Лабораторная №4

Тема: «XML, работа с SAX и DOM парсерами»

Цель

Целью настоящей лабораторной работы является освоение студентами навыков работы с данными в формате XML средствами языка Java.

Задание

В соответствии с индивидуальным вариантом задания, полученным от преподавателя, требуется:

- определить схему XML-файла;
- определить структуры данных для представления в памяти содержимого XML-файла после выполнения разбора (если такое представление необходимо по заданию);
- выбрать способ разбора (с создание документа или без такового);
- реализовать программу на языке программирования Java

Содержание отчета

Отчет о выполнении работы должен включать в себя:

- титульный лист;
- вариант задания;
- описание схемы XML-файла;
- исходный текст программы (или его часть);
- пример работы приложения XML-файла (продемонстрировать)

Теория

Структура XML-документа

XML-документ — это обычный текст, отформатированный в соответствии с правилами языка XML. Для создания XML-документа можно использовать любой текстовый редактор, например, Блокнот.

Главным отличием XML-документ от обычного текста является наличие в нем явно заданной структуры, образованной элементами. Элементы могут быть вкладываться один в другой, причем на самом верхнем уровне всегда находится только один элемент, называемый элементом документа. Таким образом, элементы XML-документа образуют дерево.

Кроме положения в дереве каждый элемент XML-документа характеризуется именем. Имя элемента является произвольной последовательностью букв, цифр, точек, дефисов и знаков подчеркивания, однако первым символом имени элемента может быть только буква или знак подчеркивания. Имена элементов чувствительны к регистру.

Начало и конец каждого элемента отмечается в тексте XML-документа соответственно

начальным и конечным тегом. Начальный тег представляет собой имя, заключенное в угловые скобки, конечный тег отличается от начального тем, что между открывающейся угловой скобкой и именем элемента ставиться наклонная черта. Имена, указанные в начальном и конечном теге элемента должны совпадать. Например:

```
<library>
    <book>
        <title>Проектирование цифровых схем на VHDL</title>
        <author>Е. А. Суворова</author>
        <author>Ю. Е. Шейнин</author>
    </book>
</library>
```

Если элемент не имеет содержимого, то вместо пары начального и конечного тегов может использоваться единственный пустой тег. Пустой тег отличается от начального только тем, что перед закрывающейся угловой скобкой ставится наклонная черта. Например:

```
<library/>
```

Помимо имени элемент может обладать набором атрибутов. Каждый атрибут представляет собой пару, состоящую из имени и значения. Атрибуты записываются в начальном или в пустом теге между именем элемента и закрывающейся угловой скобкой (в конечном теге атрибуты не пишутся). Имя атрибута подчиняется тем же правилам, что и имя элемента. Значение атрибута заключается в одинарные или двойные кавычки и отделяется от имени атрибута знаком равно. Например:

```
<book isbn="5-94157-189-5">
    <!-- ... -->
</book>
```

Помимо тегов в исходном тексте XML-документа могут встречаться ссылки. Ссылки можно употреблять как в обычном тексте, так и в значениях атрибутов. Различают сущностные и символьные ссылки.

Сущностная ссылка представляет собой имя сущности, перед которой ставится амперсанд, а после — точка с запятой. Имена сущностей подчиняются тем же правилами, что и имена элементов и атрибутов.

В любом XML-документе всегда доступно пять встроенных сущностей: amp (амперсанд), lt (открывающаяся угловая скобка), gt (закрывающаяся угловая скобка), apos (одиночная кавычка) и quot (двойная кавычка). Встроенные ссылки употребляются тогда, когда в обычный текст или в значение атрибута необходимо включить амперсанд, угловую скобку или кавычки. Например:

```
7 &lt; 10
```

Символьная ссылка начинается амперсандом и решетки, после которых следует десятичный код символа либо буква «x» и шестнадцатеричный код символа. Заканчивается символьная ссылка также как и сущностная — точкой с запятой. Например:

```
&#x41c;&#x438;&#x440;
```

