```
            address.setCity(
                  getBabyValue(addressElement, "city"));
            address.setStreet(
                  getBabyValue(addressElement, "street"));

                  students.add(student);
                  }
            return students;
      }
      // возвращает дочерний элемент по его имени и родительскому элементу
      private static Element getBaby(Element parent,
                                    String childName) {
            NodeList nlist =
                  parent.getElementsByTagName(childName);
            Element child = (Element) nlist.item(0);
            return child;
      }
      // возвращает текст, содержащийся в элементе
      private static String getBabyValue(Element parent,
                                    String childName) {
            Element child = getBaby(parent, childName);
            Node node = child.getFirstChild();
            String value = node.getNodeValue();
            return value;
      }
}
```

JDOM

JDOM не является анализатором, он был разработан для более удобного, более интуитивного для Java-программист, доступа к объектной модели XML-документа. JDOM представляет свою модель, отличную от DOM. Для разбора документа JDOM использует либо SAX-, либо DOM-парсеры сторонних производителей. Реализаций JDOM немного, так как он основан на классах, а не на интерфейсах.

Разбирать XML-документы с помощью JDOM проще, чем с помощью Xerces. Иерархия наследования объектов документа похожа на Xerces.

Content

В корне иерархии наследования стоит класс **`Content`**, от которого унаследованы остальные классы (**`Text`**, **`Element`** и др.).

Основные методы класса **`Content`**:

`Document getDocument()` – возвращает объект, в котором содержится этот элемент;

`Element getParentElement()` – возвращает родительский элемент.

Document

Базовый объект, в который загружается после разбора XML-документ. Аналогичен **`Document`** из Xerces.

`Element getRootElement()` – возвращает корневой элемент.

Parent

Интерфейс **Parent** реализуют классы **Document** и **Element**. Он содержит методы для работы с дочерними элементами. Интерфейс **Parent** и класс **Content** реализуют ту же функциональность, что и интерфейс **Node** в Xerces.

Некоторые из его методов:

List getContent() – возвращает все дочерние объекты;

Content getContent(int index) – возвращает дочерний элемент по его индексу;

int getContentSize() – возвращает количество дочерних элементов;

Parent getParent() – возвращает родителя этого родителя;

int indexOf(Content child) – возвращает индекс дочернего элемента.

Element

Класс **Element** представляет собой элемент XML-документа.

Attribute getAttribute(String name) – возвращает атрибут по его имени;

String getAttributeValue(String name) – возвращает значение атрибута по его имени;

List getAttributes() – возвращает список всех атрибутов;

Element getChild(String name) – возвращает дочерний элемент по имени;

List getChildren() – возвращает список всех дочерних элементов;

String getChildText(String name) – возвращает текст дочернего элемента;

String getName() – возвращает имя элемента;

String getText() – возвращает текст, содержащийся в элементе.

Text

Класс **Text** содержит методы для работы с текстом. Аналог в Xerces – интерфейс **Text**.

String getText() – возвращает значение содержимого в виде строки;

String getTextTrim() – возвращает значение содержимого без крайних пробельных символов.

Attribute

Класс **Attribute** представляет собой атрибут элемента XML-документа. В отличие от интерфейса **Attr** из Xerces, у класса **Attribute** расширенная функциональность. Класс **Attribute** имеет методы для возвращения значения определенного типа.

int getAttributeType() – возвращает тип атрибута;

тип **get**Тип**Type()** – (**Int**, **Double**, **Boolean**, **Float**, **Long**) возвращает значение определенного типа;

String getName() – возвращает имя атрибута;

Element getParent() – возвращает родительский элемент.

