Tallinna Ülikool
Informaatika Instituut

XML rakendused

Jaagup Kippar

Tallinn 2009

Sisukord

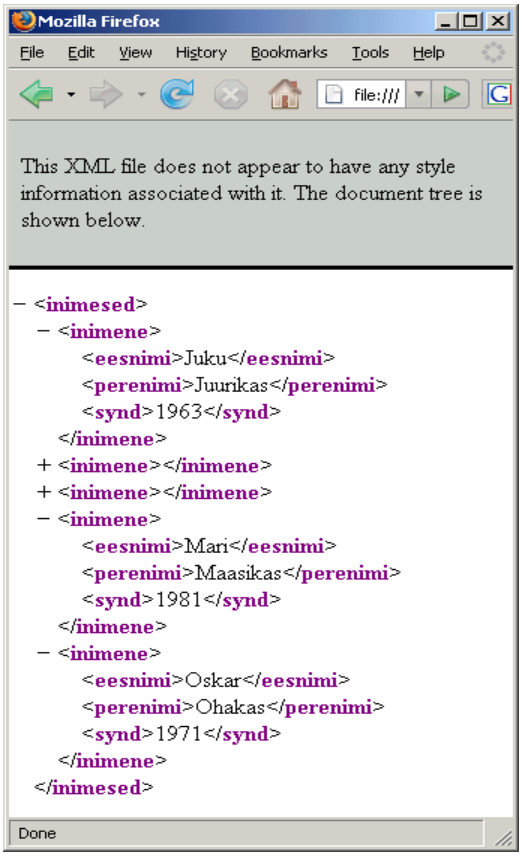
XML

Selle nime all tuntakse tekstifailivormingut. Esmamulje järgi paistavad elemendid olema kirjutatud nagu veebikoostajatele tuntud HTML-keeles. 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 mitmesuguseid 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.

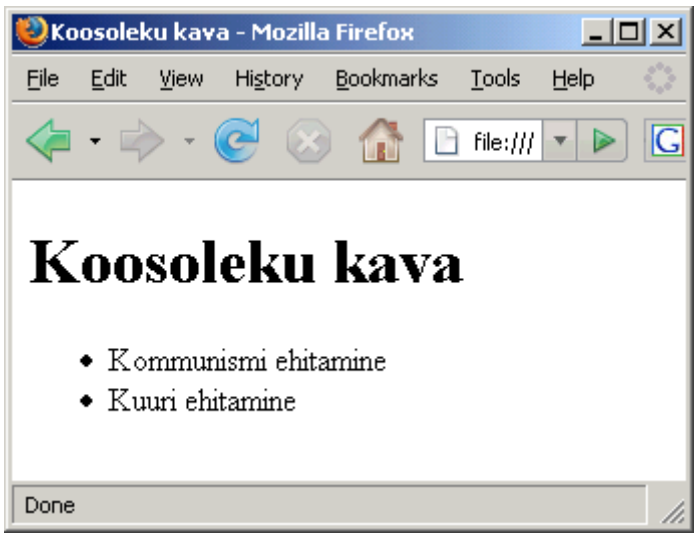
Masinaga loetavaks muudetakse XML-failid range süntaksi abil. Elementide nimed kirjutatakse märkide < ja > vahele. Iga algav element peab ka lõppema. Kui elemendil puudub sisu, paigutatakse algus ja lõpp kokku. Näiteks reavahetust tähistatakse XHTMLis
, kus siis kaldkriips näitab, et elemendil eraldi 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.

Siin juures eraldi XML-faili näide, mida edaspidi alusena kasutama hakatakse. Tegemist on lihtsa inimeste loeteluga, iga isiku kohta kirjas eesnimi, perekonnanimi ja sünniaasta. Üleval on XML-andmetele kohustuslik versioonideklaratsioon. Välimine 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
<?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>
```




```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
    <title>Koosoleku kava</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
 </head>
 <body>
  <h1>Koosoleku kava</h1>

  <ul>
    <li>Kommunismi ehitamine</li>
    <li>Kuuri ehitamine</li>
  </ul>

 </body>
</html>
```



Ülesanded XMLiga tutvumiseks.

* Ava näitena olev inimeste fail veebilehitsejas.
  Veendu, et rakendus sai failist aru, st. et elemente on võimalik kinni ja lahti klõbistada
* Muuda üks Juku Juhaniks. Kontrolli, et endiselt brauseris avaneks.
* Eemalda üks inimene.
* Lisa üks inimene.
* Eemalda üks lõpetav </perenimi>. Tutvu veateatega. Taasta endine olukord.
* Eemalda üks kaldkriips. Tutvu veateatega. Taasta endine olukord.

* Katseta täpitähtedega nimesid. Proovi tekst salvestada UTF-8na.
* Kui redaktor ei toeta UTF-8t, siis lisa faili päisesse kooditabel, nt. <?xml version="1.0" encoding="iso-8859-1"?>

* Tutvu XHTML-vormingus koosolekut kirjeldava failiga, mille nimeks koosolek.html
* Lase leht läbi validaatorist aadressil http://validator.w3.org/
* Lisa üks päevakorrapunkt, kontrolli valideeruvust.
* Lisa div-elemendi sisse selgitav tekst.
* Lisa
 abil reavahetus. Kontrolli valideeruvust. Tutvu veateatega juhul, kui kaldkriips on puudu.
* Lisa lehele pilt lihtsa kuuri joonisega. Vaata lehte, kontrolli valideeruvust.

* Koosta ise XML-andmefail, püüdes kirja panna arvuti juurde kuuluvad komponendid ja nende parameetrid.

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 andmetega 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 kokkuvõttes 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 Java abil

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 XMLi vahendid standardkomplekti sisse paigutatud ning piisab vaid sobivate pakettide impordist.

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
```

Käivitamine Visual Studio abil

Kui parajasti tuttav või käepärast .NET ja ASP.NET veebiserver näiteks Visual Studio Web Developeri näol, siis ka seal saab suhteliselt vähese vaevaga panna XSLi XMList andmeid võtma ja välja näitama. Töö alustamiseks pärast käivitamist on soovitav luua uus veebilehestik (File -> New Web Site). Edasi tasuks sinna panna XML-kujuline andmefail ning XSLT-muundefail. Ning et tulemust on mugav vaadata genereeritaval veebilehel, siis on vaja ka veebivormi (.aspx-laiendiga). XML- ja XSL-fail võivad eelmisega sarnased olla. Veebivormi sobivale kohale tasuks panna aga asp:Xml-element, mis määrab, kust andmed võtta ning millise muundamise teel nad lehele paigutada. Näiteks

```
      <asp:Xml ID="xml1" runat="server" DocumentSource="~/inimesed1.xml"
          TransformSource="~/inimesed1a.xslt" />
```

Ja kogu aspx-fail siis kujul näiteks

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Andmete leht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <asp:Xml ID="xml1" runat="server" DocumentSource="~/inimesed1.xml"
          TransformSource="~/inimesed1a.xslt" />
    </div>
    </form>
</body>
</html>
```

Kui nüüd kõik komponendid üles leitakse, siis võib lehe käivitamise peale ilusti tulemust näha, kus kaks nime veebilehele sobivatesse kohtadesse paigutatud.

Ülesandeid

* Pane näited tööle
* Küsi kolme esimese inimese perekonnanimi.
* Lisa andmefaili inimestele sugu. Küsi ka see.
* Küsi eelviimase inimese perekonnanimi.

Tekstifunktsioonid XSLis

Asukoha järgi saab andmeid XSLi abil XMList ilusti välja võtta. Et kui soovitakse küsida kolmanda inimese eesnimi, siis kirjutatakse /inimesed/inimene[3]/eesnimi sobivasse kohta, näiteks xsl:value-of elemendi select-parameetriks.

Kui soovitakse aga nimest vaid esitähte, siis võib kirjutada

```
        Esitäht:
  <xsl:value-of select="substring(/inimesed/inimene[1]/eesnimi, 1, 1)" />
```

Funktsioonile substring tuleb ette anda otsitav tekst ning kaks arvu: mitmendast tähest alates ning mitu tähte väljastada.

Funktsiooni enese kuju

```
substring(s1, start, nr?)
```

Järgnevalt siis mõned levinumad sõnefunktsioonid:

```
concat(s1, s2, s3*)
```

Parameetritena antud tekstid liidetakse. Elementide arv ei ole piiratud. Üldjuhul saab hakkama ka ilma seda funktsioonid kasutamata, pannes lihtsalt järjestikku vajalikke xsl:value-of elemente ning tekste. Aga mõnikord on concati abil andmeid lühem reastada. Näiteks initsiaalide tarbeks

```
          <xsl:value-of select="
           concat(substring(/inimesed/inimene[1]/eesnimi,  1, 1),'.',
                  substring(/inimesed/inimene[1]/perenimi, 1, 1), '.')" />
```

Ehk siis concati sees komadega eraldatult eesnime esitäht, punkt (tekstina esitamiseks ülakomade vahel), siis perekonnanime esitäht ning sellele järgnev punkt.

```
        <xsl:value-of select="substring-before('muna ja kana', 'ja')" />
```

annab tulemuseks "muna". Mõlema funktsiooni kujud näevad välja järgnevalt, teine lihtsalt annab välja eraldajale järgneva osa.

```
substring-before(s1, s2)
substring-after(s1, s2)
```

Pea igas programmeerimiskeeles on käsklus teksti pikkuse küsimiseks. Nii ka siin.

```
string-length(s1)
```

ehk siis näitena

```
<xsl:value-of select="string-length('Mati')" />
```

annab tulemuseks arvu 4.

`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'

Ülesandeid

* Koosta XML-andmefail, kus on kirjas auto registrinumber ning omaniku perekonnanimi
* Trüki välja auto registrinumber
* Trüki välja auto registrinumbri numbrite osa
* Trüki välja auto registrinumbri tähtede osa
* Trüki välja inimese perekonnanime esimene täht
* Trüki välja inimese perekonnanime viimane täht

Loendamine

Vastuse küsimusele, mitu elementi vastab soovitud otsingule, annab funktsioon count. Näiteks inimeste arvu saab kokku järgnevalt:

```
        Inimeste arv: <xsl:value-of select="count(/inimesed/inimene)" />
```

Tahtes kokku loendada kõiki inimesi, kelle eesnimi on Juku, tuleb vastav piirang kirjutada inimese taha kandiliste sulgude sisse.

```
        Jukude arv: <xsl:value-of  select="
             count(/inimesed/inimene[eesnimi='Juku'])" />
```

Saab võrrelda ka suurem ja väikesem olemist. Vastavad võrdlused aga on viisakas kirjutada XMLi jaoks sobivate märgikombinatsioonidena. "Väiksem kui" oleks siis kujul < (less than) ning "suurem kui" kujul > (greater than).

```
        Enne Moskva olümpiat sündinud: <xsl:value-of
select="count(/inimesed/inimene[synd &lt; 1980])" />
```

Ülesandeid

* Koosta/otsi XML-andmefail, kus on kirjas auto registrinumber ning omaniku perekonnanimi
* Leia, mitme inimese perekonnanimi on Kaalikas
* Leia, mitme inimese perekonnanimi algab M-tähega

* Leia, mitme auto registrimärgi numbritest viimane on 2
* Leia, mitme auto registrimärgi numbritest viimane on 1 või 2

Tingimused

Programmeerimise juures on võimalik tingimusele vastavalt midagi teha või tegemata jätta. Nii ka siin. Tavalise valiku tarvis on käsklus xsl:if. Näide:

```
<xsl:if test="starts-with(/inimesed/inimene[1]/eesnimi, 'J')">
  Esimene nimi algab jotiga
</xsl:if>
```

Kui tingimus vastab tõele, siis sealne tekst trükitakse, muidu mitte. Käsu kuju siis

`starts-with(s1, s2)`

ning selgitus

Kontrollitakse, kas esimesena antud tekst algab teisena antud tekstiga.

Mõnevõrra sarnane funktsioon on

`contains(s1, s2)`

Võrreldes eelmisega ei pruugita alustada algusest, vaid otsitakse lõigu leidmist kogu teksti ulatuses.

Tõeväärtuse saab vastuspidiseks pöörata funktsiooniga not(), eraldi else-lauset ei ole.

```
<xsl:if test="not(starts-with(/inimesed/inimene[1]/eesnimi, 'A'))">
    Esimene nimi ei alga a-ga
</xsl:if>
```

Tingimuseks sobivad ka igasugu võrdlused, mille vastuseks on "jah" või "ei". Näiteks, et kas teksti pikkus on väiksem kui viis sümbolit. Märk "on väiksem kui" ehk "<" kirjutatakse kujul < (less than), sest nii vastab see XMLi süntaksile (millele XSL leht peab töötamiseks vastama. Näiteks

```
<xsl:if test="string-length(/inimesed/inimene[1]/eesnimi) &lt; 5">
   Lühike nimi
</xsl:if>
```

Ülesandeid

* Koosta/leia XML-andmefail, kus on kirjas auto registrinumber ning omaniku perekonnanimi
* Kui registrimärgi viimane number on 5, siis teata, et ülevaatuse kuu on juuli
* Teata iga registrinumbri puhul, millises kuus auto ülevaatusele peab minema.
* Kui perekonnanimes sisaldub täht x, siis teata, et tegemist on võõrnimega.
* Muul juhul teata, et tegemist pole võõrnimega.

Kordused

Kui samatüübilisi andmeid on palju, siis tulevad appi kordused. Nii ka XSLi juures. Tsükli loomiseks sobib käsklus xsl:for-each, select-parameetrina antakse ette elemendid mille kontekstis ploki sisu korrata. Et tööd alustatakse juurelemendile vastavast mallist

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

ning sealt seest küsitakse kõiki elemente, mis vastavad mustrile inimesed/inimene, siis saadakse kätte algsest failist kõik viis inimest, kes on pöördutavad kujul /inimesed/inimene.

Tsükli sees on igal korral jooksvaks elemendiks juba konkreetne inimene. Esimesel korral siis näiteks

```
    <inimene>
        <eesnimi>Juku</eesnimi>
        <perenimi>Juurikas</perenimi>
        <synd>1963</synd>
    </inimene>
```

Seal saab juba otse pöörduda alamelementide poole. Ehk siis

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

annab vastuseks Juku. Ning iga järgmise ringi juures juba järgmise inimese eesnime.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <xsl:for-each select="inimesed/inimene">
            <xsl:value-of select="eesnimi" />
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Käivitades XSL-lehe koos andmefailiga saame tulemuse arvatavasti kujul

```
JukuJukuKalleMariOskar
```

Tahtes nimede vahele tühikuid saada, sobib abielemendina xsl:text, mis lubab elemendi alguse ja lõpu vahele kirjutatud teksti otse väljundisse saata.

