4. XML i Java

XML jest obecnie standardową reprezentacją danych zapewniającą ich przenośność. Jednakże za tym hasłem kryje się tak dużo zagadnień że nie sposób ich omówić w jednym czy dwóch wykładach. Zatem w wykładzie tym uwaga skupiona zostanie na obecnie dostępnych narzędziach Javy - interfejsach API do przetwarzania i przekształcania dokumentów w standardzie XML. Interfejsy te zostaną najpierw omówione szczegółowo po to żeby zrozumieć późniejsze przykładowe programy je wykorzystujące. Wymaga to od czytelnika dużej samodzielności w analizie tych programów.

4.1.Wprowadzenie

Co to jest **XML** (***Extensible Markup Language***) ?

Jak sama nazwa wskazuje jest to rozszerzalny język znaczników.

Nie wprowadzono w nim ani **zestawu** obowiązujących znaczników ani też nie zdefiniowano poprawnego użycia znaczników (**gramatyki języka**).

Jest zatem rozszerzalny i ma w zasadzie nieograniczone możliwości rozbudowywania.

Nie znaczy to jednak że nie ma w nim żadnych reguł .

Aktualny standard tego języka **XML 1.0**(specyfikacja organizacji W3C-World Wide Web Consortium) wprowadza podstawowe koncepcje dotyczące struktury takich dokumentów:

1. dokument XML **musi być poprawnie uformowany** (skonstruowany)
   a) **każdemu znacznikowi otwierającemu musi odpowiadać znacznik zamykający**
   b) **znaczniki mogą być zagnieżdżane ale tylko wewnętrznie** (jeden znacznik całkowicie wewnątrz drugiego)

c) **musi istnieć znacznik główny obejmujący swoim zasięgiem wszystkie inne znaczniki**

2. dokumentowi XML **można narzucić** poprawność (zawęzić) zgodnie z definicją DTD (dokument XML **może być poprawny**)

Inny sposób zawężania dokumentów XML opisuje schemat XML Schema, ale należy pamiętać że w przeciwieństwie do DTD nie jest on częścią specyfikacji **XML 1.0** . Zawężanie dokumentów XML umożliwia zrozumienie sposobu reprezentacji danych innym osobom jak również innym aplikacjom.

Uwaga : W świetle tych koncepcji dokument **poprawnie uformowany** nie musi być **poprawny**.

Do czego może służyć tak określony dokument XML ?

Dokument XML zapewnia **przenośny opis danych** i ich struktury, zatem jest **ukierunkowany na opis danych** a nie jak np. HTML na prezentację danych.

Dokument taki po utworzeniu i przesłaniu go lokalnie lub globalnie przez sieć może zostać poddany analizie (parsowaniu) w celu wydzielenia ze znaczników określonych atrybutów i danych. Atrybuty i dane po takim przetworzeniu są zazwyczaj umieszczane w pewnych strukturach danych, które z kolei mogą być przetwarzane przez inną aplikację. Parsowanie dokumentu XML jest zadaniem złożonym i z reguły do tego celu używa się profesjonalnych parserów.

Główne kryteria wyboru parsera to zgodność ze specyfikacją XML i szybkość analizy.

Do najpopularniejszych parserów należą:

Apache Xerces (http://xml.apache.org)

IBM XML4J (http://alphaworks.ibm.com/tech/xml4j)

OpenXML (http://www.openxml.org)

Oracle XML Parser (http://technet.oracle.com/tech/xml)

Sun Microsystems Project X (http://java.sun.com/products/xml)

Mimo że dokument XML opisuje dane **może jednak być przekształcony** do formatu umożliwiającego **prezentację danych** dla różnych użytkowników za pomocą arkusza stylów **XSL** (Extensible StyleSheet Language) i transformacji **XSLT** (Extensible StyleSheet Language Transformation).

Transformacje wykonują specjalne programy zwane procesorami **XSLT**. Do najbardziej znanych należą:

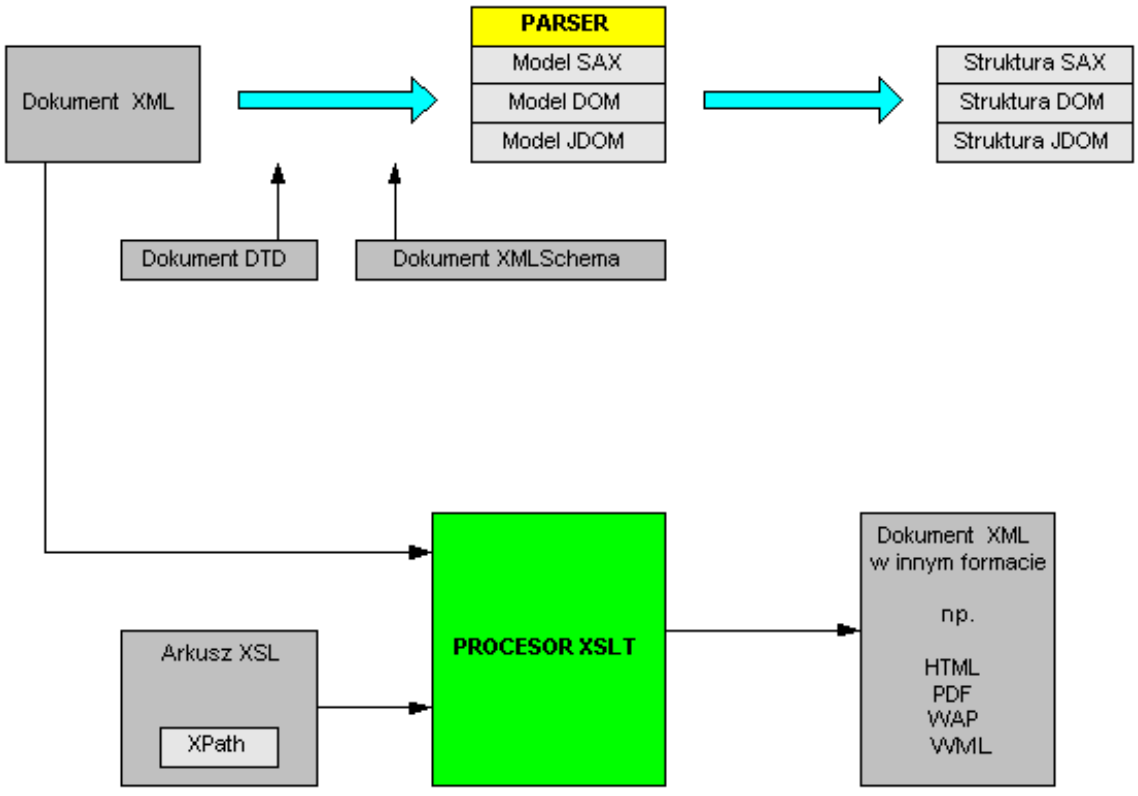
Apache Xalan (http://xml.apache.org)

Lotus XSL Processor (http://www.alphaworks.ibm.com/tech/LotusXSL)

Oracle XSL Processor (http://technet.oracle.com/tech/xml)

I w tym przypadku głównymi kryteriami wyboru jest zgodność ze specyfikacjami **XSL,XSLT** i szybkość działania.

Poniższy diagram pokazuje schemat przetwarzania (parsowania) i przekształcania (zmiany formatu) dokumentu XML.



Dokument XML można utworzyć w dowolnym edytorze tekstowym (np. Notepad) .Utworzony dokument może być następnie poddany przetworzeniu (parsing) w celu odzyskania atrybutów lub danych ze struktury znaczników.

Można rozróżnić trzy rodzaje przetwarzania:

1. **przetwarzanie bez sprawdzania poprawności**
2. **przetwarzanie ze sprawdzaniem poprawności wg DTD**
3. **przetwarzanie ze sprawdzaniem poprawności wg XML Schema**

W przypadku 2 i 3.. należy najpierw opisać w dokumencie typu DTD (Document Type Definition) lub XML Schema sposób zawężania dokumentu XML.

DTD defniuje sposób w jaki ma być skonstruowany dokument XML wprowadzając pewne ograniczenia na format znaczników i ich składnię.

To właśnie uzgodnienie formatu i składni zapewnia dokumentowi XML przenośność między aplikacjami.

Standard DTD ma jednak poważne ograniczenia: własne konstrukcje nie związane z XML, brak znajomości hierarchii, trudności w obsłudze przestrzeni nazw, niemożność określania relacji między dokumentami XML.

Schemat XML Schema jest alternatywą dla DTD o znacznie bogatszych możliwościach.

Istotne jest przede wszystkim że definiowanie XML odbywa się przy pomocy konstrukcji XML, co zapewnia spójność opisu dokumentów XML.

Należy jednakże pamiętać że nie wszystkie parsery obsługują sprawdzanie poprawności w/g XML Schema.

Po zdecydowaniu się na sposób zawężenia dokumentu XML mamy następnie do wyboru 3 modele przetwarzania dokumentu XML reprezentowane przez odpowiednie interfejsy (zestawy funkcji):

- model **SAX**
- model **DOM**
- model **JDOM (**alternatywa dla **SAX** lub **DOM)**

Wśród zestawów bibliotek można wyodrębnić zestaw JAXP (Java API for XML Processing) pozwalający na dokonywanie analizy w modelu SAX lub DOM niezależnie od konkretnego producenta parsera. Zestawienie pakietów dla poszczególnych modeli przedstawia tabela:

Model SAX	**Model DOM**	**Model JDOM**	**JAXP**	**XSLT**
org.xml.sax org.xml.sax.helpers org.xml.sax.ext	org.w3c.dom	org.jdom org.jdom.adapters org.jdom.input	javax.xml.parsers	javax.xml.transform javax.xml.transform.sax javax.xml.transform.dom

		org.jdom.output org.jdom.filter org.jdom.transform		javax.xml.transform.stream

Pakiety SAX, DOM, JAXP, XSLT są wbudowane w J2SE i dostarczane razem z JDK1.4.1. Pakiety JDOM są osobnym zestawem API i trzeba je załadować z sieci niezależnie od JDK1.4.1. Zrozumienie API Javy zawartego w tych pakietach wymaga znajomości składni i struktury dokumentu XML - krótki opis języka znajdziemy w rozdziale następnym.

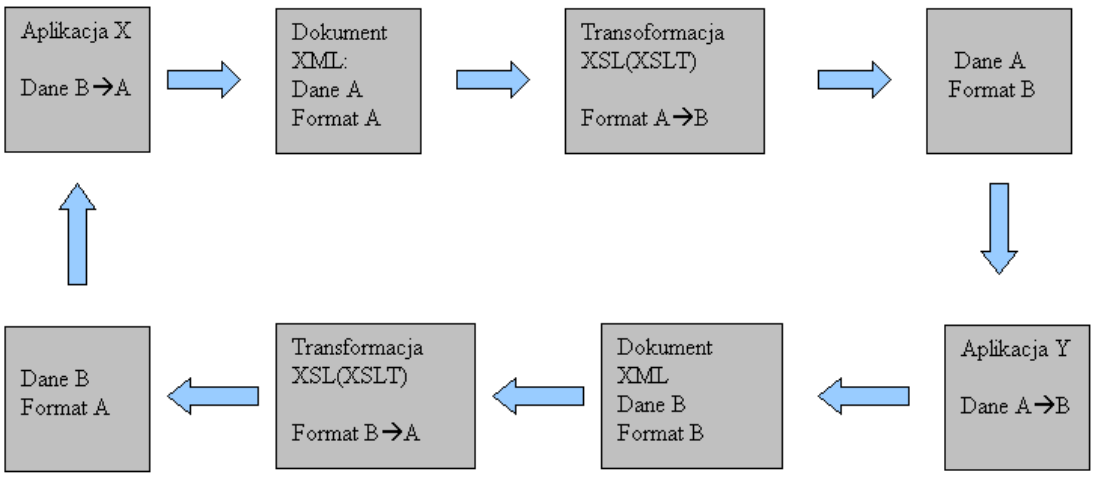
Przenośność danych XML pełni ważną rolę w komunikacji:

aplikacja <-------> aplikacja

system <-------> system

firma <-------> firma

Komunikacja taka powinna zakładać w ogólności istnienie klientów wymagających prezentacji danych jak również takich którzy nie wymagają takiej prezentacji. Tak więc ogólnie w czasie swojego życia dokument XML może podlegać cyklowi przemian danych i ich formatu.
Przykładowy cykl takich transformacji pokazuje diagram poniżej.



Warto też pamiętać że XML to dane tekstowe, które nie wymagają dużych zasobów systemowych, Łatwo też podlegają serializacji, zatem przesyłanie ich w sieci jest nie stanowi problemu i jest szybkie.

Połączenie XML i Javy – **przenośnych danych** i **przenośnego kodu** pozwala widzieć w tym połączeniu **technologię przyszłości**, niezależnie od aktualnego stanu rozwoju tego standardu i narzędzi do jego obsługi. Jeżeli dodamy do tego fakt, że API Javy umożliwia dynamiczne tworzenie dokumentów XML, odczytywanie, modyfikacje danych oraz przechowywanie danych (informacji) w standardowym formacie to już ta krótka charakterystyka XML zachęca do jego poznania tego języka i związanych z nim technologii.

4.2. Składnia języka XML i struktura dokumentu XML

Omówimy teraz krótko składnie i strukturę dokumentu XML a potem zaprezentujemy kilka przykładów takich dokumentów.

A - składnia XML:

instrukcje przetwarzania (PI)

nakazują aplikacji przetwarzającej wykonanie określonego zadania

Ich ogólna postać to :

<? cel instrukcja ?>

gdzie :

cel - nazwa aplikacji która ma przetwarzać dane XML

instrukcja - łańcuch znaków zawierający informację lub komendy dla aplikacji

deklaracje typu dokumentu - określenie dokumentu DTD

<!DOCTYPE JavaXp:Book SYSTEM ”C:\JavaXP.dtd”>

<!DOCTYPE Java:Book PUBLIC " Nazwa publiczna" ”http://www.w3.org./DTD/contents.dtd”>

Java:Xp jest elementem głównym dokumentu

Po słowie SYSTEM lub PUBLIC powinien znajdować się poprawny identyfikator URI(np. URL)

PUBLIC oznacza że definicja DTD do której następuje odwołanie ma zasięg publiczny i mogą z niej korzystać wszyscy.

W tym przypadku przed podaniem URI konieczne jest podanie nazwy publicznej.

encje

Reprezentują nietypowe znaki w danych. Po napotkaniu encji parser XML podstawia pod nią określoną wartość i nie przetwarza jej dalej.

Ogólna postać encji wygląda następująco : **&[nazwa-encji];**

W XML zdefiniowano 5 encji :

< otwierający nawias kątowy lub ‘mniejsze niż’ (<)

&gt; zamykający nawias kątowy lub ‘większe niż’ (>)

& znak ampersand

" cudzysłów

' apostrof

Encje mogą być również **definiowane przez użytkownika**, który w ten sposób może odwoływać się do zewnętrznego dokumentu lub innego zasobu.

Przykład takiej encji zobaczymy w jednym z prezentowanych dokumentów XML.

elementy, atrybuty elementów , dane elementów, przestrzenie nazw

Element określony jest przez układ znaczników : otwierający i zamykający.

W znaczniku otwierającym zawarta jest **nazwa elementu** oraz mogą być zawarte **pary (atrybut, wartość).**

Między znacznikami mogą występować **dane elementu.**

Nazwa elementu musi rozpoczynać się literą lub podkreśleniem, po którym może wystąpić dowolna liczba liter, cyfr, podkreśleń, łączników lub kropek.

Nazwy nie mogą zawierać spacji. Wielkość liter jest rozróżnialna w nazwach.

Fragment prostego dokumentu XML z elementami zawierającymi **dane i atrybuty**

```
<wiadomość >
  <do>Jan Kowalski</do>
  <od>Andrzej Talarek</od>
```

<temat> Kolokwium z Javy</temat>

<tekst zadanie1=”2” zadanie2=”2”> poszło mi fatalnie </tekst>

</wiadomość>

Powstaje naturalne pytanie kiedy używać **atrybutów** i ich wartości a kiedy **danych.** Nie istnieje niestety żadna specyfikacja ani standard mówiący o tym ale utarło się w praktyce, że atrybutów i ich wartości używa się do opisu informacji systemowych, danych elementu używa się do opisu informacji przeznaczonych do prezentacji.

Specjalnym elementem jest **element pusty** (nie zawierający danych)

<image src=”RedBall.gif”/>

Wprowadzony został dla uproszczenia takiego zapisu elementu:

<image src=”RedBall.gif”></image>

Każdy element może posiadać **nazwę kwalifikowaną przedrostkiem,** reprezentującym określoną przestrzeń nazw oraz umożliwiającym jednoznaczną identyfikację elementu i w ten sposób wyeliminowanie kolizji nazw elementów. Przedrostkowi powinno się przypisać niepowtarzalny identyfikator URI (np. URL).

Element poniższy ma nazwę **Book** która pochodzi z przestrzeni nazw **JavaXP.** Przestrzeń nazw **JavaXP** skojarzona jest z adresem URL.

<JavaXP:Book xmlns:JavaXP=”http://www.jb.com/catalog/javaxml/”>

Specyfikacja przestrzeni nazw wymaga, żeby każdy element XML należał do jakiejś przestrzeni nazw.

Jeżeli zatem nie deklarujemy jawnie przestrzeni nazw za pomocą przedrostka, to dany element należy do przestrzeni domyślnej.

W tym przykładzie element **Book** należy do domyślnej przestrzeni nazw skojarzonej z adresem URL

<Book xmlns=”http://www.jb.com/catalog/javaxml/”>

komentarze

Komentarz w XML ma postać :

<!-- oto jest komentarz -->

sekcja CDATA

Reprezentuje dane nie przetwarzane przez parser XML Stosowana jest wtedy gdy aplikacji wywołującej trzeba przekazać dużą ilość danych nieprzetworzonych przez parser XML.

W dokumencie XML sekcja **CDATA** wygląda tak:

<nieprzetwarzane-dane>

<![CDATA[Diagram:

<krok-1>zainstaluj JVM

<krok-2>znajdz odpowiedni plik properties

<krok-3> pobierz program Program.class z adresu ”ftp://ftp.pjwstk.edu.pl”

]]>

</nieprzetwarzane-dane>

B - Struktura dokument XML

Dokument XML składa się z dwóch zasadniczych części : **prologu i zawartości**.