Следующие примеры выполняют ту же функцию, что и предыдущие, только с помощью JDOM.

```
/* пример # 8 : запуск JDOM : JDOMStudentMain.java */
package chapt16.main;
import java.util.List;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import chapt16.analyzer.dom.JDOMAnalyzer;
import chapt16.entity.Student;

public class JDOMStudentMain {
   public static void main(String[] args) {
      try {
      //создание JDOM
      SAXBuilder builder = new SAXBuilder();
      //распознавание XML-документа
      Document document = builder.build("students.xml");
         List<Student> list =
            JDOMAnalyzer.listCreator(document);

      for (Student st : list) System.out.println(st);
      } catch(IOException e) {
         e.printStackTrace();
      } catch(JDOMException e) {
         e.printStackTrace();
      }
   }
}
/* пример # 9 : создание объектов с использованием JDOM: JDOMAnalyzer.java */
package chapt16.analyzer.dom;
import java.util.*;
import java.io.IOException;
import org.jdom.Element;
import org.jdom.Document;
import org.jdom.JDOMException;
import chapt16.entity.Student;

public class JDOMAnalyzer {
   public static List<Student> listCreator(Document doc)
         throws JDOMException, IOException {
      //извлечение корневого элемента
      Element root = doc.getRootElement();
      //получение списка дочерних элементов <student>
      List studElem = root.getChildren();
      Iterator studentIterator = studElem.iterator();
      //создание пустого списка объектов типа Student
      ArrayList<Student> students =
                  new ArrayList<Student>();
      while(studentIterator.hasNext()) {
```

```
            Element studentElement =
               (Element)studentIterator.next();
            Student student = new Student();
            //заполнение объекта student
            student.setLogin(
         studentElement.getAttributeValue("login"));
            student.setName(
         studentElement.getChild("name").getText());
            student.setTelephone(
         studentElement.getChild("telephone").getText());
            student.setFaculty(
         studentElement.getAttributeValue("faculty"));

            Element addressElement =
               studentElement.getChild("address");
            Student.Address address = student.getAddress();
            //заполнение объекта address
address.setCountry(addressElement.getChild("country")
                                             .getText());
address.setCity(addressElement.getChild("city").getText());
address.setStreet(addressElement.getChild("street")
                                             .getText());
            students.add(student);
         }
         return students;
      }
}
```

Создание и запись XML-документов

Документы можно не только читать, но также модифицировать и создавать совершенно новые.

Для создания документа необходимо создать объект каждого класса (**Element**, **Attribute**, **Document**, **Text** и др.) и присоединить его к объекту, который в дереве XML-документа находится выше. В данном разделе будет рассматриваться только анализатор JDOM.

Element

Для добавления дочерних элементов, текста или атрибутов в элемент XML-документа нужно использовать один из следующих методов:

Element addContent(Content child) – добавляет дочерний элемент;

Element addContent(int index, Content child) – добавляет дочерний элемент в определенную позицию;

Element addContent(String str) – добавляет текст в содержимое элемента;

Element setAttribute(Attribute attribute) – устанавливает значение атрибута;

Element setAttribute(String name, String value) – также устанавливает значение атрибута;

Element setContent(Content child) – заменяет содержимое этого элемента на элемент, переданный в качестве параметра;

Element setContent(int index, Content child) – заменяет дочерний элемент на определенной позиции элементом, переданным как параметр;

Element setName(String name) – устанавливает имя элемента;

Element setText(String text) – устанавливает текст содержимого элемента.

Text

Класс **Text** также имеет методы для добавления текста в элемент XML-документа:

void append(String str) – добавляет текст к уже имеющемуся;

void append(Text text) – добавляет текст из другого объекта **Text**, переданного в качестве параметра;

Text setText(String str) – устанавливает текст содержимого элемента.

Attribute

Методы класса **Attribute** для установки значения, имени и типа атрибута:

Attribute setAttributeType(int type) – устанавливает тип атрибута;

Attribute setName(String name) – устанавливает имя атрибута;

Attribute setValue(String value) – устанавливает значение атрибута.

Следующий пример демонстрирует создание XML-документа и запись его в файл. Для записи XML-документа используется класс **XMLOutputter**.

```
/* пример # 10 : создание и запись документа с помощью JDOM:
JDOMLogic.java */
package chapt16.saver.dom;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
import java.util.Iterator;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.XMLOutputter;
import chapt16.entity.Student;

public class JDOMLogic {
      public static Document create(List<Student> list) {
//создание корневого элемента <studentsnew>
      Element root = new Element("studentsnew");
      Iterator<Student> studentIterator =
         list.iterator();
      while(studentIterator.hasNext()) {
         Student student = studentIterator.next();
         //создание элемента <student> и его содержимого
         Element studentElement = new Element("student");
         //создание атрибутов  и передача им значений
         studentElement.setAttribute("login",
            student.getLogin());
```

```
            studentElement.setAttribute("phone",
               student.getTelephone());

               Element faculty = new Element("faculty");
               faculty.setText(student.getFaculty());
            //«вложение» элемента <faculty> в элемент <student>
               studentElement.addContent(faculty);

               Element name = new Element("name");
               name.setText(student.getName());
               studentElement.addContent(name);
            //создание элемента <address>
            Element addressElement = new Element("address");
            Student.Address address = student.getAddress();

            Element country = new Element("country");
            country.setText(address.getCountry());
            addressElement.addContent(country);

            Element city = new Element("city");
            city.setText(address.getCity());
            addressElement.addContent(city);

            Element street = new Element("street");
            street.setText(address.getStreet());
            // «вложение» элемента <street> в элемент <address>
            addressElement.addContent(street);
            //«вложение» элемента <address> в элемент <student>
            studentElement.addContent(addressElement);
            //«вложение» элемента <student> в элемент <students>
            root.addContent(studentElement);
         }
            //создание основного дерева XML-документа
         return new Document(root);
      }
public static boolean saveDocument(String fileName,
                                   Document doc) {
            boolean complete = true;
            XMLOutputter outputter = new XMLOutputter();
            // запись XML-документа
            try {
outputter.output(doc, new FileOutputStream(fileName));
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
                  complete = false;
            } catch (IOException e) {
                  e.printStackTrace();
                  complete = false;
            }
```

```
            return complete;
      }
}
```