```
        <xsl:for-each select="inimesed/inimene">
            <xsl:value-of select="eesnimi" />
            <xsl:text> </xsl:text>
        </xsl:for-each>
```

Annab siis tulemuseks

```
  Juku Juku Kalle Mari Oskar
```

Kui aga juba HTMLi sisse andmete kirjutamiseks läheb, siis pole otsesed tühikud ja reavahetused kuigi tähtsad - loeb lihtsalt nende olemasolu või puudumine, mitte kogus. Kujundamiseks on aga seal juba omad käsud. Näiteks loetelu jaoks tavapärane komplekt <ul> kogu loetelu tähistamiseks ning <li> üksiku elemendi tarbeks. Need saab rahumeeli XSLi faili sisse panna.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <ul>
        <xsl:for-each select="inimesed/inimene">
            <li><xsl:value-of select="eesnimi" /></li>
        </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

Tulemusena võib ilusti vormindatud tulemust imetleda.

```
      <ul>
  <li>Juku</li>
  <li>Juku</li>
  <li>Kalle</li>
  <li>Mari</li>
  <li>Oskar</li>
</ul>
```

Sama tulemuse saab programmide puhul ikka mitmel moel kätte. Eelnenud näites öeldi xsl:for-eachi juures, et läbi tuleb käia kõik elemendid nimega inimene ning xsl:value-ofi juures määrati, et näha soovitakse inimese seest eesnime.

Järgnevas näites saadakse sama tulemus nõnda, et juba xsl:for-eachi juures käiakse läbi kõikide inimeste eesnimed. Ning xsl:value-ofi juures pole muud vaja öelda, kui et tuleb kuvada jooksva elemendi (mida tähistatakse punktiga) väärtus.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <ul>
        <xsl:for-each select="inimesed/inimene/eesnimi">
            <li><xsl:value-of select="." /></li>
        </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

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;
```

Plokifunktsioone

Kui tsükli abil plokis liikuda, siis tuleb vahel kasuks teada, mitmendal ringil parajasti ollakse ning mitu ringi kokku on. Vastavad abiks olevad funktsioonid on järgnevad:

```
last()        viimase järjekorranumber
position()    jooksva järjekorranumber
```

Ehk siis järgnevas näites trükitakse iga eesnime ette tema järjekorranumber.

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

Käsklus position() annab järjekorranumbri, concat seob ühte ritta kokku selle numbri, sidekriipsu ning vastava inimese eesnime, kelle juures ollakse.

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

```
        <ul>
        <xsl:for-each select="inimesed/inimene">
            <li><xsl:value-of select="concat(position(),' - ',eesnimi)" /></li>
        </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

Väljund paistab välja järgnev:

```
      <ul>

  <li>1 - Juku</li>
  <li>2 - Juku</li>
  <li>3 - Kalle</li>
  <li>4 - Mari</li>
  <li>5 - Oskar</li>
</ul>
```

Kui tahta juurde määrata, et mitmes nimi mitmest, siis tuleb lisaks viimase järjekorranumbrit näitav last().

```
        <ul>
        <xsl:for-each select="inimesed/inimene">
            <li><xsl:value-of select="concat(position(),'/',last(),' -
',eesnimi)" /></li>
        </xsl:for-each>
        </ul>
```

Ning tulemus veebilehel järgmine:

* 1/5 - Juku
* 2/5 - Juku
* 3/5 - Kalle
* 4/5 - Mari
* 5/5 - Oskar

Ülesandeid

* Koosta/leia XML-andmefail, kus on kirjas autode registrinumbrid ning omaniku perekonnanimed
* Trüki välja kõik perekonnanimed
* Lisa iga perekonnanime ette tema järjekorranumber
* Need read, kus registrinumbri viimane number lõppeb ühe või kahega trüki rasvaselt.

Kujundus korduse sees

Lisaks nimede/andmete ettelugemisele on küllalt sageli vajalik/kasulik neid ka vastavalt juurdekuuluvatele omadustele kujundada, et sobivaid ridu oleks kergem üles leida. Tingimuslauseid, arvutusi ja muid käsklusi saab rahumeeli kasutada ka korduse või korduste sees. Järgnevalt värvitakse hiljem kui 1980ndal aastal sündinud inimeste eesnimed roheliseks. Tehakse eesnimele ümber span-plokk ning sealse stiiliga määratakse nime värv. Ja kui tahta ka ülejäänud nimesid näidata, ainult et tavaliste mustadena, siis üks moodus on loodud plokk lihtsalt kopeerida, tingimusele ümberpöörav not-käsklus ümber panna ning nimele sobiv värv määrata.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <ul>
        <xsl:for-each select="inimesed/inimene">
            <li>
                <xsl:if test="synd &gt; 1980">
                    <span style="color:green">
                        <xsl:value-of select="eesnimi" />
                    </span>
                </xsl:if>
                <xsl:if test="not(synd &gt; 1980)">
                    <span style="color:black">
                        <xsl:value-of select="eesnimi" />
                    </span>
                </xsl:if>
            </li>
        </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

Tulemusena siis loetelu, kus iga nime juures märgitud värv.

```
      <ul>
  <li>
    <span style="color:black">Juku</span>
  </li>
  <li>
    <span style="color:black">Juku</span>
  </li>
  <li>
    <span style="color:black">Kalle</span>

  </li>
  <li>
    <span style="color:green">Mari</span>
  </li>
  <li>
    <span style="color:black">Oskar</span>
  </li>
</ul>
```

Sama saab aga ka märgatavalt lühemalt kirja panna. Iseenesest pole ju nime värvimiseks talle vaja eraldi span-elementi ümber panna. Piisab, kui nime näitava elemendi - praegusel juhul li - stiilis on sobiv värv määratud. Käsklus xsl:attribute võimaldabki juba alanud elemendi sisse atribuute lisada. Järgnevas näites siis näha, et noorematele kui 1980ndal sündinutele pannakse

loeteluelemendile rohelise värvi atribuut külge.

```
<ul>
<xsl:for-each select="inimesed/inimene">
    <li>
        <xsl:if test="synd &gt; 1980">
            <xsl:attribute name="style">color:green</xsl:attribute>
        </xsl:if>
        <xsl:value-of select="eesnimi" />
    </li>
</xsl:for-each>
</ul>
```

Ning pärast käivitamist on see ilusti näha ka tulemuse juures.

```
<ul>
  <li>Juku</li>
  <li>Juku</li>
  <li>Kalle</li>
  <li style="color:green">Mari</li>
  <li>Oskar</li>
</ul>
```

Pikemate ridade puhul on heaks tavaks mõned read ära värvida, et oleks parem andmeid lugeda ja näpuga järge ajada. Reanumbri saame kätte käsuga position(). Nüüd, kui on tahtmine kindla arvu ridade tagant erivärviline rida luua, siis on heaks abivahendiks jagamise jääk, modulus, ehk siin tehe nimega mod. Näitab, et kui vasakul pool olev kogus jagada paremal pool olevaks arvuks võrdseteks osadeks, mitu siis jääb üle. Kui näiteks kaksteist õuna jagada võrdselt neljale lapsele, siis saab igaüks kolm õuna, ühtegi üle ei jää ehk jääk on null. Kui aga on õunu kolmteist, siis üks õun jääb üle ehk jääk on üks. Neljateistkümne puhul kaks, viieteistkümne puhul kolm ning kuusteist jagub taas neljaga, siis jääk jälle null. Nii et kui tahaks iga neljanda rea teistsuguseks teha, siis piisaks kontrollida, et kas jääk jagamisel neljaga on mingi kindel arv nendest võimalikest jääkidest. Ühe puhul oleks värvitud kohe esimene rida, kahe puhul teine, kolme puhul kolmas. Ning kõigepealt neljanda värvimiseks piisab kontrollimast, kas jääk on null. Iga teise rea värvimiseks näiteks sobib järgnev kood:

```
<li>
    <xsl:if test="position() mod 2 =0">
        <xsl:attribute name="style">
            background-color:lightgray</xsl:attribute>
    </xsl:if>
    <xsl:value-of select="eesnimi" />
</li>
```

Ning tulemus veebilehe koodis on selline.

```
<ul>
  <li>Juku</li>
  <li style="background-color:lightgray">Juku</li>
  <li>Kalle</li>
  <li style="background-color:lightgray">Mari</li>
```

```
  <li>Oskar</li>
</ul>
```

Ülesandeid

* Koosta/leia XML-andmefail, kus on kirjas autode registrinumbrid ning omaniku perekonnanimed
* Väljasta andmed tabelina: registrinumbrid ühes ning perekonnanimed teises tulbas.
* Värvi ridade taustad üle ühe halliks.
* Värvi taustad nõnda, et järjestikku oleksid värvitu, hall ja kollane.

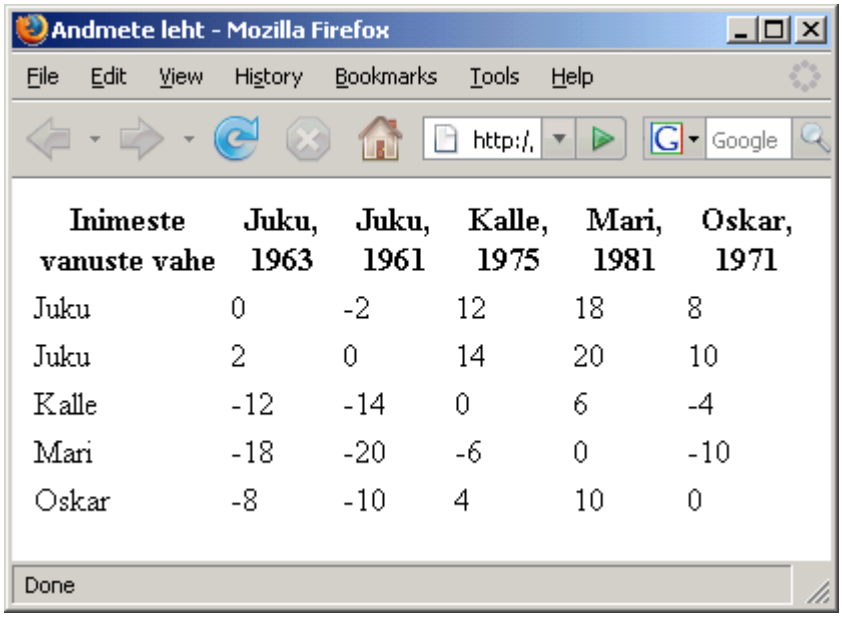
Mitu kordust samade andmetega

Tavaliselt käiakse andmed ühe korra läbi, näidatakse mis tarvis ja kogu lugu. Aga vahel võib olla põhjust ka mitu korda samu andmeid läbi käia ning elementide vahel seoseid leida ja välja märkida. Järgnevas näites luuakse tabel, kus nii veergudes kui ridades on samad inimesed. Veergude ja ridade ristumiskohal aga nende sünniaastate vahe - siis kohe näha, kes kui palju kellest vanem on. Kui kordus määratakse juurelemendist alates, siis on selline sättimine lihtne - piisab kaks for-each tsüklit üksteise sisse panna ning mõlemale määrata, et tuleb läbi käia elemendid mustriga /inimesed/inimene. Veidi mõtlemist vajab aastate vahe arvutus, sest sisemise inimese sünniaasta varjestab välimise oma ära. Aitab aga, kui välimine element jätta eraldi muutujasse meelde. Siis saab selle andmed ka sisemises plokis kätte ning võib nendega rahus toimetada.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <table>
            <tr>
                <th>Inimeste vanuste vahe</th>
              <xsl:for-each select="/inimesed/inimene">
                  <th>
                      <xsl:value-of select="
                        concat(eesnimi, ', ', synd)"></xsl:value-of>
                  </th>
              </xsl:for-each>
            </tr>
            <xsl:for-each select="/inimesed/inimene">
                <tr>
                    <xsl:variable name="v2limine" select="." />
                    <td>
                        <xsl:value-of select="eesnimi" />
                    </td>
                 <xsl:for-each select="/inimesed/inimene">
                     <td>
                         <xsl:value-of select="
                           number(synd)-number($v2limine/synd)" />
                     </td>
```

```
                </xsl:for-each>
               </tr>
            </xsl:for-each>
         </table>
    </xsl:template>
</xsl:stylesheet>
```

Veebilehel väljund näha järgmine:



Inimeste vanuste vahe	Juku, 1963	Juku, 1961	Kalle, 1975	Mari, 1981	Oskar, 1971
Juku	0	-2	12	18	8
Juku	2	0	14	20	10
Kalle	-12	-14	0	6	-4
Mari	-18	-20	-6	0	-10
Oskar	-8	-10	4	10	0

Ülesandeid

* Pane näide käima
* Vähem kui viieaastase vanusevahega lahtrid näita teise värviga

* Kirjuta lahtritesse vastavate inimeste perekonnanimed. Nt. Kalle ja Mari ristumiskohale tuleb "Kaalikas ja Maasikas"
* Võrreldes eelmisega jäta tühjaks need lahtrid, kus inimene kohtub iseenesega (kõige viisakam kindlaks teha positsiooni järgi).

