Частное учреждение образования
Колледж бизнеса и права

УТВЕРЖДАЮ
Заведующий
методическим кабинетом
_____Е.В. Паскал
«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Дисциплина: «Основы кроссплатформенного программирования»

ЛАБОРАТОРНАЯ РАБОТА № 21
Инструкционно-технологическая карта

Тема: «Работа с XML на языке Java»

Цель: Научиться создавать разметку приложения на языке Java средствами XML.

Время выполнения: 6 часов

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения;
2. Ответить на контрольные вопросы;
3. Откомпилировать примеры программ из раздела «Теоретические сведения»;
4. Выполнить ИДЗ.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

Язык разметки XML (Extensible Markup Language) был разработан W3C. Главным преимуществом XML является совместимость данных, представленных в этом формате, с различными приложениями. Для данных XML – это то же самое, что и язык Java для информационных систем.

Язык XML был разработан на базе универсального языка разметки SGML. Собственно язык HTML, как язык разметки гипертекстовых документов, также произошел от SGML.

Основная идея XML – это текстовое представление с помощью тегов, структурированных в виде дерева данных. Древовидная структура хорошо описывает бизнес-объекты, конфигурацию, структуры данных и т.п. Данные в таком формате легко могут быть как построены, так и разобраны на любой системе с использованием любой технологии – для этого нужно лишь уметь

работать с текстовыми документами. Почти все современные технологии стандартно поддерживают работу с XML. Кроме того, такое представление данных удобочитаемо (humanreadable). Если нужен тег для представления имени, его можно создать: <name>Igor</name>.

DTD

Для описания структуры XML-документа используется DTD (Document Type Definition). DTD определяет, какие теги (элементы) могут использоваться в XMLдокументе, как эти элементы связаны (например, указывать на то, что элемент <book> включает дочерние элементы <price> и <author>), какие атрибуты имеет тот или иной элемент.

Зачем это нужно? В принципе, никто не требует создания DTD для XML-документа, программы-анализаторы будут обрабатывать XML-файл и без DTD. Но в этом случае остается только надеяться, что автор XML-файла правильно его сформировал.

Для того чтобы сформировать DTD, можно создать либо отдельный файл и описать в нем структуру документа, либо включить DTD-описание непосредственно в документ XML.

В первом случае в документ XML помещается ссылка на файл DTD:

```
<?xml version="1.0" standalone="yes" ?>
<! DOCTYPE journal SYSTEM "book.dtd">
```

Во втором случае описание элемента помещается в XML-документ:

```
<?xml version="1.0" ?>
...
<! DOCTYPE book [
<!ELEMENT book (price, author)>
...
]>
```

Описание элемента

Элемент в DTD описывается с помощью дескриптора !ELEMENT, в котором указывается название элемента и его содержимое. Так, если нужно определить элемент <book>, у которого есть дочерние элементы <price> и <author>, то можно сделать это следующим образом:

```
<!ELEMENT price PCDATA>
<!ELEMENT author PCDATA>
<!ELEMENT book (price, author)> \
```

В данном случае были определены два элемента price и author и описано их содержимое с помощью маркера PCDATA. Это говорит о том, что элементы могут содержать любую информацию, с которой может работать программаанализатор (PCDATA – parseable character data). Есть также маркеры EMPTY – элемент пуст и ANY – содержимое документа специально не описывается.

При описании элемента <book>, было указано, что он состоит из дочерних элементов <price> и <author>. Можно расширить это описание с помощью символов ‘+’, ‘*’, ‘?’, используемых для указания количества вхождений элементов. Так, например,

<!ELEMENT book (price, author+, caption?)>

означает, что элемент book содержит один и только один элемент price, несколько (минимум один) элементов author и необязательный элемент caption. Если существует несколько вариантов содержимого элементов, то используется символ ‘|’. Например:

<!ELEMENT book (PCDATA | body)>

В данном случае элемент book может содержать либо дочерний элемент body, либо PCDATA.

Описание атрибутов

Атрибуты элементов описываются с помощью дескриптора !ATTLIST, внутри которого задаются имя атрибута, тип значения, дополнительные параметры:

```
<!ATTLIST article
        id ID #REQUIRED
        about CDATA #IMPLIED
        type (actual | review | teach )  'actual' ">
```

В данном случае у элемента <article> определяются три атрибута: id, about, type. Существует несколько возможных значений атрибута, это:

CDATA – значением атрибута является любая последовательность символов;

ID – определяет уникальный идентификатор элемента в документе;

IDREF (IDREFS) – значением атрибута будет идентификатор (список идентификаторов), определенный в документе;

ENTITY (ENTITES) – содержит имя внешней сущности (несколько имен, разделенных запятыми);

NMTOKEN (NMTOKENS) – слово (несколько слов, разделенных пробелами).

Опционально можно задать значение по умолчанию для каждого атрибута. Значения по умолчанию могут быть следующими:

#REQUIRED – означает, что значение должно присутствовать в документе;

#IMPLIED – означает, что если значение атрибута не задано, то приложение должно использовать свое собственное значение по умолчанию;

#FIXED – означает, что атрибут может принимать лишь одно значение, то, которое указано в DTD.

