LINKgroup

Distance Learning System

Java XML i web servisi

DOM

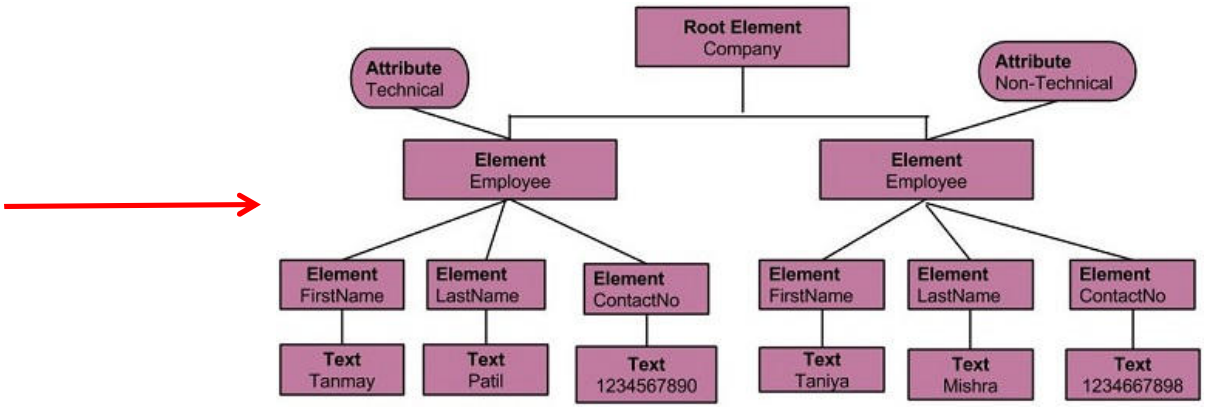
Uvod u DOM

- The Document Object Model (DOM) je API za HTML i XML dokumente. On definiše logičku strukturu dokumenta i način na koji se dokumentu pristupa i kojim se njime manipuliše.
- Iako se DOM može koristiti za strukturiranje bilo koje vrste dokumenata, mi ćemo se u kursu fokusirati na XML DOM.
- XML DOM je standardni objektni model XML-a. XML dokumenti imaju hijerarhiju koja se zasniva na elementima koji se nazivaju čvorovi (nodes). XML DOM predstavlja način za opis tih čvorova i odnosa među njima. XML DOM, omogućava dodavanje, izmenu, premeštanje i uklanjanje čvorova sa bilo koje tačke stabla.
- Ovakav pristup donosi brojne prednosti. Informacije u XML DOM-u su organizovane hijerarhijski, što znatno olakšava navigaciju i pronalazak specifične informacije. Takođe, XML DOM je dinamične prirode, što znači da je veoma lako izmenljiv.

DOM struktura

- DOM dokument je kolekcija čvorova ili drugim rečima informacija organizovanih hijerarhijski. Neki čvorovi mogu imati i decu čvorove različitog tipa, dok drugi čvorovi mogu biti takozvani listovi, što znači da nemaju potomke i da su na dnu hijerarhijske lestvice.

```
<?xml version="1.0"?>
<Company>
    <Employee category="technical">
        <FirstName>Tanmay</FirstName>
        <LastName>Patil</LastName>
        <ContactNo>1234567890</ContactNo>
    </Employee>
    <Employee category="non-technical">
        <FirstName>Taniya</FirstName>
        <LastName>Mishra</LastName>
        <ContactNo>1234667898</ContactNo>
    </Employee>
</Company>
```



Preuzimanje XML dokumenta korišćenjem Jave

- DOM je samo specifikacija, a na konkretnim tehnologijama, tj. jezicima je da pruže implementaciju ove specifikacije
- Procesom koji se naziva parsiranje od jednog XML dokumenta se kreira malopre opisana DOM struktura stabla. Nakon što DOM parser kreira stablo dokumenta može se pristupiti informacijama smeštenim u čvorovima stabla.
- Nakon parsiranja, sve komponente stabla čvorova se pretvaraju u objekte u memoriji i to tako da se stablo (koren) dokumenta u DOM-u obično označava kao **document**. To je tačka od koje sve počinje.
- Ovakav pristup je često sporiji nego prilikom sekvencijalnog parsiranja XML-a, ali je često pristupačniji, jer omogućava veću kontrolu nad samim dokumentom i jednostavniju navigaciju kroz dokument.
- Da bi se postiglo upravo opisano ponašanje, Java sadrži funkcionalnosti definisane u javax.xml.parsers paketu i njegovim klasama **DocumentBuilder** i **DocumentBuilderFactory**.

