

2018

# Программирование на Java JD02 / № 08

## Введение в Java

**Александр  
Хмелев**



# ОРИЕНТИРОВОЧНЫЙ ГРАФИК JD02 (40 часов)

- |                                       |                              |
|---------------------------------------|------------------------------|
| 1. Потоки выполнения часть 1          | (4 часа)                     |
| 2. Потоки выполнения часть 2          | (4 часа)                     |
| 3. Практика по потокам                | (4 часа)                     |
| 4. Регулярные выражения (часть 2)     | (4 часа)                     |
| 5. Интернационализация в Java         | (4 часа)                     |
| 6. Шаблоны проектирования (S.O.L.I.D) | (4 часа)                     |
| 7. XML, XSD                           | (4 часа)                     |
| 8. Парсеры (DOM, SAX, StAX)           | <b>(4 часа -2 на проект)</b> |
| 9. XSL, XSLT/XPath                    | <b>(4 часа -2 на проект)</b> |
| 10. JAXB+Serializable+JSON            | <b>(4 часа)</b>              |

# JAXP — Java API for XML Processing

- ✎ **Парсер** — это библиотека, которая читает XML-документ и содержит набор методов для обработки информации из этого документа. Три основных:
- ✎ **DOM** (Document Object Model — объектная модель документов) — библиотека управления HTML и XML
  1. Нет ограничений на структуру.
  2. Создает **дерево** объектов **в памяти (расход велик)**
  3. Узел - элемент, атрибут, текстовый, графический или любой другой объект.
  4. Узлы связаны между собой отношениями родитель-потомок.
  5. Пишет и читает.
- ✎ **SAX** (Simple API for XML) – автоматический итератор.
  1. Не создает дерево объектов, **в памяти только узел.**
  2. Двигается **самостоятельно** от узла к узлу и только вперед.
  3. Вызывает методы, реализующие **интерфейсы** SAX-парсера.
  4. Только читает.
- ✎ **StAX** (Streaming API for XML) – вызываемый итератор.
  1. Не создает дерево объектов **в памяти только узел.**
  2. Движение по XML - программное **hasNext + next()**. Только вперед.
  3. Пишет и читает. Точнее: или пишет, или читает.

# SAX API определяет ряд интерфейсов.

➤ Пример: **org.xml.sax.ContentHandler** имеет методы:

- void **startDocument()** старте обработки документа;
- void **endDocument()** завершение разбора документа;
- void **startElement**(String uri, String localName, String **qName**, Attributes attrs) — пройдены имя и атрибуты;
- void **endElement**(String uri, String localName, String **qName**) — конец тега;
- void **characters**(char[] **ch**, int **start**, int **length**) парсер встретил символьную информацию внутри элемента (тело тега). Может быть вызван более одного раза.

# Порядок запуска SAX

- 1. Создать свой класс, который
  - **реализует** один или несколько интерфейсов (**ContentHandler**, **ErrorHandler**, **DTDHandler**, **EntityResolver**, **DocumentHandler**)
  - или **наследует** класс **org.xml.sax.helpers.DefaultHandler**,
- + реализовать методы, отвечающие за обработку интересующих частей документа или ошибок.
- 2. Создать стандартный парсер, например, фабрикой **XMLReader** `reader = XMLReaderFactory.createXMLReader()`.
- 3. Передать в парсер объект класса, созданного на шаге 1 с помощью соответствующих методов:
  - **setContentHandler()**;
  - **setErrorHandler()**;
  - **setDTDHandler()**;
  - **setEntityResolver()**.
- 4. Вызвать метод **parse**(String filename) класса XMLReader, которому в качестве параметров передать путь (URI) к анализируемому документу либо `InputStream`.

