Tallinna Pedagoogikaülikool
Haapsalu Kolledž

Rakenduste programmeerimine

Jaagup Kippar

2004

Sisukord

Eessõna

Siinsesse konspekti koguti programmeerimise ja Javaga seotud teemad, mis oma pikkuse tõttu ei sobinud sama autori koostatud Java põhikursuse konspekti ning teemavaliku poolest ei kuulunud omaette konspektiks kirjutatud graafika või muusika programmeerimise alla. Mis aga on samas piisavalt tähtsad, et neist maakeelne tutvustav ülevaade anda. Ehkki koostamise aluseks oli kriteerium "kõik, mis muust üle jääb", leiab ka siit sisukorrast mõned suuremad.

Tähtsamaks ja tõenäoliselt enam kasutatavamaks osaks võiks olla relatsiooniliste andmebaasidega seonduv. Aastakümnel, kus üha enam teateid ja dokumente liigub arvuti kaudu, vajatakse ka taoliste süsteemide loojaid ja ülalpidajaid. Ning ehkki vähemasti kümmekond aastat juba räägitakse, et relatsioonilised baasid ja objektorienteeritud programmeerimismudel on omavahel vastuolus, tundub selline tava vähemasti mõnda aega veel püsima ning vähemalt osa praegu loodud süsteeme töötab ka veel aastate pärast nagu seniste kogemuste varal arvata võib.

Teiseks suuremaks teemaks on Java 2 Micro Edition. Siinsed näited koostati mobiiltelefonide emulaatoreid kasutades, kuid samade vahenditega saab programmeerida ka pihuarvuteid ning muidki Javat toetavaid miniseadmeid. Ehkki tehnoloogia juba mitme aasta vanune, sai teema siia konspekti lisamisel otsustavaks J2ME suhteliselt kiire areng 2003. aastal ning oodatav miniseadmete leviku kasv.

XMLile ja Unicodele on kuulutatud suurt võidukäiku vähemasti 1997ndast aastast alates. Järske imesid pole sündinud, kuid standardid on levinud ning nende põhjal loodud rakendused usaldusväärsemateks muutunud. Ning kui on vaja struktuurseid andmeid nii inimesele kui masinale mõistetavasse vormingusse paigutada (näiteks konfiguratsioonifaili puhul), siis peab päris tugev seletus olema, kui tahetakse põhjendada, miks nende andmete hoidmiseks just XML-vormingut ei kasutatud. Vaadatakse läbi levinumad XMLi programmse töötlemise moodused: SAX ning DOM. Esimene mahukatest dokumentidest üksikute väärtuste eraldamiseks, teine andmepuu loomiseks, muutmiseks ja põhjalikumaks analüüsiks.

Hajutatud rakendused on omaette suurem maailm. Nendest peetakse vähemasti nii Tartu Ülikoolis kui Tehnikaülikoolis omaette kursusi. Siin on vaadeldud tehnilisi lahendusi, mille kaudu õnnestub hajusalt paiknevalt osad omavahel suhtlema panna. J2EE serveriga tutvutakse vaid Sun-i näidiskeskkonna abil, kuid siin saadud kogemusi meenutades peaks mujalt loetava materjali külge olema kergem haakuda.

Läbi vaadatakse ka programmeerimiskursustes traditsioonilised teemad: nimistu ja andmepuu, samuti bititöötlus. Nende teemade juures ei püüta pakkuda midagi uut, küll aga peaks olema tegemist kõlbuliku lugemismaterjaliga inimesele, kes Java vahenditega ümber käies on jõudnud kaugusele, kus vastavad toimingud arusaadavaks ja tarvilikuks muutuvad. Samuti õnnestub nende peatükkide omandamise järel ehk paremini mõista muidki puukujuliste andmetega tegelevaid algoritme. Olgu siis tegemist failipuuga kettal või XMLi puuga mälus.

Paarileheküljeliste lõikude kaudu tutvutakse mitmete Java tehniliste võimalustega. Koodi jagamine pakettidesse aitab suuremate rakenduste puhul seda kergemini hallata ning eri loojate koodi ühendada. Arhiveerimine aitab lihtsalt ruumi kokku hoida ning mõnikord ka pildi selgemaks muuta. Klasside käskluste uurimine programmi abil või eksemplaride loomine klassi nime järgi võib tunduda imelik, kuid sealtkaudu õnnestub näiteks automaatselt mõningaid võimalusi dokumenteerida või saada üle kitsaskohtadest, mille poolest kompileeritavad keeled kipuvad interpreteeritavatele alla jääma.

Viimase peatükina paigutatud tarkvara hindamine ei sisalda kuigivõrd koodinäiteid ega seletusi, kuid seal kirjutatust võib kasu olla nii oma rakenduse kavandamisel, töökindluse hindamisel ja parandamisel. Kõiki häid mõtteid, võimalusi ja tavasid ei jõua alati korraga oma programmis rakendada, aga vahel proovida ikka tasub.

Jõudu!

Jaagup Kippar

Andmebaasid

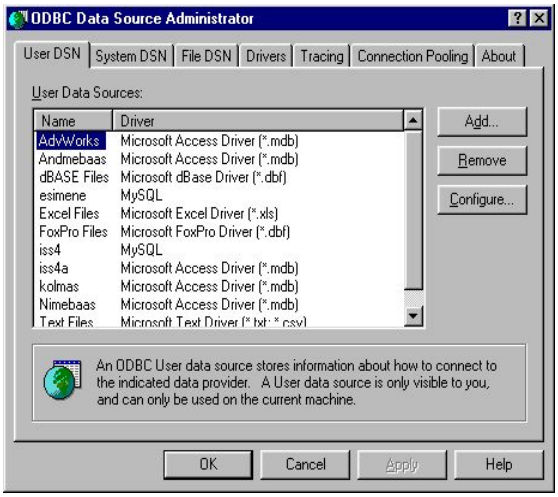
SQL, ODBC, Servlet, veebirakendus

Enamik programme talletab andmeid kusagil. Üheks levinumaks väljundiks programmi poolt vaadates on failid kettal, teiseks andmebaasid. Failide eeliseks on kohene kasutamisvalmidus. Baasiga suhtlemise puhul peab lisaks oma rakendusele ka andmebaasiga suhtlemist võimaldav vahend masinas leiduma. Peaaegu hädavajalikuks aga muutub andmebaas mitmelõimelise programmi korral, sest ise korralikke lukustusmehhanisme kirjutada võib olla päris aeganõudev ettevõtmine. Samuti aitavad andmebaaside päringuvahendid andmete keerukama ülesehituse korral sobivaid väärtusi üles leida. Andmemudeleid ja päringukeeli leidub mitte. 2004. aastal ja sellele eelnenenud paaril aastakümnel on aga valitsevaks relatsioonilised, tabelitest koosnevad andmebaasid ning baasidega suhtlemiseks SQL-keel.

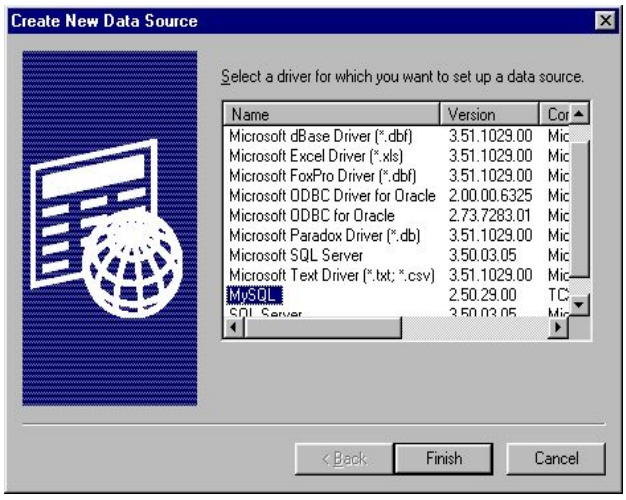
Andmebaasiühenduse loomine

Et andmeid ja tabeleid saaks kuhugi baasi paigutada, selleks peab kõigepealt andmebaas loodud olema. Mõnes paigas saab seda teha vaid administraatoriõigustes inimene, Windowsi masinates võib aga sageli igaüks omale kasutamiseks-katsetamiseks andmebaasiserveri püsti panna ning sinna nii palju baase luua kui kettamaht võimaldab. Enesele MySQLi vahendid kopeerida saab http://www.mysql.com/-i alt. Vaja läheb nii andmebaasikeskkonda ennast kui draiverit sellega ühendumiseks.

Toimingute tegemiseks peab kõigepealt baasiserveri käima lükkama. Käsuks mysqld-shareware ning selle tulemusena hakkab server kuulama väratit 3306, kui pole määratud teisiti. Käsk mysqladmin lubab luua ja kaotada baase ning teha muudki korraldusega seonduvat. Siin luuakse baas nimega pood.

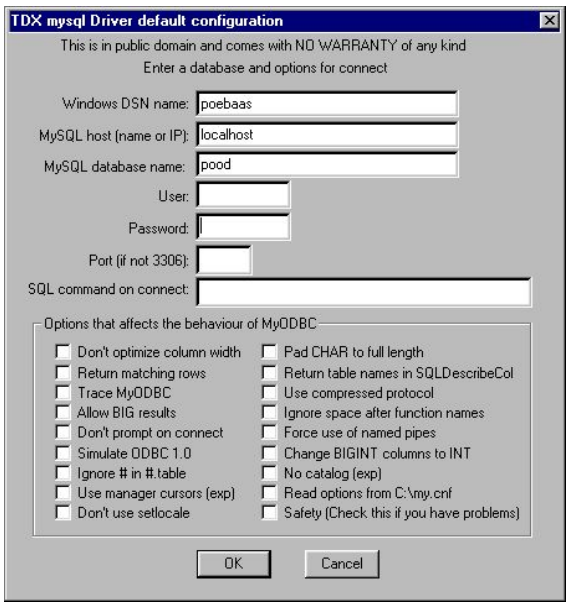


Kui aga soovida lihtsamalt läbi ajada, siis ei pea selleks mitte oma serverit püsti panema. Võib rahulikult toime tulla olemasolevate Accessi, Exceli või suisa tekstifaili draiveritega. Kui aga MySQL installeeritud, siis saab seda kasutada. Küllaltki universaalne koht andmebaasidele ligi pääsemiseks on ControlPanel'i alt avanev ODBC. Et MySQLile sealtkaudu ligi pääseks, on vaja installeerida vastav draiver, näiteks Connector/ODBC, mis vabalt kättesaadava MySQLi kodulehelt. Sealt ControlPanel'i alt on näha, millised ressursid juba kasutada on, samuti annab siit oma baasile ODBC ühendus luua, mille abil siis kergesti võib olemasolevate draiverite abil programmide kaudu sinna andmeid saatma ja sealt pärima hakata.

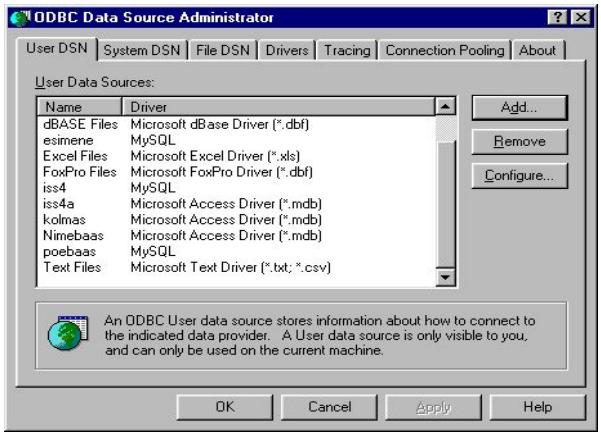


Nimetatud vahelüli (Open DataBase Connectivity) on lihtsalt ühine protokoll, mille kaudu saavad suhelda osapooled, kes üksteise keelt ei tunne (pole otseühenduseks vastavaid draivereid).

Uue andmeallika loomiseks tuleb vajutada Add... ning pakutakse toetatavatest tüüpidest välja, millist kasutaja soovib luua. Kui siin näites soovime ühenduda MySQL-i baasiga, tuleb ka vastavat tüüpi draiver valida.



Draiveri juures tuleb määrata parameetrid. Vähemasti nimi, mille alt Windows'is vastavat andmeallikat tuntakse ning milline on tegelik baasi nimi serveri



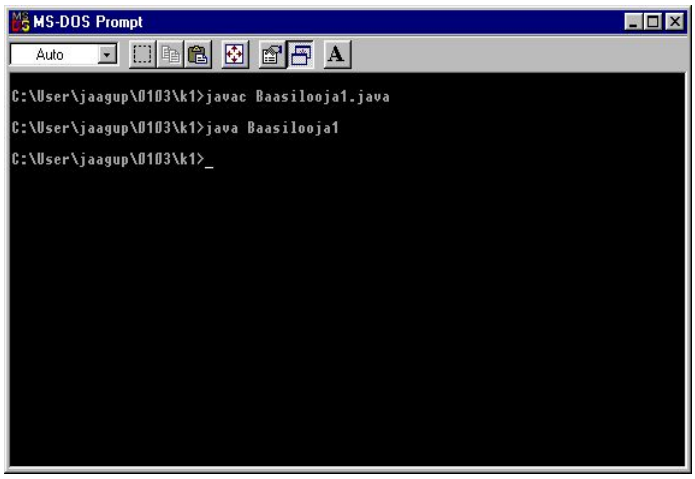
Kui kogu loomine läks õnnelikult, siis jõuame tagasi algse lehe juurde, kuhu on tekkinud ka vastloodud ühendus, siin näites nime all poebaas.

Edasi pole muud, kui asuda loodud ühendust kasutama. Baasi sisse võib tabeleid ja andmeid lisada mitut moodi. Accessi või Exceli puhul saab avada vastava programmi ning rahumeeli tähed ja numbrid tabelisse kirjutada. MySQLil oma kliendi kaudu saab ka baasi külge ühenduda ning seal SQL lausete abil soovitud muutusi tekitada. Kui tahta edaspidi panna oma programm baasi andmeid kasutama, siis on paslik alustada lühemast käsureast, mis parajasti baasi sisse ühe üheveerulise tabeli loob ning

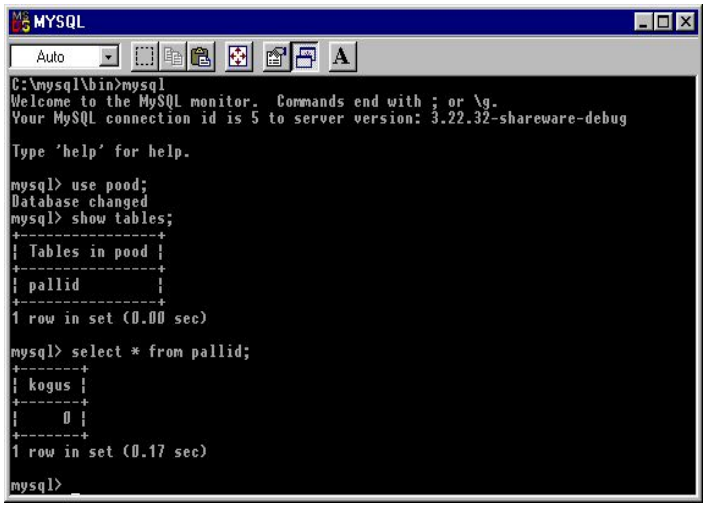
sinna sisse väärtuse paigutab. Võib ette kujutada, et tabelis on kirjas, mitu palli parajasti poe laos hoiul on.

Andmebaasiga suhtlemiseks tuleb kõigepealt mällu laadida draiver. ODBC tarvis on Java starndardkomplektis kaasas `sun.jdbc.odbc.JdbcOdbcDriver`. Luuakse ühendus, jättes kasutajanime ja parooli koht tühjaks, kuna meie katsebaasi puhul neid ei nõuta. Saadetakse käsklaused teele ning suletakse ühendus.

```
import java.sql.*;
public class Baasilooja1{
  public static void main(String argumendid[]) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=
       DriverManager.getConnection(
                     "jdbc:odbc:poebaas", "", "");
    Statement st=cn.createStatement();
    String lause="CREATE TABLE pallid (kogus int);";
    st.executeUpdate(lause);
    lause="INSERT INTO pallid (kogus) values ('0');";
    st.executeUpdate(lause);
    cn.close();
  }
}
```



Kui programmi tekst sisse kirjutatud, siis enne käivitamist tuleb see kompileerida ning seejärel käima lasta. Näidet vaadates paistab tulemus tühjavõitu olema. Kompileerimisel ei tulnud veateadet seetõttu, et ühtki viga ei leitud. Käivitamisel pole midagi näha, kuna kogu tegevus käis programmi ja baasi vahel ning polnud küsitud, et midagi ekraanile näidataks. Kui väljatrükilauseid vahele pikkida, eks siis oleks ka käivitajal rohkem midagi vaadata olnud.



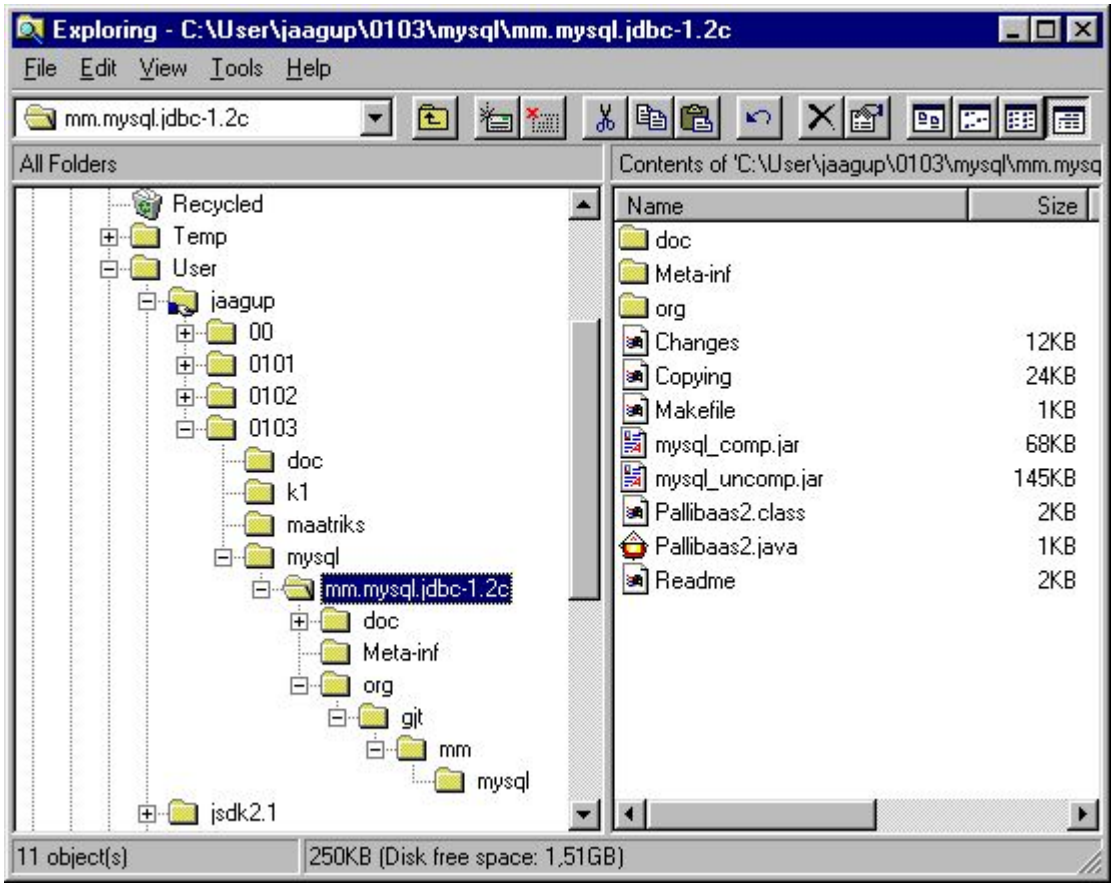
Et töö siiski päris tühi ei olnud ja midagi ka toimus, sellest annab teada järgmine pilt. Kui ühenduda MySQLi kliendiga baasi taha ning uurida, mis seal sees paikneb, siis on näha, et tekkinud on tabel nimega pallid ning sinna sisse on koguseks pandud 0.

Otsene draiver

ODBC võimaldab omavahel suhelda paljudel programmidel ning protokollidel, kuid selle puuduste juurde kuulub, et tegemist on veel ühe järjekordse vahelüliga, mis enesele ressursse nõuab ning nagu pudelikael ikka ei pruugi see mitte kõiki häid omadusi läbi lasta, mis kummalgi osapoolel olemas võivad olla. Sellepärast, kui on tegemist tohutute andmehulkade või suurte kiirustega, on mõistlik otsida programmi ja andmebaasi vahele otsest draiverit. Javat ning MySQLi ühendava vahendi leiab näiteks lehelt

http://www.mysql.com/downloads/api-jdbc.html

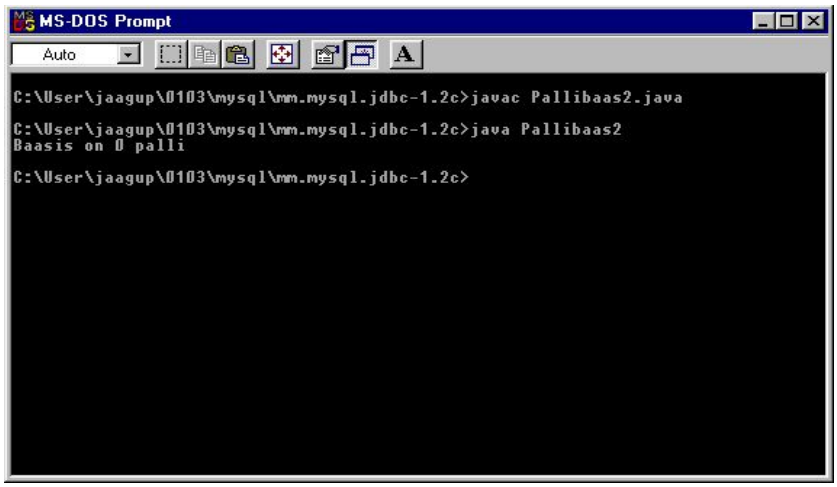
Kui sealne arhiiv maha laadida ning lahti pakkida, tekkis kataloog, milles nii draiver ise kui hulga õpetusi, kuidas temaga ümber käia.



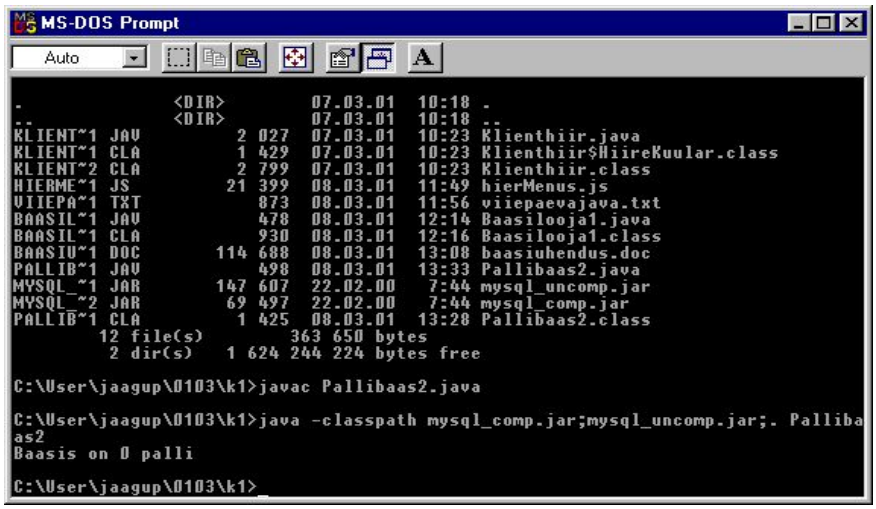
Kirjadest selgus, et muuta polegi vaja muud kui draiveri nime ning ühenduse URLi. Nüüd saab kirjutada otse jdbc:mysql: .

```
import java.sql.*;
public class Pallibaas2{
  public static void main(String argumendid[]) throws Exception{
    Class.forName("org.gjt.mm.mysql.Driver");
    Connection cn=DriverManager.getConnection(
               "jdbc:mysql:/localhost/pood", "", "");
    Statement st=cn.createStatement();
    String lause="SELECT kogus FROM pallid;";
    ResultSet rs=st.executeQuery(lause);
    rs.next();
    System.out.println("Baasis on "+
              rs.getInt("kogus")+" palli");
    cn.close();
  }
}
```

Kui programm panna tööle draiveri kodukataloogis, siis leitakse ise kõik sobivad klassid üles, sest nad on seal lihtsalt käe-jala juures.



Soovides aga kohaleveetud draiverit kusagil mujal kasutada, selleks tuleb draiveri klassid arhiividena kaasa võtta ning käivitamisel –classpath abil öelda, millistest arhiividest draiveri osad kokku korjata tuleb.



Servlet

Baasis paiknevaid andmeid võib vaja olla mitmele poole välja anda. Veebi kaudu on hea andmeid lugeda ning sinna saatmiseks sobivad servletid ehk pisiprogrammikesed veebiserveris. Nagu varsti näha, võime soovi korral oma arvutisse HTTP-serveri püsti panna ning servletid andmebaasi andmeid lugema saata. Alustada võiks aga lihtsa servleti loomisest ja käivitamisest. Ja seletusest, et millega tegu.

Java programme käivitatakse päris mitmesugustes paikades. Algseks ja "õigeks" käivitamiskohaks võidakse pidada ju main-meetodit, kuid võimalikke Java-programmide käivituskohti on tunduvalt enam. Rakendid veebilehtedel saavad käiturilt teateid sündmuste kohta ning toimivad vastavalt nendele. Rakendusserveris paiknevad EJB-nimelised komponendid ootavad aga hoopis teistsuguste käskude käivitamist. Ning miniseadmetes toimivad J2ME programmid ärkavad jälle omamoodi.

Servlettide praegusaja levinumaks kasutusalaks on veebilehtede väljastamine – kuid mitte ainult. Mitmesugused masinatevahelised teated ning teenused töötavad samuti servlettide kaudu. Kui kord loodud mugav ning suhteliselt turvaline ja kontrollitav võimalus teisest masinast andmete küsimiseks, siis võib seda ju kasutada. Sestap võibki näha servlettide päises kahe paketi importimist: javax.servlet ning javax.servlet.http. Viimane siis HTTP-vahenditega lähemalt seotud klasside tarbeks.

Sarnaselt rakendile võetakse ka servlettide puhul aluseks ülemklass ning asutakse selle meetodeid üle katma. Vaid toimingud on rakendi või mobiiliprogrammiga võrreldes teistsugused, ülekatmine ikka samasugune. Erisuseks veel, et servleti puhul iga lehe avamine piirdub funktsiooni ühekordse väljakutsega. Rakendi puhul võivad start, stop ning paint korduvalt käivituda.

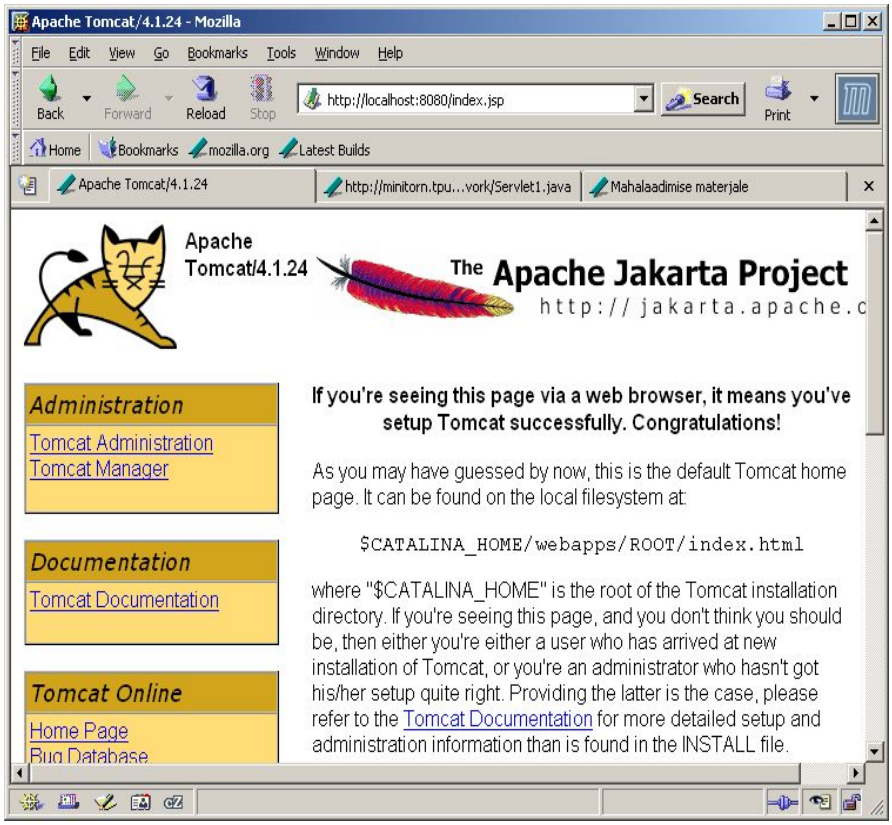
HTTP-päringute puhul on võimalike toiminguid vähemasti kuus, kuid servlettide puhul levinumateks GET ning POST, mõlemal juhul väljastatakse üldjuhul veebileht. GET-päringu puhul antakse parameetrid kaasa URLi real, nende pikkus on piiratum ning loodud leht võidakse kergemini puhverdada. Tüüpiline kasutusvaldkond on näiteks otsingumootorite juures, kus sama päringu tulemus minutite ja tundide jooksul oluliselt ei muutu.

Kui tegemist andmete sisestamisega – näiteks enese võrgu kaudu registreerimisega, siis tuleb paratamatult programm igal korral uuesti käima panna ning selleks kasutatakse POST-nimelist meetodit. Tegemise ajal katsetada on aga GET-i puhul mugavam, sest siis paistavad saadetavad andmed välja. GET-meetodi käivitamiseks tuleb üle katta servleti meetod doGet. Meetodile antud esimese parameetri kaudu saab andmeid päringu kohta: milliselt aadressilt ja masinast tuldi, millised andmed kasutaja kaasa saatis. Teine parameeter tüübist HttpServletResponse võimaldab määrata loodava lehe sisu ning päised.

Järgnevalt näha võimalikult lihtne tervitav servlet.