Kreiranje document builder-a

- Prvi korak predstavlja instanciranje DocumentBuilderFactory klase i to pozivanjem metode newInstance()
- Nakon dobijanja instance DocumentBuilderFactory klase, moguće je izvršiti pozivanje raznih konfiguracionih metoda.
- Nakon DocumentBuilderFactory klase, potrebno je doći i do objekta DocumentBuilder, koji se koristi za kreiranje konkretnog objekta dokumenta.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

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

Parsiranje dokumenta

- Metoda parse kao svoju povratnu vrednost ima objekat tipa Document. Ovaj objekat predstavlja kompletan parsirani dokument. Sledeća linija prikazuje pozivanje metode parse i kreiranje instance Document tipa.

```
Document doc = db.parse("country.xml");
```

- Moguće je parsirati i XML iz nekog stringa. Parsiranje XML iz stringa malo se razlikuje od parsiranja XML-a iz fajla, jer DocumentBuilder nije u stanju da parsira XML direktno iz Stringa. To je moguće uraditi putem klase InputSource, kojoj se prosleđuje instanca klase StringReader kao parametar.

```
        String xmlString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><books><
book id=\"01\" isbn=\"12345\"><title>Childhood's End</title><author>Arthur Cla
rke</author></book></books>";
        InputSource is = new InputSource( new StringReader( xmlString ) );
        Document doc = db.parse(is);
```

DOM interfejsi

- Da bismo naučili kako da se krećemo kroz DOM strukturu, potrebno je da se prvo upoznamo sa nekim ključnim interfejsima.
- Kada kreiramo objekat tipa Document, on predstavlja kompletan XML dokument i uključuje sve elemente, podelemente, atribute komentare prostore imena i sl. Već smo se u poglavlju o strukturi stabla jednog XML dokumenta upoznali sa njegovim elementima: čvorovima, elementima, atributima, tekstom (sadržajem) i dokumentom.
- U Java programskom jeziku za svaki od ovih elemenata DOM stabla postoji po interfejs.

Node

- Node je primarni tip podatka za svaki element DOM strukture. On predstavlja jedan čvor dokumenta i može biti različitog tipa.
- Svi ostali interfejsi sa kojima ćemo se upoznati nasleđuju metode definisane od Node interfejsa. Metode ovog interfejsa prikazane su u sledećoj tabeli

Naziv metode	Opis
getAttributes()	vraća NamedNodeMap kolekciju atributa konkretnog elementa
getChildNodes()	vraća podelemente konkretnog elementa
getLocalName()	vraća lokalni naziv elementa
getNodeName()	vraća naziv čvora
getNodeValue()	vraća vrednost čvora
getNodeType()	vraća tip čvora

Document

- Document je interfejs koji definiše osnovne metode za navigaciju, tj. kretanje kroz XML dokument. Ove metode su prikazane u sledećoj tabeli:

Naziv metode	Opis
getDoctype()	vraća DOCTYPE XML dokumenta
getDocumentElement()	vraća root element
getElementById(String)	vraća element sa specifičnim ID-om
getElementsByTagName(String)	vraća kolekciju elemenata tipa NodeList

Element

- Element je interfejs koji predstavlja element DOM strukture. Element je definisan tagovima. Korišćenjem metoda Element interfejsa može se pristupiti podelementima i atributima elementa nad kojim se metode prozivaju. Neke od metoda ovog interfejsa prikazane su u narednoj tabeli

Naziv metode	Opis
getAttributes()	vraća NamedNodeMap kolekciju atributa
getAttribute(String)	vraća vrednost atributa po njegovom nazivu
getAttributeNode(String)	vraća pripadajući čvor atributa
getElementsByTagName(String)	vraća kolekciju podelemenata tipa NodeList
getTagName()	vraća naziv taga elementa

Attr

- **Attr** predstavlja atribut XML dokumenta. Kada se govori o atributu, moguće je dobiti njegov naziv i vrednost i to korišćenjem sledećih metoda:

Naziv metode	Opis
getName()	vraća naziv atributa
getValue()	vraća vrednost atributa

Tipovi čvorova

	Skraćena vrednost	Tip čvora
1	ELEMENT_NODE	Element node
2	ATTRIBUTE_NODE	Attr node
3	TEXT_NODE	Text node
4	CDATA_SECTION_NODE	CDATASection node
5	ENTITY_REFERENCE_NODE	EntityReference node
6	ENTITY_NODE	Entity node
7	PROCESSING_INSTRUCTION_NODE	ProcessingInstruction node
8	COMMENT_NODE	Comment node
9	DOCUMENT_NODE	Document node
10	DOCUMENT_TYPE_NODE	DocumentType node
11	DOCUMENT_FRAGMENT_NODE	DocumentFragment node
12	NOTATION_NODE	Notation node

Navigacija i čitanje XML dokumenta

- Da bismo počeli sa obradom dokumenta, potrebna je neka polazna tačka. Ona je obično sam koreni element dokumenta i dobija se metodom getDocumentElement, interfejsa Document

http://pastie.org/pastes/10046019/text

```xml
<books>
    <book id="01" isbn="12345">
        <title>Childhood's End</title>
        <author>Arthur Clarke</author>
    </book>
    <book id="02" isbn="67890">
        <title>The Merchant of Venice</title>
        <author>William Shakespeare</author>
    </book>
</books>
```

```
Element root = doc.getDocumentElement();

System.out.println(root.getNodeName());
System.out.println(root.getNodeType());
```