XSL eri struktuuriga andmefailidele

Eelnevad näited olid loodud kindlate elemendinimedega andmefaili seest väärtuste kogumiseks ja esitamiseks. Vahel aga võivad andmefailid olla küll mõneti sarnased, aga sugugi mitte samade elemendinimedega ja -kogustega. Tüüpiliseks näiteks on mitmed algselt tabeli kujul olevad andmed, mis XML-i kujule viidud ning mida hiljem sealt soov taas HTML- või mõne muu tabeli kujul esitada. Sellisel juhul osutub kasulikuks käsklus name, mille abil saab etteantud ploki nime küsida. Samuti on kasulik metamärk * (tärn), mis tähistab kõiki vastavas kohas asuvaid alamelemente. Või kui küsida sellise alamelementide ploki nime, siis antakse neist esimese nimi. Siin on allikaks sama inimeste fail. Käsklus

name(*) annab vastuseks "inimesed", kuna ollakse juure juures. Käsklus name(*/*) väljastab "inimene", sest see on eelneva alamelement. Tegelikult mustrile vastavad kõik elemendid "inimene", aga kuna küsitakse nime ainult ühe kohta, siis väljastatakse esimese oma. Tahtes läbi käia esimese inimese kõik alamelemendid, sobib tsükliks

```
<xsl:for-each select="*/*[1]/*">
```

Lahtiseletatult siis esimene tärn tähendab juures asuvat elementi "inimesed". Järgmine *[1] on esimene inimene. Ning viimase tärni alt tulevad välja esimese inimese eesnimi, perenimi ja synd. Tahtes saada elementide nimesid ja mitte väärtusi, siis aitab taas käsklus name, sedakorda kujul

```
<xsl:value-of select="name(.)" />
```

kus trükitakse välja parasjagu jooksva elemendi nimi. Kogu XSLi kood:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <h1>
            Juurelement: <xsl:value-of select="name(*)" />
        </h1>
        <h2>
            Esimene alamelement: <xsl:value-of select="name(*/*)" />
        </h2>
        <h3>Alamelemendi järglased: </h3>
        <ul>
        <xsl:for-each select="*/*[1]/*">
            <li>
                <xsl:value-of select="name(.)" />
            </li>
        </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

Ning tulemuseks saadi siis järgnev HTML, kus võib veenduda, et kõik elemendid õigesti kätte saadi.

```
      <h1>Juurelement: inimesed</h1>
<h2>Esimene alamelement: inimene</h2>
<h3>Alamelemendi järglased: </h3>
<ul>
  <li>eesnimi</li>
  <li>perenimi</li>
  <li>synd</li>
</ul>
```

Edasi tasub mõelda, et millisel kujul on soov andmeid esitada. Kui veebilehel ja tabelina, siis on tabeli HTML-käsud igati asjakohased. Eelneva loetelu saab vormistada tabeli tulpade pealkirjadena.

```
        <h1><xsl:value-of select="name(*)" /></h1>
        <table>
         <tr>
          <xsl:for-each select="*/*[1]/*">
            <th>
                <xsl:value-of select="name(.)" />
            </th>
          </xsl:for-each>
         </tr>
        </table>
```

Ning väljund veebilehel on järgnev.

```
      <h1>inimesed</h1>
<table>
  <tr>
    <th>eesnimi</th>
    <th>perenimi</th>
    <th>synd</th>
  </tr>
</table>
```

Soovides lisaks pealkirjadele ka tegelikke andmeid sisse panna, tuleb lihtsalt kaks tsüklit juurde panna. Pealkirjad ja lahtrite sisud tuleb eraldi tsüklitesse paigutada lihtsalt HTMLi loogka tõttu, kus tuleb vasakult paremale oma andmeid väljastada. Ehkki vajaduse ja tahtmise korral oleks võimalik ehk ka pealkirjaplokk ja andmete läbikäigu plokk ühendada ning lihtsalt if-iga kontrollida, kas ollakse andmetega esimese rea juures. Kui jah, siis oleks paras aeg lisaks muule ka pealkirjalahtrid kirjutada. Siin näites aga pealkirjaosa ja andmete osa eraldi kirjutatud. Andmete tsüklis siis kõigepealt

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

käib läbi kõik elemendid "inimene". Sealt seest

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

võtab kõik, mis ühe inimese seest võtta on. Ning viimaks

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

annab selle elemendi väärtuse, kus sees parajasti oma läbikäiguga ollakse.

```
        <h1><xsl:value-of select="name(*)" /></h1>
        <table>
         <tr>
          <xsl:for-each select="*/*[1]/*">
            <th>
                <xsl:value-of select="name(.)" />
            </th>
          </xsl:for-each>
         </tr>
            <xsl:for-each select="*/*">
                <tr>
                    <xsl:for-each select="*">
                        <td>
                            <xsl:value-of select="." />
                        </td>
```

```
                    </xsl:for-each>
                </tr>
            </xsl:for-each>
        </table>
```

Tulemuseks on viisakas HTML-i tabel.

```
      <h1>inimesed</h1>

<table>
  <tr>
    <th>eesnimi</th>
    <th>perenimi</th>
    <th>synd</th>
  </tr>
  <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>
```

Ülesandeid

* Tee näited läbi. Lisa inimesele tunnuseks mass. Veendu, et andmed väljastatakse ka siis tabelina.
* Muuda näidet nõnda, et tabeli asemel genereeritaks loetelu, kus on näha iga elemendi vaid kaks esimest väärtust.
  Näiteks esimese puhul Juku Juurikas
* Muuda näidet nõnda, et tabeli asemel genereeritaks komadega eraldatud loetelu. Nt
  Juku,Juurikas,1963
* Muuda näidet nõnda, et tabeli asemel genereeritaks laused, mille abil

vastavaid andmeid SQL-baasi sisestada. Inimeste puhul näeks siis lause välja:
INSERT INTO inimesed (eesnimi, perenimi, synd) VALUES ('Juku', 'Juurikas', '1963');
* Koosta/leia XML-andmefail, kus on kirjas autode registrinumbrid ning omaniku perekonnanimed
* Veendu, et eelnevalt koostatud XSL-lehed töötavad ka uue andmefailiga.

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;
```

ASP.NET vahenditega sama XSL-lehte käivitades on põhjust veebilehele luua kohad parameetrite väärtuste sisestamiseks. Siinses näites kasutatakse selle tarbeks tekstivälju. Pöörates tähelepanu lehekülje päisele näeb, et sedakorda on põhjust eraldi koodifaili loomiseks, millest siis veebileht päritakse.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="parameetrid.aspx.cs"
Inherits="parameetrid" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Parameetritega leht</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <pre>
        <asp:Xml ID="xml1" runat="server" DocumentSource="~/inimesed1.xml"
TransformSource="~/inimesed4a.xslt" />
       </pre>
        <br />
        Otsitav tekst: <asp:TextBox ID="kast1" runat="server" /><br />
        Miinimumpikkus: <asp:TextBox ID="kast2" runat="server" /><br />
        <asp:Button runat="server" text="Sisesta"  />
    </div>
    </form>
</body>
</html>
```

Eraldi koodifailis, lehe laadimisel käivituvas meetodis nimega Page_Load on kirjas, et millised parameetrite väärtused tuleb XSL-failile edasi anda. Eelnevast XSL-failis on näha, et selle algusosas on defineeritud kaks parameetrit. Parameeter nimega otsing vaikeväärtusega "ar" ning parameeter pikkus väärtusega 5. Siinses näites kirjutatakse otsinguparameeter alati üle tekstiväljast tuleva väärtusega. Pikkus asendatakse aga vaid juhul, kui selle jaoks mõeldud tekstiväljas nimega kast2 oli väärtus, mida on võimalik arvuks teisendada. XsltArgumentList-tüüpi muutujasse kogutakse parameetrid ja nende väärtused kokku ning siis pannakse vastav parameetrite loetelu transformatsiooni eest hoolitseva elemendi külge.

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Xml.Xsl;

public partial class parameetrid : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        XsltArgumentList p = new XsltArgumentList();
        p.AddParam("otsing", "", kast1.Text);
        int abi;
        if(int.TryParse(kast2.Text, out abi)){
            p.AddParam("pikkus", "", kast2.Text);
        }
        xml1.TransformArgumentList = p;
    }
}
```

Nulemusena võib siis näha sisestatud andmetele vastavat tulemust veebilehel. Element pre aitas tekstina tulnud andmed viisakalt eraldi ridadele joondada.

```
        Nimed, mis sisaldavad kombinatsiooni M:
        Mari Maasikas;


        Nimed pikkusega 2 ja rohkem:
        Juku Juurikas varuks 2;
        Juku Kaalikas varuks 2;
        Kalle Kaalikas varuks 3;
        Mari Maasikas varuks 2;
        Oskar Ohakas varuks 3;



Otsitav tekst:M
Miinimumpikkus:2
```

Ülesandeid

* Koosta tervitav XSL-leht, mille pealkiri antakse ette parameetriga
* Anna parameetrina ette praeguse aasta number. Väljasta iga inimese vanus selle aasta lõpuks.
* Lase kasutajal tekstiväljast anda ette vähim ja suurim sünniaasta, mille vahel olevate inimeste andmeid näidatakse.

Mallid, alamprogrammid

Enamikes enesest lugu pidavates programmeerimiskeeltes on eraldiseisvate ning

korduvalt ettevõetavate toimingute tarbeks olemas alamprogrammid või midagi sarnast. XSLis on selleks tarbeks mallid. Kirjeldatakse ära, kuidas mingeid andmeid väljundis näidata ning hiljem saab selle malli sobivas kohas välja kutsuda. Järgnevas näiteks on omaette malliks tehtud saatja aadress. Et kui seda vaja kuhugi kirjale lisada, siis piisab vaid vastavast väljakutsest.
Defineerimise kohal siis

```
<xsl:template name="saatjaaadress">
```

ning väljakutsel

```
<xsl:call-template name="saatjaaadress" />
```

Kogu kood tervikuna:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template name="saatjaaadress">
      Tallinna Ülikooli Informaatika Instituut, Narva maantee 25-419, 10120,
Tallinn
    </xsl:template>

    <xsl:template match="/">
       Lugupeetud vilistlane, kutsume teid taas vana kooli vaatama!
       <br />
       <xsl:call-template name="saatjaaadress" />
    </xsl:template>
</xsl:stylesheet>
```

Ja nagu koodi lugedes aimata võib, tulemuseks on teade koos sinna lõppu lisatud saatja aadressiga.

```
Lugupeetud vilistlane, kutsume teid taas vana kooli vaatama!
Tallinna Ülikooli Informaatika Instituut, Narva maantee 25-419, 10120, Tallinn
```

Et malli töö tulemust vastavalt andmetele muuta saaks, selleks tuleb ette anda parameetrid. Malli enese algusesse tuleb kirjutada, et millise nimega parameeter on.

```
<xsl:param name="enimi" />
```

Koodis parameetri väärtuse kasutamiseks tuleb sinna dollarimärk ette panna.

```
Tere, <xsl:value-of select="$enimi" />
```

Väljakutsel saab temale ette anda tavalise väärtuse. Et Siim on etteantud tekstikonstant, teda ei võeta XML-faili seest, selleks on nimele ülakomad ümber pandud.

```
<xsl:call-template name="tervitus">
    <xsl:with-param name="enimi" select="'Siim'" />
</xsl:call-template>
```

XSLi kood tervikuna

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template name="tervitus">
        <xsl:param name="enimi" />
        Tere, <xsl:value-of select="$enimi" />
    </xsl:template>

    <xsl:template match="/">
        <xsl:call-template name="tervitus">
            <xsl:with-param name="enimi" select="'Siim'" />
        </xsl:call-template>
    </xsl:template>
</xsl:stylesheet>
```

Ning tekkinud väljund:

```
Tere, Siim
```

Kui nimi võetakse XMList, siis tuleb määrata asukoht, kust see leida. Praegusel juhul antakse alamprogrammile parameetrina kaasa esimese inimese eesnimi.

```
    <xsl:template match="/">
        <xsl:call-template name="tervitus">
            <xsl:with-param name="enimi" select="inimesed/inimene[1]/eesnimi" />
        </xsl:call-template>
    </xsl:template>
```

Käivitamisel tervitatakse teda rõõmsasti.

```
Tere, Juku
```

Malli parameetriks ei pruugi olla üksiks väärtus. Selleks võib olla ka etteantud kohast algav andmepuu (nt. terve inimene kõikide tema juurde kuuluvate andmetega) või siis andmekogum (massiiv). Järgnevas näites pruugitaksegi viimast varianti. Loetelu saamiseks käiakse xsl:for-each tsükliga läbi kõik etteantud andmestiku elemendid. Ning malli väljakutsel olev inimesed/inimene/eesnimi annabki kõikide vastavas XML-failis olevate inimeste eesnimed parameetrina kaasa.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template name="loetelu">
        <xsl:param name="andmestik" />
        <ul>
            <xsl:for-each select="$andmestik">
                <li>
                    <xsl:value-of select="." />
                </li>
            </xsl:for-each>
        </ul>
    </xsl:template>

    <xsl:template match="/">
        <h1>Inimeste eesnimede loetelu</h1>
```

```
        <xsl:call-template name="loetelu">
            <xsl:with-param name="andmestik" select="inimesed/inimene/eesnimi"/>
        </xsl:call-template>
    </xsl:template>
</xsl:stylesheet>
```

Tulemuseks on viisakas loetelu.

```
      <h1>Inimeste eesnimede loetelu</h1>

<ul>
  <li>Juku</li>
  <li>Juku</li>
  <li>Kalle</li>
  <li>Mari</li>
  <li>Oskar</li>
</ul>
```

Ülesandeid

* Koosta mall, mis väljastaks tärnidest rea. Katseta.
* Koosta mall, mis trükiks etteantud teksti välja punaselt.
* Võrreldes eelmisega saab ka värvi anda ette malli väljakutsel

* Koosta mall, mis trükib etteantud elemendist loeteluna välja kõikide alamelementide nimed ja nende väärtused.
* Koosta mall, mis saab etta sarnase struktuuriga elementide (näiteks inimeste) massiivi ning väljastab nende alamelementide tulemused tabelina. Katseta mitmesuguste sisendandmete korral.

