МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационных систем и технологий

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О. Г. Локтионова
«____»_____ 2013 г.

Разбор XML средствами языка Java

Методические указания по выполнению лабораторной работы по курсу «Управление данными» и «Структуры и алгоритмы обработки данных» для студентов, обучающихся по направлениям подготовки 230400.62 «Информационные системы и технологии» и 231000.62 «Программная инженерия»

Курск 2013

УДК 004.42, 004.75

Составители Е. А. Титенко, М. В. Бородин

Рецензент
Доктор технических наук, профессор О. И. Атакищев

Разбор XML средствами языка Java : методические указания по выполнению лабораторной работы / Юго-Зап. гос. ун-т; сост.: Е. А. Титенко, М. В. Бородин Курск, 2013. 36 с.: ил. 1, библиогр.: 2.

В методических указаниях описывается технология разбора XML средствами языка Java. Описание сопровождается примерами. Приводится список контрольных вопросов и вариантов заданий. Предназначены для студентов направления подготовки 230400.62 и 231000.62.

Текст печатается в авторской редакции

Подписано в печать . Формат 60 × 84 1/16.
Усл. печ. л. . Уч.-изд. л. . Тираж 100 экз. Заказ . Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94

1. Цель

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

2. 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.

3. Содержание отчета

Отчет о выполнении работы должен включать в себя:

- титульный лист;
- вариант задания;
- описание схемы XML-файла (приветствуется описание с использованием XML Schema или RELAX NG, однако неформальное описание также вполне допустимо);
- исходный текст программы;
- пример XML-файла для тестирования.

4. Теоретические сведения

4.1. 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 эле-

мент, 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 гаран-

тировать 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"?>
```

4.2. 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.

4.2.1. 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));
      }
    }
  }
}
```

4.3. 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*,[1] 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*.

---

[1] SAX означает Simple API for XML (простой интерфейс прикладного программирования для XML).

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 с. 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);
  }
}
```

5. Практический пример

Предположим, что в формате XML хранится описание векторного изображения, причем допустимы следующие элементы:

• rect — прямоугольник, заданный положением верхней (t), правой (r), нижней (b) и левой (l) границ;

• poly — произвольный многоугольник, каждая из вершин которого представлена вложенным элементом p, задающим ее координаты (x и y);

• text — однострочный текст, начинающийся в некоторой точке (x и y). С помощью вложенных элементов допустимы следующие варианты начертания: полужирной (b) и курсивное (i);

• tranls — сдвиг содержимого элемента на заданное расстояние (dx и dy);

• rot — поворот содержимого элемента вокруг заданной точки (x и y) на заданный угол (angle).

Также для элементов rect, poly и text допустимо задавать цвет в виде тройки чисел (color).

Иными словами, пусть, например, имеется следующее изображение (рис. 1.).

Рис. 1. Пример векторного изображения

С использованием вышеупомянутых элементов такое изображение может быть задано следующим образом:

```
<pic xmlns="urn:picture-schema" h="300" w="300">
  <transl dx="150" dy="150">
    <rect l="-100" t="-100" r="100" b="100"
          color="0.2 0.4 1"/>
    <rot x="0" y="0" angle="-15">
        <poly color="1 0.8 0.4">
            <p x="0" y="-70"/>
            <p x="-60" y="0"/>
            <p x="60" y="0"/>
        </poly>
    <text x="-80" y="50">Hello <b>World!</b></text>
    </rot>
  </transl>
</pic>
```

5.1. Исходный текст

Далее представим исходный текст на языке Java, содержащий определения структур данных, позволяющих представить в памяти изображения, соответствующие вышерассмотренному описанию, а также подпрограммы, предназначенные для создания структур данных на основе анализа XML-документа. Чтобы при обработке созданных структур данных не приходилось бы постоянно прибегать к динамической проверке и преобразованию типа будем использовать шаблон проектирования Посетитель (Visitor).

5.1.1. Файл Picture.java

