XML

Kasutusvaldkonnad, süntaks, töötlemine programmidega

Selle nime all tuntakse tekstifailivormingut. Esmamulje järgi paistavad elemendid olema kirjutatud nagu veebikoostajatele tuntud HTML-keeles ning XHTMLis koostatud veebilehed ongi XML-keele üks rakendusi. Kui HTMLis märgatav osa käsklusi tegelevad kujundusega ning käskluste arv on lõplik, siis XMLi puhul pole käskluste arv esimeses lähenduses piiratud ning kasutuskohana nähakse ka paljusid andmete hoidmise ning ülekandega seotud valdkondi. Tähtsamaks peetakse XMLi vormingut kohtades, kus andmete tootjad ja tarbijad omavahel kuigi tihedalt ei suhtle ning andmetest on tarvilik korrektne sisu ka ilma täiendava põhjaliku spetsifikatsioonita välja lugeda. Samuti on võimalik XMLi andmeid vaid hariliku tekstiredaktoriga täiendada ja parandada. Formaat on püütud kasutatavaks teha nii inimesele kui masinale. Selle arvelt võivad aga failid mõne muu vorminguga võrreldes mahukamaks minna.

Masinaga loetavaks muudetakse XML-failid range süntaksi abil. Elementide nimed kirjutatakse märkide < ja > vahele. Iga algav element peab ka lõppema. Kui elemendil puudub sisu, paigutatakse algus ja lõpp kokku. Näiteks reavahetust tähistatakse XHTMLis
, kus siis kaldkriips näitab, et elemendil eraldi lõppu pole. Kõik elemendid paigutatakse üksteise sisse või järele. Sellest järeldub, et andmetest saab moodustada puu, mis teeb nende töötlemise arvuti mälus mugavamaks. Samuti võimaldab nõnda XML kirjeldada hierarhilisi struktuure, mille ülesmärkimine omaloodud vorminguga tekstifailis vajaks suuremat pingutust.

Järgnevalt XML-faili näide, mida edaspidi näidete alusena kasutama hakatakse. Tegemist on lihtsa inimeste loeteluga, iga isiku kohta kirjas eesnimi, perekonnanimi ja sünniaasta. Üleval on XML-andmetele kohustuslik versioonideklaratsioon. Välimene element <inimesed> võtab ülejäänud teabe enese sisse. Taoline välimine juurelement on iga XML-faili juures tarvilik. Treppimine on loodud vaid pildi selguse tarbeks. Masinatega loodud andmekogudes sageli treppimist ei kasutata. Selle asemel kasutatakse vaatamisel programme, mis andmed mugavalt taandatult silma ette paigutavad.

```
<?xml version="1.0"?>
<inimesed>
 <inimene>
   <eesnimi>Juku</eesnimi>
   <perenimi>Juurikas</perenimi>
   <synd>1963</synd>
 </inimene>
 <inimene>
   <eesnimi>Juku</eesnimi>
   <perenimi>Kaalikas</perenimi>
   <synd>1961</synd>
 </inimene>
 <inimene>
   <eesnimi>Kalle</eesnimi>
   <perenimi>Kaalikas</perenimi>
   <synd>1975</synd>
 </inimene>
 <inimene>
   <eesnimi>Mari</eesnimi>
   <perenimi>Maasikas</perenimi>
   <synd>1981</synd>
 </inimene>
 <inimene>
   <eesnimi>Oskar</eesnimi>
   <perenimi>Ohakas</perenimi>
   <synd>1971</synd>
 </inimene>
</inimesed>
```

XSL

XML-failist andmete eraldamiseks on mitmeid vahendeid. Üheks levinumaks tekstiliste andmete eraldamise võimaluseks on XSL-i nimeline keel. Tegemist on samuti XML-i reegleid järgiva dokumendiga, elementide kohta aga kirjas, mida igaüks tähendab ning nende käskluste järgi on siis võimalik kõrvale antud XML-failist sobivaid andmeid välja küsida.

Järgneva XSL-faili ülesandeks on inimeste andmete eelnenud struktuuriga failist välja küsida esimene ning viimane eesnimi. Nagu aga näha, tuleb tulemuseni jõudmiseks kirjutada õige mitu rida ning muudki toimetada. Kõigepealt XSL-faili analüüs.

Et XSL on tavaline XML-reeglite järgi kirjutatud fail, siis peab esimeseks reaks paratamatult olema XMLi tüübideklaratsioon. See deklaratsioon peab hakkama faili täiesti algusest, see tähendab, et isegi vaba rida ega tühikut ei deklaratsioonirea ees olla. Muidu võib faili töötlev programm hätta jääda.

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

Järgnevalt tuleb element määramaks, et selle sees olevaid käske võib käsitleda XSL-i programmeerimiskeele käskudena. Atribuut xmlns:xsl ja järgnev URL teatavad, et kõik järgnevad xsl: algusega elemendid kuuluvad URLina näidatud konstandi määratud nimeruumi ning ei saa sealtkaudu muude elementidega segamini minna.

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

Kodeeringut määrav parameeter teatab, millisel kujul soovitakse tulemust saada. Käsklus pole hädavajalik, kuid vaikimisi väljastatav kahebaidiste tähtedega UTF-16 võib ühebaidiste tähtedega redaktorist lugedes keerukaks osutuda. Kaldkriips elemendi lõpu juures tähistas, et elemendil enam eraldi lõpukäsklust pole, kõik vajalik on siinsamas kirjas.

```
<xsl:output encoding="UTF-8" method="text" />
```

Edasine on juba rohkem andmetöötlusega seotud. Käsklust

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

võib võrrelda alustava alamprogrammiga programmeerimiskeeltes, nagu näiteks main-meetodiga C-s või Javas. Kui soovida vaid lihtsat teksti väljastada, siis kõik siinsesse elementi harilikult kirjutatu väljastatakse otse XML-i ja XSLi ühendamisel tekkivasse väljundisse.

Et aga XSL on loodud XML-faili andmete põhjal sobiva väljundi kokkupanekuks, siis saab siin vajalikust kohast andmeid küsida.

Element nimega xsl:value-of võimaldab oma select-atribuudis andmeid küsida ning vajadusel ka nende põhjal miskit kokku arvutada. XMLi faili andmete poole saab pöörduda elementide nime järgi. Kaldkriips algul tähendab, et alustatakse juurelemendist; inimene[1] ütleb, et järjestikku paiknevatesti inimestest küsitakse esimese andmeid, praegusel juhul tema eesnime väärtust. Ning jällegi elemendi lõpus kaldkriips näitamaks, et xsl:value-of ei vaja eraldi lõpukäsklust.

```
<xsl:value-of select="/inimesed/inimene[1]/eesnimi" />
```

Nagu tõlkides aimata võib, annab last() loetelu elementide arvu, ehk kokkuvõtts kätte viimase elemendi.

```
/inimesed/inimene[last()]/eesnimi
```

Ning edasi siis tuleb kõik lahti jäänud elemendid lõpetada.