Struktuursed andmed

XMLi hea omadus on, et selle abil kannatab mitmesuguse struktuuriga andmeid hoida ja üle kanda. Andmeid tabelis hoides peab soovitavalt iga rea puhul ühepalju veerge olema. Või kui eri ridade kohta erinev kogus andmeid teada, siis tuleb tabelite puhul mitmesuguseid trikke välja mõelda, kuidas tulemus mõistlikult hoida. XMLis aga saab viisakasti hoida kõike, mis on puukujulisse struktuuri paigutatav. Ja mõningase mõtlemise peale saab puu kujule paigutada pea kõik andmed.

Näitena ja päringute katsetamiseks sobib ette võtta sugupuu. Kirja saab panna seda mitmel kujul, üks võimalik lahendus on allpool, kus igal inimesel on oma isikuandmed. Lisaks sees element lapsed, kus üksteise järel kirjas lapsed ja nende andmed. Ja nõnda üha sügavamale ja sügavamale. Niisama lihttekstina võib suure suguvõsa korral tekst lugedes pikaks minna. Brauseri või mõne muu vaatamisvahendi abil saab aga harusid kokku-lahku klõpsida ning vaid soovitud koha välja võtta.

```
<?xml version='1.0'?>
<inimene>
    <eesnimi>Aleksander</eesnimi>
    <perekonnanimi>Kippar</perekonnanimi>
    <synniaasta>1908</synniaasta>
    <lapsed>
```

```
        <inimene>
            <eesnimi>Pille</eesnimi>
            <perekonnanimi>Kippar</perekonnanimi>
            <synniaasta>1935</synniaasta>
            <lapsed>
                <inimene>
                    <eesnimi>Jaagup</eesnimi>
                    <perekonnanimi>Kippar</perekonnanimi>
                    <synniaasta>1976</synniaasta>
                    <lapsed>
                        <inimene>
                            <eesnimi>Toomas</eesnimi>
                            <perekonnanimi>Kippar</perekonnanimi>
                            <synniaasta>2008</synniaasta>
                        </inimene>
                    </lapsed>
                </inimene>
            </lapsed>
        </inimene>
        <inimene>
            <eesnimi>Viivi</eesnimi>
            <perekonnanimi>Laas</perekonnanimi>
            <synniaasta>1939</synniaasta>
            <lapsed>
                <inimene>
                    <eesnimi>Heli</eesnimi>
                    <perekonnanimi>Kiik</perekonnanimi>
                    <synniaasta>1964</synniaasta>
                    <lapsed>
                        <inimene>
                            <eesnimi>Dorel</eesnimi>
                            <perekonnanimi>Kiik</perekonnanimi>
                            <synniaasta>1987</synniaasta>
                        </inimene>
                        <inimene>
                            <eesnimi>Grete</eesnimi>
                            <perekonnanimi>Kiik</perekonnanimi>
                            <synniaasta>1989</synniaasta>
                        </inimene>
                    </lapsed>
                </inimene>
                <inimene>
                    <eesnimi>Liina</eesnimi>
                    <perekonnanimi>Jerkku</perekonnanimi>
                    <synniaasta>1970</synniaasta>
                    <lapsed>
                        <inimene>
                            <eesnimi>Stenver</eesnimi>
                            <perekonnanimi>Jerkku</perekonnanimi>
                            <synniaasta>1990</synniaasta>
                        </inimene>
                        <inimene>
                            <eesnimi>Linnea</eesnimi>
                            <perekonnanimi>Jerkku</perekonnanimi>
                            <synniaasta>2003</synniaasta>
                        </inimene>
                    </lapsed>
                </inimene>
            </lapsed>
        </inimene>
    </lapsed>
</inimene>
```

Ehkki andmed on esiotsa puu kujul, saab sealt neid edaspidi sobivalt välja võtta. Tahtes saada kogu puust kätte kõiki inimesi, piisab kui valikumustriks kirjutada //inimene . Kaks kaldkriipsu mustri alguses ütleb, et vastava nimega elementi tuleb otsida kogu andmepuu ulatuses. Parameeter xsl:sort for-each tsükli sees määrab, et millise tunnuse alusel tuleb sisemised andmed järjestada.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <ul>
            <xsl:for-each select="//inimene">
                <xsl:sort select="synniaasta" />
                <li>
                    <xsl:value-of select="eesnimi" />
                </li>
            </xsl:for-each>
        </ul>
    </xsl:template>
</xsl:stylesheet>
```

Ja ongi inimesed sünniaastate järjekorras loetelus.

* Aleksander
* Pille
* Viivi
* Heli
* Liina
* Jaagup
* Dorel
* Grete
* Stenver
* Linnea
* Toomas

Puus allapoole saab ilusti elementide nimesid pidi küsida. Andmeid saab võtta aga ka puust kõrgemalt poolt. Nõnda nagu failisüsteemis kataloogide vahel liikudes saab kahe punktiga liikuda kõrgemale tasemele, nii saab ka XML dokumendis kõrgema taseme elemente küsida kahe punkti abil. Tahtes konkreetse inimese real küsida tema lapsevanema nime, tuleb avaldiseks ../../eesnimi . Ehk siis esimesed kaks punkti viivad tagasi elemendi "lapsed" juurde, järgmised kaks punkti ülemise "inimene" juurde. Ning sealt edasi juba võib otse eesnime küsida. Tingimuslause if sellepärast juurde, et näidataks ainult neid inimesi loetelus, kellel esivanema andmed teada. Kui test-atribuudis olev avaldis ei anna väärtust, siis if-lause sisu ei täideta.

```
        <ul>
            <xsl:for-each select="//inimene">
                <xsl:sort select="synniaasta" />
                <li>
                    <xsl:value-of select="eesnimi" />
                    <xsl:if test="../..">
```

```
                - lapsevanem <xsl:value-of select="../../eesnimi" />
            </xsl:if>
        </li>
    </xsl:for-each>
</ul>
```

Ja nõnda saabki algsest andmepuust sobiva väljavõtte teha.

* Aleksander
* Pille - lapsevanem Aleksander
* Viivi - lapsevanem Aleksander
* Heli - lapsevanem Viivi
* Liina - lapsevanem Viivi
* Jaagup - lapsevanem Pille
* Dorel - lapsevanem Heli
* Grete - lapsevanem Heli
* Stenver - lapsevanem Liina
* Linnea - lapsevanem Liina
* Toomas - lapsevanem Jaagup

Siinses ringis võetakse ette kõik inimesed, kel lapsed on. Samuti: avaldis //inimene[lapsed] annab välja ainult lastega inimesed. Sorteeritakse esimese lapse sünniaasta järgi. Ehkki lapsed/inimene/synniaasta annab iseenesest kätte kõikide laste sünniaastad. Kui aga küsitakse ainult ühte väärtust, siis antakse välja esimene.

Et käänded paika saab, tuleb tingimuse juures kindlaks teha, mitu last kellelgi on. Vastavalt sellele saab soovitud sõna kirjutada.

```
<ul>
    <xsl:for-each select="//inimene[lapsed]">
        <xsl:sort select="lapsed/inimene/synniaasta" />
        <li>
            <xsl:value-of select="eesnimi" />,
            <xsl:value-of select="count(lapsed/inimene)"/>
            <xsl:if test="count(lapsed/inimene)=1">
                laps
            </xsl:if>
            <xsl:if test="not(count(lapsed/inimene)=1)">
                last
            </xsl:if>
        </li>
    </xsl:for-each>
</ul>
```

Ja jällegi vastused käes.

```
* Aleksander, 2 last
* Viivi, 2 last
* Pille, 1 laps
* Heli, 2 last
* Liina, 2 last
* Jaagup, 1 laps
```

Tsükli seest võib ka sissepoole edasi minna. Siinses näites võetakse kõigepealt loetellu kõik andmepuus leiduvad lastega inimesed. Ning edasi loetletakse iga inimese kohta kõik tema järglased. Avaldis lapsed//inimene

```
    <xsl:template match="/">
        <ul>
            <xsl:for-each select="//inimene[lapsed]">
                <li>
                    <xsl:value-of select="eesnimi" />:
                    <xsl:for-each select="lapsed//inimene">
                        <xsl:value-of select="eesnimi" />
                        <xsl:if test="not(position()=last())">, </xsl:if>
                    </xsl:for-each>
                </li>
            </xsl:for-each>
        </ul>
```

* Aleksander: Pille, Jaagup, Toomas, Viivi, Heli, Dorel, Grete, Liina, Stenver, Linnea
* Pille: Jaagup, Toomas
* Jaagup: Toomas
* Viivi: Heli, Dorel, Grete, Liina, Stenver, Linnea
* Heli: Dorel, Grete
* Liina: Stenver, Linnea

Veebilehel tulevad andmed tavapäraselt HTMLiga kujundatult. Kui aga tahtmist lihtsaid andmeid kätte saada, siis võib tulemuse ka otse XMLina välja näidata. Järgnevas näites võetakse tsüklis ette kõik sugupuus leiduvad lastega inimesed, paigutatakse loetellu üksteise järele ning igaühe elemendi sisse kopeeritakse tema järetulijad. Käsklus xsl:copy-of kopeerib select-parameetrina ette antud elemendi väärtuse väljundisse koos alanejatega.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <inimesed>
            <xsl:for-each select="//inimene[lapsed]">
                <inimene>
                    <eesnimi><xsl:value-of select="eesnimi" /></eesnimi>
                    <xsl:copy-of select="lapsed" />
                </inimene>
            </xsl:for-each>
        </inimesed>
    </xsl:template>
```

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

Java puhul käib tulemusfaili kättesaamine sarnaselt eelnevatele näidetele. ASP.NET puhul agai tuleb HTMLi osa aspx-failist välja võtta.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="sugupuu4.aspx.cs"
Inherits="sugupuu4" %>
<asp:xml id="xml1" runat="server" documentsource="~/sugupuu.xml"
transformsource="~/sugupuu4.xslt" />
```

Koodifaili on sobiv lisada lehe andmetüübiks "text/xml". Samuti tasub lisada XML-faili algustunnus. Sellisel juhul tunneb veebilehitseja üldjuhul ära ja suudab lasta mugavalt elemente kokku/lahti klõpsida.

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Xml.Xsl;

public partial class sugupuu4 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.ContentType = "text/xml";
        Response.Write("<?xml version='1.0'?>");

    }
}
```

Nagu tulemusest näha,

```
<?xml version='1.0'?><inimesed>
  <inimene>
    <eesnimi>Aleksander</eesnimi>
    <lapsed>
      <inimene>
        <eesnimi>Pille</eesnimi>
        <perekonnanimi>Kippar</perekonnanimi>
        <synniaasta>1935</synniaasta>
        <lapsed>
          <inimene>
            <eesnimi>Jaagup</eesnimi>
            <perekonnanimi>Kippar</perekonnanimi>
            <synniaasta>1976</synniaasta>
            <lapsed>
              <inimene>

                <eesnimi>Toomas</eesnimi>
                <perekonnanimi>Kippar</perekonnanimi>
                <synniaasta>2008</synniaasta>
              </inimene>
```

```
            </lapsed>
          </inimene>
        </lapsed>

      </inimene>
      <inimene>
        <eesnimi>Viivi</eesnimi>
        <perekonnanimi>Laas</perekonnanimi>
        <synniaasta>1939</synniaasta>
        <lapsed>
...
```

Ülesandeid

* Koosta sarnane sugupuu ühest oma vanavanemast alates.
* Trüki välja kõikide inimeste sünniaastad.
* Väljastatakse nimed, kel on vähemalt kaks last.
* Väljasta sugupuus leiduvad andmed tabelina.
* Kus võimalik, seal väljasta tabelis iga inimese vanema nimi.
* Väljasta tabelis ka vanavanema nimi.
* Tabelisse paigutatakse vaid parameetriga ette antud perekonnanimega inimesed.
* Väljasta iga inimese juures, mitmendal oma vanema sünniaastal ta sündis.
* Andmepuus muudetakse sünniaasta atribuudiks.
* Andmepuus lisatakse igale inimesele element, mille sisu koosneb eesnime tähest, punktist ja perekonnanimest.

Osakond

* Koosta asutuse ühe osakonna kirjeldus ja andmed XML-ina. Nimetus, eesmärgid, juhataja, töötajad. Iga inimese kohta vähemalt eesnimi, perekonnanimi, amet ja palk.

* Väljasta XSLiga kõik ametid.
* Väljasta töötajate arv.
* Väljasta asjaajajate arv.
* Väljasta asjaajajad perekonnanimede järjekorras.
* Väljasta töötajad HTML-tabelina ameti järgi järjestatuna.
* Väljasta andmed eraldi XML-failina, kus on vaid ees- ja perekonnanimed ning osakonna nimetus.

Asutus

* Kavanda suurema ettevõtte struktuur. Ettevõtte üksused paiknevad mitmes linnas, igas neist võib olla mitu osakonda. Igas osakonnas hulk mitmesuguste ametitega inimesi.
* Koosta XML-fail andmetega. Kiiremaks loomiseks saab elemente kopeerida.