I - PROLOG

Prolog jest pierwszą częścią dokumentu XML

► deklaracja identyfikująca

Prolog powinien zawierać przynajmniej jedną deklarację która identyfikuje dany dokument jako dokument XML.

<?xml version=”1.0”?>

Znacznik ten może zawierać generalnie 3 atrybuty:

version – wersja XML

encoding -system kodowania

standalone - czy jest samodzielnym dokumentem XML

<?xml version=”1.0” encoding=”ISO-8859-2” standalone=”no”?>

► **inne instrukcje przetwarzania**

► **deklaracje typu dokumentu(DTD lub XML Schema)**

II - ZAWARTOŚĆ DOKUMENTU

► **element główny**

► **elementy zagnieżdżone**

► **encje**

► **dane nie przetwarzane**

Prześledzimy teraz kilka dokumentów XML od najprostszych aż do bardziej złożonych. Należy zaobserwować w nich strukturę i składnię dokumentu XML.

Przykład bardzo prostego dokumentu XML opisującego towar o numerze identyfikacyjnym, nazwie, cenie i ilości :

Dokument item.xml

```
<!--  PROLOG DOKUMENTU -->

<!--   deklaracja identyfikująca    -->

<?xml version="1.0"?>

<!-- ZAWARTOŚĆ DOKUMENTU -->

<!-- element główny: znacznik otwierający -->

<ITEM>

<!-- element zagnieżdżony z danymi  -->

<ID>33445</ID>
```

```
<!-- element zagnieżdżony z danymi  -->

<DESCRIPTION>JavaBook</DESCRIPTION>

<!-- element zagnieżdżony z danymi  -->

<PRICE>19.95</PRICE>

<!-- element zagnieżdżony z danymi  -->

<QUANTITY>56</QUANTITY>

<!-- element główny: znacznik zamykający -->

</ITEM>
```

Następny dokument to dokument opisujący **pozycje figur** dla danej konfiguracji szachownicy.

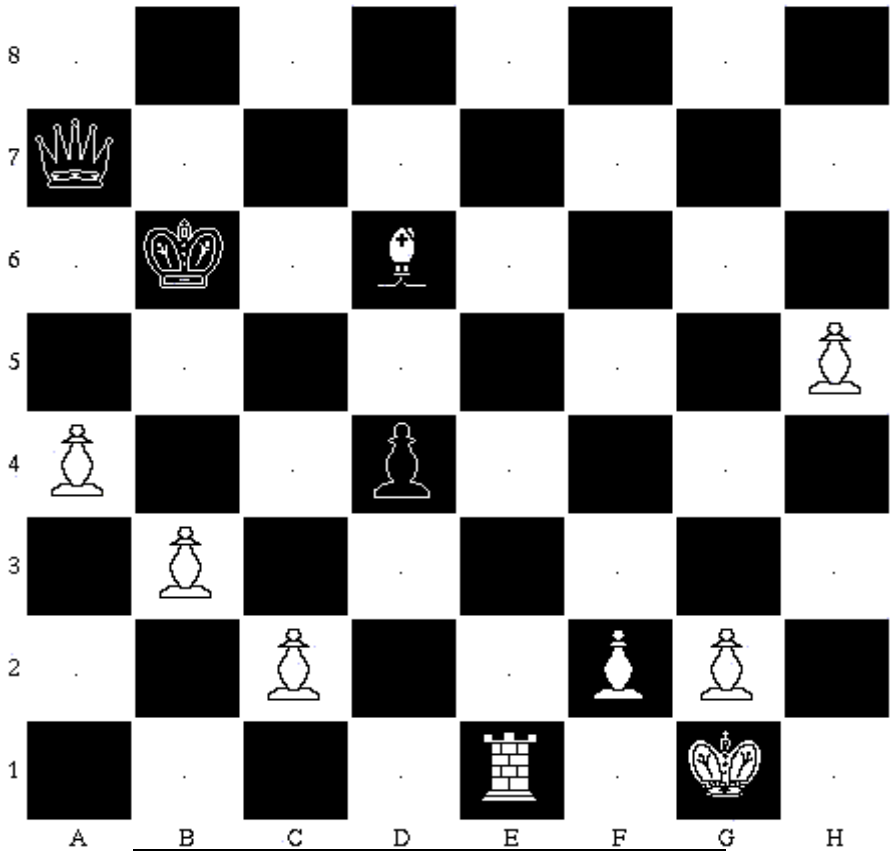


Tabela położeń figur na szachownicy:

White king (biały król)	G1
White bishop (biały goniec)	D6
White rook (biała wieża)	E1

White pawn (biały pionek)	A4
White pawn (biały pionek)	B3
White pawn (biały pionek)	C2
White pawn (biały pionek)	F2
White pawn (biały pionek)	G2
White pawn (biały pionek)	H5
Black king (czarny król)	B6
Black queen(czarna królowa)	A7
Black pawn (czarny pionek)	A5
Black pawn(czarny pionek)	D4

źródło: http://www.java.sun.com

Dokument XML opisujący tę konfigurację:

Dokument chess.xml

```
<!--  PROLOG DOKUMENTU -->

<!--  deklaracja identyfikująca -->

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

<!-- ZAWARTOŚĆ DOKUMENTU -->

<!--  element główny CHESSBOARD-->

<CHESSBOARD>

<WHITEPIECES>

  <!--  element zagnieżdżony KING -->

  <KING>

    <!--  element zagnieżdżony POSITION z atrybutami COLUMN i ROW-->

    <POSITION COLUMN="G" ROW="1"/>

  </KING>

  <BISHOP><POSITION COLUMN="D" ROW="6"/></BISHOP>

  <ROOK>< POSITION COLUMN="E" ROW="1"/></ROOK>

  <PAWN>< POSITION COLUMN="A" ROW="4"/></PAWN>

  <PAWN><POSITION COLUMN="B" ROW="3"/></PAWN>
```

```
  <PAWN><POSITION COLUMN="C" ROW="2"/></PAWN>
  <PAWN><POSITION COLUMN="F" ROW="2"/></PAWN>
  <PAWN><POSITION COLUMN="G" ROW="2"/></PAWN>
  <PAWN><POSITION COLUMN="H" ROW="5"/></PAWN>
</WHITEPIECES>
<BLACKPIECES>
  <KING><POSITION COLUMN="B" ROW="6"/></KING>
  <QUEEN><POSITION COLUMN="A" ROW="7"/></QUEEN>
  <PAWN><POSITION COLUMN="A" ROW="5"/></PAWN>
  <PAWN><POSITION COLUMN="D" ROW="4"/></PAWN>
</BLACKPIECES>
<!--  koniec elementu  głównego CHESSBOARD  -->
</CHESSBOARD>
```

źródło: http://www.java.sun.com

Następny dokument opisuje zawartość książki "Java and XML" (autor: Brett McLaughlin)

Dokument contents.xml

```
<!--  PROLOG DOKUMENTU -->
<!--  deklaracja identyfikująca -->
<?xml version="1.0"?>
<!--  instrukcje odwołujące się do arkuszy stylów XSL -->
<?xml-stylesheet href="XSL\JavaXML.html.xsl" type="text/xsl"?>
<?xml-stylesheet href="XSL\JavaXML.wml.xsl" type="text/xsl" media="wap"?>
<!--  instrukcje przetwarzania dla aplikacji ‘cocoon’ -->
<?cocoon-process type="xslt"?>
```

```
<!--  deklaracja dokumentu  DTD -->
 <!DOCTYPE JavaXML:Book SYSTEM "DTD\JavaXML.dtd">
<!-- ZAWARTOŚĆ DOKUMENTU -->
<JavaXML:Book xmlns:JavaXML="http://www.oreilly.com/catalog/javaxml/">
 <JavaXML:Title>Java and XML</JavaXML:Title>
<JavaXML:Contents>
 <JavaXML:Chapter-1 focus="XML">
  <JavaXML:Heading>Introduction</JavaXML:Heading>
  <JavaXML:Topic subSections="7">What Is It?</JavaXML:Topic>
  <JavaXML:Topic subSections="3">How Do I Use It?</JavaXML:Topic>
  <JavaXML:Topic subSections="4">Why Should I Use It?</JavaXML:Topic>
  <JavaXML:Topic subSections="0">What's Next?</JavaXML:Topic>
 </JavaXML:Chapter-1>
 <JavaXML:Chapter-2 focus="XML">
  <JavaXML:Heading>Creating XML</JavaXML:Heading>
  <JavaXML:Topic subSections="0">An XML Document</JavaXML:Topic>
  <JavaXML:Topic subSections="2">The Header</JavaXML:Topic>
  <JavaXML:Topic subSections="6">The Content</JavaXML:Topic>
  <JavaXML:Topic subSections="1">What's Next?</JavaXML:Topic>
 </JavaXML:Chapter-2>
<JavaXML:Chapter-3 focus="Java">
  <JavaXML:Heading>Parsing XML</JavaXML:Heading>
  <JavaXML:Topic subSections="3">Getting Prepared</JavaXML:Topic>
  <JavaXML:Topic subSections="3">SAX Readers</JavaXML:Topic>
  <JavaXML:Topic subSections="9">Content Handlers</JavaXML:Topic>
```

```
  <JavaXML:Topic subSections="4">Error Handlers</JavaXML:Topic>
  <JavaXML:Topic subSections="0"> A Better Way to Load a Parser </JavaXML:Topic>
  <JavaXML:Topic subSections="4">"Gotcha!"</JavaXML:Topic>
  <JavaXML:Topic subSections="0">What's Next?</JavaXML:Topic>
 </JavaXML:Chapter-3>
 <JavaXML:SectionBreak/>
 <JavaXML:Chapter-4 focus="Java">
  <JavaXML:Heading>Web Publishing Frameworks</JavaXML:Heading>
  <JavaXML:Topic subSections="4">Selecting a Framework</JavaXML:Topic>
  <JavaXML:Topic subSections="4">Installation</JavaXML:Topic>
  <JavaXML:Topic subSections="3"> Using a Publishing Framework </JavaXML:Topic>
  <JavaXML:Topic subSections="2">XSP</JavaXML:Topic>
  <JavaXML:Topic subSections="3">Cocoon 2.0 and Beyond</JavaXML:Topic>
  <JavaXML:Topic subSections="0">What's Next?</JavaXML:Topic>
 </JavaXML:Chapter-4>
</JavaXML:Contents>
 <!--  element z własną encją użytkownika : opis przetworzenia tej encji w definicji
DTD-->
 <JavaXML:Copyright> &OReillyCopyright; </JavaXML:Copyright>
<!--  znacznik zamykający element główny JavaXML:Book-->
</JavaXML:Book>
```

źródło: http://www.oreilly.com/catalog/javaxml

4.3. Model SAX

Model **SAX** (Simple API for XML) jest rozwijany przez członków listy adresowej XML-dev.

SAX dokonuje odczytywania dokumentu i rozbijania danych na odpowiednie elementy za pomocą mechanizmu obsługi zdarzeń. Definiuje on zdarzenia procesu przetwarzania dokument XML, które będą podlegały monitorowaniu i obsłudze. Umożliwia dostęp do danych w dokumencie XML.

Warto też pamiętać że SAX definiuje tylko sposób przetwarzania XML natomiast sam nie dokonuje przetwarzania dokumentów XML (nie jest parserem).

Odpowiednie dla modelu SAX pakiety Javy to : **org.xml.sax, org.xml.sax.helpers oraz org.xml.sax.ext.** Omówimy teraz te pakiety żeby zapoznać się z możliwościami analizy jakie one dają i żeby prześledzić ich działanie w przykładzie programu podanym później**.**

Interfejsy i klasy pakietu org.xml.sax:

Typ	**Nazwa**	**Opis**
Interfejsy	***Attributes***	reprezentuje listę atrybutów XML; rozpoznaje przestrzenie nazw
	ContentHandler	operacje na zawartości dokumentu XML
	DTDHandler	Obsługa podstawowych zdarzeń w dokumencie DTD
	EntityResolver	Rozpoznawanie encji
	ErrorHandler	Podstawowy interfejs do obsługi zdarzeń SAX
	Locator	Lokalizacja źródeł zdarzeń SAX w dokumencie
	XMLFilter	Filtr danych, rozszerza *XMLReader*
	XMLReader	Określenie sposobu przetwarzanie w/g modelu SAX2.0.Implementowany przez parsery różnych producentów
Klasy	**InputSource**	reprezentacja źródła danych XML do przetwarzania: strumień bajtów, strumień znaków, URI
Klasy wyjątków	**SAXException**	Zgłaszany przy wywołaniach zdarzeń SAX
	SAXNotRecognizedException	Zgłaszany przez parser gdy nie rozpoznano nazwy właściwości
	SAXNotSupportedException	Zgłaszany przez parser gdy brak jest obsługi danej własności lub cechy
	SAXParserException	Zgłaszany w czasie przetwarzania przez parser

Interfejs ***Attributes*** definiuje funkcjonalność związaną z listą atrybutów. Umożliwia dostęp do listy atrybutów na 3 różne sposoby:

- poprzez indeks atrybutu

- poprzez identyfikator przestrzeni nazw i nazwę lokalną atrybutu

- poprzez nazwę atrybutu kwalifikowaną przestrzenią nazw

Porządek atrybutów na liście jest nieokreślony i zmienia się od implementacji do implementacji.

Funkcje interfejsu *Attributes*	
`int`	**`getIndex`** `(String qName)` dostarcza indeks atrybutu poprzez nazwę atrybutu kwalifikowaną przedrostkiem przestrzeni nazw
`int`	**`getIndex`** `(String uri, String localName)` dostarcza indeks atrybutu poprzez identyfikator przestrzeni nazw i nazwę lokalną atrybutu.
`int`	**`getLength`**`()` dostarcza liczbę atrybutów na liście.
`String`	**`getLocalName`**`(int index)` dostarcza lokalną nazwę atrybutu na podstawie indeksu.
`String`	**`getQName`**`(int index)` dostarcza kwalifikowaną nazwę atrybutu na podstawie indeksu.
`String`	**`getType`** `(int index)` dostarcza typ atrybutu na podstawie indeksu.
`String`	**`getType`**`( String qName)` dostarcza typ atrybutu na podstawie nazwy kwalifikowanej atrybutu.
`String`	**`getType`**`(String uri,String localName)` dostarcza typ atrybutu na podstawie identyfikatora przestrzeni nazw i nazwy lokalnej.
`String`	**`getURI`**`(int index)` dostarcza identyfikator przestrzeni nazw atrybutu poprzez indeks.
`String`	**`getValue`**`(int index)` dostarcza wartość atrybutu poprze indeks.
`String`	**`getValue`**`(String qName)` dostarcza wartość atrybutu o podanej nazwie kwalifikowanej.
`String`	**`getValue`** `(String uri,String localName)` dostarcza wartość atrybutu na podstawie identyfikatora przestrzeni nazw i nazwy lokalnej.

Interfejs ***ContentHandler*** jest implementowany przez większość aplikacji SAX,w których istotne są informacje o logicznej zawartości dokument XML. Dzięki implementacji tego interfejsu i zarejestrowaniu implementacji za pomocą **setContentHandler()** parser otrzymuje informacje o zdarzeniach wywołanych przez poszczególne komponenty dokumentu XML**.** Kolejność otrzymanych zdarzeń odzwierciedla kolejność występowania odpowiednich komponentów.

Funkcje przedstawione w tabeli są funkcjami typu "callback" i są wywoływane po napotkaniu w dokumencie XML odpowiednich konstrukcji.

Funkcje interfejsu *ContentHandler*	
void	**characters** (char[] ch, int start, int length) udostępnia dane zawarte w elemencie w postaci tablicy znaków oraz indeks początkowy i końcowy danych do odczytania; informuje również o białych znakach
void	**endDocument** () informacja o końcu przetwarzania dokumentu
void	**endElement** (String namespaceURI, String localName, String qName) koniec elementu; namespaceURI jest jest przestrzenią tego danego elementu; localName- nazwa lokalna elementu; qName- nazwa globalna elementu
void	**endPrefixMapping** (String prefix) koniec odwzorowania przedrostka przestrzeni nazw; prefix- znaleziony przedrostek przestrzeni nazw
void	**ignorableWhitespace** (char[] ch, int start, int length) informacja o białych znakach ignorowanych w dokumencie XML; ch - tablica znaków zawartych w elemencie; start. length - indeksy danych w tablicy.
void	**processingInstruction** (String target, String data) informacja o napotkanej instrukcji przetwarzania; target- obiekt docelowy PI ; data- dane wysłane do PI
void	**setDocumentLocator** (Locator locator) ustalenie obiektu lokalizatora podającego miejsce wystąpienia wywołania wstecznego; locator - lokalizator miejsca wywołania
void	**skippedEntity** (String name) informacja o encji pominiętej przez parser; name - nazwa pominiętej encji
void	**startDocument** () informacja o początku dokumentu.
void	**startElement**(String namespaceURI, String localName, String qName, Attributes atts) początek elementu; namespaceURI - identyfikator przestrzeni nazw danego elementu; localName - nazwa lokalna elementu; qName - nazwa globalna elementu
void	**startPrefixMapping** (String prefix, String uri) początek odwzorowania przedrostka przestrzeni nazw; prefix- znaleziony przedrostek przestrzeni nazw ; uri - identyfikator URI przestrzeni nazw

Interfejs ***ErrorHandler*** to podstawowy interfejs do obsługi błędów SAX.

Obiekt klasy implementującej musi być zarejestrowany przez parser za pomocą funkcji **setErrorHandler().**

Funkcje przedstawione w tabeli są funkcjami typu **"callback"** i są wywoływane po wystąpieniu w dokumencie XML określonych błędów.

Funkcje interfejsu *ErrorHandler*	
void	**error** (SAXParseException exception) błąd niekrytyczny - naruszono regułę XML (zazwyczaj naruszenie składni); przetwarzanie może być kontynuowane
void	**fatalError** (SAXParseException exception) błąd krytyczny - naruszona została zasada XML; dalsze przetwarzanie nie jest możliwe lub niecelowe
void	**warning** (SAXParseException exception) ostrzeżenie -żadne reguły XML nie zostały naruszone ale występuje niepoprawny fragment dokumentu

Interfejs ***DTDHandler*** deklaruje 2 funkcje do obsługi zdarzeń zwiazanych z przetwarzaniem dokumentu DTD:

Funkcje interfejsu *DTDHandler*	
void	**notationDecl** (String name, String publicId, String systemId) informacja o wystąpieniu deklaracji NOTATION; name - nazwa encji; publicId - identyfikator publiczny; systemId - dentyfikator systemowy
void	**unparsedEntityDecl** (String name, String publicId, String systemId , String notationName) informacja o wystąpieniu deklaracji nie przetwarzanej encji; name - nazwa encji; publicId - identyfikator publiczny; systemId - identyfikator systemowy

Obie te metody są wykorzystywane rzadko, gdyż zdarzenia związane z czytaniem definicji DTD są znacznie mniej ważne niż te związane z przetwarzaniem dokumentu XML.

