# Лабораторная работа №1 Представление данных в формате XML

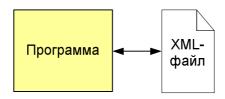
## Содержание

1.	. Лабораторная работа 1	2
	1.1 Постановка задачи	
	1.2. Требования к работе	
	1.3. Требования к оформлению отчета	
	1.4. Варианты заданий	
2.	Рекомендации по выполнению работы	
	2.1. Представление информации об объектах в программе	
	2.2. Управление объектами	
	2.3. Структура ХМL-файла	
	2.4. Чтение документа XML	
	2.5. Валидация документа по DTD.	
	2.6. Валидация документа по схеме XML	
	2.7. Формирование документа XML	
3.	Вопросы для самостоятельного изучения	

### 1. Лабораторная работа 1

#### 1.1 Постановка задачи

Разработать программу, обеспечивающую ввод и редактирование информации об объектах в соответствии с заданной предметной областью. Информация об объектах должна храниться в отдельном файле в формате XML.



## 1.2. Требования к работе

Программа не требует создания пользовательского интерфейса. Тестирование работоспособности программы осуществляется на основе сценариев, демонстрирующих возможности программы.

Сведения обо всех объектах хранятся в единственном файле XML. При запуске программы информация об объектах загружается из файла. По завершению работы с объектами, данные записываются в файл XML. Редактирование данных осуществляется в памяти программы.

Структура XML документа должна проверяться на соответствие заданному описанию DTD или XMLSchema (в соответствии с вариантом задания).

Характеристики автоматизируемых объектов определяются студентом самостоятельно. Обязательной характеристикой объекта является его уникальный идентификатор. Программа должна обеспечивать уникальность идентификатора при выполнении операций добавления и редактирования объектов.

Например, для варианта №1, объект Страна может иметь характеристики:

- код страны (уникальный идентификатор)
- название страны

а объект Город – характеристики:

- код города (уникальный идентификатор)
- ссылка на страну
- название города
- количество жителей
- признак столицы

Программа должна поддерживать выполнение следующих операций с данными:

- загрузка информации об объектах из файла
- сохранение информации об объектах в файл
- добавление нового объекта
- изменение параметров существующего объекта
- удаление объекта

• поиск объектов по заданным критериям и вывод информации об объектах

Например, для варианта №1 необходимо реализовать следующие операции:

- загрузка данных о городах и странах из файла
- сохранение данных о городах и странах в файл
- добавление новой страны
- добавление нового города для заданной страны
- удаление города
- удаление страны
- изменение параметров города и страны
- поиск города/страны по уникальному идентификатору
- выдача полного списка стран
- выдача списка городов, принадлежащих стране с заданным кодом

Обратите внимание, что во всех вариантах заданий объекты различных категорий находятся в иерархической зависимости. Например, в варианте  $N_2I$  у каждой страны может быть несколько городов, при этом один город принадлежит только одной стране.

Рекомендуемый язык программирования – Java.

#### 1.3. Требования к оформлению отчета

Отчет должен содержать:

- титульный лист
- постановку задачи
- исходный код программы
- описание программы (описание классов, методов, полей)
- пример xml-документа, содержащего информацию по заданной предметной области
- описание структуры xml-документа в формате DTD и XMLSchema (в независимости от реализованного в программе механизма проверки структуры документа, требуется ВКЛЮЧАТЬ В ОТЧЕТ ОБА ОПИСАНИЯ)

#### 1.4. Варианты заданий

Вариант 1		
Предметная область	Карта мира	
Объекты	Страны, Города	
Примечание	Карта мира содержит множество <i>стран</i> . Для каждой <i>страны</i> определено множество <i>городов</i> .	
Проверка структуры документа XML	DTD	

Вариант 2	
Предметная область	Библиотека
Объекты	Авторы, Книги
Примечание	Книги в библиотеке сгруппированы по <i>авторам</i> . У каждого <i>автора</i> имеется множество <i>книг</i> .
Проверка структуры документа XML	Схема XML

Вариант 3		
Предметная область	Отдел кадров	
Объекты	Подразделения, Сотрудники	
Примечание	Имеется множество подразделений предприятия. В каждом подразделении работает множество сотрудников.	
Проверка структуры документа XML	DTD	