* Loo XMLi põhjal XSLiga HTML-fail. Ettevõte ja üksused on pealkirjadena. Iga osakonna inimeste kohta on HTML-tabel.
* Leia iga osakonna ning kogu ettevõtte töötajate arv.
* Koosta XSLiga eraldi XML-fail, kus on kirjas kõik sekretärid. Iga

sekretäri atribuudiks on tema telefoninumber.

* Teata parameetrina ette antud osakonna töötajate arv.
* Teata osakonnad, kus töötajaid on rohkem, kui parameetrina etteantud arv.

Skeemid

Kui andmestik on lihtne ja alati samade tunnustega, siis on hea ette võtta sobiv näide, omad andmed asemele panna ning saabki uusi andmeid olemasoleval kohal kasutada. Kui aga on vaja täpsemalt teada, mis kuhu lubatud on, siis selle kirjapanekuks on välja mõeldud skeemid. Seal määratakse, millise nimega elemendid millises järjekorras on lubatud ning millist tüüpi ja millises vahemikus andmeid võib kusagil kasutada. Skeemi järgi saab andmefaili automaatselt kontrollida ning selle kaudu vähemasti märgatava osa näpuvigu avastada. Vigade avastamine võimalikult varakult aitab vältida nende mõju kasvamist. Samuti on skeemi pealt hea vaadata rakenduste loojatel, et kuidas andmeid esitada tuleb. Mis on lubatud ja mis mitte.

Lihtne näide

Tutvuda tasub alates kõige lihtsamast võimalusest. XML-dokument, kus on vaid ühe teate tekst.

```
<?xml version="1.0" ?>
<teade>Uus aasta algab 1. jaanuaril.</teade>
```

Et seal saab olla vaid üks tekstiline teade ja ei midagi muud, see tuleb skeemina kirja panna järgnevalt.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="teade" type="xs:string"/>
</xs:schema>
```

Kuna skeem ise vastab XML-i reeglitele, siis peab tal ülal olema tüübideklaratsioon.

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

Edasi teatatakse, et algab skeem.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Sealt siis teated, et millise nime ja millise tüübiga elemendid tulevad. Praegu ainult üks element nimega teade ja sisutüübiks string.

```
 <xs:element name="teade" type="xs:string"/>
```

Ja nagu XMLi puhul ikka, siis alanud element tuleb lõpetada.

```
</xs:schema>
```

Andmete vastavust skeemile saab kontrollida mitmetes programmeerimiskeeltes. Järgnevalt näide Java kohta. Kõigepealt luuakse vastavalt skeemifailile validaator ning edasi kontrollitakse selle järgi soovitud faili.

```
import javax.xml.validation.*;
import javax.xml.XMLConstants;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.*;
public class SkeemiKontroll{
  public static void main(String[] argumendid) throws Exception{
        if(argumendid.length!=2){
            System.out.println(
             "Kasuta kujul: java SkeemiKontroll skeem.xsd andmefail.xml");
            System.exit(0);
        }
        Validator validaator=SchemaFactory.newInstance(
           "http://www.w3.org/2001/XMLSchema").
               newSchema(new StreamSource(argumendid[0])).newValidator();
        validaator.validate(new StreamSource(argumendid[1]));
  }
}
```

Nagu ikka, tuleb enne käivitamist fail kompileerida. Ning edasi tuleb talle ette anda skeem ning andmefail.

```
c:\temp>javac SkeemiKontroll.java
c:\temp>java SkeemiKontroll teade.xsd teade.xml
```

Skeemifaili asukoht on võimalik ka andmefaili enese sisse kirjutada. Järgnevas näites märgitakse, et andmetele vastava skeemi leiab failist nimega teateskeem.xsd . Kui failid avada nt Microsoft Visual Studio abil ja nad asuvad samas kataloogis, siis kontrollitakse automaatselt, kas andmefail vastab skeemile. Kui kõik on korras, siis ei teatata midagi.

```
<?xml version="1.0" ?>
<teade xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="teateskeem.xsd">
  Uus aasta algab 1. jaanuaril.
</teade>
```

Kui aga andmefail ei peaks skeemile vastama, siis antakse vastav hoiatus. Siin on teate sisse lisatud skeemis kirjeldamata element nimega "selgitus". Visual Studio keskkond tõmbab sellele hoiatusjoone alla. Mõnes muus kohas aga näiteks teatatakse käsureaaknast, et millisel koodireal ja milline viga leiti.

```
<?xml version="1.0" ?>
<teade xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="teateskeem.xsd">
  Uus aasta algab 1. jaanuaril.
  <selgitus></selgitus>
</teade>
```

```
<?xml version="1.0" ?>
<teade xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="teateskeem.xsd">
  Uus aasta algab 1. jaanuaril.
  <selgitus></selgitus>
</teade>
```

Mitmest elemendist koosnev skeem.

Harva piirdub XML-fail ühe elemendiga. Ning õigupoolest pole nii lihtsa faili tarbeks põhjust ka skeemi koostada. Kui elemente rohkem, siis aga skeem kindlasti omal kohal. Järgnev kirjeldus inimese andmete kohta. Väliseks elemendiks inimene. Et tema sisse saab teisi elemente paigutada, siis määratakse, et tegemist on complexType'ga. Ning siin näites paiknevad sisemised elemendid kindlas järjekorras jadana (sequence). Kõigepealt eesnimi stringi ehk tekstina, siis perenimi ning lõpuks synniaasta. Viimane peab olema täisarvuline, et andmestik validaatorist läbi läheks.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta" type="xs:integer" />
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

Skeemile vastav näide siis järgmine.

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Kui tahta, et suudetaks ka automaatselt kontrollida andmete vastavust skeemile, siis tuleb skeemifaili asukoht juurelemendi atribuudi juures ära määrata.

```
<?xml version="1.0" ?>
<inimene xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="inimene1.xsd">
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Ülesandeid

* Koosta skeem auto registrinumbri hoidmiseks.
* Loo vastav andmefail. Testi andmefaili vastavust skeemile.
* Koosta skeem, kus auto kohta on kirjas registrinumber, mark ja väljalaskeaasta.
* Testi andmestiku vastavust skeemile.

Elemendi esinemise kordade arv

Eelmises näites on iga element ühekordselt. Kui midagi eraldi määratud pole, siis käibki nõnda. Ehk siis vaikimisi on elemendi juurde kuuluvate arvuatribuutide maxOccurs ning minOccurs väärtusteks 1. Need aga võib üle kirjutada. Lubades kuni kolm eesnime, võib lisada atribuudi maxOccurs="3".

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" maxOccurs="3"/>
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta" type="xs:integer" />
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

Nõnda muutub lubatavaks järgmine andmestik, kus inimesel kaks eesnime.

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <eesnimi>Ferdinant</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Kui element võib kord olla, kord puududa, siis tuleb kirjelduse juurde määrata minOccurs="0". Sellistes kohtades on skeemist eriti kasu, sest näitfailis ei pruugi vajalikku elementi olla, skeemist aga saab vaadata, mis on lubatud, mis mitte.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="hyydnimi" type="xs:string" minOccurs="0"/>
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta" type="xs:integer" />
        </xs:sequence>
     </xs:complexType>
```

```
  </xs:element>
</xs:schema>
```

Ning siis juurde näitena hüüdnimega andmestik.

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <hyydnimi>juks</hyydnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Atribuut

Atribuudi kirjeldus tuleb samuti xs:complexType elemendi sisse panna. Mitte aga elementide jadasse, vaid eraldi. Atribuutide puhul pole nende järjekord tähtis.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta" type="xs:integer"/>
        </xs:sequence>
        <xs:attribute name="sugu" type="xs:string" />
     </xs:complexType>
  </xs:element>
</xs:schema>
```

Atribuudiga element näeb siis välja järgmine.

```
<?xml version="1.0" ?>
<inimene sugu="m">
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Ülesandeid

* Loo skeem auto kohta, millel võib lisaks omanikule olla kuni kolm kasutajat.
* Auto kütuse liik tähistatakse atribuudina.

Tüüpide täiendamine

Harilikuks tekstitüübiks on string, täisarvuks integer, murdarvuks decimal ning kuupäevaks date. Kusjuures kuupäeva puhul tuleb andmed kirjutada alati kujul aasta-kuu-päev. Hiljem rakenduses võib neid kasutajale sobival kujul sättida, kuid XMLi standardis on määratud nõnda, et ei tekiks segadusi.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaeg" type="xs:date"/>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaeg>1989-11-05</synniaeg>
</inimene>
```

Lihtsad andmed saab "kohe ja kohapeal" ära kirjeldada. Kui aga andmestik läheb keerulisemaks, siis on võimalik üksikud osad enne gruppidena valmis teha ning alles seejärel gruppe omavahel kombineerima hakata. Nii nagu programmikoodi kirjutamisel saab enne valmis teha alamprogrammid ning neid siis sobivalt välja kutsuma hakata, nii ka skeemi puhul on võimalik kõigepealt defineerida omaette tüübid ning neid siis hiljem kasutada. Siinses lihtsas näites on kõigepealt defineeritud isikutüüp ning hiljem märgitud, et dokument koosneb ühest, isikutüüpi inimesest.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="isikutyyp">
      <xs:sequence>
         <xs:element name="eesnimi" type="xs:string" />
         <xs:element name="perenimi" type="xs:string" />
         <xs:element name="synniaeg" type="xs:date"/>
      </xs:sequence>
  </xs:complexType>

  <xs:element name="inimene" type="isikutyyp"/>
</xs:schema>
```

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaeg>1989-11-05</synniaeg>
</inimene>
```

Kui xs:sequence määras, et elemendid peavad olema kindlas järjekorras, siis xs:all järjekorda ei määra, küll ütleb aga ikka, et kõik määratud elemendid peavad olemas olema.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:all>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta" type="xs:integer"/>
        </xs:all>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

Nõnda on lubatud ja järgnev

```
<?xml version="1.0" ?>
<inimene>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
  <eesnimi>Juku</eesnimi>
</inimene>
```

Käsklus xs:choice teatab, et määratutest peab olema ainult üks. Ehk siis järgnevas näites saab inimese alla panna kooli või asutuse, aga mitte mõlemat.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:choice>
           <xs:element name="kool" type="xs:string" />
           <xs:element name="asutus" type="xs:string" />
        </xs:choice>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" ?>
<inimene>
  <kool>Nurgametsa kool</kool>
</inimene>
```

Ülesandeid

* Koosta skeem auto andmete hoidmiseks. Märgi auto registreerimise aeg kuupäevana. Kontrolli skeemi ja andmete sobivust.
* Luba auto andmed esitada vabas järjestuses.
* Defineeri isikuandmed (eesnimi, perekonnanimi, isikukood) eraldi tüübina. Määra nii auto omanik kui kasutajad vastavat tüüpi.
* Valikuna on auto kütuseks kas bensiin, diisel või gaas. Bensiini puhul on alamelementideks lubatud vähim ja suurim oktaanarv.

Tüüpide piirangud

Terviktüüpe ette võttes on kasutada string, date, integer ja decimal. Tüübi juures lubatud terve väärtuste vahemik ei pruugi aga konkreetses kohas sugugi mõistlik olla. Mida vähem on lubatud väärtusi, seda rohkem on lootust, et näpuveana sisestatud valed andmed läbi ei lähe. Arvu puhul saab näiteks seada alam- ja ülempiiri. Kui tüübi ulatust piiratakse, siis tuleb elemendi sisse eraldi elemendiks xs:simpleType. Selle sisse xs:restriction, mille base-atribuudiga määratakse ära, milline tüüp aluseks võetakse. Ja xs:restrictioni sisse siis omakorda loetelu täiendavatest piirangutest.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="inimene">
     <xs:complexType>
        <xs:sequence>
           <xs:element name="eesnimi" type="xs:string" />
           <xs:element name="perenimi" type="xs:string" />
           <xs:element name="synniaasta">
             <xs:simpleType>
              <xs:restriction base="xs:integer">
                <xs:minInclusive value="1900"/>
                <xs:maxInclusive value="2100"/>
              </xs:restriction>
             </xs:simpleType>
           </xs:element>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" ?>
<inimene>
  <eesnimi>Juku</eesnimi>
  <perenimi>Juurikas</perenimi>
  <synniaasta>1989</synniaasta>
</inimene>
```

Sobivad tüübid saab ka eraldi välja kirjutada. Siis ei lähe hilisem andmestiku määratlemine liialt pikaks. Järgnevalt siis näide täisarvu kohta, kus seatakse alumine ja ülemine piir. Nimetüübis olev regulaaravaldis näitab, et praegusel juhul peab nimi algama suurtähega (üks täht vahemikus A-st Z-ni ning edasi tuleb üks või rohkem (mida määrab +) väikest tähte. Täpitähed tuleks loetellu eraldi lisada, kui neid ka lubada. Joogitüübile on jäetud võimalus olemasolevate väärtuste hulgast valida. Lisaks on levinumate piirangute seas veel pikkuse jaoks minLength, length ja maxLength.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:simpleType name="aastatyyp">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="1900"/>
```

```
            <xs:maxInclusive value="2100"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="nimetyyp">
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][a-z]+"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="joogityyp">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Coca-Cola"/>
            <xs:enumeration value="Fanta"/>
            <xs:enumeration value="Sprite"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="inimene">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="eesnimi" type="nimetyyp" />
                <xs:element name="perenimi" type="nimetyyp" />
                <xs:element name="synniaasta" type="aastatyyp"/>
                <xs:element name="jook" type="joogityyp"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>