Navigacija i čitanje XML dokumenta

- Navigaciju kroz elemente vršimo uz pomoć dve tehnike. Direktnim „ciljanjem“ određenih elemenata i iteracijom kroz liste srodnih elemenata.

getElementById
(jwsx022014 DomGetElementById)

- U tabelama sa pregledom metoda koje su date nešto ranije u ovoj lekciji verovatno ste primetili metodu interfejsa Document **getElementById**. Naime, ova metoda predstavlja veoma dobar način za direktan pristup elementima, ali zahteva određenu intervenciju na već postojećoj XML strukturi (tripizacija id atributa)
- Ipak nije dovoljno postajanje ovih atributa na elementima. Potrebno je id definisati i unutar nekog od dokumenata za specificiranje validacionih pravila. DTD dokument sa definisanim ovim atributom bi izgledao ovako:

```
<!DOCTYPE bookschema
[
<!ELEMENT books (book)*>
<!ELEMENT book (title|author)*>
<!ATTLIST book
    id ID #IMPLIED
    isbn CDATA #IMPLIED
  >
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
]>
```

```
Element book = doc.getElementById("02");
System.out.println(book.getTextContent());
```

getElementsByTagName (jwsx022014 DomGetElementById)

- Drugi metod koji možemo upotrebiti za direktno ciljanje jeste metod getElementsByTagName. Ovaj metod nije precizan kao getElementById jer vraća kolekciju elemenata, ali je, za razliku od prethodnog, upotrebljiv na samom elementu. Obično se koristi tako što se uz pomoć njega preuzme određena kolekcija elemenata, a nakon toga izvrši filtracija:

```
Element r = doc.getDocumentElement();
NodeList books = r.getElementsByTagName("book");
for(int i=0;i<books.getLength();i++)
    {
        Node book = books.item(i);
        System.out.println(book.getTextContent());
    }
```

Čitanje atributa
(jwsx022014 DomReadAttributes)

- Kada pristupimo nekom elementu, možemo pristupiti i njegovoj listi atributa, metodom **getAttributes**. Ovaj metod takođe vraća kolekciju nodova, pa je rukovanje njime veoma slično rukovanju metodima za navigaciju kroz elemente. Sledeći primer pronalazi i ispisuje kompletne podatke za svaku knjigu:

```
Element r = doc.getDocumentElement();
NodeList books = r.getElementsByTagName("book");
for(int i=0;i<books.getLength();i++)
{
    System.out.println("Book name: " + books.item(i).getFirstChild().getTextContent());
    System.out.println("Author: " + books.item(i).getLastChild().getTextContent());
    System.out.println("ID: " + books.item(i).getAttributes().item(0).getNodeValue());
    System.out.println("ISBN: " + books.item(i).getAttributes().item(1).getNodeValue());
}
```

Upis u XML dokumente
(jwsx022014 DomWrite)

- Za razliku od SAX-a, DOM ima mogućnost pisanja u XML dokumentu, pri čemu postoji različit set metoda za izvršavanje ovih zadataka. Za dodavanje elementa koristi se metod **createElement**. Ovaj metod mora se izvršiti na nivou dokumenta, odnosno, na Document objektu. To pravilo važi i za sve podelemente kreiranog elementa

```
Element r = doc.getDocumentElement();
Element book = doc.createElement("book");
Element title = doc.createElement("title");
Element author = doc.createElement("author");
```

- Ova tri elementa, iako su kreirana, nisu dodata dokumentu, tako da bi i nakon njihovog izvršavanja struktura XML dokumenta ostala neizmenjena. Da bi elementi ušli u strukturu dokumenta, treba ih ručno dodati, metodom appendChild, roditeljskog elementa. S obzirom na to da je books koreni element, a da je roditelj elemenata title i author element book, dodavanje će izgledati ovako:

```
book.appendChild(title);
book.appendChild(author);
r.appendChild(book);

book.setAttribute("id", "03");
book.setAttribute("isbn", "111213");
title.setTextContent("The Caves of Steel");
author.setTextContent("Isaac Asimov");
```

Emitovanje XML dokumenta na izlaz

- Da bismo emitovali XML dokument u tok, prvo treba kreirati instancu **DOMSource** klase. Zatim, potrebno je kreirati instancu **Transformer** klase, uz pomoć **TransformerFactory** klase. Konačno, aktivacijom metoda **transform**, klase Transformer, emitujemo XML dokument na izlaz. Kao parametre, ovaj metod prihvata izlaz (u našem slučaju tok) i sam dokument (DOMSource objekat).

```
String outputURL = "books.xml";
DOMSource xmlDoc = new DOMSource(doc);
StreamResult result = new StreamResult(new FileOutputStream(outputURL));
TransformerFactory transFactory = TransformerFactory.newInstance();
Transformer transformer = transFactory.newTransformer();
transformer.transform(xmlDoc, result);
```

