```
    <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.

Пусть дан XML-документ с описанием медиатехники.

```
<?xml version="1.0" encoding="UTF-8"?>
<products xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation=" products.xsd">
      <category name="Audio And Video">
            <subcategory name="Audio">
                  <product>
                        <producer>Samsung</producer>
                        <model>NV678</model>
                        <year>12-12-2006</year>
                        <color>White</color>
                        <notAvailable />
                  </product>
            </subcategory>
            <subcategory name="Video">
                  <product>
                        <producer>Samsung</producer>
                        <model>VH500</model>
                        <year>12-12-2004</year>
                        <color>Black</color>
                        <cost>200</cost>
                  </product>
                  <product>
                        <producer>Samsung</producer>
                        <model>VH500</model>
                        <year>12-12-2004</year>
                        <color>White</color>
                        <notAvailable />
                  </product>
            </subcategory>
      </category>
      <category name="Computers">
            <subcategory name="Pocket">
                  <product>
                        <producer>HP</producer>
                        <model>rx371</model>
                        <year>31-01-2006</year>
                        <color>Black</color>
                        <notAvailable />
                  </product>
            </subcategory>
      </category>
</products>
```

Организация процесса разбора документа XML с помощью StAX приведена в следующем примере:

```
/* пример # 12 : реализация разбора XM-документа : StAXProductParser.java :
ProductParser.java: ParserEnum.java */
package chapt16;
```

```
public enum ParserEnum {
      PRODUCTS, CATEGORY, SUBCATEGORY, PRODUCT, PRODUCER,
MODEL, YEAR, COLOR, NOTAVAILABLE, COST, NAME
}
package chapt16;
import java.io.InputStream;

public abstract class ProductParser {
      public abstract void parse(InputStream input);

      public void writeTitle() {
            System.out.println("Products:");
      }
      public void writeCategoryStart(String name) {
            System.out.println("Category: " + name.trim());
      }
      public void writeCategoryEnd() {
            System.out.println();
      }
      public void writeSubcategoryStart(String name) {
         System.out.println("Subcategory: " + name.trim());
      }
      public void writeSubcategoryEnd() {
            System.out.println();
      }
      public void writeProductStart() {
            System.out.println("  Product Start  ");
      }
      public void writeProductEnd() {
            System.out.println("   Product End   ");
      }
      public void writeProductFeatureStart(String name) {
         switch (ParserEnum.valueOf(name.toUpperCase())) {
            case PRODUCER:
                  System.out.print("Provider: ");
                  break;
            case MODEL:
                  System.out.print("Model: ");
                  break;
            case YEAR:
                  System.out.print("Date of issue: ");
                  break;
            case COLOR:
                  System.out.print("Color: ");
                  break;
            case NOTAVAILABLE:
                  System.out.print("Not available");
                  break;
```

```
            case COST:
                  System.out.print("Cost: ");
                  break;
            }
         }
      public void writeProductFeatureEnd() {
            System.out.println();
      }

      public void writeText(String text) {
            System.out.print(text.trim());
      }
}
package chapt16;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import java.io.InputStream;

public class StAXProductParser extends ProductParser {
      // реализация абстрактного метода из суперкласса для разбора потока
      public void parse(InputStream input) {
            XMLInputFactory inputFactory =
                        XMLInputFactory.newInstance();
            try {
               XMLStreamReader reader =
                 inputFactory.createXMLStreamReader(input);
               process(reader);
            } catch (XMLStreamException e) {
               e.printStackTrace();
            }
      }
      // метод, управляющий разбором потока
      public void process(XMLStreamReader reader)
                              throws XMLStreamException {
        String name;

        while (reader.hasNext()) {
       // определение типа "прочтённого" элемента (тега)
          int type = reader.next();

          switch (type) {
            case XMLStreamConstants.START_ELEMENT:
              name = reader.getLocalName();

      switch (ParserEnum.valueOf(name.toUpperCase())) {
            case PRODUCTS:
```

```
                writeTitle();
                break;
          case CATEGORY:
    writeCategoryStart(reader.getAttributeValue(null,
          ParserEnum.NAME.name().toLowerCase()));
          break;
          case SUBCATEGORY:
    writeSubcategoryStart(reader.getAttributeValue(null,
          ParserEnum.NAME.name().toLowerCase()));
          break;
          case PRODUCT:
                writeProductStart();
                break;
          default:
                writeProductFeatureStart(name);
                break;
    }
    break;

          case XMLStreamConstants.END_ELEMENT:
                name = reader.getLocalName();

    switch (ParserEnum.valueOf(name.toUpperCase())) {
          case CATEGORY:
                writeCategoryEnd();
                break;
          case SUBCATEGORY:
                writeSubcategoryEnd();
                break;
          case PRODUCT:
                writeProductEnd();
                break;
          default:
                writeProductFeatureEnd();
                break;
    }
    break;

          case XMLStreamConstants.CHARACTERS:
                writeText(reader.getText());
                break;

          default:
                break;
          }
      }
    }
}
```

Для запуска приложения разбора документа с помощью StAX ниже приведен достаточно простой код:

```
/* пример # 13 : запуск приложения : StreamOutputExample.java*/
package chapt16;
import java.io.FileInputStream;
import java.io.InputStream;

public class StreamOutputExample {
  public static void main(String[] args) throws Exception {
      ProductParser parser = new StAXProductParser();
      // создание входного потока данных из xml-файла
      InputStream input =
            new FileInputStream("chapt16\\mediatech.xml");
      // разбор файла с выводом результата на консоль
      parser.parse(input);
  }
}
```

XSL

Документ XML используется для представления информации в виде некоторой структуры, но он никоим образом не указывает, как его отображать. Для того чтобы просмотреть XML-документ, нужно его каким-то образом отформатировать. Инструкции форматирования XML-документов формируются в так называемые таблицы стилей, и для просмотра документа нужно обработать XML-файл согласно этим инструкциям.

Существует два стандарта стилевых таблиц, опубликованных W3C. Это CSS (Cascading Stylesheet) и XSL (XML Stylesheet Language).

CSS изначально разрабатывался для HTML и представляет из себя набор инструкций, которые указывают браузеру, какой шрифт, размер, цвет использовать для отображения элементов HTML-документа.

XSL более современен, чем CSS, потому что используется для преобразования XML-документа перед отображением. Так, используя XSL, можно построить оглавление для XML-документа, представляющего книгу.

Вообще XSL можно разделить на три части: XSLT (XSL Transformation), XPath и XSLFO (XSL Formatting Objects).

XSL Processor необходим для преобразования XML-документа согласно инструкциям, находящимся в файле таблицы стилей.

XSLT

Этот язык для описания преобразований XML-документа применяется не только для приведения XML-документов к некоторому “читаемому” виду, но и для изменения структуры XML-документа.

К примеру, XSLT можно использовать для:

- удаления существующих или добавления новых элементов в XML-документ;
- создания нового XML-документа на основании заданного;
- извлечения информации из XML-документа с разной степенью детализации;