Wykład 9

DOM

SAX

JAXB, XMLDecoder i XMLEncoder

ANT

Czym jest XML?

`XML` — (Extensible Markup Language, rozszerzalny język znaczników) – to format dokumentów przeznaczony do reprezentowania różnych danych w strukturalizowany sposób. Używany do przechowywania, przesyłania i rekonstrukcji dowolnych danych,między różnymi programami.

> Format zapisu dokumentów, format tekstowy, można ten dokument podejrzeć w dowolnym edytorze tekstowym, zedytować. Ale zajmuje też więcej miejsca. (Json jest trochę bardziej ekonomiczną wersją).

Jest niezależny od platformy, co umożliwia łatwą wymianę dokumentów pomiędzy różnymi systemami. Jest najpopularniejszym obecnie uniwersalnym językiem przeznaczonym do reprezentowania danych.

Po co są takie dokumenty?

Przykład ze sklepem internetowym, aktualizacją danych w sektorze księgowym, wystawianiem automatycznie faktur, aktualizowanie stanu magazynu itp. – ogólnie wzajemna komunikacja między różnymi oprogramowaniami, często napisanymi w różnych językach, może być zastąpiona ujednolicowym systemem *XML*.

Przykład pliku *XML*:

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <rss xmlns:dc="http://purl.org/dc/elements/1.1/" version="2.0">
    <channel>
        <title>Aktualności</title>
        <link>http://www.uj.edu.pl/uniwersytet/aktualnosci/...</link>
        <description>Aktualności Uniwersytetu Jagiellońskiego</description>
        <item>
            <title>Lista Pamięci - Stanisław Szczur</title>
            <link>http://www.uj.edu.pl/uniwersytet/aktualnosci/...</link>
            <description />
            <pubDate>Thu, 20 Dec 2012 07:44:00 GMT</pubDate>
            <dc:creator>Jolanta Herian-Ślusarska</dc:creator>
            <dc:date>2012-12-20T07:44:00Z</dc:date>
        </item>
        <item>...</item>
        ...
    </channel>
</rss>
```

XML dokument ustrukturyzowany w dany sposób, nie działa jak struktura w HTML, która odpowiada danemu układowi elementów na stronie.

DOM

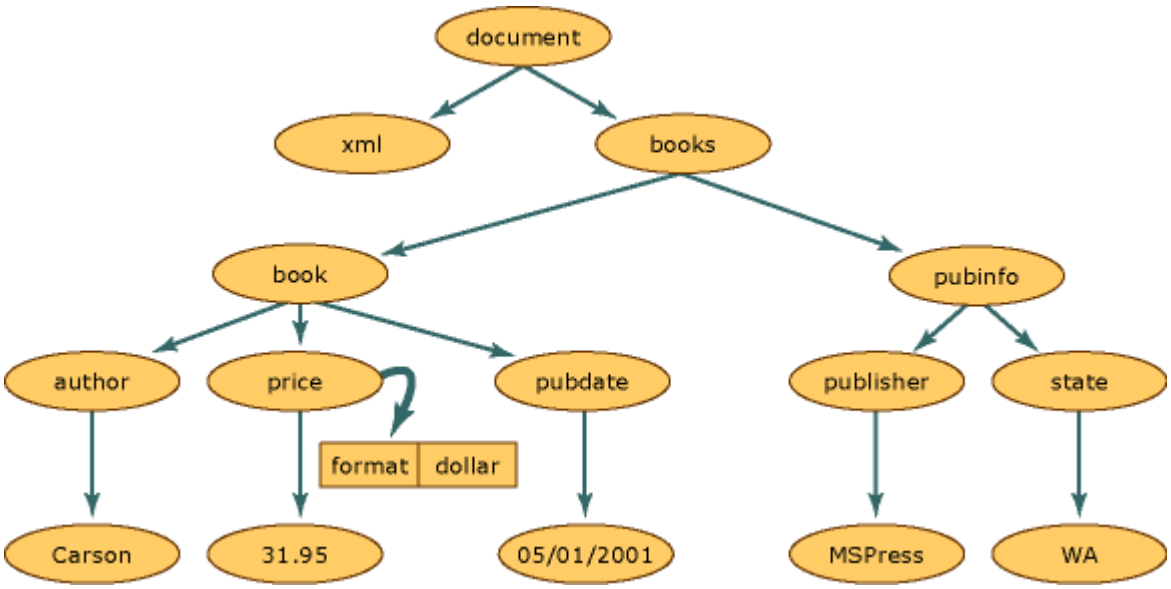
Są dwie podstawowe metody przetwarzania dokumentów xml. Jedną z nich jest `DOM` (Document Object Model).

Parsery DOM (Document Object Model) — program który odczytuje dokument *xml-owy* i na podstawie zawartości dokumentu tworzy drzewo reprezentujące dane w nim zawarte. Po zbudowaniu DOM przetwarzanie odbywa się na modelu w pamięci operacyjnej.

Przykład:

```xml
<?xml version="1.0"?>
  <books>
    <book>
        <author>Carson</author>
        <price format="dollar">31.95</price>
        <pubdate>05/01/2001</pubdate>
    </book>
    <pubinfo>
        <publisher>MSPress</publisher>
        <state>WA</state>
    </pubinfo>
  </books>
```

dane *XML* są odczytane w strukturze *DOM*.



DOM — Odczyt

Jak zrobić to w Javie?

```
import java.io.IOException;
import java.net.URL;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
public class DOMExample {
    public static void main(String[] args) throws
                    ParserConfigurationException,SAXException,IOException {
        URL url = new URL(
        "http://www.uj.edu.pl/uniwersytet/aktualnosci/-/journal/rss/10172/36018?
        doAsGroupId=10172&refererPlid=10181"); // jakis RSS

        // domyslna fabryka “obiektow tworzących dokumenty”
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

        // obiekt “tworzący dokumenty”
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

        // tworzenie modelu DOM na podstawie zrodla XML (parsowanie XML'a)
        Document doc = dBuilder.parse(url.openStream());

        // obiekt doc umozliwia dostep do wszystkich danych zawartych
        // w dokumencie XML
        System.out.println("Root element :" +
                                doc.getDocumentElement().getNodeName());
        NodeList nList = doc.getElementsByTagName("item");

        for (int i = 0; i < nList.getLength(); i++) {

            // lista “dzieci” i-tego itema
            Node n = nList.item(i);

            // czy “dziecko” jest elementem?
            if (n.getNodeType() == Node.ELEMENT_NODE) {
                Element e = (Element) n;
                System.out.println("Tytul : "
                                + getTagValue("title", e));
                System.out.println("Link : "
                                + getTagValue("link", e));
                System.out.println("dodane przez : "
                                + getTagValue("dc:creator", e));
            }
        }
```

```
    }
    // zwraca wartosc zapisana w tagu s wewnatrz elementu e
    private static String getTagValue(String s, Element e) {

        // lista “dzieci” e o nazwie s
        NodeList nl = e.getElementsByTagName(s)
        // pierwszy wpis z tej listy
                                              .item(0)
        // to co on zawiera – jego “dzieci”            pierwsze z dzieci i jedyne
                                                      .getChildNodes();


        // pierwsze z tych dzieci
        Node n = (Node) nl.item(0);

        // zawartosc, ktora tam jest
        return n.getNodeValue();
    }
}
```

`parse()` – służy do parsowania, tworzenia *DOMu* na podstawie danych, które zawierają dokument *XML* i zwraca referencję *doc*, za pomocą której mamy dostęp do drzewa. (tutaj: odczytujemy z sieci, ale moglibyśmy też np. z pliku)

`doc.getDocumentElement()` – zwraca korzeń drzewa
`getNodeName()` – zwraca nazwę korzenia (tutaj: rss)

`NodeList` – Struktura, lista węzłów, podobna strukturalnie do tablicy

`doc.getElementsByTagName(„item”)` – bierzemy wszystkie elementy, które mają nazwę "*item*"

Jak modyfikować takie drzewo?

Możliwa jest także modyfikacja zawartości *DOM* utworzonego w wyniku parsowania dokumentu *XML*.

Przykładowo:

```
// pobieramy element title i zmieniamy go
if ("title".equals(node.getNodeName())) {
    node.setTextContent("Nowy tytul");
}
// kasujemy link
if ("link".equals(node.getNodeName())) {
    itemElement.removeChild(node);  // usuwa wszystkie dzieci elementu node
}

// tworzenie nowego dokumentu
Document doc = docBuilder.newDocument();
Element e = doc.createElement("root");
doc.appendChild(e);
```

DOM — Zapis

> Plik DOM możemy z powrotem zapisać do postaci *XML* lub innej.

Przykład zapisu dokumentu:

```
// domyslna fabryka transformatorow
TransformerFactory transformerFactory = TransformerFactory.newInstance();

// nowy tranformator
Transformer transformer = transformerFactory.newTransformer();

