XML

XML (*e**X**tensible **M**arkup **L**anguage*) — расширяемый язык разметки; Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

С логической точки зрения, документ состоит из *пролога* и ***корневого элемента***. Корневой элемент — обязательная часть документа, пролог, вообще говоря, может отсутствовать.

Пролог может включать ***объявления***, ***инструкции обработки***, ***комментарии***.

Корневой элемент может включать (а может не включать) вложенные в него ***элементы*** и *символьные данные*, а также комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее.

Элементы документа должны быть *правильно вложены*: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался.

Символьные данные могут встречаться внутри элементов как непосредственно так и в специальных *секциях CDATA*.

XML

Объявление XML объявляет версию языка, на которой написан документ. Поскольку интерпретация содержимого документа, вообще говоря, зависит от версии языка, то Спецификация предписывает начинать документ с объявления XML. В первой (1.0) версии языка использование объявления не было обязательным, в последующих версиях оно обязательно. Таким образом, версия языка определяется из объявления, и если объявление отсутствует, то принимается версия 1.0.

```
<?xml version="1.1" encoding="UTF-8" ?>
```

Комментарии не относятся к символьным данным документа. Комментарий начинается последовательностью «<!--» и заканчивается последовательностью «-->», внутри не может встречаться комбинация символов «--». Символ & не используется внутри комментария в качестве разметки.

XML

Элемент является понятием логической структуры документа. Каждый документ содержит один или несколько элементов. Границы элементов представлены *начальным* и *конечным тегами*. Имя элемента в начальном и конечном тегах элемента должно совпадать. Элемент может быть также представлен тегом *пустого*, то есть не включающего в себя другие элементы и символьные данные, *элемента*.

Тег — конструкция разметки, которая содержит имя элемента.

Начальный тег: <element1>

Конечный тег: </element1>

Тег пустого элемента: <empty_element1 />

В элементе атрибуты могут использоваться только в начальном теге и теге пустого элемента.

XML

```
<?xml version="1.0" encoding="koi-8"?>
<notepad>
 <note id="1" date="12/04/14" time="13:40">
 <subject>Важная деловая встреча</subject>
 <importance/>
 <text>
 Надо встретиться с <person id="1625">Иваном Ивановичем</person>,
 предварительно позвонив ему по телефону <tel>123-12-12</tel>
 </text>
 </note>
 ...
 <note id="2" date="12/04/14" time="13:58">
 <subject>Позвонить домой</subject>
 <text>
 <tel>124-13-13</tel>
 </text>
 </note>
</notepad>
```

XML

```
<recipe name="хлеб" preptime="5" cooktime="180">
  <title>Простой хлеб</title>
  <composition>
    <ingredient amount="3" unit="стакан">Мука</ingredient>
   <ingredient amount="0.25" unit="грамм">Дрожжи</ingredient>
 <ingredient amount="1.5" unit="стакан">Тёплая вода</ingredient>
 <ingredient amount="1" unit="чайная ложка">Соль</ingredient>
  </composition>
  <instructions>
    <step>Смешать все ингредиенты и
        тщательно замесить.</step>
    <step>Закрыть тканью и
      оставить на один час в тёплом помещении.</step>
    <!-- <step>Почитать вчерашнюю газету.</step>
         - это сомнительный шаг... -->
    <step>Замесить ещё раз, положить на
      противень и поставить в духовку.</step>
  </instructions>
</recipe>
```

XML Schema

XML Schema — язык описания структуры XML-документа. Спецификация XML Schema является рекомендацией W3C.

Как большинство языков описания XML, XML Schema была задумана для определения правил, которым должен подчиняться документ. Но, в отличие от других языков, XML Schema была разработана так, чтобы её можно было использовать в создании программного обеспечения для обработки документов XML.

После проверки документа на соответствие XML Schema читающая программа может создать модель данных документа, которая включает:

словарь (названия элементов и атрибутов);

модель содержания (отношения между элементами и атрибутами и их структура);

типы данных.