Interfejs ***EntityResolver*** służy do rozpoznawania i tłumaczenia zewnętrznych encji w dokumencie XML.

Nie wszystkie aplikacje muszą implementować ten interfejs. Szczególnie pożyteczny będzie w aplikacjach tworzących dokumenty XML z baz danych lub innych specjalizowanych źródeł albo też w aplikacjach używających identyfikatorów innych niż URL.

Funkcje interfejsu *EntityResolver*	
InputSource	**resolveEntity** (String publicId, String systemId) dostarcza źródło do którego odnosi się encja; publicId, systemId- identyfikator publiczny i systemowy encji.

Obiekt **InputSource** może być również zastosowany jako argument metody **parse()** klasy **Parser** .Parser SAX będzie używał tego obiektu do określenia sposobu czytania dokumentu XML. Jeżeli dostępny jest on jako strumień znaków lub bajtów parser będzie czytał z tych strumieni bezpośrednio. Jeżeli strumienie te nie są dostępne parser będzie próbował otworzyć połączenie z zasobem identyfikowanym przez identyfikator systemowy.

Konstruktory klasy **InputSource** podaje tabela.

Konstruktory klasy InputSource
InputSource() konstruktor bezargumentowy
InputSource(**InputStream** byteStream) tworzy obiekt InputSource na bazie strumienia bajtów.
InputSource(Reader characterStream) tworzy obiekt InputSource na bazie strumienia znaków.
InputSource(String systemId) tworzy obiekt InputSource na podstawie identyfikatora systemowego.

W podanym przykładzie konstruktor **InputSource** po napotkaniu encji z identyfikatorem systemowym "http://www.myhost.com/today" dostarcza aplikacji źródła danych typu strumienia znakowego

```
import org.xml.sax.EntityResolver;
import org.xml.sax.InputSource;

public class MyResolver implements EntityResolver {

public InputSource resolveEntity (String publicId, String systemId) {

 if (systemId.equals("http://www.myhost.com/today")) {
   Reader reader = new FileReader("data.txt");
   //zwraca źródło danych typu strumienia znakowego
   return new InputSource(reader);
 }
 else return null;

} //resolveEntity()
} //class MyResolver
```

Interfejs ***Locator*** służy do kojarzenia zdarzenia SAX z miejscem w dokumencie w którym nastąpiło wywołanie wsteczne**.**

Wyniki dostarczone przez metody tego interfejsu będą określone tylko w zakresie metod obiektu **ContentHandler.**

Parser SAX nie musi dostarczać obiektu typu ***Locator*** ale jego utworzenie jest bardzo pożyteczne .

Funkcje interfejsu *Locator*	
`int`	**`getColumnNumber`** `()` dostarcza numer kolumny w dokumencie XML, gdzie wystąpiło zdarzenie.
`int`	**`getLineNumber`** `()` dostarcza numer wiersza w dokumencie XML, gdzie wystąpiło zdarzenie.
`String`	**`getPublicId`** `()` dostarcza identyfikator publiczny zdarzenia.
`String`	**`getSystemId`** `()` dostarcza identyfikator systemowy zdarzenia..

Interfejs ***XMLReader*** służy do czytania dokumentu XML za pomocą wywołań wstecznych ("callback").

Musi być implementowany prze parser SAX2.

Funkcje interfejsu *XMLReader*	
`ContentHandler`	**`getContentHandler`** `()` dostarcza zarejestrowany obiekt typu ContentHandler.
`DTDHandler`	**`getDTDHandler`** `()` dostarcza zarejestrowany obiekt typu DTDHandler.
`EntityResolver`	**`getEntityResolver`** `()` dostarcza zarejestrowany obiekt typu EntiityResolver.
`ErrorHandler`	**`getErrorHandler`** `()` dostarcza zarejestrowany obiekt typu ErrorHandler.
`boolean`	**`getFeature`** `( String name)` dostarcza stan podanej cechy parsera.
`Object`	**`getProperty`** `(String name)` dostarcza obiekt podanej właściwości parsera.
`void`	**`parse`** `( InputSource input)` analizuje dokument XML podany jako obiekt typu InputSource.
`void`	**`parse`** `(String systemId)` analizuje dokument XML na podstawie identyfikatora systemowego.
`void`	**`setContentHandler`** `(ContentHandler handler)` rejestruje obiekt typu ContentHandler.

void	**setDTDHandler** (DTDHandler handler) rejestruje obiekt typu DTDHandler.
void	**setEntityResolver** (EntityResolver resolver) rejestruje obiekt typu EntityResolver.
void	**setErrorHandler** (ErrorHandler handler) rejestruje obiekt typu ErrorHandler.
void	**setFeature** (String name, boolean value) włącza lub wyłącza nazwaną cechę parsera .
void	**setProperty** (String name, Object value) ustawia nazwaną właściwość parsera i obiekt wykorzystywany do jej realizacji.

W podstawowym interfejsie ***XMLReader*** zdefiniowano 2 funkcje do ustalania właściwości i cech danej implementacji parsera oraz 2 funkcje do uzyskiwania informacji o właściwościach i cechach danej implementacji parsera. Są to funkcje:

void setProperty(String propertyName,Object obj)

void setFeature(String featureName,boolean state)

Object getProperty(String propertyName)

boolean getFeature(String featureName)

W tych funkcjach parametry: **propertyName** i **featureName** są pełnymi identyfikatorami URI(np.URL), obiekt **obj** jest obiektem wykorzystywanym do realizacji określonej właściwości.

Identyfikator URI właściwości	**Opis właściwości**
http://xml.org/sax/properties/lexical-handler	Ustalenie implementacji interfejsu ***LexicalHandler*** do obsługi komentarzy i odwołań do definicji DTD
http://xml.org/sax/properties/declaration-handler	Ustalenie implementacji interfejsu ***DeclHandler d***o obsługi zawężeń DTD
http://xml.org/sax/properties/dom-node	Pobranie węzła bieżącego lub głównego przy przetwarzaniu w modelu DOM
http://xml.org/sax/properties/xml-string	Pobranie tekstu, który spowodował zajście bieżącego zdarzenia
Identyfikator URI cechy	**Opis cechy**
http://xml.org/sax/features/namespaces	Wykonywanie przetwarzania przestrzeni nazw
http://xml.org/sax/features/namespace-prefixes	Komunikowanie o atrybutach deklaracji przestrzeni nazw
http://xml.org/sax/features/string-interning	Internalizacja nazw elementów, przedrostków, identyfikatorów URI

http://xml.org/sax/features/validation	sprawdzanie poprawności dokumentu XML
http://xml.org/sax/features/external-general-entities	Przetwarzanie zewnętrznych encji (ogólnych)
http://xml.org/sax/features/external-parameter-entities	Przetwarzanie zewnętrznych encji (parametrów)

Interfejsy i klasy pakietu org.xml.sax.helpers

Pakiet ten zawiera klasy pomocnicze (implementacje interfejsów) dla pakietu **org.xml.sax.`**

Typ	Nazwa	Opis
klasy	**AttributesImpl**	Domyślna implementacja *Attribute*-dodawanie i usuwanie atrybutów
	DefaultHandler	Domyślna klasa bazowa dla obsługi zdarzeń SAX2 - puste implementacje interfejsów obsługi SAX: EntityResolver, DTDHandler, ContentHandler, ErrorHandler
	LocatorImpl	Implementacja interfejsu ***Locator***, ustawienie numerów wierszy i kolumn
	NamespaceSupport	Obsługa przestrzeni nazw
	ParserAdapter	Adaptacja parsera SAX1 do SAX2
	XMLFilterImpl	Domyślna implementacja ***Filter*** ; implementuje XMLFilter, EntityResolver, DTDHandler, ContentHandler
	XMLReaderAdapter	Adaptacja parsera SAX2 do SAX1
	XMLReaderFactory	Tworzenie implementacji XMLReader wg nazwy lub właściwości

Bardzo ważną klasą pomocniczą jest klasa **XMLReaderFactory** do tworzenia egzemplarza parsera.

Funkcje klasy XMLReaderFactory	
`static XMLReader`	**`createXMLReader`**`()` tworzenie obiektu typy XMLReader na podstawie właściwości systemowej `org.xml.sax.driver`
`static XMLReader`	**`createXMLReader`**`(String className)` tworzenie obiektu typy XMLReader na podstawie nazwy klasy parsera.

Metody powyższe nie będą użyteczne w środowiskach, gdzie właściwości systemowe są niedostępne lub program nie może ładować dynamicznie klas.

Przykład utworzenia egzemplarza XMLReader:

```
try { XMLReader myReader = XMLReaderFactory.createXMLReader(); }
catch (SAXException e) { System.out.println(e.getMessage());  }
```

Na zakończenie podamy jeszcze dwa interfejsy rozszerzające możliwości analizy.

Interfejsy pakietu **org.xml.sax.ext**

Typ	Nazwa	Opis
interfejsy	***DeclHandler***	Rozszerzenie obsługi zdarzeń deklaracji DTD w SAX2
	LexicalHandler	Rozszerzenie obsługi zdarzeń leksykalnych w SAX2:DTD, encji, komentarzy

---

Przykład szablonu programu wykorzystującego do przetwarzania dokumentu XML poznane interfejsy i klasy. Czytelnik powinien znaleźć w tym programie omawiane interfejsy i klasy i zaobserwować ich funkcjonalność uruchamiając ten program.

```
//import poszczególnych klas potrzebnych do przetwarzania
//można zastąpić importem całych pakietów, ale tak widać które
//konkretnie klasy są wykorzystywane w programie

import java.io.*;

import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.Locator;

import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

//import klasy producenta parsera SAX potrzebny
//gdy chcemy utworzyć jego obiekt za pomocą new SAXParser()
import org.apache.xerces.parsers.SAXParser;

class SAX {

  public static void main(String[] args)
  {
    String xmlResource="";

    int index = 0;

    index = Integer.parseInt(args[0]);
```

```
    // nazwy plików XML
    String[] file = {"item.xml","chessboard.xml","contents.xml"};

    try {

      // pobranie ścieżki pliku do przetworzenia
      xmlResource = "file:\\" + new File(file[index]).getAbsolutePath();

      System.out.println(xmlResource);
    }
    catch(Exception e){}

    System.out.println("Przetwarzanie pliku: " + xmlResource + "\n");

    // utworzenie obiektu obsługi zawartości
    ContentHandler contentHandler = new MyContentHandler();

    //utworzenie obiektu obsługi błędów
    ErrorHandler errorHandler = new MyErrorHandler();

    try {
     // utworzenie instancji parsera
     XMLReader parser =
       XMLReaderFactory.createXMLReader(
         "org.apache.xerces.parsers.SAXParser");

     //inny sposób utworzenia parsera SAX
     //XMLReader parser = new SAXParser();

     System.out.println("parser= "+ parser);

     // zarejestrowanie obiektu obsługi zawartości
     parser.setContentHandler(contentHandler);

     // zarejestrowanie obiektu obsługi błędów
     parser.setErrorHandler(errorHandler);

     // wyłączenie sprawdzania poprawności
     parser.setFeature(
        "http://xml.org/sax/features/validation",false);

     // włączenie obsługi przestrzeni nazw
     parser.setFeature(
        "http://xml.org/sax/features/namespaces",true);

     // analiza dokumentu XML
     parser.parse(xmlResource);

  }
  catch (IOException e) {
    System.out.println("Błąd czytania pliku XML: " + e.getMessage());
  }
  catch (SAXException e) {
          System.out.println("Błąd analizy pliku XML: " + e.getMessage());
  }
 } //main()
} // class SAXParser

class MyContentHandler implements ContentHandler {

    String data = "";
```

```
    private Locator locator; //przechowuje obiekt lokalizacji zdarzeń

    //ustalenie obiektu lokalizującego
    public void setDocumentLocator(Locator locator) {
      this.locator = locator;
      System.out.println("Ustalenie obiektu lokalizującego");
    }

    //początek przetwarzania dokumentu
    public void startDocument() throws SAXException {
      System.out.println("Początek przetwarzania");
    }

    //koniec przetwarzania dokumentu
    public void endDocument() throws SAXException {
      System.out.println("Koniec przetwarzania");
    }

    //napotkanie instrukcji przetwarzania
    public void processingInstruction(String target, String data)
    throws SAXException {

      System.out.println("PI: Cel PI:" + target + "  Dane:" + data);
    }

    //początek odwzorowania przestrzeni nazw
    public void startPrefixMapping(String prefix, String uri) {
      System.out.println(
        "Przedrostek " + prefix +" odwzorowywany na " + uri
      );
    }

    //koniec odwzorowywania przestrzeni nazw
    public void endPrefixMapping(String prefix) {
      System.out.println("koniec odwzorowywania przedrostka " + prefix);
    }

    //początek elementu;pobranie nazwy elementu,atrybutu i jego wartości
    public void startElement(
          String namespaceURI, String localName,String rawName, Attributes
atts
    )
    throws SAXException {

      System.out.print("początek elementu " + localName);
        if (!namespaceURI.equals("")) {
          System.out.println(
                " nazwa elementu " + namespaceURI + " [" + rawName + "]");
        }
        else System.out.println(" brak przestrzeni nazw");

        for (int i=0; i<atts.getLength(); i++)
          System.out.println(
                "  Atrybut: " + atts.getLocalName(i) +"=" +
atts.getValue(i));

         //dane znakowe elementu
         System.out.println("dane elementu " + localName + ":" + data);

    }//startElement()
```

```
    //koniec elementu
    public void endElement(
      String namespaceURI, String localName, String rawName
    )
    throws SAXException {

      System.out.println("koniec elementu: " + localName + "\n");
    }

    //dane znakowe elementu
    public void characters(char[] ch, int start, int end)
    throws SAXException {

      data = new String(ch, start, end);
    }

    //pominięte białe znaki
    public void ignorableWhitespace(char[] ch, int start, int end)
    throws SAXException {

        String s = new String(ch, start, end);
        System.out.println("pominięte białe znaki: [" + s + "]");
    }

    //nie analizowane encje
    public void skippedEntity(String name) throws SAXException {
        System.out.println("Nie analizowane encja " + name);
    }
}

class MyErrorHandler implements ErrorHandler  {

    //ostrzeżenie:czegoś brakuje lub coś jest niewłaściwe
    public void warning(SAXParseException e)
    throws SAXException {

        System.out.println(
             "*Ostrzeżenie w czasie analizy*\n" +
             "  Linia:    "  + e.getLineNumber() + "\n" +
             "  URI:     "   + e.getSystemId()   + "\n" +
             "  Komunikat: " + e.getMessage());
        throw new SAXException("Ostrzeżenie");
    }

    //błąd niekrytyczy;naruszona zasada XML;kontynuacja analizy
    public void error(SAXParseException e)
    throws SAXException {

      System.out.println(
           "**Parsing Error**\n" +
           "  Linia:    "  + e.getLineNumber() + "\n" +
           "  URI:     "   + e.getSystemId()   + "\n" +
           "  Komunikat: " + e.getMessage());
        throw new SAXException("Błąd niekrytyczny");
    }

    //błąd krytyczny;naruszone zasada XML;niemożność kontynuacji analizy
    public void fatalError(SAXParseException e)
    throws SAXException {
```

```
        System.out.println(
            "**Parsing Fatal Error**\n" +
            "  Linia:    "   + e.getLineNumber() + "\n" +
            "  URI:      "   + e.getSystemId()   + "\n" +
            "  Komunikat: " + e.getMessage());
        throw new SAXException("Błąd krytyczny");
    }

}//class SAX
```

4.4. Model DOM

Model **DOM** (Document Object Model) został zdefiniowany przez W3C DOM Working Group. DOM umożliwia dostęp do danych oraz manipulowanie danymi.

W interfejsie DOM dokument XML reprezentowany jest jako struktura drzewa- cały dokument XML jest wczytywany do pamięci a wszystkie dane umieszczane są w węzłach tego drzewa. Aplikacja może działać teraz na strukturze drzewa przeszukując węzły i przetwarzając dane w węzłach.

Jednakże umieszczanie całego dokumentu w pamięci powoduje dla dużych dokumentów wyraźne spowolnienie przetwarzania a nawet uniemożliwienie dalszego działania aplikacji.

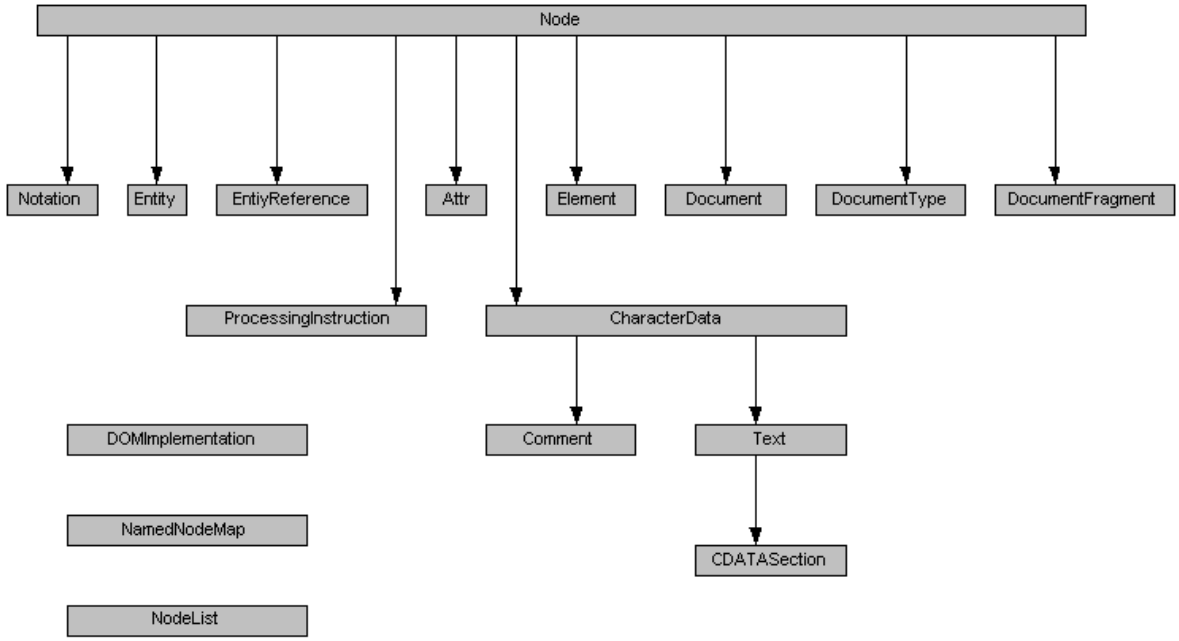
Odpowiedni dla modelu DOM pakiet Javy to: **org.w3c.dom**.Nie będziemy ty razem dokładnie omawiać tych interfejsów. Podamy później ich zastosowanie na konkretnym przykładzie programu analizującego.