Vežba 1 (jwsx022014 ParseHtml)

http://pastie.org/pastes/10054404/text

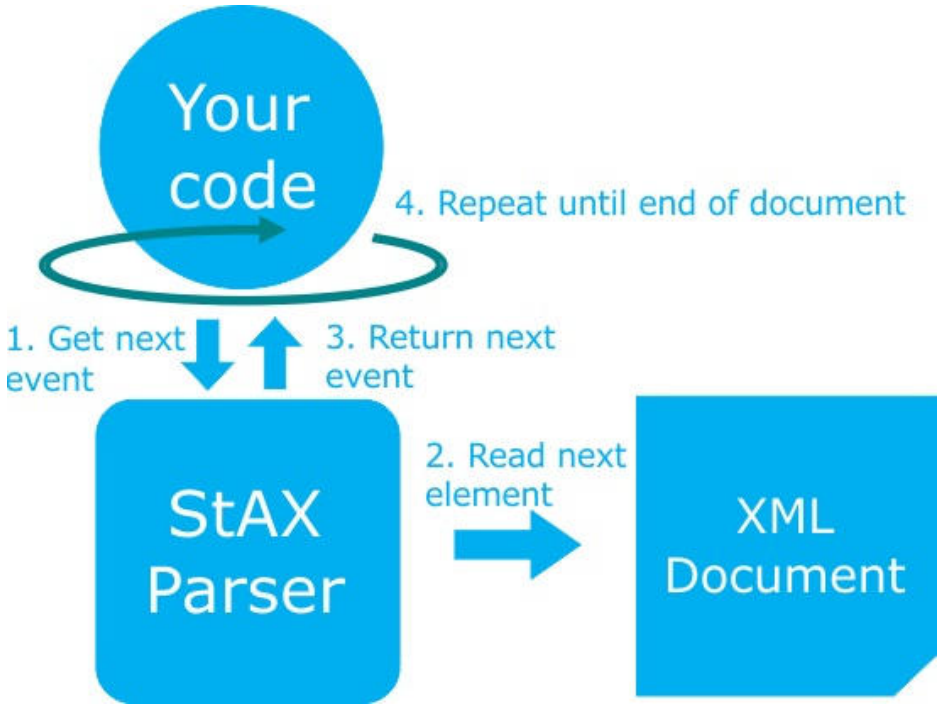
- Dat je sledeći HTML dokument pod nazivom page.html:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
</head>
<body>
<p><a href="http://www.java.com">Hello</a></p>
<p><a href="http://www.oracle.com/us/sun">From</a></p>
<p><a href="http://www.java.net">Java</a></p>
</body>
</html>
```

- Potrebno je parsirati dokument tako da se preuzmu samo linkovi iz dokumenta. Zatim treba preuzeti sadržaj sa tih adresa i snimiti ga u odgovarajuće fajlove (svaki fajl treba da ima naziv adrese (bez specijalnih karaktera).

StAX

- Streaming API for XML (StAX) je API za parsiranje XML dokumenta, sekvencijalno, od početka do kraja.
- Može se s pravom postaviti pitanje koja je opravdanost, tj. svrha postojanja još jednog API-ja za parsiranje XML, pored već dva postojeća sa kojima smo se upoznali u prethodnim lekcijama. Jednostavno rečeno, StAX kombinuje najbolje od oba pristupa i na taj način prevazilazi nedostatke prethodno opisanih tehnika parsiranja.
- StAX baš kao i SAX poseduje prednost pri parsiranju velikih XML dokumenata. DOM je ograničen na veličinu interne memorije uređaja koji vrši parsiranje, pa kod uređaja sa manjom količinom radne memorije, parsiranje velikih XML dokumenata može predstavljati problem.
  - SAX se ne može koristiti za kreiranje XML dokumenata i izmenu strukture već postojećih.
  - StAX se može koristiti za kreiranje XML dokumenata.
- StAX takođe preuzima pozitivne osobine SAX-a kada je reč o brzini, tako da je pristup elementima znatno brži i nije potrebno čekati da se dokument potpuno smesti u memoriju, kao što je to slučaj sa DOM parsiranjem.
- StAX koristi pull model, po kojem je aplikacija ta koja kontroliše parsiranje, tj. govori parseru kada je spremna da primi određeni element



Funkcionisanje StAX-a

- Java implementira StAX kroz tipove koji se nalaze u **javax.xml.stream**, **javax.xml.stream.events** i **javax.xml.stream.util** paketima.
- StAX API poseduje dva različita API-ja za parsiranje i kreiranje XML dokumenata, jedan zasnovan na kursoru, a drugi na iteratoru.
- Kursor može da pokazuje samo na jedan element u jednom trenutku i uvek se kreće samo unapred. Ovakav pristup se preporučuje ukoliko postoji potreba za uštedom radne memorije. Interfejs XMLStreamReader se koristi za parsiranje dokumenta korišćenjem kursora.
- Parsiranje iteratorom funkcioniše veoma slično standardnom korišćenju iteratora i za to se koristi interfejs XMLEventReader. Za razliku od parsiranja uz pomoć kursora, moguće je vratiti se na prethodne događaje.

Parsiranje XML dokumenata korišćenjem kursora (jwsx022014 STAXCursorRead)

- XMLStreamReader interfejs predstavlja najefikasniji način za čitanje XML dokumenta korišćenjem StAX-a. Ovaj interfejs poseduje metodu **hasNext()**, koja vraća logičku vrednost true ili false, u zavisnosti od toga da li postoji sledeći element za čitanje. Metoda **next()** se koristi za pomeranje kursora unapred, a pored toga ova metoda vraća celobrojnu vrednost kojom je moguće utvrditi tip pročitanog elementa. Stoga će se čitanje elemenata obaviti unutar jedne while petlje, unutar koje će se kursor pomerati sa elementa na element i na taj način vršiti parsiranje. Tip elementa koji će metoda next vratiti služiće za definisanje specifičnih funkcionalnosti, na osnovu tipa.