В тексте XML-документа могут встречаться комментарии. Как и в обычных языках

программирования, комментарии предназначены для придания ясности тексту и обычно пропускаются программой, обрабатывающей XML-документ. Комментарий начинается открывающейся угловой скобкой, за которой следует восклицательный знак и два дефиса, а заканчивается двумя дефисами, за которыми следует закрывающаяся скобка. Приведем пример комментария:

```
<!-- это комментарий -->
```

Пространства имен используются для того, чтобы предотвратить конфликты имен элементов и атрибутов. Одноименные элементы и атрибуты, принадлежащие разным пространствам имен, считаются разными. Каждое пространство имен характеризуется идентификатором, который, в отличие от имен элементов и атрибутов, может состоять из любых символов. Для того чтобы гарантировать глобальную уникальность в качестве идентификатора пространства имен принято использовать идентификатор ресурса (URI — Uniform Resource Identifier).

Поскольку идентификатор пространства имен обычно состоит из довольно большого количества символов, необходимость записывать его вместе с каждым элементом и атрибутом сделала бы текст XML-документа слишком длинным и трудным для понимания и редактирования. Поэтому в тексте XML-документа каждому пространству имен сопоставляют короткий префикс, который и записывают в начале имен элементов и атрибутов, отделяя его от остальной части имени двоеточием. Имя элемента и атрибута содержащее префикс пространства имен называется квалифицированным именем, а часть имени без префикса — локальным именем.

Сопоставление префикса пространству имен осуществляется с помощью специального атрибута. Значением этого атрибута должен быть идентификатор пространства имен, в том время, как имя должно включать предопределенный префикс xmlns, а в качестве локального имени должен использовать сопоставляемый пространству имен префикс. Область действия префикса распространяется на как элемент, в котором этот префикс объявлен, так и на все вложенные в него элементы. Во вложенных элементах префиксы можно переопределить. В одном элементе можно объявить несколько префиксов. Например:

```
<l:library xmlns:l="http://www.kstu.kursk.ru/library/"
xmlns:html="http://www.w3.org/TR/html4/">
        <l:book isbn="5-94157-189-5">
        <l:title>Проектирование цифровых схем на VHDL</l:title>
        <l:author>Е. А. Суворова</l:author>
        <l:author>Ю. Е. Шейнин</l:author>
        <l:description>
            <html:p>
                В книге рассматривается язык <html:i>VHDL</html:i>
            </html:p>
            <html:p>
                Приводится описание следующих популярных САПР
                <html:ul>
                    <html:li>OrCAD Express</html:li>
                    <html:li>Xilinx Foundation Express</html:li>
                </html:ul>
            </html:p>
        </l:description>
    </l:book>
        <!-- другие книги -->
```

```
</l:library>
```

Если имя какого-либо элемента или атрибута не содержит префикса, то считается, что оно принадлежит пространству имен по умолчанию. Задание пространства имен по умолчанию осуществляется с помощью атрибута с именем xmlns (в данном случае xmlns — это имя, а не префикс). Значением этого атрибута должен быть идентификатор пространства имен. Во вложенных элементах пространство имен по умолчанию можно изменить.

```
<l:library xmlns:l="http://www.kstu.kursk.ru/library/"
xmlns="http://www.w3.org/TR/html4/">
<l:book isbn="5-94157-189-5">
<l:title>Проектирование цифровых схем на VHDL</l:title>
<l:author>Е. А. Суворова</l:author>
<l:author>Ю. Е. Шейнин</l:author>
<l:description>
<p>
    В книге рассматривается язык <i>VHDL</i>
</p>
<p>
    Приводится описание следующих популярных САПР
    <ul>
        <li>OrCAD Express</li>
        <li>Xilinx Foundation Express</li>
    </ul>
</p>
</l:description>
</l:book>
<!-- другие книги -->
</l:library>
```