Если в документе атрибуту не будет присвоено никакого значения, то его значение будет равно заданному в DTD.

Определение сущности

Сущность (entity) представляет собой некоторое определение, чье содержимое может быть повторно использовано в документе. Описывается сущность с помощью дескриптора !ENTITY:

```
<!ENTITY company 'Sun Microsystems'>
 ...
<sender>&company;</sender> ...
```

Программа-анализатор, которая будет обрабатывать файл, автоматически подставит значение Sun Microsystems вместо &company. В XML включено несколько внутренних определений:

&amplt – символ <;

&ampgt – символ >;

&ampamp – символ &;

&ampapos – символ апострофа ‘;

&ampquot – символ двойной кавычки “.

Кроме этого, есть внешние определения, которые позволяют включать содержимое внешнего файла:

<!ENTITY logotype SYSTEM "/image.gif" NDATA GIF87A>

Пусть существует XML-документ, содержащий данные адресной книги:

```
<?xml version="1.0"?>
<!DOCTYPE notepad SYSTEM "notepad.dtd">
 <notepad>
        <note login="rom">
                <name>Valera Romanchik</name>
                <tel>217819</tel>
                <url>http://www.bsu.by</url>
                <address>
                        <street>Main Str., 35</street>
                        <city>Minsk</city>
                        <country>BLR</country>
                </address>
        </note>
        <note login="goch">
                <name>Igor Blinov</name>
                <tel>430797</tel>
                <url>http://bsu.iba.by</url>
                <address>
                        <street>Deep Forest, 7</street>
                        <city>Polock</city>
                        <country>VCL</country>
                </address>
         </note>
</notepad>
```

Тогда файл DTD для этого документа будет иметь вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT notepad (note+)>
<!ELEMENT note (name,tel,url,address)>
<!ELEMENT address (street,city,country)>
<!ATTLIST note login ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT url (#PCDATA)>
```

XML-анализаторы

Анализатор (parser) – самый важный инструмент при работе с XML. Каждое приложение, работающее с XML, использует анализатор, который представляет собой некоторый компонент, находящийся между приложением и файлами XML.

Валидирующие и невалидирующие анализаторы

Документы XML могут быть либо well-formed, либо valid. Документы wellformed составлены в соответствии с синтаксическими правилами построения XML-документов. Документы не только сформированы синтаксически правильно, но и следуют некоторой структуре, которая описана в DTD.

Соответственно есть валидирующие и невалидирующие анализаторы. И те, и другие проверяют XML-документ на соответствие синтаксическим правилам, но только валидирующие анализаторы знают, как проверить XML-документ на соответствие структуре, описанной в DTD.

Никакой связи между видом анализатора и видом XML-документа нет. Валидирующий анализатор может разобрать XML-документ, для которого нет DTD, и, наоборот, невалидирующий анализатор может разобрать XML-документ, для которого есть DTD. При этом он просто не будет учитывать описание структуры документа.

Древовидная и событийная модели

Существует два вида взаимодействия приложения и анализатора: использовать модель, основанную на представлении содержимого файла XML в виде дерева объектов, либо событийную модель. Анализаторы, которые строят древовидную модель, – это DOM-анализаторы (Dynamic Object Model).

Анализаторы, которые генерируют события, – это SAXанализаторы (Simple API for XML).

В первом случае анализатор строит в памяти дерево объектов, соответствующее XML-документу. Далее вся работа ведется именно с этим деревом.

Во втором случае анализатор работает следующим образом: когда происходит анализ документа, анализатор генерирует события, связанные с различными участками XML-файла, а программа, использующая анализатор, решает, как реагировать на эти события. Так, анализатор будет генерировать событие о том, что он встретил начало документа либо его конец, начало элемента либо его конец, символьную информацию внутри элемента и т.д.

Когда следует использовать DOM-, а когда – SAX-анализаторы?

DOM-анализаторы следует использовать тогда, когда нужно знать структуру документа и может понадобиться изменять эту структуру либо использовать информацию из XML-файла несколько раз.

SAX-анализаторы используются тогда, когда нужно извлечь информацию о нескольких элементах из XML-файла либо когда информация из документа нужна только один раз.

Событийная модель

Как уже отмечалось, SAX-анализатор не строит дерево элементов по содержимому XML-файла. Вместо этого анализатор читает файл и генерирует события, когда находит элементы, атрибуты или текст. На первый взгляд, такой подход менее естествен для приложения, использующего анализатор, так как он не строит дерево, а приложение само должно догадаться, какое дерево элементов описывается в XML-файле.

Однако нужно учитывать, для каких целей используются данные из XMLфайла. Очевидно, что нет смысла строить дерево объектов, содержащее десятки тысячи элементов в памяти, если все, что необходимо, – это просто посчитать точное количество элементов в файле.

SAX-анализаторы и Java

SAX API определяет ряд событий, которые будут сгенерированы при разборе документов:

startDocument – событие, сигнализирующее о начале документа;

endDocument – событие, сигнализирующее о завершении документа;

startElement – данное событие будет сгенерировано, когда анализатор полностью обработает содержимое открывающего тега, включая его имя и все содержащиеся атрибуты;

endElement – событие, сигнализирующее о завершении элемента;

characters – событие, сигнализирующее о том, что анализатор встретил символьную информацию внутри элемента;

warning, error, fatalError – эти события сигнализируют об ошибках при разборе XML-документа.

В пакете org.xml.sax.helpers содержится класс DefaultHanlder, который содержит методы для обработки всех вышеуказанных событий. Для того чтобы создать простейшее приложение, обрабатывающее XML-файл, достаточно сделать следующее:

1. Создать класс, суперклассом которого будет DefaultHandler, и переопределить методы, отвечающие за обработку интересующих событий.

2. Создать объект-парсер класса org.xml.parsers.SAXParser.

3. Вызвать метод parse(), которому в качестве параметров передать имя разбираемого файла и экземпляр созданного на первом шаге класса. Следующий пример выведет на консоль содержимое XML-документа. Вывод производится в ответ на события, генерируемые анализатором.

Пример 1.Чтение и вывод XML-документа :

DemoSAXParser.java

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.Attributes;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
class MyHandler extends DefaultHandler {
public void startElement(String uri, String localName, String qName,  Attributes attrs) {
String s = "";
for (int i = 0; i < attrs.getLength(); i++) {
```

```
 s = attrs.getQName(i) + "="  + attrs.getValue(i) + " ";
}
System.out.print(qName + " " + s.trim());  }
public void endElement(String uri,  String localName, String qName) {
System.out.print(qName);
}
public void characters(char[] ch,  int start, int length) {
System.out.print(new String(ch, start, length));
} }
public class DemoSAXParser {
public static void main(String[] args) {
try {
SAXParser parser =  SAXParserFactory.newInstance().newSAXParser();
parser.parse("notepad.xml", new MyHandler());
} catch (Exception e) {
 e.printStackTrace();   }
} }
```