Interfejsy i klasy pakietu **org.w3c.dom**

Typ	Nazwa	Opis
Interfejsy	***Attr***	Ustawianie wartości atrybutu, dostęp do nazwy I wartości
	CDATASection	Rozszerza Text; znacznik sekcji CDATA
	CharacterData	Dostęp do węza tekstowego, ustawianie jego wartości, operacje na znakach
	Comment	Reprezentacja (znacznik) komentarza
	Document	Reprezentacja dokumentu XML; tworzenie nowych elementów XML
	DocumentFragment	Operowanie na części obiektu Document
	DocumentType	Reprezentacja deklaracji DOCTYPE z dokumentu XML
	DOMImplementation	Udostępnienie implementacji parsera DOM określonego producenta
	Element	Reprezentacja elementu XML; nazwy I atrybuty elementów, ustawianie wartości

	Entity	Reprezentacja encji; dostęp do identyfikatorów
	EntityReference	Reprezentacja wyniku przetworzenia encji
	NamedNodeMap	Lista węzłów nazwanych
	Node	Główny interfejs dla DOM; operacje na węzłach
	NodeList	Kolekcja węzlów
	Notation	Reprezentacja konstrukcji NOTATION z DTD(deklarowanie encji lub PI)
	ProcessingInstruction	Reprezentacja instrukcji przetwarzania (PI)
	Text	Reprezentacja danych tekstowych elementu XML
Klasy wyjątków	**DOMException**	Zgłaszany w wyniku błędu przetwarzania, zawiera kody błędów

Struktura drzewa w modelu **DOM:**



Nie będziemy tutaj omawiali funkcjonalności tych interfejsów - przedstawimy niektóre z metod na konkretnym przykładzie analizy dokumentu XML w modelu **DOM.**

```
import java.io.*;

//importy pakietów DOM
import org.w3c.dom.Document;
import org.w3c.dom.DocumentType;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//import klasy parsera danego producenta
import org.apache.xerces.parsers.DOMParser;

public class DOM {
```

```
 public static void main (String args []) throws IOException {

    int index = 0;

    index = Integer.parseInt(args[0]);

    // nazwy plików XML
    String[] file={"item.xml","chessboard.xml","contents.xml"};

    try {

    // pobranie ścieżki do pliku do przetworzenia
    String xmlResource = "file:\\" + new
File(file[index]).getAbsolutePath();

    System.out.println(xmlResource);

    //utworzenie parsera danego producenta
    DOMParser parser = new DOMParser();

    //ustawienie cechy parsera 'validation'
    parser.setFeature("http://xml.org/sax/features/validation",false);

    //ustawienie cechy parsera 'namespaces'
    parser.setFeature("http://xml.org/sax/features/namespaces",true);

    //dokonanie analizy
    parser.parse(xmlResource);

    //uzyskanie wyniku analizy - obiektu typu Document
    Document  doc = parser.getDocument();

    //wydrukowanie zawartości drzewa którego węzłem jest obiekt typu
Document
    printTree(doc, "");

   }
   catch(Exception e) {e.printStackTrace();}

  }//main()

  public static void printTree(Node node,String insets)
  {
    switch(node.getNodeType()){ //określenie typu węzła

    case Node.DOCUMENT_NODE: //węzeł typu Document

      //DOM-Level2 nie udostępnia deklaracji XML
      System.out.println("<xml version=\"1.0\">\n");

      Document doc=(Document)node;

      //pobranie elementu głównego i wywołanie rekurencyjne printTree()
      printTree(doc.getDocumentElement(),"");

      break;

   case Node.ELEMENT_NODE: //węzeł typu NODE

     String name=node.getNodeName();
```

```
      System.out.print(insets+"<"+name);

      NamedNodeMap attributes=node.getAttributes();

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

        Node current=attributes.item(i);

        System.out.print(" "+current.getNodeName()+

          "=\""+current.getNodeValue()+"\"");

      }

      System.out.print(">"); //formatowanie

      //rekurencyjne przetwarzanie elementów potomnych
      NodeList children = node.getChildNodes();

      if(children!=null)
        for(int i=0;i<children.getLength();i++)
          printTree(children.item(i),insets + " ");

      break;

    case Node.TEXT_NODE: //węzeł typu Text

      //wyświetlenie danych tekstowych
      System.out.print(node.getNodeValue());

      break;

    case Node.CDATA_SECTION_NODE: //węzeł typu CDATASection

      //wyświetlenie danych tekstowych z bloku CDATA
      System.out.print(node.getNodeValue());

      break;

    case Node.PROCESSING_INSTRUCTION_NODE: //węzeł typu
ProcessingInstruction

      //wyświetlenie  instrukcji przetwarzania PI
      System.out.print("<?"+node.getNodeName()+
        "  "+node.getNodeValue()+  "?>");

      break;

    case Node.ENTITY_REFERENCE_NODE: //węzeł typu EntityReference

      //wyświetlenie encji...

      break;

    case Node.DOCUMENT_TYPE_NODE: //węzeł typu DocumentType

      //wyświetlenie deklaracji DTD...

      break;
```

```
  } //switch
 } //printTree()
} //class DOM
```

Jak widać z tego przykładu cała struktura elementów XML w postaci drzewa jest przetwarzana przez program.

Porównanie głównych cech interfejsów **SAX** i **DOM** umożliwia poniższa tabela:

Interfejs SAX	**Interfejs DOM**
Dane pobierane w obsłudze zdarzeń	Dane pobierane ze struktury drzewa
Sekwencyjny dostęp do danych	Swobodny dostęp do danych
Małe zużycie pamięci	Znaczne zużycie pamięci
Przetwarzanie w pamięci części dokumentu	Przetwarzanie w pamięci całego dokumentu
Przetwarzanie jednokrotne dokumentu	Przetwarzanie wielokrotne dokumentu

Pewnej alternatywy w stosunku do obu modeli dostarcza model JDOM, który omawiamy w następnym punkcie.

4.5. Model JDOM

JDOM został wyspecyfikowany przez Bretta McLaughlina i Jasona Huntera (K&A Sofware).

Interfejs **JDOM** jest zamiennikiem (w większości zastosowań) interfejsu **SAX** lub **DOM** bazującym na Javie ale nie jest oparty ani na **SAX** ani na **DOM**.

Pozwala utworzyć dokument XML o strukturze drzewa bez stosowania rozwiązań typowych dla DOM, a jednocześnie jest bardzo szybki podobnie jak SAX.

Ponadto zawiera konkretne klasy (a nie tylko interfejsy) umożliwiające bezpośrednie tworzenie obiektów.

Odpowiednie dla modelu JDOM pakiety Javy to : **org.jdom, org.jdom.input, org.jdom.output,org.jdom.adapters, org.jdom.filter, org.jdom.transform**

Interfejsy i klasy pakietu **org.jdom**

Typ	Nazwa	Opis
Interfejsy	***Attribute***	reprezentacja atrybutu XML; uzyskanie wartości atrybutu; informacja o przestrzeni nazw
	CDATA	reprezentacja sekcji CDATA z dokumentu XML
	Comment	Tekst komentarza
	DocType	Reprezentacja deklaracji DOCTYPE z dokumentu XML
	Document	reprezentacja dokumentu XML; ustawienie i pobranie wartości DocType i listy instrukcji PI
	Element	Reprezentacja elementu XML z obsługą przestrzeni nazw
	EntityRef	Reprezentacja referencja zawartej w encji
	NameSpace	Reprezentacja przestrzeni nazw; tworzenie przestrzeni nazw
	ProcessingInstruction	Reprezentacja instrukcji PI; pobranie i ustawienie danych instrukcji
	Text	zawartość tekstowa dokumentu XML
	Verifier	Weryfikacja nazw, danych i innych komponentów XML
Klasy wyjątków	**DataConversionException**	Rozszerza JDOMException; błąd konwersji elementu *Attribute* lub *Element* na określony typ
	IllegalADDException	dodawanie nielegalnego obiektu do struktury JDOM
	IllegalDataException	dodawanie nielegalnego tekstu do struktury JDOM
	IllegalNameException	dodawanie nazwy do struktury JDOMnie spełniającej konwencji XML
	IllegalTargetException	dodawanie nielegalnego obiektu docelowego do struktury JDOMo niewłaściwej nazwie
	JDOMException	Podstawowy wyjątek JDOM; komunikaty o błędach

Klasy pakietu **org.jdom.adapters :**

Pakiet ten zawiera klasy adapterów pozwalające na uzyskanie obiektu DOM Document z dowolnego parsera DOM

Typ	Nazwa	Opis
Interfejs	***DOMAdapter***	Interfejs, który musi by zaimplementowany przez klasy adapterów
	AbstractDOMAdapter	Implementacja DOMAdapter
	CrimsonDOMAdapter	Adapter dla parsera Crimson

Klasy	**JAXPDOMAdapter**	Adapter dla parsera JAXP
	OracleV1DOMAdapter	Adapter dla parsera Oracle Version1
	OracleV2DOMAdapter	Adapter dla parsera Oracle Version2
	ProjectXDOMAdapter	Adapter dla parsera Sun Project X
	XercesDOMAdapter	Adapter dla parsera Apache Xerces
	XML4JDOMAdapter	Adapter dla parsera IBM XML4J DOM

Interfejsy i klasy pakietu **org.jdom.input**

Typ	**Nazwa**	**Opis**
Interfejsy	***JDOMFactory***	Implementowany do tworzenia obiektów JDOM
Klasy	**BuilderErrorHandler**	Implementuje org.xml.sax.ErrorHandler
	DefaultJDOMFactory	Tworzenie klas JDOM(Element,Document,Comment
	DOMBuilder	Tworzenie obiektu JDOM Document na bazie parsera
	SAXBuilder	Tworzenie obiektu JDOM Document na bazie parsera
	SAXHandler	Obsługa i pomoc dla SAX Builder

Interfejsy i klasy pakietu **org.jdom.output**

Typ	Nazwa	Opis
klasy	**DOMOutputter**	Przetwarza drzewo JDOM na drzewo DOM
	SAXOutputter	dla drzewa JDOM generuje zdarzenia SAX2
	XMLOutputter	Obsługa wyjściowego obiektu Document; wysłanie do strumienia OutputStream w formacie XML

Interfejsy i klasy pakietu **org.jdom.filter**

Typ	Nazwa	Opis
Interfejsy	***Filter***	Filtrowanie listy obiektów JDOM
Klasy	**ContentFilter**	Implementacja *Filter*; filtrowanie zawartości elementu
	ElementFilter	Implementacja *Filter*; filtrowanie elementów

Obiekt **ContentFilter** opisuje wszystkie dozwolone obiekty JDOM i pozwala ustalać widoczność tych obiektów. Filtrowanie jest dokonywane za pomocą odpowiedniej maski w której każdy bit informuje o tym czy obiekt JDOM ma być widoczny czy nie.

Przykładowo w celu uwidocznienia węzłów typu Text i CDATA w elemencie **elem** użyć można następujących instrukcji:

```
 Filter filter =  new ContentFilter(ContentFilter.TEXT |
ContentFilter.CDATA);
 List content = elem.getContent(filter);
```

Alternatywą dla maskowania bitów jest użycie odpowiednich funkcji filtrujących jak w przykładzie, gdzie chcemy uwidocznić tylko węzły typu **Comment:**

```
   Filter filter =  new ContentFilter();

   Filter.setCommentVisible(true);
   List content = elem.getContent(filter);
```

Klasy pakietu **org.jdom.transform**

Typ	Nazwa	Opis
Klasy	**JDOMResult**	Przechowuje wynik transformacji XSLT w postaci dokumentu JDOM
	JDOMSource	Stanowi źródło dokumentu JDOM dla transformacji XSLT

Następujący przykład pokazuje jak zastosować transformację **XSLT** do dokumentu JDOM i uzyskać wynik w postaci innego dokumentu JDOM:

```
   public static Document transform(Document in, String stylesheet)
     throws JDOMException {

     try {

       Transformer transformer =
       TransformerFactory.newInstance().newTransformer(new
StreamSource(stylesheet));

       JDOMResult out = new JDOMResult();
       transformer.transform(new JDOMSource(in), out);
       return out.getDocument();
     }
     catch (TransformerException e) {
       throw new JDOMException("XSLT Transformation failed", e);
     }
   }
```

Następny program prezentuje przetwarzanie dokumentu XML za pomocą interfejsów JDOM. Do sformatowania wyników przetwarzania użyto obiektu **XMLOutputter.**

```
import java.io.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
```

```
import org.w3c.dom.*;
import org.xml.sax.*;

import javax.xml.parsers.*;

//----------------------------------------------------------------

public class JDOM {

  public static void main (String args [])
  throws
IOException,JDOMException,ParserConfigurationException,SAXException
  {
    int index=0;

    index=Integer.parseInt(args[0]);

    String xmlFile="";

    // nazwy plików XML
    String[] file = {"item.xml","chessboard.xml","contents.xml"};

    try {

      // pobranie ścieżki pliku do przetworzenia
      xmlFile = "file:\\" + new File(file[index]).getAbsolutePath();

      System.out.println(xmlFile);

     }
     catch (Exception e) { e.printStackTrace(); }

     System.out.println();

     System.out.println(
        "### budowanie dokumentu JDOM za pomocą SAXBuilder z pliku XML "
     );

     // budowanie dokumentu JDOM za pomocą SAXBuilder z pliku XML
     saxDocument(xmlFile);

     System.out.println();

     // budowanie dokumentu JDOM za pomocą DOMBuilder z dokumentu DOM
     System.out.println(
       "### budowanie dokumentu JDOM za pomocą DOMBuilder z dokumentu DOM "
     );

     domDocument(xmlFile);

    System.exit(0);

  }//main()


  public static void saxDocument(String fileName)
  throws IOException,JDOMException
  {
    //utworzenie SAXBuilder bez sprawdzania poprawności dokumentu
    SAXBuilder builder = new SAXBuilder(false);
```

```
    //utworzenie obiektu typu JDOM Document
    org.jdom.Document doc = builder.build(fileName);

    //wydrukowanie dokumentu wyjściowego w formacie XML
    printDocument(doc);

  }

  public static void domDocument(String fileName)
  throws
ParserConfigurationException,SAXException,IOException,JDOMException
  {
    //utworzenie fabryki
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

    //utworzenie obiektu DocumentBuilder
    DocumentBuilder build = dbf.newDocumentBuilder();

    //utworzenie obiektu typu DOM-Document o strukturze drzewa
    org.w3c.dom.Document domDoc = build.parse(fileName);

    //brak sprawdzania poprawności dokumentu
    DOMBuilder builder = new DOMBuilder(false);

    //utworzenie obiektu typu JDOM-Document
    org.jdom.Document jdomDoc = builder.build(domDoc);

    //wydrukowanie dokumentu wyjściowego w formacie XML
    printDocument(jdomDoc);

  }

  public static void printDocument(org.jdom.Document doc)throws IOException
  {
    XMLOutputter fmt = new XMLOutputter();

    System.out.println(

"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    );

    fmt.output(doc,System.out);

    System.out.println(

"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    );

  }//printDocument()


}//class JDOM
```

Obiekt **XMLOutputter** jest szczególnie użyteczny gdy nie mamy gotowego dokumentu XML lecz tworzymy go dynamicznie od podstaw.

Tworzenie dokumentu od podstaw i wyprowadzanie w postaci sformatowanej do strumienia standardowego i do pliku ilustruje poniższy program.

```
import java.io.*;
import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.Namespace;
import org.jdom.output.XMLOutputter;

class XMLGenerator {
  public static void main(String[] args) {

   //uzyskanie obiektu Namespace
   Namespace ns =  Namespace.getNamespace("JavaXML",
"http://www.pjwstk.edu.pl/javaxml/");

   // utworzenie korzenia pustego drzewa
   Element root = new  Element("drzewo-rodowe", ns);
   Document doc = new Document(root);

   //tworzenie węzłów i  dodawania na odpowiednich poziomach
   Element dziadek=new Element("dziadek",ns) ;
   dziadek.setAttribute("wiek","80").addContent("Jan");
   root.addContent(dziadek);

   Element ojciec=new Element("ojciec",ns) ;
   ojciec.setAttribute("wiek","50").addContent("Kazimierz");
   dziadek.addContent(ojciec);

   Element syn=new Element("syn",ns) ;
   syn.setAttribute("wiek","20").addContent("Ryszard");
   ojciec.addContent(syn);

   Element corka=new Element("corka",ns) ;
   corka.setAttribute("wiek","18").addContent("Katarzyna");
   ojciec.addContent(corka);

   try {
     //konfiguracja obiektu formatującego: 2 spacje wcięć,nowa linia
     XMLOutputter fmt = new  XMLOutputter(" ",true);

     //wydrukowanie na konsolę sformatowanego dokumentu
     fmt.output(doc,System.out);

     //wydrukowanie do pliku sformatowanego dokumentu
     FileOutputStream fos = new  FileOutputStream("drzewo.xml");
     fmt.output(doc, fos);
   }
   catch(IOException e){e.printStackTrace();}

 }//main()

}//class XMLGenerator
```

zawartość pliku drzewo.xml po zadziałaniu programu

```
<?xml version="1.0" encoding="UTF-8"?>
<JavaXML:drzewo-rodowe xmlns:JavaXML="http://www.pjwstk.edu.pl/javaxml/">
  <JavaXML:dziadek wiek="80">
     Jan
    <JavaXML:ojciec wiek="50">
     Kazimierz
     <JavaXML:syn wiek="20">Ryszard</JavaXML:syn>
     <JavaXML:corka wiek="18">Katarzyna</JavaXML:corka>
    </JavaXML:ojciec>
  </JavaXML:dziadek>
</JavaXML:drzewo-rodowe>
```

4.6. Zastosowanie JAXP

Stosowanie interfejsów **SAX i DOM** wymaga importowania i odwoływania się do klas parsera danego producenta, a za tym idzie przy zmianie parsera wymaga zmiany kodu i rekompilacji.