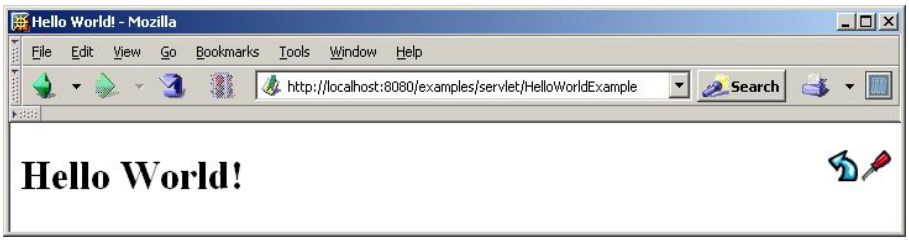
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Servlet1 extends HttpServlet{
  public void doGet(HttpServletRequest kysimus, HttpServletResponse vastus)
              throws IOException, ServletException{
    PrintWriter valja=vastus.getWriter();
    valja.println("Tervist!");
  }
}
```

Enne tulemuse nägemist tuleb veel pingutada servletile sobiva keskkonna loomise nimel. Kel juba sobiv käivitusserver eelnevalt püsti, sel võib piisata faili kompileerimisest ning sobivasse kataloogi paigutamisest. Kel aga mitte, siis tuleb veidi installeerimisega pead vaevata. 2004. aastal tundub levinud servletikäituriks olevat näiteks Apache Tomcati nimeline veebiserver http://jakarta.apache.org/tomcat/. Sealt allalaetud faili lahtipakkimisel või käivitamisel saab õnnelike juhuste kokkulangemisel tööle oma masinas veebiserveri. Täpsemaid seadistamise juhiseid leiab näiteks aadressilt http://www.coreservlets.com/Apache-Tomcat-Tutorial/.

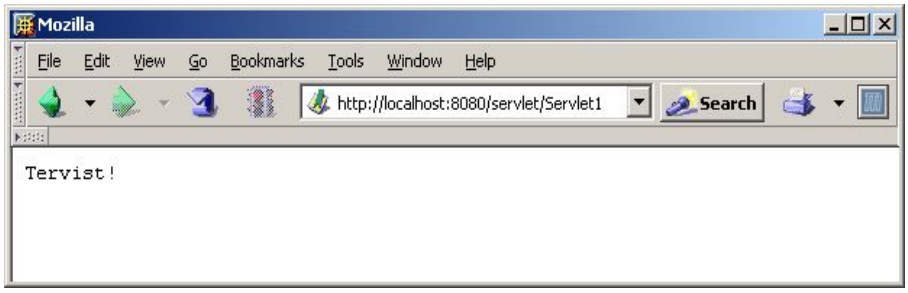


Loodud koodi kompileerimiseks peavad kättesaadavad olema servlettide alusklassid. Need leiab Tomcati installeerimiskataloogi alamkataloogist `common\lib\` . Tomcati 4. versiooni puhul on failiks servlet.jar, viienda versiooni puhul servlet-api.jar. Neid võib kättesaadavaks teha CLASSPATH-nimelise muutuja kaudu. Teiseks võimaluseks on aga kopeerida nimetatud fail java interpretaatori laienduste

kataloogi, milleks siinses masinas on näiteks C:\j2sdk1.4.2_01\jre\lib\ext, mujal siis vastavalt Java installeerimise asukohale. Edasi võib koodi kompileerida nagu tavalist Java faili. Üheks mugavaks käivitamise kohaks on asukoht Tomcati enese näidete juures nt. C:\Program Files\Apache Group\Tomcat 4.1\webapps\examples\WEB-INF\classes , kuid konfiguratsioonifailide abil saab siin paljutki sättida. Kaasatulnud näited saab käivitada aadressireal examples/servlet-kataloogi kaudu.



Mõningase nikerdamise tulemusena võib aga servletid ka juurkataloogis oleva servlet-kataloogi all tööle lükata.



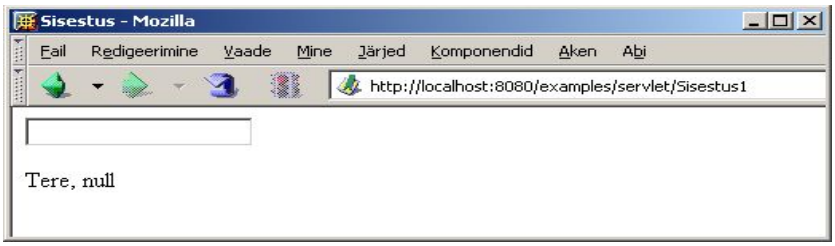
Sisestus

Kui soovida programmilt vastuseid omapoolsetele andmetele, siis tuleb need kuidagi ka programmile ette anda. Servlettide puhul sisestab kasutaja enamasti andmed veebilehel paiknevasse tekstivälja või muusse sisestuskomponenti. Andmete sisestusnupule vajutamisel jõuavad need paremeetritena järgmisena avatava veebilehte loova programmi kasutusse ning edasi tuleb juba seal otsustada, mida saadud andmetega peale hakatakse. Igal andmeid edastaval sisestuskomponendil sõltumata tüübist on nimi, järnevas näites näiteks “eesnimi”. Andmeid vastuvõttev programm saab selle nime järgi küsida just konkreetse elemendi väärtust.

Siin pole eraldi määratud, kuhu faili andmeid saata. Sel juhul käivitatakse uuesti sama fail ning uue ringi peal jõuavad eelmisel korral sisestatud andmed kohale.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sisestus1 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
        vastus.setContentType("text/html");
        PrintWriter valja = vastus.getWriter();
        valja.println("<html>"+
           "<head><title>Sisestus</title></head>\n"+
           "<body><form><input type='text' name='eesnimi' />"+
           "</form>");
        valja.println("Tere, "+ kysimus.getParameter("eesnimi"));
        valja.println("</body></html>");
    }
}
```

Esimesel korral aga pole veel andmeid kusagilt võtta ning kysimus.getParameter annab vastuseks tühiväärtuse null.

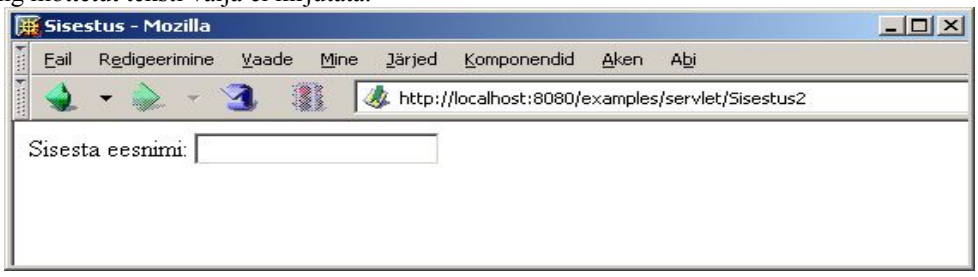


Kui nüüd tekstivälja sisse nimi kirjutada ning sisestusklahvile vajutada, siis võib järgmisel ringil näha, et nimi jõudis avanevale lehele kohale. Lehe keskel ilutseb rõõmsasti "Tere, Juku". Kui tähelepanelikumalt piiluda, siis võib märgata, et programmi nime taga aadressireal paikneb küsimärk ning selle järel sisestatud parameetri nimi ja võrdusmärgi taga väärtus. Sealtkaudu on liikuvad andmed programmeerijale ilusti näha ning tal võimalus kontrollida, mis ja millise nime all serverisse saadeti.

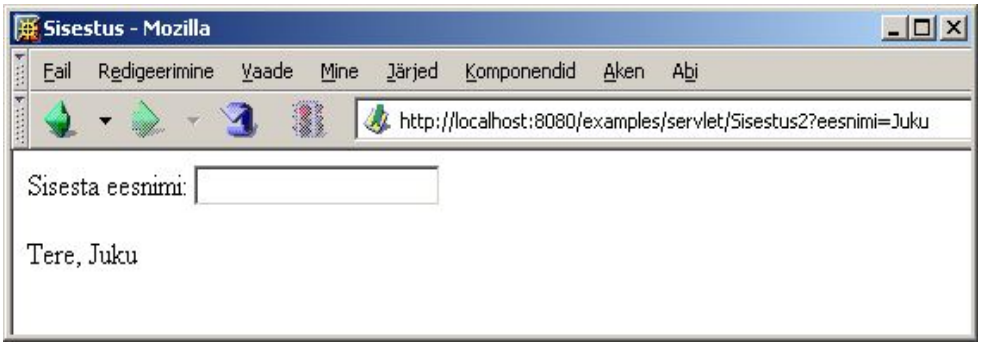
Esmakordselt näidatavast tühiväärtusest on täiesti võimalik hoiduda. Selleks tuleb enne nime välja trükkimist kontrollida, kas ikka midagi teele saadeti. Andmed loeti eesnime-nimelisse muutujasse kahel põhjusel. Lühema nimega muutujaga on lihtsalt kergem ümber käia kui pidevalt parameetrit käskluse kaudu küsides. Samuti võib juhtuda, et kui kord mõni parameeter Request-i käest küsitud, siis võidakse arvata, et selle väärtus juba programmeerijal teada on ning seda rohkem enam algses kohas ei säilitata. Tugevamalt tuleb sellise kadumisvõimalusega arvestada andmebaaside juures mõne draiveri ja seadistuse puhul, kuid ka siin on tegemist sama nähtusega.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sisestus2 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
        vastus.setContentType("text/html");
        PrintWriter valja = vastus.getWriter();
        valja.println("<html>"+
           "<head><title>Sisestus</title></head>\n"+
           "<body><form>Sisesta eesnimi: "+
           "<input type='text' name='eesnimi' />"+
           "</form>");
        String eesnimi=kysimus.getParameter("eesnimi");
        if(eesnimi!=null){
          valja.println("Tere, "+eesnimi);
        }
        valja.println("</body></html>");
    }
}
```

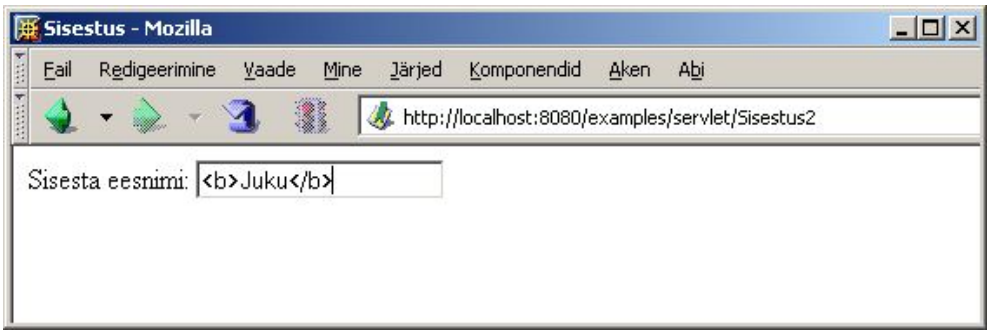
Kui nüüd kontrollitakse enne väljatrükki tühiväärtust, siis võib näha, et lehe vastuseosa on ilusti tühi ning mõttetut teksti välja ei kirjutata.



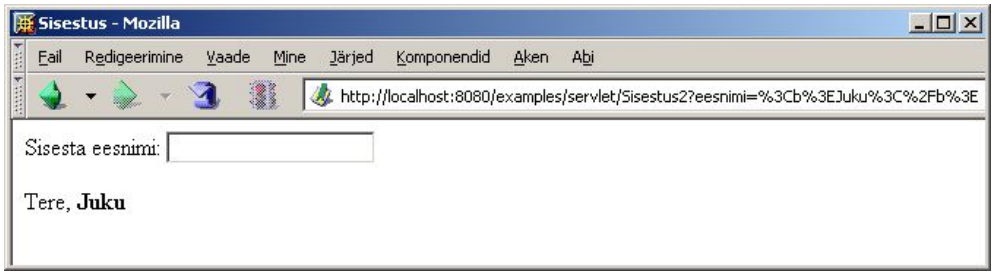
Viisakalt sisse kirjutatud nime puhul aga tervitatakse sama rõõmsasti vastu.



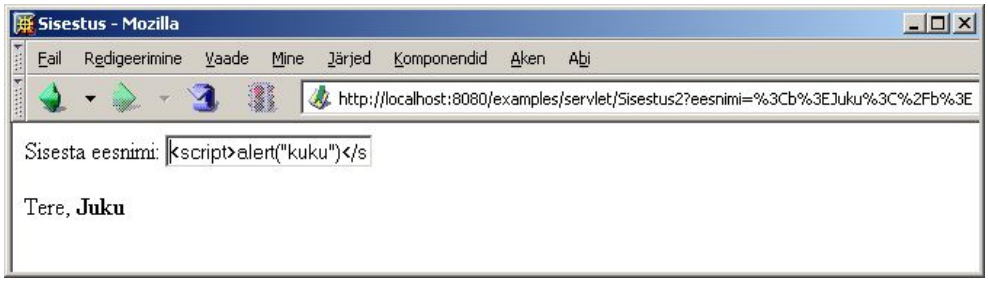
Räägitakse, et veebilehtede koostamisel tuleb arvestada igasuguste turvaprobleemidega. Ning et üheks märgatavaks ohuks on kasutajate nii kogemata kui meelega sisestatud erisümbolid. Lihtsamatel juhtudel võidakse kirjutada HTML-i kujunduskäsklusi oma teksti ilmestamiseks.



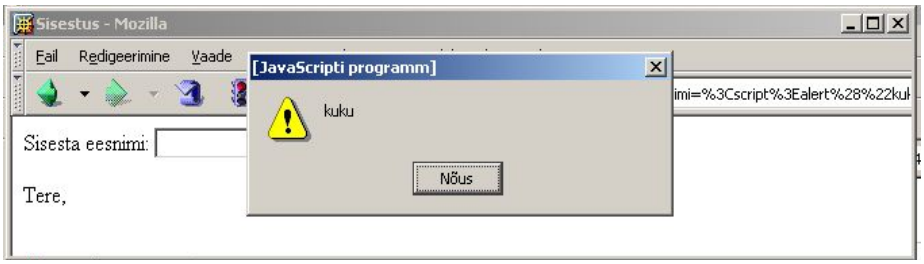
Tulemus võib sellisel juhul päris meediv olla.



Samas ei takista miski ka Javaskripti koodilõike teksti sisse kirjutamast ning nende tööga ei pruugi kasutaja enam rahul olla.



Näiteks praegusel juhul avatakse teateaken. Kui selliseid akent avavaid teateid aga mõnda külalisraamatusse hulgem saab, siis võib lehekülje avamine päris vaevaliseks osutuda.



Suuremaks probleemiks siinjuures on, et lehel toimetav Javaskript võib ka näiteks kasutaja sisestatud andmed oma kontrolli alla saada ning hoopis võõrasse serverisse teele saata, mille üle kasutaja sugugi rõõmus ei pruugi olla. Samuti võib juhtuda, et üksik valesse kohta sisestatud < või “-märk tekitab seilris sedavõrra segadust, et järgnev tekst jääb sootuks näitamata või muutub keerulisema paigutusega lehel pilt ees segaseks.

Kui kasutaja sisestatud erisümbolid HTML-i reeglitele vastavalt kodeerida, siis pääseb eelpool kirjeldatud muredest. Et kodeerimist läheb vaja siinsest näitest tunduvalt rohkemates paikades, siis sai abifunktsioon paigutatud omaette klassi. Staatilise funktsiooni saab kättesaadava klassi kaudu kohe käima tõmmata, ilma et peaks objekti loomisele jõudu kulutama. Tähtede asendamiseks on kasutatud StringBuffer-tüüpi objekti, kuna puhvrile liitmine on tunduvalt odavam tegevus kui sõnede liitmine. Eriti juhul, kui tekstid kipuvad pikemaks minema. Sest kord valmis loodud Stringi enam muuta ei saa. Tähe lisamiseks tuleb uus mälupiirkond leida ning algsed andmed sinna üle kopeerida. StringBuffer on aga siinkirjeldatud toiminguteks just loodud.

```
public class Abi{
  /**
  * Etteantud tekstis asendatakse HTML-i erisümbolid
  * lehele sobivate kombinatsioonidega.
  */
  public static String filtreeriHTML(String tekst){
     if(tekst==null){return null;}
     StringBuffer puhver=new StringBuffer();
     for(int i=0; i<tekst.length(); i++){
       char c=tekst.charAt(i);
       switch(c){
         case '<': puhver.append("&lt;"); break;
         case '>': puhver.append("&gt;"); break;
         case '&': puhver.append("&amp;"); break;
         case '"': puhver.append("&quot;"); break;
         default: puhver.append(c);
       }
     }
     return puhver.toString();
  }
}
```

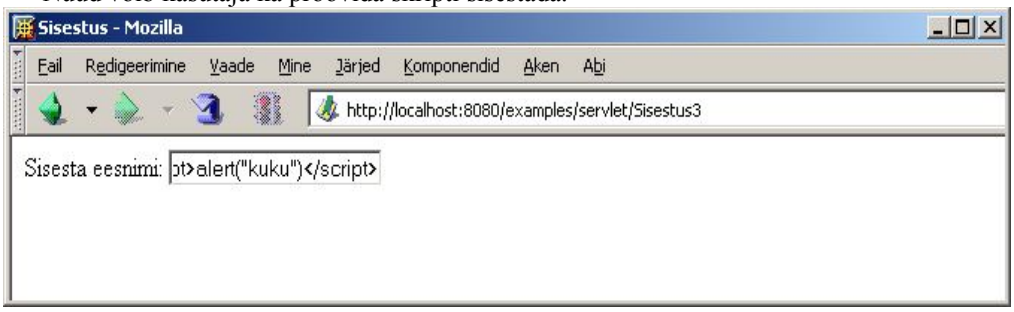
Andmete filtreerimiseks piisab järgnevast käsust.

```
        String eesnimi=Abi.filtreeriHTML(kysimus.getParameter("eesnimi"));
```

Kui klass Abi asub käivituva servletiga samas kataloogis, siis leitakse klass üles.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sisestus3 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
        vastus.setContentType("text/html");
        PrintWriter valja = vastus.getWriter();
        valja.println("<html>"+
           "<head><title>Sisestus</title></head>\n"+
           "<body><form>Sisesta eesnimi: "+
           "<input type='text' name='eesnimi' />"+
           "</form>");
        String eesnimi=Abi.filtreeriHTML(kysimus.getParameter("eesnimi"));
        if(eesnimi!=null){
          valja.println("Tere, "+eesnimi);
        }
        valja.println("</body></html>");
    }
}
```

Nüüd võib kasutaja ka proovida skripti sisestada.



Tulemusena aga asendatase tekst ära ning lehele jõuab sarnane pilt kui kasutaja kirjutatugi.

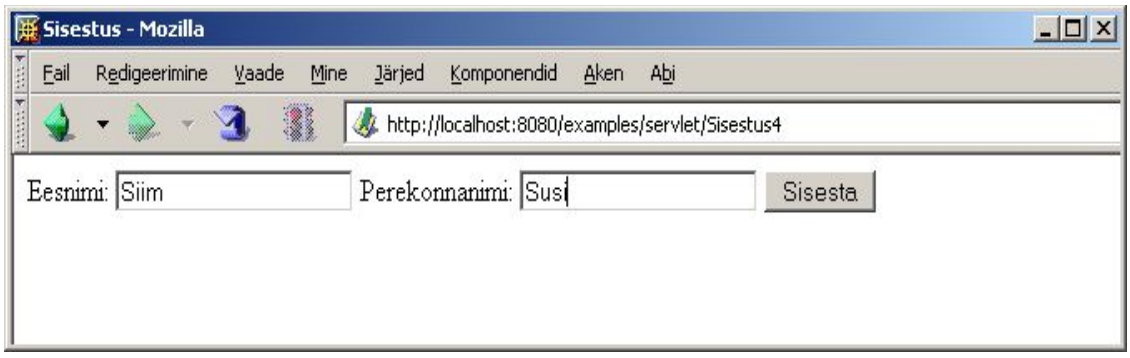
Kui tahta programmi muundamistööd täpsemalt piiluda, siis tulemuse leiab lehe lähtekoodi vaadates.

```
<html><head><title>Sisestus</title></head>
<body><form>Sisesta eesnimi: <input type='text' name='eesnimi' /></form>
Tere, &lt;script&gt;alert(&quot;kuku&quot;)&lt;/script&gt;
</body></html>
```

Enamasti sisestatakse lehele rohkem kui üks väärtus. Kui ühe teksti puhul piisas tekstiväljast ning sisestusklahvile vajutades läksid andmed teele, siis rohkemate saadetavate andmete puhul on lisaks vaja ka sisestusnuppu. Selleks siis sisestusväli tüübiga “submit”.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Sisestus4 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
        vastus.setContentType("text/html");
        PrintWriter valja = vastus.getWriter();
        valja.println("<html>"+
           "<head><title>Sisestus</title></head>\n"+
           "<body><form>"+
           "Eesnimi: <input type='text' name='eesnimi' />\n"+
           "Perekonnanimi: <input type='text' name='perenimi' />\n"+
           "<input type='submit' value='Sisesta' />"+
           "</form>");
        String eesnimi=Abi.filtreeriHTML(kysimus.getParameter("eesnimi"));
        String perenimi=Abi.filtreeriHTML(kysimus.getParameter("perenimi"));
        if(eesnimi!=null){
          valja.println("Tere, "+eesnimi+" "+perenimi);
        }
        valja.println("</body></html>");
    }
}
```

Nii võib väärtusi sisestada loodud tekstiväljadesse



ning pärast rakenduse töö tulemust imetleda.

Pilt servleti väljundina

Arvutis liikuvaid andmeid võib enamikul juhul käsitleda baidijadana ning ka servlet pole selle poolest erand. Nõnda võib servleti panna väljastama ka pilti või heli. HTTP-ühenduse päiseridadega antakse teada, millist tüüpi andmeid saadetakse ning edasine on juba vastuvõtja ülesanne. Et Javas leiduvad vahendid pildi kirjutamiseks voogu, siis saab andmeid võrdselt õnnelikult saata nii faili kui võrku. Ning siinses näites jõuavadki andmed üle võrgu kasutajani. Pilt luuakse mälus valmis ning lõpuks saadetakse andmed voogu pidi teele. Mugavaks pildi loomise vahendiks on BufferedImage ning sealt küsitud graafiline kontekst. Joonistamine toimub sarnaste käskude puhul nagu mujalgi.

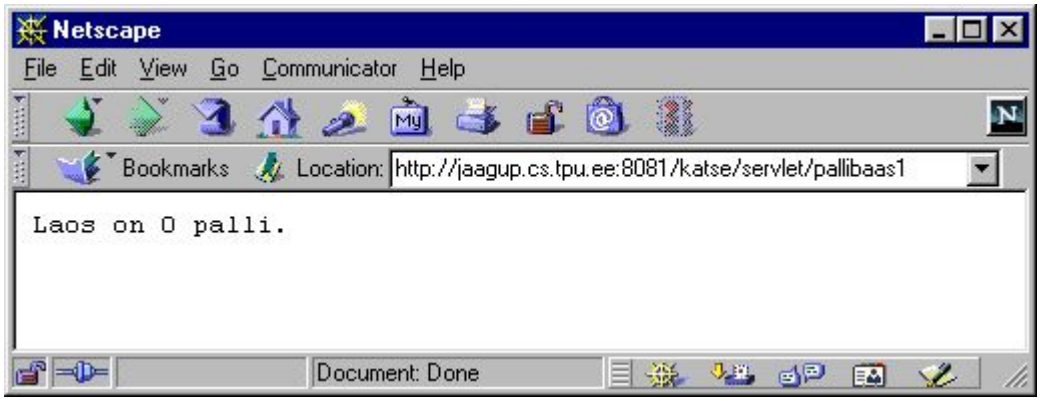
```
import javax.servlet.*;
import javax.servlet.http.*;
import com.sun.image.codec.jpeg.*;  //kuulub SUNi JDK-sse
import java.awt.image.*;
import java.awt.*;
import java.io.*;

public class piltservlet2 extends HttpServlet{
  public void doGet(HttpServletRequest kysimus, HttpServletResponse vastus)
      throws IOException, ServletException{
    int suurus=(int)(20+Math.random()*60);
    BufferedImage pilt=new BufferedImage(100, 100, BufferedImage.TYPE_INT_RGB);
    Graphics2D  piltg=pilt.createGraphics();
    piltg.setColor(Color.red);
    piltg.fillOval(50-suurus/2, 50-suurus/2, suurus, suurus);
    vastus.setContentType("image/jpeg");
    JPEGCodec.createJPEGEncoder(vastus.getOutputStream()).encode(pilt);
  }
}
```

Ning kui tulemus valmis, võib seda imetleda

Servlet ja andmebaas.

Veebist saabuvaid ja küsitavaid andmeid on küllalt mõistlik talletada andmebaasis. Sellisel juhul ei pea programmeerija liialt palju pead vaevama andmete poole üheaegsest pöördumisest tekkivate murede üle, sest selle eest hoolitsemine on juba andmebaasimootoritesse sisse ehitatud. Üldjuhul käib andmebaasiühenduse loomine servleti puhul nii nagu mujalgi programmis. Vaja laadida draiver, luua ühendus. Statement-objekt lause edastamiseks ning ResultSet andmete lugemiseks. Kuna ResultSet arvestab, et päringu tulemusel väljastatakse tabel, siis tuleb andmeid ka nõnda välja lugeda.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class pallibaas1 extends HttpServlet{
  public void doGet(HttpServletRequest kysimus,
                    HttpServletResponse vastus)
          throws IOException, ServletException{
    vastus.setContentType("text/plain");
    PrintWriter valja=new PrintWriter(
                          vastus.getWriter());
   try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection(
            "jdbc:odbc:poebaas", "", "");
    Statement st=cn.createStatement();
    String lause="SELECT kogus FROM pallid;";
    ResultSet rs=st.executeQuery(lause);
    rs.next();
    valja.println(
          "Laos on "+rs.getInt("kogus")+" palli.");
   }catch(Exception viga){
    valja.println("Probleem andmebaasiga: " +viga);
   }
  }
}
```



JSP

Servletid on mugavad olukordades, kus lehtedel on staatilist teksti vähe ning enamik sisust tuleb kokku arvutada. Suuremate püsivate tekstide puhul on võimalik neid teistest failidest või andmebaasikirjetest sisse lugeda. Siin lisatakse rakenduse juurkataloogis (nt. webapps/examples) paiknev fail SISU.JSP.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class kaasamiskatse extends HttpServlet{
  public void doGet(HttpServletRequest kysimus, HttpServletResponse vastus)
         throws IOException, ServletException{
    PrintWriter valja=vastus.getWriter();
    ServletContext kontekst=getServletConfig().getServletContext();
```

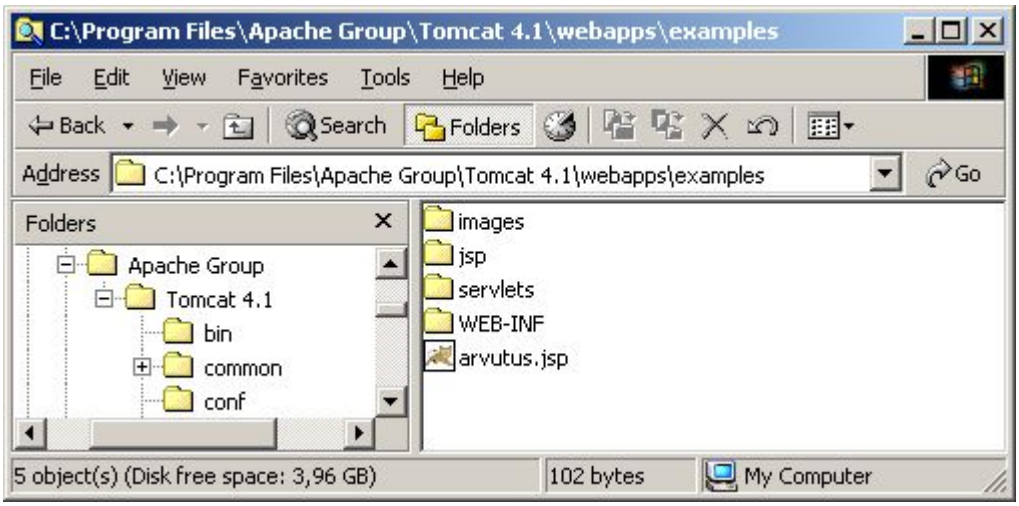
```
    RequestDispatcher rd=kontekst.getRequestDispatcher("/SISU.JSP");
    rd.include(kysimus, vastus);
  }
}
```

Kui aga väljaarvutamist nõudvaid paiku lehel suhteliselt vähem ning märkimisväärse osa lehe loomisest moodustab kujundus, siis sobib kasutada JSP-nimelist võimalust. Siin moodustab lehe põhiosa otse väljastatav tekst ning vaid erisümbolite vahel paiknevates lõikudes käivitatakse programm. Lihtsamal juhul kirjutatakse JSP-kood <% ja %> vahele, kuid uuema standardi järgi võimaldatakse ja soovitatakse JSP-lehed kirjutada XML-standardile vastaval kujul, kus jsp:ga algavad elemendid siis juhivad koodi käivitamist.

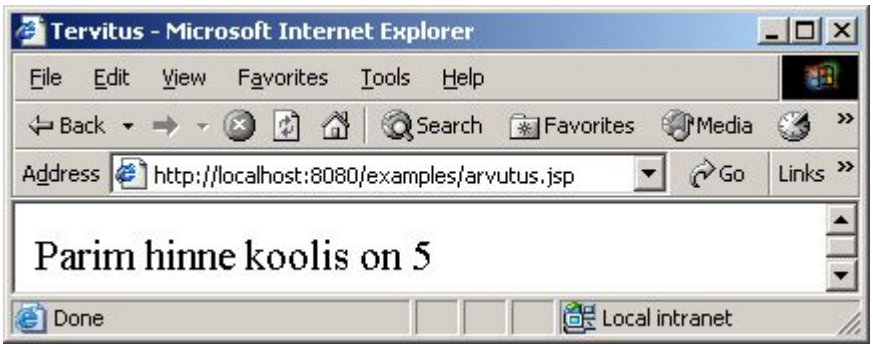
Järgnevalt on tegemist võimalikult lihtsa näitega, kus lihtsalt arvutatase kokku kahe arvu summa.

```
<html><head><title>Tervitus</title></head>
<body>
 Parim hinne koolis on <%=3+2 %>
</body></html>
```

Käivitamiseks tuleb failid panna samasse kataloogi kuhu harilikud html-failidik, nagu järgnevalt jooniselt näha.



Ning veebist vaatamiseks piisab sobiva aadressi sissetoksimisest.



JSP-lehti eraldi kompileerida pole vaja, selle eest hoolitseb juba käitur. Sestap ka lehe aeglane avamine esimesel algsel või muutmisjärgsel käivitamisel, sest seal tuleb kogu kompileerimise töö ära teha. Selline lähenemine võib eriti mugav olla näiteks inimestele, kel varem pole kompileerimisega kogemusi olnud ning PHP või muid skripte kirjutades juba harjunud, et piisabki vaid koodi kirjutamisest, kui juba võibki tulemusi imetlema asuda.

Et sisimas aga muudetakse JSP lehed enne servlettideks ja alles siis kompileeritakse/käivitatakse, kipuvad saabuvad veateated küllalt arusaamatud ja vähemasti algul häirivad olema. Näiteks võib juhtuda, et liitmisel sattus kogemata üks x-täht arvu taha.

```
<html><head><title>Tervitus</title></head>
<body>
```

```
 Parim hinne koolis on <%=3+2x %>
</body></html>
```

Tegemist on ju küllalt lihtsa ja sageli esineva veaga, mis võiks õnnestuda ilma suuremate muredeta ära parandada. Kui aga nüüd lehte avama asuda, ilmneb päris põhjalik veateade:

```
org.apache.jasper.JasperException: Unable to compile class for JSP

An error occurred at line: -1 in the jsp file: null

Generated servlet error:
    [javac] Since fork is true, ignoring compiler setting.
    [javac] Compiling 1 source file
    [javac] Since fork is true, ignoring compiler setting.
    [javac] C:\Program Files\Apache Group\Tomcat
4.1\work\Standalone\localhost\examples\arvutus_jsp.java:47: ')' expected
    [javac]       out.print(3+2x );
    [javac]                    ^
    [javac] 1 error



        at org.apache.jasper.compiler.DefaultErrorHandler.javacError
(DefaultErrorHandler.java:130)
        at org.apache.jasper.compiler.ErrorDispatcher.javacError
(ErrorDispatcher.java:293)
        at org.apache.jasper.compiler.Compiler.generateClass(Compiler.java:353)
        at org.apache.jasper.compiler.Compiler.compile(Compiler.java:370)
        at org.apache.jasper.JspCompilationContext.compile
(JspCompilationContext.java:473)
        at org.apache.jasper.servlet.JspServletWrapper.service
(JspServletWrapper.java:190)
        at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:295)
```

... lisaks veel paarkümmend rida näitamaks millised funktsioonid kust välja kutsuti. Ning seda kõike vaid ühe puuduva tähe pärast. Mõningase piilumise peale leiab koha, mis servletis vigaseks osutus.

```
    [javac] C:\Program Files\Apache Group\Tomcat
4.1\work\Standalone\localhost\examples\arvutus_jsp.java:47: ')' expected
    [javac]       out.print(3+2x );
```

Ning sealt pealt õnnestub aimata, et <%= kujul antud avaldis muudetakse print-käsu sisuks ning sinna taoline uitama asunud x ei sobi. Kui x eemaldada ning leht uuesti laadida, töötab ta jälle. Möödunud veateatest loeb välja ka näiteks loodava servleti asukoha: Tomcati alamkataloog work. Kui muu ei aita, tuleb asuda vastava servleti koodi lähemalt uurima. Aga enamasti ikka õnnestub JSP lehel soovitud kohti muutes ja välja kommenteerides segased kohad kindlaks teha ja parandada.

Tsükkel

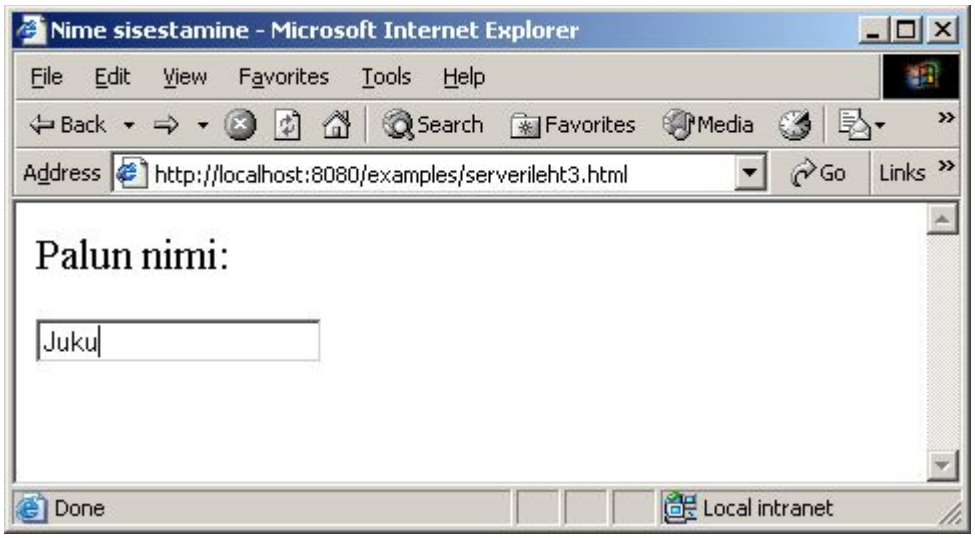
JSP lehe sisse saab Java koodi täiesti rahumeeles kirjutada. Et lõppkokkuvõttes muudetakse JSP-leht ikkagi servletiks, siis käivitamisel polegi nende vahel kuigivõrd vahet. Vaid kasutaja mugavuse mõttes kannatab JSP-lehel tavalist teksti kergemini edasi anda. Nõnda töötab harilik tsükkel

```
<html><head><title></title></head>
<body>
Arvude ruudud ühest kahekümneni on:
<p>
  <% for(int i=1; i<=20; i++){ %>
    <%= i*i+"<br>" %>
  <% }  %>
</body></html>
```

Ja tulemus lehel nagu oodatud.

Nagu mujal, nii ka siin soovitakse kasutaja käest andmeid saada, muidu poleks ju põhjust lasta programmil lehte kokku panna. Andmed nagu ikka kirjutatakse vormi ning selleks sobib täiesti tavaline HTML-leht. Vormi action-atribuudiga määratakse koht, kuhu sisestatud andmed saadetakse.