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

Nüüd siis esimeseks näiteks toodud stiililehe kood tervikuna silma ette.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="UTF-8" method="text" />

 <xsl:template match="/">
   Esimene:<xsl:value-of select="/inimesed/inimene[1]/eesnimi" />;
   Viimane:<xsl:value-of select="/inimesed/inimene[last()]/eesnimi" />
 </xsl:template>
</xsl:stylesheet>
```

Käivitamine

Valmis kirjutatud XSLi fail võib sama rahulikult kettal seista nagu iga muu fail. Faili sisust saab kasu vaid juhul, kui miski programmi abil omavahel ühendada XML- ning XSL-fail. Vastavad vahendid on olemas mitme programmeerimiskeele juures. Samuti on loodud mitmeid käsurealt ja mujaltki käivitatavaid vahendeid, mille ülesandeks XSL-i kujunduse ning XML-i andmete põhjal soovitud tulemus välja küsida. Java keeles saab sellega hakkama objekt tüübist Transformer, millele tuleb siis ette anda nii sisendandmed kui voog, kuhu tulemused saata. Alates versioonist 1.4 on ennem eraldi moodulina kasutatava XMLi vahendid standardkomplekti sisse paigutatud ning piisab vaid sobivate pakettide impordist.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;
public class InimXSL1{
  public static void main(String argumendid[]) throws Exception{
    Transformer tolkija=TransformerFactory.newInstance().
      newTransformer(new StreamSource("inimesed1.xsl"));
    tolkija.transform(
      new StreamSource("inimesed.xml"),
      new StreamResult(new FileOutputStream("inimesed1.txt"))
    );
  }
}
```

Et edaspidi tarvidust mitmesuguste nimedega faile ühendada, siis ei kirjutata failinimesid mitte koodi sisse, vaid palutakse need eraldi käsurealt sisestada. Algusesse ka väikene seletus juhuks kui kasutajal pole programmikoodi käepärast või tahab ta lihtsalt mugavamalt teada, mis parameetrid sisestada tuleb. System.err'i erisuseks System.out'iga võrreldes on väljatrükk ka juhul, kui tavaväljund toruga kusagile mujale suunatakse.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;
public class XSLMuundur{
  public static void main(String argumendid[]) throws Exception{
    if(argumendid.length!=3){
      System.err.println("Kasuta kujul java XSLMuundur andmefail.xml muundefail.xsl
tulemusfail.html");
      System.exit(0);
    }
    Transformer tolkija=TransformerFactory.newInstance().
      newTransformer(new StreamSource(argumendid[1]));
    tolkija.transform(
      new StreamSource(argumendid[0]),
```

```
      new StreamResult(new FileOutputStream(argumendid[2]))
    );
  }
}
```

Kui kood kompileeritud, võib sella käima panna.

```
E:\kasutaja\jaagup\xml>java XSLMuundur inimesed.xml inimesed1.xsl tulemus.txt
```

Ning loodud tekstifaili sisu piiludes saab programmi töö tulemusi imetleda.

```
E:\kasutaja\jaagup\xml>more tulemus.txt

   Esimene:Juku;
   Viimane:Oskar
```

Tahtes väljundit otse ekraanile suunata, piisab DOS-i keskkonna puhul määramaks väljundfaili nimeks con.

```
E:\kasutaja\jaagup\xml>java XSLMuundur inimesed.xml inimesed1.xsl con

   Esimene:Juku;
   Viimane:Oskar
```

Ühenimelised elemendid

Üheks võimaluseks andmete poole pöördumisel on anda ette elemendi asukoht alates dokumendi juurest. Kui aga soovitakse kõiki samanimelisi elemente sõltumata nende asukohast dokumendis, siis võib päringus kirjutada elemendi nime ette kaks kriipsu ning võimegi pöörduda kõigi inimeste kui ühise massiivi poole. Edasi nurksulgudes lisatakse piirang, juhul kui soovitakse lähemalt tegelda vaid alamhulgaga.

```
<?xml version="1.0"?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="/">
       <xsl:for-each select="//inimene[synd>'1970']">
         <xsl:value-of select="eesnimi" />
       </xsl:for-each>
 </xsl:template>
 </xsl:stylesheet>

KalleMariOskar
```

Andmed tabelina

Sarnast tüüpi andmete esitamiseks on tabel üks hea ja lihtne moodus. Ning HTML-i käsklusi saab väljundisse paigutada sarnaselt harilikulegi tekstile. Kui on vaja sama tüüpi elemendid tsükliga läbi käia, siis XSLi juures selleks käsklus xsl:for-each. Atribuudis select määratakse, siis millise nime ja paiknemisega elemendid läbitakse.

```
  <xsl:for-each select="/inimesed/inimene">
```

Et siin soovitakse iga inimese puhul ka kõik alamelemendid läbi käia, siis tärn annab selleks võimaluse.

```
      <xsl:for-each select="*">
```

Tabeli enese, rea ning lahtri alguse ja lõpu elemendid paigutatakse lihtsalt sobivatesse kohtadesse vahele. Paljas punkt xsl:value-of elemendi sees palub kirjutada jooksva elemendi väärtuse. Nõnda kui välimene for-each käib läbi kõik inimesed ning

sisemine iga inimese alamtunnused ning andmefailis on kõigil inimestel sama palju tunnuseid saabki kokku täiesti viisaka HTML-tabeli.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="UTF-8" method="html" />

 <xsl:template match="/">
  <html><body>
  <table>
  <xsl:for-each select="/inimesed/inimene">
    <tr>
      <xsl:for-each select="*">
        <td>
          <xsl:value-of select="." />
        </td>
      </xsl:for-each>
    </tr>
  </xsl:for-each>
  </table>
  </body></html>
 </xsl:template>

</xsl:stylesheet>
```