Вариант 4		
Предметная область	Учебный отдел	
Объекты	Группы, Студенты	
Примечание	Имеется множество учебных <i>групп</i> . Каждая группа включает в себя множество <i>студентов</i> .	
Проверка структуры документа XML	Схема XML	

Вариант 5			
Предметная область	Автосалон		
Объекты	Производители Марки	авто	омобилей,
Примечание	Марки сгруппированы производителям. производителя множество марок.	авт У	гомобилей по каждого имеется
Проверка структуры документа XML	DTD		

Вариант 6		
Предметная область	Агентство новостей	
Объекты	Категории новостей, Новости	
Примечание	Новости сгруппированы по	
	категориям. У каждой категории	
	имеется множество новостей.	
Проверка структуры	Схема XML	
документа XML		

Вариант 7		
Предметная область	Продуктовый магазин	
Объекты	Категория продукта, Продукт	
Примечание	Продукты в магазине	
	сгруппированы по категориям.	
	Для каждой категории	
	определено множество	
	продуктов.	
Проверка структуры	DTD	
документа XML		

Вариант 8		
Предметная область	Футбол	
Объекты	Команды, Игроки	
Примечание	Имеется множество футбольных команд. Для каждой команды определено множество игроков.	
Проверка структуры документа XML	Схема XML	

Вариант 9		
Предметная область	Музыкальный магазин	
Объекты	Исполнители, Альбомы	
Примечание	В музыкальном магазине альбомы	
	сгруппированы по исполнителям.	
	Для каждого исполнителя задано	
	множество альбомов.	
Проверка структуры	DTD	
документа XML		

Вариант 10			
Предметная область	Аэропорт		
Объекты	Авиакомпании,	Рейсы	
Примечание	Имеется	множество	
	авиакомпаний.	Для каждой	
	авиакомпании	определены ее	
	рейсы.		
Проверка структуры	Схема XML		
документа XML			

Вариант 11		
Предметная область	Файловая система	
Объекты	Папки, Файлы	
Примечание	Имеется множество <i>папок</i> (независимых друг от друга). Для каждой <i>папки</i> определено множество <i>файлов</i> .	
Проверка структуры документа XML	DTD	

Вариант 12	
Предметная область	Расписание занятий
Объекты	Дни недели, Занятия
Примечание	Имеется множество <i>дней</i> . Для каждого <i>дня</i> определен перечень <i>занятий</i> .
Проверка структуры документа XML	Схема XML

Вариант 13	
Предметная область	Записная книжка
Объекты	Календарные дни, Мероприятия
Примечание	Имеется множество <i>дней</i> . Для каждого <i>дня</i> определен перечень <i>мероприятий</i> .
Проверка структуры документа XML	DTD

Вариант 14	
Предметная область	Видеомагазин
Объекты	Жанры, Фильмы
Примечание	Имеется множество жаров. Для
	каждого жанра определен
	перечень фильмов.
Проверка структуры	Схема XML
документа XML	

Вариант 15	
Предметная область	Железная дорога
Объекты	Дороги, Станции
Примечание	Имеется множество железных
	дорог. В ведомстве каждой
	дороги находится множество
	станций.
Проверка структуры	DTD
документа XML	

Вариант 16	
Предметная область	Склад
Объекты	Секции, Товары
Примечание	Товары на складе сгруппированы
	по секциям. Для каждой секции
	задано множество товаров.
Проверка структуры	Схема XML
документа XML	

Вариант 17	
Предметная область	Кафедра университета
Объекты	Преподаватели, Дисциплины
Примечание	На кафедре имеется множество преподавателей. Для каждого преподавателя задано множество дисциплин.
Проверка структуры документа XML	DTD

Вариант 18	
Предметная область	Программное обеспечение
Объекты	Производители, Программные продукты
Примечание	Программные продукты сгруппированы по производителям. Для каждого производителя задано множество продуктов.
Проверка структуры документа XML	Схема XML

Вариант 19	
Предметная область	Геометрия
Объекты	Многоугольники, Вершины
Примечание	Имеется множество многоугольников. Каждый многоугольник состоит из
Проверка структуры документа XML	произвольного числа <i>вершин</i> .  DTD