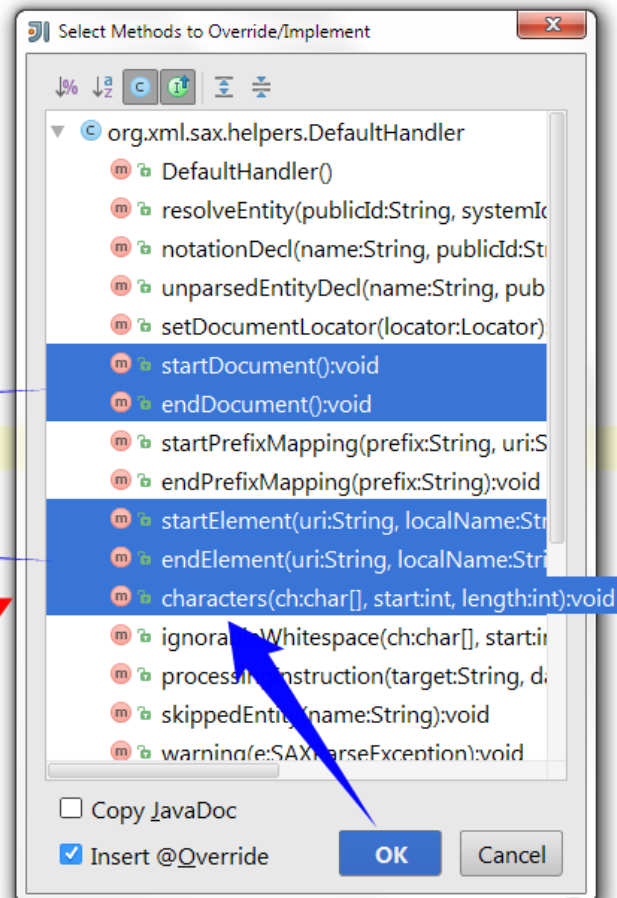
# Собираем SAX- парсер

```
package by.it.akhmelev.JD02_08;

import org.xml.sax.helpers.DefaultHandler;

//
public class Ex01_Sax extends DefaultHandler {
    public static void main(String[] args) {
        //runner
    }
}
```

Перекрываем методы



# Собираем SAX- парсер

```
public class Ex01_Sax extends DefaultHandler {
    public static void main(String[] args) {
        //runner
        try {
            //возьмем пример из предыдущего занятия
            String fileName = "src/by/it/akhmelev/JD02_07/04+XSD.xml";
            //создадим фабрику и стандартный парсер
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            //создадим собственный вариант SAX-класса
            Ex01_Sax myImplementationSax = new Ex01_Sax();
            //пуск парсера
            parser.parse(new File(fileName), myImplementationSax);
        } catch (Exception e) {
            System.out.print("Ошибка! " + e.toString());
        }
    }

    //наша реализация методов DefaultHandler
    private String tab = "";
    private String value;
    @Override
    public void startDocument() throws SAXException {System.out.println("Начало обработки");}
    @Override
    public void endDocument() throws SAXException {System.out.println("Конец обработки");}
```

# Собираем SAX- парсер

```
@Override //тут печатаем начало тега (и его атрибуты в цикле)
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    System.out.print(tab + "[" + qName);
    for (int i = 0; i < attributes.getLength(); i++) {
        String name = attributes.getLocalName(i);
        String value = attributes.getValue(i);
        System.out.print(" " + name + "=" + value);
    }
    System.out.println("]");
    tab = '\t' + tab; //добавим табулятор
    value = "";
}

@Override //печатаем конец тега
public void endElement(String uri, String localName, String qName) throws SAXException {
    if (!value.isEmpty())
        System.out.println(tab + value);
    value = "";
    tab = tab.substring(1); //уберем табулятор
    System.out.println(tab + "[/" + qName + "]");
}

@Override //а здесь собираем из кусочков value. Обрывы будут на ' " &
public void characters(char[] ch, int start, int length) throws SAXException {
    value = value.concat(new String(ch, start, length).trim());
}
```