Tabel ise selline nagu ikka seda ette kujutada võib. Et ta loodud programmikoodi abil ning kood paistis küllalt kergesti kontrollitav olema, siis võib loota, et loodud tabelis kõik read ja lahtrid ilusti alustatud ja lõpetatud on.

```
<html>
<body>
<table>
<tr>
<td>Juku</td><td>Juurikas</td><td>1963</td>
</tr>
<tr>
<td>Juku</td><td>Kaalikas</td><td>1961</td>
</tr>
<tr>
<td>Kalle</td><td>Kaalikas</td><td>1975</td>
</tr>
<tr>
<td>Mari</td><td>Maasikas</td><td>1981</td>
</tr>
<tr>
<td>Oskar</td><td>Ohakas</td><td>1971</td>
</tr>
</table>
</body>
</html>
```

Mallid

Tabeli saab algandmete põhjal kokku panna ka täiesti tsükleid kasutamata. Juhul, kui soovitakse andmed algses failis ning tulemusfailis enamjaolt samasse järjekorda jätta, siis sobivad elementide nimede muutmiseks ja väärtuste lisamiseks malllid. Iga elemendi puhul saab määrata, millisel kujul teda väljundis näidatakse. Järgnevat näidet täheepanelikumalt silmitsedes võib näha plokke, kus elemendi nimeks on xsl:template ning sees HTML-i käsud ja keskel xsl:apply-templates. Lahtiseletatult tähendavad koostatud template'd juhiseid XSLi järgi algandmeid töötlevale programmile. Kogu dokumendi algust tähistab "/", ülejäänud mallid vastavad igaüks oma elemendile, mille nimi match-atribuudis kirjas on.

```
 <xsl:template match="inimesed">
   <table>
     <xsl:apply-templates />
   </table>
 </xsl:template>
```

Sellline kirjeldus tähendab näiteks, et kui faili töötlemisel on jõutud elemendini nimega "inimesed", siis kirjutatakse väljundisse kõigepealt <table>. Edasi töödeldakse inimesed-nimelise elemendi olemasolev sisu ning lõppu kirjutatakse </table>. Sarnaselt asendatakse element "inimene" tabeli ühe reaga. Elemendi sisu väljatrükk on näidatud järnevas lõigus. Malli sees olles saab jooksva elemendi poole pöörduda sümboli "." kaudu. Siin on pandud igale väärtusele ka tabeli lahtrit tähistab <td> element ümber. Vaikimisi juhul, kui palutakse apply-templates käsu abil faili dokumenti edasi analüüsida, siis ettejääv tekst trükitaks samuti lihtsalt ekraanile.

```
<xsl:template match="eesnimi|perenimi|synd">
   <td>
     <xsl:value-of select="." />
   </td>
</xsl:template>
```

Ning nüüd siis tervikuna XSLi kood, kus mallide abil muudetakse inimeste andmete loetelu tabeliks, kus igal real inimese andmed. Kogu dokumendi algusesse ja lõppu paigutatakse vastavad HTML-märgendid. Kõiki inimesi ühendav element inimesed muudetakse HTML-i elemendiks <table>. Igale inimesele hakkab vastama tabeli rida <tr> ning iga inimese iga üksiku tunnuse väärtusele tabeli lahter <td>.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="UTF-8" method="html" />

<xsl:template match="/">
 <html><body>
   <xsl:apply-templates />
 </body></html>
</xsl:template>

<xsl:template match="inimesed">
  <table>
    <xsl:apply-templates />
  </table>
</xsl:template>

<xsl:template match="eesnimi|perenimi|synd">
   <td>
     <xsl:value-of select="." />
   </td>
</xsl:template>

<xsl:template match="inimene">
  <tr>
    <xsl:apply-templates />
  </tr>
</xsl:template>
</xsl:stylesheet>
```

Tekstikontroll

Ehkki XSL pole mõeldud suuremamahulisteks arvutusteks ja tekstitöötluseks, õnnestub lihtsamad võrdlused ja kokkuvõtted siin täiesti teha. Järgnevalt siis loetelu nende inimeste perekonnanimedest, kelle eesnimi algab J-iga. Algandmete peale vaadates leiame sealt kaks Jukut: üks Juurikas ning teine Kaalikas. Vastavad perekonnanimed saadakse jätte järgneva avaldise abil.

```
<xsl:for-each select="/inimesed/inimene[starts-with(eesnimi, 'J')]/perenimi" >
```

Kandiliste sulgude sees määratakse ära, millistele tingimustele vastavaid inimesi loetellu võetakse. Siin juhul siis kontrolliks XSLi funktsioon nimega starts-with, parameetriteks kontrollitav tekst ning otsitav algustekst. Ning nagu muud

tõeväärtusfunktsioonid, nii ka siin on väärtuseks jah või ei. Ning loetellu jäävad need inimesed, kel avaldise puhul väärtuseks jah. Ning nõnda saab tsükli sees otsitud tulemused välja kirjutada.

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="UTF-8" method="html" />

<xsl:template match="/">
  <xsl:for-each select="/inimesed/inimene[starts-with(eesnimi, 'J')]/perenimi" >
     <xsl:value-of select="." />;
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Käivitamine nii nagu eelnenud näite puhul

```
E:\kasutaja\jaagup\xml>java XSLMuundur inimesed.xml inimesed4.xsl inimesed4.txt
```

Ning faili sisu piiludes võime just soovitud read selle seest avastada.

```
E:\kasutaja\jaagup\xml>more inimesed4.txt
Juurikas;
  Kaalikas;
```

XSL-i funktsioone

Võrreldes "päris" programmeerimiskeeltega võib XSLi käsustikku väikeseks pidada, kuid ega algses Basicus neid kuigi palju rohkem polnud. Järgnevalt on ära toodud suurem osa levinumatest käsklustest.

```
last()        viimase järjekorranumber
position()    jooksva järjekorranumber
count(plokk)  elementide arv plokis
name(plokk)   ploki juurelemendi nimi
```

Funktsiooni name läheb näiteks vaja, kui soovitakse luua mõnd üldist malli, mis saaks hakkama mitmesuguse struktuuriga algandmete korral ning kus soovitakse algse elemendi nime säilitada või näiteks tabeli lahtri pealkirjana kasutada.

Tõeväärtusfunktsioonid nagu ikka arvata võib; not keerab olemasoleva väärtuse vastupidiseks, ülejäänud kahe puhul on tegemist lihtsalt konstandiga.

```
not(toevaartus)
true()
false()
```

Samuti võib nime järgi ära aimata enamike arvudega tegelevate funktsioonide ülesanded. Käsklust number kasutatakse lihtsalt tüübimuunduseks, samuti nagu käsuga string saab andmed taas tagasi tekstikujule.