В следующем примере производится разбор документа notepad.xml, и инициализация на его основе набора объектов.

Пример 2: формирование коллекции объектов на основе XML-документа

```
MyParserDemo.java */
import org.xml.sax.Attributes;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.net.URL; import java.net.MalformedURLException;
import java.util.Vector;
interface ConstNote {
int   NAME = 1,   TEL = 2,  URL = 3,  STREET = 4, CITY = 5, COUNTRY = 6;
}
class DocHandler extends DefaultHandler  implements ConstNote {
Vector notes = new Vector();
Note curr = new Note();
int current = -1;
public Vector getNotes() {
return notes;  }
public void startDocument() {
System.out.println("parsing started");  }
public void endDocument() {
System.out.print("");  }
public void startElement(String uri,  String localName, String qName,    Attributes attrs) {
if (qName.equals("note")) {
curr = new Note();
curr.setLogin(attrs.getValue(0));   }
if (qName.equals("name"))
current = NAME;
else if (qName.equals("tel"))
current = TEL;
else if (qName.equals("url"))
current = URL;
```

```
else if (qName.equals("street"))
current = STREET;
else if (qName.equals("city"))
current = CITY;
else if (qName.equals("country"))
current = COUNTRY;  }
public void endElement(String uri,  String localName, String qName) {
if (qName.equals("note"))
notes.add(curr);  }
public void characters(char[] ch,  int start, int length) {
String s = new String(ch, start, length);
try {
switch (current) {
case NAME:    curr.setName(s);    break;
case TEL:    curr.setTel(Integer.parseInt(s));    break;
case URL:
try {
curr.setUrl(new URL(s));
} catch (MalformedURLException e) {    }    break;
case STREET:    curr.address.setStreet(s);    break;
case CITY:    curr.address.setCity(s);    break;
case COUNTRY:    curr.address.setCountry(s);    break;   }
} catch (Exception e) {   System.out.println(e);   }
 } }
public class MyParserDemo {
public static void main(String[] args) {
try {
SAXParser parser =  SAXParserFactory.newInstance().newSAXParser();
DocHandler dh = new DocHandler();
Vector v;
if (dh != null)
parser.parse("notepad.xml", dh);
v = dh.getNotes();
for (int i = 0; i < v.size(); i++)
System.out.println(((Note)
v.elementAt(i)).toString());
} catch (Exception e) {
e.printStackTrace();   }
} }
```

В результате на консоль будет выведена следующая информация: parsing started
rom
Valera Romanchik 217819 http://www.bsu.by address:Main Str., 35 Minsk BLR
goch
Igor Blinov 430797 http://bsu.iba.by address:Deep Forest, 7 Polock VCL

Класс, объект которого формируется на основе информации из XMLдокумента, имеет следующий вид:

Пример 3: класс сущности:

Note.java

```
import java.net.URL;
class Note {
private String name, login;
```

```
        private int tel;
        private URL url;
        public Address address = new Address();
        public void setAddress(Address address) {
        this.address = address;  }
        public void setLogin(String login) {
        this.login = login;  }
        public void setName(String name) {
        this.name = name;  }
        public void setTel(int tel) {
        this.tel = tel;  }
        public String toString() {
        return login + " " + name + " " + tel + " "  + url + "\n\t address:" + address.street + " "  +
address.city + " " + address.country;  }
        class Address {
        String street, city, country;
        public void setCity(String city) {
        this.city = city;   }
        public void setCountry(String state) {
        this.country = state;   }
        public void setStreet(String street) {
        this.street = street;   }  }
        public void setUrl(URL url) {   this.url = url;  }
```

Древовидная модель

DOM (Dynamic object model) представляет собой некоторый общий интерфейс для работы со структурой документа. Одна из целей разработки заключалась в том, чтобы код, написанный для работы с каким-либо DOM-анализатором, мог работать и с любым другим DOM-анализатором.

DOM-анализатор строит дерево, которое представляет содержимое XMLдокумента, и определяет набор классов, которые представляют каждый элемент в XML-документе (элементы, атрибуты, сущности, текст и т.д.).

В Java включена поддержка DOM. В пакете org.w3c.dom можно найти интерфейсы, которые представляют вышеуказанные объекты. Реализацией этих интерфейсов занимаются разработчики анализаторов. Разработчики приложений, которые хотят использовать DOM-анализатор, имеют готовый набор методов для манипуляции деревом объектов и не зависят от конкретной реализации используемого анализатора.

Node

Основным объектом DOM является Node – некоторый общий элемент дерева. Большинство DOM-объектов унаследовано именно от Node. Для представления элементов, атрибутов, сущностей разработаны свои специализации Node.

Node определяет ряд методов, которые используются для работы с деревом:

getNodeType() – возвращает тип объекта (элемент, атрибут, текст, CDATA и т.д.);

getParentNode() – возвращает объект, являющийся родителем текущего узла Node;

getChildNodes() – возвращает список объектов, являющихся дочерними элементами;

getFirstChild(), getLastChild() – возвращает первый и последний дочерние элементы;

getAttributes() – возвращает список атрибутов данного элемента.

Attr, Element, Text

Данные интерфейсы унаследованы от интерфейса Node и используются для работы с конкретными объектами дерева.

Document

Используется для получения информации о документе и изменения его структуры. Это интерфейс представляет собой корневой элемент XML-документа и содержит методы доступа ко всему содержимому документа.

В следующем примере производится разбор документа notepad.xml с использованием DOM-анализатора и инциализация на его основе набора объектов. При этом используется анализатор XML4J от IBM.

Пример 4: создание объектов на основе XML:

MyDOMDemo.java

```
import org.w3c.dom.Element;
import org.w3c.dom.Document;
import org.w3c.dom.Node; import org.w3c.dom.NodeList;
import org.w3c.dom.Text;
import org.apache.xerces.parsers.DOMParser;
import java.net.URL;
import java.util.Vector;
public class MyDOMDemo {
public static String getValue(Element e, String name) {
NodeList nList = e.getElementsByTagName(name);
Element elem = (Element) nList.item(0);
Text t = (Text) elem.getFirstChild();
return t.getNodeValue();  }
public static void main(String[] args) {
 Document doc = null;
DOMParser parser = new DOMParser();
 Vector entries = new Vector();
try {
parser.parse("notepad.xml");
doc = parser.getDocument();
Element root = doc.getDocumentElement();
NodeList noteList =  root.getElementsByTagName("note");
Element noteElem;
for (int i = 0; i <noteList.getLength(); i++) {
noteElem = (Element) noteList.item(i);
Note e = new Note();
NodeList list = noteElem.getChildNodes();
Node log =  noteElem.getAttributes().item(0);
e.setLogin(log.getNodeValue());
```

```
e.setName(getValue(noteElem, "name"));
e.setTel(Integer.parseInt(getValue(noteElem, "tel")));
e.setUrl(new URL(getValue(noteElem, "url")));
Element n =  (Element)noteElem.getElementsByTagName("address").item(0);
e.address.setStreet(getValue(n, "street"));
e.address.setCountry(getValue(n, "country"));
e.address.setCity(getValue(n, "city"));
entries.add(e);    }
} catch (Exception e) {
System.out.println(e);   }
for (int i = 0; i < entries.size(); i++)
System.out.println( ((Note) entries.elementAt(i)).toString());  } }
```

XML-документы можно не только читать, но и корректировать. /*

Пример 5: замена информации в файле XML :

```
JDOMChanger.java
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import java.util.*;
import java.io.FileOutputStream;
public class JDOMChanger {
static void lookForElement(String name, String element, String content, String login) {
SAXBuilder builder = new SAXBuilder();
try {
Document document = builder.build(name);
Element root = document.getRootElement();
List c = root.getChildren();
Iterator i = c.iterator();
while (i.hasNext()) {
Element e = (Element) i.next();
if (e.getAttributeValue("login").equals(login)) {
e.getChild(element).setText(content);     }    }
XMLOutputter serializer = new XMLOutputter();
serializer.output(document, new FileOutputStream(name));
System.out.flush();
} catch (Exception e) {
System.out.println(e);   }  }
public static void main(String[] args) {
String name = "notepad.xml";
FieldChanger.lookForElement(name, "tel", "09", "rom");  } }
```

В этом примере использован DOM-анализатор JDOM основанный на идее "if something doesn't work, fix it"

XSL

XML используется для представления информации в виде некоторой структуры, но никоим образом не указывает, как отображать XML-документ. Для того чтобы просмотреть XML-документ, нужно его каким-то образом отформатировать.