```
XMLInputFactory xmlif = XMLInputFactory.newFactory();
Reader reader = new FileReader("books.xml");
XMLStreamReader xmlsr = xmlif.createXMLStreamReader(reader);
while (xmlsr.hasNext()) {
    switch (xmlsr.next()) {
        case XMLStreamReader.START_ELEMENT:
            System.out.println("START_ELEMENT");
            System.out.println(" Qname = " + xmlsr.getName());
        case XMLStreamReader.END_ELEMENT:
            System.out.println("END_ELEMENT");
            System.out.println(" Qname = " + xmlsr.getName());
    }
}
```

Pregled i vrednosti stream konstanti

Za tip proizvedenog događaja se koristi neka od konstanti definisanih u interfejsu ***javax.xml.stream.XMLStreamConstants***. Pregled i vrednosti ovih konstanti su sledeće:

Naziv konstante	Opis	Celobrojna vrednost
START_DOCUMENT	Početak dokumenta	7
START_ELEMENT	Početak elementa	1
ATTRIBUTE	Atribut elementa	10
NAMESPACE	Deklaracija prostora imena	13
CHARACTERS	Tekst i prazni karakteri	4
COMMENT	Komentar	5
SPACE	Prazan prostor koji se ignoriše	6
PROCESSING_INSTRUCTION	Instrukcije za procesuiranje	3
DTD	DTD	11
ENTITY_REFERENCE	Referenca na entitet	9
CDATA	CDATA sekcija	12
END_ELEMENT	Kraj elementa	2
END_DOCUMENT	Kraj dokumenta	8
ENTITY_DECLARATION	Deklaracija entiteta	15
NOTATION_DECLARATION	Notacija deklaracije	14

Vežba 1
(jwsx022014 GpxTrackStaxParse)

- Potrebno je parsirati gpx fajl yosemite.gpx pomoću StAX kurzor parsera, tako da na izlaz programa bude sledeći:

```
########### LOCATION ###########
Latitude: 37.744100
Longitude: -119.573190
Name: Y001
########### LOCATION ###########
Latitude: 37.750760
Longitude: -119.536530
Name: Y002
########### LOCATION ###########
Latitude: 37.785760
Longitude: -119.288180
Name: Y003
```

Parsiranje XML dokumenata korišćenjem iteratora (jwsx022014 STAXIteratorRead)

- Drugi pristup parsiranju korišćenjem StAX-a, kao što smo rekli, zasnovan je na upotrebi iteratora. Objekat tipa **XMLEventReader** parsira XML dokument korišćenjem iteratora i pri tome generiše **XMLEvent** događaje. Da bi se došlo do objekta **XMLEventReader** tipa, potrebno je prvo kreirati **XMLInputFactory** objekat, korišćenjem metode newInstance(), baš kao i u prethodnom primeru.

```
XMLInputFactory xmlif = XMLInputFactory.newFactory();
Reader reader = new FileReader("books.xml");
XMLEventReader xmlEventReader = xmlif.createXMLEventReader(reader);
```

Parsiranje XML dokumenata korišćenjem iteratora

- Da bi se utvrdilo postajanje sledećeg događaja, koristi se metoda **hasNext** XMLEventReader interfejsa.
- XMLEvent objekat predstavlja događaj XML dokumenta. Naredni događaj se može dobiti korišćenjem metode**nextEvent**, objekta tipa XMLEventReader.
- Da bi se dobio tip događaja, koristi se metoda **getEventType**. Tipovi odgovaraju onima koji su prikazani u prethodnoj tabeli, kada je bilo reči o parsiranju korišćenjem kursora.