```
number(objekt)
string(objekt)

sum(plokk)    väljastab summa juhul, kui elemendid on numbrid
floor(number)
round(number)
```

Sõnefunktsioonid

`concat(s1, s2, s3*)`
: Parameetritena antud tekstid liidetakse. Elementide arv ei ole piiratud.

`starts-with(s1, s2)`
: Kontrollitakse, kas esimesena antud tekst algab teisena antud tekstiga.

`contains(s1, s2)`
: Võrreldes eelmisega ei pruugita alustada algusest, vaid otsitakse lõigu leidmist kogu teksti ulatuses.

`substring-before(s1, s2)`
`substring-after(s1, s2)`
`substring(s1, start, nr?)`
: Käsud lõigu eraldamiseks tekstist

`string-length(s1)`
: Nagu nimest näha, küsitakse teksti pikkust.

`normalize-space(s1)`
: Võtab algusest ja otstest tühikud, muud vahed teeb üheks tühikuks. Kasulik näiteks erikujuliste sisestatud tekstide võrdlemisel või lihtsalt väljundi viisakamaks muutmisel. XMLi andmete juures ei mängi korduvad tühikud rolli, küll aga neist võib tüli tekkida mõnda muusse kohta suunduva väljundi puhul.

`translate (s1, algsümbolid, lõppsümbolid)`
: Tähtede asendamiseks leiab harjumatu kujuga funktsiooni. Näite järgi on aga ehk toimimine mõistetav: translate('pann', 'an', 'ek') -> 'pekk'

Parameetrid

Kui samade algandmete põhjal tahetakse kokku panna märgatavalt erinevaid tulemusi, siis tuleb üldjuhul igaks muundamiseks valmis kirjutada omaette XSL-leht. Näiteks HTML- ja WAP-väljund näevad nõnda erinevad välja, et ühist muundajat kirjutada oleks raske. Kui aga valida lihtsalt eri resolutsioonidele arvestatud HTML-i vahel, siis võib XSLi parameetri abil kord rohkem, kord vähem lähteandmeid sisse võtta. Samuti, kui näiteks soovitakse näidata lehel inimeste vanuseid, salvestatud on aga sünniaastad, siis parameetrina antud praeguse aastaarvu järgi saab vähemalt ligikaudugi tulemuse parajaks sättida.

Parameetri väärtus tuleb määrata eraldi elemendina enne mallikirjelduste algust. Nagu näha, tuleb parameetri nimi atribuudina, väärtus aga elemendi väärtusena.

```
<xsl:param name="pikkus">5</xsl:param>
```

Hiljem atribuudi väärtust küsides tuleb avaldises selle nimele dollarimärk ette panna. Ilma dollarita tähendaks see vastavanimelist XML-elementi.

```
<xsl:value-of select="$pikkus" />
```

Andmete sortimiseks tuleb tsükli sisse paigutada alamelement nimega xsl:sort ning parameetrina määrata, millise elemendi väärtuse järgi sorteeritakse. Nagu mujal, nii ka siin oleks võimalik parameetriks koostada avaldis, mis järjestamise peenemalt ette määraks.

```
    <xsl:sort select="eesnimi" order="descending" />
```

Soovides väljatrüki abil tühikuga eraldatud ees- ja perekonnanime, aitab funktsioon concat. Muul juhul tuleks sama tulemuse saavutamiseks xsl:value-of element mitmel korral välja kutsuda.

```
     <xsl:value-of select="concat(eesnimi, ' ', perenimi)" />;
```

Ning näide tervikuna.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="UTF-8" method="html" />

<xsl:param name="otsing">ar</xsl:param>
<xsl:param name="pikkus">5</xsl:param>

<xsl:template match="/">
  Nimed, mis sisaldavad kombinatsiooni <xsl:value-of select="$otsing" />:
  <xsl:for-each select="/inimesed/inimene[contains(eesnimi, $otsing)]">
    <xsl:sort select="eesnimi" order="descending" />
     <xsl:value-of select="concat(eesnimi, ' ', perenimi)" />;
  </xsl:for-each>

  Nimed pikkusega <xsl:value-of select="$pikkus" /> ja rohkem:
  <xsl:for-each select="/inimesed/inimene[string-length(eesnimi)>=$pikkus]" >
     <xsl:value-of select="concat(eesnimi, ' ', perenimi,
           ' varuks ', string-length(eesnimi)-$pikkus)" />;
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Käivitus

```
E:\kasutaja\jaagup\xml>java XSLMuundur inimesed.xml inimesed4a.xsl inimesed4a.txt
```

ja tulemus

```
E:\kasutaja\jaagup\xml>more inimesed4a.txt

  Nimed, mis sisaldavad kombinatsiooni ar:
  Oskar Ohakas;
  Mari Maasikas;


  Nimed pikkusega 5 ja rohkem:
  Kalle Kaalikas varuks 0;
  Oskar Ohakas varuks 0;
```

Parameetrite väärtuste muutmiseks ei pea alati tekstiredaktoriga muutma XSLi faili sisu. Neid saab sättida ka otse XMLi ja XSLi kokkusiduvas koodis ning ka ühendava programmi väljakutsel. Nõnda on ka siin näha, kus pikkusele antakse väärtuseks neli.

```
    tolkija.setParameter("pikkus", "4");
```

Muus osas näeb faile ühendav käivitusprogramm eelnenuga sarnane välja.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;
public class XSLParameetrid{
  public static void main(String argumendid[]) throws Exception{
    Transformer tolkija=TransformerFactory.newInstance().
      newTransformer(new StreamSource("inimesed4a.xsl"));
    tolkija.setParameter("pikkus", "4");
    tolkija.transform(
      new StreamSource("inimesed.xml"),
      new StreamResult(new FileOutputStream("inimesed4a.txt"))
    );
  }
}
```

Ka käivitamine sarnane

```
E:\kasutaja\jaagup\xml>java XSLParameetrid
```

Ning tulemusena näeb siis nelja tähe pikkusi ja pikemaid nimesid.

```
E:\kasutaja\jaagup\xml>more inimesed4a.txt

  Nimed, mis sisaldavad kombinatsiooni ar:
  Oskar Ohakas;
  Mari Maasikas;


  Nimed pikkusega 4 ja rohkem:
  Juku Juurikas varuks 0;
  Juku Kaalikas varuks 0;
  Kalle Kaalikas varuks 1;
  Mari Maasikas varuks 0;
  Oskar Ohakas varuks 1;
```

Ülesandeid

XML

```
Sisestusharjutus

* Koosta eesnimede loetelu.
* Koosta inimeste loetelu, kus isikuandmeteks on eesnimi,
perekonnanimi ja
  synniaasta.
```

Andmepuu

```
* Kirjuta XML-i abil üles sugupuu alates oma vanaisast.
* Näita puus ka abikaasad.
```