// wejscie transformatora (skad transformator bierze dane)
DOMSource source = new DOMSource(doc);
// wyjscie transformatora (gdzie “zmienione” dane zostana zapisane)
StreamResult result = new StreamResult(new File("file.xml"));

// uruchomienie transformatora – zapis DOM do pliku w formacie XML
transformer.transform(source, result);
```

Dokument XML to merytoryczna zawartość jakiegoś tam „dokumentu”. Ta zawartość (xml) może zostać zapisana do formatu *.docx .txt .html* itp. Jeśli chcemy tą zawartość mieć zapisaną w różnych formatach, to musimy nadać odpowiednim tagom w xmlu informację, jak mają zostać wyświetlone w zależności od rozszerzenia pliku w jakim ten xml został zapisany. **I za to odpowiadają transformatory użyte powyżej**.

SAX

`Parser SAX` (Simple API for XML) — Alternatywa technika przetwarzania dokumentów *xml-owych*. W miarę czytania dokumentu wywołuje zdarzenia związane z parsowaniem. Parsery SAX są szybsze i nie wymagają tak dużej ilości pamięci jak DOM.

Działa w ten sposób, że nie tworzy żadnego drzewa, nie ma prawie żadnego związku z pamięcią operacyjną.

> **Czytają dokument xml i generują zdarzenia z tym czytaniem**. `Parser SAX` dostaje dokument xml i obiekt na którym będzie wywoływał metody związane ze zdarzeniami w tym xmlu.

```
import java.io.IOException;
import java.net.URL;
import javax.xml.parsers.*;
import org.xml.sax.*;
public class SAXExample {
   public static void main(String[] args) throws SAXException,
                                        IOException, ParserConfigurationException{
   URL url = new URL("http://www.uj.edu.pl/...");

   SAXParserFactory f = SAXParserFactory.newInstance();
   SAXParser saxParser = f.newSAXParser();

   DefaultHandler handler = new ExampleSAXHandler();
   saxParser.parse(url.openStream(), handler);
   }
}
```

obiekt **handler** jest nasłuchiwaczem zdarzeń (analogia do Swinga). Reaguje na akcje generowane przez parser SAX.

DefaultHandler

`DefaultHandler` — to klasa, która posiada szereg pustych metod, każda z tych metod odpowiada innemu zdarzeniu które może wywołać parser SAX podczas czytania XML'a.

Jak używamy?
Tworzymy własne rozszerzenie klasy w którym nadpisujemy metody zdarzeń na które chcemy w poszczególny sposób reagować

- `void characters(char[] ch, int start, int length)` — wywoływane przy odczycie znaku wewnątrz elementu
- `void endDocument()` — wywoływane gdy koniec dokumentu
- `void endElement(String uri, String localName, String qName)` — koniec elementu
- `void error(SAXParseException e)` — błąd (możliwy do naprawienia)
- `void fatalError(SAXParseException e)` — błąd
- `void ignorableWhitespace(char[] ch, int start, int length)` — ignorowany pusty znak
- `void startDocument()` — początek dokumentu
- `void startElement(String uri, String localName, String qName, Attributes attributes)` — początek elementu
- `void warning(SAXParseException e)` — ostrzeżenie parsera

...

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
class ExampleSAXHandler extends DefaultHandler {
    public void startElement(String uri, String localName, String qName,
                             Attributes attributes) throws SAXException {
        System.out.println("Element :" + qName);
    }
    public void endElement(String uri, String localName, String qName)
                                                    throws SAXException {
        System.out.println("Konie elementu :" + qName);
    }
    public void characters(char ch[], int start, int length)
                                                throws SAXException {
        System.out.println("zawartosc : "+new String(ch, start, length));
    }
}
```

Popdsumowanie *DOM* & *SAX*

Kryterium	SAX	DOM
Model	Zdarzeniowy	Drzewiasty
Przetwarzanie	Sekwencyjne	Cały dokument jest ładowany do pamięci

Kryterium	SAX	DOM
Pamięć	Niewielkie zużycie pamięci	Pamięciożerne
Prędkość	Szybkie	Wolniejsze
Funkcjonalność	Ograniczona	Uniwersalne
Zastosowanie	Duże pliki	Małe pliki
Tworzenie XML	Nieodpowiednie	Odpowiednie
Struktura wewnętrzna	Nie tworzy	Tworzy

JAXB

`JAXB` — (Java Architecture for XML Binding) Służy do **serializacji obiektów** javy do formatu *XML*