Инструкции форматирования XML-документов формируются в так называемые таблицы стилей, и для просмотра XML-документа нужно обработать XML файл согласно этим инструкциям.

Создание XML-документа.

Документы можно не только читать, но также модифицировать и создавать совершенно новые. Для этого необходимо создать объекты классов Document, Element, добавить к последнему атрибуты и текстовое содержимое, после чего присоединить их к объекту, который в дереве XML-документа будет находиться выше. Следующий пример демонстрирует создание XML-документа и запись его в файл. Для записи XML-документа используется класс Transformer.

Создание и запись документа # CreateXmlDocumentMain.java */

```
package by.epam.learn.xml.transform;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
public class CreateXmlDocumentMain {
public static void main(String[] args) {
DocumentBuilderFactory documentBuilderFactory =
DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = null;
try {
documentBuilder = documentBuilderFactory.newDocumentBuilder();
} catch (ParserConfigurationException e) {
e.printStackTrace();    }    // forming a document tree
Document document = documentBuilder.newDocument();
String root = "book";
Element rootElement = document.createElement(root);
document.appendChild(rootElement);
Element elementName = document.createElement("name");
String name = "Java";
elementName.appendChild(document.createTextNode(name));
Element elementAuthor = document.createElement("author");
String author = "Blinov";
elementAuthor.appendChild(document.createTextNode(author));
elementAuthor.setAttribute("id", "777");
rootElement.appendChild(elementName);
rootElement.appendChild(elementAuthor);    // write tree to file
```

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
try {
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(new FileWriter("data_xml/book.xml"));
transformer.transform(source, result);
} catch (TransformerConfigurationException e) {
e.printStackTrace();    }
catch (TransformerException | IOException e) {
e.printStackTrace();    } } }
```

В результате будет создан документ book.xml следующего содержания:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<book>
        <name>Java</name>
        <author id="777">Blinov</author>
</book>
```

Существует два стандарта стилевых таблиц, опубликованных W3C. Это CSS (Cascading Stylesheet) и XLS (XML Stylesheet Language)

CSS изначально разрабатывался для HTML и представляет из себя набор инструкций, которые указывают браузеру, какой шрифт, размер, цвет использовать для отображения элементов HTML-документа.

XSL более современен, чем CSS, потому что используется для преобразования XML-документа перед отображением. Так, используя XSL, можно построить оглавление для XML-документа, представляющего книгу.

Вообще XSL можно разделить на две части: XSLT (XSL Transformation) и XSLFO (XSL Formatting Objects).

Для того чтобы XML-документ преобразовать согласно инструкциям, находящимся в файле таблицы стилей, необходим XSL Processor.

XSLT

Язык для описания преобразований XML-документа. XSLT используется не только для приведения XML-документов к некоторому “читаемому” виду, но и для изменения структуры XML-документа.

К примеру, XSLT можно использовать для:

• добавления новых элементов в XML-документ;

• создания нового XML-документа на основании заданного (список имен адресной книги);

• предоставления информации из XML-документа с разной степенью детализации;

• преобразования XML-документа в документ HTML.

Пусть требуется построить новый HTML-файл на основе файла notepad.xml, который в виде таблицы будет выводить login, name и street для каждой записи, присутствующей в адресной книге. Следует воспользоваться XSLT для решения данной задачи. В следующем коде приведено содержимое файла таблицы стилей, который решает поставленную проблему.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"version="1.0">
<xsl:template match="/">
```

```
<html>
<head>
<title>Notepad Contents</title>
</head>

 <body> <table border="1">
        <tr>
                <th>Login</th>
                <th>Name</th>
                <th>Street</th>
        </tr>
 <xsl:for-each select="notepad/note">
 <tr>
 <td><xsl:value-of select="@login"/></td>
 <td><xsl:value-of select="name"/></td>
 <td><xsl:value-of select="address/street"/></td>
 </tr>
 </xsl:for-each>
 </table>
 </body>
 </html>
 </xsl:template>
 </xsl:stylesheet>
```

Для трансформации одного документа в другой можно использовать, например, следующий код.

Пример 6: трансформация XML в HTML :

SimpleTransform.java

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
public class SimpleTransform {
public static void main(String[] args) {
try {
TransformerFactory tFact =  TransformerFactory.newInstance();
Transformer transformer =  tFact.newTransformer(new StreamSource("notepad.xsl"));
transformer.transform(new StreamSource("notepad.xml"),    new
StreamResult("notepad.html"));
} catch (TransformerException e) {
e.printStackTrace();   }
} }
```

В результате получится HTML-документ следующего вида:

```
<html><head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Notepad Contents</title>
</head>
        <body>
        <table border="1">
<tr>
        <th>Login</th>
```

```
<th>Name</th>
<th>Street</th>
 </tr>
<tr>
<td>rom</td>
<td>Valera Romanchik</td>
<td>Main Str., 35</td>
</tr>
<tr>
<td>goch</td>
<td>Igor Blinov</td>
<td>Deep Forest, 7</td>
</tr>
</table></body></html>
```

Элементы таблицы стилей

Таблица стилей представляет собой well-formed XML-документ. Эта таблица описывает изначальный документ, конечный документ и то, как трансформировать начальный документ в конечный.