<?xml version="1.0" ?>
<inimene xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="tyybid.xsd">
    <eesnimi>Juku</eesnimi>
    <perenimi>Juurikas</perenimi>
    <synniaasta>1989</synniaasta>
    <jook>Coca-Cola</jook>
</inimene>
```

Ülesandeid

* Koosta piirang Eesti Vabariigi valitsuse määruse aastanumbri kohta. Aastanumbri väärtus peab olema vähemalt 1918.
* Koosta piirang parooli jaoks. Pikkus peab olema vähemalt kuus tähte.
* Koosta skeem auto andmete jaoks. Registrimärk peab olema kujul kolm numbrit ja kolm tähte, omaniku perekonnanimi peab algama suure tähega ja ülejäänud tähed peavad olema väikesed.

Pikemad andmestikud

Elemente võib olla ka hulgem üksteise sees ning seal igaühes omakorda alamelemendid. Kui elementide arv pole piiratud, siis selle tähistamiseks sobib atribuudi maxOccurs väärtus unbounded.

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="inimesed">
  <xs:complexType>
    <xs:sequence>
     <xs:element name="inimene" maxOccurs="unbounded">
        <xs:complexType>
           <xs:sequence>
              <xs:element name="eesnimi" type="xs:string" />
              <xs:element name="perenimi" type="xs:string" />

              <xs:element name="synniaasta" type="xs:integer" />
           </xs:sequence>
        </xs:complexType>
     </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<inimesed>
 <inimene>
   <eesnimi>Juku</eesnimi>
   <perenimi>Juurikas</perenimi>
   <synniaasta>1963</synniaasta>
 </inimene>
 <inimene>

   <eesnimi>Juku</eesnimi>
   <perenimi>Kaalikas</perenimi>
   <synniaasta>1961</synniaasta>
 </inimene>
 <inimene>
   <eesnimi>Kalle</eesnimi>
   <perenimi>Kaalikas</perenimi>

   <synniaasta>1975</synniaasta>
 </inimene>
 <inimene>
   <eesnimi>Mari</eesnimi>
   <perenimi>Maasikas</perenimi>
   <synniaasta>1981</synniaasta>
 </inimene>

 <inimene>
   <eesnimi>Oskar</eesnimi>
   <perenimi>Ohakas</perenimi>
   <synniaasta>1971</synniaasta>
 </inimene>
</inimesed>
```

Kui sama ülesehitusega elementi soovitakse kasutada mitmel pool, siis on otse loomulik, et ta tuleb eraldi ära kirjeldada. Järgnevas näites on nii õpetaja kui õpilased isikutüüpi ning pääseb eesnime ja muude isikuandmete mitmekordsest kirjeldamisest.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="isikutyyp">
      <xs:sequence>
```

```
        <xs:element name="eesnimi" type="xs:string" />
        <xs:element name="perenimi" type="xs:string" />
        <xs:element name="synniaasta" type="xs:integer"/>
      </xs:sequence>
 </xs:complexType>
 <xs:element name="inimesed">

  <xs:complexType>
    <xs:sequence>
     <xs:element name="kirjeldus" type="xs:string" />
     <xs:element name="opetaja" type="isikutyyp" />
     <xs:element name="opilane" type="isikutyyp" maxOccurs="50" />
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
<?xml version="1.0"?>
<inimesed>
 <kirjeldus>Nurgametsa kooli 7. klass</kirjeldus>
 <opetaja>
   <eesnimi>Juku</eesnimi>
   <perenimi>Juurikas</perenimi>
   <synniaasta>1963</synniaasta>
 </opetaja>

 <opilane>
   <eesnimi>Juku</eesnimi>
   <perenimi>Kaalikas</perenimi>
   <synniaasta>1991</synniaasta>
 </opilane>
 <opilane>
   <eesnimi>Kalle</eesnimi>

   <perenimi>Kaalikas</perenimi>
   <synniaasta>1991</synniaasta>
 </opilane>
 <opilane>
   <eesnimi>Mari</eesnimi>
   <perenimi>Maasikas</perenimi>
   <synniaasta>1992</synniaasta>

 </opilane>
 <opilane>
   <eesnimi>Taavi</eesnimi>
   <perenimi>Ohakas</perenimi>
   <synniaasta>1992</synniaasta>
 </opilane>
</inimesed>
```

Kui andmestik paisub veel suuremaks ja keerukamaks, siis saab skeemid jagada laiali mitmesse faili. Nõnda võib üksikutes failides defineerida kogu dokumendi jaoks vajalikud tüübid ning hiljem alles neid vajalikes kohtades pruukida. Siin ühes failis kõigepealt defineeritakse tänavatüüp (mis praegusel juhul võib koosneda kuni kolmest sõnast) ja indeksitüüp. Nende põhjal ehitatakse kokku aadressitüüp. Katseks on defineeritud ka üks aadress, kuid see ei ole kohustuslik.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="tänavatüüp">
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][a-z]+( [a-z]+)?( [a-z]+)?"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="indeksitüüp">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="10000" />
            <xs:maxInclusive value="99999" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="aadressitüüp">
        <xs:all>
            <xs:element name="indeks" type="indeksitüüp" />
            <xs:element name="aadressivalik">
                <xs:complexType>
                    <xs:choice>
                        <xs:element name="maaaadress">
                            <xs:complexType>
                                <xs:all>
                                    <xs:element name="talu" type="xs:string" />
                                </xs:all>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="linnaaadress">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="tänav" type="tänavatüüp" /
>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:all>
    </xs:complexType>
    <xs:element type="aadressitüüp" name="aadress" />
</xs:schema>
```

Edasi saab teises failis sisse lugeda aadressikontrollifailist tüübikirjeldused ning neid oma kirjelduse juures kasutada. Nagu näha, siinse osakonnatüübi juures märgitakse, et aadress on aadressitüüpi. See pikk ja keeruline tüüp on aga juba eelnevalt defineeritud failis nimega aadressikontroll.xsd, mis xs:include käsuga sisse loetakse.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include schemaLocation="aadressikontroll.xsd"></xs:include>
    <xs:complexType name="osakonnatüüp">
        <xs:sequence>
            <xs:element name="nimetus" type="xs:string" />
            <xs:element name="aadress" type="aadressitüüp" />
        </xs:sequence>
    </xs:complexType>
    <xs:element name="osakond" type="osakonnatüüp"  />
</xs:schema>
```

Nii ongi kogu süsteem jaotatud kolme faili. XML-andmefail loeb osakonna tüübi välja failist

osakonnakontroll.xsd. See aga omakorda kasutab veel faili aadressikontroll.xsd.

```
<?xml version="1.0" encoding="utf-8" ?>
<osakond xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="osakonnakontroll.xsd">
    <nimetus>Kultuuriosakond</nimetus>
    <aadress>
        <indeks>17655</indeks>
        <aadressivalik>
            <linnaaadress>
                <tänav>Lai</tänav>
            </linnaaadress>
        </aadressivalik>
    </aadress>
</osakond>
```

Ülesandeid

* Koosta skeem ülevaatuse läbinud autode loetelu kuvamiseks. Iga auto kohta registrinumber, mark, valmimisaasta, omanik. Sea tüüpidele mõistlikud piirangud.
* Koosta skeem arvuti võimalike omaduste ja osade kirjeldamiseks. Samuti sea elementide väärtustele piirangud, mis võimalust mööda vähendaksid näpuvigu.

Rekursiivsed andmed, sugupuu

Skeemi üheks heaks omaduseks on võimalus defineerida üksteisesse sisenevaid samatüübilisi andmeid. Sellist andmestruktuuri nimetatakse rekursiivseks. Tuleb ta ette näiteks failisüsteeme, kus kataloogid võivad sisaldada faile ja katalooge, viimased taas omakorda faile ja katalooge ning nõnda edasi. Või siinses näites, kus inimese elemendi sees on tema lapsed. Ning laste sees on jällegi loetelu inimestest, kellel igaühel taas võivad lapsed olla. Sellise struktuuri saab kirja panna nõnda, kus sisenevalt korduv tüüp on eraldi defineeritud ning ta enese sees sisaldab ka sama tüüpi elemente. Et ehkki laste loetelu juures ei ole isikutüübi defineerimine veel lõppenud, võib juba määrata, et laste sees on loetelu määramata arvust isikutüüpi inimestest. Siinne skeem sobib kirjelduseks sugupuule, mis omaette andmestikuna juba eelpool näidati.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="isikutyyp">
        <xs:sequence>
            <xs:element name="eesnimi" type="xs:string" />
            <xs:element name="hyydnimi" type="xs:string" minOccurs="0" />
            <xs:element name="perekonnanimi" type="xs:string" />
            <xs:element name="synniaasta" type="xs:int" />
            <xs:element name="lapsed" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="inimene"
                           type="isikutyyp" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
```

```
        </xs:sequence>
        <xs:attribute name="sugu" type="xs:string"  />
    </xs:complexType>
    <xs:element name="inimene" type="isikutyyp" />
</xs:schema>
```

Ülesandeid

* Koosta rekursiivse andmestruktuuri skeem kirjeldamaks kataloogides asuvaid faile ja alamkatalooge. Testi skeemi sobivust tegelike andmetega.
* Koosta skeem suurfirma organisatsiooni kirjeldamiseks. Kus firma jaguneb osakondadeks, igal neist võivad olla aga omakorda alamosakonnad. Osakonnas on töötajad, neil igaühel ametinimetus ja palk.

SAX

Lühend lahtikirjutatult on "Simple API for XML" ehk kokkuvõtlikult vahend XMList andmete kätte saamiseks. Ei saa öelda, et see programmeerija tarbeks eelnevalt tutvustatud DOMist lihtsam oleks. Aga masinale on kindlasti. SAXi abil kannatab ka tagasihoidliku arvuti abil hiiglaslikust andmestikust sobivad tulemused välja noppida.

Iseenesest on ju võimalik XML-faile lugeda nagu tavalisi tekstifaile. Et reajagu teksti sisse. Sealt sobivad analüüsid teha ning siis jälle järgmine tekstirida lugeda. Kui parasjagu kasutatavas programmeerimiskeeles muid võimalusi pole, siis sealtkaudu saab enamiku kindlasti tehtud. SAXi puhul on aga tööd programmeerija jaoks veidi lihtsustatud. Plokkideks pole enam mitte tekstiread, vaid XMLi elemendid. Reageerimiseks luuakse sobivad funktsioonid. Lihtsaimal juhul vaid teated elemendi alguse ja lõpu kohta. Sellisel juhul pole vaja ise enam tekstifunktsioonidega elemente otsida, vaid piisab sobivale funktsioonile reageerimisest.

```
$xml_parser = xml_parser_create();
```

loob uue parseriobjekti.

```
xml_set_element_handler($xml_parser, "startElement", "endElement");
```

määrab, milliste nimedega funktsioonid on vastaval parseril elemendi algusele ja lõpule reageerimiseks. Tavaliselt jäetakse need inglisekeelsed levinud nimed. Ning allpool siis avatakse XMLi fail (või ka näiteks mujalt võrgust veebiaadressi järele tulev voog) lugemiseks. Andmed loetakse plokkide kaupa ning antakse parserile töötlemiseks. Edasi juba saab igaüks funktsioonide sees teha nende andmetega seda, mida ta just vajalikuks peab.

```
<?php
 function startElement($parser, $element, $atribuudid){
   echo "Algas $element ";
 }

 function endElement($parser, $element){
   echo "Lõppes $element ";
 }

 $xml_parser = xml_parser_create();
```

```
 xml_set_element_handler($xml_parser, "startElement", "endElement");
 $f=fopen("inimesed.xml", "r");
 while($andmed=fread($f, 4096)){
   xml_parse($xml_parser, $andmed, feof($f));
 }

 xml_parser_free($xml_parser);

?>
```

Eelmistes peatükkides kasutatud inimeste loetelu puhul saadakse järgmine väljund:

```
Algas INIMESED Algas INIMENE Algas EESNIMI Lõppes EESNIMI Algas PERENIMI Lõppes
PERENIMI Algas SYND Lõppes SYND Lõppes INIMENE Algas INIMENE Algas EESNIMI
Lõppes EESNIMI Algas PERENIMI Lõppes PERENIMI Algas SYND Lõppes SYND Lõppes
INIMENE Algas INIMENE Algas EESNIMI Lõppes EESNIMI Algas PERENIMI Lõppes
PERENIMI Algas SYND Lõppes SYND Lõppes INIMENE Algas INIMENE Algas EESNIMI
Lõppes EESNIMI Algas PERENIMI Lõppes PERENIMI Algas SYND Lõppes SYND Lõppes
INIMENE Algas INIMENE Algas EESNIMI Lõppes EESNIMI Algas PERENIMI Lõppes
PERENIMI Algas SYND Lõppes SYND Lõppes INIMENE Lõppes INIMESED
```

Nagu näha, PHP muutis vaikejuhul elementide nimede tähed suurteks.

Nimede loendamine

Sugugi ei pea kõigile sündmustele reageerima. Kui tahetakse teada saada, et mitu eesnime on, siis piisab ainult elementide algusele reageerimisest. Ning seal saab valikuga kontrollida, et kui elemendi nimi on otsitav, siis see sündmus muutujasse kirja panna.

```
<?php
 $eesnimedeArv=0;

 function startElement($parser, $element, $atribuudid){
   global $eesnimedeArv;
   if($element=="EESNIMI"){$eesnimedeArv++;}
 }

 function endElement($parser, $element){

 }

 $xml_parser = xml_parser_create();

 xml_set_element_handler($xml_parser, "startElement", "endElement");
 $f=fopen("inimesed.xml", "r");
 while($andmed=fread($f, 4096)){
   xml_parse($xml_parser, $andmed, feof($f));
 }

 xml_parser_free($xml_parser);

 echo "Kokku $eesnimedeArv eesnime.\n";

?>
```

```
Kokku 5 eesnime.
```

SAXi ülesehitus on programmeerimiskeelte juures suhteliselt sarnane. Et Java on rohkem objektorienteeritud keel, siis tuleb seal luua sündmustele reageeriv objekt. Ehk siis kõigepealt sobivat tüüpi klass ning pärast new-käsuga klassi põhjal objekt. Samast failist saadakse aga ikka samapalju eesnimesid.

```
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");
  }
}
```

Ülesandeid

* Loo XML-fail, kus on kirjas autode registreerimisnumbrid. Teata, mitu registreerimisnumbrit failis on.
* Teata XML-failis leiduvad kõik erinevad elemendinimed.