Вариант 20	
Предметная область	Схема метро
Объекты	Линии, Станции
Примечание	Имеется множество линий
	метрополитена. Каждая линия
	состоит из последовательности
	станций.
Проверка структуры	Схема XML
документа XML	

#### 2. Рекомендации по выполнению работы

#### 2.1. Представление информации об объектах в программе

Для представления информации об автоматизируемых объектах в программе рекомендуется использовать *классы*. В следующем примере мы объявим классы на языке Java, описывающие предметную область в соответствии с вариантом №1.

```
// Класс "Страна"
public class Country
    public int code; // Уникальный код страны
    public String name; // Название страны
    public Country(int inCode, String inName)
        // ... инициализация полей
// Класс "Город"
public class City
   public int code;  // Уникальный код города
public String name;  // Название города
    public boolean isCapital; // Признак столицы
    public int count; // Количество жителей
    public Country country; // Страна
    public City(int inCode, String inName, boolean inCapital,
                int inCount, Country inCountry)
       // ... инициализация полей
    }
}
```

Для хранения множества объектов удобно использовать динамические массивы. Управление динамическими массивами в Java можно осуществлять при помощи класса java.util.ArrayList:

```
import java.util.ArrayList;
...
// Объявляем массив стран
ArrayList<Country> countries = new ArrayList();
...
// Добавляем объект в массив
Country cntr = new Country(1, "Россия");
countries.add(cntr);
...
// Определяем размер массива
System.out.println(countries.size());
...
// Обращаемся к элементу массива
System.out.println(countries.get(0).name);
```

### 2.2. Управление объектами

Для управления объектами следует предусмотреть отдельный класс. Данный класс будет содержать массивы объектов и предоставлять методы для редактирования объектов, а также для загрузки и сохранения данных в XML-файл:

```
public class Worldmap
    // Массив стран
   private ArrayList<Country> countries;
    // Массив городов
   private ArrayList<City> cities;
   // Записать данные в файл XML
   public void saveToFile(String filename)
    { //...
   // Прочитать данные из файла XML
   public void loadFromFile(String filename)
    { // ...
    // Добавить новую страну
   public void addCountry(int code, String name)
      // если страны с заданным кодом в массиве countries еще нет -
         добавляем новую страну в массив
     // в противном случае генерируем исключение
    // Получить страну с заданным кодом
   public Country getCountry(int code)
      // возвращаем страну с заданным кодом
     // если страны с заданным кодом в массиве countries нет -
     //
         генерируем исключение
    // Получить страну с заданным номером
   public Country getCountryInd(int index)
      // возвращаем страну с заданным порядковым номером
     // если номер выходит за границы индексов массива
      // генерируем исключение
    // Получить количество стран
   public int countCountries()
        // возвращаем количество стран
```

```
// Удалить страну
public void deleteCountry(int code)
    // Удаляем страну с заданным кодом, а также все города,
    // ссылающиеся на данну страну
    // Если страны с заданным кодом в массиве countries нет -
    // генерируем исключение
}
// Добавить новый город для заданной страны
public void addCity(int code, String name, boolean isCapital,
            int count, int countryCode)
  // если город с заданным кодом code уже есть
  // - генерируем исключение
  // если страны с заданным кодом countryCode нет
  // - генерируем исключение
  // в противном случае, добавляем новый город
 . . .
 . . .
```

## 2.3. Структура ХМL-файла

}

Для хранения информации об автоматизируемых объектах используется один XML-файл. Файл имеет иерархическую структуру, отражающую зависимости между объектами.

Например, для варианта №1, XML-файл может выглядеть следующим образом:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
< map >
    <country id="1" name="Россия">
        <city id="1" name="Mocква" iscap="1" count="11000000"/>
        <city id="2" name="Владивосток" iscap="0" count="850000"/>
        <city id="3" name="Воронеж" iscap="0" count="500000"/>
    </country>
    <country id="2" name="Украина">
        <city id="4" name="Киев" iscap="1" count="3000000"/>
        <city id="5" name="Севастополь" iscap="0" count="400000"/>
    </country>
    <country id="3" name="Белоруссия">
        <city id="6" name="Минск" iscap="1" count="1700000"/>
        <city id="7" name="Витебск" iscap="0" count="350000"/>
    </country>
</map>
```

#### 2.4. Чтение документа XML

Обработка документа XML внутри программы Java может осуществляться при помощи парсеров SAX, DOM, JDOM, а также других стандартных средств. Далее мы посмотрим, как работать с DOM-парсером.

DOM (Document Object Model – объектная модель документа) – используется для древовидного представления информации, хранящейся в документе XML. DOM включает в себя набор интерфейсов, содержащихся в пакете org.w3c.dom, в частности:

- Document представление документа XML
- Element элемент XML
- Text текстовая строка

Перед работой с документом XML, необходимо создать парсер для документа (javax.xml.parsers.DocumentBuilder). Делается это при помощи специальной фабрики паресеров (javax.xml.parsers.DocumentBuilderFactory):