# Собираем SAX-парсер (все вместе)

```
public class Ex01_Sax extends DefaultHandler {
    public static void main(String[] args) {
        //runner
        try {
            //возьмем пример из предыдущего занятия
            String fileName = "src/by/it/akhmelev/JD02_07/04/XSD.xml";
            //создадим фабрику и стандартный парсер
            SAXParserFactory factory = SAXParserFactory.newInstance();
            SAXParser parser = factory.newSAXParser();
            //создадим собственный вариант SAX-класса
            Ex01_Sax myImplementationSax = new Ex01_Sax();
            //пуск парсера
            parser.parse(new File(fileName), myImplementationSax);
        } catch (Exception e) {
            System.out.print("Ошибка! " + e.toString());
        }
    }
}
```

```
//наша реализация методов DefaultHandler
private String tab = "";
private String value;

@Override
public void startDocument() throws SAXException {System.out.println("Начало обработки");}

@Override
public void endDocument() throws SAXException {System.out.println("Конец обработки");}
```

```
@Override //здесь печатаем начало тега (и его атрибуты в цикле)
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    System.out.print(tab + "[" + qName);
    for (int i = 0; i < attributes.getLength(); i++) {
        String name = attributes.getLocalName(i);
        String value = attributes.getValue(i);
        System.out.print(" " + name + "=" + value);
    }
    System.out.println("]");
    tab = '\t' + tab; //добавим табулятор
    value = "";
}

@Override //печатаем конец тега
public void endElement(String uri, String localName, String qName) throws SAXException {
    if (!value.isEmpty())
        System.out.println(tab + value);
    value = "";
    tab = tab.substring(1); //уберем табулятор
    System.out.println(tab + "[/" + qName + "]\n");
}

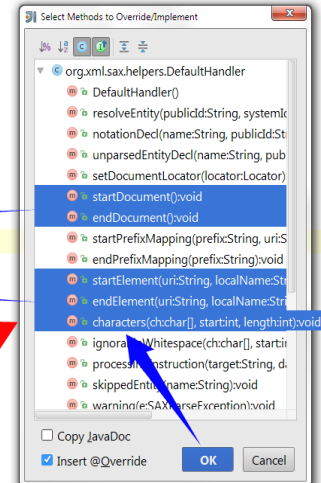
@Override //а здесь собираем из кусочков value. Обрывы будут на ' ' &
public void characters(char[] ch, int start, int length) throws SAXException {
    value = value.concat(new String(ch, start, length).trim());
}
```

```
package by.it.akhmelev.JD02_08;

import org.xml.sax.helpers.DefaultHandler;
```

```
//
public class Ex01_Sax extends DefaultHandler {
    public static void main(String[] args) {
        //runner
    }
}
```

Перекрываем методы



```
Ex01_Sax
D:\Java\jdk1.8.0_73\bin\java ...
Начало обработки
[students xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xmlns=http://
  [student login=IvanovMitarAlex7 course=it.jd.02]
    [name]
      Ivanov Alexander
    [/name]
    [nickname]
      "M&M's"
    [/nickname]
    [telephone]
      4586954
    [/telephone]
    [address]
      [country]
        Belarus
      [/country]
      [city]
        Minsk
      [/city]
      [street]
        Lenina 3
      [/street]
    [/address]
  [/student]
[student login=zPav16 course=it.jd.02]
  [name]
```

# Как работает StAX

- StAX (Streaming API for XML) - pull-парсер, где не требуется реализация интерфейсов.
  - приложение само указывает StAX-парсеру перейти к следующему элементу XML.
  - есть API для создания XML-документа.
- Классами StAX являются **XMLInputFactory**, **XMLStreamReader**, **XMLOutputFactory**, **XMLStreamWriter**, которые, соответственно, используются для чтения и создания XML-документа и расположены в пакете `javax.xml.stream`.
- Для чтения XML требуется получить ссылку на экземпляр `XMLStreamReader`:
  - `StringReader input = new StringReader(fileName); // из пакета java.io`
  - Или `InputStream input = new FileInputStream(new File(fileName));`
  - далее
  - `XMLInputFactory inputFactory = XMLInputFactory.newInstance();`
  - `XMLStreamReader reader = inputFactory.createXMLStreamReader(input);`
- по экземпляру `XMLStreamReader` работает Iterator с **hasNext()** и **next()**:
- При попытке вызове на типе **int type=next()** несоответствующего метода генерируется исключительная ситуация `IllegalStateException`.
- Чаще всего данные извлекаются с применением методов:
  - `String getLocalName()` — возвращает название тега (элемента) для текущей константы;
  - `String getText()` — возвращает текст для констант `CHARACTERS`, `CDATA`, `COMMENT` и др.
  - `String getAttributeValue(String namespaceURI, String localName)` — возвращает значение атрибута по имени;
  - `String getAttributeValue(int index)` — возвращает значение атрибута по номеру позиции;