Какие же элементы используются в данном листинге?

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

Данная инструкция говорит о том, что конечный документ, который получится после преобразования, будет являться XML-документом.

<xsl:template match="notepad">

<names>

<xsl:apply-templates/>

</names> </xsl:template>

Инструкция <xsl:template...> задает шаблон преобразования. Набор шаблонов преобразования составляет основную часть таблицы стилей. В предыдущем примере приводится шаблон, который преобразует элемент notepad в элемент names.

Шаблон состоит из двух частей:

1) Параметр match, который задает элемент или множество элементов в исходном дереве, к которым будет применяться данный шаблон;

2) Содержимое шаблона, которое будет вставлено в конечный документ.

Нужно отметить, что содержимое параметра math может быть довольно сложным. В предыдущем примере просто ограничились именем элемента. Но, к примеру, следующее содержимое параметра math указывает на то, что шаблон должен применяться к элементу url, содержащему атрибут protocol со значением mailto:

<xsl:template match=”url[@protocol=’mailto’]”>

Кроме этого, существует набор функций, которые также могут использоваться при объявлении шаблона:

<xsl:template match=”chapter[position()=2]”>

Данный шаблон будет применен ко второму по счету элементу chapter исходного документа.

Инструкция <xsl:apply-templates/> сообщает XSL-процессору о том, что нужно перейти к просмотру дочерних элементов.

XSL-процессор работает по следующему алгоритму. После загрузки исходного XML-документа и таблицы стилей процессор просматривает весь документ от корня до листьев. На каждом шагу процессор пытается применить к данному элементу некоторый шаблон преобразования; если в таблице стилей для текущего просматриваемого элемента есть шаблон, процессор вставляет в результирующий документ содержимое этого шаблона. Когда процессор встречает инструкцию <xsl:apply-templates/>, он переходит к дочерним элементам текущего узла и повторяет процесс, т.е. пытается для каждого дочернего элемента найти соответствие в таблице стилей.

Проверка документа

С помощью DTD и схемы XSD можно проверить документ на корректность. Схема XSD представляет собой более строгое описание XML-документа, чем DTD. Для адресной книги XML-схема notepad.xsd выглядит следующим образом.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
        <xs:element name="notepad">
        <xs:complexType>       <xs:sequence>
                <xs:element ref="note" minOccurs='1'
                maxOccurs='unbounded'/>
                </xs:sequence>
        </xs:complexType>
        </xs:element>
<xs:element name="note">
<xs:complexType>
<xs:sequence>
<xs:element ref="name"/>
<xs:element ref="tel" />
<xs:element ref="url" />
<xs:element ref="address"/>
</xs:sequence>
<xs:attribute name="login" type="xs:ID"  use='required'/>
</xs:complexType>
</xs:element>
<xs:element name="name" type="xs:string"/>
<xs:element name="tel" type="xs:int" />
<xs:element name="url" type="xs:anyURI"/>
<xs:element name="street" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="address">
<xs:complexType>
<xs:sequence>
<xs:element ref="street" />
<xs:element ref="city" />
<xs:element ref="country" />
```

```
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Тогда для проверки документа объекту-парсеру следует дать указание использовать DTD и схему XSD и в XML-документ вместо ссылки на DTD добавить к корневому элементу атрибуты вида:

```
<notepad xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:noNamespaceSchemaLocation='notepad.xsd'>
```

Следующий пример выполняет проверку документа.

Пример 7: проверка корректности документа XML :

```
Validation.java
import org.w3c.dom.Document;
import org.apache.xerces.parsers.DOMParser;
public class Validation {
public static void main(String[] args) {
String filename = "notepad.xml";
DOMParser parser = new DOMParser();  //установка обработчика ошибок
parser.setErrorHandler(new MyErrorHandler());
try {  //установка способа проверки с использованием DTD
parser.setFeature( "http://xml.org/sax/features/validation", true);  //установка способа
проверки с использованием XSD
parser.setFeature( "http://apache.org/xml/features/validation/schema", true);
parser.parse(filename);
Document doc = parser.getDocument();
} catch (Exception e) {
System.out.println(e);   }
System.out.print("проверка " + filename  + " завершена");  } }
```

Класс обработчика ошибок может выглядеть следующим образом.