В самом начале XML-файла может размещаться декларация версии XML-документа и использованной кодировки. Декларация напоминает тег элемента, но отличается тем, что вместо имени элемента используется ключевое слово xml, а после открывающейся скобки и перед закрывающейся скобкой ставятся вопросительные знаки. Версия и кодировка документа задаются «атрибутами» version и encoding соответственно. «Атрибут» version обязателен и должен быть первым. Единственной возможной версией является 1.0. «Атрибут» encoding не обязателен. Если он отсутствует или если декларации вовсе нет, то считается, что текст документа использует кодировку UTF-7. Напомним, что тексты на русском языке обычно используют кодировку windows-1251 либо koi8-r. Например:

```
<?xml version="1.0" encoding="windows-1251"?>
```

Разбор XML-документа

Синтаксический разбор текста XML-документа в стандартной библиотеке Java может выполняться в одном из двух режимов.

Первый режим характеризуется тем, что результатом разбора является объект документа, в котором в иерархическом виде представлены все данные XML-документа. Этот объект используется затем прикладной программой для выбора необходимых ей данных.

Второй режим отличается от первого тем, что объект документа не создается, вместо этого в процессе разбора, по мере того как в тексте XML-документа встречаются те или иные

конструкции (простой текст, теги элементов, комментарии и т. п.), вызывается соответствующий код прикладной программы, которому передается описание встретившейся конструкции. Вызываемый код может сохранять получаемые им данные в какой-либо структуре данных либо сразу же их обрабатывать. Очевидно, что второй подход требует от разработчика прикладной программы сравнительно больших усилий, однако обладает тем преимуществом, что позволяет избежать хранения в памяти всех данных XML-документа, причем, если какие-либо данные XML-документа хранить все же необходимо, то для этого может быть использована структура данных наилучшим образом соответствующая специфике прикладной программы.

Теперь рассмотрим каждый из режимов разбора XML-документа подробнее.

Разбор с созданием объекта документа

Разбор с созданием объекта документа осуществляется построителем, для создания которого, в свою очередь, используется фабрика построителей. Фабрика построителей представлена абстрактным классом DocumentBuilderFactory, который находится в пакете javax.xml.parsers. Рассмотрим методы фабрики:

- newInstance() — создает фабрику. Этот метод является статическим;
- setNamespaceAware(f) — включает или выключает поддержку пространств имен;
- newDocumentBuilder() — создает новый построитель. В случае ошибки выбрасывает исключение ParserConfigurationException.

Построитель представлен абстрактным классом DocumentBuilder, который также находится в пакете javax.xml.parsers. Рассмотрим методы построителя:

- parse(s) — разбирает текст XML-документа и создает в процессе этого разбора объект документа. Параметр s может быть строкой, именем файла либо входным байтовым потоком. Если параметр s — строка, то она интерпретируется как универсальный идентификатор ресурса. В случае ошибки ввода-вывода выбрасывает исключение IOException, а в остальных случаях — исключение SAXException;
- newDocument() — создает пустой объект документа.