XSL

```
* Loo XSL leht, mis sõltumata sisendandmetest väljastab "Tere".

Loetelus on vähemalt viie inimese andmed: eesnimi, perekonnanimi ning
sünniaasta.

* Väljastatakse teise inimese sünniaasta
* Andmed väljastatakse tabelina nii mallide (template) abil.
* Andmed väljastatakse tabelina  for-each tsükli
abil. Tulpade pealkirjad on rasvased.
* Luuakse SQL-laused inimeste andmete lisamiseks baasi.
* Väljastatakse semikoolonitega eraldatud loetelu vanuste
järjekorras.
* Parameetrina antakse ette käesoleva aasta number.
  Iga inimese kohta väljastatakse nimi ja vanus.
```

Sugupuu

```
Sugupuus on vähemalt kolme põlve andmed, iga inimese kohta vähemalt
ees- ja
 perekonnanimi ning sünniaasta.

* Trükitakse välja inimeste nimed ning nende laste arv.
* Väljastatakse nimed, kel on vähemalt kaks last.
* Väljastatakse nimed, kellel on sugupuus vanavanem.
* Andmepuus muudetakse sünniaasta atribuudiks.
```

XML ja kassid

```
*  Loo XML-fail kus on kirjas kasside nimed ja nende sünniaastad
*  Kirjuta XSL-i abil andmed välja, määrates nimed pealkirjadeks ning
iga
pealkirja alla kirjutada teksti sisse, millisel aastal vastav kass
sündis.
*  Lisaks eelmisele väljasta andmed sorteerituna sünniaastate järgi.
```

XML ja koerad

```
*  Loo XML-fail, kus kirjas koert nimed ja tõud.
*  Väljasta iga koer eraldi real ning muuda tõug rasvaseks.
*  Lisaks eelmisele muuda paiguta kõik viie tähe pikkused nimed
kaldkirja.
```

DOM

Nagu mitemetes keeltes, nii ka Java puhul leiduvad vahendid XMLi andmepuu loomiseks, lugemiseks ja muutmiseks. Võrrelduna lihtsalt tekstikäskude abil töötlemisele saab siin programmeerija enam keskenduda andmete paiknemise loogikale. Samuti hoiab puu elementide loomine ja nende poole pöördumine ära teksti loomisel tekkida võivad trükivead. Abikäskudega õnnestub küsida sobivaid elemente. Kasutatavad avaldised pole küll veel nõnda paindlikud kui XSLi juurde kuuluva XPathi omad, kuid märgatavat kasu on neistki.

Järgnevalt DOM-i tutvustus väikese näite abil. Keskseks klassiks on Document. Selle eksemplari kasutatakse nii uute elementide loomiseks kui elemendihierarhia algusena. Dokument võidakse luua kas tühjana tühjale kohale

```
    Document d=DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument
();
```

või siis lugeda sisse olemasolevast failist.

```
//    Document d=DocumentBuilderFactory.newInstance().newDocumentBuilder().parse
("linnad.xml");
```

Nagu näha, ei kasutata dokumendi loomisel mitte klassi konstruktorit, vaid selle asemel koostatakse vabriku (DocumentBuilderFactory) eksemplar, mille abil siis dokument kokku pannakse. Selline pikk lähenemine võimaldab mitmel pool seada parameetreid, samuti kasutada ilma koodi muutmata mitmete tootjate abitükke.

Et dokumenti saaks elemente lisada, peab selles olema vähemasti juurelement. Failist lugedes tuleb see kaasa, tühja dokumendi loomisel tuleb aga ka juur luua ja määrata. Ning nagu XMLi spetsifikatsioonis öeldakse, peab igal failil või dokumendil olema üks ja ainult üks juur. Nii nagu hiljemgi elementide puhul, nii ka siin tuleb eraldi käskudena element luua ning siis sobivasse kohta lisada.

```
Element juur=d.createElement("linnad");
d.appendChild(juur);
```

Edasi siis dokumendile külge ka sisulised andmed, mis praegu lihtsuse mõttes võetakse massiivist.

```
String[] linnanimed={"Tallinn", "Tartu", "Narva"};
```

Tekstiliste andmete XML-i puusse kinnitamiseks tuleb kõigepealt luua teksti puusse kinnitatavaks tervikuks ühendav TextNode, mis siis omakorda nimega elemendi sisse paigutada. Et siin näites on juurelemendiks "linnad", selle all elemendid nimega "linn" ning edasi vastava elemendi sees omakorda linnanimi, näiteks "Tartu".

```
for(int i=0; i<linnanimed.length; i++){
  Element e=d.createElement("linn");
  e.appendChild(d.createTextNode(linnanimed[i]));
  juur.appendChild(e);
}
```

Soovides valmisehitatud puud talletada, tuleb puu viia voo kujule. Seda aitab klassi Transformer eksemplar. Piisab vaid käsust transform, ning andmepuu muudetaksegi vooks. Kas voog suunatakse faili, ekraanile või võrku, see on juba programmeerija mure ning selleks piisab vastava voo ots transformeerija väljundisse pakkuda. Siin nagu näha viib System.out tulemuse ekraanile.

```
Transformer t=TransformerFactory.newInstance().newTransformer();
t.transform(new DOMSource(d), new StreamResult(System.out));
```

Järgnevalt programmi kood.

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

import org.w3c.dom.*;

public class UusDokument{
  public static void main(String argumendid[]) throws Exception{
    String[] linnanimed={"Tallinn", "Tartu", "Narva"};
//    Document d=DocumentBuilderFactory.newInstance().newDocumentBuilder().parse
("linnad.xml");
    Document d=DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument
();
    Element juur=d.createElement("linnad");
    d.appendChild(juur);
    for(int i=0; i<linnanimed.length; i++){
      Element e=d.createElement("linn");
      e.appendChild(d.createTextNode(linnanimed[i]));
```

```
      juur.appendChild(e);
    }
    Transformer t=TransformerFactory.newInstance().newTransformer();
    t.transform(new DOMSource(d), new StreamResult(System.out));
  }
}
```

Ning väljund.

```
E:\kasutaja\jaagup\xml>java UusDokument
<?xml version="1.0" encoding="UTF-8"?>
<linnad><linn>Tallinn</linn><linn>Tartu</linn><linn>Narva</linn></linnad>
```

Lihtsalt tekstiekraanile kirjutatuna võib linnade rida olla halvasti loetav. Kui aga sama XMLi lõik salvestada faili ja avada seiluriga, siis õnnestub hulga selgemini lugeda.