/ пример # 11 : создание списка и запуск приложения : JDOMMainSaver.java*/*

```
package chapt16.main;
import java.io.IOException;
import java.util.ArrayList;
import chapt16.entity.Student;
import chapt16.saver.dom.JDOMLogic;

public class JDOMMainSaver {
   public static void main(String[] args) {
         //создание списка студентов
ArrayList<Student> students = new ArrayList<Student> ();
         for(int j = 1; j < 3; j++) {
            Student st = new Student();
            st.setName("Petrov" + j);
            st.setLogin("petr" + j);
            st.setFaculty("mmf");
            st.setTelephone("454556"+ j*3);
          Student.Address adr = st.getAddress();
            adr.setCity("Minsk");
            adr.setCountry("BLR");
            adr.setStreet("Gaja, " + j);
                  st.setAddress(adr);
                        students.add(st);
         }
         //создание «дерева» на основе списка студентов
         Document doc = JDOMLogic.create(students);
         //сохранение «дерева» в XML-документе
      if(JDOMLogic.saveDocument("studentsnew.xml", doc))
            System.out.println("Документ создан");
      else
            System.out.println("Документ НЕ создан");
   }
}
```

В результате будет создан документ **studentsnew.xml** следующего содержания:

```
<?xml version="1.0" encoding="UTF-8"?>
<studentsnew>
      <student login="petr1" phone="4545563">
            <faculty>mmf</faculty>
            <name>Petrov1</name>
            <address>
                  <country>BLR</country>
                  <city>Minsk</city>
                  <street>Gaja, 1</street>
            </address>
      </student>
```

```
    <student login="petr2" phone="4545566">
          <faculty>mmf</faculty>
          <name>Petrov2</name>
          <address>
                <country>BLR</country>
                <city>Minsk</city>
                <street>Gaja, 2</street>
          </address>
    </student>
</studentsnew>
```

В этом примере был использован JDOM, основанный на идее "if something doesn't work, fix it".

StAX

StAX (Streaming API for XML), который еще называют pull-парсером, включен в JDK, начиная с версии Java SE 6. Он похож на SAX отсутствием объектной модели в памяти и последовательным продвижением по XML, но в StAX не требуется реализация интерфейсов, и приложение само командует StAX-парсеру перейти к следующему элементу XML. Кроме того, в отличие от SAX, данный парсер предлагает API для создания XML-документа.

Основными классами StAX являются **XMLInputFactory**, **XMLStreamReader** и **XMLOutputFactory**, **XMLStreamWriter**, которые соответственно используются для чтения и создания XML-документа. Для чтения XML надо получить ссылку на **XMLStreamReader**:

```
StringReader stringReader = new StringReader(xmlString);
XMLInputFactory inputFactory=XMLInputFactory.newInstance();
XMLStreamReader reader = inputFactory
                  .createXMLStreamReader(stringReader);
```

после чего **XMLStreamReader** можно применять аналогично интерфейсу **Iterator**, используя методы **hasNext()** и **next()**:

boolean hasNext() – показывает, есть ли еще элементы;

int next() – переходит к следующей вершине XML, возвращая ее тип.

Возможные типы вершин:

XMLStreamConstants.START_DOCUMENT
XMLStreamConstants.END_DOCUMENT
XMLStreamConstants.START_ELEMENT
XMLStreamConstants.END_ELEMENT
XMLStreamConstants.CHARACTERS
XMLStreamConstants.ATTRIBUTE
XMLStreamConstants.CDATA
XMLStreamConstants.NAMESPACE
XMLStreamConstants.COMMENT
XMLStreamConstants.ENTITY_DECLARATION

Далее данные извлекаются применением методов:

String getLocalName() – возвращает название тега;

String getAttributeValue(NAMESPACE_URI, ATTRIBUTE_NAME) – возвращает значение атрибута;

String getText() – возвращает текст тега.