```
<html><head><title>Nime sisestamine</title></head>
<body>
 Palun nimi:
 <form action="serverileht3.jsp">
   <input type="text" name="nimi">
 </form>
</body></html>
```



Et tekstivälja nimeks oli “nimi”, siis võib selle kaudu ka andmed kinni püüda.

```
<html><head><title>Nime lugemine</title></head>
<body>
  Tere,
  <%= request.getParameter("nimi") %>
</body></html>
```

Ning nagu pildilt paistab, saabuvad vaikimisi GET-päringu korral andmed URL-rea kaudu.

Teate kaasamine

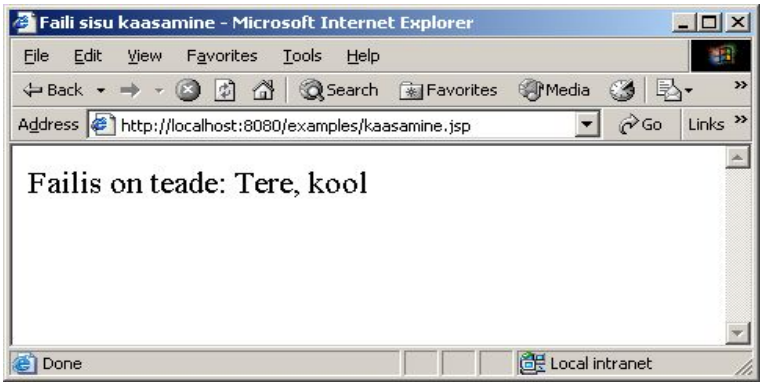
Kui samu andmeid soovitakse mitmel veebilehel kasutada, siis on mugav andmed ühte kohta kirja panna ning sobivates paikades faili sisse lugeda. Nagu järgnevast näitest paistab, on selliseks käsuks include, nii nagu mõnes muuski veebikirjutuskeeles (PHP, ASP).

```
<html><head><title>Faili sisu kaasamine</title></head>
<body>
 Failis on teade:
<%@ include file="teade.txt" %>
</body></html>
```

Ning vastav fail peab lihtsalt samas kataloogis omaette kättesaadav olema.

```
Tere, kool
```

Nõnda võibki töö tulemust imetleda.



Enam kasutatakse taolist kaasamist olukordades, kus soovitakse mitmele lehele luua ühesugune päis.

Kommentaarid

Enamikes keeltes jäetakse programmeerijale võimalus omi märkusi koodi juurde lisada ilma, et tavakasutaja sellest aimu saaks. Olgu siis tegemist rakenduse tutvustamise, koodi üksikute lõikude seletamise või ebasoovitavate lõikude ajutise eemaldamisega. JSP puhul on selleks kolm märgatavalt

erinevat võimalust. Esiteks võis HTMLi koodi sisse kirjutada oma “nähtamatu” tekst käskude <!-- ja --> vahele. Selline tekst jõuab küll kasutaja masinasse, kuid ei ole lehe tavalisel vaatamisel nähtav.

Kui soovida tervet JSP-lõiku eemaldada, siis võis selle panna <%-- ja --%> vahele. Nõnda saab lõike kergesti sisse ja välja lülitada. Ning lõppeks kehtivad ka tavalised Java-kommentaarid: // ühe rea tarvis ning /* ja */ pikema lõigu jaoks.

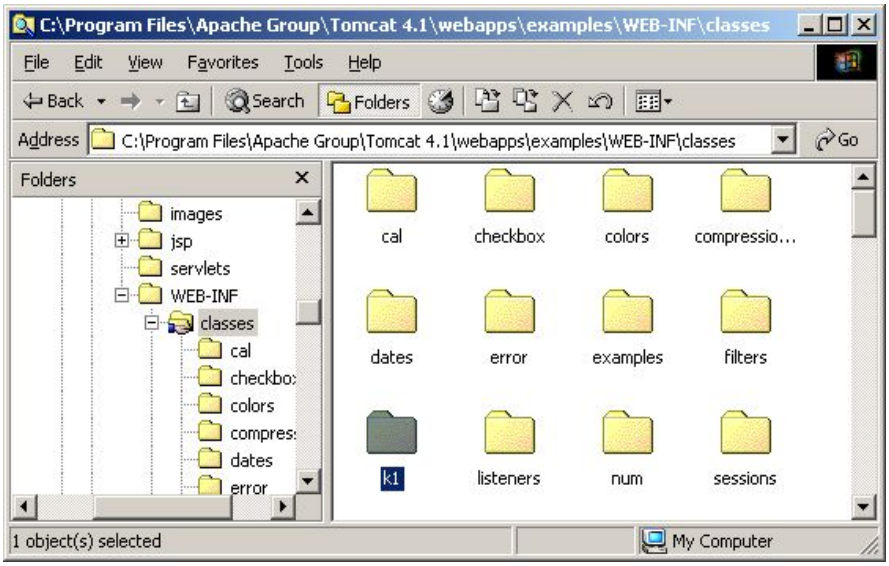
```
<html><head><title>Kommentaarid</title></head>
<body>
 <!-- Harilik kommentaar -->

 <%-- varjatud kommentaar --%>
<%
 //kommentaar koodi sees
%>
</body></html>
```

Uba

JSP lehtede sisu soovitatakse võimalikult lihtsaks jätta. Siis on neid võimelised kujundama ka programmeerimiskauged inimesed. Äriloogika ehk arvutused ning andmetega seotud toimingud saab paigutada eraldi ubadeks nimetatud klassidesse ning sealt siis sobivate käskude abil teenuseid küsida. Klassid paigutatakse sinna kuhu servletidki, ainult et soovitavalt veel iga teemaga seotud oad omaette kataloogi ehk paketti. Siinsel paketi nimeks on pandud k1, ning alt pildilt paistab ta ilusti classes-kataloogi alamkataloog olema.

Siinse oa ülesandeks on vaid nime meeles pidamine. Ning vaikimisi nimeks on Triin.

```
package k1;
import java.io.Serializable;
public class Uba1 implements Serializable{
  String nimi="Triin";
  public void paneNimi(String nimi1){
    nimi=nimi1;
  }
  public String annaNimi(){
    return nimi;
  }
  public String tutvusta(){
    return "Mu nimi on "+nimi;
  }
}
```



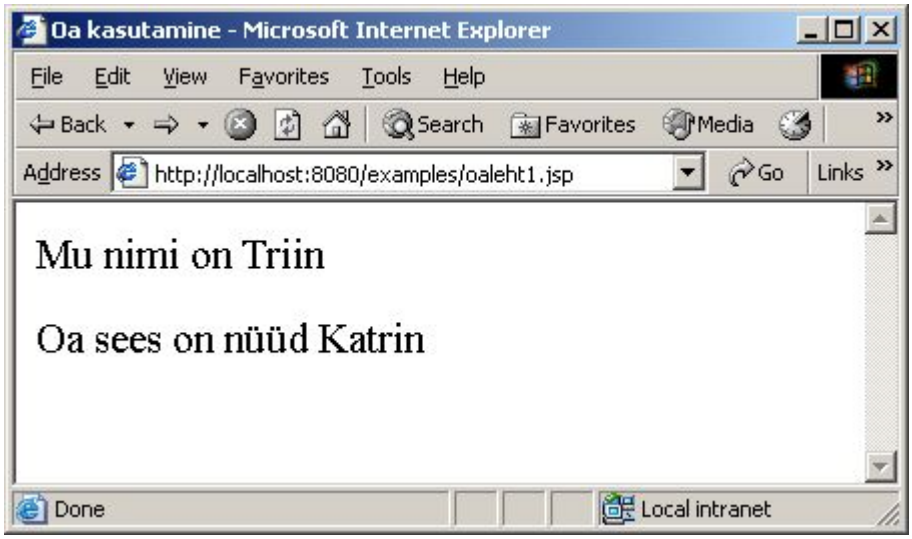
Kui klass loodud, tuleb see kompileerda nagu enamikele muudelegi Java-programmidele kohane. Kompileerimisel peaks aktiivne kataloog olema WEB-INF\classes, nii et kompileerimisel tuleb pakkettide tee mööda katalooge ette anda. Ning kompileeritud class-fail paigutatakse java-failiga samasse kataloogi.

C:\Program Files\Apache Group\Tomcat 4.1\webapps\examples\WEB-INF\classes>javac k1\Uba1.java

Kui uba valmis, võib teda kasutama hakata. Oa kirjeldamiseks lehel käsklus jsp:useBean. Atribuudiga annan oale nime, mille järgi hiljem selle poole pöörduda. Edasi saan osalt kasutada uba nagu tavalist muutujat. Kuigi – ubade tarbeks on JSP sisse ka mitmesuguseid muid pöördumisvõimalusi leitud.

```
<html><head><title>Oa kasutamine</title></head>
<body>
  <jsp:useBean id="nimehoidja" class="k1.Uba1"/>
  <%=nimehoidja.tutvusta() %>
  <%nimehoidja.paneNimi("Katrin"); %>
<p>Oa sees on nüüd
  <%=nimehoidja.annaNimi() %>
</body></html>
```

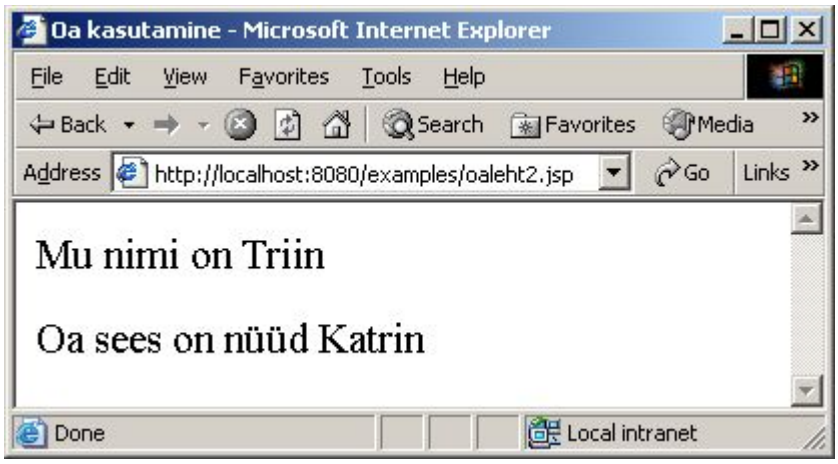
Kui algselt oli oa sees Triin ning eraldi käsuga määrati nimeks Katrin, siis nõnda võib tulemust ka veebilehelt näha.



Vaikimisi on oa poole võimalik pöörduda vaid sama lehe avamise jooksul. Kui soovida aga andmeid pikemks talletada, võib määrata skoobiks sessiooni. Nõnda püsivad andmed paigal sama kasutaja mitme järjestikuse pöördumise ajal ning seal võib meeles pidada näiteks teadet, et kasutaja on end juba sisse meldinud.

```
<html><head><title>Oa kasutamine</title></head>
<body>
  <jsp:useBean id="nimehoidja" class="k1.Uba1" scope="session" />
  <%=nimehoidja.tutvusta() %>
  <%nimehoidja.paneNimi("Katrin"); %>
<p>Oa sees on nüüd
  <%=nimehoidja.annaNimi() %>
</body></html>
```

Siin töötab leht esimese pöördumise puhul nii nagu eelmisel korral: kuna midagi pole veel eraldi salvestatud, siis alguses teatatakse ikka vaikimisi nimeks olev Triin.

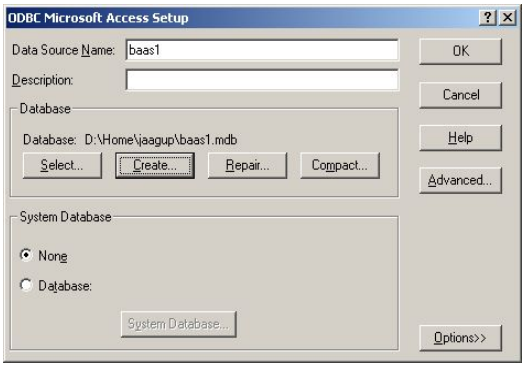


Kui nüüd aga järgmisel korral sama sessiooni jooksul minna oast andmeid küsima, siis on seal kirjas juba eelmisest korrast meelde jäänud nimi.

Uba ja andmebaas

Et igasugused programmeerimiskäsud püütakse JSP-lehest eemal hoida, siis on mõistlik ka andmebaasiga seotud toimingud oa sisse peita.

Kui peitmine korralik, siis ei pruugi JSP-lehe looja sageli teadagi, kus baasis andmeid hoitakse. See võimaldab vajadusel andmekandjat küllalt kergesti vahetada ning vajadusel näiteks andmebaasi sootuks tavalise tekstifailiga asendada. Et ühendamine libedalt läheks, loome ka siin ODBC alla andmeallika.



Edasi koostame oa, millel oskused nii andmetabeli loomiseks kui väärtuse seadmiseks ja küsimiseks. Ning kui eelmise oaga võrrelda, siis näevad nad küllalt sarnased välja – ikka käsud väärtuste seadmiseks ja küsimiseks. Ning oa kasutaja ei peagi teadma, et andmeid just baasis hoitakse.

```
package k1;
import k1.*;
import java.sql.*;
public class Baasiuba1{
  Connection cn;
```

```
  Statement st;
  public Baasiuba1(){
   try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    cn=DriverManager.getConnection("jdbc:odbc:baas1", "", "");
    st=cn.createStatement();
   }catch(Exception e){
     System.out.println(e);
   }
  }
  public void looBaas() throws SQLException{
    String lause="CREATE TABLE pallid (kogus int);";
    st.executeUpdate(lause);
    lause="INSERT INTO pallid (kogus) values ('0');";
    st.executeUpdate(lause);
  }
  public void setPalliarv(int arv) throws SQLException{
    String lause="UPDATE pallid SET kogus="+arv+";";
    st.executeUpdate(lause);
  }
  public int getPalliarv() throws SQLException{
    String lause="SELECT kogus FROM pallid;";
    ResultSet rs=st.executeQuery(lause);
    rs.next();
    return rs.getInt("kogus");
  }
}
```

Kui paketi sees olev uba loodud, tuleb ta kompileerida nagu Java-fail ikka.

```
C:\Program Files\Apache Group\Tomcat 4.1\webapps\examples\WEB-INF\classes>javac
k1\Baasiuba1.java
```

Et pallilao administreerimine mugavamalt läheks, selleks on ka tabeli loomiseks omaette "administraatorileht" tehtud. Võrgust leitavate rakenduste puhul võib sageli kohata juhendit, kus üles seadmiseks tuleb vaid andmebaasi nimi määrata või sobiva nimega baas luua ning edasi õnnestub kõik veebi kaudu paika sättida. Siin vaid öeldakse oale, et looBaas (mille juures praegu küll vaid üks tabel luuakse) ning võibki asuda juba rakenduse teeneid kasutama. Kontrolliks küsitakse välja baasis leiduvate palllide arv. Nagu näha, ei tehta seda mitte tavalise funktsiooniväljakutsega, vaid oa väärtuste küsimiseks sobib element jsp:getProperty. Mis küll toimimiseks eeldab, et oal oleks vastavanimeline get-liitega algav meetod.

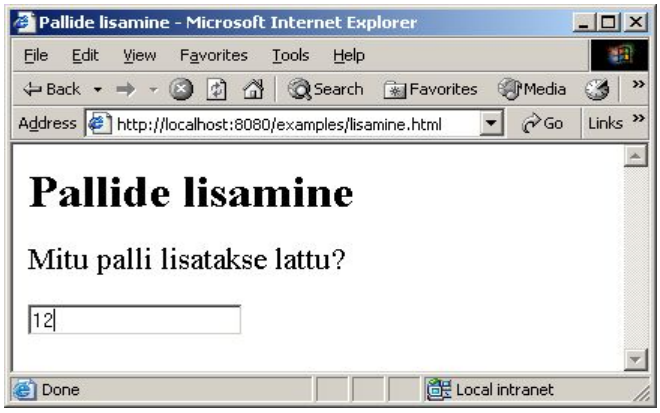
```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<% pallibaas.looBaas(); %>
<html><head><title>Baas loodud</title></head>
<body><h2>Baas loodud</h2>
 Baas pallide arvu loomiseks õnnelikult loodud.
 Laos on <jsp:getProperty name="pallibaas" property="palliarv"/> palli.
</body></html>
```

Nii võib koodilõigu tööd veebist imetleda ning pärast ka andmebaasifailist piiluma minna, et soovitud tabel ka tegelikult loodud ning väärtus sinna sisse pistetud on.

Andmete lisamisel küsitakse kasutaja käest lisatavate pallide arvu ning saadetakse tulemused edasi lehele lisamine.jsp.

```
<html><head><title>Pallide lisamine</title></head>
```

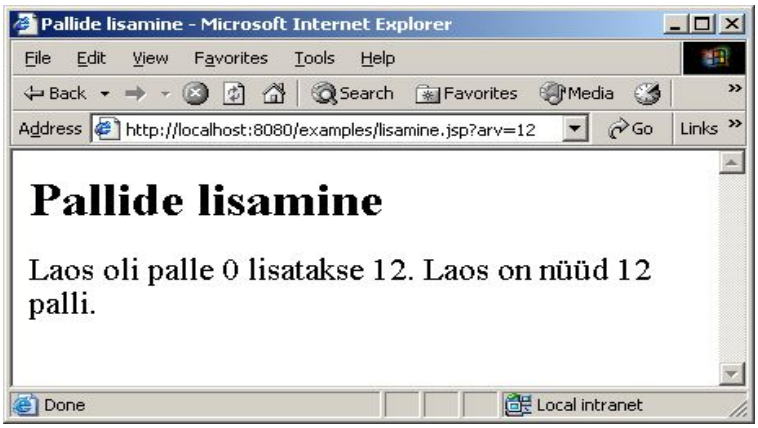
```
<body><h2>Pallide lisamine</h2>
 Mitu palli lisatakse lattu? <br>
 <form name="vorm1" action="lisamine.jsp">
   <input type="text" name="arv">
 </form>
</body></html>
```



Too leht saab väärtused URL-i rea pealt kätte ning määrab baas uue pallide arvu. Pärastine väärtuse küsimine nagu ennegi – getProperty kaudu.

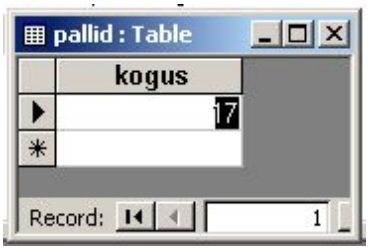
```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<html><head><title>Pallide lisamine</title></head>
<body><h2>Pallide lisamine</h2>
 <%
  int olemas=pallibaas.getPalliarv();
  int juurde=Integer.parseInt(request.getParameter("arv"));
  int kokku=olemas+juurde;
  out.println("Laos oli palle "+olemas+" lisatakse "+juurde+".");
  pallibaas.setPalliarv(kokku);
 %>
  Laos on nüüd <jsp:getProperty name="pallibaas" property="palliarv"/> palli.
</body></html>
```

Ja võibki koodi tööd veebilehel imetleda.



Lisada kannatab ka olemasolevatele juurde.

Ning andmebaasi tabelisse vaatama minnes võib veenduda, et sinna ka arv 17 jõudnud on



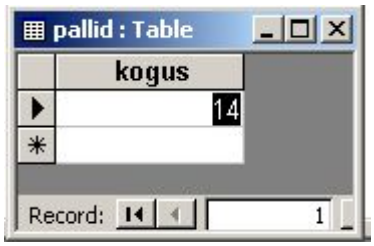
Ostmise puhul on toiming lihtsalt teistpidine. Algul tasub ikka küsida, kas laost üldse midagi võtta on ning siis teada anda, mitut palli osta soovitakse.

```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<html><head><title>Pallide ostmine</title></head>
<body><h2>Pallide ostmine</h2>
 Laos on <jsp:getProperty name="pallibaas" property="palliarv"/> palli.
 Mitu palli ostetakse? <br>
 <form name="vorm1" action="eemaldamine.jsp">
   <input type="text" name="arv">
 </form>
</body></html>
```

Müümisel kõigepealt kontrollitakse, et nõnda palju kaupa ikka jagub ning vaid sobivuse korral võetakse tehing ette.

```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<html><head><title>Pallide eemaldamine</title></head>
<body><h2>Pallide eemaldamine</h2>
 <%
  int olemas=pallibaas.getPalliarv();
  int maha=Integer.parseInt(request.getParameter("arv"));
  int tulemus=olemas-maha;
  out.println("Laos oli palle "+olemas+" väljastada soovitakse "+maha);
  if(tulemus<0){
     out.println("Väljastada saab vaid "+olemas+" palli");
     tulemus=0;
  }
  pallibaas.setPalliarv(tulemus);
 %>
  Lattu jäi <jsp:getProperty name="pallibaas" property="palliarv"/> palli.
</body></html>
```

Ning andmetabelist võib taas järele kontrollida, et veebi väljastatud andmed ikka õiged on.



JDBC käskude ülevaade

Kui kord ühendus loodud, siis edasised toimetused võiksid juba mõnevõrra lihtsamalt sujuda. Järgnevalt proovitakse läbi mitmed levinumad andmetega ümber käimise võtted.

Kõikide ridade väljastus.

Ehk levinumaiks esimeseks andmebaasiga seotud rakenduse ülesandeks ongi andmete kasutajale näitamine soovitud kujul. Tüüpilisel juhul tuleb luua ühendus, koostada päring, käivitada see ning saabunud andmed sobival kujul kuvada. Järgnevas näites küsitakse eelnevalt loodud inimeste andmete tabelist välja kõikide veergude andmed. Ekraanile aga näidatakse neist eesnime ja sünniaasta väärtused.

```
import java.sql.*;
public class Inimloetelu{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    ResultSet rs=st.executeQuery("select * from inimesed");
    while(rs.next()){
      System.out.println(rs.getString("eesnimi")+":"+rs.getInt("synniaasta"));
    }
    cn.close();
  }
}

/*
C:\jaagup\andmed>java Inimloetelu
Juku:1989
Kati:1987
Mati:1983
*/
```

Andmed andmete kohta

Kui vaja rakenduse võimalusi sageli muuta, või kui soovitakse sama andmeväljastuslõiku kasutada mitmesuguste andmete korral, siis aitab päringu vastusega koos tulev metadata ehk andmed andmete kohta. Mõne andmebaasimootori või draiveri puhul võib vastav võimalus puududa või töötada nuditult. Kui aga andmeid kirjeldavate andmete küsimise võimalus olemas, siis saab neid sarnaselt kätte sõltumata kasutatavast andmebaasist.

Nagu alljärgnevast näitest näha, tuleb andmete kirjeldus ResultSet'ist eraldi käsuga välja küsida. Edasi õnnestub juba ResultSetMetaData tüüpi objektist üksikute käskude abil omale soovitavaid andmeid teada saada.

```
    ResultSetMetaData rmd=rs.getMetaData();
    int veergudearv=rmd.getColumnCount();
```

Kui veergude arv ja andmed teada, siis saab koostada koodilõigu, mis juba vastavalt andmed välja küsib ja nendega edasi toimetab. Siin näites trükitakse tulemused vaid ekraanile, kuid sarnaselt võib ette valmistada väljastuse veebilehele või mujalegi. Kes juhtub kasutama andmebaasihaldusvahendeid nagu näiteks veebipõhine PHP MyAdmin, võib aimata, et taoliste rakenduste koostamisel kuluvad saabuvad andmed andmete kohta väga marjaks ära.

```
import java.sql.*;
public class Inimloetelu2{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    ResultSet rs=st.executeQuery("select * from inimesed");
    ResultSetMetaData rmd=rs.getMetaData();
    int veergudearv=rmd.getColumnCount();
    System.out.println("Tabelis on "+veergudearv+" veergu:");
    for(int i=1; i<=veergudearv; i++){
      System.out.println("Nimi:         "+rmd.getColumnName(i));
      System.out.println("Kirjeldus:    "+rmd.getColumnLabel(i));
      System.out.println("Tyyp:         "+rmd.getColumnTypeName(i));
      System.out.println("Java klass:   "+rmd.getColumnClassName(i));
      System.out.println("Isesuurenev:  "+rmd.isAutoIncrement(i));
      System.out.println("Suurim laius: "+rmd.getColumnDisplaySize(i));
      System.out.println();
    }

    while(rs.next()){
       System.out.print(rs.getRow()+". ");
       for(int i=1; i<=veergudearv; i++){
         System.out.print(rs.getObject(i)+"    ");
       }
       System.out.println();
    }
    cn.close();
  }
}
```

Ning programmi töö tulemusena anti ilus selge ülevaade kasutatavast andmetabelist. Nii tulpade kirjeldused ükshaaval, kui pärast kogu tabeli sisu.

```
/*
Tabelis on 3 veergu:
Nimi:         id
Kirjeldus:    id
Tyyp:         COUNTER
Java klass:   java.lang.Integer
Isesuurenev:  true
Suurim laius: 11

Nimi:         eesnimi
Kirjeldus:    eesnimi
Tyyp:         VARCHAR
Java klass:   java.lang.String
Isesuurenev:  false
Suurim laius: 50

Nimi:         synniaasta
Kirjeldus:    synniaasta
Tyyp:         INTEGER
Java klass:   java.lang.Integer
Isesuurenev:  false
Suurim laius: 11

1. 1    Juku    1989
2. 2    Kati    1987
3. 3    Mati    1983

*/
```

Päringu tulemuste hulgas liikumine.

Esimese lähendina võib relatsioonilisest andmebaasist andmete välja küsimine tunduda küllalt selgena. Kõik andmed ja nende vahelised seosed esitatakse baasis tabelitena ning ka tulemuseks on tabel - sarnane programmeerijale tuttava kahemõõtmelise massiiviga. Et toimingute sisemine keerukus püütakse rakenduse loojate eest võimalikult peita, siis ideaaljuhul polegi vaja muule mõelda kui rea numbrile ja veeru nimele või numbrile, kust andmed enesele välja küsida.

Andmebaasimootorid peavad hakkama saama suurte andmemahtude ning mitmete üheaegsete kasutajatega. Selle toimimise tarvis tuleb arvestada andmebaasiühenduse kasutatavate enesekaitsemehhanismidega. Vaikimisi seadete korral õnnestub päringust andmeid välja meelitada vaid ridu järjest eest tahapoole lugedes ning iga väärtust vaid ühe korra küsides. Sellise lähenemise puhul ei pea sugugi kõik päringu väljastatavad andmed olema korraga baasist välja küsitud. Mällu loetakse ja üle kantakse vaid need, mis parasjagu tarvilikud. Ülejäänud võivad veel oma aega oodata ning uuritud ridade arvelt võib sootuks mälu vabastada. Selline vaikimisi järjest küsimine peab ka kõigi töötavate draiverite puhul ühtviisi leiduma ja toimima. Kui teada ja arvestada taolist ette kirjutatud andmete küsimise järjekorda, siis enamike rakenduste puhul olulist probleemi ei teki. Ekraanile või veebilehele saadetaksegi andmed sageli saabumise järjekorras. Ning kui vaja kokkuvõtteid teha või mõnda väärtust korduvalt kasutada, siis tuleb lihtsalt vajalikud andmed muutujatesse ja massiividesse kirjutada ning edaspidi kasutada kui tavalisi programmeerimise juures tarvilikke andmeid.

On aga mingil põhjusel kindel vajadus mööda andmeid sageli edasi-tagasi liikuda, siis tuleb vajadust juba eelnevalt arvestada. Javakeelsete programmide puhul luuakse kõigepealt ühendus (Connection). Iga ühenduse kaudu võib korraga töötada mitu käsklust (Statement). Ning iga käskluse küljes võib vajadusel olla avatud korraga kuni üks vastuste kogum (ResultSet). Vastuste kogumi omadused määratakse juba käskluse loomisel. Järgnevas näites paistavad konstandid

```
ResultSet.TYPE_SCROLL_INSENSITIVE
ResultSet.CONCUR_READ_ONLY
```

Esimene neist määrab, et saabunud vastustehulka võib soovitud suunas kerida. Olgu siis edaspidi või tagurpidi. Suurema andmehulga korral võib selline nõudmine märgatavalt ressursse nõuda või riistvarale sootuks üle jõu käia. Kuid kui näiteks Swingi vahenditega tabelit luua, siis on päris mugav, kui võib otse ResultSetist saabuvaid andmeid usaldada ning ei pea hakkama veel lisaks oma andmekogumit looma.

Teise parameetri tagamaad on veel mõnevõrra keerulisemad. CONCUR_READ_ONLY tähendab, et andmeid võib päringust vaid lugeda, mis nagu päringu juures võikski loomulik tunduda. Nagu aga hilisematest näidetest paistab, ei pruugi see päringu kasutamise ainuke võimalus olla.

Järgnevas näites käiakse läbi ResultSet'is liikumise tähtsamad käsud. Enamik neist võiks olla otse inglise keelest tõlgitavad. Käsk last palub minna viimasele reale, isLast kontrollib, kas ollakse viimasel real; relative liigub soovitud arvu ridasid jooksvast reast alates, absolute loendab ridasid alates päringu algusest.

```
import java.sql.*;
public class Inimloetelu3{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_READ_ONLY
    );
    ResultSet rs=st.executeQuery("select * from inimesed");
    rs.last();
    System.out.println("Ridu kokku: "+rs.getRow());
    rs.previous();
    System.out.println("Eelviimane eesnimi: "+rs.getString("eesnimi"));
    rs.absolute(3);
    System.out.println("Kolmas eesnimi: "+rs.getString("eesnimi"));
    rs.relative(2);
    System.out.println("Ylejärgmine eesnimi: "+rs.getString("eesnimi")+
                       ", reanr: "+rs.getRow());
    if(rs.isLast()){
      System.out.println("Tegemist on viimase reaga");
    }
    rs.relative(-1);
    System.out.println("Eelmine eesnimi: "+rs.getString("eesnimi")+
                       ", reanr: "+rs.getRow());
    cn.close();
  }
}
```

```
/*
Ridu kokku: 6
Eelviimane eesnimi: Juk's
Kolmas eesnimi: Mati
Ylejõrgmine eesnimi: Juk's, reanr: 5
Eelmine eesnimi: Sass, reanr: 4

____

Nimed:

Juku
Kati
Mati
Sass
Juk's
Jass

*/
```

Päringu mahu piiramine

Rakendusi koostades on vahel raske aimata, kui suurte andmekogustega tuleb tegemist teha. Näited, mis kahe või kümne rea juures ilusti toimivad, ei pruugi tuhandete vastusridade puhul enam sugugi kasutatavad olla. Olgu siis tegemist ekraani nähtava ala ummistumisega, arvuti mälumahu või arvutusvõimsuse lõpuga. Üheks võimaluseks on eelneva päringu abil kontrollida, millises suurusjärgus vastustehulgaga võiks tegemist olla ning edasi juba mitme võimaluse tarbeks kood kokku panna. Lihtsamaks ning mõnigikord kasutatavaks lähendiks aga piisab päringu ridade arvu või tööaja piiramisest. Kui ka ei saa soovitud kohast kõiki andmeid kätte, siis vähemasti jääb rakendus ellu, arvuti ei jookse kokku ning võimalik näiteks täpsustatud parameetritega uus päring kokku panna.