# Возможные значения типа `int type=next()`;

START_ELEMENT	<code>next()</code> , <code>getName()</code> , <code>getLocalName()</code> , <code>hasName()</code> , <code>getPrefix()</code> , <code>getAttributeXXX()</code> , <code>isAttributeSpecified()</code> , <code>getNamespaceXXX()</code> , <code>getElementText()</code> , <code>nextTag()</code>
ATTRIBUTE	<code>next()</code> , <code>nextTag()</code> <code>getAttributeXXX()</code> , <code>isAttributeSpecified()</code> ,
NAMESPACE	<code>next()</code> , <code>nextTag()</code> <code>getNamespaceXXX()</code>
END_ELEMENT	<code>next()</code> , <code>getName()</code> , <code>getLocalName()</code> , <code>hasName()</code> , <code>getPrefix()</code> , <code>getNamespaceXXX()</code> , <code>nextTag()</code>
CHARACTERS	<code>next()</code> , <code>getTextXXX()</code> , <code>nextTag()</code>
CDATA	<code>next()</code> , <code>getTextXXX()</code> , <code>nextTag()</code>
COMMENT	<code>next()</code> , <code>getTextXXX()</code> , <code>nextTag()</code>
SPACE	<code>next()</code> , <code>getTextXXX()</code> , <code>nextTag()</code>
START_DOCUMENT	<code>next()</code> , <code>getEncoding()</code> , <code>getVersion()</code> , <code>isStandalone()</code> , <code>standaloneSet()</code> , <code>getCharacterEncodingScheme()</code> , <code>nextTag()</code>
END_DOCUMENT	<code>close()</code>
PROCESSING_INSTRUCTION	<code>next()</code> , <code>getPITarget()</code> , <code>getPIData()</code> , <code>nextTag()</code>
ENTITY_REFERENCE	<code>next()</code> , <code>getLocalName()</code> , <code>getText()</code> , <code>nextTag()</code>

# StAX пример

```
public class Ex02_StAX {
    static String tab="";
    public static void main(String[] args) {
        //runner
        try {
            //возьмем пример из предыдущего занятия
            String fileName = "src/by/it/akhmelev/JD02_07/04+XSD.xml";
            //создадим фабрику и стандартный парсер
            FileInputStream input=new FileInputStream(fileName);
            XMLInputFactory inputFactory = XMLInputFactory.newInstance();
            XMLStreamReader reader = inputFactory.createXMLStreamReader(input);
            //пуск парсера
            String el="";
            while (reader.hasNext()) {
                int type=reader.next();
                switch (type) {
                    case XMLStreamConstants.START_ELEMENT:
                    {
                        System.out.println(tab+"[" + reader.getLocalName() + "]" );
                        tab=tab+"\t";
                        break;
                    }
                    case XMLStreamConstants.CHARACTERS:
```



# StAX пример

```
        case XMLStreamConstants.CHARACTERS:
        {
            el=el.concat(reader.getText().trim());
            break;
        }
        case XMLStreamConstants.END_ELEMENT:
        {
            if (!el.isEmpty())
                System.out.println(tab+el);
            tab=tab.substring(1);
            el="";
            System.out.println(tab+"[/\" + reader.getLocalName() + "\"]");
            break;
        }
    }
}

} catch (Exception e) {
    System.out.print("Ошибка! " + e.toString());
}

}

}
```