Каждый элемент в этой модели ассоциируется с определённым типом данных, позволяя строить в памяти объект, соответствующий структуре XML-документа.

```
<?xml version="1.0" encoding="utf-8"?>
<country>
    <country_name>France</country_name>
    <population>59.7</population>
</country>
```

XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="country">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="country_name" type="xs:string"/>
        <xs:element name="population" type="xs:decimal"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML

***Правило XML #1*:** Адекватный XML файл должен в точности соответствовать своей схеме.

***Правило XML #2*:** XML чувствителен к регистру.

***Правило XML #3*:** Тэги принято называть элементами и каждый открывающийся тэг, должен иметь соответствующий ему закрывающийся тэг. Следуя этому правилу, у вас получится правильный XML файл. Это очень важно, потому что до тех пор, пока XML файл не будет правильно оформлен, он не будет проанализирован и не загрузится в объектную модель документов. Заметьте, если элемент не содержит значений и не содержит других (вложенных) элементов, закрывающий тэг может иметь вид <Element /> вместо более громоздкой конструкции <Element></Element>.

***Правило XML #4*:** Элементы могут содержать атрибуты, а значения атрибутов должны быть заключены в кавычки (одинарные или двойные).

***Правило XML #5*:** Можно несколько раз использовать имена атрибутов, но имена элементов должны быть уникальны для всего файла. Значение атрибута зависит от контекста его использования.

***Правило XML #6*:** В XML есть несколько специальных символов, которые не могут быть использованы напрямую, потому что являются зарезервированными в синтаксисе XML.

DOM

DOM (от англ. *Document Object Model* — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями "родительский-дочерний".

XML документ представляет собой набор тегов — узлов. Каждый узел может иметь неограниченное количество дочерних узлов. Каждый дочерний тоже может содержать много-много потомков или не содержать их вовсе. Таким образом получается некое дерево. DOM представляет собой всё это дерево в виде специальных объектов Node. Каждый Node соответствует своему XML-тегу. Каждый Node содержит полную информацию о том, что это за тег, какие он имеет атрибуты, какие дочерние узлы содержит внутри себя и так далее.

DOM

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <class name = "MainClass">
        <method name = "main"/>
    </class>
    <class name = "Window">
        <method name1 = "open"/>
      <comment> test comment for window </comment>
        <method name = "close"/>
        <method name = "show"/>
        <method name = "hide"/>
    </class>
    <class name = "DataBase">
        <method name = "connect"/>
        <method name = "disconnect"/>
      <comment> test comment for Database </comment>
        <method name = "getData"/>
    </class>
</application>
```

DOM

На самой вершине иерархии находится Document.
Для того, чтобы получить объект Document для нашего XML-файла необходимо выполнить следующий код.

```
DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = f.newDocumentBuilder();
Document doc = builder.parse(new File("1.xml"));
```

Далее, можно получить корневой элемент:

```
Element root = doc.getDocumentElement();
System.out.println(root.getNodeName());
```

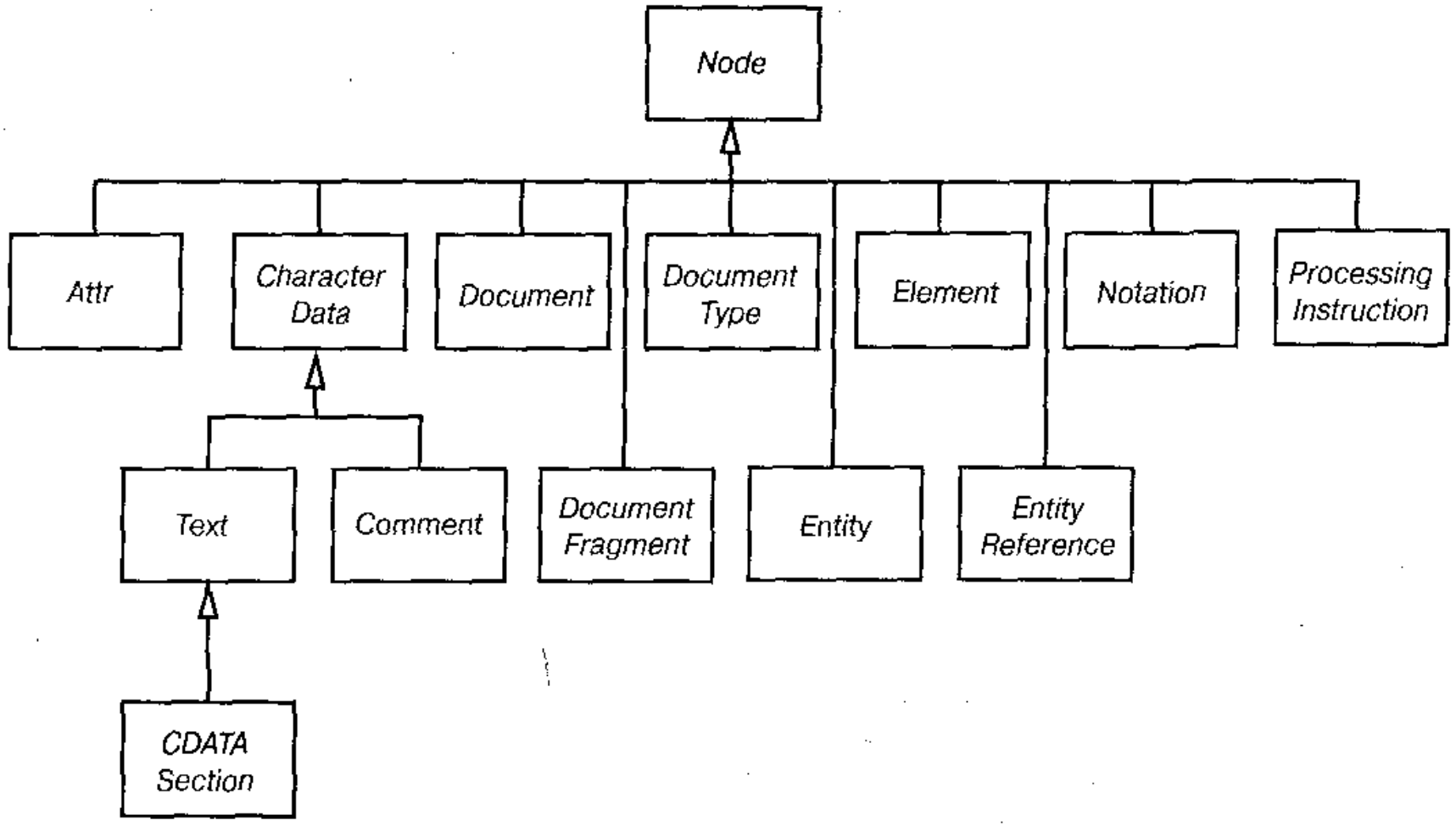
Для того, чтобы посмотреть все дочерние элементы можно использовать следующий код:

```
NodeList nl = root.getChildNodes();
for (int i=0;i<nl.getLength();i++){
      Node n = nl.item(i);
      System.out.println(n);}
```

Нужно обратить внимание на то, что результат будет не совсем совпадать с тем, что мы ожидаем увидеть

```
if( n instanceof Element) System.out.println(n);
```

DOM


Node
Attr
Character Data
Document
Document Type
Element
Notation
Processing Instruction
Text
Comment
Document Fragment
Entity
Entity Reference
CDATA Section

DOM

```
NodeList nl = root.getChildNodes();
for (int i=0;i<nl.getLength();i++){
  Node n = nl.item(i);
  if( n instanceof Element) {
  Element e = (Element)n;
  System.out.println(e.getAttribute("name"));
  NodeList nl_e = e.getElementsByTagName("comment");
  for(int j=0;j<nl_e.getLength();j++){
    if (nl_e.item(j) instanceof Element){
       System.out.println(
            nl_e.item(j).getFirstChild().getNodeValue());
    }
  }
}
```