```
import java.sql.*;
public class Inimloetelu4{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
    st.setMaxRows(2);             //Suurim väljastatavate ridade arv
    //    st.setQueryTimeout(2);
    //Valikuline käsklus, päringu suurim aeg sekundites
    ResultSet rs=st.executeQuery("select eesnimi from inimesed");
    while(rs.next()){
      System.out.println(rs.getString("eesnimi"));
    }
    cn.close();
  }
}

/*
Juku
Kati
*/
```

Lisamine.

Harilik lisamine käib pea kõikide keelte ja vahendite puhul sarnaselt. Ikka tuleb kokku panna SQLi INSERT-lause ning siis käivitada. Java käskluseks nii lisamise kui muutmiste korral on executeUpdate. Vaid päringu puhul oli executeQuery, kus siis tulemuseks väljastati ResultSet.

```
import java.sql.*;
public class Inimlisamine1{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement();
```

```
    st.executeUpdate(
     "INSERT INTO inimesed (eesnimi, synniaasta) values ('Sass', 1977)"
    );
    cn.close();
  }
}
```

PreparedStatement

Lisamist levinud operatsioonina on püütud mitmel moel paindlikumaks muuta. Kui vaja kümneid kordi sama lauset erinevate andmetega käivitada, siis igakordne masinapoolne SQL-lause analüüs ning enesele sobivaks seadmine võtavad oma aja. Kui aga kord koostada PreparedStatement ning hiljem vaid väärtusi sees vahetada, siis võiks tulemus mõnevõrra kiiremini saabuda.

Teiseks mureks sisestuste juures on erisümbolid. Ehkki mõeldakse välja mitmeid viise andmete sisse jäävate ülakomade ja muude märkide varjestamiseks, kipub ikka turvaauke sisse jääma, kus lihtsalt sisestuse kaudu suudetakse SQL-laused sassi ajada ning rakenduse tööd muuta. Või teistpidi juhtub vahel, et sümboleid varjestatakse mitmekordselt ning hiljem kipuvad varjestuse jäljed väljundisse sisse jääma.

Kui kasutada aga PreparedStatementi, siis varjestusega muresid pole. Sest sisestatavaid andmeid otse SQL-lausesse ei kirjutatagi. Algsesse lausesse pannakse andmete kohale küsimärgid. Ning alles hiljem määratakse, millised andmed selle käivitamise korral küsimärkide asemele paigutatakse.

```
import java.sql.*;
public class Inimlisamine2{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    PreparedStatement st=cn.prepareStatement(
     "INSERT INTO inimesed (eesnimi, synniaasta) values (?, ?)"
    );
    st.setString(1, "Juk's");
    st.setInt(2, 1972);
    st.executeUpdate();
    cn.close();
  }
}
```

Päringus lisamine

Lihtsustamaks rakenduste loomist, kus nähtavaid andmeid ka kohe muuta saab, võimaldavad osa andmebaasimootoreid lihtsamate päringute korral ka päringust väljastatud tulemusi muuta või sama tüüpi ridasid algsetesse andmetesse juurde luua. Kui summeeritaks päringus näiteks inimeste arv, siis seda loomulikult muuta ei lubata - pole ju võimalik nõnda lihtsalt olematuid inimesi juurde tekitada. Kui aga päringuks on lihtsalt ühe tabeli esitus või ka lihtsam ühend, siis võib muutmine ja lisamine täiesti õnnestuda. Allpoolses näites küsitakse inimeste andmed ning edaspidiste käsklustega lisatakse üks inimene loetellu juurde.

```
    rs.moveToInsertRow();
    rs.updateString(1, "Jass");
    rs.updateInt(2, 1968);
    rs.insertRow();
```

räägib igaüks enese eest. Nii nagu võib mõnikord vormis andmeid viimasele reale juurde kirjutada, nii lubatakse ka siin programmi abil üks rida päringu poolt väljastatud tabelisse juurde panna, lahtrid väärtustega täita ning siis tulemused paika saata.

```
import java.sql.*;
public class Inimlisamine3{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    Statement st=cn.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_UPDATABLE
    );
    ResultSet rs=st.executeQuery(
      "SELECT eesnimi, synniaasta FROM inimesed"
```

```
    );
    rs.moveToInsertRow();
    rs.updateString(1, "Jass");
    rs.updateInt(2, 1968);
    rs.insertRow();
    cn.close();
  }
}
```

Transaktsioonid

Vahel pidada olema pool muna halvem kui tühi koor. Et kui töö jäi tervikuna tegemata, siis järgmisel korral teada, et võib kõike otsast alustada. Kui aga miskit poole peale rippuma jäi, võib kergemini juhtuda, et mõni tegevus hiljem kaks korda tehtud saab või sootuks kahe silma vahele jääb. Tüüpiliseks näiteks tuuakse pangaülekannet, kus ühelt kontolt võtmine ning teisele ülekandmine ikka paarikaupa peavad käima. Ning enne lõplikku tulemuste kinnitamist peab veenduma, et mõlemad toimingud õnnestuvad. Selliste seoste loomiseks võib automaatse täitmise peatada käsuga.

```
    cn.setAutoCommit(false);
```

Edasised toimingud jäävad ootele. Kui selgub, et midagi tuli vahele, siis saab algseisu taastada käsuga

```
    cn.rollback();
```

```
import java.sql.*;
public class Inimlisamine4{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    cn.setAutoCommit(false);
    Statement st=cn.createStatement();
    st.executeUpdate(
      "INSERT INTO inimesed(eesnimi, synniaasta) values('Elmar', 1955)"
    );
    cn.rollback();
    cn.close();
  }
}
```

On aga kõik õnneks läinud, siis kannatab öelda

```
    cn.commit();
```

ning muutused kinnistatakse.

```
import java.sql.*;
public class Inimlisamine5{
  public static void main(String[] argumendid) throws Exception{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection("jdbc:odbc:poebaas");
    cn.setAutoCommit(false);
    Statement st=cn.createStatement();
    st.executeUpdate(
      "INSERT INTO inimesed(eesnimi, synniaasta) values('Mann', 1911)"
    );
    cn.commit();
    cn.close();
  }
}
```

SQL-laused

Järgnevalt vaatame läbi enamlevinud SQL-käsklused. Näited on tehtud MySQLi-nimelise andmebaasi abil, kuid samalaadsed käsklused leiduvad ka teiste andmebaaside juures. Mõnel korral lihtsalt vaja täpne kuju vastavast manuaalist järele vaadata.

Tabeli loomiseks kasutatakse käsklust CREATE TABLE. Tavaks on kirjutada otse SQL-keele käsklused suurte tähtedega, muud väikestega, kuid iseenesest on SQL-andmebaasid tõstutundetud. Tabeli nimeks siin näites teated2. Edasi tulevad sulgudes komadega eraldatult tulpade nimed ja kirjeldused. Järgnevalt on tulpade nimedeks id, teade ja nimi. Tüüpideks vastavalt int, text ja text. NOT NULL esimese välja taga tähendab, et tulbas ei tohi olla tühiväärtusi; auto_increment aga, et andmete lisamisel tabelisse paigutatakse sinna lahtrisse automaatselt leitud unikaalne väärtus. PRIMARY KEY(id) loetelu

lõpus näitab, et tulp nimega id on primaarvõtmeks ehk üldjuhul kui viidatakse selle tabeli reale, siis kasutatakse selleks primaarvõtme unikaalset väärtust.

```
CREATE TABLE teated2(
  id int NOT NULL auto_increment,
  teade TEXT,
  nimi TEXT,
  PRIMARY KEY(id)
);
```

Järgmisena andmete lisamise lause, mis peaks sellisel kujul kõikidele SQL-andmebaasidele arusaadav olema. INSERT INTO, millele järgneb tabeli nimi, sulgudes tulpade loetelu kuhu lisatakse, seejärel sõna VALUES ning edasi väärtuste loetelu. Tekstilised väärtused ülakomade vahel. Sõltuvalt andmebaasimootorist on sellel käsklusel mitmeid erikujusid mitme rea andmete korraga sisestamiseks või tulpade nimede ja väärtuste lähemale kirjutamiseks, kui siintoodu peaks kõige üldisem ja töökindlam olema.

```
INSERT INTO teated2(teade, nimi) VALUES
  ('Kool hakkab kell 10', 'Mati');
```

MySQLi-spetsiifiline kirjeldus tabeli tutvustuse kuvamiseks. Ka teistel andmebaasidel leiab selliseid kirjeldavaid vahendeid, olgu siis tekstipõhiseid või graafilisi.

```
mysql> explain teated2;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(11) |      | PRI | NULL    | auto_increment |
| teade | text    | YES  |     | NULL    |                |
| nimi  | text    | YES  |     | NULL    |                |
+-------+---------+------+-----+---------+----------------+
3 rows in set (0.26 sec)
```

Kõige universaalsem pärngulause. Tärn tähendab, et näha soovitakse kõiki ridu. Tärni asemel võiks olla ka soovitavate tulpade loetelu.

```
mysql> select * from teated2;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  1 | Kool hakkab kell 10          | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  4 | Otsi riiuli tagant           | Kati |
|  5 | Mina toon palli              | Siim |
|  6 | Mina ka                      | Mati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
7 rows in set (0.38 sec)
```

Kui tahta näha vaid erinevaid väärtusi, siis selle juures aitab käsklus distinct. Ehkki Mati on saatnud tunduvalt rohkem kui ühe teate, siis siin kuvatakse iga nimi ikkagi ainult ühe korra.

```
mysql> select distinct nimi from teated2;
+------+
| nimi |
+------+
| Mati |
| Kati |
| Siim |
+------+
3 rows in set (0.34 sec)
```

Kui soovitakse mõne tulba järgi järjestada, siis selleks võib lisada lauseosa order by ning soovitud tulba nime.

```
mysql> select * from teated2 order by nimi;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
```

```
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
|  1 | Kool hakkab kell 10          | Mati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  6 | Mina ka                      | Mati |
|  5 | Mina toon palli              | Siim |
+----+------------------------------+------+
7 rows in set (0.07 sec)
```

Soovides vastava tulba järgi tagurpidises järjestuses tulemust näha, tuleb lisada võtmesõna desc. Soovides rõhutada päripidist järjestust, võib kirjutada sõna asc, kuid see kehtib ka vaikimisi.

```
mysql> select * from teated2 order by nimi desc;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  5 | Mina toon palli              | Siim |
|  1 | Kool hakkab kell 10          | Mati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  6 | Mina ka                      | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
7 rows in set (0.03 sec)
```

MySQL võimaldab juba SQL-lauses vastuste arvu piirata. Praegusel juhul seati suurimaks väljastatavate vastuste arvuks neli. Mõne andmebaasimootori korral on vastavaks piiravaks käskluseks top.

```
mysql> select * from teated2 limit 4;
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  1 | Kool hakkab kell 10          | Mati |
|  2 | Võta vihikud kaasa           | Kati |
|  3 | Matemaatika vihik on kadunud | Mati |
|  4 | Otsi riiuli tagant           | Kati |
+----+------------------------------+------+
4 rows in set (0.14 sec)
```

Siin antakse teada, et soovitakse näha teateid alates kolmandast kaks tükki. Selline piirang on näiteks mugav lehtede loomisel, kus kõiki andmeid ei soovita korraga ühele lehele paigutada, vaid vastavalt kasutaja soovile näidatakse järgmisi lehekülgi.

```
mysql> select * from teated2 limit 3, 2;
+----+--------------------+------+
| id | teade              | nimi |
+----+--------------------+------+
|  4 | Otsi riiuli tagant | Kati |
|  5 | Mina toon palli    | Siim |
+----+--------------------+------+
2 rows in set (0.00 sec)
```

Tahtes konkreetse väärtuse järgi piirata väljastatavaid ridu, tuleb selline piirang kirjutada where-lausesse. Kui tingimusi on rohkem, siis ühendamiseks sobivad sõnad AND ning OR. Võrdlemiseks märgid < ja > nagu muudelgi puhkudel.

```
mysql> select * from teated2 where nimi='Kati';
+----+------------------------------+------+
| id | teade                        | nimi |
+----+------------------------------+------+
|  2 | Võta vihikud kaasa           | Kati |
|  4 | Otsi riiuli tagant           | Kati |
|  7 | Jätke mu ilus kleit valgeks  | Kati |
+----+------------------------------+------+
3 rows in set (0.00 sec)
```

Loendamiseks sobib käsklus count. Sõnapaar "as nr" avaldise count(*) taga tähendab, et loenduse tulemus väljastatakse tulbana, mil nimeks nr.

```
mysql> select count(*) as nr from teated2 where nimi='Kati';
+----+
```

```
| nr |
+----+
|  3 |
+----+
1 row in set (0.32 sec)
```

Ühe tabeliga seotud lihtsamad rakendused enamasti eeltoodud päringutega piirduvadki. Loendava statistika puhul aga teeb järgnev vahend elu mõnevõrra mugavamaks. Lisand "group by" võimaldab väljundisse jätta näidatud tulba väärtustest vaid erinevad. Samas mitme rea andmeid arvestavad funktsioonid nagu count, sum ja avg töötavad siis eraldi iga sellise grupi kohta ning nõnda võibki leida ühe käsuga iga inimese teadete arvu või kokku kulutatud summa.

```
mysql> select nimi, count(*) as kogus from teated2 group by nimi;
+------+-------+
| nimi | kogus |
+------+-------+
| Kati |     3 |
| Mati |     3 |
| Siim |     1 |
+------+-------+
3 rows in set (0.01 sec)
```

Nii nagu harilikel päringutel saab piiranguid seada WHERE-lausega nii gruppide jaoks on piiranguid tähistav sõna HAVING.

```
mysql> select nimi, count(*) as kogus from teated2 group by nimi having kogus>1;

+------+-------+
| nimi | kogus |
+------+-------+
| Kati |     3 |
| Mati |     3 |
+------+-------+
2 rows in set (0.01 sec)
```

Ja saidki ühe tabeliga katsetused ühele poole.

```
mysql>
```

Kaks tabelit

Andmebaaside puhul peetakse tähtsaks iga sisestatud väärtust hoida vaid ühe eksemplarina. Et iga uue kauba ostmisel ei peaks uuesti kirja panema kliendi aadressi või konto numbrit. Kui on vaja sisestust kontrollida, siis kasutatakse selleks mõnd kontrollsummat või muud piirajat, kuid ideaaljuhul samu andmeid andmebaasis mitmes kohas ei hoita. Sama lugu nagu koodilõikude puhul: mis kord tehtud, seda uuesti kirjutada pole hea.

Tabelid seotakse omavahel üldjuhul täisarvudega. Kui allpool loodi tabelid toitude ja jooksjate tarvis ning jooksjate tabelis olev tulp lemmiktoidu_id näitab toidutabeli vastava ID-numbriga toidule, siis toitude tabeli ID-tulpa nimetatakse primaarvõtmeks ning tulba lemmiktoidu_id väärtusi võõrvõtmeks.

```
CREATE TABLE jooksjad (ID int NOT NULL AUTO_INCREMENT, eesnimi varchar(30),
lemmiktoidu_id int, PRIMARY KEY(ID));

mysql> CREATE TABLE toidud(ID int NOT NULL AUTO_INCREMENT, nimetus varchar(30), PRIMARY
KEY(ID));
```

Tabelitesse mõned väärtused, et oleks pärast mille peal katsetada. Esimesele toidule pannakse automaatselt järjekorranumbriks 1

```
mysql> insert into toidud(nimetus) values ('Hernesupp');
Query OK, 1 row affected (0.10 sec)
```

Ka jooksja id-number pannakse automaatselt. Juku lemmiktoidu number tuleb aga määrata.

```
mysql> INSERT INTO jooksjad(eesnimi, lemmiktoidu_id) values ('Juku', 1);
```

```
Query OK, 1 row affected (0.00 sec)
```

Väljatrükk näitamaks, milliste andmetega edaspidi katsetatakse. Matile on jäetud lemmiktoit määramata ning selle välja väärtuseks on NULL.

```
mysql> select * from jooksjad;
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
+----+---------+----------------+
|  1 | Juku    |              1 |
|  2 | Kati    |              1 |
|  3 | Mati    |           NULL |
+----+---------+----------------+
3 rows in set (0.01 sec)

mysql> SELECT * FROM toidud;
+----+------------+
| ID | nimetus    |
+----+------------+
|  1 | Hernesupp  |
|  2 | Kapsasupp  |
|  3 | Pannkoogid |
+----+------------+
3 rows in set (0.01 sec)
```

Kõige tavalisem päring paigutamaks ühte tabelisse nii jooksjad kui nende lemmiktoidud. WHERE-osa puudumisel antaks välja kõikvõimalikud kahe tabeli ridade omavahelised kombinatsioonid. Praegusel juhul 3*3 ehk üheksa rida. WHERE seab aga piirangu ning välja näidatakse vaid kaks - need, kus jooksja lemmiktoidu_id vastab toitude tabelis leiduvale ID-veeru väärtusele. Et Mati juures olevat NULL-väärtust toitude tabeli ID-väärtuste hulgas pole, siis jääb Mati ka nimekirja kuvamata.

```
mysql> SELECT * FROM jooksjad, toidud WHERE jooksjad.lemmiktoidu_id=toidud.ID;

+----+---------+----------------+----+-----------+
| ID | eesnimi | lemmiktoidu_id | ID | nimetus   |
+----+---------+----------------+----+-----------+
|  1 | Juku    |              1 |  1 | Hernesupp |
|  2 | Kati    |              1 |  1 | Hernesupp |
+----+---------+----------------+----+-----------+
2 rows in set (0.06 sec)
```

Kui soovitakse kõiki ühe tabeli väärtusi näha ning paremale poole lisada mittetühjadena vaid need väärtused, mida võõrvõtme kaudu võimalik leida on, siis aitab tabeleid ühendada LEFT JOIN. ON-lauseosas tuleb siis määrata, millised tulbad omavahel seotud on. Suuremate rakenduste korral võib nõnda kokku ühendada tunduvalt rohkem kui kaks tabelit. Juhul, kui näiteks soovitakse uurida, millised sobivas vanuses inimesed töötavad ettevõttes, mille leidub filiaal ka Pärnus.

```
mysql> SELECT * FROM jooksjad LEFT JOIN toidud ON toidud.id=jooksjad.lemmiktoidu_id;
+----+---------+----------------+------+-----------+
| ID | eesnimi | lemmiktoidu_id | ID   | nimetus   |
+----+---------+----------------+------+-----------+
|  1 | Juku    |              1 |    1 | Hernesupp |
|  2 | Kati    |              1 |    1 | Hernesupp |
|  3 | Mati    |           NULL | NULL | NULL      |
+----+---------+----------------+------+-----------+
3 rows in set (0.00 sec)
```

Nii nagu failinimede küsimisel aitasid elu hõlpsamaks teha tärn ning küsimärk, nii saab MySQLi puhul kasutada LIKE-võrdluses protsenti ning alljoont.

```
mysql> select * from jooksjad where eesnimi like 'J%';
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
+----+---------+----------------+
|  1 | Juku    |              1 |
+----+---------+----------------+
1 row in set (0.01 sec)


mysql> select * from jooksjad where eesnimi like 'J_ku';
+----+---------+----------------+
| ID | eesnimi | lemmiktoidu_id |
```

```
+----+---------+----------------+
|  1 | Juku    |              1 |
+----+---------+----------------+
1 row in set (0.00 sec)

mysql> select * from jooksjad where eesnimi like 'J_k';
Empty set (0.01 sec)
```

SQL-keelest võib leida ka komplekti muudeski keeltes kasutatavaid funktsioone. Olgu siis arvutamise või tekstitöötluse tarbeks.

```
mysql> select eesnimi, length(eesnimi) from jooksjad;
+---------+-----------------+
| eesnimi | length(eesnimi) |
+---------+-----------------+
| Juku    |               4 |
| Kati    |               4 |
| Mati    |               4 |
+---------+-----------------+
3 rows in set (0.08 sec)

mysql> select eesnimi, left(eesnimi, 1) from jooksjad;
+---------+------------------+
| eesnimi | left(eesnimi, 1) |
+---------+------------------+
| Juku    | J                |
| Kati    | K                |
| Mati    | M                |
+---------+------------------+
3 rows in set (0.07 sec)
```

Lauluandmetega rakendus

Järgnevalt kinnistatakse eelpool kirja pandud tarkused väikese programmi abil. Andmed on esiotsa ühes ja pärast kolmes tabelis. Ning väljund saadetakse nii tekstiekraanile kui veebilehtedele.

Alustame võimalikult lihtsast väljamõeldud olukorrast, kus soovitakse meeles pidada laulude pealkirju ning laulude esitajaid. Viiekümnest tähest kummagi välja salvestamisel võiks piisata. Nagu tavaks, lisatakse igale tabelireale ka võtmeväli, et oleks hiljem kindlasti võimalik kontreetsetele ridadele viidata.

```
CREATE TABLE laulud (
  id int(11) NOT NULL auto_increment,
  pealkiri varchar(50) default NULL,
  esitaja varchar(50) default NULL,
  PRIMARY KEY  (id)
)
```

Mõned väljamõeldud andmed sisse

```
INSERT INTO laulud VALUES (1,'Valged Roosid','Joala');
INSERT INTO laulud VALUES (2,'Kuldkannike','Joala');
INSERT INTO laulud VALUES (3,'Mererannal','Linna');
INSERT INTO laulud VALUES (4,'Kungla Rahvas','Veskimaja');
INSERT INTO laulud VALUES (5,'Koolisellid','Tammik');
```

ning võibki tulemust imetleda.

```
mysql> select * from laulud;
+----+---------------+-----------+
| id | pealkiri      | esitaja   |
+----+---------------+-----------+
|  1 | Valged Roosid | Joala     |
|  2 | Kuldkannike   | Joala     |
|  3 | Mererannal    | Linna     |
|  4 | Kungla Rahvas | Veskimaja |
|  5 | Koolisellid   | Tammik    |
+----+---------------+-----------+
5 rows in set (0.30 sec)
```

Eeltoodud näidete põhjal saab andmeid väljastava käsureaprogrammi kokku küllalt lihtsalt – juhul kui tarvilikud ühendused on valmis seatud.

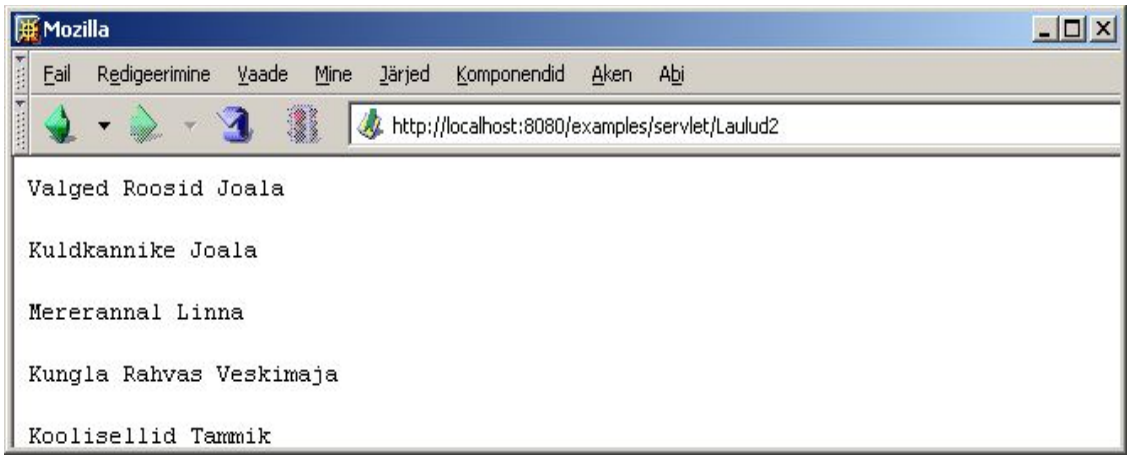
```
import java.sql.*;

public class Laulud1{
 public static void main(String argumendid[]) throws Exception{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
  Statement st=cn.createStatement();
  String lause="SELECT pealkiri, esitaja FROM laulud";
  ResultSet rs=st.executeQuery(lause);
  while(rs.next()){
    System.out.println(rs.getString("pealkiri")+
                              " "+rs.getString("esitaja"));
  }
  cn.close();
 }
}
```

Kui programm tööle panna, võib ka väljundit näha.

```
C:\temp>Java Laulud1
Valged Roosid Joala
Kuldkannike Joala
Mererannal Linna
Kungla Rahvas Veskimaja
Koolisellid Tammik
```

Kui andmeid rohkem või kliendid üle võrgu kaugemal, siis muudab servletiväljund andmed kergemini kättesaadavaks. Kui veebiserver jookseb avalikult kättesaadavas masinas, siis piisab kasutajal teada vaid rakenduse aadressi ning võibki omale vajaliku teabe ekraanilt ammutada.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud2 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/plain");
      PrintWriter valja = vastus.getWriter();
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, esitaja FROM laulud";
        ResultSet rs=st.executeQuery(lause);
        while(rs.next()){
          valja.println(rs.getString("pealkiri")+
                    " "+rs.getString("esitaja")+"\n");
        }
        cn.close();
      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



Standardile vastav lehekülg.

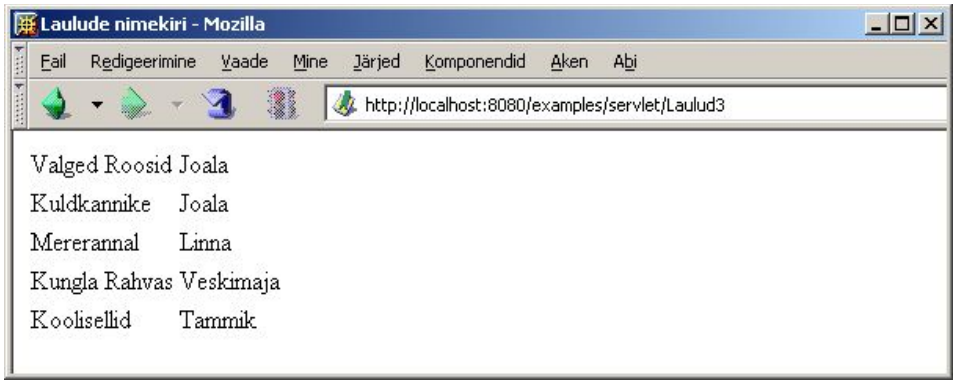
HTMLi keel on aastate jooksul arenenud. Algsest kümnekonna käsuga tekstiillustreerimisvahendist kasvas välja sajakonna käsuga kujundusvahend. Seiluritootjad on omalt poolt võimalusi lisanud ning aegapidi on neid ka standardisse võetud. W3-konsortsium on taoline firmade ja muude asutuste ühendus, mille kaudu lepitakse kokku veebiga seotud standardeid. Nõnda on rohkem lootust, et kusagil koostatud leheküljed või muud failid ka mujal kasutatavad on.

Standard pannakse kirja kas tekstilise kirjeldusena, tabelina, XML-failide puhul ka DTD või Schema abil. Programmeerija loeb kirjeldust ja püüab tulemuse selle järgi sättida, kuid iga inimene tahab ja vajab tagasisidet, et kas tema koostatu ka loodetud ootustele vastab. Kui Pascali, C või Java koodi kirjutada, siis teatab kompilaator süntaksivead julgesti välja. Veebilehte avades aga on seilur tagasihoidlikum ning väikesed näpuvead jäävad enamasti märkamata. Mõnikord võib aimamine nii ilusti välja tulla, et lehte vaadates ei leia mingit märki HTML-koodi trükiveast. Teises seiluris võib aga aimamise algoritm muud moodi käituda ning lehe sisu võib imelikult paista või sootuks märkamatuks jääda. Taoliste viperuste vältimiseks saab kasutada HTMLi validaatorit - programmi kontrollimaks HTMLi õigekirja. Siin uuritakse, et elementide nimed oleks õigesti kirjutatud, lõpetamist vajavad elemendid lõpetatud ning et elemendid paikneksid ka üksteise sees lubatud kujul.

Et validaator teaks millise standardi järgi kontrollida, peab vastav rida ka faili alguses kirjas olema. Lisaks on päises nõutud ka pealkirja ja kooditabeli märkimine. Edasi mõistab validaatorprogramm ülal kirjeldatud standardi järgi lehte kontrollima hakata.

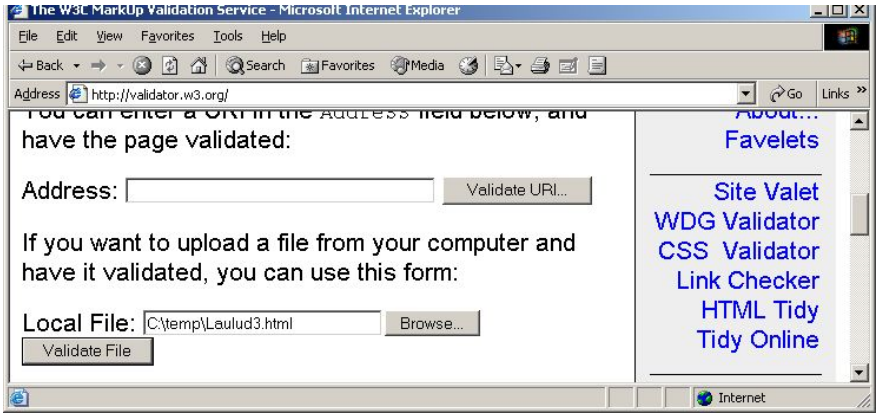
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud3 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, esitaja FROM laulud";
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        while(rs.next()){
         valja.println("<tr><td>"+rs.getString("pealkiri")+
```

```
          "</td><td>"+rs.getString("esitaja")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



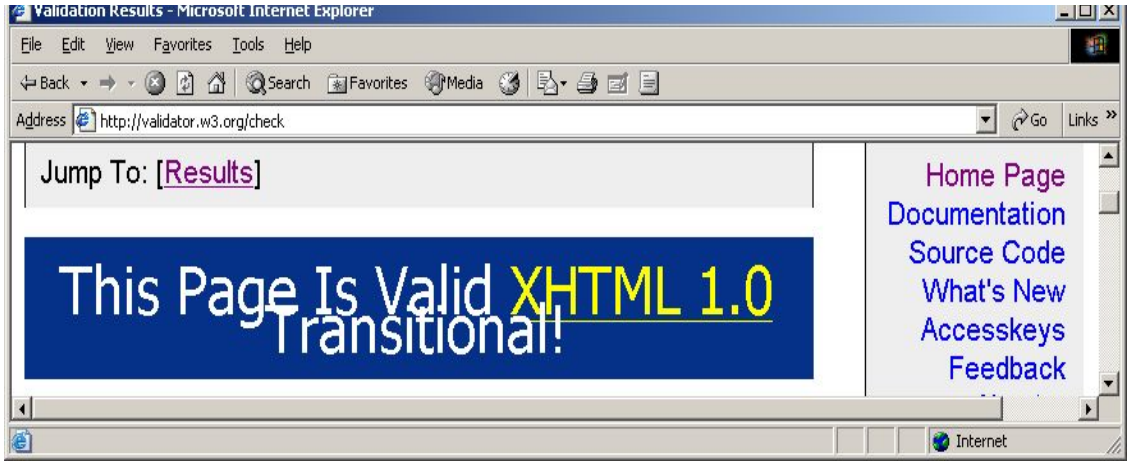

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Laulude nimekiri</title></head>

<body>
<table>
<tr><td>Valged Roosid</td><td>Joala</td></tr>
<tr><td>Kuldkannike</td><td>Joala</td></tr>
<tr><td>Mererannal</td><td>Linna</td></tr>
<tr><td>Kungla Rahvas</td><td>Veskimaja</td></tr>
<tr><td>Koolisellid</td><td>Tammik</td></tr>
</table>
</body>
</html>
```

Kontrollija leiab aadressilt http://validator.w3.org/ Seal võimalik kontrollimiseks pakkuda nii veebiaadress kui oma kettal leiduv fail. Esimest võimalust on mugav kasutada, kui koostatav leht avalikult veebi kaudu kättesaadav. Kui aga kohalik veebiserver otse avalikku võrku ei paista, siis saab servleti loodud HTML-faili kohalikku masinasse salvestada ning siis võrku üles laadida.



Leht kontrollijasse loetud, antakse sealt vastus. Kui päiserida puudu või vigane või mõni tähtsam element puudu, siis antakse vastav teade. Suudab aga validaator lehe struktuurist aru saada, siis vaadatakse kõik kujunduselemendid ükshaaval üle ning kui kusagil midagi kahtlast, siis antakse teada. Mitmete teineteise sees paiknevate tabelite või taanete korral oleks käsitsi käskude ja nende paiknemise

õigsuse kontroll küllaltki vaevaline. Validaator aga leiab kohe üles kui mõni märk vales kohas, üle või puudu. Siis ei jää muud midagi üle, kui veateadet uurida, leida, et mille poolest siis loodud HTML kahtlane on. Koodis püüda koht parandada ning uuesti proovida. Kui pilt tundub väga segane ning kuidagi ei oska veateate asukohta leida, siis aitab jupikaupa uurimine. Et algul tõsta uude tühja faili vaid päise ja tühja body-osaga leht ning kontrollida selle korrektsust. Siis tasapisi lisada/kopeerida sisemisi elemente ja iga sammu järel kehtivust kontrollida. Nii on kohe selge, millise sammu juures raskus tekkis ning võimalik seda sammu lähemalt uurida. Vajadusel osadeks jagada ning uuesti uurida. Mõnikod võib ka juhtuda, et pealtnäha oleks justkui kõik korras, aga sellegipoolest näidatakse, et ühes kindlas kohas on midagi viltu. Sellisel juhul võib põhjuseks olla miski klahvikombinatsiooni tulemusena tekkinud salapärane sümbol, millest siis kustutamise ja uuesti kirjutamise abil võitu saab. Ning kui lõpuks õnnestub validaatorilt välja meelitada teade lehekülje korrektsuse kohta, siis võib tulemusega rahule jääda. Ehkki tasuks korrektsust kontrollida ka mitmesuguste andmete põhjal sama servleti loodud lehtede puhul.