```
package picture;
import java.util.Collection;
// векторное изображение
public final class Picture implements DrawableContainer {
  private final float width;
  private final float height;
  private final DrawableContainerMixin drawableContainer;
  public Picture(float width, float height,
          Collection<Drawable> drawables) {
    this.width = width;
    this.height = height;
    drawableContainer = new DrawableContainerMixin(
            drawables);
  }
  public float getWidth() {
    return width;
  }
  public float getHeight() {
```

```
    return height;
  }
  @Override
  public int getDrawableCount() {
    return drawableContainer.getDrawableCount();
  }
  @Override
  public Drawable getDrawable(int index) {
    return drawableContainer.getDrawable(index);
  }
}
```

5.1.2. Файл Drawable.java

```
package picture;
// общий предок всех  классов,
// представляющих элементы изображения
public interface Drawable {
  <R> R accept(DrawableVisitor<R> visitor);
}
```

5.1.3. Файл DrawableVisitor.java

```
package picture;
// посетитель для интерфейса Drawable
public interface DrawableVisitor<R> {
  R visitPrimitive(Primitive primitive);
  R visitGroup(Group group);
}
```

5.1.4. Файл DrawableContainer.java

```
package picture;
// предок всех классов,
// могущих содержать в себе элементы изображения
public interface DrawableContainer {
  int getDrawableCount();
  Drawable getDrawable(int index);
}
```

5.1.5. Файл DrawableContainerMixin.java

```
package picture;
import java.util.Collection;
// реализация интерфейса DrawableContainer
final class DrawableContainerMixin {
  private final Drawable[] drawables;
  DrawableContainerMixin(Collection<Drawable> drawables) {
    this.drawables = drawables.toArray(
```

```
            new Drawable[drawables.size()]);
  }
  int getDrawableCount() {
    return drawables.length;
  }
  Drawable getDrawable(int index) {
    return drawables[index];
  }
}
```

5.1.6. Файл Primitive.java

```
package picture;
// предок классов, представляющих прямоугольники,
// многоугольники и строки текста
public abstract class Primitive implements Drawable {
  private final Color color;
  public Primitive(Color color) {
    this.color = color;
  }
  public Color getColor() {
    return color;
  }
  public abstract <R> R accept(
          PrimitiveVisitor<R> visitor);
  @Override
  public <R> R accept(DrawableVisitor<R> visitor) {
    return visitor.visitPrimitive(this);
  }
}
```

5.1.7. Файл PrimitiveVisitor.java

```
package picture;
// посетитель для класса Primitive
public interface PrimitiveVisitor<R> {
  R visitRectangle(Rectangle rectangle);
  R visitPolygon(Polygon polygon);
  R visitText(Text text);
}
```

5.1.8. Файл Rectangle.java

```
package picture;
// прямоугольник
public final class Rectangle extends Primitive {
  private final float top;
  private final float right;
  private final float bottom;
```

```
  private final float left;
  public Rectangle(Color color, float top, float right,
          float bottom, float left) {
    super(color);
    this.top = top;
    this.right = right;
    this.bottom = bottom;
    this.left = left;
  }
  public float getTop() {
    return top;
  }
  public float getRight() {
    return right;
  }
  public float getBottom() {
    return bottom;
  }
  public float getLeft() {
    return left;
  }
  @Override
  public <R> R accept(PrimitiveVisitor<R> visitor) {
    return visitor.visitRectangle(this);
  }
}
```

5.1.9. Файл Polygon.java

```
package picture;
import java.util.Collection;
// многоугольник
public final class Polygon extends Primitive {
  private final Point[] points;
  public Polygon(Color color, Collection<Point> points) {
    super(color);
    this.points = points.toArray(new Point[points.size()]);
  }
  public int getPointCount() {
    return points.length;
  }
  public Point getPoint(int index) {
    return points[index];
  }
  @Override
  public <R> R accept(PrimitiveVisitor<R> visitor) {
    return visitor.visitPolygon(this);
```