Wyjściem z tej sytuacji jest stosowanie interfejsu **JAXP** (Java API for XML Parsing).Tutaj klasę parsera definiujemy za pomocą właściwości systemowej "javax.xml.parsers.SaxParserFactory" lub "javax.xml.parsers.DocumentBuilderFactory" przy użyciu opcji -D w linii komend lub **System.setProperty()** w kodzie Javy. Zmiana implementacji parsera wymaga tylko zmiany tej właściwości.

Podstawowy pakiet JAXP to **javax.xml.parsers.**

Interfejsy i klasy pakietu **javax.xml.parsers**

Typ	Nazwa	Opis

Klasy	**DocumentBuilderFactory**	Tworzenie egzemplarzy DocumentBuilder; włączenie/wyłączenieobsługi przestrzeni nazw lub sprawdzania poprawności
	DocumentBuilder	Implementowana przez parser DOM; przetwarzanie niezależnie od producenta
	SAXParserFactory	Tworzenie egzemplarzy SAXParser; włączenie/wyłączenie obsługi przestrzeni nazw lub sprawdzania poprawności
	SAXParser	Implementowana przez parser SAX; przetwarzanie niezależnie od producenta
Klasy wyjątków	**ParserConfigurationException**	Błąd zgłoszenia pobrania parsera gdy podane ustawienia są niewłaściwe
	FactoryConfigurationError	Nie jest możliwe utworzenie egzemplarza klasy

Tworzenie egzemplarzy **DocumentBuilderFactory** lub **SAXParserFactory** możliwe jest za pomocą statycznej metody **newInstance()** z tych klas.

Metoda ta używa następującego porządku poszukiwania klasy implementującej do załadowania do JVM:

- **sprawdza właściwość systemową "javax.xml.parsers.DocumentBuilderFactory/SAXParserFactory"**
- **korzysta z pliku konfiguracyjnego "jre/lib/jaxp.properties" zawierającego kwalifikowaną nazwę klasy implementującej.**
- **uruchamia Services API do poszukiwania nazwy klasy w plikach .jar dostępnych na ścieżce klas.**
- **stosuje domyślną instancję klasy dostępną na platformie.**

Porządek ten należy uwzględniać chcąc uzyskać działanie określonego parsera.

Poniższy program pokazuje wykorzystanie klas **JAXP** do przetwarzania niezależnego od producenta w modelu **SAX.**

```
import java.io.*;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Enumeration;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

//import klas JAXP
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
```

```
class SAX_JAXP {

  public static void main (String args []) throws IOException {

    int index = 0;

    index = Integer.parseInt(args[0]);

    // nazwy plików XML
    String[] file = {"item.xml","chessboard.xml","contents.xml"};

    try {

      // pobranie ścieżki pliku do przetworzenia
      String xmlResource = "file:\\" + new
File(file[index]).getAbsolutePath();

      System.out.println(xmlResource);

      // utworzenie obiektu SAXParserFactory
      SAXParserFactory spf = SAXParserFactory.newInstance();

      // utworzenie obiektu SAXParser
      SAXParser sp = spf.newSAXParser();

      System.out.println("parser = " + sp);

      //obsługa przestrzeni nazw
      spf.setNamespaceAware(true);

      // utworzenie obiektu SAXHandler do obsługi zdarzeń
      SAXHandler handler = new SAXHandler();

      //uruchomienie analizy z podaną obsługą zdarzeń
      sp.parse(xmlResource, handler);

      // pobranie kolekcji wynikowych
      ArrayList[] tagData=handler.getArrays();

      // wydrukowanie zawartości kolekcji
      System.out.println("----Znaczniki i dane znaczników----------");

      for(int i=0;i<tagData[0].size();i++){

        System.out.println("<"+tagData[0].get(i)+">"+"
"+tagData[1].get(i));
      }
    }
    catch (Exception e) { e.printStackTrace(); }

    System.exit(0);
  }
}//class SAX_JAXP

//--------------------------------------------------

class SAXHandler extends DefaultHandler {

//DefaultHandler -
```

```
//puste implementacje ContentHandler,ErrorHandler,DTDHandler,EntityResolver


//--------implementacja ContentHandler-----------------

  private Locator locator;

  // utworzenie dwóch kolekcji typu ArrayList do nazw znaczników i danych
  private ArrayList[] list=new ArrayList[]{new ArrayList(),new
ArrayList()};

  private String currentElement = null;
  private String currentData = null;

  public void setDocumentLocator(Locator locator)
  {
    this.locator = locator;
  }

  public void startDocument()throws SAXException
  {
    System.out.println("==== start
document:line="+locator.getLineNumber());
  }

  public void endDocument()throws SAXException
  {
    System.out.println("=== stop document:line="+locator.getLineNumber());

  }

  //napotkano instrukcję przetwarzania PI
  public void processingInstruction(String target,String data)
  throws SAXException {
    System.out.println("PI: target="+target+" data="+data);
  }
  //początek odwzorowywania przedrostka przestrzeni nazw
  public void startPrefixMapping(String prefix,String uri)
  throws SAXException {
    System.out.println("prefix = "+prefix+":"+uri);
  }

  //koniec odwzorowywania przedrostka przestrzeni nazw
  public void endPrefixMapping(String prefix)
  throws SAXException {

    System.out.println("prefix =" + prefix);
  }

  // metoda dostępu do przetworzonych wartości-obiekty ArrayList
  public ArrayList[] getArrays()
  {
    return list;
  }

  // metoda wywoływana kiedy analizowany jest nowy element
  public void
  startElement(String nsUri,String localName,String tag, Attributes attrs)
  throws SAXException {

    //zapamiętanie etykiety nowego elementu
```

```
    currentElement = tag;
    System.out.println("TAG =" + tag);

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

          System.out.println("line = " + locator.getLineNumber()+
            ":  atrybut " + attrs.getQName(i) + "=" + attrs.getValue(i));

        }
        System.out.println("data for " + tag + " tag" + ":" + currentData);
  }

  // informacja o białych znakach
  public void ignorableWhitespace(char[] ch,int start,int end)
  throws SAXException
  {

  }

  // wywoływana po znalezieniu danych w elemencie
  public void characters(char[] ch, int start, int length)
  throws SAXException {

    //utworzenie łańcucha ze znaków znalezionych w elemencie
    currentData = new String(ch, start, length).trim();

    if(currentData.equals(""))currentData = null;

    if (currentData != null) {
      list[0].add(currentElement);
      list[1].add(currentData);
    }

  }

  // wywoływana po znalezieniu znacznika końca elementu
  public void endElement(String nsUri,String localName,String tagEnd)
  throws SAXException
  {
    // znacznik końca elementu tagEnd
    System.out.println("TAG-END = " + tagEnd);

  }

  // gdy parser pomija encję
  public void skippedEntity(String name)throws SAXException
  {
    System.out.println("skipped entity");
  }

  //----------koniec implementacji ContentHandler----------

  //----------implementacja ErrorHandler------------------

  //ostrzeżenie - niepoprawność składniowa dokumentu,
  //niezgodność z definicjami DTD
  public void warning(SAXParseException e)
  throws SAXException
  {

System.out.println("*warning:"+e.getMessage()+"/line="+e.getLineNumber());
```

```
  }

  //błąd niekrytyczny - niezgodność ze specyfikacją XML
  public void error(SAXParseException e)
  throws SAXException
  {

System.out.println("*error:"+e.getMessage()+"/line="+e.getLineNumber());
  }

  //błąd krytyczny - niepoprawne formatowanie dokumentu
  //zatrzymanie procesu przetwarzania
  public void fatalError(SAXParseException e)
  throws SAXException
  {
    System.out.println("*fatal
error:"+e.getMessage()+"/line="+e.getLineNumber());
  }
  //--------koniec imlementacji ErrorHandler--------------

}//class SAXHandler
```

Poniższy program pokazuje wykorzystanie klas **JAXP** do przetwarzania niezależnego od producenta w modelu **DOM.**

```
import java.io.*;

import org.w3c.dom.*;

import org.xml.sax.*;

//import klas JAXP
import javax.xml.parsers.*;

class DOM_JAXP {

  public static void main (String args []) throws IOException {

    int index = 0;

    index = Integer.parseInt(args[0]);

    // nazwy plików XML
    String[] file = {"item.xml","chessboard.xml","contents.xml"};

    try {

      // pobranie ścieżki pliku do przetworzenia
      String xmlResource = "file:\\" +
        new File(file[index]).getAbsolutePath();

      System.out.println(xmlResource);

      // utworzenie obiektu DocumentBuilderFactory
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

      //utworzenie obiektu DOMBuilder
      DocumentBuilder builder = dbf.newDocumentBuilder();
```

```
    System.out.println("parser= " + builder);

    //utworzenie obiektu typu DOM-Document o strukturze drzewa
    Document doc = builder.parse(xmlResource);

    //rekurencyjne wyświetlenie zawartości drzewa
    System.out.println();

    printTree(doc,"");

   }
   catch (Exception e) { e.printStackTrace(); }

  System.exit(0);

}//main()

public static void printTree(Node node,String insets)
{
  switch(node.getNodeType()){

  case Node.DOCUMENT_NODE:
    //DOM-Level2 nie udostępnia deklaracji XML
    System.out.println("<xml version=\"1.0\">\n");

    Document doc=(Document)node;
    printTree(doc.getDocumentElement(),"");
    break;

  case Node.ELEMENT_NODE:

    String name=node.getNodeName();
    System.out.print(insets+"<"+name);
    NamedNodeMap attributes=node.getAttributes();

    for(int i=0;i<attributes.getLength();i++){
      Node current=attributes.item(i);
      System.out.print(" "+current.getNodeName()+
                       "=\""+current.getNodeValue()+"\"");

    }
    System.out.print(">"); //formatowanie

    //rekurencyjne przetwarzanie elementów potomnych
    NodeList children=node.getChildNodes();

    if(children!=null)
      for(int i=0;i<children.getLength();i++)
        printTree(children.item(i),insets + " ");


    break;

  case Node.TEXT_NODE:
    //wyświetlenie danych tekstowych
    System.out.print(node.getNodeValue());
    break;

  case Node.CDATA_SECTION_NODE:
    //wyświetlenie danych tekstowych z bloku CDATA
```

```
      System.out.print(node.getNodeValue());
      break;

    case Node.PROCESSING_INSTRUCTION_NODE:
      //wyświetlenie instrukcji przetwarzania PI
      System.out.print("<?"+node.getNodeName()+
                       " "+node.getNodeValue()+ "?>");
      break;

    case Node.ENTITY_REFERENCE_NODE:
      //wyświetlenie encji
      break;

    case Node.DOCUMENT_TYPE_NODE:
      //wyświetlenie deklaracji DTD
      break;

  }//switch

 }//printTree()

}//class DOM_JAXP
```

4.7. Przetwarzanie względem DTD

Specyfikacja DTD

Zadanie definicji DTD jest określenie sposobu formatowania danych.
Zdefiniowany musi być każdy element dozwolony w dokumencie XML, sposoby zagnieżdżania elementów, atrybuty oraz zewnętrzne encje.

Konstrukcje DTD umożliwiają :

a) określanie elementów

<!ELEMENT [nazwa elementu] [definicja/typ elementu]>

typ ANY oznacza element mogący zawierać dane tekstowe, inne elementy oraz kombinacje obu poprzednich

typ EMPTY oznacza element pusty

typ (#PCDATA) oznacza typ przetwarzanych danych tekstowych

b) określanie sposobu zagnieżdżania elementów

<!ELEMENT [nazwa elementu] ([zagnieżdżony element] ,[zagnieżdżony element]...)>

c) grupowanie elementów, operatory rekurencji (wielokrotne występowanie, powtórzenia)

<!ELEMENT [nazwa elementu] ((grupa1Element1,grupa1Element2), (grupa2Element1,grupa2Element2))>

Do elementów jak również do każdej grupy elementów można stosować operatory rekurencji określający ile razy ma pojawić się dany element czy dana grupa elementów.

Operator rekurencji	Opis operatora
Domyślnie	Musi wystąpić dokładnie raz
?	Musi wystąpić raz albo wcale
+	Musi wystąpić przynajmniej raz (1...n razy)
*	Może wystąpić dowolną liczbę razy(0...n razy)

d) definiowanie atrybutów i typy atrybutów

<!ATTLIST [element zamykający] [nazwa atrybutu] [typ] [modyfikator]>

typ CDATA oznacza atrybut o wartości tekstowej

typ wyliczeniowy (wartość1|wartość2|...) umożliwia określenie wartości atrybutu

modyfikatory określają czy atrybut jest wymagany w danym elemencie :

modyfikator #IMPLIED – atrybut nie jest wymagany

modyfikator #REQUIRED – atrybut jest wymagany

modyfikator #FIXED – określa że uzytkownik nie może zmienić wartości atrybutu

e) określanie encji

<!ENTITY [nazwa encji] ”[znaki podstawiane/identyfikator]”>

Konstrukcja ta jak widać pozwala podać zarówno znaki podstawiane pod nazwę encji jak i odwołanie do pliku zewnętrznego.W tym ostatnim przypadku trzeba podać adres URI (np. URL) zasobu.

f) określanie encji nieprzetwarzanej

Encje nieprzetwarzane występują w dokumentach XML odwołujących się do danych binarnych (np.plików multimedialnych).

Ponieważ parser nie potrafi przetwarzać plików binarnych powinien te dane pozostawić w postaci nieprzetworzonej.

Przykładowo jeżeli w dokumencie XML mamy znacznik z encją:

<myImage>&Image</myImage>

to w dokumencie DTD umieszczamy następującą konstrukcję:

<!ENTITY Image SYSTEM "images/duke.gif" NDATA gif>

Wystąpienie tej deklaracji w dokumencie DTD spowoduje wywołanie wsteczne funkcji **unparsedEntityDecl()** z interfejsu *DTDHandler.*

Niezbędnym warukiem do tego jest przetwarzanie dokumentu przez parser z zarejestrowaną implementacją *DTDHandler.*

g) deklaracje notacji

Deklaracje notacji skojarzone są z encjami nieprzetwarzanymi i dla powyższej deklaracji ma postać :

<!NOTATION gif SYSTEM "http//www.gif.com">

Deklaracja ta wiąże typ nieprzetwarzanej encji(gif) z identyfikatorem URI danego typu.

Wystąpienie tej deklaracji w dokumencie DTD spowoduje wywołanie wsteczne funkcji **notationDecl()** z interfejsu *DTDHandler.*

Oczywistym warunkiem tego jest przetwarzanie dokumentu przez parser z zarejestrowaną implementacją *DTDHandler.*

Podajemy teraz przykłady dokumentów DTD. W przykładach tych czytelnik powinien rozpoznać omawiane konstrukcje DTD.

Dokument chessboard.dtd

```
<!ELEMENT CHESSBOARD (WHITEPIECES, BLACKPIECES)>

<!ENTITY % pieces "KING, QUEEN?, BISHOP?, BISHOP?, ROOK?,
ROOK?, KNIGHT?, KNIGHT?, PAWN?, PAWN?, PAWN?, PAWN?, PAWN?,
PAWN?, PAWN?, PAWN?" >

<!ELEMENT WHITEPIECES (%pieces;)>

<!ELEMENT BLACKPIECES (%pieces;)>

<!ELEMENT POSITION EMPTY>

<!ATTLIST POSITION COLUMN (A|B|C|D|E|F|G|H) #REQUIRED ROW
(1|2|3|4|5|6|7|8) #REQUIRED >

 <!ELEMENT KING (POSITION)>

<!ELEMENT QUEEN (POSITION)>

<!ELEMENT BISHOP (POSITION)>

 <!ELEMENT ROOK (POSITION)>

<!ELEMENT KNIGHT (POSITION)>

<!ELEMENT PAWN (POSITION)>
```

Dokument contents.dtd

```
 <!ELEMENT JavaXML:Book (JavaXML:Title, JavaXML:Contents,
JavaXML:Copyright)>

<!ATTLIST JavaXML:Book xmlns:JavaXML CDATA #REQUIRED >

<!ELEMENT JavaXML:Title (#PCDATA)>

<!ELEMENT JavaXML:Contents ((JavaXML:Chapter+) (JavaXML:Chapter+,
JavaXML:SectionBreak?)+)>

<!ELEMENT JavaXML:Chapter (JavaXML:Heading?,JavaXML:Topic+)>

<!ATTLIST JavaXML:Chapter focus (XML|Java) "Java" >
```

```
<!ELEMENT JavaXML:Heading (#PCDATA)>

<!ELEMENT JavaXML:Topic (#PCDATA)>

<!ATTLIST JavaXML:Topic  subSections CDATA #IMPLIED >

<!ELEMENT JavaXML:SectionBreak EMPTY>

<!ELEMENT JavaXML:Copyright (#PCDATA)>

<!ENTITY OReillyCopyright
SYSTEM   "http://www.oreilly.com/catalog/javaxml/docs/copyright.xml">
```

4.7.1. Przetwarzanie względem DTD w modelu SAX

Poniżej przedstawiono przykładowy program do analizy XML w modelu **SAX** ze sprawdzaniem poprawności według definicji **DTD**. Do utworzenia instancji parsera wykorzystano biblioteki **JAXP1.0.**

```
import java.io.*;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Enumeration;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;

public class SAX_Valid_DTD {

  public static void main (String args []) throws IOException {

    // nazwy plików XML
    String[]  file={"item.xml","chess.xml","contents.xml"};
    try {
      // pobranie ścieżki pliku do przetworzenia
      String xmlResource =  "file:\\" + new
File(file[2]).getAbsolutePath();
      System.out.println(xmlResource);

      // utworzenie obiektu SAXParserFactory
      SAXParserFactory spf =  SAXParserFactory.newInstance();

      //sprawdzanie poprawności dokumentu XML
      spf.setValidating(true);

      // utworzenie obiektu SAXParser
      SAXParser sp =  spf.newSAXParser();

      //obsługa przestrzeni nazw
      spf.setNamespaceAware(true);
```

```
      // utworzenie obiektu SAXHandler do obsługi zdarzeń
      SAXHandler handler = new SAXHandler();

      //uruchomienie analizy z podaną obsługą zdarzeń
      sp.parse(xmlResource, handler);

      // pobranie kolekcji wynikowych
      ArrayList[] tagData=handler.getArrays();

      // wydrukowanie zawartości kolekcji
      System.out.println("\n----Znaczniki i dane  znaczników----------");

      for(int i=0;i<tagData[0].size();i++){

         System.out.println("<"+tagData[0].get(i)+">"+"
"+tagData[1].get(i));
      }
    }
    catch (Exception e) {  e.printStackTrace(); }
    System.exit(0);
    }
   }
   //-------------------------------------------------

   class SAXHandler extends  DefaultHandler {

    //...tak jak w programie SAX_JAXP w punkcie 4.6

   }//class SAXHandler
```