Пример 8: обработчик ошибок:

```
MyErrorHandler.java
import org.xml.sax.ErrorHandler;
import ml.sax.SAXParseE org.x xception;
public class MyErrorHandler implements ErrorHandler {
public void warning(SAXParseException e) {
System.out.println(getLineAddress(e) + " - "  + e.getMessage());  }
public void error(SAXParseException e) {
System.out.println(getLineAddress(e) + " - "  + e.getMessage());  }
public void fatalError(SAXParseException e) {
System.out.println(getLineAddress(e) + " - "  + e.getMessage());  }
private String getLineAddress(SAXParseException e) { //определение строки и столбца
ошибки
return e.getLineNumber() + " : " + e.getColumnNumber();  } }
```

Для того чтобы убедиться в работоспособности кода следует внести в исходный XML-документ ошибку. Например, сделать идентичными значения атрибута login. Тогда в результате запуска на консоль будет выведено следующее сообщение обработчика об ошибке вида:

```
14 : 22 - Datatype error: ID 'goch'  has to be unique. проверка notepad.xml завершена
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Язык разметки XML
2. Для чего необходим DTD?
3. Виды анализаторов.
4. Когда следует использовать DOM-, а когда – SAX-анализаторы?
5. Как можно проверить документ на корректность?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Создать файл XML и соответствующую ему схему XSD.
2. При разработке XSD использовать простые и комплексные типы, перечисления, шаблоны и предельные значения.
3. Создать класс, соответствующий данному описанию.
4. Создать приложение для разбора XML-документа и инициализации коллекции объектов информацией из XML-файла. Для разбора использовать SAX, DOM или StAX-парсеры. Для сортировки объектов использовать интерфейс Comparator.
5. Произвести проверку XML-документа с привлечением XSD.
6. Определить метод, производящий преобразование разработанного XML документа в документ, указанный в каждом задании.

1. Оранжерея.

Растения, содержащиеся в оранжерее, имеют следующие характеристики: – Name – название растения; – Soil – почва для посадки, которая может быть следующих типов: подзолистая, грунтовая, дерново-подзолистая;– Origin – место происхождения растения; – Visual parameters (должно быть несколько) – внешние параметры: цвет стебля, цвет листьев, средний размер растения; – Growing tips (должно быть несколько) – предпочтительные условия произрастания: температура (в градусах), освещение (светолюбиво либо нет), полив (мл в неделю); – Multiplying – размножение: листьями, черенками либо семенами. Корневой элемент назвать Flower. С помощью XSL преобразовать XML-файл в формат HTML.

2. Алмазный фонд.

Драгоценные и полудрагоценные камни, содержащиеся в павильоне, имеют следующие характеристики: – Name – название камня; – Preciousness – может быть драгоценным либо полудрагоценным; – Origin – место добывания; – Visual parameters (должно быть несколько) – могут быть: цвет (зеленый, красный, желтый и т.д.), прозрачность (измеряется в процентах 0–100%), способы огранки (количество граней 4–15); – Value – вес камня (измеряется в каратах). Корневой элемент назвать Gem. С помощью XSL преобразовать XML-файл в формат XML, где корневым элементом будет место происхождения.

3. Тарифы мобильных компаний.

Тарифы мобильных компаний могут иметь следующую структуру: – Name – название тарифа; – Operator name – название сотового оператора, которому принадлежит тариф; – Payroll – абонентская плата в месяц (0–n рублей); – Call prices (должно быть несколько) – цены на звонки: внутри сети (0–n рублей в минуту), вне сети (0–n рублей в минуту), на стационарные телефоны (0–n рублей в минуту); – SMS price – цена за смс (0–n рублей); – Parameters (должно быть несколько) – наличие любимого номера (0–n), тарификация (12-секундная, поминутная), плата за подключение к тарифу (0–n рублей). Корневой элемент назвать Tariff. С помощью XSL преобразовать XML-файл в формат HTML.

4. Лекарственные препараты.

Лекарственные препараты имеют следующие характеристики: – Name – наименование препарата; – Pharm – фирма-производитель; – Group – группа препаратов, к которым относится лекарство (антибиотики, болеутоляющие, витамины и т.п.); – Analogs (может быть несколько) – содержит наименование аналога; – Versions – варианты исполнения (консистенция/вид: таблетки, капсулы, порошок, капли и т.п.). Для каждого варианта исполнения может быть несколько производителей лекарственных препаратов со следующими характеристиками: Certificate – свидетельство о регистрации препарата (номер, даты выдачи/истечения действия, регистрирующая организация); Package – упаковка (тип упаковки, количество в упаковке, цена за упаковку); Dosage – дозировка препарата, периодичность приема; Корневой элемент назвать Medicine. С помощью XSL преобразовать XML-файл в формат HTML.

5. Компьютеры.