```
while (xmlEventReader.hasNext()) {
    XMLEvent event = xmlEventReader.nextEvent();
    switch (event.getEventType()) {
        case XMLStreamReader.START_ELEMENT:
            StartElement start_element = event.asStartElement();
            System.out.println("START_ELEMENT");
            System.out.println(" Qname = " + start_element.getName());
            Iterator i = start_element.getAttributes();
            while (i.hasNext()) {
                Attribute attr = (Attribute) i.next();
```

Upis u XML korišćenjem kursora (jwsx022014 STAXCursorWrite)

- SAX je dvosmerna tehnologija, koja omogućava kako čitanje, tako i upis u XML dokumente. Oba do sada opisana principa poseduju svoje ekvivalentne metode za upis u XML. Korišćenjem kursora, odnosno toka, u XML je moguće upisivati korišćenjem objekta **XMLStreamWriter**.
- Bez obzira na način koji se koristi, prvo je neophodno dobaviti instancu tipa XMLOutputFactory. **XMLOutputFactory** se koristi za kreiranje gore pomenutog objekta.
- Za dobijanje instance tipa **XMLStreamWriter** koristi se metoda **createXMLStreamWriter**.
- Za upis se koristi objekat tipa XMLStreamWriter, čije metode se pozivaju, kako bi se različiti elementi upisali u XML dokument.
- Za svaki od tipova sa kojima smo se već upoznali postoji i odgovarajuća metoda. Tako se za početak elementa poziva metoda **writeStartElement** kojoj se prosleđuje naziv elementa. Za završetak elementa poziva se metoda**writeEndElement**.
- Za upis atributa poziva se metoda **writeAttribute**, koja prima dva parametra, odnosno naziv i vrednost atributa.
- Sam sadržaj tagova, tj. tekst koji se nalazi između tagova, upisuje se metodom **writeCharacters**.

```
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLStreamWriter writer = factory.createXMLStreamWriter(new FileWriter("books.xml"));

writer.writeStartDocument();
writer.writeStartElement("books");
writer.writeStartElement("book");
writer.writeAttribute("id", "01");
writer.writeAttribute("isbn", "12345");

writer.writeStartElement("title");
writer.writeCharacters("Childhood's End");
writer.writeEndElement();

writer.writeStartElement("author");
writer.writeCharacters("Arthur Clarke");
writer.writeEndElement();

writer.writeEndElement();
writer.writeEndDocument();
```

Upis u XML korišćenjem iteratora (jwsx022014 STAXIteratorWrite)

- Drugi način za upis u XML fajl jeste korišćenjem **XMLEventWriter** tipa. Ovakav pristup zahteva kreiranje i jednog objekta više, i to objekta tipa **XMLEventFactory**. Da bismo postigli kreiranje dokumenta identičnog prethodnom, možemo napisati ovakav kod:

```
XMLOutputFactory factory = XMLOutputFactory.newInstance();
XMLEventWriter writer = factory.createXMLEventWriter(new FileWriter("books.xml"));
XMLEventFactory xmlef = XMLEventFactory.newFactory();

XMLEvent event = xmlef.createStartDocument();
writer.add(event);

event = xmlef.createStartElement("", "", "books");
writer.add(event);

event = xmlef.createStartElement("", "", "book");
writer.add(event);

event = xmlef.createAttribute("id", "01");
writer.add(event);
event = xmlef.createAttribute("isbn", "12345");
writer.add(event);
```

Vežba
(jwsx022014 SakilaExport)

- Potrebno je kreirati aplikaciju koja će u vremenskom intervalu (bilo kom) eksportovati tabelu **actor**, baze podataka **sakila** u xml dokument
- Za rešenje treba koristiti **StAX** API

XPath

- XPath je deklarativni, upitni jezik za rukovanje, odnosno selektovanje podataka iz XML dokumenata. Za razliku od sekvencijalnog pregleda sa kojim smo se upoznali, ovaj jezik omogućava pronalaženje ciljnih nodova uz pomoć upita. Takođe, ovaj jezik se ne zasniva na XML-u.
- XPath nije ograničen na određenu programsku strukturu jezika i možemo ga koristiti uz pomoć bilo koje od biblioteka koje rukuju XML-om.
- Xpath se veoma često koristi kako bi se pojednostavio pristup DOM stablu, ali i u sprezi sa transformacijama, najčešće za selektovanje elemenata ulaznih dokumenata.

Kako XPath funkcioniše?

- XPath funkcioniše veoma slično putanjama koje postoje na fajl sistemu. Da bismo izolovali neki nod uz pomoć XPatha, koristimo poznatu sintaksu:

  /books/book
- Rezultat XPath upita je obično jedan element ili niz elemenata i kada jednom do njega dođemo, možemo mu se obraćati na standardne načine.
- XPath vidi ceo dokument kao stablo čvorova (nodova) koje započinje korenim čvorom.
- XPath razlikuje sedam tipova čvorova:
  - element,
  - atribut,
  - text,
  - prostor imena,
  - instrukcija za procesuiranje,
  - komentar i
  - dokument.
- XPath ne prepoznaje CDATA sekcije i DOCTYPE deklaracije

Navigacija

Za navigaciju kroz dokument, u xpath-u se koriste naredbe slične naredbama za navigaciju kroz fajl sistem

Izraz	Opis
node_name	Ukoliko se unese direktno naziv čvora, selektuju se svi čvorovi sa tim imenom
/	označava da selektovanje počinje od korenog čvora
//	označava da selektovanje počinje od navedenog čvora
.	selektuje trenutni čvor
..	selektuje roditelja trenutnog čvora
@	selektuje atribut

Korišćenje Xpath API-ja

- JAXP XPath API poseduje paket javax.xml.xpath u kome su definisani različiti interfejsi za rad sa Xpathom iz Java programskog jezika

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("books.xml");
XPathFactory factory = XPathFactory.newInstance();
XPath xPath = factory.newXPath();
XPathExpression xPathExpression = xPath.compile("/books/book");
Object result = xPathExpression.evaluate(doc, XPathConstants.NODESET);
NodeList nodeList = (NodeList)result;
for (int i = 0; i < nodeList.getLength(); i++) {
    System.out.print("Content: " + nodeList.item(i).getTextContent());
}
```