```
  }
}
```

5.1.10. Файл Point.java

```
package picture;
// точка многоугольника
public final class Point {
  private final float x;
  private final float y;
  public Point(float x, float y) {
    this.x = x;
    this.y = y;
  }
  public float getX() {
    return x;
  }
  public float getY() {
    return y;
  }
}
```

5.1.11. Файл Text.java

```
package picture;
import java.util.Collection;
// строка текста
public final class Text extends Primitive
        implements TextRunContainer {
  private final float x;
  private final float y;
  private final TextRunContainerMixin runContainerMixin;
  public Text(Color color, float x, float y,
          Collection<TextRun> runs) {
    super(color);
    this.x = x;
    this.y = y;
    runContainerMixin = new TextRunContainerMixin(runs);
  }
  public float getX() {
    return x;
  }
  public float getY() {
    return y;
  }
  @Override
  public int getRunCount() {
    return runContainerMixin.getRunCount();
```

```
  }
  @Override
  public TextRun getRun(int index) {
    return runContainerMixin.getRun(index);
  }
  @Override
  public <R> R accept(PrimitiveVisitor<R> visitor) {
    return visitor.visitText(this);
  }
}
```

5.1.12. Файл TextRun.java

```
package picture;
// фрагмент строки текста
public interface TextRun {
  <R> R accept(TextRunVisitor<R> visitor);
}
```

5.1.13. Файл TextRunVisitor.java

```
package picture;
// посетитель для интерфейса TextRun
public interface TextRunVisitor<R> {
  R visitLiteral(TextLiteral literal);
  R visitSpan(TextSpan span);
}
```

5.1.14. Файл TextRunContainer.java

```
package picture;
// предок классов, представляющих строку текста
// и фрагмент строки с заданным начертанием
public interface TextRunContainer {
  int getRunCount();
  TextRun getRun(int index);
}
```

5.1.15. Файл TextRunContainerMixin.java

```
package picture;
import java.util.Collection;
// реализация интерфейса TextRunContainer
final class TextRunContainerMixin {
  private final TextRun[] runs;
  TextRunContainerMixin(Collection<TextRun> runs) {
    this.runs = runs.toArray(new TextRun[runs.size()]);
  }
  int getRunCount() {
```

```
    return runs.length;
  }
  TextRun getRun(int index) {
    return runs[index];
  }
}
```

5.1.16. Файл TextLiteral.java

```
package picture;
// простой текст без настойки начертания
public final class TextLiteral implements TextRun {
  private final String string;
  public TextLiteral(String string) {
    this.string = string;
  }
  @Override
  public String toString() {
    return string;
  }
  @Override
  public <R> R accept(TextRunVisitor<R> visitor) {
    return visitor.visitLiteral(this);
  }
}
```

5.1.17. Файл TextSpan.java

```
package picture;
import java.util.Collection;
// фрагмент строки текста с заданным начертанием
public final class TextSpan implements TextRun,
        TextRunContainer {
  private final TextStyle style;
  private final TextRunContainerMixin runContainerMixin;
  public TextSpan(TextStyle s, Collection<TextRun> r) {
    this.style = s;
    runContainerMixin = new TextRunContainerMixin(r);
  }
  public TextStyle getStyle() {
    return style;
  }
  @Override
  public int getRunCount() {
    return runContainerMixin.getRunCount();
  }
  @Override
  public TextRun getRun(int index) {
```

```
    return runContainerMixin.getRun(index);
  }
  @Override
  public <R> R accept(TextRunVisitor<R> visitor) {
    return visitor.visitSpan(this);
  }
}
```

5.1.18. Файл TextStyle.java

```
package picture;
// допустимые начертания
public enum TextStyle {
  BOLD, ITALIC
}
```

5.1.19. Файл Color.java

```
package picture;
// представляет цвет, состоящий из трех компонентов
public final class Color {
  public static final Color BLACK = new Color(0, 0, 0);
  private float red;
  private float green;
  private float blue;
  public Color(float red, float green, float blue) {
    this.red = red;
    this.green = green;
    this.blue = blue;
  }
  public float getRed() {
    return red;
  }
  public float getGreen() {
    return green;
  }
  public float getBlue() {
    return blue;
  }
}
```

5.1.20. Файл Group.java