Sortimine

Vähegi pikemate loetelude puhul võib sobiva väärtuse leidmine päris tülikaks osutuda. Find-käsu kõrval on sobivaks vahendiks järjestamine. Andmebaasi põhjal toimivate rakenduste eeliseks on võimalus paari sõna abil määrata väljastatavad andmed soovitavasse järjekorda. Suuremaks nuputamiseks on, et mille põhjal seda järjestust määrata. Lihtsam tundub olema jagada ülesanne kaheks osaks. Ühelt poolt valmistada ette lehekülg, mis väljastaks andmed etteantud parameetritele vastavas järjekorras. Ning teiselt poolt hoolitseda, et kasutajal oleks võimalikult mugav järjestust määrata. Esimeses lähenduses võib andmeid väljastavat lehte vaadelda kui musta kasti. Servletile antakse andmeid ette nagu veebilehtede puhul ikka - aadressirea parameetrite kaudu. Määran, et parameetri nimeks oleks "sorttulp" ning väärtuseks tulba nimi, mille järgi soovitatakse sortida. Seega siis, kui lehe aadressiks oleks

```
http://masinanimi/kataloog/servletinimi?sorttulp=esitaja
```

siis tuleksid andmed lehele järjestatuna tähestikulises järjekorras esitaja järgi. Kui aga

```
http://masinanimi/kataloog/servletinimi?sorttulp=pealkiri
```

siis tuleksid tähestiku algupoolel asuvad pealkirjad eespool nähtavale.

SQL-keeles määratakse sorteerimist määrav tulp lauseosa ORDER BY järgi. Lihtsaim lahendus oleks kasutaja poolt tulnud tulba nimi otse sellesse lausesse paigutada ning tulemus välja näidata. Et aga veebirakenduste puhul soovitakse kõiki veebi poolt tulevaid andmeid umbusaldada, siis saabuvat parameetrit SQL-lausesse ei kirjutata. Esiteks tekiks probleem pahatahtliku sisestuse korral, mis võiks sobivalt seatuna muudele andmetele liiga teha või neid välja näidata. Teiseks mureks on aga, et vigase sisestuse korral tuleks nähtavale süsteemi loodud veateade või sootuks mitte midagi. Viisakam oleks aga kasutajale sõnaliselt teada anda, mis lahti on.

Siin näites on sorteerimist määrava tulba nimi küsitud muutujasse sort. Vaikimisi väärtuseks on "id". On aga parameetriks pealkiri või esitaja, siis määratakse see sorteerimist seadvaks tulbaks. Nõnda pole võimalust, et kasutaja saadetud suvaline sisestus andmeid võiks rikkuda või kahtlase veateate

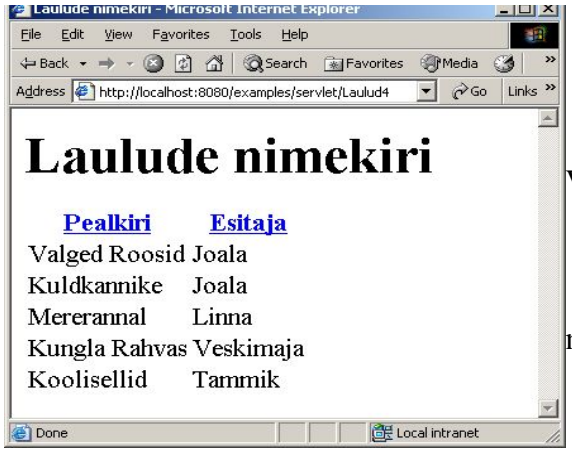
ekraanile manada, vaid igasuguse vigase sisestuse korral näidatakse ekraanile read sorteerituna id järgi. Kui sobiva tulba nimi lauses olemas, siis võib andmed ekraanile näidata nagu eelmiselgi korral.

Teiseks tuleb miskil kasutajale mugavamal ja vastuvõetavamal moel anda võimalus sorteerimisjärjekord valida. Ning tulba nime aadressireal sissetippimine ei pruugi selleks mitte olla. Siinne

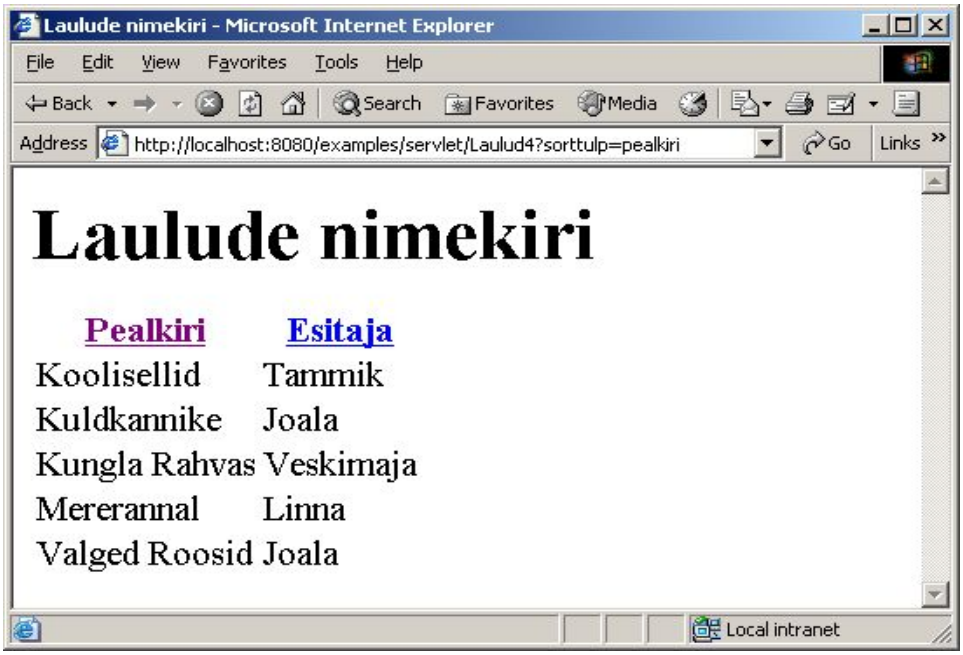
```
"<th><a href='"+kysimus.getRequestURI()+"?sorttulp=pealkiri'>Pealkiri</a></th>"+
```

näitab, kuidas võimalik viide nõnda koostada, et see samale lehele näitaks ning sorteerimiseks sobiva tulba nimi andmetena kaasa antaks. HttpServletRequesti käsklus getRequestURI annab tekstina välja jooksva lehe URLi alates serveri juurkataloogist, piisab juurde lisada vaid sobivad parameetrid. Teise võimalusena saaks jooksva kataloogi puhul kirjutada ka vaid failinime. Kui iga tulba pealkirjal taoline viide küljes, jääbki kasutajale võimalus, et igale pealkirjale vajutades sorteeritakse andmed vastava tulba järgi.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud4 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        valja.println("<tr>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=pealkiri'>Pealkiri</a></th>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=esitaja'>Esitaja</a></th>"+
           "</tr>");
        while(rs.next()){
         valja.println("<tr><td>"+rs.getString("pealkiri")+
          "</td><td>"+rs.getString("esitaja")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



Vaikimisi järjestus id järgi ning määratud järjestus pealkirja järgi.



Andmete lisamine

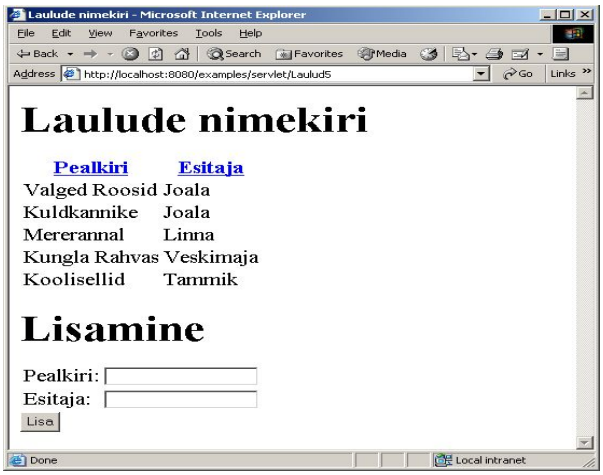
Lihtsa otsinguvahendi puhul võib baasi veebiliides piirduda andmete välja näitamisega. Muutmised-lisamised võetakse ette andmebaasi enese vahenditega, või on selle tarvis hoopis omaette rakendus loodud. Turvalisuse mõttes on veebist vaid vaatamist lubav rakendus lihtsam - pahalastel pole kuigivõrd põhjust ega võimalust andmeid ohtu seada. Ohtudeks jäävad vaid serveri või võrguühenduse võimsust ületav päringute hulk või logifailide abil ketta täiskirjutamise oht. Kui ka veebi kaudu liituval kasutajanimel pole andmebaasis muutmise õigusi, siis võib taolisel rakendusel suhteliselt mureta toimida lasta.

Nõudmiste kasvades aga vaid vaatamiseks mõeldud rakendusest ei piisa. Andmete veebikaudseks lisamiseks on võimalused täiesti olemas, lihtsalt peab arvestama pahatahtlike kasutajate tekitatud segaduste ohuga.

Andmete lisamiseks tuleb need kasutajalt kõigepealt kätte saada. Praegusel juhul on selleks otstarbeks tarvitatud tekstivälju. Graafikakomponendid jällegi vormi sees. Atribuut action='#' tähendab jällegi, et andmete vastuvõtjaks on sama leht uuel avamisel. Võrreldes muude näidetega, on siin ka submit-nupule nimi antud. Vormi andmete teele saatmisel pannakse kaasa nimega komponentide andmed, subm
it-nupu puhul vajutatud nupu andmed. Nõnda on võimalik kontrollida, kas nuppu vajutati. Ning mitme nupu puhul kontrollida, et millist nuppu vajutati.

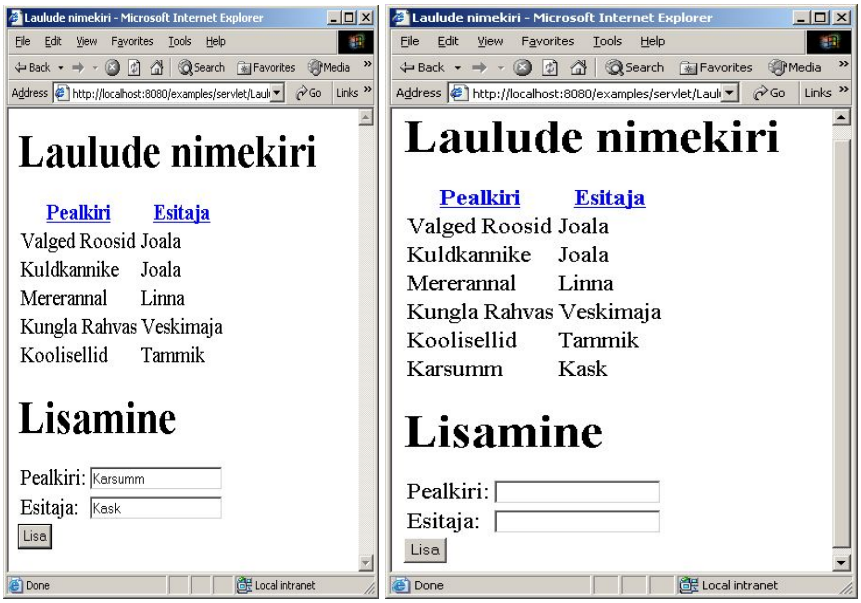
Siinse lehe avamisel kontrollitakse, kas parameeter "lisamine" pole null. Et kas vajutati lisamisnuppu. Vaid sel juhul koostatakse SQL-lause andmete lisamiseks ning käivitatakse. PreparedStatement'i eelis tavalise käskluse ees on, et kasutaja saadetud erisümbolid ei saa serveris segadust tekitada, vaid need kirjutatakse samasuguse rahuga edasi andmebaasi tabelisse. Muu andmete näitamise osa sarnane kui eespool.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud5 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                              "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud (pealkiri, esitaja) VALUES (?, ?)"
           );
           ps.setString(1, kysimus.getParameter("pealkiri"));
           ps.setString(2, kysimus.getParameter("esitaja"));
           ps.executeUpdate();
        }
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        valja.println("<tr>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=pealkiri'>Pealkiri</a></th>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=esitaja'>Esitaja</a></th>"+
           "</tr>");
        while(rs.next()){
         valja.println("<tr><td>"+
          Abi.filtreeriHTML(rs.getString("pealkiri"))+
          "</td><td>"+
          Abi.filtreeriHTML(rs.getString("esitaja"))+
          "</td></tr>");
        }
        valja.println("</table>");
        valja.println("<h1>Lisamine</h1>");
        valja.println(
          "<form action='#'><table>"+
          "<tr><td>Pealkiri:</td><td>"+
           "<input type='text' name='pealkiri' /></td></tr>\n"+
          "<tr><td>Esitaja:</td><td>"+
           "<input type='text' name='esitaja' /></td></tr>\n"+
          "</table>"+
          "<input type='submit' name='lisamine' value='Lisa' />"+
          "</form>"
        );
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```



Lisamisleht

- Algul
- Väärtuste sisestamine
- Pikenenud loetelu



Mitme tabeliga rakendus

Märgatav osa lihtsatest veebirakendustest võibki ühe tabeliga piirduda. Harilikuks andmete lisamiseks ning välja näitamiseks piisab sellest sageli. Kui mõista sobivalt pealkirju valida ning mõnikord ka andmete põhjal tulemusi arvutada, siis võib ühe tabeliga näite edukalt kokku panna nii külalisraamatu, tunniplaani, sünnipäevade hoidla kui muudki. Keerukamate andmete puhul ei pruugi aga kõige ühes tabelis hoidmine kuigi ökonoomne olla.

Andmebaasiteoreetikud soovitavad, et võimaluse korral tuleks andmed sättida nii, et midagi korduvalt ei hoitaks. Andmete korrektsuse jaoks kasutatagu pigem kontrollsummasid, kui mingi väärtus aga kusagil kirjas, siis sama asja poleks võimaluse korral mõistlik enam kusagile mujale kirjutada. Ehk sama põhimõte nagu koodigi kirjutamise juures: mis kord tehtud, seda püüa ka tulevikus kasutada, mitte sama asja uuesti kirja panema hakata.

Liiatigi võib vaja minna küllalt erinevaid andmeid. Näiteks inimeste ja bussiliinide andmeid oleks teoreetiliselt võimalik hoida samas tabelis, kuid üldjuhul ei tundu see loogiline. Ehkki taolist andmete segamist vahel kasutatakse, ei tohiks see mitte harilike rakenduste puhul tavaks saada. Pigem tulevad kindla semantikata tabelitulbad ette rakendustes, kus programmeerija mõtleb objektide tasandil ning andmete talletamise ja väljalugemise eest otsustab eraldi koostatud vahelüli. Siin aga püüame tabeli andmed mõistetavad hoida ning nende kohale üheselt mõistetava veebiliidese koostada.

Lisaks laulude andmetele koostame tabelid heliplaatide tarbeks: väljadeks plaadi nimi ning väljalaskeaasta.

```
CREATE TABLE plaadid (
  id int(11) NOT NULL auto_increment,
  plaadinimi varchar(30) default NULL,
  aasta int(11) default NULL,
  PRIMARY KEY  (id)
);
```

Ning mõned andmed ka sisse.

```
INSERT INTO plaadid VALUES (1,'Tantsulood',1985);
INSERT INTO plaadid VALUES (2,'Laululood',1988);
```

Samuti loome juurde tabeli, mille abil määratakse, millised lood millise plaadi juurde kuuluvad. Seosetabelisse otseseid tekste ei kirjutatagi. Tabelis on kolm tulpa: id, plaadi_id ning laulu_id. Iga taoline rida märgib üht seost, kus märgitakse, et vastava reanumbriga laul kõlab märgitud numbriga plaadil.

Sellist teise tabeli reale viitava lahtri väärtust nimetatakse võõrvõtmeks (foreign key). MySQL neljandas tavaversioonis veel ei kontrolli võõrvõtmete viidete korrektsust, see jäetakse programmeerija hooleks. Mõne tabeliga ja paarisaja andmereaga rakenduse puhul saab sellise järje pidamisega täiesti hakkama. Kui aga andmete hulk ja keerukus märkimisväärselt kasvab, siis aitab andmebaasi võime viidete korrektsust kontrollida süsteemi töökindlust tagada ning murdunud ja vigastest viidetest hoiduda.

```
mysql> CREATE TABLE laulud_plaadid(id INT NOT NULL auto_increment PRIMARY KEY, laulu_id
INT, FOREIGN KEY lauluvoti (laulu_id) REFERENCES laulud (id), plaadi_id
INT, FOREIGN KEY plaadivoti(plaadi_id) REFERENCES plaadid(id));
```

Kui võõrvõtme loomist käsuna eraldi mitte kirja panna, siis tegelikult on tegemist lihtsa kolme tulbaga tabeliga.

```
CREATE TABLE laulud_plaadid (
  id int(11) NOT NULL auto_increment,
  laulu_id int(11) default NULL,
  plaadi_id int(11) default NULL,
  PRIMARY KEY  (id)
)
```

Järgnevalt näide, kuidas võivad andmed tabelites paikneda. Kõigepealt laulude tabel:

```
mysql> select * from laulud;
+----+---------------+-----------+
| id | pealkiri      | esitaja   |
+----+---------------+-----------+
|  1 | Valged Roosid | Joala     |
|  2 | Kuldkannike   | Joala     |
|  3 | Mererannal    | Linna     |
|  4 | Kungla Rahvas | Veskimaja |
|  5 | Koolisellid   | Tammik    |
|  6 | Karsumm       | Kask      |
+----+---------------+-----------+
6 rows in set (0.03 sec)
```

Siis plaadid:

```
mysql> select * from plaadid;
+----+------------+-------+
| id | plaadinimi | aasta |
+----+------------+-------+
|  1 | Tantsulood |  1985 |
|  2 | Laululood  |  1988 |
+----+------------+-------+
2 rows in set (0.05 sec)
```

Ning edasi seosetabel, milliseid laule millistelt plaatidelt leida võib. Näiteks viimasest reast annab välja lugeda, et seos id-numbriga 3 teatab, et laulu number 4 ehk “Kungla rahvas” võib kuulda plaadilt number 1 ehk “Tantsulood”.

```
mysql> select * from laulud_plaadid;
```

```
+----+----------+-----------+
| id | laulu_id | plaadi_id |
+----+----------+-----------+
|  1 |        1 |         1 |
|  6 |        2 |         2 |
|  3 |        4 |         1 |
+----+----------+-----------+
3 rows in set (0.06 sec)
```

Hoolimata sellest, et tabelitesse laiali jagatuna võib andmete paiknemine keerukas tunduda, annab SQL-lausetega väärtused küllalt sobival kujul välja küsida. Tabeleid ühendavaid lauseid annab mitmel kujul kirja panna, üks neist näha järgnevalt; seletus:

Küsimiseseks SELECT-lause nagu ikka. Järgneb näha soovitavate tulpade loetelu. Mõnikord võivad tulpade nimed eraldi tabelitest kattuda. Sellisel juhul tuleb määrata tulba nimi koos tabeli nimega - näiteks kujul plaadid.id . Edasi järgneb FROM ning kasutatavate tabelite loetelu. Edasi tuleb seada, milliste tabelite millised tulbad omavahel seotud on. Kuna praegu tabeli laulud_plaadid laulu_id näitab laulude tabeli id-veerule ning laulud_plaadid plaadi_id näitab plaatide tabeli id-veerule, siis see tuleb siin ka kirja panna.

```
mysql> SELECT pealkiri, plaadinimi FROM laulud, laulud_plaadid, plaadid WHERE
laulud.id=laulud_plaadid.laulu_id AND laulud_plaadid.plaadi_id=plaadid.id;
```

Päringu tulemuseks nagu ikka - andmed tabeli kujul.

```
+---------------+------------+
| pealkiri      | plaadinimi |
+---------------+------------+
| Valged Roosid | Tantsulood |
| Kungla Rahvas | Tantsulood |
| Kuldkannike   | Laululood  |
+---------------+------------+
3 rows in set (0.03 sec)
```

Edasi tuleb vaid servlet päringule ümber kirjutada ning tulemus lehele välja näidata. Väärtusi saab küsida sarnaselt tulba nime järgi nagu ühestki tabelist andmeid korjava päringu korral.

```
       valja.println("<tr><td>"+rs.getString("pealkiri")+
            "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
```

Teiseks võimaluseks oleks väärtuste küsimine tulba järjekorranumbri järgi. Eriti on sellest kasu juhul, kui tabelite liitmise tulemusena võivad loetellu sattuda samanimelised tulbad. Samuti tasuks väärtusi küsida tulpadega samas järjekorras. Mõne draiveri ja baasi puhul on võimalik väärtusi küsida vaid ühes järjekorras: tulpi vasakult paremale ning ridu ülevalt alla. Kui pole olulist põhjust taolisest järjestusest kõrvale hoidmiseks, siis tasuks seda järgida. Siis tekib vähem probleeme rakenduse teise serverisse või teise baasi külge tööle panekul.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil1 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
```

```
        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, plaadinimi "+
            "FROM laulud, laulud_plaadid, plaadid "+
            "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
            "laulud_plaadid.plaadi_id=plaadid.id";
        valja.println("<table>");
        ResultSet rs=st.executeQuery(lause);
        while(rs.next()){
           valja.println("<tr><td>"+rs.getString("pealkiri")+
            "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
        }
        valja.println("</table>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
   }
}
```

Ning servleti töö tulemuseks on ilus lihtne tabel.

Nõnda nagu laule üldloendisse lisatakse, nii ka tuleb luua võimalus määramiseks, millised laulud millistele plaatidele lindistatud on. Ehkki tehniliselt tuleb vaid laule ja plaate siduvasse tabelisse lisada kaks id-numbrit, on kasutajal mugavam nime järgi valida, milline plaat ja milline pala omavahel ühendatud on.

Et nii laulude kui plaatide nimed juba kirjas, siis pole neid mõistlik enam uuesti sisse kirjutada. Kuni andmeil on kuni mõniteist, siis võiks sobiva väärtuse välja valimiseks sobida rippmenüü. Suuremate mahtude puhul tuleks kasutada järjestamist, raadionuppe või otsinguvahendit. Siinne näide töötab rippmenüüga. Samuti on mõistlik lasta kasutajal valida väärtuste hulgast, tegelikult serverisse salvestamiseks saata aga id-numbrid. Rippmenüül tuleb selleks iga väärtuse (option) juurde kirjutada väärtus (value), mida siis tegelikult serverisse saata soovitakse.

```
          valja.println("  <option value='"+rs.getString("id")+
"'>"+rs.getString("plaadinimi")+"</option>");
```

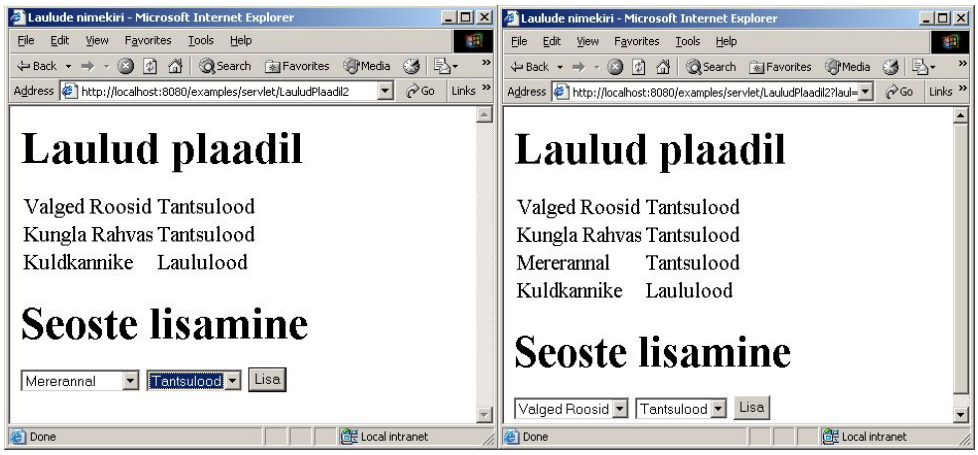
Selline ID-numbrite järgi valimine ja andmete näitamine lihtsustaks ka näiteks rakenduse tõlkimist: kui andmeid oleks vaja mitmes keeles näidata, siis numbrid võiksid samaks jääda, tõlkida oleks tarvis vaid sõnu.

Lisamine näeb laulude tabeliga võrreldes välja küllalt sarnane. Et servleti parameetritena saabuvad vaid ühendatava laulu ja plaadi ID-numbrid ning need numbrid ongi vaja seosetabelisse kirjutada, siis pole muud kui väärtused tabelisse kirjutada. Et andmete lisamine on lehel tulemuste näitamisest eespool, siis võib juba kohe pärast lisamist näha, et valitud laulu ja plaadi paar ka loetellu on ilmunud.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil2 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
```

```
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
       if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud_plaadid (laulu_id, plaadi_id) VALUES (?, ?)"
           );
           ps.setInt(1, Integer.parseInt(kysimus.getParameter("laul")));
           ps.setInt(2, Integer.parseInt(kysimus.getParameter("plaat")));
           ps.executeUpdate();
        }

        Statement st=cn.createStatement();
        String lause="SELECT pealkiri, plaadinimi "+
            "FROM laulud, laulud_plaadid, plaadid "+
            "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
            "laulud_plaadid.plaadi_id=plaadid.id";
        valja.println("<table>");
        ResultSet rs=st.executeQuery(lause);
        while(rs.next()){
           valja.println("<tr><td>"+rs.getString("pealkiri")+
            "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
        }
        valja.println("</table>");
        valja.println("<h1>Seoste lisamine</h1>");
        valja.println("<form action='#'>");
        lause="SELECT id, pealkiri FROM laulud";
        rs=st.executeQuery(lause);
        valja.println("<select name='laul'>");
        while(rs.next()){
          valja.println("  <option value='"+rs.getString("id")+
              "'>"+rs.getString("pealkiri")+"</option>");
        }
        valja.println("</select>");

        lause="SELECT id, plaadinimi FROM plaadid";
        rs=st.executeQuery(lause);
        valja.println("<select name='plaat'>");
        while(rs.next()){
          valja.println("  <option value='"+rs.getString("id")+
              "'>"+rs.getString("plaadinimi")+"</option>");
        }
        valja.println("</select>");
        valja.println("<input type='submit' name='lisamine' value='Lisa' />");
        valja.println("</form>");
        cn.close();
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
   }
}
```

Järgnevatel piltidel võibki imetleda, kuidas lisati laul “Mererannal” plaadile “Tantsulood”,



Kustutamine

Enamikel asjadel on nii algus kui ots. Ehkki suuremate andmebaaside puhul soovitatakse andmed kustutamise puhul arhiveerida ning arhiveerimisatribuudi abil määrata, et seda rida pole vaja enam arvestada. Nõnda on kergem jälgida baasi arengulugu ning võimalike vigade puhul olukord taastada. Rakendust jälle lihtsam teha ning omal pilt selgem, juhul kui toimimiseks hädavajalikke veerge ja andmeid vähem on.

Arvutikettalt kustudades kipuvad andmed kasutaja jaoks jäädavalt kadunud olema. Kas ja kui täpselt on endist seisu võimalik taastada, sõltub serveri andmete varundamisest ning andmebaasimootori vastavatest võimalustest. Küllalt sageli seetõttu lisatakse pärast kustutatavate andmete märkimist kasutaja tarbeks koht üle kontrollimaks, kas ikka soovitakse kustutada. Siin näites aga on piirdutud lihtsama võimalusega ning lisakontrolli ette ei võeta.

Kustutuskasutajaliidese võimalusi on mitmeid. Lihtsaim on luua arvatavasti lehte, kus iga andmerea taga on viide, millele vajutades vastav rida baasis kustutatakse. Kas siis kustutustoiming tehakse eraldi servleti abil ning siis suunatakse kasutaja loetelulehele tagasi või on kustutusoskused juba loetelulehele sisse ehitatud, see on juba realiseerimise küsimus. Esimese võimaluse eeliseks on, et lehe värskenduse korral ei saadeta kustutusteavet uuesti.

Tehniliselt ligikaudu sama lihtne või keerukas on võimalus lasta kasutajal raadionupuga määrata kustutatav rida. Ikkagi jõuab andmeid vastu võtvale lehele kaasa kustutamist määrav väärtus, enamasti kustutatava rea id-number.

Siin näites aga tehti läbi võimalus, kus kasutaja saab kustutamiseks valida need read, mida ta parajasti soovib. Sellise olukorra lahendus on keeleti erinev. Üheks ning siingi kasutatud võimaluseks on määrata kõigile märkeruutudele sama nimi. Vaid elemendi väärtus on igal puhul erinev.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class LauludPlaadil3 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                          "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulud plaadil</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
       if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud_plaadid (laulu_id, plaadi_id) VALUES (?, ?)"
```

```
            );
            ps.setInt(1, Integer.parseInt(kysimus.getParameter("laul")));
            ps.setInt(2, Integer.parseInt(kysimus.getParameter("plaat")));
            ps.executeUpdate();
         }

         Statement st=cn.createStatement();
         String lause="SELECT laulud_plaadid.id as seosenr, pealkiri, plaadinimi "+
             "FROM laulud, laulud_plaadid, plaadid "+
             "WHERE laulud.id=laulud_plaadid.laulu_id AND "+
             "laulud_plaadid.plaadi_id=plaadid.id";
         valja.println("<form action='Kustutus' method='post'><table>");
         ResultSet rs=st.executeQuery(lause);
         while(rs.next()){
            valja.println("<tr>"+
             "<td><input type='checkbox' name='kustutus' value='"+
             +rs.getInt("seosenr")+"' /></td>"+
             "<td>"+rs.getString("pealkiri")+
             "</td><td>"+rs.getString("plaadinimi")+"</td></tr>");
         }
         valja.println("</table><input type='submit' value='Kustuta' /></form>");
         valja.println("<h1>Seoste lisamine</h1>");
         valja.println("<form action='#'>");
         lause="SELECT id, pealkiri FROM laulud";
         rs=st.executeQuery(lause);
         valja.println("<select name='laul'>");
         while(rs.next()){
           valja.println("  <option value='"+rs.getString("id")+
               "'>"+rs.getString("pealkiri")+"</option>");
         }
         valja.println("</select>");

         lause="SELECT id, plaadinimi FROM plaadid";
         rs=st.executeQuery(lause);
         valja.println("<select name='plaat'>");
         while(rs.next()){
           valja.println("  <option value='"+rs.getString("id")+
               "'>"+rs.getString("plaadinimi")+"</option>");
         }
         valja.println("</select>");
         valja.println("<input type='submit' name='lisamine' value='Lisa' />");
         valja.println("</form>");
         valja.println("<br /><a href='Laulud6'>Laulude haldus</a>");
         cn.close();
         valja.println("</body>");
         valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
   }
}
```

Nõnda koostatuna jõuab serverisse sama nimega nii mitu parameetrit, palju kasutaja märkeruute märkinud on. Mõnikord on taoliste andmete välja lugemine keeruline – ka Java puhul suudab käsklus getParameter väljastada vaid parameetri ühe väärtuse. Õnneks on siin kasutada ka käsklus

getParameterValues, mis väljastab küsitud nimega parameetrite väärtused massiivina. Ning siin on näha ka PreparedStatement'i tähtsam kasutusvaldkond. Olukord, kus sarnast lauset tuleb käivitada mitmete andmetega. Kui õnnestub, siis PreparedStatement'i luues harutatakse SQL-tekst lahti masina jaoks kiiremini käsitsetavale kujule (näiteks kahendpuusse) ning järgmistel kordadel kulub energiat vaid uute andmetega käskluse andmebaasiserveris käivitamiseks.

Kustutuslehelt saadetakse kasutaja jälle algsele lehele tagasi.