```
- <linnad>
    <linn>Tallinn</linn>
    <linn>Tartu</linn>
    <linn>Narva</linn>
  </linnad>
```

Joonistusvahend

Järgnevalt veidi pikem näide, kus pildi andmete salvestamisel kasutatakse XMLi andmepuud. Puu loomise käsud samad kui lühemagi näite puhul. Juures käsklused puust andmete lugemiseks ning joonistuspool. Tähtsamad nupud avamise ja salvestamise jaoks. Iga hiire vedamisega tekib joon, joone juures jäetakse meelde koordinaadid, mida hiir läbinud. Nii nagu eelmises näites oli juurelemendiks "linnad" ning selle all hulk elemente "linn", nii siin on juurelemendiks "koordinaadid" ning juure küljes elemendid "joon". Iga joone sees omakorda hulk elemente nimega "punkt", milles koordinaate tähistavad "x" ja "y".

Algus nagu eelmiselgi korral. Nii alustuse kui kustutuse puhul luuakse uus tühi dokument ning sinna sisse juurelement.

```
d=DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
juur=d.createElement("koordinaadid");
d.appendChild(juur);
```

Iga hiirevajutuse puhul luuakse uus element nimega "joon" ning jäetakse vastava nimega muutujasse meelde. Nii vajutuse kui lohistuse puhul lisatakse joone külge punkt.

```
  public void mousePressed(MouseEvent e) {
    joon=d.createElement("joon");
    juur.appendChild(joon);
    lisaPunkt(e.getX(), e.getY());
  }

  public void mouseDragged(MouseEvent e){
     lisaPunkt(e.getX(), e.getY());
  }
```

Selle suhteliselt tervikliku tegevuse tarbeks loodi omaette alamprogramm. Isendi piires kättesaadava muutuja joon külge lisatakse element punkt, mille juurde omakorda x ja y oma koordinaatide tekstilise väärtusega.

```
void lisaPunkt(int x, int y){
  Element punkt=d.createElement("punkt");
  Element px=d.createElement("x");
  px.appendChild(d.createTextNode(x+""));
  punkt.appendChild(px);
  Element py=d.createElement("y");
  py.appendChild(d.createTextNode(y+""));
  punkt.appendChild(py);
  joon.appendChild(punkt);
}
```

Joonistamine näeb välja mõnevõrra keerukam. Graafikakomponentide puhul soovitatakse kogu joonistus ette võtta paint-meetodis ning meetodit sobival ajal välja kutsuda. Ka pole lihtsuse mõttes siin näites puhvrit vahele loodud, nii et iga ülejoonistuskäsu peale koostatakse kogu pilt uuesti.

```
public void paint(Graphics g){
```

Alustuseks küsitakse massiivina välja kõik dokumendis paiknevad jooned. Et elementide paiknemine on teada, siis piisab küsimisest asukoha ning mitte nime järgi. Dokumendi d käsk getFirstChild() väljastab juurelemendi "koordinaadid", sealt käsk getChildNodes() väljastab joonte kogumi.

```
NodeList jooned=d.getFirstChild().getChildNodes();
```

Edasises tsüklis käiakse jooned ükshaaval läbi ning palutakse nad kõik ekraanile joonistada.

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

Iga joon määratakse punktikogumiga, mis jooneelemendist samuti välja küsitakse nagu jooned juurelemendi küljest. Ning näiliselt vabakäejoon koosneb üksikute punktide vahele tõmmatud sirglõikudest.

```
NodeList punktid=jooned.item(i).getChildNodes();
for(int j=1; j<punktid.getLength(); j++){
```

Sirglõigu tõmbamiseks läheb vaja kahte punkti. Nõnda käiaksegi tsükkel läbi ühe korra vähem kui punkte kokku. Ning igal korral küsitakse punkt nii loenduri juurest kui ka ühe võrra eelnevast kohast.

```
Node p1=punktid.item(j-1);
Node p2=punktid.item(j);
```

Edasiseks joonistamiseks on vaja kätte saada punktide sees paiknevate koordinaatide arvulised väärtused. Näidatakse mitut võimalust selle välja küsimiseks. Klassi Element eksemplaril leidub käsklus getElementsByTagName, mis väljastab kogumi soovitud nimega elementidega. Et iga punkti juures leidub täpselt üks x, siis item(0) peaks just selle väljastama. Koordinaadi väärtuse saamiseks tuleb aga elemendi seest küsida "nähtamatu" TextNode ning selle seest omakorda sõnena väärtus ehk getNodeValue(). Integer.parseInt annab arvulise väärtuse.

```
int x1=Integer.parseInt(((Element)p1).getElementsByTagName("x").item(0).
                                                    getFirstChild().getNodeValue
());
```

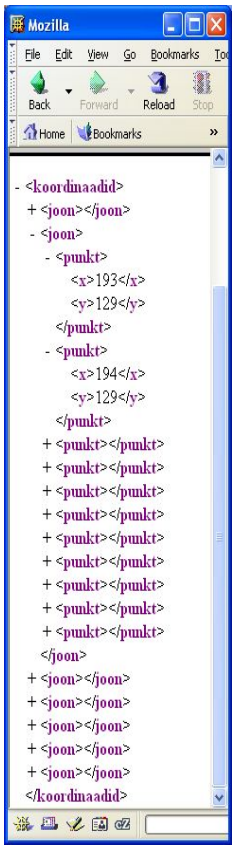
Väärtuse võib välja küsida ka vaid asukoha järgi. Lahti seletatult: p1.getFirstChild annab punkti esimese alamelemendi ehk x-i. Käsk getNextSibling küsib järgmise sama taseme elemendi ehk y-i. Sealt juba TextNode ning selle väärtus.

```
        int y1=Integer.parseInt(p1.getFirstChild().getNextSibling().
                                          getFirstChild().getNodeValue());
        int x2=Integer.parseInt(p2.getFirstChild().getFirstChild().getNodeValue());
        int y2=Integer.parseInt(p2.getFirstChild().getNextSibling().
                                           getFirstChild().getNodeValue());
```

Kui kõik neli koordinaati nõnda käes, võib tõmmata joone.

```
        g.drawLine(x1, y1, x2, y2);
      }
    }
  }
```

Salvestamise juures mõistavad valmiskäsud suurema osa tööst ära teha. Transformeerijale tuleb vaid öelda, kust andmed võtta ja kuhu panna. Praegusel juhul siis mälus paiknevast DOMi puust faili suunduvasse väljundvoogu. Süntaktiliselt oleks saanud failivoo loomise ka otse transform-käsu parameetri sisse paigutada, kuid sellisel juhul pole kindel, et fail ka suletakse ning kõik baidid faili jõuavad. Eraldi close-käskluse puhul seda muret pole.