```
package picture;
import java.util.Collection;
// предок всех классов, представляющих преобразования
// (сдвиг и поворот)
public abstract class Group
        implements Drawable, DrawableContainer {
```

```
  private final DrawableContainerMixin drawableContainer;
  protected Group(Collection<Drawable> drawables) {
    drawableContainer = new DrawableContainerMixin(
            drawables);
  }
  @Override
  public int getDrawableCount() {
    return drawableContainer.getDrawableCount();
  }
  @Override
  public Drawable getDrawable(int index) {
    return drawableContainer.getDrawable(index);
  }
  @Override
  public <R> R accept(DrawableVisitor<R> visitor) {
    return visitor.visitGroup(this);
  }
  public abstract <R> R accept(GroupVisitor<R> visitor);
}
```

5.1.21. Файл GroupVisitor.java

```
package picture;
// посетитель для класса Group
public interface GroupVisitor<R> {
  R visitTranslation(Translation translation);
  R visitRotation(Rotation rotation);
}
```

5.1.22. Файл Translation.java

```
package picture;
import java.util.Collection;
// преобразование сдвига
public final class Translation extends Group {
  private final float deltaX;
  private final float deltaY;
  public Translation(float deltaX, float deltaY,
          Collection<Drawable> drawables) {
    super(drawables);
    this.deltaX = deltaX;
    this.deltaY = deltaY;
  }
  public float getDeltaX() {
    return deltaX;
  }
  public float getDeltaY() {
    return deltaY;
```

```
  }
  @Override
  public <R> R accept(GroupVisitor<R> visitor) {
    return visitor.visitTranslation(this);
  }
}
```

5.1.23. Файл Rotation.java

```
package picture;
import java.util.Collection;
// преобразование поворота
public final class Rotation extends Group {
  private final float x;
  private final float y;
  private final float angle;
  public Rotation(float x, float y, float angle,
          Collection<Drawable> drawables) {
    super(drawables);
    this.x = x;
    this.y = y;
    this.angle = angle;
  }
  public float getX() {
    return x;
  }
  public float getY() {
    return y;
  }
  public float getAngle() {
    return angle;
  }
  @Override
  public <R> R accept(GroupVisitor<R> visitor) {
    return visitor.visitRotation(this);
  }
}
```

5.1.24. Файл Constants.java

```
package picture;
// константы, задающие пространство имен,
// а также имена элементов и атрибутов
public interface Constants {
  String NAMESPACE = "urn:picture-schema";
  String PICTURE_ELEMENT = "pic";
  String RECTANGLE_ELEMENT = "rect";
  String POLYGON_ELEMENT = "poly";
```

```
  String TEXT_ELEMENT = "text";
  String TRANSLATION_ELEMENT = "transl";
  String ROTATION_ELEMENT = "rot";
  String BOLD_ELEMENT = "b";
  String ITALIC_ELEMENT = "i";
  String POINT_ELEMENT = "p";
  String WIDTH_ATTRIBUTE = "w";
  String HEIGHT_ATTRIBUTE = "w";
  String LEFT_ATTRIBUTE = "l";
  String RIGHT_ATTRIBUTE = "r";
  String TOP_ATTRIBUTE = "t";
  String BOTTOM_ATTRIBUTE = "b";
  String X_ATTRIBUTE = "x";
  String Y_ATTRIBUTE = "y";
  String DELTA_X_ATTRIBUTE = "dx";
  String DELTA_Y_ATTRIBUTE = "dy";
  String ANGLE_ATTRIBUTE = "angle";
  String COLOR_ATTRIBUTE = "color";
}
```

5.1.25. Файл PictureIO.java

```
package picture;
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import static picture.Constants.*;
// анализирует XML-документ, содержащий векторное
// изображение, и строит соответствующие структуры данных
public class PictureIO {
  private static DocumentBuilder documentBuilder;
  private PictureIO() {
  }
  // загружает изображение из ресурса,
  // заданного универсальным идентификатором
  public static Picture read(String sourceUri)
          throws PictureIOException {
    try {
      return fromDocument(
              getDocumentBuilder().parse(sourceUri));
    } catch (SAXException | IOException ex) {
      throw new PictureIOException(ex);
    }
  }
  // загружает изображение из входного бинарного потока
```