# Пример парсинга StAX

```
public class Ex02_StAX {
    static String tab="";
    public static void main(String[] args) {
        //runner
        try {
            //возьмем пример из предыдущего занятия
            String fileName = "src/by/it/akhmelev/JD02_07/04+XSD.xml";
            //создадим фабрику и стандартный парсер
            FileInputStream input=new FileInputStream(fileName);
            XMLInputFactory inputFactory = XMLInputFactory.newInstance();
            XMLStreamReader reader = inputFactory.createXMLStreamReader(input);
            //пуск парсера
            String el="";
            while (reader.hasNext()) {
                int type=reader.next();
                switch (type) {
                    case XMLStreamConstants.START_ELEMENT:
                    {
                        System.out.println(tab+"[" + reader.getLocalName() +
                            tab=tab+"\t";
                        break;
                    }
                    case XMLStreamConstants.CHARACTERS:
                    {
```

```
D:\Java\jdk1.8.0_73\bin\java ...
```

```
[students]
  [student]
    [name]
      Ivanov Alexander
    [/name]
    [nickname]
      "M&M's"
    [/nickname]
    [telephone]
      4586954
    [/telephone]
    [address]
```

@ jdk8000

```
        {
            el=el.concat(reader.getText().trim());
            break;
        }
        case XMLStreamConstants.END_ELEMENT:
        {
            if (!el.isEmpty())
                System.out.println(tab+el);
            tab=tab.substring(1);
            el="";
            System.out.println(tab+"[" + reader.getLocalName() + "]" );
            break;
        }
    }

}

} catch (Exception e) {
    System.out.print("Ошибка! " + e.toString());
}

}
```



# Обработка DOM

- **org.w3c.dom.Document** - представляет собой корневой элемент XML-документа и содержит методы доступа ко всему содержимому документа.
  - Element **getDocumentElement()** — возвращает корневой элемент документа.
  
- **org.w3c.dom.Node** - общий элемент дерева:
  - 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;
  - NamedNodeMap **getAttributes()** — возвращает список атрибутов данного элемента.
  
- **org.w3c.dom.Element** элемент XML-документа:
  - String **getTagName()** — 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;
  - NodeList **getElementsByTagName(String name)** — возвращает список дочерних элементов с определенным именем.
  
- **org.w3c.dom.Attr** - атрибуты элемента XML-документа:
  - String **getName()** — имя атрибута;
  - Element **getOwnerElement()** — элемент, который содержит этот атрибут;
  - String **getValue()** — возвращает значение атрибута;
  - boolean **isId()** — проверяет атрибут на тип ID.

# Пример с DOM

```
public static void main(String[] args) {  
    //runner  
    //возьмем пример из предыдущего занятия  
    String fileName = "src/by/it/akhmelev/JD02_07/04+XSD.xml";  
    //создадим фабрику и стандартный парсер  
    DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();  
    try {  
        DocumentBuilder builder=factory.newDocumentBuilder();  
        Document doc=builder.parse(fileName);  
        Element el=doc.getDocumentElement();  
        printDom("", el);  
    } catch (Exception e) {  
        System.out.print("Ошибка! " + e.toString());  
    }  
}
```

```
D:\Java\jdk1.8.0_73\bin\java ...  
students >  
students > student >  
students > student > name > Ivanov Alexander  
students > student >  
students > student > nickname > "M&M's"  
students > student >  
students > student > telephone > 4586954  
students > student >  
students > student > address >
```

```
//Демо. Рекурсивный обход DOM-дерева.  
private static void printDom(String prefix, Node node) {  
    String text=node.getNodeValue();  
    if (text!=null) {  
        System.out.println(prefix + text.trim());  
    }  
  
    NodeList children = node.getChildNodes();  
    for (int i = 0; i < children.getLength(); i++) {  
        printDom(prefix+node.getNodeName() + " > ", children.item(i));  
    }  
}
```



# Вопросы



**XSL** (e**X**tensible **S**tylesheet **L**anguage) — семейство рекомендаций консорциума **W3C**, описывающее языки преобразования и визуализации **XML**-документов. Состоит из трех частей:

- XSL Transformations (**XSLT**) — язык преобразований XML-документов.
- XSL Formatting Objects (**XSL-FO**) — язык разметки типографских макетов и иных предпечатных материалов.
- **XPath** — язык путей и выражений, используемый в XSLT для доступа к отдельным частям XML-документа.