```
public static void printDomTree(Node node){
   int type = node.getNodeType();
   switch (type){
     case Node.DOCUMENT_NODE:{
      System.out.println("<?xml version=\"1.0\" ? >");
     printDomTree (((Document)node).getDocumentElement());
      break;}
      case Node.ELEMENT_NODE:{
        System.out.print("<");
        System.out.print(node.getNodeName());
        NamedNodeMap attrs = node.getAttributes();
        for (int i = 0; i < attrs.getLength(); i++)
            printDomTree (attrs.item(i));
        System.out.print(">");
        if (node.hasChildNodes()){
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
            printDomTree (children.item(i));}
        System.out.print("</");
        System.out.print(node.getNodeName());
        System.out.print(">");break;}
```

DOM

```
    case Node.ATTRIBUTE_NODE:
      {
       if (!node.getNodeName().equals("name"))
       System.out.print(" " + node.getNodeName() + "=\"" +
        ((Attr)node).getValue() + "\"");
        break;
      }
      case Node.TEXT_NODE:
      {
        System.out.print(node.getNodeValue());
        break;
      }
    }
}
```

XPath

XPath (XML Path Language) — язык запросов к элементам XML-документа.
Подробнее про XPath(прочитать самостоятельно!!!):
http://codingcraft.ru/xpath.php

```
XPathFactory xpf = XPathFactory.newInstance();
XPath xp =xpf.newXPath();
nl=(NodeList)xp.evaluate("application/class/comment",
                                doc,XPathConstants.NODESET);
String st = xp.evaluate("application/class[3]/@name",
                                                    doc);
System.out.println(st);
System.out.println(nl.getLength());
```

Simple API for XML - SAX

•DOM строит в памяти дерево XML-документа. Если документ действительно большой, дерево DOM может требовать огромного объема памяти.
•Дерево DOM содержит множество объектов, представляющих содержимое исходного XML- документа. Если вам нужно всего несколько вещей из документа, создание всех этих объектов является расточительством.
•Парсер DOM должен построить полное дерево DOM прежде, чем код сможет работать с ним. Если вы разбираете большой XML-документ, вы можете получить значительную задержку, ожидая, пока парсер закончит работу.

SAX - это **толчковый** API: вы создаете парсер SAX, и парсер сообщает вам (он проталкивает вам события), когда он находит в XML-документе разные вещи. В частности, вот как SAX решает вопросы, упомянутые выше:
SAX не строит в памяти дерево XML-документа. Парсер SAX посылает вам события по мере того, как он находит в XML-документе разные вещи. На ваше усмотрение решать, как (и если) вы хотите сохранять эти данные.
Парсер SAX не создает никаких объектов. Вы можете на выбор создавать объекты, если вы хотите, на это - ваше решение, а не парсера.
SAX начинает посылать вам события немедленно. Вы не должны ожидать, пока парсер закончит чтение документа.

SAX

API SAX определяет много событий. Вашей работой является написание кода Java, который что-то делает, откликаясь на все эти события. Чаще всего вы используете в вашем приложении вспомогательные классы SAX. Когда вы используете вспомогательные классы, вы пишете обработчики для нескольких событий, которые вас интересуют, и предоставляете остальную работу вспомогательным классам. Если вы не обрабатываете событие, парсер удаляет его, так что вы не должны беспокоиться об использовании памяти, ненужных объектах и других проблемах, которые вы имеете, используя парсер DOM.

Имейте в виду, что события SAX **не сохраняют состояние**. Другими словами, вы не можете посмотреть на отдельное событие SAX и вычислить его содержимое. Если вам нужно знать, что определенная часть текста находится внутри элемента <lastName>, который находится внутри элемента <author>, на вас ложится задача отслеживать, в каком элементе находился парсер до того. Все, что событие SAX сообщает вам, это: "Вот некоторый текст". Это ваша работа - вычислить, к какому элементу этот текст относится.

SAX

startDocument()

Сообщает вам, что парсер нашел начало документа. Это событие не передает вам никакой информации, оно просто дает вам знать, что парсер начал сканирование документа.

endDocument()

Сообщает вам, что парсер нашел конец документа.

startElement(...)

Сообщает вам, что парсер нашел начальный тег. Это событие сообщает вам имя элемента, имена и значения атрибутов элемента и дает вам также некоторую информацию пространства имен.

characters(...)

Сообщает вам, что парсер нашел некоторый текст. Вы получаете массив символов и переменные смещения в массиве и длины; вместе эти три переменные дают вам текст, который нашел парсер.

endElement(...)

Сообщает вам, что парсер нашел конечный тег. Это событие сообщает вам имя элемента, а также связанную с ним информацию пространства имен.

SAX