```
DocumentBuilderFactory dbf = null;
DocumentBuilder db = null;
try
{
    dbf = DocumentBuilderFactory.newInstance();
    db = dbf.newDocumentBuilder();
}
catch(ParserConfigurationException e)
{
}
```

После того как парсер создан, можно создавать документ на основе файла xml:

```
Document doc = null;
try
{
    doc = db.parse(new File("d:\\map.xml"));
}
catch (SAXException ex)
{
}
catch (IOException ex)
{
}
```

Чтобы получить корневой элемент документа используется метод getDocumentElement:

```
Element root = doc.getDocumentElement();
```

Для получения названия элемента используется метод getTagName.

Для получения дочерних элементов, содержащихся внутри данного элемента, можно использовать метод getElementsByTagName. Данный метод возвращает коллекцию элементов с заданным именем.

Для получения значения атрибута используется метод элемента getAttribute.

Для получения текста, содержащегося внутри элемента (и всех его дочерних элементов) используется метод getTextContent.

В следующем примере мы пройдем по xml-документу, представленному в предыдущем разделе, и выведем на экран информацию об объектах, описанных в документе.

```
// Получаем корневой элемент
Element root = doc.getDocumentElement();
if (root.getTagName().equals("map"))
    // Получаем коллекцию стран
   NodeList listCountries = root.getElementsByTagName("country");
    // Проходим по странам
    for (int i=0; i<listCountries.getLength(); i++)</pre>
        // Получаем текущую срану
        Element country = (Element)listCountries.item(i);
        String countryCode = country.getAttribute("id");
        String countryName = country.getAttribute("name");
        System.out.println(countryCode + " " +countryName + ":");
        // Получаем коллекцию городов для страны
        NodeList listCities = country.getElementsByTagName("city");
        // Проходим по городам
        for (int j=0; j<listCities.getLength(); j++)</pre>
            // Получаем текущий город
            Element city = (Element)listCities.item(j);
            String cityName = city.getAttribute("name");
            System.out.println("
                                  " +cityName);
        }
    }
}
   Результат работы данного примера будет выглядеть вот так:
```

```
1 Россия:
    Москва
    Владивосток
    Воронеж
2 Украина:
    Киев
    Севастополь
3 Белоруссия:
    Минск
    Витебск
```

Импорты, которые могут потребоваться для работы с XML:

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import java.io.*;
```

## 2.5. Валидация документа по DTD

Валидация документа XML – проверка корректности структуры документа XML. Валидация может осуществляться на основе описания DTD или схемы XML.

Для валидации документа XML по описанию DTD следует настроить соответствующим образом фабрику парсеров и парсер.

Для фабрики парсеров следует при помощи метода setValidating включить проверку структуры документа:

```
dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(true);
```

Для парсера следует разработать специальный обработчик ошибок, поддерживающий интерфейс *org.xml.sax.ErrorHandler*. Данный обработчик определяет действия, которые будут производиться парсером при возникновении ошибок в документе.

Ниже приводится пример простого обработчика, выводящего в консоль сообщения об ошибках в документе XML:

```
class SimpleErrorHandler implements ErrorHandler {
    // метод для обработки предупреждений
    public void warning(SAXParseException e) throws SAXException {
        System.out.println("Строка " +e.getLineNumber() + ":");
        System.out.println(e.getMessage());
    }
    // метод для обработки ошибок
    public void error(SAXParseException e) throws SAXException {
        System.out.println("Строка " +e.getLineNumber() + ":");
        System.out.println(e.getMessage());
    }
    // метод для обработки критических ошибок
    public void fatalError(SAXParseException e) throws SAXException {
        System.out.println("Строка " +e.getLineNumber() + ":");
        System.out.println(e.getMessage());
    }
}
```