Zadatak

http://jsfiddle.net/js56m/

- Dat je GPX xml dokument
- Potrebno je kreirati XPath upit koji preuzima sve latitude dokumenta, kao i XPath upit koji preuzima longitude dokumenta
- Potrebno je upite implementirati u Javi

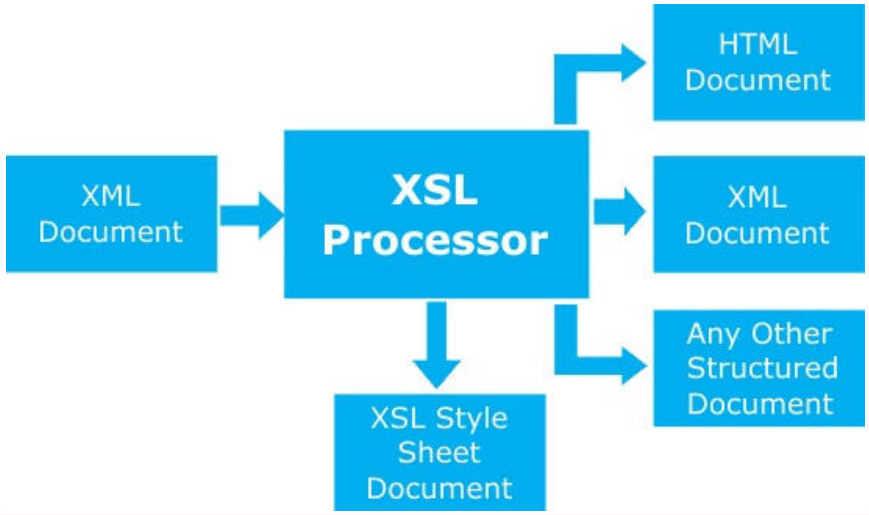
```
<?xml version="1.0" encoding="UTF-8"?>
<gpx>
 <metadata>
  <link href="http://www.garmin.com">
   <text>Garmin International</text>
  </link>
  <time>2009-10-17T22:58:43Z</time>
 </metadata>
 <trk>
  <name>Example GPX Document</name>
  <trkseg>
   <trkpt lat="47.644548" lon="-122.326897">
    <ele>4.46</ele>
    <time>2009-10-17T18:37:26Z</time>
   </trkpt>
   <trkpt lat="47.644548" lon="-122.326897">
    <ele>4.94</ele>
    <time>2009-10-17T18:37:31Z</time>
   </trkpt>
   <trkpt lat="47.644548" lon="-122.326897">
    <ele>6.87</ele>
    <time>2009-10-17T18:37:34Z</time>
   </trkpt>
  </trkseg>
 </trk>
</gpx>
```

EXtensible Stylesheet Language - XSLT

- XSL je zapravo jedan skup jezika za transformisanje, formatiranje i selektovanje XML dokumenata, koji sadrži sledeće:
  - XPath koji definiše konstrukte za adresiranje čvorova, tj. elemenata XML dokumenata,
  - XSL-FO (XSL Formatting Objects) – koristi se za definisanje semantičkih pravila i
  - XSLT – predstavlja jezik za transformisanje XML dokumenata.
- Na sličan način na koji se CSS koristi za stilizovanje HTML dokumenata, XSL se koristi za stilizovanje dokumenata sa XML kodom. Kod HTML značenje tagova je unapred poznato, tako da browser zna kako je potrebno da prikaže neki element. Sa druge strane, kod XML moguće je kreirati korisnički definisane tagove, tako da, podrazumevano, pretraživač neće znati kako da ih prikaže. Tu na scenu stupa XSL i pomenuti skup jezika za odabiranje, formatiranje i transformisanje.