```
public static Picture read(InputStream source)
        throws PictureIOException {
  try {
    return fromDocument(
            getDocumentBuilder().parse(source));
  } catch (SAXException | IOException ex) {
    throw new PictureIOException(ex);
  }
}
// загружает изображение из входного источника
public static Picture read(InputSource source)
        throws PictureIOException {
  try {
    return fromDocument(
            getDocumentBuilder().parse(source));
  } catch (SAXException | IOException ex) {
    throw new PictureIOException(ex);
  }
}
// преобразует XML-документ в изображение
public static Picture fromDocument(Document document)
        throws PictureIOException {
  Element element = document.getDocumentElement();
  checkNamespace(element);
  switch (element.getLocalName()) {
    case PICTURE_ELEMENT:
      return createPicture(element);
    default:
      throw unexpectedElement(element.getLocalName(),
              PICTURE_ELEMENT);
  }
}
private static Picture createPicture(
        Element e) throws PictureIOException {
  float w = getFloatAttr(e, WIDTH_ATTRIBUTE);
  float h = getFloatAttr(e, HEIGHT_ATTRIBUTE);
  List<Drawable> d = toDrawableList(e.getFirstChild());
  return new Picture(w, h, d);
}
private static List<Drawable> toDrawableList(
        Node n) throws PictureIOException {
  List<Drawable> drawables = new ArrayList<>();
  for (; null != n; n = n.getNextSibling()) {
    switch (n.getNodeType()) {
      case Node.ELEMENT_NODE:
        drawables.add(toDrawable((Element) n));
        continue;
```

```
      case Node.TEXT_NODE:
      case Node.CDATA_SECTION_NODE:
        if (0 == n.getTextContent().trim().length()) {
          continue;
        }
    }
    throw new PictureIOException();
  }
  return drawables;
}
private static Drawable toDrawable(
        Element e) throws PictureIOException {
  checkNamespace(e);
  switch (e.getLocalName()) {
    case RECTANGLE_ELEMENT:
      return createRectangle(e);
    case POLYGON_ELEMENT:
      return createPolygon(e);
    case TEXT_ELEMENT:
      return createText(e);
    case TRANSLATION_ELEMENT:
      return createTranslation(e);
    case ROTATION_ELEMENT:
      return createRotation(e);
  }
  throw unexpectedElement(e.getLocalName(),
          RECTANGLE_ELEMENT, POLYGON_ELEMENT,
          TEXT_ELEMENT, TRANSLATION_ELEMENT,
          ROTATION_ELEMENT);
}
private static Rectangle createRectangle(
        Element e) throws PictureIOException {
  Color c = getColorAttr(e, COLOR_ATTRIBUTE,
          Color.BLACK);
  float t = getFloatAttr(e, TOP_ATTRIBUTE);
  float r = getFloatAttr(e, RIGHT_ATTRIBUTE);
  float b = getFloatAttr(e, BOTTOM_ATTRIBUTE);
  float l = getFloatAttr(e, LEFT_ATTRIBUTE);
  return new Rectangle(c, t, r, b, l);
}
private static Polygon createPolygon(
        Element e) throws PictureIOException {
  Color c = getColorAttr(e, COLOR_ATTRIBUTE,
          Color.BLACK);
  List<Point> p = toPointList(e.getFirstChild());
  return new Polygon(c, p);
}
```