```
class MyParser extends DefaultHandler {
@Override
public void startElement(String uri, String localName,
 String qName,Attributes attributes) throws SAXException {
    System.out.println("Тег: "+qName);
    if(qName.equals("class"))
System.out.println("класс"+attributes.getValue("name"));
}
@Override
public void characters(char[] c, int start, int length)
                                        throws SAXException {
      for(int i=start;i< start+length;++i)
                System.out.print(c[i]);
    }
@Override
public void endElement(String uri, String localName,
                  String qName) throws SAXException {
      System.out.println("Тег разобран: "+qName);
}
```

```
 @Override
    public void startDocument() throws SAXException {
        System.out.println("Начало разбора документа!");
        super.startDocument();
    }
    @Override
    public void endDocument() throws SAXException {
        super.endDocument();
        System.out.println("Разбор документа окончен!");
    }
}
--------------
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser;
InputStream xmlData = null;
xmlData = new FileInputStream("1.xml");
parser = factory.newSAXParser();
parser.parse(xmlData, new MyParser());
```

Главный недостаток - сложный код в случае сложной структуре XML файла. То есть если XML простой и линейный, то его легко анализировать SAX-парсером. Для XML со сложной структурой придется по возиться с алгоритмизацией.

Генерация XML

```
DocumentBuilderFactory f =
                    DocumentBuilderFactory.newInstance();
DocumentBuilder builder = f.newDocumentBuilder();
Document d = builder.newDocument();
Element root = d.createElement("root");
Element child = d.createElement("child");
Text tn = d.createTextNode("someText");
d.appendChild(root);
root.appendChild(child);
child.appendChild(tn);
child.setAttribute("myatr","myValue");
Transformer t =
         TransformerFactory.newInstance().newTransformer();
t.transform(new DOMSource(d),
      new StreamResult(
                new FileOutputStream(new File("2.xml"))));
```

JAXB

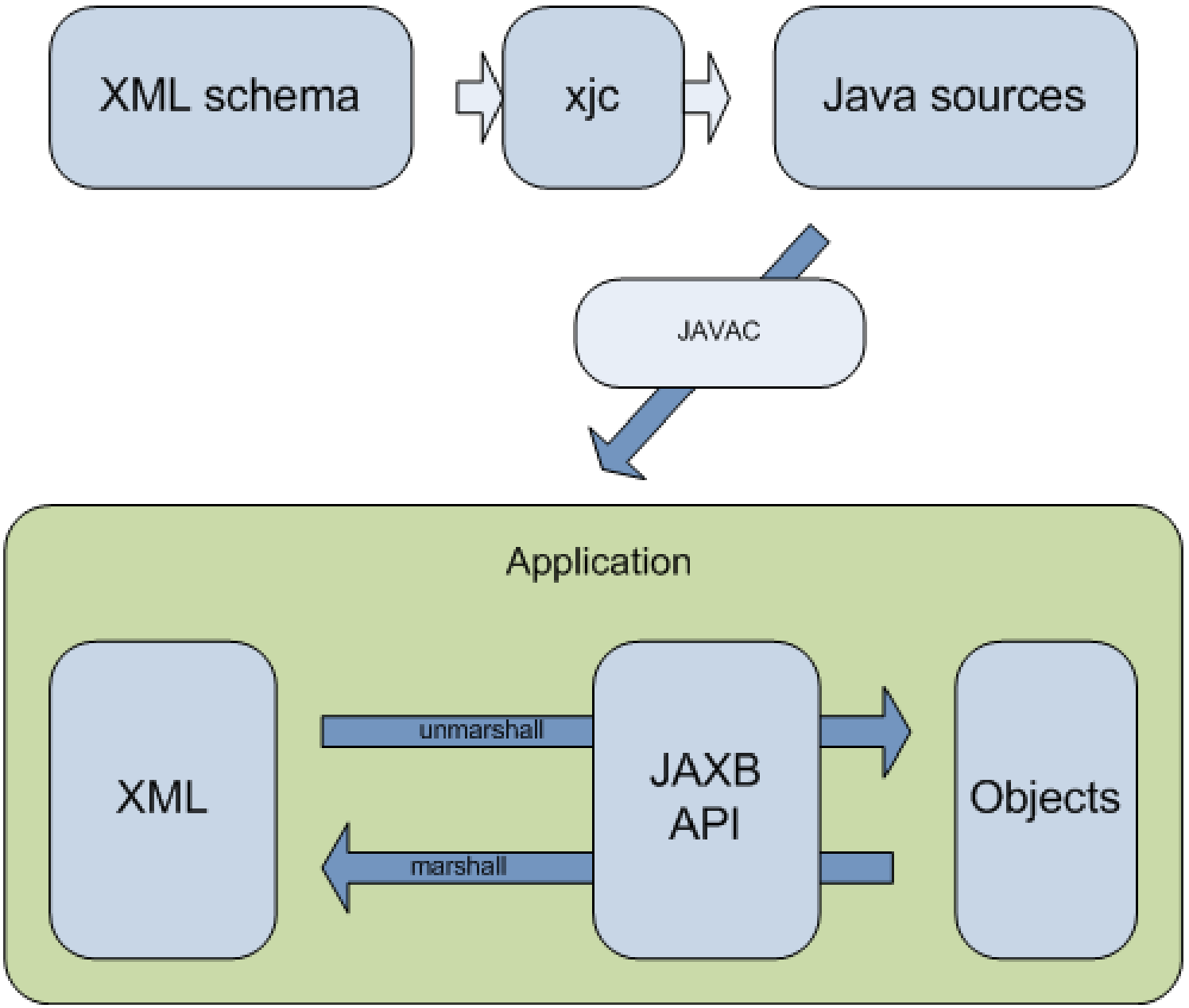