4.7.2. Przetwarzanie względem DTD w modelu DOM

```
import java.io.*;

import org.w3c.dom.*;

import org.xml.sax.*;

import javax.xml.parsers.*;

//-----------------------------------------------------------------

class DOM_Valid_DTD {

 public static void main (String[] args) throws IOException {

   // nazwy plików XML
   String[]  file={"item.xml","chess.xml","contents.xml"};

   try {
     // pobranie ścieżki pliku do przetworzenia
     String xmlResource =  "file:\\" + new File(file[2]).getAbsolutePath();
     System.out.println(xmlResource);

     // utworzenie obiektu DocumentBuilderFactory
     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

```
      //sprawdzenie poprawności dokumentu XML
      dbf.setValidating(true);

      //utworzenie obiektu  parsera
      DocumentBuilder  builder=dbf.newDocumentBuilder();

      //utworzenie własnej obsługi błędów
      ErrorHandler eh=new MyHandler();
      //zarejestrowanie własnej obsługi błędów
      builder.setErrorHandler(eh);

      System.out.println();

      //utworzenie obiektu typu DOM-Document o strukturze drzewa
      Document doc=builder.parse(xmlResource);
      //rekurencyjne wyświetlenie zawartości drzewa
      printTree(doc);
    }
    catch (Exception e) { e.printStackTrace(); }

    System.exit(0);

  }//main()

  public static void printTree(Node node) {

    //...tak jak w programie DOM_JAXP w punkcie 4.6

  }//printTree()

}//class DOM_Valid_DTD

class MyHandler implements ErrorHandler {

    //...tak jak w programie DOM_JAXP w punkcie 4.6

}//class MyHandler
```

4.7.3. Przetwarzanie względem DTD w modelu JDOM

W tym przypadku wywołujemy metodę **setValidating(true)** z klasy fabryki **DocumentBuilderFactory.**

```
import java.io.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

import org.w3c.dom.*;

import org.xml.sax.*;

import javax.xml.parsers.*;

//----------------------------------------------------------------------
```

```
class JDOM_Valid_DTD {

  public static void main (String[] args)
  throws
IOException,JDOMException,ParserConfigurationException,SAXException
  {
    String xmlFile="";
    // nazwy plików XML
    String[] file={"chess.xml","contents.xml"};

    try {
    // pobranie ścieżki pliku do przetworzenia
    xmlFile = "file:\\" + new  File(file[1]).getAbsolutePath();
    System.out.println(xmlFile);
  }
  catch (Exception e) { e.printStackTrace(); }

  System.out.println();
  System.out.println(
      "####### budowanie dokumentu JDOM za pomocą SAXBuilder z pliku XML "
  );

  // tworzenie dokumentu JDOM za pomocą SAXBuilder z pliku XML
  saxDocument(xmlFile);

  System.out.println();

  // budowanie dokumentu JDOM za pomocą DOMBuilder z dokumentu DOM
  System.out.println(
    "####### budowanie dokumentu JDOM za pomocą DOMBuilder z dokumentu DOM
"
  );

  domDocument(xmlFile);

  System.exit(0);

 }//main()

 public static void saxDocument(String fileName)
 throws IOException,JDOMException
 {
   //utworzenie SAXBuilder ze sprawdzaniem  poprawności dokumentu
   SAXBuilder builder=new  SAXBuilder(true);

   //utworzenie obiektu typu JDOM Document
   org.jdom.Document doc = builder.build(fileName);

   //wydrukowanie dokumentu wyjściowego w postaci kodu XML
   printDocument(doc);
 }
 public static void domDocument(String fileName)
 throws ParserConfigurationException,SAXException,IOException,JDOMException
 {
   DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

   //sprawdzenie poprawności względem DTD
   dbf.setValidating(true);

   DocumentBuilder build=dbf.newDocumentBuilder();
```

```
   //klasa wewnętrzna lokalna do własnej obsługi błędów
   class MyHandler implements  ErrorHandler {

     //ostrzeżenie - niepoprawnosc skladniowa dokumentu,
     //niezgodność z definicjami DTD
     public void  warning(SAXParseException e)
     throws SAXException
     {
         System.out.println("***********warning: line="+e.getLineNumber());
     }

     //błąd niekrytyczny - niezgodność ze specyfikacją XML
     public void error(SAXParseException e)
     throws SAXException
     {
        System.out.println("*********error: line="+e.getLineNumber());
     }

     //błąd krytyczny - niepoprawne formatowanie dokumentu
     //zatrzymanie procesu przetwarzania
     public void fatalError(SAXParseException e)
     throws SAXException
     {
        System.out.println("*********fatal error:
line="+e.getLineNumber());
     }

     }//class MyHandler

     ErrorHandler eh=new MyHandler();

     //zarejestrowanie własnej obsługi błędów
     build.setErrorHandler(eh);

     //utworzenie obiektu typu DOM-Document o strukturze drzewa
     org.w3c.dom.Document domDoc = build.parse(fileName);

     //sprawdzanie poprawności dokumentu
     DOMBuilder builder = new DOMBuilder(true);

    //utworzenie obiektu typu JDOM-Document
    org.jdom.Document jdomDoc = builder.build(domDoc);

    //wydrukowanie dokumentu wyjściowego w postaci kodu XML
    printDocument(jdomDoc);

  }//domDocument()

  public static void printDocument(org.jdom.Document doc)throws IOException
  {
    XMLOutputter fmt = new XMLOutputter();

    System.out.println(

"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    );

    fmt.output(doc,System.out);

    System.out.println(
```

```
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
    );

  }//printDocument()
}//class JDOM_Valid_DTD
```

4.8. Przetwarzanie względem XML Schema

Specyfikacja XML Schema

W przeciwieństwie do dokumentu DTD, który korzysta z ze specyficznego formatu opisu elementów, XML Schema stosuje format dokumentu XML.

Konstrukcje XML Schema umożliwiają :

a) określanie przestrzeni nazw dokumentu XML i przestrzeni nazw schematu

<xsd:schema targetNamespace="http://www.java.sun.com/javaxml"

xmlns:xsd="http://www.w3.org./2003/XMLSchema"

xmlns:JavaXML="http://www.java.sun.com/javaxml">

b) określanie elementów i sposobu ich zagnieżdżania

-----> ***określanie elementów jawnych (nazwanych)***

<element name="nazwa elementu" type="typ elementu" [opcje]>

w nazwie elementu nie powinno być przedrostka przestrzeni nazw

"typ elementu" określa typ predefiniowany lub typ zdefiniowany przez użytkownika

Typy predefiniowane :

Typ	Podtypy	Znaczenie
String	NMTOKEN, Language	Łańcuchy znaków
Boolean	---	Binarna wartość
Float	---	32 bitowy typ zmiennoprzecinkowy
Double	---	64 bitowy typ zmiennoprzecinkowy

Decimal	Integer	Zapis dziesiętny
TimeInstant	---	Data i czas
TimeDuration	---	Czas trwania
RecurringInstant	Date,time	Czas powtarzający się przez okres timeDuration
Binary	---	Dane w postaci binarnej
Uri	Enumeration	Identyfikator zasobów

Np. **<element name="title" type="string" />**

Typy zdefiniowane przez użytkownika określamy za pomocą elementu **complexType:**

<complexType name="[nazwa typu]">

<[specyfikacja elementu]>

<[specyfikacja elementu]>

...

</complexType>

---> ***określanie elementów niejawnych (nienazwanych)***

<complexType>

...

</complexType>

c) grupowanie elementów, operatory rekurencji (wielokrotne występowanie, powtórzenia)

Do określania ile razy ma pojawić się element używa się atrybutów **minOccurs** i **maxOccurs**:

<element name="[nazwa]" type="[typ]" minOccurs="[ile]" maxOccurs="[ile]">

wartości domyślne **minOccurs** = 1, **maxOccurs**= *(zero lub więcej).

d) definiowanie atrybutów, typu atrybutów i wartości domyślnych

Atrybuty w XMLSchema definiuje się za pomocą elementu attribute

<attribute name=”[nazwa atrybutu” type=”[typ atrybutu]” [opcje atrybutu]>

Do określania czy dany atrybut ma się pojawić służy **minOccurs** (wartość domyślna=0).

Do określania wartości domyślnej atrybutu służy atrybut **default**

Np. **<attribute name=”temat” type=”string” default=”java” / >**

Wyliczenia możliwych wartości atrybutu dokonujemy za pomocą elementu **simpleType** podając jego typ bazowy za pomocą słowa kluczowego **base** i elementów **enumeration**

np .

```
<attribute name=”temat” default=”java”>
  <simpleType base=”string”>
    <enumeration value=”xml” />
    <enumeration value=”javaxml” />
  </simple type>
</attribute>
```

e) określanie encji

<!ENTITY [nazwa encji] ”[znaki podstawiane/identyfikator]”>

Konstrukcja ta jak widać pozwala podać zarówno znaki podstawiane pod nazwę encji jak i odwołanie do pliku zewnętrznego.W tym ostatnim przypadku trzeba podać identyfikator URI (np. URL) zasobu.

Przykład dokumentu **JavaXML.xsd** zawężającego dokument **contents.xml** wedlug XML Schema. Czytelnik powinien rozpoznać w nim konstrukcje charakterystyczne dla XML Schema.

Dokument JavaXML.xsd

```
<?xml version="1.0"?>

<schema targetNamespace="http://www.oreilly.com/catalog/javaxml/"
```

```
 xmlns="http://www.w3.org/1999/XMLSchema"
 xmlns:JavaXML="http://www.oreilly.com/catalog/javaxml/">
<element name="Book" type="JavaXML:BookType" />
 <complexType name="BookType">
    <element name="Title" type="string" />
    <element name="Contents" type="JavaXML:ContentsType" />
    <element name="Copyright" type="string" />
 </complexType>
 <complexType name="ContentsType">
  <element name="Chapter" maxOccurs="*">
     <complexType>
        <element name="Heading" type="string" minOccurs="0" />
        <element name="Topic" maxOccurs="*">
           <complexType content="string">
              <attribute name="subSections" type="integer" />
           </complexType>
        </element>
        <attribute name="focus" default="Java">
          <simpleType base="string">
            <enumeration value="XML" />
            <enumeration value="Java" />
          </simpleType>
        </attribute>
    </complexType>
```

```
  </element>

  <element name="SectionBreak" minOccurs="0" maxOccurs="*">

    <complexType content="empty" />

  </element>

 </complexType>

</schema>
```

źródło: http://www.oreilly.com/catalog/javaxml

Obecnie dostępny jest w Internecie pakiet **JAXP** w wersji 1.2 (**JAXP1.2**) który obsługuje sprawdzanie poprawności zarówno względem **DTD, j**ak również względem **XMLSchema**. Dołączany jest do niego standardowo parser Xerces-J obsługujący XMLSchema.

Poniższy przykład pokazuje wykorzystanie bibliotek **JAXP1.2** do przetwarzania dokumentu XML ze sprawdzaniem poprawności według **DTD** lub **XMLSchema.**

Poniższy program stosuje biblioteki **JAXP1.2** do obliczania ilości elementów w dokumencie XML:

Program stosujący biblioteki JAXP1.2.Należy uruchomić ten program i sprawdzić czy wynik jest zgodny z przewidywaniami. Należy następnie dokonać celowych zmian w dokumencie XML niezgodnych z arkuszem stylów i zaobserwować reakcję programu. Dla analizy ze sprawdzaniem poprawności względem XMLSchema program ten należy uruchomić z argumentami: **-xsdss personal.xsd personal-schema.xml** pamiętając o umieszczeniu na ścieżce klas parsera Xerces-J obsługującego **JAXP1.2.**

```
//klasy JAXP
import javax.xml.parsers.*;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

import java.util.*;
import java.io.*;

public class SAXLocalNameCount extends DefaultHandler {

    //stałe używane przez  JAXP 1.2
    static final String JAXP_SCHEMA_LANGUAGE =
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
    static final String W3C_XML_SCHEMA =
        "http://www.w3.org/2001/XMLSchema";
    static final String JAXP_SCHEMA_SOURCE =
```

```
        "http://java.sun.com/xml/jaxp/properties/schemaSource";

    //tablica mieszania z parami (nazwa znacznika,liczba wystąpień)
    private Hashtable tags;

    //na poczatku dokumentu tworzymy obiekt HashTable
    public void startDocument() throws SAXException {
        tags = new Hashtable();
    }

    // po napotkaniu nowego elementu pobieramy jego nazwę i sprawdzamy
    // czy już występuje w tablicy jeżeli nie to dodajemy do mapy;
    // jeżeli już występuje to dodajemy do mapy z aktualnym licznikiem
    public void startElement(
      String namespaceURI, String localName,String qName, Attributes atts
    )
    throws SAXException {

        String key = localName;
        Object value = tags.get(key);
        if (value == null) {
          // dodajemy nową parę
          tags.put(key, new Integer(1));
        }
        else {
          // pobieramy wartość licznika,zwiekszamy o 1 i dodajemy do mapy
          int count = ((Integer)value).intValue();
          count++;
          tags.put(key, new Integer(count));
        }
    }

    // na końcu dokumentu uzyskujemy iterator kluczy
    // i drukujemy nazwy znaczników i liczby ich wystąpień
    public void endDocument() throws SAXException {
        Enumeration e = tags.keys();
        while (e.hasMoreElements()) {
            String tag = (String)e.nextElement();
            int count = ((Integer)tags.get(tag)).intValue();
            System.out.println("Local Name \"" + tag + "\" occurs " + count
                               + " times");
        }
    }

    //nazwę pliku zamieniamy na adres URL pliku
    private static String convertToFileURL(String filename) {

        String path = new File(filename).toURL().toString();
        return "file:" + path;
    }
    // w przypadku błędnych argumentów
    // informacja o parametrach wywołania programu z linii komend
    private static void usage() {
      System.err.println("Usage: SAXLocalNameCount [-options] <file.xml>");
      System.err.println(" -dtd = DTD validation");
      System.err.println(
        " -xsd | -xsdss <file.xsd> = W3C XML Schema validation using xsi:
hints");
      System.err.println("  in instance document or schema source
<file.xsd>");
      System.err.println(
```

```
        " -xsdss <file> = W3C XML Schema validation using schema source
<file>");
      System.err.println(" -usage or -help = this message");
      System.exit(1);
    }

    static public void main(String[] args) throws Exception {

        String filename = null;
        boolean dtdValidate = false; //czy poprawność według DTD
        boolean xsdValidate = false; //czy poprawność według informacji w
dokumencie XML
        String schemaSource = null;  //czy poprawność według podanego
arkusza XML Schema

        // analiza argumentów wywołania
        for (int i = 0; i < args.length; i++) {

            if (args[i].equals("-dtd")) dtdValidate = true;

            else if (args[i].equals("-xsd")) xsdValidate = true;

            else if (args[i].equals("-xsdss")) {
                if (i == args.length - 1) usage();
                xsdValidate = true;
                schemaSource = args[++i];
            }
            else if (args[i].equals("-usage")) usage();

            else if (args[i].equals("-help")) usage();

            else {
                filename = args[i];
                if (i != args.length - 1) usage();
            }

        }//for

        if (filename == null) usage();

        // utworzenie fabryki JAXP SAXParserFactory
        SAXParserFactory spf = SAXParserFactory.newInstance();

        // włączenie rozpoznawania przestrzeni nazw.
        spf.setNamespaceAware(true);

        // ustawienie sprawdzania poprawności
        spf.setValidating(dtdValidate || xsdValidate);

        // utworzenie parsera SAX
        SAXParser saxParser = spf.newSAXParser();
        System.out.println("parser="+saxParser);

        // ustawienie języka schematu
        if (xsdValidate) {
            try {
                saxParser.setProperty(JAXP_SCHEMA_LANGUAGE,
W3C_XML_SCHEMA);
            }
            catch (SAXNotRecognizedException x) {
                // jeżeli parser nie obsługuje JAXP1.2
```

```
                System.err.println(
                    "Error: JAXP SAXParser property not recognized: "+
JAXP_SCHEMA_LANGUAGE
                );
                System.err.println("Check to see if parser conforms to JAXP
1.2 spec.");

                System.exit(1);
            }
        }

        //ustawienie właściwości dla źródła schematu
        if (schemaSource != null) {
            saxParser.setProperty(JAXP_SCHEMA_SOURCE, new
File(schemaSource));
        }

        // uzyskanie obiektu XMLReader
        XMLReader xmlReader = saxParser.getXMLReader();

        // zarejestrowanie obsługi zawartości
        xmlReader.setContentHandler(new SAXLocalNameCount());

        // zarejestrowanie obsługi błędów z wydrukiem na konsolę
        xmlReader.setErrorHandler(new MyErrorHandler(System.err));

        // dokonanie analizy dokumentu XML
        xmlReader.parse(convertToFileURL(filename));
    }

    // definicja obsługi błędów i ostrzeżeń
    private static class MyErrorHandler implements ErrorHandler {

        //informacje o błędach zapisywane do strumienia
        private PrintStream out;

        MyErrorHandler(PrintStream out) {
            this.out = out;
        }

        //dostarcza informacji o wyjatkach w procesie parsowania
        private String getParseExceptionInfo(SAXParseException spe) {

            String systemId = spe.getSystemId();

            if (systemId == null) systemId = "null";

            String info = "URI=" + systemId +
                " Line=" + spe.getLineNumber() +
                ": " + spe.getMessage();
            return info;
        }