```
    if(e.getSource()==salvesta){
      try{
          Transformer t=TransformerFactory.newInstance().
                        newTransformer();
          FileOutputStream valja=
                      new FileOutputStream(failinimi);
          t.transform(new DOMSource(d),
                      new StreamResult(valja));
          valja.close();
      } catch(Exception viga){ viga.printStackTrace();   }
    }
```

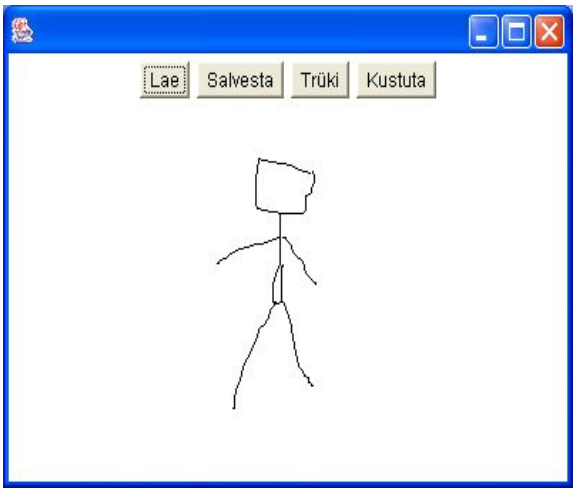
Ekraanile testiks trükkimine veelgi lihtsam. Kui soovida XML-i koodi väljastada, siis piisab Node/sõlme väljatrükist. Meetod toString hoolitseb juba ise vajaliku väljundi kuju eest.

```
    if(e.getSource()==tryki){
       System.out.println(d.getFirstChild());}
```

Ning rakenduse kood tervikuna.

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

import org.w3c.dom.*;

public class XMLJoonis extends Frame implements ActionListener,
   MouseListener, MouseMotionListener{

  Button nupp = new Button("Lae");
  Button salvesta=new Button("Salvesta");
  Button tryki=new Button("Trüki");
  Button kustuta=new Button("Kustuta");
  String failinimi="joonistusandmed.xml";
  Document d;
  Node juur, joon;

  public XMLJoonis(){
      setLayout(new FlowLayout());
```

```
    add(nupp);    add(salvesta);
    add(tryki);   add(kustuta);
    nupp.addActionListener(this);
    salvesta.addActionListener(this);
    tryki.addActionListener(this);
    kustuta.addActionListener(this);
    addMouseListener(this);
    addMouseMotionListener(this);
    addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent evt){
       System.exit(0);
      }
    });
    alusta();
    setSize(400,300);
    setVisible(true);
}

public void actionPerformed(ActionEvent e){
  if(e.getSource()==nupp){
   try{
       d=DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(failinimi);
       juur=(Element)d.getFirstChild();
       repaint();
   } catch(Exception viga){System.out.println("Probleem lugemisel: "+viga);   }
  }
  if(e.getSource()==salvesta){
    try{
           Transformer t=TransformerFactory.newInstance().newTransformer();
           FileOutputStream valja=new FileOutputStream(failinimi);
           t.transform(new DOMSource(d), new StreamResult(valja));
           valja.close();
    } catch(Exception viga){ viga.printStackTrace();   }
  }
  if(e.getSource()==tryki){System.out.println(d.getFirstChild());}
  if(e.getSource()==kustuta){alusta();}
}


public void paint(Graphics g){
  NodeList jooned=d.getFirstChild().getChildNodes();
  for(int i=0; i<jooned.getLength(); i++){
    NodeList punktid=jooned.item(i).getChildNodes();
    for(int j=1; j<punktid.getLength(); j++){
      Node p1=punktid.item(j-1);
      Node p2=punktid.item(j);
      int x1=Integer.parseInt(((Element)p1).getElementsByTagName("x").item(0).
               getFirstChild().getNodeValue());
      int y1=Integer.parseInt(p1.getFirstChild().getNextSibling().getFirstChild().
                  getNodeValue());
      int x2=Integer.parseInt(p2.getFirstChild().getFirstChild().getNodeValue());
      int y2=Integer.parseInt(p2.getFirstChild().getNextSibling().getFirstChild().
                  getNodeValue());
      g.drawLine(x1, y1, x2, y2);
    }
  }
}

public void mousePressed(MouseEvent e) {
  joon=d.createElement("joon");
  juur.appendChild(joon);
  lisaPunkt(e.getX(), e.getY());
}

public void mouseDragged(MouseEvent e){
   lisaPunkt(e.getX(), e.getY());
}

void lisaPunkt(int x, int y){
  Element punkt=d.createElement("punkt");
  Element px=d.createElement("x");
  px.appendChild(d.createTextNode(x+""));
  punkt.appendChild(px);
  Element py=d.createElement("y");
  py.appendChild(d.createTextNode(y+""));
  punkt.appendChild(py);
  joon.appendChild(punkt);
}


public void alusta(){
```



```
      try{
         d=DocumentBuilderFactory.newInstance().newDocumentBuilder().newDocument();
         juur=d.createElement("koordinaadid");
         d.appendChild(juur);
         repaint();
      } catch(Exception viga){System.out.println("Viga dokumendi loomisel: "+viga);}
  }

  public void mouseClicked(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { repaint(); }
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mouseMoved(MouseEvent e){}


  public static void main(String argumendid[]) throws Exception{
    new XMLJoonis();
  }
}
```

SAX

Sellise nimega vahend on loodud mahukate XML-andmekogude töötlemiseks, mida pole võimalik või mõistlik korraga mällu lugeda. Mõnevõrra sarnaneb siinne töötlus lihtsa tekstifaili reakaupa lugemisele. Ainult, et tervikuteks pole mitte faili read, vaid XML-i elemendid. Nõnda päästetakse programmeerija XMLi süntaksi lahtiharutamisest ning ta saab keskenduda tegelike andmete töötlemisele.

