
    <fone-list>
        <home-fone>333311</home-fone>
        <work-fone>334455</work-fone>
        <work-fone>334455</work-fone>
    </fone-list>
    <email>Ivanov@tula.ru</email>
    <email>Ivan@tula.ru</email>
    <url>www.Ivanov.tula.ru</url>
  </person ID="2">
  <person>
    . . .
  </person>
</notebook>
```

Вариант 5, 10:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<SalesTransactions Year="2011">
    <January>
      <Transaction Type="Cash" Day="1">
        <Processor>Beth</Processor>
        <Sale>
           <Amount>100</Amount>
           <Customer>Acme Imports"</Customer>
         </Sale>
      </Transaction>
```

```
      <Transaction Type="Credit" Day="10">
        <Processor>Sara</Processor>
        <Approved>Beth</Approved>
        <Credit Amount="500" Customer="Premier Business Software" />
      </Transaction>
      <Transaction Type="Credit" Day="11">
        <Processor>Beth</Processor>
        <Credit Amount="5" Customer="Premier Business Software" />
      </Transaction>
    </January>
  </SalesTransactions>
```

4.3. Выполнить верификацию XML-документа на соответствие каждой из двух XML-схем в одном из инструментальных средств, например, в редакторе XML Notepad или в инструментальной системе Altova XMLSpy.

5. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

6. ОФОРМЛЕНИЕ ОТЧЕТА

Отчет должен содержать:
цель и задачи работы;
исходный XML-документ;
две XML-схемы на языке XSD с именованными и неименованными типами элементов и атрибутов;
результаты верификации XML-документа разработанным XML-схемам;
анализ полученных результатов.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Хабибуллин И.Ш. Самоучитель XML. – СПб.: БХВ-Петербург, 2003. – 336 с.