Приведем разбора XML-документа с созданием объекта документа:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
factory.setIgnoringComments(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse("file://c:/data/test.xml");
```

Объект документа представлен интерфейсом Document, находящимся в пакете org.w3c.dom. В этом же пакете находятся все остальные интерфейсы, используемые для работы с объектом документа. Интерфейс Document наследован от интерфейса Node, который служит базовым для всех интерфейсов, представляющих узлы дерева документа. Помимо уже упомянутого интерфейса Document, использующегося в качестве корня дерева документа, от интерфейса Node наследованы интерфейсы представляющие элементы (интерфейс Element), текст (интерфейс Text), комментарии (интерфейс Comment) и т. д.

Сначала рассмотрим некоторые методы интерфейса Node (т. е. методы, которые являются общими для всех узлов дерева документа):

- getNodeName() — возвращает имя узла. Если данный узел является элементом, то метод возвращает имя элемента, а если атрибутом — имя атрибута. В большинстве остальных случаев метод
- getNodeName возвращает фиксированную строку, определяющуюся типом узла;
- getLocalName(), getPrefix() и getNamespaceURI() — если данный узел является элементом

или атрибутом, и если включена поддержка пространств имен, возвращают локальную часть имени узла, префикс пространства имен или идентификатор пространства имен соответственно. В противном случае возвращают null;

- getNodeValue() — возвращает значение узла. Если данный узел является текстом или комментарием, то метод возвращает соответствующий текст, а если атрибутом — значение атрибута. В большинстве остальных случаев метод getNodeValue() возвращает null;
- setNodeValue(v) — устанавливает значение узла равным v. Этот метод можно вызывать только если данный узел может иметь значение (т. е. является, например, атрибутом или текстом);
- getAttributes() — если данный узел является элементом, возвращает список его атрибутов. Во всех остальных случаях возвращает null;
- getNodeType() — возвращает короткое целое, представляющее тип узла. Возможные типы узлов представлены в интерфейсе Node следующими константами: DOCUMENT_NODE (документ), ELEMENT_NODE (элемента), TEXT_NODE (текст), COMMENT_NODE (комментарий) и т. д.;
- getFirstChild() и getLastChild() — возвращают соответственно первый и последний дочерний узел;
- getNextSibling() и getPreviousSibling() — возвращают соответственно следующий и предыдущий узел;
- getParentNode() — возвращает родительский узел;
- getChildNodes() — возвращает список всех дочерних узлов;
- appendChild(n) — добавляет новый дочерний узел n (узел n становится последним дочерним узлом данного узла);
- insertBefore(n, m) — вставляет новый дочерний узел n перед существующим дочерним узлом m;
- removeChild(n) — удаляется дочерний узел n;
- replaceChild(n, m) — заменяет существующий дочерний узел m новым дочерним узлом n;
- isEqualNode(n) — возвращает истину если данный узел и узел n одинаковы. Два узла называются одинаковыми, если они одинакового типа, их имена и значения равны, и они имеют одинаковые атрибуты и дочерние узлы;
- getTextContent() — возвращает одной строкой весь текст находящийся в данном узле либо во всех его дочерних узлах (дочерние узлы просматриваются рекурсивно);
- getOwnerDocument() — возвращает документ, которому принадлежит данный узел.

Пример перечисления дочерних узлов:

```
for (Node n = node.getFirstChild(); n != null; n = n.getNextSibling()) {
    System.out.println(n.getNodeName());
}
```

Список дочерних узлов, возвращаемый методом getChildNodes, представлен интерфейсом NodeList. Это интерфейс имеет только два метода:

- item(i) — возвращает i-ый дочерний узел (индекс первого узла равен 0);
- getLength() — возвращает количество дочерних узлов.

Список именованных узлов, возвращаемый методом getAttributes, представлен интерфейсов NamedNodeMap. Этот интерфейс имеет следующие методы:

- getNamedItem(n) — возвращает узел с именем n или null, если узла с таким именем в списке нет;

- removeNamedItem(n) — удаляет из списка узел с именем n;
- setNamedItem(d) — добавляет в список узел d;
- getNamedItemNS(u, n), removeNamedItemNS(u, n) и
- setNamedItemNS(d) — аналогичны методам getNamedItem, removeNamedItem и setNamedItem соответственно, но используются в том случае, если включена поддержка пространств имен. Строка u — задает идентификатор пространства имен, а строка n — локальное имя;
- item(i) и getLength() — аналогичны одноименным методам интерфейса NodeList (однако, интерфейс NamedNodeMap не наследован от интерфейса NodeList).

Перейдем теперь к рассмотрению некоторых интерфейсов, наследованных от интерфейса Node. Сначала рассмотрим методы интерфейса Element:

- getTagName() — возвращает имя данного элемента;
- hasAttribute(n) — возвращает истину, если данный элемент имеет атрибут с именем n;
- getAttribute(n) — возвращает значение атрибута, имеющего имя n или null если данный элемент не имеет такого атрибута;
- removeAttribute(n) — удаляет атрибут с именем n;
- setAttribute(n, v) — присваивает атрибуту с именем n значение v;
- hasAttributeNS(u, n), getAttributeNS(u, n), removeAttributeNS(u, n) и setAttributeNS(u, n, v) — аналогичны методам hasAttribute, getAttribute, removeAttribute и setAttribute. Во всех методах u является идентификатором пространства имен. В методе setAttribute имя n должно быть квалифицированным (т. е. с префиксом), а в остальных методах — локальным;
- getElementsByTagName(n) — возвращает список дочерних элементов с именем n. Если вместо имени задана звездочка (*), то метод возвращает список всех дочерних элементов. Метод
- getElementsByTagName рекурсивно обходит дочерние элементы;
- getElementsByTagNameNS(u, n) — аналогичен методу
- getElementsByTagName, но используется в том случае, когда включена поддержка пространств имен. Звездочку можно указывать как вместо идентификатора пространства имен, так и вместо локального имени.

Теперь рассмотрим методы интерфейса Document:

- getDocumentElement() — возвращает элемент документа (напомним, что это тот элемент, в который вложены все остальные элементы);
- getElementsByTagName(n) и getElementsByTagNameNS(u, n) — то же, что в интерфейсе Element;
- createElement(n) — создает новый элемент с именем n. Для добавления созданного элемента в дерево узлов надо использовать метод addChild либо метод insertBefore;
- createElementNS(u, n) — аналогичен createElement, но используется тогда, когда включена поддержка пространств имен;
- createTextNode(s) и createComment(s) — создают новый текстовый узел или новый комментарий. Строка s задает текст нового узла. Для добавления созданного узла в дерево узлов надо использовать метод addChild либо метод insertBefore.

Интерфейсы Text и Comment, представляющие соответственно текстовый узел и комментарий наследованы не непосредственно от интерфейса Node, а от интерфейса CharacterData (который, в свою очередь, наследован он Node). Рассмотрим методы интерфейса CharacterData:

- getData() — возвращает весь текст, находящийся в данном узле;

- substringData(i, n) — возвращает n символов текста начиная с i-ой позиции;
- getLength() — возвращает длину текста;
- appendData(s) — добавляет к тексту строку s;
- insertData(i, s) — вставляет в текст строку s начиная с i-ой позиции;
- deleteData(i, n) — удаляет n символов начиная с i-ой позиции;
- replaceData(i, n, s) — заменяет n символов начиная с i-ой позиции строкой s.

Теперь рассмотрим следующий пример:

```
Document doc = builder.parse("file:///c:/data/test.xml");
Element lib = doc.getDocumentElement();
if ("library".equals(lib.getTagName())) {
    NodeList books = lib.getElementsByTagName("book");
    for (int i = 0; i < books.getLength(); ++i) {
        Element book = (Element) books.item(i);
        String isbn = book.getAttribute("isbn");
        String title = null;
        List<String> authors = new ArrayList<>();
        NodeList props = book.getElementsByTagName("*");
        for (int j = 0; j < props.getLength(); ++j) {
            Element prop = (Element) props.item(j);
            if ("title".equals(prop.getTagName())) {
                title = prop.getTextContent();
            } else if ("author".equals(prop.getTagName())) {
                authors.add(prop.getTextContent());
            }
        }
        if (title != null) {
            System.out.println(title);
            if (isbn != null) {
                System.out.println(isbn);
            }
            for (int j = 0; j < authors.size(); ++j) {
                System.out.println(authors.get(j));
            }
        }
    }
}
```

Разбор без создания объекта документа

Разбор без создания объекта документа осуществляется разборщиком, для создания которого, аналогично случаю с построителем, используется фабрика разборщиков. Фабрика разборщиков представлена абстрактным классом SAXParserFactory, который находится в пакете javax.xml.parsers. Рассмотрим методы фабрики:

- newInstance() — создает фабрику. Этот метод является статическим;
- setNamespaceAware(f) — аналогичен одноименному методу фабрики построителей;
- newSAXParser() — создает новый разборщик. В случае ошибки выбрасывает исключение ParserConfigurationException.

Разборщик представлен абстрактным классом SAXParser, который также находится в пакете javax.xml.parsers. Рассмотрим методы разборщика:

- parse(s, h) — разбирает текст XML-документа, вызывая в процессе разбора обработчик h.

Параметр s может быть строкой, именем файла либо входным байтовым потоком. Если параметр s — строка, то она интерпретируется как универсальный идентификатор ресурса. В случае ошибки ввода-вывода выбрасывает исключение IOException, а в остальных случаях — исключение SAXException.

Приведем пример разбора XML-документа без создания объекта документа:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
SAXParser parser = factory.newSAXParser();
parser.parse("file://c:/data/test.xml", myHandler);
```