        // standardowe metody obsługi błędów

        public void warning(SAXParseException spe) throws SAXException {
            out.println("Warning: " + getParseExceptionInfo(spe));
        }

        public void error(SAXParseException spe) throws SAXException {
            String message = "Error: " + getParseExceptionInfo(spe);
```

```
            throw new SAXException(message);
        }

        public void fatalError(SAXParseException spe) throws SAXException {
            String message = "Fatal Error: " + getParseExceptionInfo(spe);
            throw new SAXException(message);
        }
    }
}
```

źródło - Apache Software Foundation (http://www.apache.org/)

4.9. Transformacje XSLT

Transformacja **XSLT** (XSL Transformation) była zdefiniowana przez W3C XSL Working Group.

XSL (Extensible Stylesheet Language) służy do przekształcenia danych XML z jednego formatu na inny za pomocą arkuszy stylów, które muszą być uprzednio przygotowane. Dokument **XSL** jest dokumentem w formacie XML wykorzystującym konstrukcje XPath oraz obiekty formatujące w postaci specjalnych znaczników używanych przez procesor **XSLT** do zmiany formatu danych.

Procesor **XSLT,** wykorzystując arkusz stylów oraz dokument XML w postaci drzewa, tworzy dokument w nowym formacie np. HTML.

Jako argumenty dla procesora **XSLT** podaje się dokument XML z odwołaniem do arkusza stylów, arkusz stylów XSL oraz plik w którym ma być umieszczony wynik transformacji.

Xpath (XML Path Language) określa w jaki sposób zlokalizować określony komponent dokumentu XML.

Z punktu widzenia **Xpath** dokument XML stanowi strukturę drzewiastą . W węzłach tego drzewa znajduje się określone komponenty XML : elementy, atrybuty, dane tekstowe. W celu zlokalizowania określonego komponentu XML stosuje się odwołania do węzłów drzewa XML poprzez określenie położenia danego węzła w drzewie.

Transformacje **XSLT** umożliwiają w Javie pakiety **javax.xml.transform, javax.xml.transform.sax,javax.xml.transform.dom, javax.xml.transform.stream.**

Interfejsy i klasy pakietu **javax.xml.transform**

Pakiet dostarcza interfejsy i klasy do przetwarzania instrukcji transformacyjnych **XSL** i dokonywania transformacji **XSLT.**

Typ	Nazwa	Opis
Interfejsy	***ErrorListener***	Nasłuch ostrzeżeń i błędów przetwarzania
	Result	Uzyskiwanie Informacji potrzebnych do zbudowania wynikowego drzewa transformacji
	Source	Uzyskiwanie informacji potrzebnych do działania jako źródła transformacji (dokumentu XML lub instrukcji transformacyjnych)
	SourceLocator	Uzyskiwanie Informacji o występowaniu błędów w źródle transformacji
	Templates	Reprezentacja przetwarzanych instrukcji transformacyjnych
	URIResolver	Rozpoznawanie identyfikatorów URI w instrukcjach transformacyjnych
Klasy	**OutputKeys**	Zawiera stałe łańcuchowe potrzebne do ustalania lub pobierania właściwości obiektu typu Transformer lub Template
	Transformer	Przetwarzanie drzewa źródłowego w drzewo wynikowe
	TransformerFactory	Fabryka do tworzenia instancji klasy Transformer
Klasy Wyjątków	**TransformerConfigurationException**	Nie można utworzyć instancji klasy Transformer np. z powodu błędów składniowych instrukcji
	TransformerException	Ogólny błąd transformacji; informacje o błędzie poprzez wywołanie getMessageAndLocation()
	TransformerFactoryConfigurationError	Błąd konfiguracji instancji TransformerFactory; klasa fabryki transformacji nie znaleziona lub nie można utworzyć jej egzemplarza

Interfejsy i klasy pakietu javax.xml.transform.sax

Zawiera API specyficzne dla transformacji w modelu SAX2

Typ	Nazwa	Opis

Interfejsy	***TemplatesHandler***	Tworzenie obiektów typu Templates w oparciu o zdarzenia SAX2
	TransformerHandler	Tworzenie transformacji w oparciu o zdarzenia SAX2
Klasy	**SAXResult**	Pozwala na ustalenie obiektu ContentHandler dla zdarzeń SAX2 pochodzących z procesu transformacji
	SAXSource	Ustalenie obiektu XMLReader i InputSource
	SAXTransformerFactory	rozszerza TransformerFactory, dostarczając metod do tworzenia instancji TemplatesHandler, TransformerHandler, XMLReader

Interfejsy i klasy pakietu javax.xml.transform.dom

Zawiera API specyficzne dla transformacji w modelu DOM:

Typ	Nazwa	Opis
Interfejs	***DOMLocator***	Wskazuje pozycję węzła w strukturze DOM; raport błędów
Klasy	**DOMResult**	Określa drzewo wyjściowe jako obiekt typu Node
	DOMSource	Określa drzewo wejściowe jako obiekt typu Node

Klasy pakietu javax.xml.transform.stream

Typ	Nazwa	Opis
klasy	**StreamResult**	przechowuje wynik transformacji w odpowiednim formacie (np.XML,HTML)
	StreamSource	przechowuje źródło transformacji jako strumień znaczników XML

```
import java.io.*;

import org.w3c.dom.*;

import org.xml.sax.*;

import javax.xml.parsers.*;

//importy pakietów XSLT
import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
```

```
//--------------------------------------------------------------------

public class XSLT {

  public static void main (String args []) throws IOException {

    int index = 1;

    index = Integer.parseInt(args[0]);

    // nazwy plików XML,XSL,HTML
    String[] xmlFile = {"contents.xml","contents.xml" ,"contents.xml"};
    String[] xslFile = {"contents.xsl","contentsA.xsl" ,"contentsB.xsl"};
    String[] htmlFile =
{"contents.html","contentsA.html","contentsB.html"};

    try {

      // utworzenie obiektu TransformerFactory
      TransformerFactory tf = TransformerFactory.newInstance();

      //uzyskanie procesora XSLT
      Transformer transformer = tf.newTransformer(
        new SAXSource(new InputSource(xslFile[index])));

      //utworzenie strumienia wysciowego dla wyniku transformacji
      OutputStream out = new FileOutputStream(htmlFile[index]);

      //dokonanie transformacji
      transformer.transform(
        new SAXSource(new InputSource(xmlFile[index])),new
StreamResult(out)
      );

      System.out.println();

     }
     catch (Exception e) { e.printStackTrace(); }

     System.exit(0);

  }//main()

}//class XSLT
```

Jak widać zastosowanie interfejsów do transformacji XSLT jest bardzo proste. Cała trudność polega na przygotowaniu arkuszy XSL opisujących na czym ma polegać transformacja (zmiana formatu) dokumentu XML. Omówimy zatem podstawy tworzenia takich arkuszy.

4.9.1. Zastosowanie języka Xpath

XPath stanowi samodzielną specyfikację którą można znaleźć na stronie http://www.w3org/TR/xpath.

Jej pełny opis w tym wykładzie jest niemożliwy więc zaznaczone będą tylko główne zasady posługiwania się tym językiem.

Jak już wspomniano **Xpath** służy do odwołania się do poszczególnych elementów i ich atrybutów dokumentu XML.

Odwołania te są czynione przy użyciu adresów względnych (względem elementu bieżącego) lub absolutnych w postaci ścieżki dostępu do danego węzła (i jego potomków) i ewentualnie jego atrybutu. Przy przetwarzaniu wyrażenia **XPath** zwracany jest zestaw węzłów, który może być poddany różnym operacjom m.in. przekształceniu na inny format.

Oprócz samego wyboru węzłów **XPath** udostępnia funkcje operujące na zestawach węzłów t.j. **not()** i **count().**

Funkcje te omówione zostaną w zastosowaniu razem z konstrukcjami arkusza **XSL.**

4.9.2. Tworzenie arkusza stylów w języku XSL (XML Stylesheet Language)

Rozszerzalny język arkuszy stylów (informacje o XSL można uzyskać na stronie http://www.w3.org/Style/XSL) służy do tworzenia arkuszy stylów. Składa się ze zbioru słów w formacie XML służących do formatowania dokumentu XML.

Główne jego składniki to:

- **szablony XSL**
- **filtry**
- **instrukcje iteracji (pętle)**
- **instrukcje wyboru**

Szablony XSL

mają ogólną postać:

```
<xsl:template match="[wyrażenie XPath]">

        <!-- tu wstawiamy sposób formatowania -->

</xsl:template>
```

nazwa **template** oznacza szablon a atrybut **match** oznacza dopasowanie do sposobu formatowania elementu którego ścieżkę podano w wyrażeniu **XPath.**

Jako dokument XML który posłuży do zaprezentowania konstrukcji XSL wybierzmy plik **contents.html** zawierający spis treści książki "Java and XML" (autor: Brett MacLaughlin)

Jeżeli teraz w arkuszu XSL umieścimy szablon

```
<xsl:template match="Java:Book">

   Hello XML!

</xsl:template>
```

to ponieważ element **Java:Book** jest elementem głównym to do szablonu zastanie zwrócona cała hierarchia elementów dokumentu XML.Procesor XSLT przetwarza tę hierarchię elementów i po napotkaniu każdego węzła dodaje dane do strumienia wyjściowego transformacji. Jeżeli dla danego elementu nie podano szablonu dane wynikowe nie będą nic zawierały

W tym przypadku wynikowy dokument transformacji zawierać będzie napis **Hello XML!.**

Jeżeli chcielibyśmy żeby procesor dopasował wszystkie elementy podrzędne elementu bieżącego za pomocą wszystkich szablonów w arkuszu to trzeba użyć konstrukcji **<xsl:apply-templates> ...</xsl:apply-templates>**

Następujący arkusz przekształca dokument XML do formatu HTML i w jego ciele umieszcza dane ze wszystkich elementów XML.

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3c.org/2000/xsl/transform"
"http://www.oreily.com/catalog/java.xml" version="1.0">

<xsl:template match="Java:Book">

   <html>

      <head><title>Java Book Html</title></head>

    <body>

       <xsl:apply-templates />

     </body>
```

</html>

</xsl:template>

</xsl:stylesheet>>

Wynikowy dokument **HTML** ma postać:

<html xmlns:JavaXML="http://www.oreily.com./catalog/javaxml" >

<head><title>Java Book Html</title></head>

<body>

Java and XML

Introduction

What is it?

How Do I Use It?

Why should I Use It?

Creating XML

An XML Document

The Header

The Content What's next?

<!-- pozostałe rozdziały -->

</body>

</html>

Po tym przykładzie powstaje pytanie jak dopasowywać indywidualnie elementy XML.

Służy do tego konstrukcja **<xsl:value-of select="[sciezka XPath]">**

Zatem żeby w poprzednim przykładzie znacznik html **<title>** zawierał dane z elementu **<JavaXML:Title>**

trzeba użyć szablonu:

<xsl:template match="JavaXML:Book">

<html>

<head>

<title><> <xsl:value-of select="JavaXML:Title" /> </title>

</head>

<body>

<xsl:apply-templates />

</body>

</html>

</xsl:template>

Istotne w tym przykładzie jest to że mimo że wartość elementu **JavaXML:Title** została wstawiona do znacznika html **<title>** to element ten nie został usunięty z hierarchii elementów dostępnych w szablonie. Zatem wartość tego elementu pojawi się w tytule dokumentu html jak również w jego ciele tak jak w poprzednim przykładzie.

Wynika to z ogólnej zasady że dane wejściowe procesora **XSLT** są niezmienne- można najwyżej coś do nich dodać i dokonać przekształcenia formatu.

Filtry

Często zachodzi potrzeba wyłączenia z przekształcania pewnych węzłów struktury drzewiastej dokumentu XML.

W takim przypadku najwygodniej jest wykorzystać funkcję XPath o nazwie **not().**

Jeżeli w poprzednim przykładzie chcielibyśmy wyłączyć z przetwarzania element **JavaXML:Title** zagnieżdżony w elemencie bieżącym **Java:Book** to odpowiedni szablon powinien wyglądać następująco:

<xsl:template match="JavaXML:Book">

<html>

<head>

<title><> <xsl:value-of select="JavaXML:Title" /> </title>

</head>

<body>

<xsl:apply-templates select="*[not(self::JavaXML:Title)]">

</body>

</html>

</xsl:template>

Słowo **self** oznacza w tym przypadku że węzły występujące po tym słowie są potomne względem węzła bieżącego.

Instrukcje iteracyjne (pętle)

mają postać **<xsl:for-each select="[wyrażenie XPath]">...</xsl:for-each>** i służą do iteracji po danych w ramach jednego typu elementu ;
np. w elemencie **<Java:XML:Contents>** chcemy wydrukować tytuły rozdziałów- dane elementów **< JavaXML : Heading >**

Wówczas nasz szablon który formatuje wydruk w postaci html mógłby wyglądać tak:

<xsl:template match="JavaXML:Contents">

<h1>Contents</h1>

<ul>

<xsl:for-each select="JavaXML:Chapter">

** <xsl:value-of select="JavaXML:Heading" /> **

<xsl:for-each>

**
**

</xsl:template>

Instrukcje wyboru

pozwalają przetwarzać tylko te węzły które spełniają pewne kryteria wyboru.

Podstawową konstrukcją iteracyjną jest **<xsl:if test="[wyrażenie logiczne]">...<xsl:if>**

Jeżeli wynikiem testu jest prawda to element **<xsl:if >** będzie obliczany w przeciwnym razie nie.

Jeżeli zatem w poprzednim przykładzie chcemy wypisać tylko te rozdziały których atrybut **focus** ma wartość XML, możemy to uzyskać w następujący sposób:

```
<xsl:template match="JavaXML:Contents">
 <h1>Contents</h1>
 <ul>
  <xsl:for-each select="JavaXML:Chapter">
   <xsl:if test="@focus='XML' ">
    <li> <xsl:value-of select="JavaXML:Heading" /> </li>
   <xsl:if>
  <xsl:for-each>
 </ul>
</xsl:template>
```

Następną pożyteczną konstrukcją jest konstrukcja złożona **<xsl:choose>... </xsl:choose>** o postaci:

```
<xsl:choose>
        <xsl:when test="[wyrażenie logiczne]">...</xsl:when>
        <xsl:otherwise>...</xsl:otherwise>
    </xsl:choose>
```

Pozwala ona na dokonanie testu warunku **<xsl:when test=..>**.i wykonanie jednej operacji jeżeli jest spełniony i innej operacji jeżeli warunek nie jest spełniony **<xsl:otherwise>....**

Ostatni szablon zmodyfikujemy w ten sposób żeby wypisywał nazwę elementu i w zależności od niej tematykę.

```
<xsl:template match="JavaXML:Contents">
 <h1>Contents</h1>
<ul>
 <xsl:for-each select ="JavaXML:Chapter">
  <xsl:choose>
    <xsl:when test ="@focus ='Java' ">
      <li> <xsl:value-of select ="JavaXML:Heading" />(focus:Java) </li>
    </xsl:when>
   <xsl:otherwise>
     <li> <xsl:value-of select="JavaXML:Heading" />(focus:XML) </li>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:for-each>
</ul>
</xsl:template>
```

Dodatkową możliwością w XML jest konstrukcja **<xsl:copy-of select="[wyrażenie XPath]">** która przekazuje zestaw węzłów bezpośrednio na wyście procesora XSLT. Zestaw węzłów nie jest wówczas przetwarzany.

Na zakończenie tego krótkiego wprowadzenia w arkusze XSL warto zaznaczyć, że daje on również możliwość definiowania własnych węzłów i atrybutów poprzez konstrukcje:

<xsl:element name="nazwa elementu">...</xsl:element>

<xsl:attribute name="nazwa atrybutu">...</xsl:attribute>

Mogą one stanowić elementy pomocnicze w procesie przetwarzania lub mogą być dodane do danych wyjściowych.

Przykład arkusza stylów dla pliku **contents.xml.**
Należy znaleźć w tym dokumencie omawiane wyżej konstrukcje **XPath** i **XSL**.