XSL Transformacije

- Inicijalno, XSLT je razvijen kao jezik za transformisanje određenog XML dokumenta, korišćenjem semantičkih pravila formatiranja definisanih u XSL-FO vokabularu. Uprkos ovoj činjenici, XSLT je takođe osmišljen i kao jezik za transformisanje koji se može koristiti nezavisno od XSL-FO-a. Mi ćemo se u ovom kursu usredsrediti na XSLT koji se ne oslanja na XSL-FO.
- XSLT jezik je potpuno zasnovan na XML-u. Stoga, transformacije napisane korišćenjem XSLT-a postoje kao validno formatirani XML dokumenti. Ovakvi dokumenti se drugačije nazivaju stilovi (style sheets)



Primer transformacije (jwsx022014 XSLTSimpleTransformation)

http://pastie.org/10068002

- Recimo da hoćemo da sledeći xml dokument na izlazu bude reprezentovan na način prikazan desno:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <book id="01" isbn="12345">
        <title>Childhood's End</title>
        <author>Arthur Clarke</author>
    </book>
    <book id="02" isbn="67890">
        <title>The Merchant of Venice</title>
        <author>William Shakespeare</author>
    </book>
</books>
```

```xml
<table border="1">
<tr>
    <td>Id</td>
    <td>ISBN</td>
    <td>Title</td>
    <td>Author</td>
</tr>
<tr>
    <td>01</td>
    <td>12345</td>
    <td>Childhood's End</td>
    <td>Arthur Clarke</td>
</tr>
<tr>
    <td>02</td>
    <td>67890</td>
    <td>The Merchant of Venice</td>
    <td>William Shakespeare</td>
</tr>
</table>
```

Primer transformacije

http://pastie.org/10068002

- Transformacioni (stilski) dokument bi mogao izgledati ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>books.xsl</title>
            </head>
            <body>
                <table border="1">
                    <tr>
                        <td>Id</td>
                        <td>ISBN</td>
                        <td>Title</td>
                        <td>Author</td>
```

XSLT pravila

- **xsl:output**
- Element xsl:output je podelement elementa xsl:stylesheet i koristi se za definisanje osobina izlaza. Ovaj element može imati više atributa i njihovo značenje je sledeće:
- **method** atribut koristi se za definisanje tipa izlaznog dokumenta i neke od najčešće korišćenih vrednosti su xml, html ili text,
- **version** atribut se koristi za definisanje verzije XML deklaracije izlaznog dokumenta,
- atribut **omit-xml-declaration** se može postaviti na yes i u tom slučaju XML deklaracija se izostavlja iz izlaznog dokumenta i
- encoding atribut definiše enkoding izlaznog dokumenta

xsl:template

- XSL stilovi se sastoje od jednog ili više pravila koja se nazivaju šabloni (templates). Šabloni sadrže pravila koja je potrebno da se primene na određenom elementu.

- Element <xsl:template> se koristi za izgradnju šablona. Ovaj element poseduje match atribut koji se koristi da poveže šablon i odgovarajući XML element. S druge strane, ovaj match atribut se takođe koristi za definisanje šablona za čitav dokument. Vrednost ovog atributa je Xpath izraz. Na primer:

<xsl:template match="/">

- Vrednost atributa match u ovom slučaju se odnosni na kompletan dokument.

xsl:apply-templates

- Element xsl:apply-templates se može koristiti za selekciju većeg broja čvorova iz izvorišnog dokumenta. Pored instrukcije xsl:for-each, predstavlja jedan od glavnih konstrukata za dobavljanje seta čvorova.

xsl:for-each

- Za prolazak (iteriranje) kroz set čvorova, može se koristiti xsl:for-each element. Select atribut ovog elementa definiše Xpath izraz za selektovanje.

Uslovno izvršavanje

- XSLT specifikacija poseduje i dva elementa za izgradnju uslovnih blokova. To su xsl:if i xsl:choose elementi. Na primer:

```
<xsl:if test="$param1='param1'">
        <xsl:apply-templates/>
</xsl:if>
```

sl:value-of

- Ovaj element može da se koristi za selektovanje neke od vrednosti izvorišnog dokumenta i njegovo ugrađivanje u izlazni.

Vežba
(jwsx022014 XSLTEmployees)

Potrebno je transformisati kreirati xml dokument sa leve strane, tako da ima izgled slike desno

```
<?xml version="1.0"?>
<Company>
    <Employee category="technical">
        <FirstName>Tanmay</FirstName>
        <LastName>Patil</LastName>
        <ContactNo>1234567890</ContactNo>
    </Employee>
    <Employee category="non-technical">
        <FirstName>Taniya</FirstName>
        <LastName>Mishra</LastName>
        <ContactNo>1234667898</ContactNo>
    </Employee>
</Company>
```

Category	FirstName	Lastname	ContactNo
technical	Tanmay	Patil	1234567890
non-technical	Taniya	Mishra	1234667898