Обработчик XML-документа должен быть потомком класса DefaultHandler, находящегося в пакете org.xml.sax.helpers. Класс DefaultHandler, в свою очередь, реализуют интерфейс ContentHandler, находящийся в пакете org.xml.sax. В пользовательском обработчике нужно заново заместить один или несколько методов, определенных в интерфейсе ContentHandler и замещенных по умолчанию в классе DefaultHandler. Рассмотрим некоторые методы интерфейса ContentHandler:

- startElement(u, m, n, a) — вызывается, когда встретился начальный тег элемента. Строки u, m и n задают соответственно идентификатор пространства имен, локальное имя и квалифицированное имя элемента. Если поддержка пространств имен не была включена, то строки u и m — пустые. Параметр a представляет список атрибутов элемента;
- endElement(u, m, n) — вызывается, когда встретился конечный тег элемента. Параметры u, m и n такие же, как в методе startElement;
- characters(c, i, n) — вызывается, когда встретился текст. Символы текста находится в отрезке массиве c. Этот отрезок начинается с i-го символа и имеет длину равную n;
- startDocument() — вызывается перед началом разбора документа;
- endDocument() — вызывается после окончания разбора документа.

Все методы, определенные в интерфейсе ContentHandler, могут выбрасывать исключение SAXException.

Список атрибутов, передаваемый в метод startElement представлен интерфейсом Attributes, находящимся в пакете org.xml.sax. Рассмотрим методы этого интерфейса:

- getValue(u, m | n | i) — возвращает значение атрибута заданного идентификатором пространства имен u и локальным именем m, квалифицированным именем n либо индексом i;
- getURI(i), getLocalName(i) и getQName(i) — возвращают соответственно идентификатор пространства имен, локальное имя и квалифицированное имя i-го атрибута;
- getLength() — возвращает количество атрибутов;
- getIndex(u, m | n) — возвращает индекс атрибута заданного идентификатором пространства имен u и локальным именем m либо квалифицированным именем n. Индекс атрибута в списке атрибутов может не совпадать с положением атрибута в тексте документа.

Приведем теперь пример определения класса обработчика:

```
class MyHandler extends DefaultHandler {
    private StringBuilder text = new StringBuilder();
    private String isbn;
    private String title;
    private List<String> authors = new ArrayList<>();
    public void startElement(String u, String m, String n, Attributes a) {
```

```
            if ("book".equals(n)) {
                isbn = a.getValue("isbn");
            } else if ("title".equals(n) || "author".equals(n)) {
                text.setLength(0);
            }
        }
        public void endElement(String u, String m, String n) {
            if ("book".equals(n)) {
                if (title != null) {
                    System.out.println(title.toUpperCase());
                    if (isbn != null) {
                        System.out.println(isbn);
                    }
                    for (int i = 0; i < authors.size(); ++i) {
                        System.out.println(authors.get(i));
                    }
                }
                title = null;
                authors.clear();
            } else if ("title".equals(n)) {
                title = text.toString();
            } else if ("author".equals(n)) {
                authors.add(text.toString());
            }
        }
        public void characters(char c[], int i, int n) {
            text.append(c, i, n);
        }
}
```