> **JAXB jest teraz poza JDK i JRE**. Aby teraz korzystać z JAXB trzeba w programie obsłużyć odpowiednie archiwa jar.

```
import java.io.File;

import javax.xml.bind.*;
import javax.xml.bind.annotation.*;

@XmlRootElement
class Person {

    String name;
    int age;

    public String getName() {
        return name;
    }

    @XmlElement
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    @XmlElement
    public void setAge(int age) {
        this.age = age;
    }

    public String toString() {
        return this.name + " (" + this.age + ")";
    }
}
public class JAXBExample{
    public static void main(String[] args) throws JAXBException {
        Person p = new Person();
        p.setName("Barnaba");
        p.setAge(33);

        File f = new File("person.xml");
        JAXBContext ctx = JAXBContext.newInstance(Person.class);
        Marshaller marshaller = ctx.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
```

```
            marshaller.marshal(p, System.out);
            marshaller.marshal(p, f);
            p = null;
            Unmarshaller unmarshaller = ctx.createUnmarshaller();
            p = (Person)unmarshaller.unmarshal(f);
            System.out.println(p);
        }
    }
```

Klasa, która ma zostać zserializowana musi zawierać odpowiednie adnotacje!

> Czym są adnotacje?
> to są adnotacje: `@XmlRootElement` , `@XmlElement`

`@XmlRootElement` — zaznacza, że klasa *Person* jest korzeniem drzewa xml, w którym będzie instancja tej klasy
Adnotacje to komentarze na poziomie JVM. Komentarze tego typu przechodzą do bajtkodu, ale nie zmieniają działania programu. **Dają informacje**.

Adnotacja `@XmlElement` powoduje, że instancja klasy *Person* będzie miała dwa elementy: *name* i *age*
— Nazwy do elementów bierze z Setterów i Getterów, a nie z nazw zmiennych.
Zostaną one potem wpisane do xml'a i na podstawie xml'a później będzie mozna odtworzyć te wartości.

> Na tym polega **serializacja**, tworzeniu nowego pustego obiektu i on jest wypełniany zawartością tego, co jest w postaci zserializowanej.

`Marshaller` — obiekt odpowiedzialny za serializację, zapis obiektu do xml'a,
createMarshaller() — metoda serializująca

`Unmarshaller` — tworzenie obiektu na bazie danych z xml'a — za pomocą metody *createUnmarshaller()*

XMLEncoder i XMLDecoder

Istnieje inny prostszy sposób serializacji i deserializacji xml'owej niektórych rodzajów obiektów (np. komponenty Swingowe). Możemy wówczas użyć klas `XMLEncoder` i `XMLDecoder` .

`XMLEncoder` :

```
XMLEncoder e = new XMLEncoder(new FileOutputStream("jbutton.xml"));
e.writeObject(new JButton("Hello world"));
e.close();
```

1. Tworzymy nowy obiekt *e*, do którego zapisujemy zserializowany obiekt.
2. podajemy co chcemy zserializować, wczytujemy obiekt Swing'owy.
3. Zamykamy strumień

`XMLDecoder` :

```
XMLDecoder d = new XMLDecoder(new FileInputStream("jbutton.xml"));
obj = d.readObject();
d.close();
```

ANT

***ANT* to javowy odpowiednik "make" w C**. Służy do automatyzacji wszelkich procesów potrzebnych do budowania programów.

Jego podstawowe cechy to:

- konfiguracja zadań zapisana w formacie XML,
- wieloplatformowość – m. in. Linux, Unix (np. Solaris and HP-UX), Windows 9x i NT, OS/2 Warp, Novell Netware 6 oraz MacOS X.
- rozszerzalność w oparciu o klasy napisane w Javie. Ant jest rozwijany w ramach *Apache Software Foundation*. Strona domowa projektu: http://ant.apache.org

Przykładowy plik konfiguracyjny dla anta (domyślnie build.xml):

```
<project name="project" default="default" basedir=".">
    <description>
        jakis opis
    </description>

    <target name="default" depends="depends" description="description">
        <jar destfile="bendmetter.jar" includes="**/*.class"
                                excludes="tests/*.*" basedir="bin"></jar>
    </target>

    <target name="depends">
    </target>
</project>
```

> Aby go “wykonać” wpisujemy w konsoli polecenie *ant*.

default zostanie wykonany dopiero gdy zostanie wykonany najpierw cel depends niżej.

cel *default:* tworzy plik *.jar* który zawiera wszystkie pliki z rozszerzeniem *.class*, wchodząć do katalogu *bin* i wykluczając podkatalog *tests*

ANT jest na dole hierarchii, są inne analogiczne narzędzia z których korzysta się częściej (*Maven* — javowy, *Gradle* — międzyplatformowy).