```
vastus.sendRedirect("LauludPlaadil3");
```

Sel käsklusel on mõtet olukorras, kus midagi pole veel välja trükitud. Nii saab HTTP päiste kaudu teada anda, et soovitakse hoopis järgmise lehe avamist. Ning seiluri ülesandeks on siis sobivat lehte avama hakata.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Kustutus extends HttpServlet {
    public void doPost(HttpServletRequest kysimus,
                       HttpServletResponse vastus)
    throws IOException, ServletException
    {
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement ps=cn.prepareStatement(
          "DELETE FROM laulud_plaadid WHERE id=?"
        );
        String[] vastused=kysimus.getParameterValues("kustutus");
        if(vastused!=null){
         for(int i=0; i<vastused.length; i++){
          ps.setInt(1, Integer.parseInt(vastused[i]));
          ps.executeUpdate();
         }
        }
        cn.close();
        vastus.sendRedirect("LauludPlaadil3");
      }catch(Exception viga){
         viga.printStackTrace(vastus.getWriter());
      }
    }
}
```

Ja võibki näha, kuidas lugu “Mererannal” on taas plaadilt maha võetud.

Laulude haldus

Et laulude andmed hoitakse eraldi ning hoopis teine tabel osutab, milline laul millistel plaatidel paikneb, siis on mugavam ja töökindlam koostada rakendus nõnda, et ka laulude sisestamine ning nende plaatidele määramine on eraldi toimingud. Kui tahta lõppkasutajale sisestamist võimalikult mugavaks teha, siis võib ju püüda nii laulu sisestamine kui plaadile määramine ühele lehele paigutada. Sellisel juhul aga jääb programmi hooleks hoolitseda, et andmed ka sobivalt tabelitesse saaksid ning kogemata ühe laulu andmeid mitut korda ei sisestataks.

Kui aga kasutajate näol pole tegemist väga arvutikaugete inimestega, siis läheb programmeerija töö märkimisväärselt selgemaks, kui on võimalik ühe tabeli andmete lisamise, kustutamise ja muutmisega korraga tegelda, vajadusel lihtsalt kasutajatele teistest tabelitest lisainfot juurde näidates.

Kui nüüd laule plaadile lisades selgub, et mõni laul puudu või valesti kirja pandud, siis tuleb vastava viite kaudu liikuda lihtsalt laulude halduse lehele. Sinna lauludega tegelevale lehele on võrreldes varasemate näidetega paigutatud ka laulude andmete muutmine. Laulude loetelu lehel on lihtsalt juurde tulnud viide LauluMuutusVormile, kus siis pala andmetega edasi tegeldakse.

```
"<a href='LauluMuutusVorm?id="+rs.getInt("id")
```

Nagu näha, antakse vormile kaasa ka laulu ID-number. Siis on vormil teada, millise lauluga tuleb toimetama asuda.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class Laulud6 extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException
    {
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
"+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=ISO-8859-1\" />");
      valja.println("<title>Laulude nimekiri</title></head>\n");
      valja.println("<body><h1>Laulude nimekiri</h1>");
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        Statement st=cn.createStatement();
        if(kysimus.getParameter("lisamine")!=null){
           PreparedStatement ps=cn.prepareStatement(
             "INSERT INTO laulud (pealkiri, esitaja) VALUES (?, ?)"
           );
           ps.setString(1, kysimus.getParameter("pealkiri"));
           ps.setString(2, kysimus.getParameter("esitaja"));
           ps.executeUpdate();
        }
        String sort="id";
        String sortTulp=kysimus.getParameter("sorttulp");
        if(sortTulp==null){sortTulp="id";}
        if(sortTulp.equals("pealkiri")){sort="pealkiri";}
        if(sortTulp.equals("esitaja")){sort="esitaja";}
        String lause="SELECT id, pealkiri, esitaja FROM laulud ORDER BY "+sort;
        ResultSet rs=st.executeQuery(lause);
        valja.println("<table>");
        valja.println("<tr>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=pealkiri'>Pealkiri</a></th>"+
           "<th><a href='"+kysimus.getRequestURI()+
           "?sorttulp=esitaja'>Esitaja</a></th>"+
           "<th>Muutmine</th>"+
           "</tr>");
        while(rs.next()){
         valja.println("<tr><td>"+
          Abi.filtreeriHTML(rs.getString("pealkiri"))+
          "</td><td>"+
```

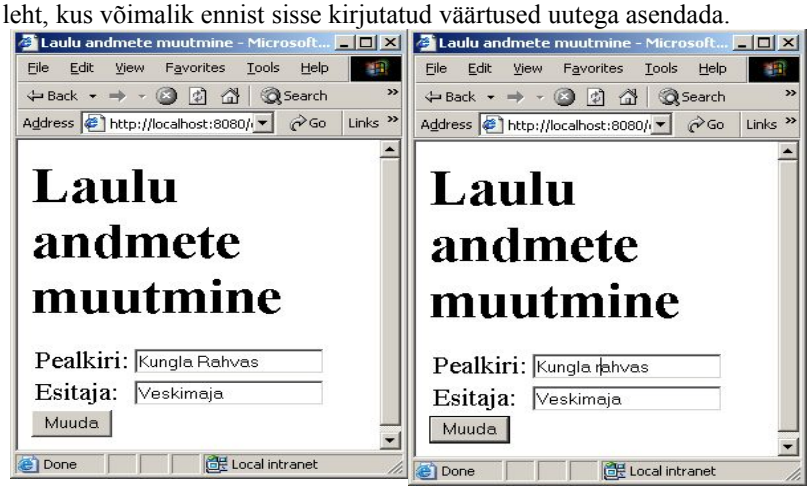
```
          Abi.filtreeriHTML(rs.getString("esitaja"))+
          "</td><td>"+
          "<a href='LauluMuutusVorm?id="+rs.getInt("id")+
          "' />muuda</a></td></tr>");
        }
        valja.println("</table>");
        valja.println("<h1>Lisamine</h1>");
        valja.println(
          "<form action='#'><table>"+
          "<tr><td>Pealkiri:</td><td>"+
           "<input type='text' name='pealkiri' /></td></tr>\n"+
          "<tr><td>Esitaja:</td><td>"+
           "<input type='text' name='esitaja' /></td></tr>\n"+
          "</table>"+
          "<input type='submit' name='lisamine' value='Lisa' />"+
          "</form>"
        );
        cn.close();
        valja.println("<br />");
        valja.println("<a href='LauludPlaadil3'>Laulude paiknemine plaatidel</a>");
        valja.println("</body>");
        valja.println("</html>");

      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```

Siin näites on muutmisvormi ülesandeks näidata etteantud id-numbriga laulu andmed tekstiväljadesse, et inimesel oleks võimalik väärtusi parandada. Edasine tegelik muutmistöö baasis usaldatakse järgmisele servletile nimega LauluMuutusLeht. Lisaks pealkirjale ja esitajale tuleb LauluMuutusLehele edasi anda ka muudetava pala ID – muidu poleks ju teada, millise pala väärtused vahetada tuleb. Identifikaatori kaasa andmiseks sobib varjatud väli, sisestuselement parameetriga hidden.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.sql.*;
public class LauluMuutusVorm extends HttpServlet {
    public void doGet(HttpServletRequest kysimus,
                      HttpServletResponse vastus)
    throws IOException, ServletException{
      vastus.setContentType("text/html");
      PrintWriter valja = vastus.getWriter();
      valja.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML "+
          "1.0 Transitional//EN\" "+
          "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">");
      valja.println("<html><head>");
      valja.println("<meta http-equiv=\"Content-Type\" "+
                          "content=\"text/html; charset=ISO-8859-1\" />");
      valja.println("<title>Laulu andmete muutmine</title></head>\n");
      valja.println("<body><h1>Laulu andmete muutmine</h1>");
      try{
        String id=kysimus.getParameter("id");
        if(id==null){
          valja.println("Laul määramata</body></html>");
          return;
        }
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement st=cn.prepareStatement(
          "SELECT pealkiri, esitaja FROM laulud WHERE id=?");
          st.setInt(1, Integer.parseInt(id));
        ResultSet rs=st.executeQuery();
        if(!rs.next()){
          valja.println("Soovitud laul puudub.</body></html>");
          return;
        }
        valja.println("<form action='LauluMuutusLeht' method='post'>\n"+
          "<table><tr><td>Pealkiri:"+
          "</td><td><input type='text' name='pealkiri' value='"+
          rs.getString("pealkiri")+"' /></td></tr>"+
          "<tr><td>Esitaja:"+
          "</td><td><input type='text' name='esitaja' value='"+
          rs.getString("esitaja")+"' /></td></tr>"+
          "</table><input type='submit' value='Muuda' />"+
          "<input type='hidden' name='id' value='"+id+"'"+
          "</form>"
        );
        cn.close();
      }catch(Exception viga){
         viga.printStackTrace(valja);
      }
    }
}
```

Ning ongi ees leht, kus võimalik ennist sisse kirjutatud väärtused uutega asendada.



Muutmislehele jõuavad nii pealkiri, esitaja kui laulu identifikaator. Lehel paikneva UPDATE-lause ülesandeks on etteantud id-ga laulu parameetrite väärtused asendada. Turvalisuse mõttes ikka PreparedStatementi kasutades.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
```

```
public class LauluMuutusLeht extends HttpServlet {
    public void doPost(HttpServletRequest kysimus,
                       HttpServletResponse vastus)
    throws IOException, ServletException
    {
      try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection cn=DriverManager.getConnection(
           "jdbc:odbc:esimene", "", "");
        PreparedStatement ps=cn.prepareStatement(
          "UPDATE laulud SET esitaja=?, pealkiri=? WHERE id=?"
        );
        ps.setString(1, kysimus.getParameter("esitaja"));
        ps.setString(2, kysimus.getParameter("pealkiri"));
        ps.setInt(3, Integer.parseInt(kysimus.getParameter("id")));
        ps.executeUpdate();
        cn.close();
        vastus.sendRedirect("Laulud6");
      }catch(Exception viga){
         viga.printStackTrace(vastus.getWriter());
      }
   }
}
```

Kui andmed muudetud ning loetellu tagasi vaatama minna, siis ka seal on Kungla rahva teine sõna ilusti väikese r-iga. Ning et andmed võetakse ikka samast tabelist, siis muutus paistab ka lehel, kus kirjas laulude paiknemine plaatidel.

Ülesandeid

Söökla menüü

* Loo lehestik sööklas pakutavate toitude lisamiseks ja vaatamiseks. Igal toidul on nimi ja hind.
* Sisestatud toidu andmeid on võimalik ka muuta ning olemasolevaid toite kustutada.
* Loo vahend, mille abil saaks määrata, milliseid toite millisel päeval pakutakse.

Bussiplaan

* Lehele on võimalik salvestada ja sealt vaadata busside väljumise aegu.
* Buss sõidab sihtkohta 2 tundi ja 30 minutit. Vastavalt väljumisaegadele arvutatakse ja sihtkohta jõudmise ajad.
* Lisaks eelmisele on olemasolevaid aegu võimalik kustutada või muuta.

Kirjasõprade otsing sünniaasta järgi

* Inimese kohta saab veebilehtl salvestada nime, sünniaasta ning sünniaastate vahemiku, millises vanuses kirjasõpra ta otsib.
* Iga inimese lehel kuvatakse sisestatutest nende inimeste andmed, kelle sünniaasta jääb soovitud vahemikku.
* Lisaks kuvatakse inimesele need, kelle soovitud vahemikku vaataja ise jääb ning eraldi loeteluna nimed, kelle puhul mõlemad vahemikud sobivad.

Kirjasõprade otsing huviala järgi.

* Loo administraatorile võimalus huvialade sisestamiseks ja vaatamiseks.
* Lisa võimalus isikute andmete sisestamiseks. Iga isik saab valida, millised huvialad talle loetelust sobivad.
* Näita inimese lehel loetelu teistest inimestest, kellega tal vähemasti üks huviala kattub. Vajutades vastava inimese nimele näidatakse loetelu kattuvatest huvialadest.

Autovaruosade otsing

* Loo administraatorile leht automarkide sisestamiseks ning leht osade nimetuste sisestuseks.
* Loo leht, kus saab valida margi ja osa. Samuti, kas soovitakse müüa või osta ning lisada kommentaari (näit. väljalaskeaasta).
* Loo leht, kus müüjale näidatakse kõik vastava osa ostusoovid ning ostjale müügisoovid.

Telefoninumbrite märkmik

* Kasutaja saab veebi kaudu talletada ja vaadata inimeste eesnimesid, perekonnanimesid ja telefoninumbreid.
* Andmeid on võimalik parandada ning perekonnanime järgi otsida.

* Andmeid saab ka kustutada ning rippmenüüst iga numbri juurde valida, kas tegemist on kodu- või töötelefoniga.

Veahaldusvahend

* Loo leht veateadete salvestamiseks ja vaatamiseks.
* Lisa administraatorileht, kus on võimalik olemasolevate parandajate hulgast määrata, kes konkreetse teatega tegelema hakkab.
* Parandaja näeb ainult enesele määratud veateateid. Saab teate juurde märkida, millised probleemid on lahendatud.

Korrapidajate tabel

* Lehel on võimalik sisestada kuupäev ning nimi, kes sel päeval korrapidaja. Tulemusi saab ka näha.
* Võrreldes eelmisega kontrollitakse, et samaks päevaks ei määrataks korraga mitut korrapidajat.
* Salvestatud väärtusi saab muuta ja kustutada vaid vastava rolliga kasutaja.

Linnuvaatlusmärkmik

* Lehel saab salvestada nähtud linnuliigi, vaatlusaja ja isendite arvu. Tuemusi õnnestub vaadata.
* Eraldi vahend on olemasolevate linnuliikide sisestamiseks. Vaatlusandmete sisestamisel valitakse linnuliik rippmenüüst.
* Liikide loendi kõrval näidatakse, mitu korda vastavat liiki on loendatud ning mitut isendit vastavast liigist kokku nähtud.

Taimevaatlusmärkmik

* Lehel sisestatakse vaatleja nimi, taime liik ja vaatluskoht. Tulemusi õnnestub vaadata.
* Vajutades vaatleja nimele, näidatakse kõik andmed, mis see inimene on kirja pannud.
* Lisaks eelmisele näidatakse kõik erinevad vaatlejanimed, nime taga number, mitu erinevat liiki taimi on see inimene kirja pannud (sõltumata vaatluskohast).

Putukate kogu

* Lehel sisestatakse kogutud putuka liik, vaatluspäev/aeg, piirkond, koht, leidja ning määraja nimi. Andmeid õnnestub vaadata.
* Loetelus näidatatakse vaid putukaid, kes on kogutud enne kasutaja sisestatud kellaaega.
* Lisaks eelmisele asub iga rea ees märkeruut. Märgistatud putukad näidatakse eraldi lehele, kus õnnestub igale neist etikett trükkida. Etiketis on andmed tabelina, iga väärtuse ees tunnuse nimetus.

Loodusfotograafi märkmik

* Iga pildistuse kohta talletatakse pildi teema, kaugus, säriaeg ja ava suurus. Andmeid näeb lehel.
* Säriajad valitakse rippmenüüst. Õnnestub otsida kasutaja määratud säriajaga pildistuste andmeid.

* Lisaks eelmisele luuakse statistika, millise säriaja ja ava suuruse kombinatsiooniga kui mitu pilti on tehtud. Igale pildile saab lisada kommentaare.

Ilmavaatlusandmed

* Igal vaatluskorral sisestatakse temperatuur, tuule kiirus (m/s), suund (nt. NNO). Andmeid näidatakse tabelina.
* Eraldi lehele luuakse statistika, mitu korda millist temperatuuri olnud on.
* Lisaks luuakse tabel, mitu korda millisest suunast tuul puhunud on. Mida tugevam on vastavast suunast puhunud tuule keskmine kiirus, seda tumedamalt vastav arv joonistatakse.

Videokahuri kasutusgraafik

Programmi eesmärgiks on pakkuda veebipõhiselt võimalust jälgida ja reserveerida videokahuri kasutusvõimalust.

* Tabelis on kirjas kahuri kasutusaegade algused ja kestused. Tabel kuvatakse.
* Lisaks eelmisele võib kasutaja sisestada aja. Väljastatakse, kas sel ajal on kahur vaba.
* Lisaks eelmisele on registreeritud kasutajatel võimalik kahurit reserveerida.

Raamatukogulaenutus

* Tabelis on raamatute ilmumisandmed. Tabel kuvatakse veebilehele.
* Lisaks eelmisele on loodud kasutajate tabel ning seosetabel, kus on näidatud, milliseks ajaks on raamat kasutajale laenatud. Väljastatakse, kas praegu on soovitud raamat vaba.
* Lisaks eelmisele on võimalik lisada nii raamatuid, kasutajaid kui laenutusi ning otsida sobivat parajasti vabade raamatute hulgast.

Tööaja arvestusgraafik

* Tööle sisenedes ning töölt lahkudes sisestab kasutaja oma koodi. Kood, liikumissuund ning kellaaeg talletatakse tabelisse.
* Lisaks eelmisele on võimalik kontrollida, et tabelis poleks järjest kaht sisenemist või väljumist. Kui andmed korras, väljastatakse inimese tööl viibitud aeg.
* Inimese tööl oldud aeg väljastatakse kuude lõikes. Eraldi näidatakse puhkepäevadel tööl viibitud aeg. Iga kuu kohta leitakse aeg, kus korraga on olnud kõige rohkem inimesi kohal.

Komandeeringuaruanded

* Kasutaja saaab veebi kaudu tabelisse märkida oma nime, reisi sihtpunkti ja reisi kestuse
* Lisaks saab kasutaja märkida, kui suured on tema sõidukulud. Päringulehel on võimalik vaadata nii kogu suurt tabelit kui inimese kaupa, kui palju on ta kokku kulutanud komandeeringusõitudele.
* Eraldi tabelid on nii kasutajate, komandeeringute kui üksikute sõidutsekkide tarvis. Veebilehelt on võimalik lisada komandeeringuid ning sinna kuuluvaid pileteid.

Uksed

Abiprogramm kaardipõhisele ustesüsteemile.
* Tabelites on kirjas väljaantud magnetkaardid, majas paiknevad kaardiga avanevad uksed ning õigused, millise kaardi omanik millist ust avada tohib.
* Sisestades kaardi numbri ja ukse numbri, vastatakse, kas vastava kaardi omanik tohib sellest uksest siseneda.
* Lisaks eelmisele saab sissepääsu õigusi jagada piiratud ajaks. Vale kaardi või ukse numbri sisestamisel antakse veateade.

Laulude andmebaas

* Võimalda tabelisse lisada laulu sõnad, pealkiri ja märksõnad.
* Lisaks eelmisele on võimalik nii sisu, pealkirja kui märksõnade järgi otsida.
* Lisaks eelmisele on registreeritud kasutajatel võimalik laulu andmeid täiendada ja parandada.

Vilistlaste kontaktandmed

* Tabel, milles on vilistlaste nimed, aadressid, telefoninumbrid ja elektronposti aadressid, kuvatakse veebilehele.
* Inimestel on võimalus oma andmeid muuta. Logisse jäetakse kirja, millised olid andmed enne, millisest masinast ning milliseks andmed muudeti.
* Administraatoril on võimalik vaadata valitud kirjega seotud logisid ning määrata, millise sammuni olukord ennistatakse.

Tunniplaan

* Tabelis on kirjas aine nimetus, grupi tähis, ruumi nr, algusaeg ning kestus. Andmed kuvatakse veebilehele. Soovi korral saab vaadata vaid ühe grupi aineid.
* Programm kontrollib andmete korrektsust: hoiatus antakse, kui samal grupil on korraga mitu ainet, samuti kui samas ruumis on korraga mitu ainet.
* Kui sisestatakse grupi tähis ja ruumi nr, siis väljastatakse võimalikud ajavahemikud, mil tund võiks toimuda. Seejärel saab sisestada aine nime, valida aja ja pikkuse ning andmed talletatakse tabelisse.

Jõe vooluhulgad

Igal keskpäeval saadetakse Pikasilla mõõtmispunktist kesktabelisse teade, mitu liitrit sekundis voolab vett Väikesest Emajõest Võrtsjärve.
* Tabel väljastatakse veebilehele, näidates, vooluhulgad nii liitrites sekundi kohta kui kuupmeetrites ööpäeva kohta.
* Lisaks eelmisele näidatakse väärtused lehele tulpdiagrammina.
* Tubad koostatakse viie päeva keskmise kaupa. Kasvava vooluhulgaga piirkondades on tulbad teise värviga eristatud.

Ülesannete kogu

* Veebilehe kaudu salvestatakse tabelisse tulpadena nii ülesanded kui nende pealkirjad. Andmeid on väljastamisel võimalik sortida nii pealkirja kui sisu järgi.
* Eraldi tabeli(te)s on võimalikud märksõnad nii teemade kui raskusastmete kohta. Kasutaja saab valida, millised märksõnad konkreetse ülesande juurde kuuluvad. On võimalik otsida nii pealkirja kui märksõna järgi.
* Kord sisestatud andmeid ja seoseid on võimalik muuta. Samuti saab lisada märksõnu.

Koodinäidete kogu

* Koodinäidete tekstid ning pealkirjad salvestatakse tabelisse. Vaatamisel saab valida, kas näide väljastatakse veebilehele lihttekstina või HTML-kodeeritult.
* Igal näitel on pealkiri ning valikuline selgitustekst. Eraldi saab näite juures määrata, milliste teemade juurde ta kuulub. Samuti saab määrata, millised näited on loodud näitega lähedased. Otsida on võimalik nii teemade, pealkirja kui näites või selgituses leiduva sõna järgi.
* Lisaks eelmisele on teemad puukujulises hierarhias. Näidatakse valitud teemast alanev teemade ning näidete puu.

Failipuu

* Veebis näidatakse välja programmi sees määratud kataloogis paiknevate failinimede ja alamkataloogide loetelu.
* Vajutades failinimele, on võimalik näha selle sisu. Vajutades kataloginimele, näidatakse vastav kataloog oma failide ja alamkataloogide nimedega.
* Andmed on näha puuna, iga kataloogi ees + või - vastavalt sellele, kas sisu näidatakse. Plussile vajutades kataloog avaneb, miinusele vajutades sulgub.

Baasipõhine failipuu

* Andmetabeli kirjeteks on failid (nimed) ning kataloogid. Veergudes paiknevate arvude abil on võimalik tuvastada, kuhu kataloogi miski fail või kataloog kuulub. Trüki puu alates juurkataloogist välja.
* Koosta programm lisamaks kasutaja määratud fail või kataloog (koos alanejatega) määratud kohta andmetabelipuusse.
* Võimalda talletada ka failide sisu baasi, liikuda andmepuus harusid avades ja sulgedes ning näha valitud faili sisu.

Restorani ladu

* Tabelis on kirjas laos leiduvad toiduained, sees olevad kogused ning iga aine kohta alampiir, mis puhul tuleks seda juurde tellida. Laohoidja sisenemisel näidatakse talle loetelu nimetustest, mida oleks vaja tellida.
* Lisaks eelmisele, kui töötaja toob laost kaupa, valib ja märgib ta arvutisse, mida ja kui palju võttis. Töötaja saab jätta laohoidjale soovi, millist toodet tellida. Soovid kaovad laohoidja ees olevast nimekirjast, kui vastav aine saabub lattu või kui laohoidja märgib, et seda pole võimalik muretseda.
* Lisaks eelmisele liiguvad soovid laohoidjale viimase postkasti. Statistikas on võimalik vaadata, milline töötaja milliseid tooteid ja millises koguses võtnud on.

Tähed muusikas

* Laulust on arvutil meeles kuus järjestikust sõna. Väljastatakse nii mitmes sõna, kui suure arvu kasutaja kirjutab. Sobimatu numbri puhul palutakse uuesti valida.
* Lisaks eelmisele jäävad valitud sõnad ekraanile näha, juba avatud sõna valides antakse veateade.
* Lisaks eelmisele on arvutil meeles paljude laulude kuus järjestikust sõna. Administraator saab valida, millised neist tulevad sel korral võistlusse. Vajutades nupule "järgmine", võetakse ette administraatori valitutest juhuslikult uus lugu kuni lugusid jagub.

Kuldvillak

* Arvutil on meeles 25 küsimust jaotatuna võrdselt viie teema vahel. Lehel on näha 5*5 tabel, kus igale küsimusele vastab viide. Vajutades viitele, kuvatakse lehel vastava küsimuse tekst.
* Lisaks eelmisele ei ole võimalik sama küsimust rohkem kui korra valida. Pärast küsimuse aktiivseks muutmist administraatori poolt on võimalik kolmel mängijal see enesele vastamiseks küsida/vajutada. Kärmemale antakse võimalus tekstiväljas vastata, ülejäänutele teatatakse hilinemisest.
* Lisaks eelmisele loetakse iga mängija punkte.

Eurovisiooni hääletus

* Tabelis on kirjas osalevate maade ja laulude nimekiri. Andmed näidatakse veebilehel.
* Iga maa esindaja saab täita tabeli, kus määrab, mitu punkti ta igale teisele andis. Pärast täitmist näidatakse, mitu punkti millisel lool on ning mitmendal kohal sellega ollakse.
* Lisaks eelmisele kontrollitakse, et ükski maa ei saaks hääletada rohkem kui korra. Samuti, et punkte antaks korrektselt, st., et 1. koht 12 punkti, teine 10, kolmas 8, neljas 7 ning edasi
ühe punkti kaupa vähenedes allapoole.

Miljonimäng

* Pärast administraatori vajutust algküsimuse väljakuulutamiseks on võistlejatel võimalus enesele laadida küsimus ja järjestatavad vastusevariandid. Enneaegse päringu peale teatatakse "vara veel".
* Vastuste salvestamisel jäetakse meelde kulunud aeg ning kas vastus on õige. Tulemused väljastatakse veebilehel. Eraldi tuuakse välja kõige kiirem õige vastus.
* Lisaks eelmisele loo vahend põhimängu kajastamiseks. Küsimused ja õiged vastused loetakse andmekandjalt. Peetakse arvestust kroonide ning kasutatud õlekõrte üle. Publik saab pakkuda õiget vastust. 50-50 eemaldab juhuslikult kaks vale vastust.

Kahevõitlus

* Ekraanil on küsimus ning kümme vastusevarianti. Antakse teada, kas valitud variant oli vale või õige.
* Peetakse meeles ja näidatakse, kumma mängija vastamiskord on. Loetakse punkte, vale vastuse korral kukuvad vastava mängija punktid nulli, samuti läheb järg automaatselt üle teisele mängijale.
* Mäng realiseeritakse täielikult: küsimused loetakse sisse, neljas voor kolmekordsete punktidega, lõpus võitjal eraldimäng.

7 vaprat

* Tabelis on kirjas loo pealkirjad, esitajad ning eelmisel korral antud punktid. Andmed kuvatakse veebilehele punktide arvu järjekorras, 2 uut lugu kõige viimasena.
* Inimestel on võimalik valida lugu ning seda hinnata. Lehte vaadates on näha iga loo hääletamiste arv ning keskmine hinne.
* Lisaks eelmisele tehakse analüüs, milliselt IP-lt on millist lugu mitmel korral hinnatud. Kui arv ületab administraatori seatud limiiti, siis sealt tulnud hääli vastava loo kohta lõpptabelis ei arvestata.

Kuldlaul

* Tabelis on kirjas 9 eelmisest saatest jäänud ning 3 uut lugu. Iga loo kohta peetakse meeles, mitmendat korda too tabelis. Tulemused väljastatakse veebilehele.
* Külastajad saavad administraatori määratud ajavahemikus lauludele hääli anda. Hääletuse lõpus näidetakse, palju hääli iga laul saanud, samuti järjestatakse laulud saadud häälte arvu järjekorras.
* Järgmisse saatesse jõuavad 9 enim hääli saanud lugu. Kui mõni laul on suutnud tabelis püsida 12 korda, siis see arvatakse kuldlauluks ning ka tema asemele tuleb administraatoril järgmise korra tarbeks sisestada uus lugu. Aasta lõpus väljastatakse sel aastal tekkinud kuldlaulude loetelu.

Autoregister

* Tabelis on kirjas registris olevate autode number, mark ning väljalaskeaasta. Andmed kuvatakse veebilehele.
* Markide loetelu on omaette tabelis. Samuti on loodud isikute tabel ning seosetabel, milline isik millise autoga seotud on. Väljastatakse sisestatud isikukoodiga seotud autode andmete loetelu.
* Lisaks eelmisele on võimalik veebi kaudu nii isikute kui autode andmeid lisada ja muuta. Samuti on eraldi leht müügitehingu tarbeks, kus auto läheb ühe isiku juurest teise valdusesse.

Õppetooli raamatukogu

Rakenduse eesmärgiks on anda ülevaade õppetoolis kasutatavatest ja laenutatavatest materjalidest.
* Iga inimene saab veebi kaudu sisestada oma riiulil leiduvate raamatute pealkirjad. Andmed kuvatakse veebilehele.
* Inimeste andmete jaoks on eraldi tabel. Üks rida selles tabelis tähendab, et raamat on õppetooli oma, muul juhul on tegemist inimese isikliku raamatuga. Ühe seose järgi on võimalik näha, kelle oma raamat on, teise seose abil õnnestub vaadata, kas ja kellele on raamat laenutatud. Lisamisel on võimalik rippmenüüst valida, kelle oma on raamat.
* Lisaks eelmisele on võimalik laenutus veebi kaudu registreerida. Samuti saab end raamatule järjekorda panna. Peetakse logi, mitu korda ja millal on millist raamatut laenutatud.

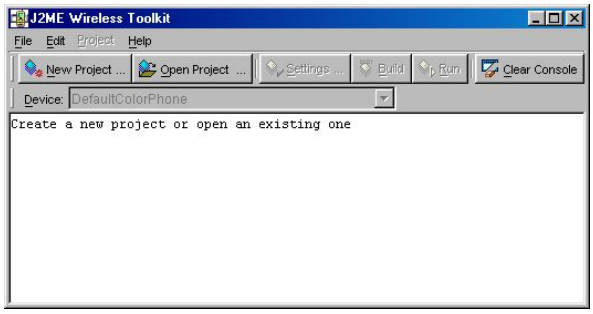
J2ME

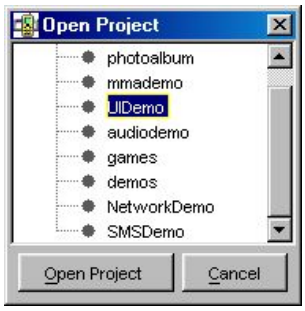
Mobiilirakendused, graafika, salvestamine, võrguühendus

Miniseadmete programmide koostamiseks pakub SUN J2ME Wireless Toolkiti nimelist keskkonda aitamaks koodi kompileerida, kontrollida ja käivitada. Samad toimingud õnnestub teha ka käsurealt, kuid vähemasti alustada tundub mugavam olema abikeskkonnast, KToolbari nimelisest vahendist.

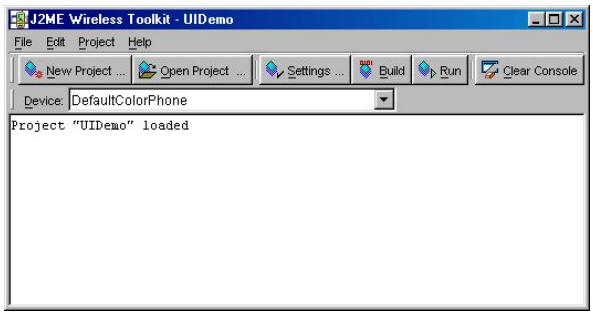
Demonstratsiooniprogrammid

Võimalustega tutvumiseks sobivad kaasatulevad demoprogrammid. Vajutades nupule "Open Project", võib valida kümnekonna väikese valmisrakenduse seast.





Kui projekt valitud, muutuvad töötavaks ka ülejäänud nupud. Nagu aimata võib, Build kompileerib ja teeb muud vajalikud ettevalmistustööd, Run käivitab.

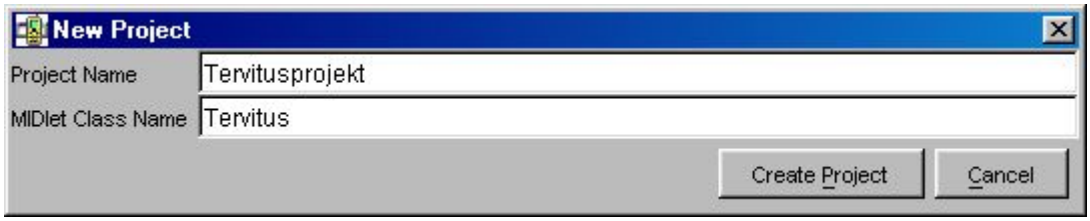


Seadme saab valida paari erikujulise emulaatori hulgast, et õnnestuks kontrollida rakenduse töövõimet mitmesuguse ekraaniresolutsiooni ning nuppude asetuse korral. Konkreetsete telefonimudelite emulaatoreid võib juurde laadida tootjate veebilehtedelt.