Компьютерные комплектующие имеют следующие характеристики: – Name – название комплектующего; – Origin – страна производства; – Price – цена (0–n рублей); – Type (должно быть несколько) – периферийное либо нет, энергопотребление (ватт), наличие кулера (есть либо нет), группа комплектующих (устройства ввода-вывода, мультимедийные), порты (COM, USB, LPT); – Critical – критично ли наличие комплектующего для работы компьютера. Корневой элемент назвать Device. С помощью XSL преобразовать XML-файл в формат XML, при выводе корневым элементом сделать Critical.

6. Электроинструменты.

Электроинструменты можно структурировать по следующей схеме: – Model – название модели; – Handy – одно- или двуручное; – Origin – страна производства; – TC (должно быть несколько) – технические характеристики: энергопотребление (низкое, среднее, высокое), производительность (в единицах в час), возможность автономного функционирования и т.д.; – Material – материал изготовления. Корневой элемент назвать PowerTools или Power. С помощью XSL преобразовать XML-файл в формат XML, при выводе корневым элементом сделать страну производства.

7. Столовые приборы.

Столовые приборы можно структурировать по следующей схеме:– Type – тип (нож, вилка, ложка и т.д.); – Origin – страна производства; – Visual (должно быть несколько) – визуальные характеристики: лезвие, зубец (длина лезвия, зубца [10–n см], ширина лезвия [10–n мм]), материал (лезвие [сталь, чугун, медь и т.д.]), рукоять (деревянная [если да, то указать тип дерева], пластик, металл); – Value – коллекционный либо нет. Корневой элемент назвать FlatWare. С помощью XSL преобразовать XML-файл в формат HTML.

8. Самолеты.

Самолеты можно описать по следующей схеме: – Model – название модели; – Origin – страна производства; – Chars (должно быть несколько) – характеристики, могут быть следующими: тип (пассажирский, грузовой, почтовый, пожарный, сельскохозяйственный), количество мест для экипажа, характеристики (грузоподъемность, число пассажиров), наличие радара; – Parameters – длина (в метрах), ширина (в метрах), высота (в метрах); – Price – цена (в талерах). Корневой элемент назвать Plane. С помощью XSL преобразовать XML-файл в формат HTML.

9. Конфеты.

Схема: – Name – название конфеты; – Energy – калорийность (ккал); – Type (должно быть несколько) – тип конфеты (карамель, ирис, шоколадная [с начинкой либо нет]); – Ingredients (должно быть несколько) – ингредиенты: вода, сахар (в мг), фруктоза (в мг), тип шоколада (для шоколадных), ванилин (в мг); – Value – пищевая ценность: белки (в г), жиры (в г) и углеводы (в г); – Production – предприятие-изготовитель. Корневой элемент назвать Candy. С помощью XSL преобразовать XML-файл в формат HTML.

10. Периодические издания.

Схема: – Title – название; – Type – тип (газета, журнал, буклет); – Monthly – переодичность выхода; – Chars (должно быть несколько) – характеристики: цветность (да либо нет), объем (n страниц), глянцевое (да [только для журналов и буклетов] либо нет [для газет]), подписной индекс (только для газет и журналов). Корневой элемент назвать Paper. С помощью XSL преобразовать XML-файл в HTML с выводом информации в табличном виде.

11. Туристические путевки.

Туристические путевки, предлагаемые агентством, имеют следующие характеристики: – Type – тип (выходного дня, экскурсионная, отдых, паломничество и т.д.); – Country – страна, выбранная для путешествия; – Number days/nights – количество дней и ночей; – Transport – вид перевозки туристов (авиа, ж/д, авто, лайнер); – Hotel characteristic (должно быть несколько) – количество звезд, включено ли питание и какое (HB, BB, Al), какой номер (1-, 2-, 3-местные), есть ли телевизор, кондиционер и т.д.; – Cost – стоимость путевки (сколько и что включено). Корневой элемент назвать Tourist voucher. С помощью XSL преобразовать XML-файл в формат HTML, с выводом информации в табличном виде.

12. Старые открытки.

Схема: – Thema – тема изображения (городской пейзаж, природа, люди, религия, спорт, архитектура...); – Type – тип (поздравительная, рекламная, обычная). Была ли отправлена; – Country – страна производства; – Year – год издания; – Author – имя автора/ов (если известен); – Valuable – историческая, коллекционная или тематическая ценность. Корневой элемент назвать Old Card. С помощью XSL преобразовать XML-файл в HTML, с выводом информации в табличном виде.

13.Банковские вклады.

Схема: – Name – название банка; – Country – страна регистрации; – Type – тип вклада (до востребования, срочный, расчетный, накопительный, сберегательный, металлический); – Depositor – имя вкладчика; – Account id – номер счета; – Amount on deposit – сумма вклада; – Profitability – годовой процент; – Time constraints – срок вклада. Корневой элемент назвать Bank. С помощью XSL преобразовать XML-файл в формат HTML с выводом информации в табличном виде.

ДОМАШНЕЕ ЗАДАНИЕ

Индивидуальное задание

ЛИТЕРАТУРА

И. Н. Блинов В. С. Романчик, Java, Четыре четверти, 2020.

Преподаватель А.С.Кибисова

Рассмотрено на заседании цикловой комиссии программного обеспечения информационных технологий
Протокол № ____ от «___» _____ 2021
Председатель ЦК _____В.Ю.Михалевич