```
private static Text createText(
        Element e) throws PictureIOException {
  Color c = getColorAttr(e, COLOR_ATTRIBUTE,
          Color.BLACK);
  float x = getFloatAttr(e, X_ATTRIBUTE);
  float y = getFloatAttr(e, Y_ATTRIBUTE);
  List<TextRun> r = toTextRunList(e.getFirstChild());
  return new Text(c, x, y, r);
}
private static Translation createTranslation(
        Element e) throws PictureIOException {
  float dx = getFloatAttr(e, DELTA_X_ATTRIBUTE);
  float dy = getFloatAttr(e, DELTA_Y_ATTRIBUTE);
  List<Drawable> d = toDrawableList(e.getFirstChild());
  return new Translation(dx, dy, d);
}
private static Rotation createRotation(
        Element e) throws PictureIOException {
  float x = getFloatAttr(e, X_ATTRIBUTE);
  float y = getFloatAttr(e, Y_ATTRIBUTE);
  float angle = getFloatAttr(e, ANGLE_ATTRIBUTE);
  List<Drawable> d = toDrawableList(e.getFirstChild());
  return new Rotation(x, y, angle, d);
}
private static List<Point> toPointList(
        Node n) throws PictureIOException {
  List<Point> points = new ArrayList<>();
  for (; null != n; n = n.getNextSibling()) {
    switch (n.getNodeType()) {
      case Node.ELEMENT_NODE:
        points.add(toPoint((Element) n));
        continue;
      case Node.TEXT_NODE:
      case Node.CDATA_SECTION_NODE:
        if (0 == n.getTextContent().trim().length()) {
          continue;
        }
    }
    throw new PictureIOException();
  }
  return points;
}
private static Point toPoint(
        Element e) throws PictureIOException {
  checkNamespace(e);
  switch (e.getLocalName()) {
    case POINT_ELEMENT:
```

```
      return createPoint(e);
    default:
      throw unexpectedElement(e.getLocalName(),
              POINT_ELEMENT);
  }
}
private static Point createPoint(
        Element e) throws PictureIOException {
  float x = getFloatAttr(e, X_ATTRIBUTE);
  float y = getFloatAttr(e, Y_ATTRIBUTE);
  return new Point(x, y);
}
private static List<TextRun> toTextRunList(
        Node n) throws PictureIOException {
  List<TextRun> textRuns = new ArrayList<>();
  for (; null != n; n = n.getNextSibling()) {
    switch (n.getNodeType()) {
      case Node.ELEMENT_NODE:
        textRuns.add(toTextSpan((Element) n));
        continue;
      case Node.TEXT_NODE:
      case Node.CDATA_SECTION_NODE:
        textRuns.add(new TextLiteral(
                n.getTextContent()));
        continue;
    }
    throw new PictureIOException();
  }
  return textRuns;
}
private static TextSpan toTextSpan(
        Element e) throws PictureIOException {
  checkNamespace(e);
  switch (e.getLocalName()) {
    case BOLD_ELEMENT:
      return createTextSpan(TextStyle.BOLD, e);
    case ITALIC_ELEMENT:
      return createTextSpan(TextStyle.ITALIC, e);
  }
  throw unexpectedElement(e.getLocalName(),
          BOLD_ELEMENT, ITALIC_ELEMENT);
}
private static TextSpan createTextSpan(TextStyle s,
        Element e) throws PictureIOException {
  List<TextRun> r = toTextRunList(e.getFirstChild());
  return new TextSpan(s, r);
}
```

```
private static float getFloatAttr(Element e, String n)
        throws PictureIOException {
  String a = e.getAttribute(n);
  if (a.isEmpty()) {
    throw new PictureIOException(
            String.format("No %s attribute", n));
  }
  try {
    return Float.parseFloat(a);
  } catch (NumberFormatException ex) {
    throw new PictureIOException(ex);
  }
}
private static Color getColorAttr(Element e, String n,
        Color c) throws PictureIOException {
  String a = e.getAttribute(n);
  if (a.isEmpty()) {
    return c;
  }
  StringTokenizer t = new StringTokenizer(a);
  if (3 != t.countTokens()) {
    throw new PictureIOException(
            "The color specification should have"
            + " exactly three components.");
  }
  try {
    float r = Float.parseFloat(t.nextToken());
    float g = Float.parseFloat(t.nextToken());
    float b = Float.parseFloat(t.nextToken());
    return new Color(r, g, b);
  } catch (NumberFormatException ex) {
    throw new PictureIOException(ex);
  }
}
private static void checkNamespace(Node n)
        throws PictureIOException {
  String namespace = n.getNamespaceURI();
  if (!NAMESPACE.equals(n.getNamespaceURI())) {
    throw new PictureIOException(String.format(
            "Unexpected namespace '%s'.", namespace));
  }
}
private static PictureIOException unexpectedElement(
        String actualName, String... expectedNames) {
  String s = expectedNames[0];
  for (int i = 1; i < expectedNames.length; ++i) {
    s = String.format("%s or %s", s, expectedNames[i]);
```