XMLi elementide ja väärtuste teated saadetakse dokumendis paiknemise järjekorras selleks otstarbeks loodud objektile. Sarnaselt, nagu näiteks hiiresündmustele reageerimise puhul peab olema ka määratud objekt andmete püüdmiseks. Hiireteadete puhul on vastavaks liideseks MouseListener, siin ContentHandler. Ning et kõiki käske ei peaks üle katma, selleks on hiireteadete puhul olemas MouseAdapter, siin aga DefaultHandler. Kui oma sündmuseid püüdva klassi loome DefaultHandleri alamklassina, siis piisab meil vaid nende meetodite ülekatmisest, millele reageerida soovime. Järgnevas näites on nendeks näha startElement ja endDocument, aga ContentHandleri liideses on neid kokku kümmekond. Nagu aimata võib, tähtsamatena veel startDocument ja endElement ning vahend elementide sisu püüdmiseks. Elemendi nime saab kätte startElement-nimelise funktsiooni kolmandast parameetrist. Kui on vaja atribuutide sisu küsida, siis nendeni pääseb neljanda parameetri alt tuleva atribuutide kogumi kaudu.

Nimede loendur

Siin näites lihtsalt iga elemendi algamise puhul kontrollitakse, kas elemendi nimeks oli "eesnimi" ning sel juhul suurendatakse loendurit. Kui saabub teade endDocument, siis järelikult on dokument läbi ning võime kokku loetud arvu välja trükkida. Jooksev eesnimede arv on kogu aeg kirjas vastavanimelises muutujas.

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class EesnimedeLoendaja extends DefaultHandler{
  int eesnimedeArv=0;
  public void startElement(String nimeruum, String kohalik,
            String element, Attributes at){
     if(element.equals("eesnimi"))eesnimedeArv++;
  }
  public void endDocument(){
    System.out.println("Leiti "+eesnimedeArv+" eesnime.");
  }
```

```
  public static void main(String argumendid[]) throws Exception{
    XMLReader lappaja=  SAXParserFactory.newInstance().newSAXParser().getXMLReader();
    lappaja.setContentHandler(new EesnimedeLoendaja());
    lappaja.parse("inimesed.xml");
  }
}
```

Väljund nagu algandmete põhjal aimata oligi:

```
E:\kasutaja\jaagup\xml>java EesnimedeLoendaja
Leiti 5 eesnime.
```

Suurema analüüsi puhul võib meelespeetavaid väärtusi rohkem olla. XMLReader hoolitseb lihtsalt selle eest, et dokumendis leidumise järjekorras saaks õiged käsklused õigete parameetritega välja kutsutud. Kõik muu jääb programmeerija hooleks. Järgnevalt loetakse kokku ja trükitakse välja eesnimede väärtused.

Elementide sisu

Kui elementide nimed saab kergesti kätte, siis elementide sisu kinnipüüdmiseks tuleb veidi rohkem vaeva näha. Põhjus tõenäoliselt selles, et suuremates XML-dokumentides võivad tekstiosad ka näiteks mitme megabaidi pikkused olla ning neid ei pruugi mõistlik ega võimalik olla korraga mällu lugeda. Samuti kasutatakse SAXi olukordades, kus suurest dokumendist vajatakse vaid üksikuid andmeid ning siis pole ülejäänud andmete muundamine kergesti loetavale kujule vajalik.

Igal pool, kus faili läbimisel jõutakse elemendi sildist väljapool asuvale tekstile, kutsutakse välja meetod characters. Meetodi väljakutsumise kordade arv võib olla ka suurem kui üks - juhul, kui tekstilõik edastatakse kuularile mitmes osas. Nõnda siis peab tervikliku teksti kokku lappimiseks programmeerija mõned read kirjutama, et ühe elemendi sisese teksti tervikuna kätte saada. Samuti võib ebatavalisena paista teksti esitamise moodus: tähemassiiv ning kaks arvu funktsiooni parameetritena. Mõte on tõenäoliselt jälle seotud jõudlusega. Faili analüüsil loetakse sellest korraga mällu miski suurusega plokk. Edaspidi pole seda plokki vaja mälus liigutada enne, kui uus plokk selle asemele loetakse. Ning kui tahetakse kasutajale teada anda, et tal on võimalus tekstilõik enese valdusesse hankida, siis teatatakse talle vaid ploki asukoht, teksti esimese tähe järjekorranumber ning tähtede arv. Ning ainult juhul, kui kasutajal teksti parajasjagu vaja läheb, tuleb tal see lõik omale sobivas formaadis välja küsida. Õnneks on klassil String olemas konstruktor, mis tahab just samad kolm parameetrit: tähemassiivi, vajaliku teksti alguse ja tähtede arvu. Nõnda saab vajadusel ühe käsuga soovitud sõne kätte.

Tingimuslause abil tuleb kontrollida, kas parasjagu soovitud teksti vaja on. Praegusel juhul näitab muutuja kasEesnimi, et kas ollakse analüüsiga eesnime elemendi sees.

```
public void characters(char[] tahed, int algus, int pikkus){
  if(kasEesnimi){
      puhver.append(new String(tahed, algus, pikkus));
  }
 }
```

Iga kord eesnime elementi sisenedes luuakse uus tühi puhvri eksemplar. Meetodis characters liidetakse eesnimi tükkidest kokku, juhul kui andmed peaksid osade kaupa saabuma. Muidu pannakse puhvrisse lihtsalt üks terviklik tükk. Kui

eesnime element lõpeb, siis trükitakse puhvri sisu ekraanile ning nõnda saabki kasutaja näha eesnimede loetelu.

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;


/**
* Trükitakse välja eesnimed. Tükid koondatakse puhvris kokku tervikuks.
*/
public class EesnimedeLugeja extends DefaultHandler{
  boolean kasEesnimi=false;
  StringBuffer puhver;
  public void startElement(String nimeruum, String kohalik,
            String element, Attributes at){
     if(element.equals("eesnimi")){
       kasEesnimi=true;
       puhver=new StringBuffer();
     }
  }

  public void endElement(String nimeruum, String kohalik,
            String element){
     if(element.equals("eesnimi")){
       kasEesnimi=false;
       System.out.println(puhver);
     }
  }

  public void characters(char[] tahed, int algus, int pikkus){
   if(kasEesnimi){
       puhver.append(new String(tahed, algus, pikkus));
   }
  }

  public static void main(String argumendid[]) throws Exception{
    XMLReader lappaja=  SAXParserFactory.newInstance().newSAXParser().getXMLReader();
    lappaja.setContentHandler(new EesnimedeLugeja());
    lappaja.parse("inimesed.xml");
  }
}


/*
E:\kasutaja\jaagup\xml>java EesnimedeLugeja
Juku
Juku
Kalle
Mari
Oskar

*/
```

Üheks levinud SAXi kasutamise kohaks on XML-faili osade lugemine andmebaasi. XML-faili lapates kogutakse muutujatesse kokku sobivad väärtused. Kui terve rea jagu koos, siis õnnestub tulemus ühe insert-lause abil andmebaasi tabeli reaks kirjutada.