Маршаллизация - это процесс преобразования находящихся в памяти данных в формат их хранения. Так, для технологий Java и XML, маршаллизация представляет собой преобразование некоторого набора Java-объектов в XML-документ

Демаршаллизация - это процесс преобразования данных из формата среды хранения в память, т.е. процесс, прямо противоположный маршаллизации. Иначе говоря, вы можете демаршиллизировать XML-документ в Java VM.

Кругооборот данных - для технологий Java и XML это означает перемещение данных из XML-документа в экземпляры переменных Java и обратно в XML-документ. Корректный кругооборот данных требует идентичности исходных и полученных XML-документов в предположении, что данные во время этой операции не менялись.

Java Architecture for XML Binding (**JAXB**) позволяет Java разработчикам ставить в соответствие Java классы и XML представления. JAXB предоставляет две основные возможности: *маршаллирование* Java объектов в XML и наоборот, то есть *демаршализация* из XML обратно в Java объект. Другими словами, JAXB позволяет хранить и извлекать данные в памяти в любом XML-формате, без необходимости выполнения определенного набора процедур загрузки и сохранения XML.

JAXB


XML schema
xjc
Java sources
JAVAC
Application
XML
unmarshall
marshall
JAXB
API
Objects

JAXB

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ncedu.ru"
 xmlns:tns= "http://ncedu.ru"
elementFormDefault="qualified">
<xs:element name="student">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="fio" type="xs:string"/>
            <xs:element name="age" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

JAXB

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "fio",
    "age"
})
@XmlRootElement(name = "student")
public class Student {

    @XmlElement(required = true)
    protected String fio;
    protected int age;
    public String getFio() {return fio;}
    public String toString() {return fio;}
    public void setFio(String value) {this.fio = value;}
    public int getAge() {return age;}
    public void setAge(int value) {this.age = value;}

}
```