Перед вызовом метода parse, следует создать экземпляр обработчика и зарегистрировать его в парсере:

```
db = dbf.newDocumentBuilder();
db.setErrorHandler(new SimpleErrorHandler());
...
Document doc = db.parse(new File(...));
```

Следует отметить, что ссылка на требуемое описание DTD должна размещаться внутри документа XML:

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<!DOCTYPE map SYSTEM "map.dtd">
<map> ... </map>
```

#### 2.6. Валидация документа по схеме XML

Один из распространенных способов валидации документа по схеме XML – использование средств *javax.xml.validation*.

Для валидации документа следует создать объект *схема* класса javax.xml.validation.Schema на основе требуемой схемы XML и использовать этот объект в фабрике парсеров.

Схема создается внутри фабрики схем javax.xml.validation.SchemaFactory:

```
// Создаю фабрику схем
SchemaFactory sf =
    SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
// Создаю схему из хsd файла
Schema s = sf.newSchema(new File("map.xsd"));
```

Привязка схемы к фабрике парсеров осуществляется при помощи метода setSchema:

```
// Создаю фабрику парсеров
dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(false);
// Привязываю схему к фабрике парсеров
dbf.setSchema(s);
```

Для парсера следует разработать специальный обработчик ошибок, поддерживающий интерфейс *org.xml.sax.ErrorHandler*. Данный обработчик определяет действия, которые будут производиться парсером при возникновении ошибок в документе:

```
class SimpleErrorHandler implements ErrorHandler {
    public void warning(SAXParseException e) throws SAXException {
        System.out.println(e.getMessage());
    }

    public void error(SAXParseException e) throws SAXException {
        System.out.println(e.getMessage());
    }

    public void fatalError(SAXParseException e) throws SAXException {
        System.out.println(e.getMessage());
    }
}
```

Перед вызовом метода parse, следует создать экземпляр обработчика и зарегистрировать его в парсере:

```
db = dbf.newDocumentBuilder();
db.setErrorHandler(new SimpleErrorHandler());
...
Document doc = db.parse(new File(...));
```

#### 2.7. Формирование документа ХМL

Теперь посмотрим, как сформировать документ XML на основе имеющихся данных с использованием дерева DOM.

```
DocumentBuilderFactory dbf = null;
DocumentBuilder db = null;
Document doc = null;
dbf = DocumentBuilderFactory.newInstance();
db = dbf.newDocumentBuilder();
// Создаем "чистый" документ XML
doc = db.newDocument();
// Создаем корневой элемент
Element root = doc.createElement("map");
doc.appendChild(root);
// Создаем объект "страна"
Element country1 = doc.createElement("country");
country1.setAttribute("id", "1");
country1.setAttribute("name", "Россия");
root.appendChild(country1);
// Создаем еще один объект "страна"
Element country2 = doc.createElement("country");
country2.setAttribute("id", "2");
country2.setAttribute("name", "Украина");
root.appendChild(country2);
// Создаем объекты "города"
Element city1 = doc.createElement("city");
city1.setAttribute("id", "1");
city1.setAttribute("name", "Москва");
city1.setAttribute("count", "11000000");
city1.setAttribute("iscap", "1");
country1.appendChild(city1);
Element city2 = doc.createElement("city");
city2.setAttribute("id", "5");
city2.setAttribute("name", "Севастополь");
city2.setAttribute("count", "400000");
city2.setAttribute("iscap", "0");
country2.appendChild(city2);
. . .
```

Чтобы сохранить полученное дерево в файл, можно использовать Transformation API для XML (пакет javax.xml.transform):

```
Source domSource = new DOMSource(doc);
Result fileResult = new StreamResult(new File("d:\\map2.xml"));
TransformerFactory factory = TransformerFactory.newInstance();
```

```
Transformer transformer = factory.newTransformer();
transformer.setOutputProperty(OutputKeys.ENCODING,"WINDOWS-1251");
transformer.transform(domSource, fileResult);
```

#### Для этого понадобятся следующие дополнительные импорты:

```
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```

## 3. Вопросы для самостоятельного изучения

- Язык ХМС
- Спецификация XML Schema
- Язык описания схем DTD