Andmete püüdmine

Elemendisisude kätte saamise jaoks on tüüpilise käskluse nimeks characters. Enamasti saadakse sisu kätte ühekorraga, aga selles ei pruugi kindel olla. Võib vabalt juhtuda, et loetav sisu ei satu ühte plokki ning sel juhul võib see mitme osana kohale jõuda, igal korral käivitatakse characters just selle sisuga. Kui soovitakse saada vaid kindla nimega elemetide sisusid, siis tuleb startElement ning endElement-käskluste sees muutujas lülitina meeles pidada, et kas parajasti tulevat teksti on vaja välja trükkida või mitte. Et kui ollakse eesnime alguse ja lõpu vahel, siis tuleb, muidu mitte.

```
<?php
 $kasEesnimi=false;
 function startElement($parser, $element, $atribuudid){
   global $kasEesnimi;
   if($element=="EESNIMI"){$kasEesnimi=true;}
 }

 function endElement($parser, $element){
```

```
  global $kasEesnimi;
  if($element=="EESNIMI"){
     $kasEesnimi=false;
     print "\n";
  }
}

function characters($parser, $tekst){
  global $kasEesnimi;
  if($kasEesnimi){
    print $tekst;
  }
}

$xml_parser = xml_parser_create();

xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characters");
$f=fopen("inimesed.xml", "r");
while($andmed=fread($f, 4096)){
  xml_parse($xml_parser, $andmed, feof($f));
}

xml_parser_free($xml_parser);


?>
```

Nagu näha, tulid eesnimed välja:

```
Juku Juku Kalle Mari Oskar
```

Kui soovida kõiki XMLi kirjetes olevaid andmeid püüda, siis üheks võimaluseks on vastavaid muutujaid lihtsalt juurde teha. Ning kui lõppeb element "inimene", siis peaksid selleks ajaks ka kõik selle inimese andmed ükshaaval muutujates olema ning on võimalik andmestikuga seda ette võtta, mis parajasti vajalik on.

```
<?php
 $kasEesnimi=false;
 $kasSynniaasta=false;
 $eesnimi="";
 $synniaasta="";
 function startElement($parser, $element, $atribuudid){
   global $kasEesnimi, $kasSynniaasta,
          $eesnimi, $synniaasta;
   if($element=="EESNIMI"){$kasEesnimi=true;}
   if($element=="SYND"){$kasSynniaasta=true;}
   if($element=="INIMENE"){$eesnimi="";$synniaasta="";}
 }

 function endElement($parser, $element){
   global $kasEesnimi, $kasSynniaasta,
          $eesnimi, $synniaasta;
   if($element=="EESNIMI"){$kasEesnimi=false;}
   if($element=="SYND"){$kasSynniaasta=false;}
   if($element=="INIMENE"){
     echo "$eesnimi sündis $synniaasta. aastal.<br />";
```

```
  }
}

function characters($parser, $tekst){
  global $kasEesnimi, $kasSynniaasta,
         $eesnimi, $synniaasta;
  if($kasEesnimi){$eesnimi=$eesnimi.$tekst;}
  if($kasSynniaasta){$synniaasta=$synniaasta.$tekst;}
}

$xml_parser = xml_parser_create();

xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characters");
$f=fopen("inimesed.xml", "r");
while($andmed=fread($f, 4096)){
  xml_parse($xml_parser, $andmed, feof($f));
}

xml_parser_free($xml_parser);


?>
```

Juku sündis 1963. aastal.
Juku sündis 1961. aastal.
Kalle sündis 1975. aastal.
Mari sündis 1981. aastal.
Oskar sündis 1971. aastal.

Ülesandeid

* Teata auto registreerimisnumbrite failist kõik need numbrid, mis lõppevad ühe või kahega.
* Koosta XML-fail autode registreemisnumbrite, markide ja väljalaskeaastatega. Moodusta SAXi abil SQL-laused nende andmete üle kandmiseks andmebaasi.

RSS näide

XML on lihtsalt põhireeglite kogum, mis aitab andmeid masinal kergemini leida ja töödelda. Põhiline XMLi kasutus leiab ikka aset XMLi peale ehitatud vormingus andmeid vahetades. Üheks selliseks levinud vorminguks on RSS. Iseenesest lihtne moodus teadete ülesmärkimiseks ja edasi saatmiseks. Kui järgnevat Eesti Päevalehest pärinevat näidet piiluda, siis on näha, et kõigepealt on kanali pealkiri ja kirjeldus ning siis tulevad üksikud teated. Standardi kinnitamine on vajalik selleks, et eri programmid teaksid samu asju samade nimede alt otsida. Ehk siis pealkirja jaoks on element nimega "title" ning kirjelduse jaoks "description". Hoolimata sellest, et kohustuslikke elemente on suhteliselt vähe, on RSSi kirjeldavas skeemis kirjas päris suur hulk omadusi, mida saab soovi korral sisse panna ning millele on ka nimed antud. Skeemi võib leida aadressilt

http://www.thearchitect.co.uk/schemas/rss-2_0.xsd

Järgnevalt lõik Eesti Päevalehe RSSist, mida võib lugeda aadressilt

http://epl.ee/rss/

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

<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
<channel>
<title>Eesti Päevaleht Online</title>
<copyright>Copyright (c) 2006 Eesti Päevalehe AS</copyright>
<link>http://www.epl.ee/</link>
<description>Päevalehe artiklite rss voog.</description>
<language>et</language>
<ttl>5</ttl>

<image>
<title>Eesti Päevaleht Online</title>
<link>http://www.epl.ee/</link>
<url>http://www.epl.ee/i/general/logo.gif</url>
<width>144</width>
<height>30</height>
</image>
<item>
<guid isPermaLink="true">http://www.epl.ee/artikkel/462863</guid>

<title>Vahtre: Koivisto puhul kõneles terve mõistus</title>
<link>http://www.epl.ee/artikkel/462863</link>
<pubDate>Sat, 21 Mar 2009 20:45:00 +0200</pubDate>
<author>Eesti Päevaleht Online <online@epl.ee></author>
<description> Ajaloos ei saa korraldada eksperimenti ja öelda, milline käitumine oleks õige või vale olnud, arvas ajaloolane ja riigikogu liige Lauri Vahtre kommenteerides Soome riigi käitumist Eesti taasiseseisvumise ajal.</description>

</item>

<item>
<guid isPermaLink="true">http://www.epl.ee/artikkel/462861</guid>
<title>Eesti tüdruk filmis kiskjakoera ja tibude sõpruse maailmakuulsaks</title>
<link>http://www.epl.ee/artikkel/462861</link>
<pubDate>Sat, 21 Mar 2009 19:59:00 +0200</pubDate>

<author>Eesti Päevaleht Online <online@epl.ee></author>
<description>Texases elav 29aastane Helen Arnold Jürlau on vändanud valmis koduvideo, mida Youtube'is on vaadatud enam kui viis miljonit korda.</description>
</item>

<item>
<guid isPermaLink="true">http://www.epl.ee/artikkel/462860</guid>
<title>Harrastuskaluritel jagub Peipsil püügilusti veel küllaga</title>

<link>http://www.epl.ee/artikkel/462860</link>

```
                <pubDate>Sat, 21 Mar 2009 19:16:00 +0200</pubDate>
                <author>Eesti Päevaleht Online &lt;online@epl.ee&gt;</author>
                <description>Elukutselistele kaluritele on talvine hooaeg Peipsil juba lõppenud, kuid harrastajatele jagub püügilusti veel küllaga - piirivalveameti prognoosi kohaselt peaksid jääolud lubama järvele minna veel paar nädalat.</description>
        </item>

        <item>
        <guid isPermaLink="true">http://www.epl.ee/artikkel/462778</guid>
          <title>Politsei unustas norralase Mercedese aastateks parklasse</title>
                <link>http://www.epl.ee/artikkel/462778</link>
                <pubDate>Sat, 21 Mar 2009 00:00:00 +0200</pubDate>

                <author>Mirko Ojakivi &lt;online@epl.ee&gt;</author>
                <description>Auto hoidmine tasulises parklas läheb riigile maksma sadu tuhandeid kroone.</description>
        </item>

  </channel>
</rss>
```

Et elementide nimed teada - title all peitub iga uudise pealkiri - siis saab SAXi abil uudisvoost suhteliselt kergesti oma lehele uudiste pealkirjad näidata. Tuleb nende sisu lihtsalt elemendi algul püüdma hakata, characters-käsu juures meelde jätta ning elemendi lõpus sobivana välja trükkida.

```
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
     <title>Uudised</title>
  </head>
  <body>
    <h1>Uudiste loetelu</h1>
    <ul>
    <?php
      $elemendiNimi="";
      $pealkiri="";
      function startElement($parser, $element, $atribuudid){
        global $elemendiNimi, $pealkiri;
        $elemendiNimi=$element;
        if($element=="TITLE"){$pealkiri="";}
      }
      function endElement($parser, $element){
        global $pealkiri;
        if($element=="TITLE"){
          echo "<li>$pealkiri</li>";
        }
      }
      function characters($parser, $tekst){
        global $elemendiNimi, $pealkiri;
```

```
        if($elemendiNimi=="TITLE"){
          $pealkiri.=$tekst;
        }
      }
      $xml_parser=xml_parser_create();
      xml_set_element_handler($xml_parser, "startElement", "endElement");
      xml_set_character_data_handler($xml_parser, "characters");
      $f=fopen("http://www.epl.ee/rss", "r");
      while($rida=fgets($f, 500)){
        xml_parse($xml_parser, $rida);
      }
      fclose($f);
      xml_parser_free($xml_parser);
    ?>
    </ul>
  </body>
</html>
```

Ülesandeid

* Tutvu RSSi lugemise näitega.
* Katseta muudest ajalehtedest ja kohtadest RSSi pealkirjade näitamist.
* Täienda näidet nõnda, et oleks näha pealkiri ja lühikirjeldus.
* Pealkirjale vajutades avatakse uudisega kaasas käiv viide.
* Juhul, kui pealkirja pikkus ületab 20 sümbolit, näidatakse vaid pealkirja algusotsa.

DOM

Mitemetes keeltes 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 Java 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().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.

```
E:\jaagup\09\09\xml\dom>java UusDokument > linnad.xml

E:\jaagup\09\09\xml\dom>start linnad.xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <linnad>
    <linn>Tallinn</linn>
    <linn>Tartu</linn>
    <linn>Narva</linn>
  </linnad>
```

Ülesandeid

* Väljasta DOMi abil XML-loeteluna kolme inimese eesnimed.
* Küsi nimed kasutajalt ja väljasta DOMi abil XML-loeteluna.
* Võimalda kasutajal valida, kas luuakse uus fail või lisatakse nimesid olemasolevasse faili juurde.
* Salvesta DOMi abil faili ühe inimese eesnimi, perekonnanimi ja sünniaasta.
* Küsi kasutajalt mitme inimese eesnimi, perekonnanimi ja sünniaasta ning salvesta nad DOMi abil faili.
* Koosta rakendusele graafiline liides. Võimalda inimeste andmeid lisada ning olemasolevaid muuta ja kustutada.

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.

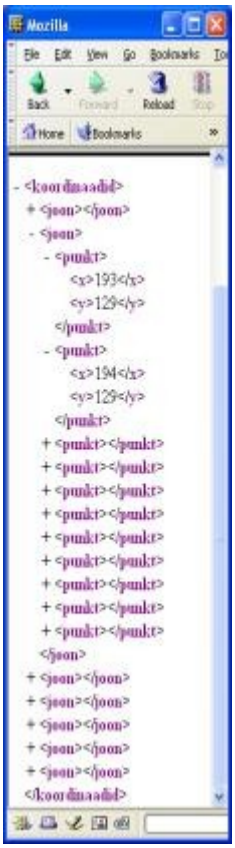
```
    if(e.getSource()==tryki){
       System.out.println(d.getFirstChild());}
```

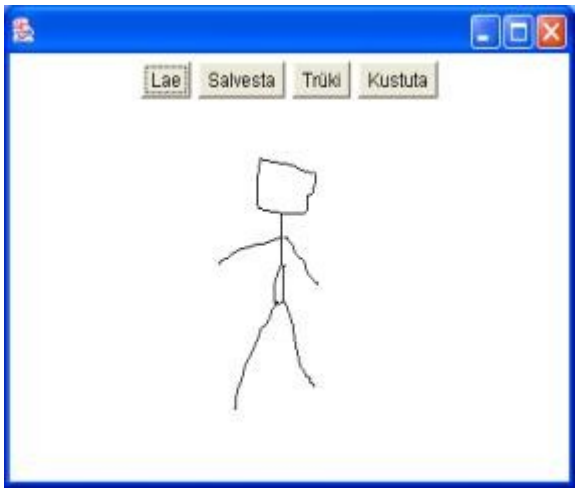
Ning rakenduse kood tervikuna.

```
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();
  }
}
```





Ülesandeid

* Tutvu joonistusrakendusega. Muuda tekstiredaktori abil koordinaatide väärtusi. Veendu tulemuse muutuses.
* Võimalda ühte joont märgistada. Märgistatuks osutub esimene joontest, mille kasvõi üks salvestatud punkt on hiirevajutusele lähemal kui kolm ekraanipunkti. Märgistatud joon tähistatakse punaselt.
* Märgistatud joont saab kustutada.
* Märgistatud joont saab soovitud suunas nihutada.
* Lisa kujunditena valikusse ning salvestusfaili ristkülik ja ring.
* Võimalda ka neid kustutada ning nihutada.