Dokument contents.xsl

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

 <xsl:template match="JavaXML:Book">

  <html>

   <head>
    <title><xsl:value-of select="JavaXML:Title" /></title>
   </head>

   <body>
    <xsl:apply-templates select="*[not(self::JavaXML:Title)]" />
   </body>

  </html>

 </xsl:template>

 <xsl:template match="JavaXML:Contents">

  <center>
   <h2>Table of Contents</h2>
  </center>

  <hr />

  <ul>

   <xsl:for-each select="JavaXML:Chapter">

    <xsl:choose>

     <xsl:when test="@focus='Java'">

      <li><xsl:value-of select="JavaXML:Heading" /> (Java Focus)</li>

     </xsl:when>

     <xsl:otherwise>

      <li><xsl:value-of select="JavaXML:Heading" /> (XML Focus)</li>
```

```
    </xsl:otherwise>
   </xsl:choose>
  </xsl:for-each>
 </ul>
</xsl:template>
<xsl:template match="JavaXML:References">
 <p>
  <center><h3>Useful References</h3></center>
  <ol>
   <xsl:for-each select="JavaXML:Reference">
    <li>
     <xsl:element name="a">
      <xsl:attribute name="href">
       <xsl:value-of select="JavaXML:Url" />
      </xsl:attribute>
      <xsl:value-of select="JavaXML:Name" />
     </xsl:element>
    </li>
   </xsl:for-each>
  </ol>
 </p>
</xsl:template>
<xsl:template match="JavaXML:Copyright">
  <xsl:copy-of select="*" />
</xsl:template>
```

```
</xsl:stylesheet>
```

źródło: http://www.oreilly.com/catalog/javaxml

4.10. Zaawansowane zastosowania dokumentów XML

4.10.1. Struktura publikacji WWW(publishing framework)

Termin ten nie ma formalnej definicji, ale jest w większości przypadków jest interpretowany jako zestaw narzędzi XML wykonujących przetwarzanie, przekształcanie oraz inne operacje na dokumentach XML w ramach danej aplikacji. działającej zazwyczaj po stronie serwera obsługującego mechanizm serwletów. Struktura publikacji odpowiada na żądanie pobrania publikowanej wersji pliku (published file) np. w formacie HTML lub PDF. Plik publikowany powstaje dynamicznie z pliku XML na skutek zastosowania transformacji XSLT lub jest wynikiem przekształcenia pliku z innego formatu.

Do najbardziej znanych struktur publikacji należą :

Apache Cocoon: http://xml.apache.org

Ehydra Application Server: http://www.enhydra.org

Bluestone XML Server: http://www.bluestone.com/xml

SAXON: http://users.iclway.co.uk/mhkay/saxon

Szczególną pozycję ze względu na stabilność i integrację z narzędziamy XML zajmuje struktura Apache Cocoon.

Domyślnie obsługuje ona Apache Xerces i Apache Xalan, pozwala na wykorzystanie dowolnego parsera XML. Wykorzystuje również strukturę serwletów Javy.

Po pobraniu i zainstalowaniu Apache Cocoon trzeba zainstalować serwer obsługujący mechanizm serwletów (np. Jakarta Tomcat)a następnie skonfigurować ten mechanizm podając ścieżki dostępu do bibliotek Cocoon oraz do pliku właściwości Coccoon.

Generalnie instalacja struktury aplikacji jest dość złożona więc warto posłużyć się pomocą online znajdującą się pod adresami:

http://xml.apache.org, http://xml.apache.org./cocoon/faqs.html.

Po zainstalowaniu i skojarzeniu struktury publikacji z serwerem korzystanie ze struktury publikacji polega na użyciu przeglądarki WWW i wpisywaniu adresów URL odpowiednich plików XML.

Interesującą cechą Apache Cocoon jest możliwość wykorzystania technologii łączności bezprzewodowej. Odpowiednie wpisy w pliku właściwości Cocoon pozwalają wykryć klienta bezprzewodowego (telefon komórkowy z dostępem do Internetu) i wysłanie odpowiedzi odpowiedniej dla urządzenia WAP (umieszczonej w znacznikach **<wml>...</wml>**).

4.10.2. Rozszerzalne strony serwera XSP(Extensible Server Pages)

Rozszerzalne strony XSP powstały na gruncie rozwijania struktury publikacji

Strony XSP to dokumenty w formacie XML, które stanowią rozwiązanie problemów JSP, które nie zostały do końca rozwiązane, a mianowicie rozdzieleniem zawartości i prezentacji oraz zmianą formatu JSP oraz użycia JSP do komunikacji między aplikacjami.

Na stronach XSP można używać także logiki biznesowej wykorzystując komponenty Javy dostępne po stronie serwera (np. Enterprise Java Beans).

Generalnie strony XSP są bardziej elastyczne i uniwersalne niż strony JSP.

Przykład prostej strony XSP wyliczającej ile razy została ona pobrana :

```
<?xml version="1.0"?>

<?cocoon-process type="xsp"?>

<?cocoon-process type="xslt"?>

<?xml-stylesheet href="myStylesheet.xsl" type="text/xsl"?>

<xsp:page language="java" xmlns:xsp="http://www.apache.org/1999/XSP/Core">

 <xsp:logic>

  private static int numHits = 0;

    private synchronized int getNumHits() {

        return ++numHits;
```

```
  }
</xsp:logic>
<page>
 <title>Hit Counter</title>
 <p>I've been requested <xsp:expr>NumHits()</xsp:expr> times.</p>
</page>
</xsp:page>
```

źródło: http://www.oreilly.com/catalog/javaxml

W znaczniku **<xsp:logic>** zawarta jest logika aplikacji-obliczanie ile razy dana strona została pobrana.

Ponieważ jest to dokument XML i jako taki może zostać poddany przetworzeniu w modelu SAX lub DOM, jak również poddany transformacji XSLT zgodnie z określonym arkuszem stylów XSL i w ten sposób oddzieleniu zawartości od prezentacji.

Jeden programista może zatem generować treść statycznie albo dynamicznie przy użyciu serwletu lub innej aplikacji Javy, drugi zaś może zmieniać sposób prezentacji poprzez modyfikację arkusza stylów XSL.

4.10.3. XML-RPC (XML-Remote Procedure Call)

W technologii RPC serwer definiuje usługę jako zestaw procedur, które klient może wywoływać chcąc uzyskać dostęp do tej usługi.

Za pomocą tej technologii możliwe jest wywoływanie procedur przez sieć i otrzymywania odpowiedzi też przez sieć bez bezpośredniego komunikowania się z obiektem zdalnym oferującym usługę.

W Javie ta technologia przestała być używana od momentu powstania technologii RMI, w której klient uzyskując referencję do zdalnego obiektu zarejestrowanego i wyeksportowanego po stronie serwera, wywołuje jego metody i otrzymuje wyniki tych wywołań. Klient RMI uzyskuje komunikację za pośrednictwem namiastek (po stronie klienta) i szkieletów (po stronie serwera) ładowanych przez sieć.

Największy problem w technologii RPC związany był z kodowaniem i odkodowywaniem przesyłanych danych o złożonej strukturze - pojawienie się

standardu XML zmieniło radykalnie tę sytuację. Technologia XML pozwoliła przywrócić znaczenie technologii RPC dzięki reprezentowaniu dowolnych danych oraz ich struktury za pomocą dokumentów tekstowych w standardzie XML.

W konsekwencji wysyłanie i odbieranie danych tekstowych za pomocą mechanizmu XML-RPC w pewnych sytuacjach jest wydajniejsze niż technologia RMI.

Informacje o XML-RPC uzyskać można na stronie **http://www.xml-rpc.com ,** natomiast pakiet API Javy dla XML-RPC można uzyskać ze strony **http://helma.at/hannes/xmlrpc.**

Pakiet ten o nazwie **helma.xmlrpc** zawiera m.inn.klasy XmlRpc, XmlRpcClient, XmlRpcServer, XmlRpcHandler służące do tworzenia klienta i serwera XML-RPC oraz do kodowania i przetwarzania przesyłanych danych . Klasa WebSerwer opisuje prosty serwer HTTP do obsługi żądań klienta XML-RPC.

W komunikacji XML-RPC kluczową rolę odgrywają dwie procedury obsługi:

- procedura obsługi żądania

- procedura obsługi odpowiedzi

Procedura obsługi żądania zawarta jest w bibliotekach XML-RPC w klasie helma.xmlrpc.XmlRpcServer, zatem programista musi zdefiniować tylko procedurę obsługi odpowiedzi, która musi być zarejestrowana w serwerze.

W sygnaturze tej metody mogą wystąpić typy zmiennych obsługiwane przez XML-RPC.

Typy te i ich odpowiedniki w Javie zawiera poniższa tabela:

Typy XML-RPC	Typy Javy
int	int
boolean	boolean
string	java.lang.String
double	double
dateTime.iso8601	java.util.Date
struct	java.util.HashTable
array	java.util.Vector
base64	byte[]

Przy pisaniu aplikacji klient-serwer XML-RPC uwzględnić trzeba następujące etapy:

- zdefiniowanie klasy a w niej metody, która ma być uruchomiona zdalnie (klasa procedury obsługi odpowiedzi)

- ustalenie parsera SAX do przetwarzania i kodowania XML po stronie serwera

- utworzenie serwera XML-RPC

- zarejestrowanie klasy procedury obsługi

- ustalenie parsera SAX do przetwarzania i kodowania XML po stronie klienta

- utworzenie klienta

- wywołanie procedury obsługi

W ramach prostego przykładu rozpatrzmy aplikację typu klient - server XML-RPC, która wywołuje zdalnie metodę **sayHello()**. Metoda ta pobiera argument **text** typu String i zwraca łańcuch "Hello "+text.

Definicja klasy procedury obsługi odpowiedzi i metody wywoływanej zdalnie:

```
public class HelloHandler {

 public String sayHello(String text) {
   return "Hello"+text;
 }
}
```

źródło: http://www.oreilly.com/catalog/javaxml

Przykładowy serwer XML-RPC rejestrujący powyższą klasę z procedurą obsługi:

```
import java.io.IOException

import helma.xml.rpc.XmlRpc;
import helma.xmlrpc.WebSerwer;

class ServerHello {

  public static void main (String[] args){

    //numer poru serwera z linii komend
    if(args.length<1){
      System.out.println("Port number missing");
      System.exit(-1);
    }
    try {
```

```
      //ustalenie parsera Apache Xerces SAX
      XmlRpc.setDriver()("org.apache.xerces.parsers.SAXParser")

      //utworzenie serwera HTTP
      WebServer server = new WebServer(Integer.parseInt(args[0]);

      //rejestracja klasy procedury obsługi,której nadano nazwę "hello"
      server.addHandler("hello",new HelloHandler());
    }
    catch(ClassNotFoundException e) {
      System.out.println("Nie odnaleziono klasy parsera SAX");
    }
    catch (IOException e) {
      System.out.println(e.getMessage());
    }

  } //main()

} //class SerwerHello
```

źródło: http://www.oreilly.com/catalog/javaxml

Przykład klasy klienta wywołującego zdalnie zarejestrowaną metodę :

```
//import bibliotek
import java.io.IOException;
import java.net.MalformedURLException;
import java.util.Vector;

import helma.xmlrpc.XmlRpc;
import helma.xmlrpc.XmlRpcClient;
import helma.xmlrpc.XmlRpcException;

public class ClientHello {

    public static void main(String args[]) {

    //tekst do wysłania z linii komend
    if (args.length < 1) {
      System.out.println("Text missing");
      System.exit(-1);
    }

    try {

      // ustalenie parsera SAX:Apache Xerces
      XmlRpc.setDriver("org.apache.xerces.parsers.SAXParser");

      // utworzenie klienta;serwer na porcie lokalnym
      XmlRpcClient client =
new XmlRpcClient("http://localhost:8080/");

      // utworzenie żądania
      Vector params = new Vector();
      params.addElement(args[0]);

      // wysłanie żądania
```

```
      String result
=(String)client.execute("hello.sayHello",params);

      //wydrukowanie odpowiedzi
      System.out.println("Response from server:" + result); }

    catch (ClassNotFoundException e) {

      System.out.println("Could not locate SAX Driver");

    }
    catch (MalformedURLException e) {

      System.out.println("Incorrect URL for XML-RPC server format: " +
e.getMessage());

    }

    catch (XmlRpcException e) {
      System.out.println("XML-RPC Exception:" + e.getMessage());
    }

    catch (IOException e) {
      System.out.println("IO Exception: " + e.getMessage());
    }

  } //main()

} //class ClientHello
```

źródło: http://www.oreilly.com/catalog/javaxml

W powyższej aplikacji nie widać jawnego użycia formatu XML-odbywa sie to niejawnie. Wysłane żądanie zostało przetłumaczone na wywołanie HTTP w którym dane związane z metodą zdalną występują w formacie XML , a wynik wykonania metody kodowany jest też do formatu XML. Żądanie i odpowiedź prezentuje poniższa tabela:

Żądanie	**Odpowiedź**
POST /RPC2 HTTP/1.1 User-Agent:Tomcat Web Server/3.1 Beta (Sun Solaris 2.6) Host: new Instance.com Content-Type: text/xml Content-length: 234 <?xml version="1.0"?>	HTTP/1.1 200 OK Connection: close Content-Type: text/xml Content-length: 149 <?xml version="1.0"?> <methodResponse>

<methodCall> <methodName>hello.sayHello</methodName> <params> <param> <value><string>XML</string></value> </param> </params> </methodCall>	<params> <param> <value><string>Hello XML</string></value> </param> </params> </methodResponse>

źródło: http://www.oreilly.com/catalog/javaxml

4.10.4. XML w plikach konfiguracji serwerów

Standard XML ma również zastosowanie w serwerach aplikacji.

Przykładem może być **Enterprise JavaBean**, której specyfikacja wymaga żeby pliki deskryptorów wdrożeń oparte były o XML

Innym przykładem jest wykorzystanie XML w konfiguracji całego mechanizmu serwletów jak również w konfiguracji indywidualnych serwletów.

Jako prosty przykład wykorzystania XML w plikach konfiguracyjnych podamy plik konfiguracyjny dla klienta i serwera dla omówionej powyżej aplikacji klient serwer XML-RPC.

W pliku takim powinny być zawarte informacje dotyczące:

- **serwera:** port serwera, klasa parsera od strony serwera, procedury obsługi (identyfikator klasy i nazwa klasy)
- **klienta**: nazwa hosta na którym zainstalowano serwer, port serwera, klasa parsera od strony klienta

Dodatkowo plik konfiguracyjny powinien zostać zawężony zgodnie z definicją DTD, tak aby był rozumiany przez każdy serwer.

Przykład pliku konfiguracyjnego i jego zawężenia:

Plik konfiguracyjny klienta i serwera	**Plik DTD do którego odwołuje się plik konfiguracyjny**
<?xml version="1.0"?> <!DOCTYPE JavaXML:xmlrpc-config SYSTEM "DTD/XmlRpc.dtd"> <JavaXML:xmlrpc-config xmlns:JavaXML= "http://www.oreilly.com/catalog/javaxml/" > <!-- Configuration Information for Server and Clients --> <JavaXML:hostname>localhost</JavaXML:hostname> <JavaXML:port type="unprotected">8585</JavaXML:port> <JavaXML:parserClass> org.apache.xerces.parsers.SAXParser </JavaXML:parserClass> <!-- Server Specific Configuration Information --> <JavaXML:xmlrpc-server> <!-- List of XML-RPC handlers to register --> <JavaXML:handlers> <JavaXML:handler> <JavaXML:identifier>hello</JavaXML:identifier> <JavaXML:class>HelloHandler</JavaXML:class> </JavaXML:handler> </JavaXML:handlers> </JavaXML:xmlrpc-server> </JavaXML:xmlrpc-config>	<!ELEMENT JavaXML:xmlrpc-config (JavaXML:hostname, JavaXML:port, JavaXML:parserClass, JavaXML:xmlrpc-server)> <!ATTLIST JavaXML:xmlrpc-config xmlns:JavaXML CDATA #REQUIRED > <!ELEMENT JavaXML:hostname (#PCDATA)> <!ELEMENT JavaXML:port (#PCDATA)> <!ATTLIST JavaXML:port type (protected|unprotected) "unprotected" > <!ELEMENT JavaXML:parserClass (#PCDATA)> <!ELEMENT JavaXML:xmlrpc-server (JavaXML:handlers)> <!ELEMENT JavaXML:handlers (JavaXML:handler)+> <!ELEMENT JavaXML:handler (JavaXML:identifier, JavaXML:class)> <!ELEMENT JavaXML:identifier (#PCDATA)> <!ELEMENT JavaXML:class (#PCDATA)>

źródło: http://www.oreilly.com/catalog/javaxml

4.11. Rozszerzenia Javy stosowane w technologii XML

4.11.1. JAXB (Java Architecture for XML/Java Binding)

Narzędzia do automatycznego mapowania dokumentów XML na obiekty Javy. JAXB kompiluje DTD lub XML Schema do jednej lub kilku klas Javy, które obsługują wszystkie szczegóły parsowania i formatowania dokumentu XML. w wielu przypadkach wygenerowane klasy są bardziej efektywne niż parsery SAX lub DOM.

4.11.2. JAXM (Java API for XML Messaging)

Pakiet JAXM umożliwia wysyłanie i odbieranie wiadomości w formacie XML przy użyciu API Javy. JAXM implementuje protokół SOAP 1.1(Simple Object Access Protocol) z attachmentami, dzięki czemu użytkownik może koncentrować się na wysyłaniu, odbieraniu i dekompozycji wiadomości w aplikacji zamiast programowania komunikacji XML na niskim poziomie.

4.11.3. JAX- RPC (Java API for XML RPC)

Umożliwia budowanie aplikacji sieciowych i usług sieciowych przy zastosowaniu zdalnego wywoływania procedur (Remote Procedure Call) bazującego na XML.

4.11.4. JAXR (Java API for XML Registries)

Dostarcza standardowego Java API uzyskiwania dostępu do różnych rejestrów XML.
Rejestr XML stanowi infrastrukturę do tworzenia, rozwijania i znajdywania usług sieciowych.
JAXR współdziała z innymi technologiami Javy t.j. JAXP, JAXB, JAXM, JAX-RPC w ramach J2EE (Java 2 Enterprise Edition).

Obecnie w Internecie (http://www.java.sun.com/xml) dostępny jest pakiet Java WSDP (Java Web Services Developer Pack1.2)- zestaw narzędzi do tworzenia, testowania i wdrażania aplikacji XML włączając w to interfejsy API do przetwarzania i przekształcania XML oraz narzędzia do JAXB.

4.12. Ćwiczenia i zadania

Zadanie-1:
Napisz własny dokument XML a następnie dokonaj jego przetwarzania (parsingu) w modelu **SAX, DOM, JDOM.**

Wymyśl sposób testowania i porównania działania programów dokonujących analizy zgodnie z każdym z 3 modeli.
Weź pod uwagę takie parametry jak zużycie pamięci i szybkość analizy.

Zadanie-2:
Dokonaj zawężenia dokumentu z zadania-1 zgodnie z definicją DTD i dokonaj analizy tak zawężonego dokumentu w każdym z trzech modeli.

Zadanie-3:
Dokonaj zawężenia dokumentu z zadania-1 według XML Schema i dokonaj analizy tak zawężonego dokumentu w modelu SAX i DOM stosując biblioteki JAXP1.2.

Zadanie-4:
Napisz aplikację sieciową firma-firma w której dane przesyłane są między firmami w formacie XML.

Firma pierwsza to fabryka samochodów osobowych i dostawczych.
Firma druga sieć to dealerów samochodowych.

Fabryka informuje wszystkich zarejestrowanych dealerów o nowych modelach samochodów dostępnych aktualnie.

Dealerzy uzyskaną informację przekształcają w ten sposób żeby móc zaprezentować ją klientom na swojej stronie www.

Na stronie tej umieszczony jest formularz pozwalający przyjmować zamówienia od klientów na dany model samochodu.

Obsługą formularza powinien zajmować się serwlet działający po stronie dealera.

Wśród dealerów mamy albo dealerów samochodów osobowych albo dostawczych, zatem każdy z nich musi przefiltrować informację z fabryki i wydzielić tylko tę która go interesuje.

Każdy z dealerów informuje fabrykę o ilości sprzedanych samochodów danego modelu. Przy każdym modelu dealer podaje informację ile takich modeli ma fabryka jeszcze przysłać.

Protokołem komunikacyjnym ma być protokół HTTP.

4.13 Literatura, źródła

- **Brett McLaughlin "Java and XML" O'Reilly & Associates 2000; http://www.oreilly.com/catalog/javaxml**

- **Thierry Violleau "Java Technology and XML-Part I" 2001; http://www.java.sun.com**

- **Java 2 SDK Standard Edition Documentation (version 1.4.1); http://www.java.sun.com**