- Технология XSL/XSLT позволяет преобразовывать XML файлы в другой XML или любой другой формат, например, HTML, чистый текст, rtf и т.д.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

```
<xsl:output method="html"/>
```

- XML состоит из тегов, каждый из которых является шаблоном преобразования.

- Общий вид такого тега:

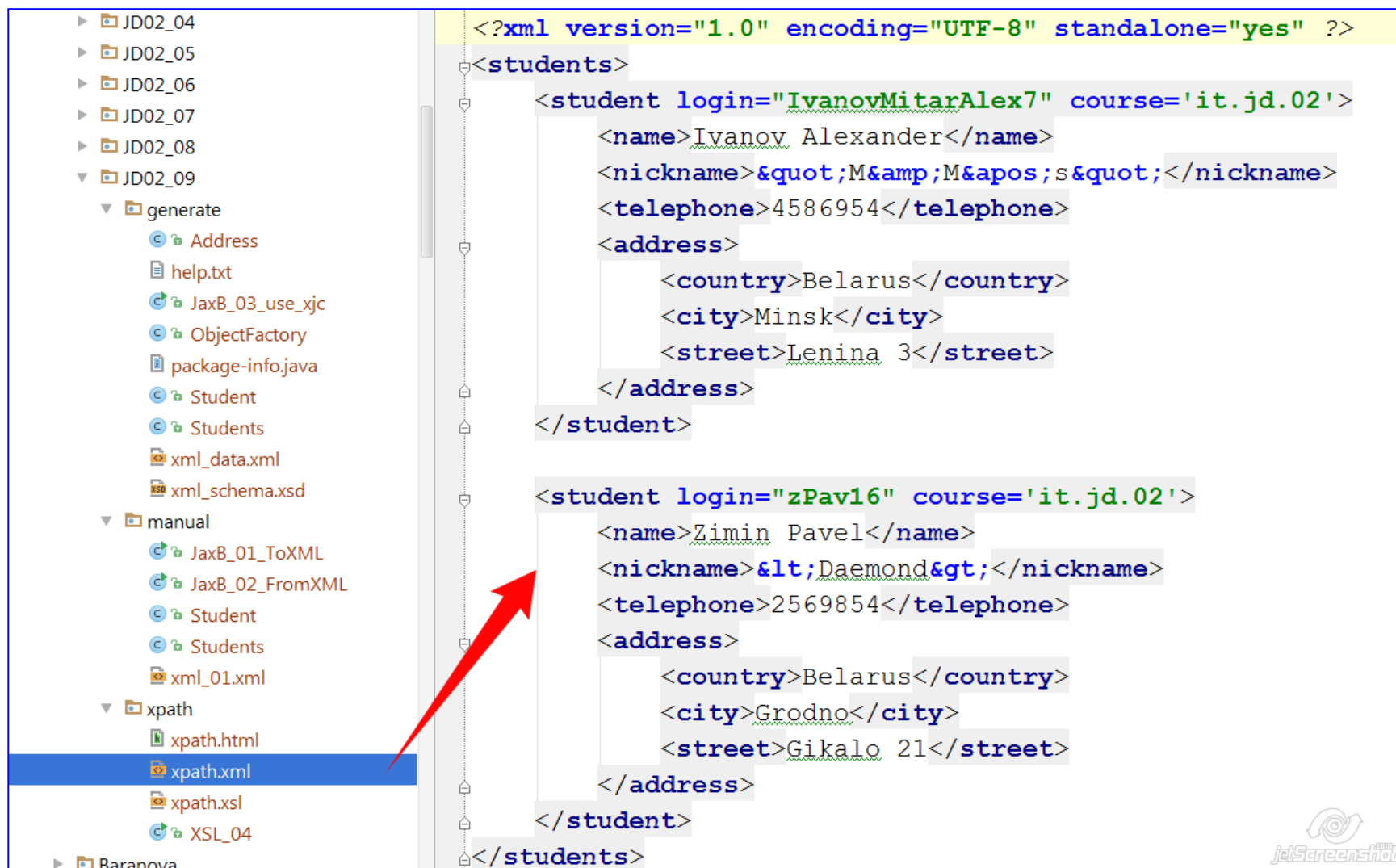
```
<xsl:template match="путь_к_элементу">
```