XMLil põhinevaid vorminguid

XML ise on vaid reeglite kogumik, mis võimaldab masinatel ja inimestel sama andmestikuga enamvähem mugavalt töötada. XMLi vormingu enese jaoks ei tähenda elementide nimed ja sisud midagi. Tegelikuks kasutamiseks on loojal ja lugejal vaja mõlemal teada, et mida miskis kohas

hoitakse - vaid siis on andmetega midagi peale hakata.

KML

Nime alla "Keyhole Markup Language" on kokku pandud XMLi reegleid järgiv keel, mille abil on võimalik kohti ja nende kirjeldusi märkida. KMLi võimalusi kirjeldav skeem on pikk mituteist lehekülge:

http://code.google.com/intl/et/apis/kml/schema/kml21.xsd

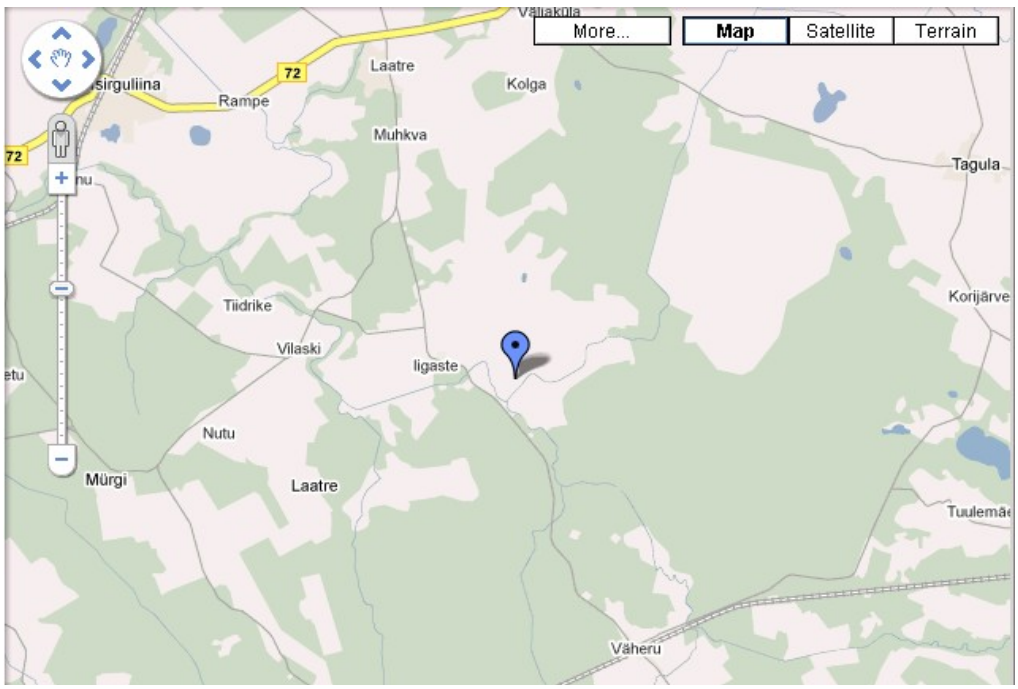
Enamiku sellest võtavad aga mitmesuguste tüübipiirangute ning valikuliste elementide kirjeldused. Lihtne asukoht koos nimetusega on võimalik edasi anda paari reaga. Järgnevalt ühe turismitalu lehel paiknev fail, mis märgistab talu asukoha.

http://www.kaldatalu.ee/pictures/placemark.kml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
<Placemark>
  <name>Kalda talu</name>
  <Point><coordinates>26.27867,57.83022</coordinates></Point>
</Placemark>
</Document>
</kml>
```

Tavainimene suudab siit silmadega välja lugeda koha nime ja koordinaadid. Kui rakendusel on vastavad oskused, siis saab ta ka sellega hakkama. KMLi mõistavad lugeda mitmesugused mobiiltelefonid. Samuti on Google kaardilehel teenus, mille abil võimalik veebis olemasolevat kaarti vabalt vaadata. Järgnevalt antakse teenusele ette eelnev KMLi fail ning soovitud suurendus.

http://maps.google.com/maps?q=http://www.kaldatalu.ee/pictures/placemark.kml&z=12



Selle peale näidatakse piirkonna kaarti koos markeriga.

Veidi juhtnuppe näppides võib muuta suurendust ning näidatava kaardi tüüpi ja vaadata küla satelliidifoto pealt. Marker ikka samas kohas, kuhu KML ta määras.

Markereid, teekondi ja teateid saab kujundada mitmesuguste stiilide kaudu. Järgnevalt on kõigepealt kirjeldatud väljakuikooni jaoks eraldi stiil ning siis see määratud Tallinna Lauluväljaku kohakirjelduse juurde. Elemendi description sees on CDATA-sektsioon puhuks, et seal saaks vabalt HTMLi koodi kasutada ilma muretsemata, et koodis tavalised < ja > märgid võiksid XMLi faili struktuuri segi ajada.

http://minitorn.tlu.ee/~jaagup/kool/java/kursused/08/sygisxml/naited/kml/lauluvaljak.kml

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
<Style id="valjakuikoon">
    <IconStyle>
      <Icon>
        <href>http://minitorn.tlu.ee/~jaagup/kool/java/kursused/08/sygisxml/nait
ed/kml/valjak.png</href>
      </Icon>
    </IconStyle>
</Style>
<Placemark>

  <name>Lauluvaljak</name>
  <description>
    <![CDATA[
      <a href="http://www.lauluvaljak.ee/">Tallinna lauluv&auml;ljak</a>
    ]]>
  </description>
  <styleUrl>#valjakuikoon</styleUrl>
  <Point>
    <coordinates>
      24.80360984802246,59.44493838137077
    </coordinates>
```

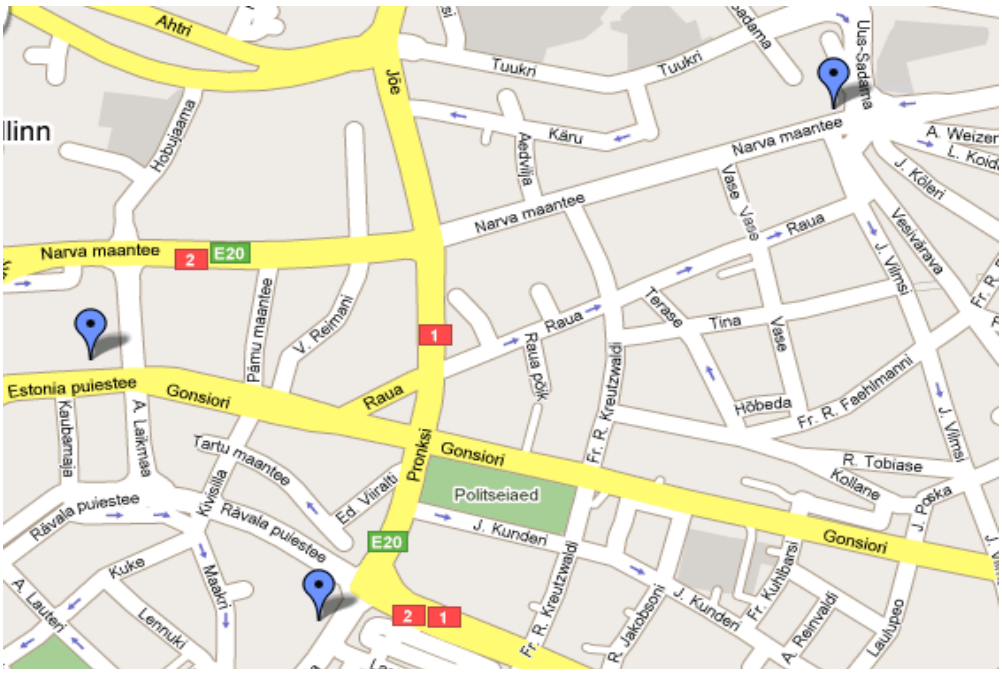
```
  </Point>
</Placemark>
</Document>
</kml>
```

Omajoonistatud ikoonike tuleb nõnda kergesti määratud kohta tähistama.

Asukohti võib olla kirjeldatud ka mitu. Sellisel juhul paigutatakse Placemark'id lihtsalt üksteise järele. Mõni piirdub vaid nimega, mõnele lisatakse kirjeldus.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Document>
<Placemark><name>kaubamaja</name><Point><coordinates>24.7564,59.4358</coordinate
s></Point></Placemark>
<Placemark><name>stockmann</name><Point><coordinates>24.76193,59.43233</coordina
tes></Point></Placemark>
<Placemark>
  <name>TLU</name>
  <description>Tallinna Ulikool</description>
  <Point>
    <coordinates>24.77442741394043,59.43913754833896</coordinates>
  </Point>
</Placemark>
</Document>
</kml>
```

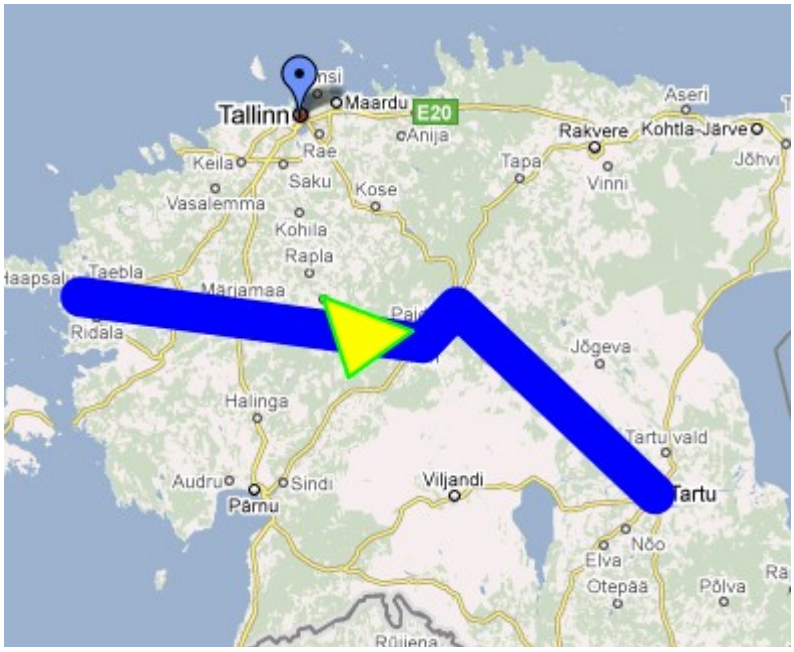
Google poolt pakutud kaardi peal võib nõnda kõiki neid kohti imetleda.



KMLi osa lõpetuseks ka näited teekonnajoone ning piirkonnapolügooni kohta. Stiili abil määratakse joone värvi ja paksust. Kasutusel samad värvikoodid mis HTMLigi juures. Esimene paar neljast on aga kasulik ff-iks määrata, siis pole muret, et joon läbipaistvaks ja seega nähtamatuks arvestataks. Joone puhul tuleb järjestikku üles tähendada selle punktid, polügooni puhul selle külgede punktid.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">

  <Document>
    <name>Kaart</name>
    <description>Proov piki Eestimaad</description>
    <Style id="blueLine">
      <LineStyle>
        <color>ffff0000</color>
        <width>20</width>
      </LineStyle>
    </Style>
    <Style id="rohelisejoonegakollane">
      <LineStyle>
        <width>1.5</width>
        <color>ff00ff00</color>
      </LineStyle>
      <PolyStyle>
        <color>ff00ffff</color>
      </PolyStyle>
    </Style>

  <Placemark>
    <name>Tallinn</name>
    <description>Suureks paisunud Eestimaa linn</description>
    <Point>
      <coordinates>24.7467041015625,59.42989896039298,0</coordinates>
    </Point>
  </Placemark>

    <Placemark>
      <name>Tartlase koolitee</name>
      <styleUrl>#blueLine</styleUrl>
```

```
      <LineString>
        <altitudeMode>relative</altitudeMode>
        <coordinates>
26.7022705078125,58.37455017815522,0
25.609130859375,58.90625330793137,0
25.42236328125,58.801128890372254,0
23.5491943359375,58.93177587724222,0
        </coordinates>
      </LineString>
    </Placemark>
    <Placemark>
      <name>Karjamaa</name>
      <styleUrl>#rohelisejoonegakollane</styleUrl>
      <Polygon>
        <extrude>1</extrude>
        <altitudeMode>relativeToGround</altitudeMode>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates>
24.8785400390625,58.9346105541337,0
25.0213623046875,58.70424642957549,0
25.3729248046875,58.838100868375264,0
24.8785400390625,58.9346105541337,0
            </coordinates>
          </LinearRing>
        </outerBoundaryIs>
      </Polygon>
    </Placemark>
  </Document>
</kml>
```



Ülesandeid

* Pane KMLi abil marker oma kodukohale. Koordinaatide teada saamiseks võib kasutada nt. lehte http://tigu.hk.tlu.ee/~jaagup/07/skriptkeeled/kaart2/kooliproov.html - hiirevajutuse peale näidatakse

valitud koha koordinaate. Pane fail avalikku veebi ning vaata markeriga kaarti http://maps.google.com/maps abil.
* Pane punktide abil kirja oma koolitee, vaata kaardil.
* Koosta väike matkamarsruut, punktide kirjelduste juurde märgi sinna jõudmise aeg, mõnele paigale lisa foto.