Valitud UIDemo tutvustab kasutajaliidese kujundamiseks tarvilikke elemente.

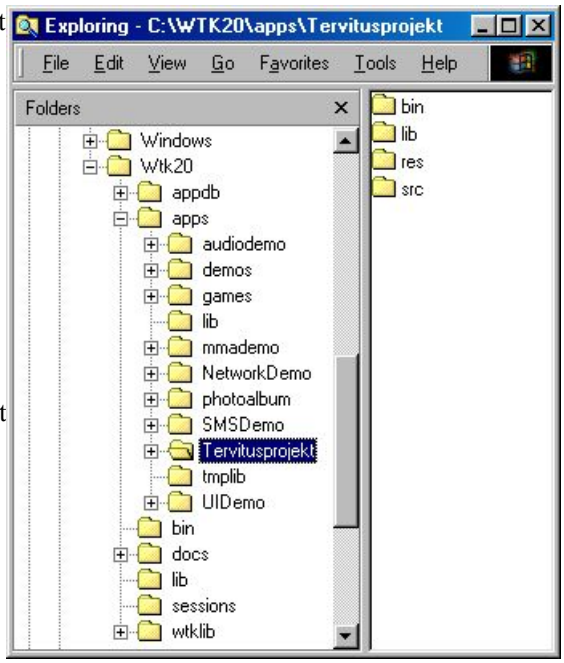
Omaloodud projekt

Ka demokoode uurides, muutes ja tulemust piiludes võib enesele küllalt meelepärase tulemuse saada. Mingil hetkel aga tekib tahtmine või vajadus "päris omi" programme kokku panema hakata. Selleks tasub luua uus projekt.



Projektile nimi ning kõrvale käivitatava MIDlet-i klassi nimi, millest programmi töö peale algab.

Kui failisüsteemi piiluda, siis võib näha, et iga projekti tavis on loodud \Wtk20\apps kataloogi alla omaette kataloog. Nii ka nüüd tehtud tervitusprojekti tarbeks. Ning projekti kataloogi sisse luuakse kohe hulk alamkatalooge, millest esialgu meile tarvilik on src oma programmikoodide paigutamiseks. Nii teatab ka arenduskeskkond:



Place Java source files in "C:\WTK20\apps\Tervitusprojekt\src"

Ülejäänud WTK20 all olevatest kataloogidest on esiotsa tarvilikum docs. Selle seest leiab nii J2ME API täieliku kirjelduse kui mõningaid näiteid ja seletusi.

Tervitav programm

Esimene programm nagu ikka – võimalikult lihtne ja lühike. Lihtne tervitus võtab siin küll mõnevõrra enam koodiridu kui käsurealt käivitatavas programmis, kuid mitte midagi hirmsat. Mobiilseadmes käivitatav programm peab olema kirjutatud javax.microedition.midlet.MIDlet-i alamklassina sarnaselt nagu näiteks veebilehel töötav rakend vajab enesele ülemklassiks java.applet.Applet-i. Kohustuslikeks ülekaetavateks meetoditeks on startApp, pauseApp ja destroyApp. Nagu nimedki ütlevad, esimene käivitatakse rakenduse töö alustamisel, teine töö ajutisel katkemisel (näiteks sissetuleva kõne korral) ning kolmas töö lõppemisel. Siin näites on nad aga kõik tühjaks jäetud ning lihtsalt klassi konstruktoris luuakse teade ja paigutatakse see rakenduse ekraanile. Display.getDisplay(this) annab juurdepääsu rakenduse ekraanile ning käsklus setCurrent määrab, milllise objektiga ekraan täidetakse.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Tervitus extends MIDlet{
```

```
  public Tervitus(){
    Alert a=new Alert("Tervitamine", "Tere", null,
           AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(a);
  }
  protected void startApp() throws MIDletStateChangeException{}
  protected void pauseApp(){}
  protected void destroyApp( boolean p1 )
                     throws MIDletStateChangeException{}
}
```

Pärast kompileerimist käivitades võib rakenduse tööd imetleda. Esmalt tuleb ette projektis määratud MIDlet-i nimi ning selle valides avaneb rakenduse sisu, milleks siin vaid ekraanile näidatud tervitus.

Nagu näha toimib sama rakendus loetavana ka suurema seadme peal.

Kalkulaator

... ehk lihtsaim rakendus, mille töö tulemustega ka juba midagi peale hakata on. Nagu ikka, tuleb valmis ehitada nii kujundus kui arvutusloogika. Ning nagu lihtsamate programmide puhul ikka, kulub rohkem koodiridu väljanägemise sättimise tarvis. Tekstiväli nii kummagi liidetava sisestamiseks kui vastuse vaatamiseks. Tüübiks NUMERIC, mis – nagu nimestki näha – lubab välja sisse vaid numbreid kirjutada. Soovides J2SE vahenditega taolist tekstivälja luua tuleks mõnevõrra rohkem nikerdada. Vorm üksikute tükkide koos hoidmiseks. Üks käsklusnupp arvutamise ning teine väljumise tarbeks. Ning vormi küljes käsklus setCommandListener(this), mis tähendab, et vormis loodud käsklused antakse töötlemiseks this-ile ehk jooksvale kalkulaatori eksemplarile.Võrrelduna J2SE-ga on J2ME-s ressursside kokkuhoiu mõttes tavaks teated saata töötlemiseks korraga vaid ühte kohta. Nii ongi addActionListeneri asemel siin kirjas setCommandListener.

Sündmuste püüdmiseks realiseerib Kalkulaatori klass liidest CommandListener, mis annab kohustuse luua meetod CommandAction. Esimese parameetri järgi saab otsustada, milline käsklus saabus, teine näitab aktiivset ekraani, millel käsklusele vajutati. Nagu käskluste nimedestki näha, ühe ülesandeks on paluda tulemus arvutada, teise omaks rakenduse töö lõpetada. Et tekstivälja väärtust küsitakse sõnena, liitmise tulemust arvutatakse aga arvuna, siis tuleb ette võtta mõlemasuunaline tüübimuundus, mis teebki käsu pikaks. J2ME arvutuskäsud kasutavad vaid täisarve, kuna vastavate seadmete protsessorid ei pruugi suuta osata muude arvudega otse tehteid teha ning "ümber nurga" arvutamine võtab jälle märgatavalt ressursse.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Kalkulaator extends MIDlet implements CommandListener{
  TextField tf1=new TextField("Esimene arv", "", 5,
                      TextField.NUMERIC);
  TextField tf2=new TextField("Teine arv", "", 5, TextField.NUMERIC);
  TextField tf3=new TextField("Vastus", "", 5, TextField.NUMERIC);
  Command c1=new Command("Arvuta", Command.SCREEN, 1);
  Command c2=new Command("Välju", Command.EXIT, 1);
  Form f=new Form("Arvutaja");
  public Kalkulaator(){
    f.append(tf1);
    f.append(tf2);
    f.append(tf3);
    f.addCommand(c1);
    f.addCommand(c2);
    f.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
  }
  protected void startApp( ) throws
                  MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 )
   throws MIDletStateChangeException{}

  public void commandAction(
            Command p1, Displayable p2 ){
     if(p1==c1){
       tf3.setString(String.valueOf(
           Integer.parseInt(tf1.getString())+
           Integer.parseInt(tf2.getString()))
       );
     }
     if(p1==c2){
       notifyDestroyed();
     }
  }
}
```

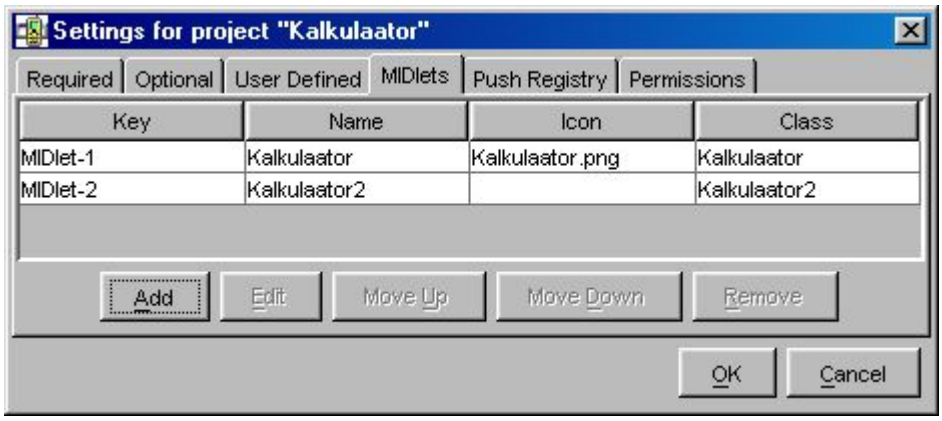
Tehtevalikuga kalkulaator

Eelmise rakenduse täiendus. Liitmise asemel võib kasutada nelja põhitehet. Ühel vormil endiselt tekstiväljad andmete sisestuseks ning koht vastuse tarvis. Sedakorda vastuse näitamiseks StringItem, mille sisu saab küll programmi poolt määrata, kuid mitte kasutaja ise kirjutada. Ka parasjagu aktiivne tehe näidatakse StringItem-i abil. Tehtevalikuks kasutatakse ekraanisuurust komponenti List, mis siis vastava käsu peale välja kutsutakse. Kui tehe valitud, asendatakse ekraaninäit jälle endise vormiga ning valitud element paigutatakse vormi sisse tehte kohale.

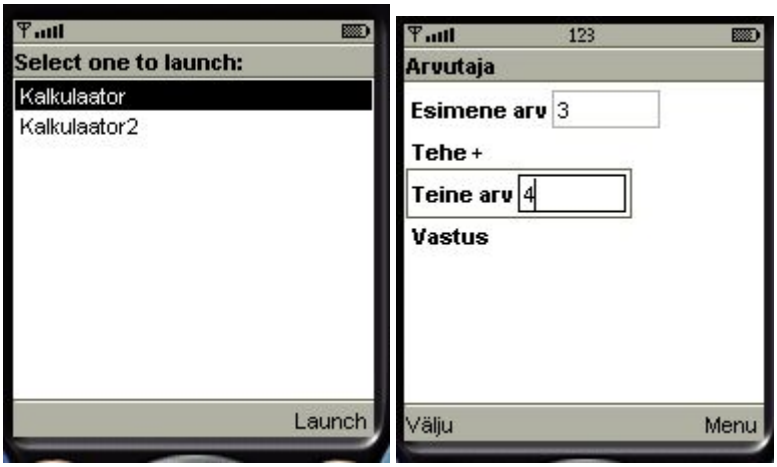
```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class Kalkulaator2 extends MIDlet implements CommandListener{
  TextField tf1=new TextField("Esimene arv", "", 5, TextField.NUMERIC);
  TextField tf2=new TextField("Teine arv", "", 5, TextField.NUMERIC);
  StringItem vastus=new StringItem("Vastus", "");
  String[] tehted={"+", "-", "*", "/"};
  List nimistu=new List("Tehte valik", List.EXCLUSIVE, tehted, null);
  StringItem silt=new StringItem("Tehe", "+");
  Command c1=new Command("Arvuta", Command.SCREEN, 1);
  Command c2=new Command("Välju", Command.EXIT, 1);
  Command c3=new Command("Vali tehe", Command.SCREEN, 1);
  Command c4=new Command("Tehe valitud", Command.SCREEN, 1);
  Form f=new Form("Arvutaja");
  public Kalkulaator2(){
    f.append(tf1);
    f.append(silt);
    f.append(tf2);
    f.append(vastus);
    f.addCommand(c1);
    f.addCommand(c2);
    f.addCommand(c3);
    nimistu.addCommand(c4);
    f.setCommandListener(this);
    nimistu.setCommandListener(this);
    Display.getDisplay(this).setCurrent(f);
  }
  protected void startApp( ) throws MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 ) throws MIDletStateChangeException{      }

  void arvuta(){
    int arv1=Integer.parseInt(tf1.getString());
    int arv2=Integer.parseInt(tf2.getString());
    String tulemus="";
    if(silt.getText().equals("+")){tulemus=String.valueOf(arv1+arv2);}
    if(silt.getText().equals("-")){tulemus=String.valueOf(arv1-arv2);}
    if(silt.getText().equals("*")){tulemus=String.valueOf(arv1*arv2);}
    if(silt.getText().equals("/")){tulemus=String.valueOf(arv1/arv2);}
    vastus.setText(tulemus);
  }

  public void commandAction( Command p1, Displayable p2 ){
    if (p1==c1){
      arvuta();
    }
    if(p1==c2){
      notifyDestroyed();
    }
    if(p1==c3){
      Display.getDisplay(this).setCurrent(nimistu);
    }
    if(p1==c4){
      silt.setText(nimistu.getString(nimistu.getSelectedIndex()));
      Display.getDisplay(this).setCurrent(f);
    }
  }
}
```

Kui loodud klassi fail lihtsalt eelmise kõrvale kopeeritakse, siis fail küll kompileeritakse, kuid kusagilt ei leia kohta rakenduse uue versiooni käivitamiseks. Valides "Settings" ning sealt alt "MIDlets", saab vaadata, muuta ja lisada, millise nime all millises klassis olev kood käima tõmmatakse. Kõik rakenduses paiknevad klassid ei pea sugugi eraldi käivitatavad olema, mõnigi võib olla lihtsalt abiks rakenduse avaklassile. Ning samas kataloogis asuvaid abiklasse saab ka vajadusel kasutada mitme teise

klassi koodi sees. Tahtes aga loodud MIDleti teha rakenduse avamisel valitavaks, tuleb ta vastavasse loetellu lisada.

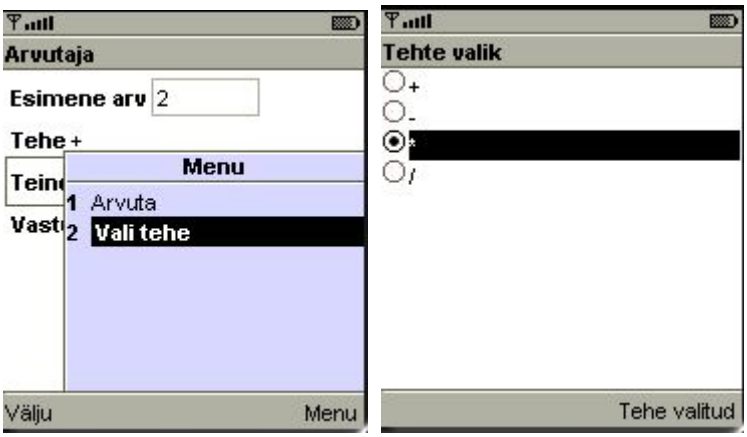


Nii võibki nüüd käivitamisel näha projekti sees kahte rakendust. Valides neist teise, tuleb ette tehtevalikuga kalkulaator.



Kui andmed sisestatud ning menüüst vajutada "arvuta", näeb tulemust.

Paludes aga tehe valida, ilmub eraldi List, mille hulgast siis tuleb sobiv enesele välja valida, kinnitades valikut käsuga "Tehe valitud".



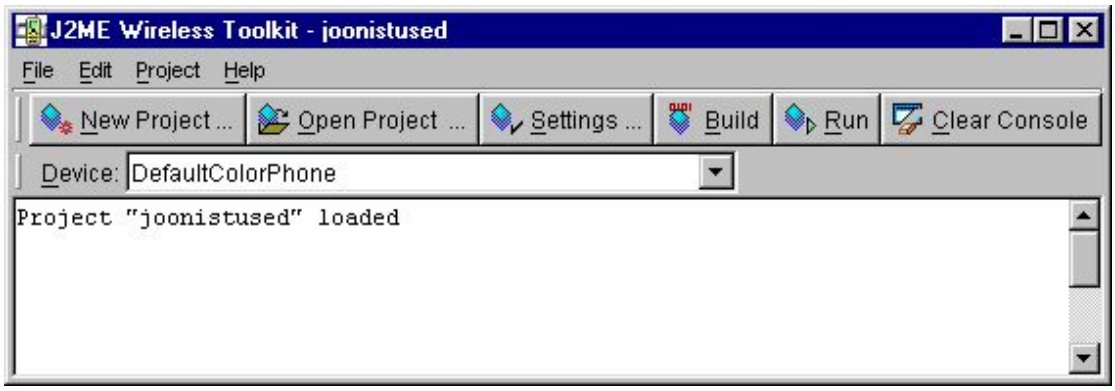
Siis pääseb tagasi avaekraanile ning võib vastava tehtega arvutustulemust vaadata.

Joonistused

Nagu enamike programmeerimisvahendite puhul, nii ka siin saab otse ekraanipunktidega arvutada ning sinna kujundeid joonistada. Nagu J2SE puhul, nii ka siin sobib joonistamise aluspinnaks Canvas ehk lõuend. Tavaline lõuend jätab enese alt välja nii tiitliriba kui käsuriba, kuid enamasti rakenduse ajal sellist lahendust vajataksegi. Kel tahtmist täisekraaniga tegelda, see vajab alates MIDP 2.0-st kättesaadavat GameCanvast. Joonistamiskäske antakse ikka meetodisse paint saabuva Graphicsi eksemplari kaudu. Kuid nagu mitmete muude klassidega, on ka osa siinsetest joonistusvahenditest samanimeliste J2SE klasside omadest mõnevõrra erinevad. Ning kui ka "hariliku" Java puhul on viisakas arvestada ekraani suuruse ja võimalustega, siis mobiilirakenduste puhul peab arvestama seadme parameetrite küllalt suure kõikumisega.

Üksik joon

Joonistuste tarbeks eraldi projekt ning koos sellega ka kataloog. Nii on sama teema rakendused ilusti ühes paigas koos.



Kui joonistuse oskused paigutada eraldi klassi, siis peaprogrammi saab jätta küllalt lühikeseks. Vaid rakenduse avamisel määratakse lõuendi eksemplar rakenduse paneelil nähtavaks.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Joonistus1 extends MIDlet{
  Canvas louend=new Louend1();
  protected void startApp() throws MIDletStateChangeException{
```

```
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
                throws MIDletStateChangeException{}
}
```

Lõuendi joonis ise kuvatakse ekraanile paint-meetodis, nii nagu tavarakendustegi puhul kombeks. Et vana pilt ekraanile paistma ei jääks, tuleb kõigepealt taustale ristkülik joonistada. Esimene joonis koosneb vaid ühest joonest, käsu parameetriteks nagu ikka kummagi otspunkti kaks koordinaati.

```
import javax.microedition.lcdui.*;
class Louend1 extends Canvas{
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawLine(10, 10, 80, 10); //x1, y1, x2, y2
  }
}
```

Ehkki rakendus koosneb kahest klassist, piisab, kui eraldi MIDletina on kirjas vaid Joonis1. Louend1 leitakse kataloogist üles.

Mitmekülgsem joonis

Nagu ennegi mainitud, käsud sarnanevad J2SE omadele, kuid ei pruugi kattuda. Puudub näiteks fillOval, tema ülesanded aga täidab fillArc, tuleb lihtsalt sobivad parameetrid ette anda. Punktiirjoon, mille tekitamiseks muidu oli vaja Graphics2D poole pöörduda lubatakse siin harilikus Graphics-klassis ekraanile paigutada. Teksti puhul võib määrata, kuhu jääb etteantud punkt teksti joonistamisel.

```
import javax.microedition.lcdui.*;

class Louend2 extends Canvas{
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.fillArc(10, 10, getWidth()-20, 50, 0, 180);
```

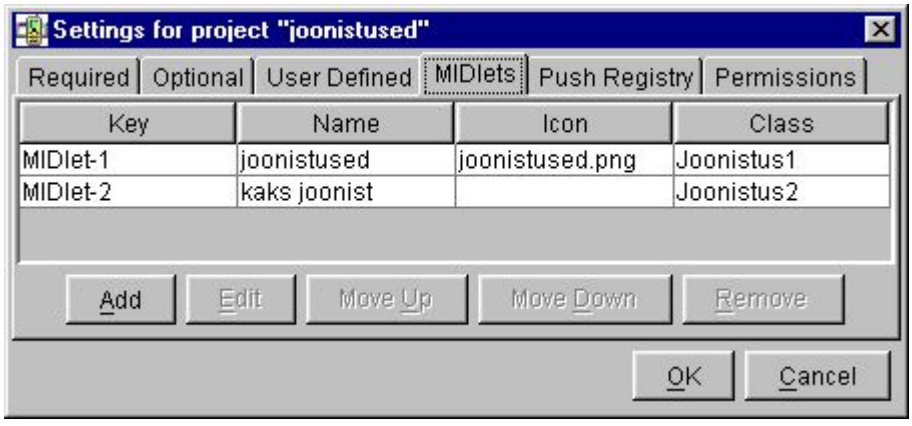
```
      //vasakult, ülalt, laius, kõrgus, algnurk, kaarenurk
     g.setStrokeStyle(Graphics.DOTTED);
     g.drawLine(10, 30, 10, getHeight());
     g.drawLine(getWidth()-10, 30, getWidth()-10, getHeight());
     g.drawString("Tervitus", getWidth()/2, getHeight()/2,
                              Graphics.BOTTOM | Graphics.HCENTER);
  }
}
```

Kaks joonist

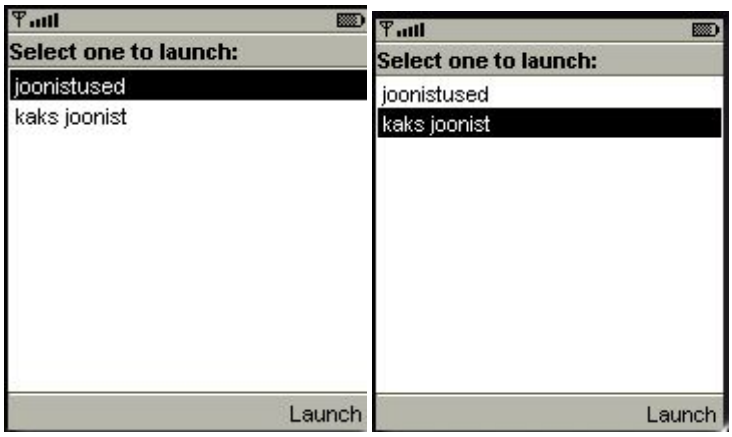
Peaprogrammis võimaldatakse valida, millist lõuendit vaadata. Kusjuures esimese lõuendi küljes on käsklus teise lõuendi vaatamiseks ning teisel jälle esimese juurde jõudmiseks. Käskude sisu nagu ikka kirjas commandAction'is. Ning algselt määratakse nähtavaks lõuend1.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Joonistus2 extends MIDlet implements CommandListener{
  Canvas louend1=new Louend1();
  Canvas louend2=new Louend2();
  Command c1=new Command("Esimene", Command.ITEM, 1);
  Command c2=new Command("Teine", Command.ITEM, 1);
  public Joonistus2(){
    louend1.addCommand(c2);
    louend2.addCommand(c1);
    louend1.setCommandListener(this);
    louend2.setCommandListener(this);
  }
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend1);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
  public void commandAction(Command c, Displayable d){
    if(c==c1){Display.getDisplay(this).setCurrent(louend1);}
    if(c==c2){Display.getDisplay(this).setCurrent(louend2);}
  }
}
```

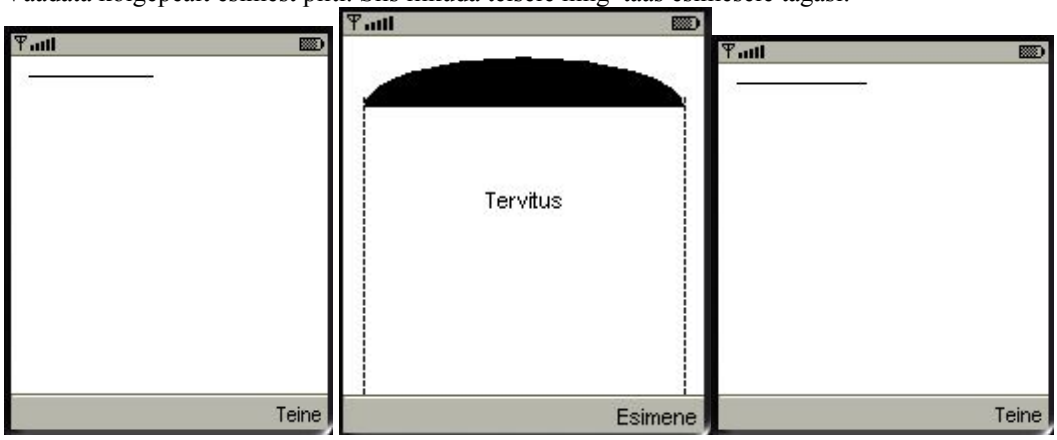
Taas vajab märkimist, et ja Joonistus2 võiks eraldi rakendusena tööle hakata.



Ning edasi võibki rakenduse valida ja käivitada.



Vaadata kõigepealt esimest pilti. Siis liikuda teisele ning taas esimesele tagasi.



Teine joonis arvestab ekraani mõõtudega ning on suurema ekraani peal ka tunduvalt suuremalt nähtav.

Liigutamine

Olgu tegemist andmebaasi graafilise väljundiga, mänguga või lihtsalt olukorra simulatsiooniga – ikka soovitakse ekraanipilti kasutaja tegevuse tulemusena muuta. Põhimõte sarnane kui Java tavarakenduste juures – andmemuutuse järel tuleb ekraanipilt uuesti joonistada. Klahvisündmuste kuulamiseks piisab üle katta vaid keyPressed – liikuvad teated peaksid automaatselt sinna alamprogrammi jõudma nii nagu näiteks Java esimese versiooni puhul. Et mitmesuguse ehitusega

masinatega hakkama saada, kasutatakse kaht väärtust. Meetodi parameetrisse jõudev arv vastab konkreetsele klahvile. Canvase käsklus getGameAction aga leiab koodile vastava loogilise tähenduse. Nõnda võib eri masinatel hoopis isesugune klahv või nende kombinatsioon tähistada noolt vasakule. Käsu getGameAction abil saab aga kontrollida, et kas konkreetse masina ja klahvi puhul sooviti anda vasakule liikumise käsklust. Meetod repaint nagu ikka palub pildi uuesti joonistada.

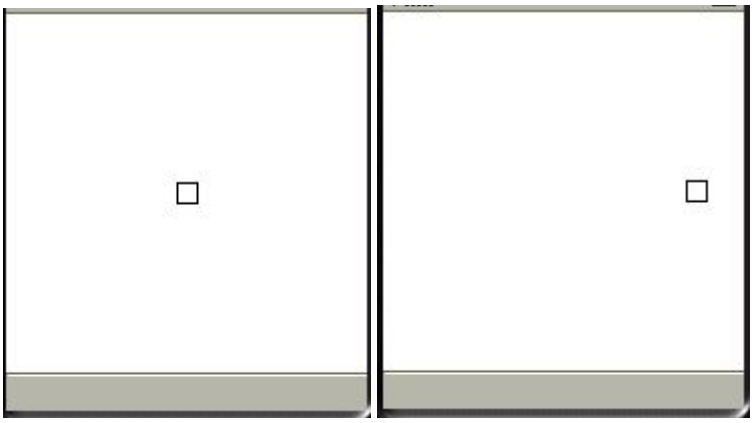
Et soovin ruudu algselt paigutada ekraani keskele, siis käsu showNotify käivitumise ajal leian vastavad koordinaadid. Varem poleks see võimalik, sest ekraani suurus ei pruugi programmile teada olla.

```
import javax.microedition.lcdui.*;

class Louend3 extends Canvas{
  int x, y;
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawRect(x-5, y-5, 10, 10);
  }
  protected void keyPressed(int kood){
    if(getGameAction(kood)==Canvas.LEFT){x--;}
    if(getGameAction(kood)==Canvas.RIGHT){x++;}
    repaint();
  }
  public void showNotify(){
    x=getWidth()/2;
    y=getHeight()/2;
  }
}
```

Käivitamislõik sama lühike nagu mujalgi.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Joonistus3 extends MIDlet{
  Canvas louend=new Louend3();
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
}
```

Valides liigutamist näitava rakenduse võib edaspidi vastavate klahvide abil liigutada ekraanil paiknevat ruutu.



Liikumine

Soovides, et ekraanil miski iseeneslikult liiguks, tuleb pilt iga natukese aja tagant uuesti arvutada ja joonistada. Üheks võimaluseks on kasutada klasside Timer ja TimerTask abi. Viimases kirjeldatakse, mida iga sammu jooksul tegema peab. Esimene aga palub tegevust soovitud aja tagant korrata. Liikumisega seotud teevused koondati meetodisse liigu, TimerTaski eksemplari ülesandeks on vaid vastav käsklus välja kutsuda. Selleks kaetakse üle run-meetod ning pannakse liikumiskäsklus sinna sisse. Käsklus schedule käsib kokkupandud tegevust iga saja millisekundi tagant välja kutsuda, alustades esimest korda 500 millisekundit pärast schedule käsu väljakutset. Ekraani peitmisel välja kutsutav hideNotify katkestab Timeri kordamistöö.

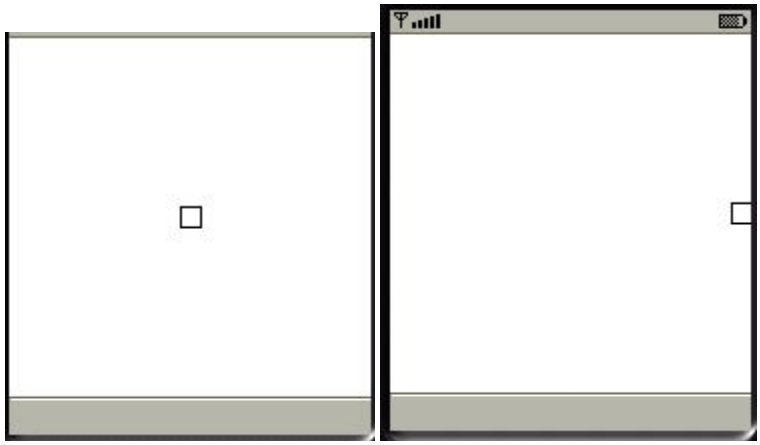
Liikumisel hoolitsetakse, et ruut üle ekraani serva ei liiguks. Kui käiguga satutaks üle serva, siis enne seda pannakse samm nulliks ning uuesti liikuda kannatab vaid klahvivajutuse puhul ja suunaga seinast eemale.

```
import javax.microedition.lcdui.*;
import java.util.*;

class Louend4 extends Canvas{
  int x, y, samm=0, r=5;
  Timer t;
  protected void paint(Graphics g){
     g.setColor(255, 255, 255);
     g.fillRect(0, 0, getWidth(), getHeight());
     g.setColor(0, 0, 0);
     g.drawRect(x-r, y-r, 2*r, 2*r);
  }
  protected void keyPressed(int kood){
    if(getGameAction(kood)==Canvas.LEFT){samm=-1;}
    if(getGameAction(kood)==Canvas.RIGHT){samm=1;}
  }
  public void showNotify(){
    x=getWidth()/2;
    y=getHeight()/2;
    t=new Timer();
    TimerTask tt=new TimerTask(){
      public void run(){
        liigu();
      }
    };
    t.schedule(tt, 500, 100); //tegevus, viivitus, intervall
  }

  public void hideNotify(){
    t.cancel();
  }

  void liigu(){
     if((samm>0 && x+samm+r>getWidth()) ||
        (samm<0 && x+samm-r<0)){
       samm=0;
     } else {
       x+=samm;
       repaint();
     }
  }
}
```

Käivituskood jälle võimalikult lühike.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Joonistus4 extends MIDlet{
  Canvas louend=new Louend4();
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(louend);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{}
}
```

Nagu pildilt näha, jääb liikuv ruut servani jõudnult pidama.