**Новое содержимое**

```
</xsl:template>
```

- Параметр *match* содержит путь к элементу, который будет заменяться новым содержимым.
- Допустим, мы хотим сделать из нашего списка студентов таблицу HTML. Это значит, что нам надо заменить корневой тег **students** на тег **<table>**. Параметр **match** примет значение **"/ students"**.

# Исходный XML



```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<students>

  <student login="IvanovMitarAlex7" course='it.jd.02'>
    <name>Ivanov Alexander</name>
    <nickname>&quot;M&amp;M&apos;s&quot;</nickname>
    <telephone>4586954</telephone>
    <address>
      <country>Belarus</country>
      <city>Minsk</city>
      <street>Lenina 3</street>
    </address>
  </student>

  <student login="zPav16" course='it.jd.02'>
    <name>Zimin Pavel</name>
    <nickname>&lt;Daemond&gt;</nickname>
    <telephone>2569854</telephone>
    <address>
      <country>Belarus</country>
      <city>Grodno</city>
      <street>Gikalo 21</street>
    </address>
  </student>

</students>
```

# Пример XSL файла

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/students">
    <table border="1">
      <tr><td>Name</td><td>NickName</td><td>Phone</td><td>Address</td></tr>
      <xsl:apply-templates/>
    </table>
  </xsl:template>

  <xsl:template match="/students/student">
    <tr><xsl:apply-templates/></tr>
  </xsl:template>

  <xsl:template match="/students/student/name">
    <td><xsl:apply-templates/></td>
  </xsl:template>

  <xsl:template match="/students/student/nickname">
    <td><xsl:apply-templates/></td>
  </xsl:template>

  <xsl:template match="/students/student/telephone">
    <td><xsl:apply-templates/></td>
  </xsl:template>

  <xsl:template match="/students/student/address">
    <td><xsl:apply-templates/></td>
  </xsl:template>
</xsl:stylesheet>
```



# Код транслятора

```
package by.it.akhmelev.JD02_09.xpath;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class XSL_04 {
    final static String root="src/by/it/akhmelev/JD02_09/xpath/";
    public static void main(String[] args) {
        String fileName=root+"xml_data.xml";
        try {
            TransformerFactory tf = TransformerFactory.newInstance();
            // установка используемого XSL-преобразования
            Transformer transformer = tf.newTransformer(new StreamSource(root+"xpath.xsl"));
            // установка исходного XML-документа и конечного XML-файла
            transformer.transform(new StreamSource(root+"xpath.xml"),
                                new StreamResult(root+"xpath.html"));
            System.out.println("Transform " + fileName + " complete");
        } catch (TransformerException e) {
            System.err.println("Impossible transform file " + fileName + " : " + e);
        }
    }
}
```



# Результат конвертации

The screenshot shows an IDE with a project named JD2016\_02\_20\_11-18. The file explorer on the left shows a directory structure with files like Address, help.txt, JaxB\_03\_use\_xjc, ObjectFactory, package-info.java, Student, Students, xml\_data.xml, xml\_schema.xsd, JaxB\_01\_ToXML, JaxB\_02\_FromXML, xml\_01.xml, xpath.html, xpath.xml, xpath.xsl, and XSL\_04. The main editor displays the content of xpath.html, which is an XML snippet representing a table:

```
table|  |
| --- |
| <table border="1"> |
| <td>Name</td><td>NickName</td><td>Phone</td><td>Address</td> |
| <td>Ivanov Alexander</td> |
| <td>"M&M's"</td> |
| <td>4586954</td> |
| <td> |
| Belarus |
| Minsk |
| Lenina 3 |
| </td> |
| </tr> |

```

Two red arrows point from the XML code to a browser window. The browser window shows the rendered HTML table:

Name	NickName	Phone	Address
Ivanov Alexander	"M&M's"	4586954	Belarus Minsk Lenina 3
Zimin Pavel	<Daemond>	2569854	Belarus Grodno Gikalo 21

The browser window title is 'xpath.html' and the address bar shows 'file:///C:/JavaProjects/JD2016\_02\_20\_11-18/src/by/it/akhmelev/'. The status bar at the bottom indicates 'Compilation completed successfully in 2 sec (15 minutes ago)' and '9:29 CRLF UTF-8 Git: master'.

# Хрath доступен и из Java

```
import org.w3c.dom.Node;
import org.xml.sax.InputSource;

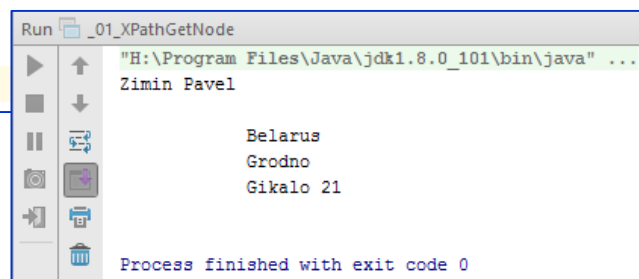
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

public class _01_XPathGetNode {

    //прямой доступ к узлам через XPath
    private static InputSource inputSource = new InputSource("src/by/it/akhmelev/jd02_09/xpath/xpath.xml");
    private static XPath xpath = XPathFactory.newInstance().newXPath();

    private static String getText(String expression) throws XPathExpressionException {
        Node node = (Node) xpath.evaluate(expression, inputSource, XPathConstants.NODE);
        return node.getTextContent();
    }

    public static void main(String[] args) throws XPathExpressionException {
        String expression = "/students/student[@login=\"zPav16\"]/name";
        System.out.println(getText(expression));
        expression = "/students/student[@login=\"zPav16\"]/address";
        System.out.println(getText(expression));
    }
}
```



```
Run _01_XPathGetNode
"H:\Program Files\Java\jdk1.8.0_101\bin\java" ...
Zimin Pavel

Belarus
Grodno
Gikalo 21

Process finished with exit code 0
```





# Вопросы



➤ **SAX**

➤ <http://www.quizful.net/post/sax-parser-java>

➤ **StAX**

➤ <https://www.ibm.com/developerworks/ru/library/x-stax1/>

➤ **DOM**

➤ [http://www.tutorialspoint.com/java\\_xml/java\\_dom\\_parse\\_document.htm](http://www.tutorialspoint.com/java_xml/java_dom_parse_document.htm)

➤ **XSLT (with XPath)**

➤ <http://www.codenet.ru/webmast/xml/xslt/w3c.php>

➤ **XPath (from Java)**

➤ <https://examples.javacodegeeks.com/core-java/xml/xpath/xpath-search-id-example/>

## Задание по теме JAXP

### ➤ Вариант А. *SAX и StAX*

- Выполните по аналогии с **Заданием 10** или **11** из рабочей тетради – парсинг **XML** для своей предметной области (**используйте свой XML-файл из предыдущего задания**). Нужно сделать два парсера **SAX** и **StAX** + вывод на экран названий тегов и их содержимого
- (**используйте свой XML-файл без схемы и DTD из предыдущих заданий**).

### ➤ Вариант В. *Доработка SAX StAX + реализация XSLT*

- Доработайте парсинг **SAX** и **StAX**, чтобы работала также и обработка атрибутов.
- Выполните по аналогии с **Заданием 12** из рабочей тетради – преобразование **XML** в HTML файл с таблицей, в которой каждая отдельная строка это одна из повторяющихся сущностей XML.

### ➤ Вариант С. *DOM*

- Реализуйте парсинг **DOM**. Во всех случаях **выводите атрибуты**. XML на экране должен быть читабелен и одинаков для всех трех способов **SAX StAX DOM**.

**Видео:** <https://youtu.be/1L3CJluoXoQ>