```
    }
    return new PictureIOException(String.format(
            "Unexpected element %s. Expects %s",
            actualName, s));
  }
  private static DocumentBuilder getDocumentBuilder() {
    if (null == documentBuilder) {
      DocumentBuilderFactory factory =
              DocumentBuilderFactory.newInstance();
      factory.setIgnoringComments(true);
      factory.setNamespaceAware(true);
      try {
        documentBuilder = factory.newDocumentBuilder();
      } catch (ParserConfigurationException ex) {
        throw new RuntimeException("Internal error", ex);
      }
    }
    return documentBuilder;
  }
}
```

6. Контрольные вопросы

1. Что такое элемент XML?
2. Что такое атрибут элемента XML?
3. Какие виды тегов бывают и для чего они используются?
4. Что называют ссылкой?
5. Какие сущностные ссылки встроены в XML?
6. Для чего используются пространства имен?
7. Как сопоставить префикс пространству имен?
8. Для чего используется XML-декларация?
9. Какие режимы разбора XML-документа поддерживаются стандартной библиотекой языка Java?
10. Для чего используется построитель документов и как его создать?
11. Какие опции поддерживаются построителем документов?
12. Что называется деревом документа?
13. Что представляет интерфейс *Node*?
14. Как получить тип, имя, пространство имен и значение узла?
15. Как получить значение атрибута узла?
16. Какие методы можно использовать для навигации по дереву документа?

17. Какие методы можно использовать для модификации дерева документа?
18. Как создать XML-разборщик и какие опции им поддерживаются?
19. Когда следует реализовывать интерфейс *ContentHandler*?
20. Как можно получить значения атрибутов в методе *startElement* интерфейса *ContentHandler*?

7. Варианты заданий

1. Построить гистограмму по данным из XML-файла;
2. Построить круговую диаграмму по данным из XML-файла;
3. Нарисовать на экране фигуру, записанную в XML-файле в виде команд «черепашьей» графики;
4. Загрузить из XML-файла главное меню программы;
5. Вывести, загруженную из XML-файла, строку текста (используя разные начертания, цвета и т. д.);
6. Вывести многоуровневый список, загруженный из XML-файла;
7. Вывести таблицу произвольной структуры, загруженную из XML-файла;
8. Провести тестирование знаний с использованием записанных в XML-файле заданий, связанных с выбором правильного варианта;
9. Провести тестирование знаний с использованием записанных в XML-файле заданий, связанных с упорядочением элементов в правильном порядке;
10. Провести тестирование знаний с использованием записанных в XML-файле заданий, связанных с установлением соответствий;
11. Вывести на экран кроссворд, записанный в XML-файле, и предоставить пользователю возможность решить его;
12. Смоделировать работу комбинационной схемы, записанной в XML-файле;
13. Выполнить алгоритм, записанный в XML-файле в виде команд стековой машины;
14. Выполнить алгоритм, записанный в XML-файле в виде команд, похожих на инструкции ассемблера;
15. Выполнить алгоритм, записанный в XML-файле в виде блок-схемы;

16. Выполнить алгоритм, записанный в XML-файле в виде команд, похожих на операторы языка высокого уровня.

8. Список литературы

1. Хабибуллин И. Ш. Самоучитель XML. — СПб.: БХВ-Петербург, 2003. — 336 с.
2. Гарольд Э., Минс С. XML. Справочник. — СПб.: Символ-Плюс, 2002. — 576 с.