JAXB

```
JAXBContext jc = JAXBContext.newInstance(Student.class);
Marshaller m = jc.createMarshaller();
Unmarshaller um = jc.createUnmarshaller();
Student s = (Student)um.unmarshal(new File("f1.xml"));
System.out.println(s.getFio());
s.setFio("Sidorov");
m.marshal(s, new File("f2.xml "));
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:student xmlns:ns1="http://ncedu.ru">
    <fio>Petrov</fio>
    <age>22</age>
</ns1:student>
```

JAXB

```
@XmlType(propOrder = { "fio", "birthDate" }, name = "student")
public class Student {
    private Date birthDate;
    private String fio;
    public Date getBirthDate() {
        return birthDate;
    }
    public void setBirthDate(Date birthDate) {
        this.birthDate = birthDate;
    }
    public String getFio() {
        return fio;
    }

    public void setFio(String fio) {
        this.fio = fio;
    }
}
```

JAXB

```
@XmlAccessorType(XmlAccessType.NONE)
@XmlType(propOrder = { "groupName", "students" }, name = "group")
@XmlRootElement
public class StudentGroup {
    @XmlElement(name = "gn")
    private String groupName;
    @XmlElementWrapper(name = "oursrudents")
    @XmlElement(name = "student")
    private List<Student> students = new ArrayList<Student>();

    private int notUsedField;



    public void setGroupName(String name) {
        this.groupName = name;
    }

    public void addStudent(Student s){
        students.add(s);
    }
}
```

JAXB

```
Student s1 = new Student();
s1.setBirthDate(format.parse("1990.01.01"));
s1.setFio("Ivanov");
Student s2 = new Student();
s2.setBirthDate(format.parse("1990.01.01"));
s2.setFio("Petrov");
StudentGroup sg = new StudentGroup();
sg.setGroupName("Group1");
sg.addStudent(s1);
sg.addStudent(s2);
JAXBContext jc = JAXBContext.newInstance(StudentGroup.class);
Marshaller m = jc.createMarshaller();
m.marshal(sg, new File("sg.xml"));
Unmarshaller um = jc.createUnmarshaller();
StudentGroup sg2 =(StudentGroup)um.unmarshal(new File("sg.xml"));
```

JAXB – Основные аннотации

- Пакет javax.xml.bind.annotation содержит разнообразнейшие аннотации, описывающие параметры маршалинга и анмаршалинга

- **`@XmlRootElement`**
  Обозначает корневой элемент сохраняемой структуры

- **`@XmlElement`**
  Обозначает поля и свойства (для JavaBeans)

- **`@XmlTransient`**
  Обозначает то, что поле не будет сохраняться

@XmlAccessorType – указывает с чем будем работать:

XmlAccessType.FIELD - работа ведётся с полями класса напрямую. Все атрибуты, кроме аннотированного как @Transient, попадают в XML-представление

XmlAccessType.NONE - будут использоваться непосредственные аннотации на атрибутах.

XmlAccessType.PROPERTY - будут использоваться property от JavaBeans.

http://dev64.wordpress.com/2012/05/15/using-annotations-with-jaxb/
https://netbeans.org/kb/docs/websvc/jaxb_ru.html
http://www.oracle.com/technetwork/java/jaxb-141136.html

Домашнее задание

1. Разработать программу, позволяющую вводить/удалять/редактировать информацию о студентах(ФИО, дата рождения, и еще что-нибудь). В конечном итоге должно вся информация должна сохраняться в файле XML. Нужно сделать два варианта программы: Первый с использованием DOM/SAX. Второй с JAXB.
2. * разработать механизм сохранения объекта в XML. Будем считать, что в качестве полей объекта служат примитивы(+String), поля, если необходимо, могут быть помечены некоторыми аннотациями(их разработать самостоятельно)
3. * Разработать механизм сохранения и восстановления форм в/из XML