```
        public void setCountry(String country) {
            this.country = country;
        }
        public String getCity() {
            return city;
        }
        public void setCity(String city) {
            this.city = city;
        }
        public String getStreet() {
            return street;
        }
        public void setStreet(String street) {
            this.street = street;
        }
    }
}
```

Древовидная модель

Анализатор DOM представляет собой некоторый общий интерфейс для работы со структурой документа. При разработке DOM-анализаторов различными вендорами предполагалась возможность ковариантности кода.

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

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.

Существуют различные общепризнанные DOM-анализаторы, которые в настоящий момент можно загрузить с указанных адресов:

Xerces – http://xerces.apache.org/xerces2-j/;

JAXP – входит в JDK.

Существуют также библиотеки, предлагающие свои структуры объектов XML с API для доступа к ним. Наиболее известные:

JDOM – http://www.jdom.org/dist/binary/jdom-1.0.zip.

dom4j – http://www.dom4j.org

Xerces

В стандартную конфигурацию Java входит набор пакетов для работы с XML. Но стандартная библиотека не всегда является самой простой в применении, поэтому часто в основе многих проектов, использующих XML, лежат библиотеки сторонних производителей. Одной из таких библиотек является Xerces, замечательной особенностью которого является использование части стандартных возможностей XML-библиотек JSDK с добавлением собственных классов и методов, упрощающих и облегчающих обработку документов XML.

org.w3c.dom.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.

`Element getDocumentElement()` – возвращает корневой элемент документа.

org.w3c.dom.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`**.

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

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

`String getNodeValue()` – возвращает значение **`Node`**;

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

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

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

`NamedNodeMap getAttributes()` – возвращает список атрибутов данного элемента.

У интерфейса **`Node`** есть несколько важных наследников – **`Element`**, **`Attr`**, **`Text`**. Они используются для работы с конкретными объектами дерева.

org.w3c.dom.Element

Интерфейс предназначен для работы с содержимым элементов XML-документа. Некоторые методы:

`String getTagName(String name)` – возвращает имя элемента;

`boolean hasAttribute()` – проверяет наличие атрибутов;

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

`Attr getAttributeNode(String name)` – возвращает атрибут по его имени;

`void setAttribute(String name, String value)` – устанавливает значение атрибута, если необходимо, атрибут создается;

`void removeAttribute(String name)` – удаляет атрибут;

`NodeList getElementsByTagName(String name)` – возвращает список дочерних элементов с определенным именем.

org.w3c.dom.Attr

Интерфейс служит для работы с атрибутами элемента XML-документа.

Некоторые методы интерфейса **`Attr`**:

`String getName()` – возвращает имя атрибута;

`Element getOwnerElement` – возвращает элемент, который содержит этот атрибут;

String getValue() – возвращает значение атрибута;
void setValue(String value) – устанавливает значение атрибута;
boolean isId() – проверяет атрибут на тип ID.

org.w3c.dom.Text

Интерфейс **Text** необходим для работы с текстом, содержащимся в элементе.
String getWholeText() – возвращает текст, содержащийся в элементе;
void replaceWholeText(String content) – заменяет строкой **content** весь текст элемента.

В следующих примерах производятся разбор документа **students.xml** с использованием DOM-анализатора и инциализация на его основе набора объектов.

```
/* пример # 6 : создание анализатора и загрузка XML-документа:
DOMLogic.java*/
package chapt16.main;
import java.util.ArrayList;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
//import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;
import chapt16.analyzer.dom.Analyzer;
import chapt16.entity.Student;

public class DOMLogic {
      public static void main(String[] args) {
            try {
            // создание DOM-анализатора(JSDK)
            DocumentBuilderFactory dbf=
                  DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            // распознавание XML-документа
            Document document = db.parse("students.xml");

             // создание DOM-анализатора (Xerces)
            /* DOMParser parser = new DOMParser();
            parser.parse("students.xml");
            Document document = parser.getDocument();*/

            Element root = document.getDocumentElement();
ArrayList<Student> students = Analyzer.listBuilder(root);

            for (int i = 0; i < students.size(); i++) {
                  System.out.println(students.get(i));
                  }
```

```
      } catch (SAXException e) {
         e.printStackTrace();
         System.out.print("ошибка SAX парсера");
      } catch (ParserConfigurationException e) {
         e.printStackTrace();
         System.out.print("ошибка конфигурации");
      } catch (IOException e) {
         e.printStackTrace();
         System.out.print("ошибка I/O потока");
            }
      }
}
```

/ пример # 7 : создание объектов на основе объекта типа Element : Analyzer.java */*

```
package chapt16.analyzer.dom;
import java.util.ArrayList;
import java.io.IOException;
import org.xml.sax.SAXException;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import chapt16.entity.Student;

public class Analyzer {
 public static ArrayList<Student> listBuilder(Element root)
                  throws SAXException, IOException {
            ArrayList<Student> students
                              = new ArrayList<Student>();
                  // получение списка дочерних элементов <student>
                  NodeList studentsNodes =
                      root.getElementsByTagName("student");
                  Student student = null;
      for (int i = 0; i < studentsNodes.getLength(); i++) {
            student = new Student();
            Element studentElement =
                        (Element) studentsNodes.item(i);
                        // заполнение объекта student
student.setFaculty(studentElement.getAttribute("faculty"));
student.setName(getBabyValue(studentElement, "name"));
student.setLogin(studentElement.getAttribute("login"));
student.setTelephone(
            getBabyValue(studentElement, "telephone"));

            Student.Address address = student.getAddress();
                        // заполнение объекта address
            Element addressElement =
                  getBaby(studentElement, "address");
            address.setCountry(
                getBabyValue(addressElement, "country"));
```

```
            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.