Andmed veebist

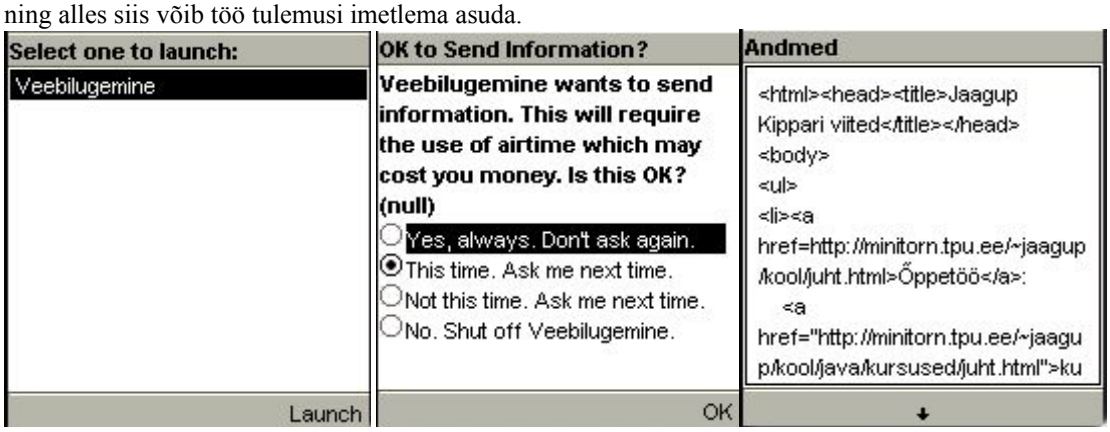
Et mobiiltelefon küllalt hulgaliselt andmeid välisilmaga vahetab, siis oleks patt jätta sealsed Java-programmid suhtlusvõimalusest välja. Samas aga igasugune ühendamine võib kulutada nii telefoniomaniku raha kui pakub võõrale programmile võimalusi veebiserverite värateid lõgistada ning muidki salapäraseid toimetusi ette võtta. Et Java puhul on püütud turvataset küllalt kõrgel hoida, siis ligipääs seadme võimalustele on küllaltki piiratud. Tavavahenditega ei pääse ligi ei telefoniraamatule, äratuskellale ega mujalegi. Mis ära keelatud, sealt pole ka ohtu karta. Samuti pole ju kõik J2ME seadmed sugugi mobiiltelefonid ning nende soovid ja võimalused sootuks teistsugused. Võrguliiklust aga mõnevõrra siiski lubatakse. Esimestest J2ME versioonidest alates töötab HTTP, vaikselt lisandub muidki ühenduskanaleid. Ärirakenduste loojate südameid soojendab HTTPS. Sest päris paljud ei raatsi oma paroole ja teadmisi vabalt üle võrgu liikuma saata. Samuti võib sidet pidada pistikliidese kaudu – seal juba antakse programmeerijale päris piisavalt ise otsustamisvõimalusi. Õnnelikul kombel saavad igasugused ühendused alguse klassist Connector. Nõnda tuleb siis lihtsalt käskude ja parameetrite abil proovida ja otsustada, et milline ühendusliik sobivaks osutub.

Kui liigutatavad andmed keerukamad, siis tasub alumise kihi peale ehitada/paigutada mõni paindlikum ülekandevahend. Näiteks XMLi-põhine, kui andmete suurem maht liialt raskusi ei valmista. Siin aga avatakse lihtsalt veebiaadres ning loetakse sealsed andmed tekstialasse. Nähtud näite järgi kannatab aga tunduvalt keerukamaid andmete küsimise ja salvestamise rakendusi kokku panna.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.*;
public class Veebilugeja1 extends MIDlet {
  TextBox t1;
  public Veebilugeja1(){
   try{
    String aadress="http://www.tpu.ee/~jaagup/";
    InputStream sisse=Connector.openInputStream(aadress);
    ByteArrayOutputStream baos=new ByteArrayOutputStream();
    int arv=sisse.read();
    while(arv!=-1){
      baos.write(arv);
      arv=sisse.read();
```

```
    }
    byte[] b=baos.toByteArray();
    String tekst=new String(b);
    t1=new TextBox("Andmed", tekst, tekst.length(), TextField.ANY);
    Display.getDisplay(this).setCurrent(t1);
   }catch(Exception e){
     e.printStackTrace();
   }
  }
  protected void startApp( ) throws MIDletStateChangeException{}
  protected void pauseApp( ){}
  protected void destroyApp( boolean p1 ) throws MIDletStateChangeException{}
}
```

Nagu kopeeritud piltidelt paistab, küsitakse kõigepealt kasutajalt luba võrguühenduse loomiseks ning alles siis võib töö tulemusi imetlema asuda.



Salvestus.

Miniseadmetes talletatakse andmeid kirjetena. Java käskudega õnnestub andmeid talletada vaid baidimassiivi. Ülejäänud muundused tuleb kirjutamisel ja lugemisel ise läbi teha. Andmete hoidmisega seotud vahendid paiknevad paketis javax.microedition.rms. Järgneva näitrakendusena koostatakse loendur teatamaks, mitmendat korda rakendusse sisenetakse.

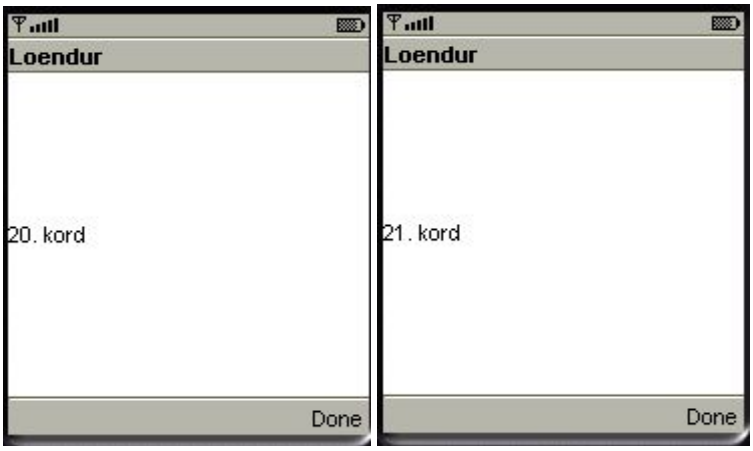
Loendur

Hoidla avatakse käsuga RecordStore.openRecordStore parameetriteks hoidla nimi sõnena ning tõeväärtusmuutuja teatamaks, kas vastava nimega hoidla puudumisel hoidla luuakse. Kui hoidlas pole ühtegi kirjet (`rs.getNumRecords()` väljastab 0), siis lisatakse hoidlasse ühebaidine kirje, mille arvuliseks väärtuseks siis sisenemiskordade arv järgmisel korral. Kui aga kirje baasis juba olemas, siis küsitakse esimese kirje bait number 0 (ehk selle kirje ainuke bait). Jäetakse selle väärtus muutujasse meelde ning kirjesse kirjutatakse eelmisest ühe võrra suurem väärtus. Käsklus `rs.setRecord(1, new byte[]{(byte)(arv+1)}, 0, 1)` tähendab lahtiseletatult, et kirje nr. 1 väärtuseks määratakse baidimassiivi baidid alates numbrist 0 üks bait. Ning massiiv luuaks kohapeal, pannes selle ainukese elemendi väärtuseks endisest arvust ühe võrra suurema väärtuse. Teade antakse välja Alert-akna abil.

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;

public class Salvestus1 extends MIDlet{
  String hoidlanimi="hoidla1";
  protected void startApp() throws MIDletStateChangeException{
   try{
    RecordStore rs=RecordStore.openRecordStore(hoidlanimi, true);
    byte arv=1;
    if(rs.getNumRecords()==0){
      rs.addRecord(new byte[]{(byte)(arv+1)}, 0, 1);
    } else {
      arv=rs.getRecord(1)[0];
      rs.setRecord(1, new byte[]{(byte)(arv+1)}, 0, 1);
    }
    Alert a=new Alert("Loendur", arv+". kord", null, AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
```

```
    Display.getDisplay(this).setCurrent(a);
   }catch(Exception e){e.printStackTrace();}
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
              throws MIDletStateChangeException{}
}
```

Nagu allolevalt pildilt näha võib, iga avamisega loenduri väärtus kasvab.



Neljabaidine salvestus

Vähegi rohkemate andmete salvestamisel tuleb liht- ja struktuuritüüpides paiknevaid väärtusi baidimassiiviks ja tagasi muundama hakata. Üheks võimaluseks on ise bitte nihutada, |-märkidega ühendada ning pärast &-märkidega eraldama asuda. Teiseks võimaluseks on ka J2ME-sse kaasa võetud java.io pakettidega arvestada. ByteArrayOutputStream võimaldab ette määramata hulga baite mällu saata ning sealt pärast massiivina välja lugeda. DataOutputStream'i ülesandeks on etteantud lihttüübid või sõned baitidena teele saata.. Teistpidise lugemise juures aitavad DataInputStream ning ByteArrayInputStream. Vastavalt siis allpool näha funktsioonid nimedega intBaitideks ning baididIntiks. Kui andmed sobivale kujule teisendatud on, siis edasine salvestamine ja küsimine käib sarnaselt kui eelmisegi näite puhul. rs.addRecord(bm, 0, bm.length) salvestab lihtsalt massiivi kogu pikkuses kirjena ning rs.getRecord(1) annab vastavalt kirje numbrile massiivi taas välja. Rakendus töötab sarnaselt kui eelminegi, oskab aga ühebaidilise väärtuse piiridest hulga kaugemale lugeda.

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class Salvestus1a extends MIDlet{
  String hoidlanimi="hoidla1a";
  protected void startApp() throws MIDletStateChangeException{
   try{
    RecordStore rs=RecordStore.openRecordStore(hoidlanimi, true);
    int arv=1;
    if(rs.getNumRecords()==0){
      byte[] bm=intBaitideks(arv+1);
      rs.addRecord(bm, 0, bm.length);
    } else {
      arv=baididIntiks(rs.getRecord(1));
      byte[] bm=intBaitideks(arv+1);
      rs.setRecord(1, bm, 0, bm.length);
    }
    rs.closeRecordStore();
    Alert a=new Alert("Loendur",
          arv+". kord", null, AlertType.CONFIRMATION);
    a.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(a);
   }catch(Exception e){e.printStackTrace();}
  }
  protected void pauseApp(){}
```

```
  protected void destroyApp(boolean kohustus)
                throws MIDletStateChangeException{}
  byte[] intBaitideks(int a) throws IOException{
     ByteArrayOutputStream bos=new ByteArrayOutputStream();
     DataOutputStream dos=new DataOutputStream(bos);
     dos.writeInt(a);
     dos.close();
     return bos.toByteArray();
  }
  int baididIntiks(byte[] b) throws IOException {
    ByteArrayInputStream bis=new ByteArrayInputStream(b);
    DataInputStream dis=new DataInputStream(bis);
    return dis.readInt();
  }
}
```

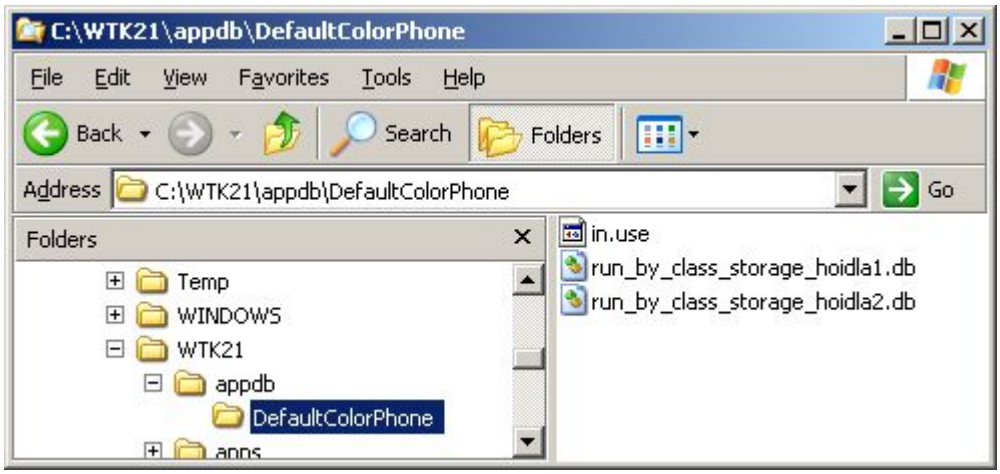
Salvestus vormist

Järgnev näide salvestab ühise kirjena inimese nime ja ta sünnipäeva. Lugemisnäide toodud hiljem. Ühendus andmehoidlaga avatakse kohe rakenduse algul ning suletakse rakenduse sulgumisel destroyApp-funktsiooni sees. (Vea)teadete edastamiseks loodi eraldi väike alamprogramm.

```
  void teade(String teade){
    Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
    a.setTimeout(5);
    Display.getDisplay(this).setCurrent(a);
  }
```

Nõnda õnnestub teade kergema vaevaga kasutajale ekraanile manada.

Kõrvalepilguks: arenduskeskkonnas salvestatakse andmed eraldi kataloogi. Keskkonna juurkataloogist alates appdb ning vastava emulaatori kataloog. Iga andmebaasi tarvis luuakse omaette fail ning fail in.use näitab, kui baas parajasti rakenduse poolt hõivatud on.



Järgnevalt salvestusvõimelise rakenduse kood.

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class Salvestus2 extends MIDlet implements CommandListener{
  String hoidlanimi="hoidla2";
  Form f=new Form("Sünnipäevad");
  Command lisa=new Command("Lisa", Command.SCREEN, 1);
  TextField tfnimi=new TextField("Eesnimi", "", 20, TextField.ANY);
  DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
  RecordStore rs;
  public Salvestus2(){
    f.append(tfnimi);
    f.append(dfsynniaeg);
    f.addCommand(lisa);
    try{
     rs=RecordStore.openRecordStore(hoidlanimi, true);
    }catch(Exception e){teade(e.getMessage());}
  }
  protected void startApp() throws MIDletStateChangeException{
```

```
    Display.getDisplay(this).setCurrent(f);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus)
                throws MIDletStateChangeException{
    try{
      rs.closeRecordStore();
    } catch (Exception e){teade(e.getMessage());}
  }
  public void commandAction(Command c, Displayable d){
    if(c==lisa){
      lisaKirje();
    }
  }

  void lisaKirje(){
    byte[] bm=leiaBaidiMassiiv();
    try{
     rs.addRecord(bm, 0, bm.length);
    }catch(Exception e){
      teade(e.getMessage());
    }
  }
  byte[] leiaBaidiMassiiv(){
   ByteArrayOutputStream bos=new ByteArrayOutputStream();
   DataOutputStream dos=new DataOutputStream(bos);
   try{
    dos.writeUTF(tfnimi.getString());
    dos.writeLong(dfsynniaeg.getDate().getTime());
   }catch(Exception e){teade(e.getMessage());}
   return bos.toByteArray();
  }

  void teade(String teade){
    Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
    a.setTimeout(5);
    Display.getDisplay(this).setCurrent(a);
  }
}
```

Mitu ekraanivormi

Eelmise täiendatud variant, kus peale andmete salvestamise ka neid näidata saab. Tegutsemine on jäetud mitme ekraanivormi peale. Vormide vahel liikumine on püütud korraldada nõnda nagu suuremate rakenduste puhul tavaks. Et liikudes mööda puud sügavamale õnnestub jälle tuldud teed pidi tagasi tulla. Põhiprogrammi sisse on paigutatud lisaks kolm klassi: valikuvormi, lisamise ning vaatamise tarbeks. MIDleti klassi kasutatakse vaid käivitamise tarbeks ning üldiste väärtuste ja vahendite kogumina.

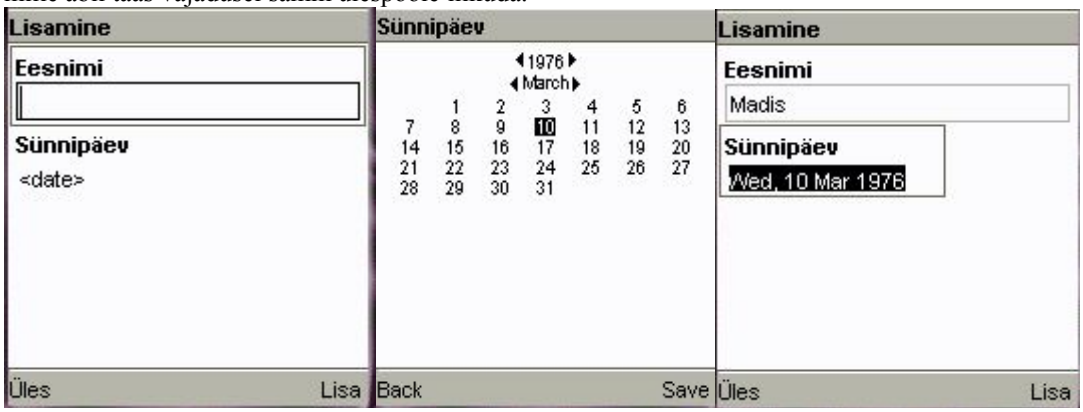
Valikuklass on loodud Listi alamklassina. List nagu Form või TextBox on üle kogu ekraani paiknev element, nõnda ka selle alamklass. Vaatamisvalikul paistab ees olema väike pilt.

Listi elementide lisamiseks kasutatakse käsku append. Pildi puudumisel on teiseks parameetriks null, pidi leidumisel aga Image-tüüpi objekt.

```
      super("Tegevuse valik", List.IMPLICIT);
      append("Lisa", null);
      Image vaatamispilt=null;
      try{
        vaatamispilt=Image.createImage("/silmad.png");
      } catch(IOException e){
        teade("Probleem pildi laadimisel");
      }
      append("Vaata", vaatamispilt);
```

Pildid ja muud ressursifailid tuleb paigutada rakenduse alamkataloogi res. Sealt leitakse need üles.

On kord lisamisvorm valitud, võib nime lihtsalt sisse tippida. Sünnipäeva sisestamiseks sobib aga DateField, mille abil siis õige kuupäev välja vaadata. Ning iga ekraanitäie vasakul pool asub käsklus, mille abil taas vajadusel samm ülespoole liikuda.



Nii lisamisvormi kui vaatamisvormile antakse kaasa nende ülemakna muutuja – praegusel juhul siis valikuvormi oma. Nii õnnestub kohe sobivat teed pidi taas puud pidi samm tagasi astuda.

```
LisamisVorm lisamine=new LisamisVorm(this);
VaatamisVorm vaatamine=new VaatamisVorm(this);
```

Peremehe ehk väljakutsuva akna meeles pidamiseks hoitakse meeles Displayable-tüüpi muutuja.

```
Displayable avaja;
LisamisVorm(Displayable avaja){
   super("Lisamine");
   this.avaja=avaja;
   ...
}
```

Ülesliikumise nupule vajutades saab siis lihtsalt avaja-ekraani aktiivseks muuta. Põhiklassi saab kätte konstruktsiooniga Salvestus3.this. Mõnes paigas soovitatakse selle asemel kasutada ka vastava klassi külge pandud staatilist meetodit, mis siis väljastaks vastava klassi ainukese eksemplari. Et aga siin ehitus sisemiste klasside abil kokku pandud, õnnestub ka praegusel kujul ligi pääseda.

```
public void commandAction(Command c, Displayable d){
  if(c==yles){
    Display.getDisplay(Salvestus3.this).setCurrent(avaja);
  }
  ....
}
```

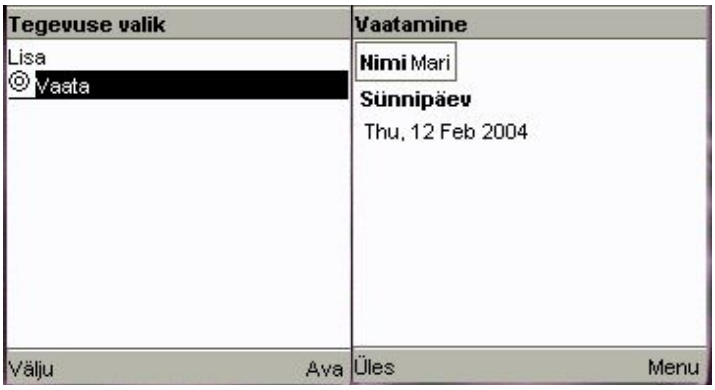
Vaatamise puhul siis õnnestub lihtsalt mööda kirjeid edasi ja tagasi liikuda ning väärtusi vaadata. Liikumist hõlbustab

```
RecordEnumeration re;
```

Kui see kord baasiühenduse käest küsitud ning viimase true abil uuendatavaks muudetud, siis õnnestub lihtsalt käskude abil mööda kirjeid edasi/tagasi liikuda.

```
re=rs.enumerateRecords(null, null, true);
```

Eelnevate null-väärtuseliste parameetrite abil võinuks omaloodud filtri abil osa kirjeid välja sorteerida või siis enesele soovitud järjekorda sättida. Sellisel juhul tuleks aga kirje baidijada järgi otsustada, milline kirje enesele sobib või mille järgi kirjeid omavahel reastada. Nagu pildi pealt paistab, võib vaatamise alt inimese andmeid näha.



Ja lõpuks siis selle salvestusrakenduse kood.

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import java.io.*;
import java.util.*;
public class Salvestus3 extends MIDlet{
  String hoidlanimi="hoidla2";
  RecordStore rs;
  RecordEnumeration re;
  ValikuVorm valik=new ValikuVorm();
  public Salvestus3(){
    try{
     rs=RecordStore.openRecordStore(hoidlanimi, true);
     re=rs.enumerateRecords(null, null, true);
    }catch(Exception e){teade(e.getMessage());}
  }
  protected void startApp() throws MIDletStateChangeException{
    Display.getDisplay(this).setCurrent(valik);
  }
  protected void pauseApp(){}
  protected void destroyApp(boolean kohustus) throws MIDletStateChangeException{
    try{
      rs.closeRecordStore();
    } catch (Exception e){teade(e.getMessage());}
  }


  void teade(String teade){
    Alert a=new Alert("Teade", teade, null, AlertType.CONFIRMATION);
    a.setTimeout(5);
    Display.getDisplay(this).setCurrent(a);
  }
  class ValikuVorm extends List implements CommandListener{
     LisamisVorm lisamine=new LisamisVorm(this);
     VaatamisVorm vaatamine=new VaatamisVorm(this);
     Command ava=new Command("Ava", Command.SCREEN, 1);
     Command valju=new Command("Välju", Command.EXIT, 1);
     ValikuVorm(){
       super("Tegevuse valik", List.IMPLICIT);
       append("Lisa", null);
       Image vaatamispilt=null;
       try{
         vaatamispilt=Image.createImage("/silmad.png");
       } catch(IOException e){
         teade("Probleem pildi laadimisel");
       }
       append("Vaata", vaatamispilt);
       setSelectCommand(ava);
       addCommand(valju);
       addCommand(ava);
       setCommandListener(this);
     }
     public void commandAction(Command c, Displayable d){
       if(c==valju){
          notifyDestroyed();
       }
       if(c==ava){
         if(getSelectedIndex()==0){
           Display.getDisplay(Salvestus3.this).setCurrent(lisamine);
         }
         if(getSelectedIndex()==1){
           Display.getDisplay(Salvestus3.this).setCurrent(vaatamine);
```

```
        }
      }
    }
}
class LisamisVorm extends Form implements CommandListener{
  TextField tfnimi=new TextField("Eesnimi", "", 20, TextField.ANY);
  DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
  Command lisa=new Command("Lisa", Command.SCREEN, 1);
  Command yles=new Command("Üles", Command.EXIT, 1);
  Displayable avaja;
  LisamisVorm(Displayable avaja){
     super("Lisamine");
     this.avaja=avaja;
     append(tfnimi);
     append(dfsynniaeg);
     addCommand(lisa);
     addCommand(yles);
     setCommandListener(this);
  }
  public void commandAction(Command c, Displayable d){
    if(c==yles){
      Display.getDisplay(Salvestus3.this).setCurrent(avaja);
    }
    if(c==lisa){
      lisaKirje();
    }
  }

  void lisaKirje(){
    byte[] bm=leiaBaidiMassiiv();
    try{
     rs.addRecord(bm, 0, bm.length);
     dfsynniaeg.setDate(null);
     tfnimi.setString("");
    }catch(Exception e){
      teade(e.getMessage());
    }
  }

  byte[] leiaBaidiMassiiv(){
   ByteArrayOutputStream bos=new ByteArrayOutputStream();
   DataOutputStream dos=new DataOutputStream(bos);
   try{
    dos.writeUTF(tfnimi.getString());
    dos.writeLong(dfsynniaeg.getDate().getTime());
   }catch(Exception e){teade(e.getMessage());}
   return bos.toByteArray();
  }
}

class VaatamisVorm extends Form implements CommandListener{
  StringItem snimi=new StringItem("Nimi", "");
  DateField dfsynniaeg=new DateField("Sünnipäev", DateField.DATE);
  Command edasi=new Command("Edasi", Command.SCREEN, 1);
  Command tagasi=new Command("Tagasi", Command.SCREEN, 1);
  Command yles=new Command("Üles", Command.BACK, 1);
  Displayable avaja;
  VaatamisVorm(Displayable avaja){
     super("Vaatamine");
     this.avaja=avaja;
     append(snimi);
     append(dfsynniaeg);
     addCommand(edasi);
     addCommand(tagasi);
     addCommand(yles);
     setCommandListener(this);
  }
  public void commandAction(Command c, Displayable d){
   try{
    if(c==yles){
      Display.getDisplay(Salvestus3.this).setCurrent(avaja);
    }
    if(c==edasi){
     if(re.hasNextElement()){
      loeBaidiMassiivist(re.nextRecord());
     }
    }
    if(c==tagasi){
      if(re.hasPreviousElement()){
        loeBaidiMassiivist(re.previousRecord());
      }
    }
```

```
     }catch(Exception e){
       teade(e.getMessage());
     }
    }
    void loeBaidiMassiivist(byte[] b){
     try{
      DataInputStream dis=
          new DataInputStream(new ByteArrayInputStream(b));
      snimi.setText(dis.readUTF());
      dfsynniaeg.setDate(new Date(dis.readLong()));
     }catch(IOException e){
       teade(e.getMessage());
     }
    }
  }
}
```

Ülesandeid

Mobiiliprogrammidega tutvumine.

* Installeeri arendusvahend.
* Tutvu kaasatulnud demonäidetega.
* Loo uus projekt.
* Lisa src-kataloogi tervitusnäide.
* Kompileeri, käivita.
* Muuda tervituse teksti. Testi.

* Lisa samasse projekti kalkulaatori näide. Testi.
* Lisa tehtevalikuga kalkulaatori näide.
* Lisa kalkulaatorile astendustehe.

Hinnaotsing

* Massiivis on meeles kõneminuti hinnad (sentides) võrkude kaupa, teises massiivis võrkude nimed. Valitakse võrk ning sisestatakse minutite arv ning väljastatakse kõne maksumus.
* Iga võrgu kohta on kirjas, millisest kellaajast alates kehtib milline tariif. Valitakse võrk, sisestatakse alguskellaaeg ja minutite arv ning väljastatakse kõne maksumus.
* Lisaks eelmisele arvestatakse maksumus õigesti ka juhul, kui kõne jooksul minutihind muutub.

Joonistus

Tutvu joonistusnäidetega.
* Koosta pilt: päike, kuusepuu, istepink.
* Arvesta joonistamisel ekraani suurusega.
* Kasuta juhuarve nii, et igal avamisel tuleks pilt mõnevõrra erinev.

Aardepüüdmismäng

* Tutvu liigutamisnäitega
* Võimalda kujundil noolte abil liikuda kõigi nelja ilmakaare suunas.
* Ekraanil juhuslikku kohta tekib ring. Selleni jõudmisel hüppab ring uude
  kohta.
* Platsi keskel on sein, millest ei saa liikumisel läbi minna.
* Iga tabamusega tuleb üks sein juurde.

Munapüüdja

Kaheksakümnendatel levinud mängu mobiilivariant.
* Joonista ekraani kummagisse serva kaks kaldteed.
* Mööda ühte teed veereb alla muna.
* Kui üks muna on alla jõudnud, hakkab kukkuma teine.
* Mune võib ühel teel liikuda korraga mitu.
* Mune võib igal teel liikuda korraga mitu.
* Joonistamisel ja arvutamisel arvestatakse ekraani mõõtmetega.

* Klahvivajutusega saab määrata, millise kaldtee all on püüdmislaud.
* Loetakse kinni püütud ning maha kukkunud mune.
* Lihtsa laua asemel on iga asendi puhul püüdja pilt.

Salvestusrakendus

* Tutvu salvestusnäidetega.
* Muuda andmevoogude abil baidimassiiviks punkti tekstiväljadest loetud punkti koordinaate tähistavad kaks täisarvu.
* Salvesta andmed kirjena hoidlasse.
* Loo eraldi nupp andmete lugemiseks hoidlast ja välja näitamiseks.

Kaart

* Nooltega saab liigutada ekraanil paiknevat ristkülikut.
* Jooksva asukoha koordinaadid talletatakse vastava käsu peale kirjena ning joonistatakse pildil ringina. Testi ringide säilimist rakenduse taasavamisel.
* Olemasoleva ringi kohal olles saab vastava käskluse abil ringi kustutada.
* Iga punkti juurde saab lisada tekstilise seletuse.

Veebirakenduse mobiililiides

* Tutvu näitega J2ME veebiühenduse kohta.
* Otsi üles/koosta veebi näidatav andmetabel laulude ja esitajatega.
* Loo veebileht, kus oleks näha vaid laulude nimed, üks nimi ühel real.
* Loe lehe sisu tekstialasse mobiiliekraanil.

* Loo veebi väljastav leht, kus oleks real näha laulu id, tühik ning laulu nimi.
* Loo mobiili neist valik, kui kasutaja saab soovitud laulu märgistada.
* Pärast märgistamist ja valimist näidatakse selle laulu andmeid eraldi mobiiliekraanil.

* Loo vahend laulude ja esitajate lisamiseks mobiili kaudu.
* Loo vahend andmete kustutamiseks.

* Hoolitse, et lisada ja kustutada saaks vaid registreeritud kasutajanime ja parooliga.

XML

Kasutusvaldkonnad, süntaks, töötlemine programmidega

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

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

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

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

XSL

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

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

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

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

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

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

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

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

Edasine on juba rohkem andmetöötlusega seotud. Käsklust

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

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

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

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

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

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

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

Ning edasi siis tuleb kõik lahti jäänud elemendid lõpetada.

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

Nüüd siis esimeseks näiteks toodud stiililehe kood tervikuna silma ette.

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

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

Käivitamine

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

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

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

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;
public class XSLMuundur{
  public static void main(String argumendid[]) throws Exception{
    if(argumendid.length!=3){
      System.err.println("Kasuta kujul java XSLMuundur andmefail.xml muundefail.xsl
tulemusfail.html");
      System.exit(0);
    }
    Transformer tolkija=TransformerFactory.newInstance().
      newTransformer(new StreamSource(argumendid[1]));
    tolkija.transform(
      new StreamSource(argumendid[0]),
      new StreamResult(new FileOutputStream(argumendid[2]))
    );
  }
}
```

Kui kood kompileeritud, võib sella käima panna.

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

Ning loodud tekstifaili sisu piiludes saab programmi töö tulemusi imetleda.

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

   Esimene:Juku;
   Viimane:Oskar
```

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

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

   Esimene:Juku;
   Viimane:Oskar
```

Ühenimelised elemendid

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

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

```
 <xsl:template match="/">
       <xsl:for-each select="//inimene[synd>'1970']">
         <xsl:value-of select="eesnimi" />
       </xsl:for-each>
 </xsl:template>
 </xsl:stylesheet>

KalleMariOskar
```

Andmed tabelina

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

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

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

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

Tabeli enese, rea ning lahtri alguse ja lõpu elemendid paigutatakse lihtsalt sobivatesse kohtadesse vahele. Paljas punkt xsl:value-of elemendi sees palub kirjutada jooksva elemendi väärtuse. Nõnda kui välimene for-each käib läbi kõik inimesed ning sisemine iga inimese alamtunnused ning andmefailis on kõigil inimestel sama palju tunnuseid saabki kokku täiesti viisaka HTML-tabeli.

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

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

</xsl:stylesheet>
```

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

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

```
</body>
</html>
```

Mallid

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

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

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

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

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

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

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

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

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

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

Tekstikontroll

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

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

Kandiliste sulgude sees määratakse ära, millistele tingimustele vastavaid inimesi loetellu võetakse. Siin juhul siis kontrolliks XSLi funktsioon nimega starts-with, parameetriteks kontrollitav tekst ning otsitav algustekst. Ning nagu muud tõeväärtusfunktsioonid, nii ka siin on väärtuseks jah või ei. Ning loetellu jäävad need inimesed, kel avaldise puhul väärtuseks jah. Ning nõnda saab tsükli sees otsitud tulemused välja kirjutada.

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

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

Käivitamine nii nagu eelnenud näite puhul

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

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

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

XSL-i funktsioone

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

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

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

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

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

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

```
number(objekt)
```

```
string(objekt)

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

Sõnefunktsioonid

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

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

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

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

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

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

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

Parameetrid

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

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

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

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

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

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

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

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

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

Ning näide tervikuna.

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

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

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

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

Käivitus

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

ja tulemus

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

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


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

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

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

Muus osas näeb faile ühendav käivitusprogramm eelnenuga sarnane välja.

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

Ka käivitamine sarnane

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

Ning tulemusena näeb siis nelja tähe pikkusi ja pikemaid nimesid.

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

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

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

Ülesandeid

XML

Sisestusharjutus

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

Andmepuu

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

XSL

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

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

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

Sugupuu

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

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

XML ja kassid

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

XML ja koerad

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

DOM

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

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

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

või siis lugeda sisse olemasolevast failist.

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

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

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

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

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

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

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

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

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

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

Järgnevalt programmi kood.

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

import org.w3c.dom.*;

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

Ning väljund.

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

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

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

Joonistusvahend

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

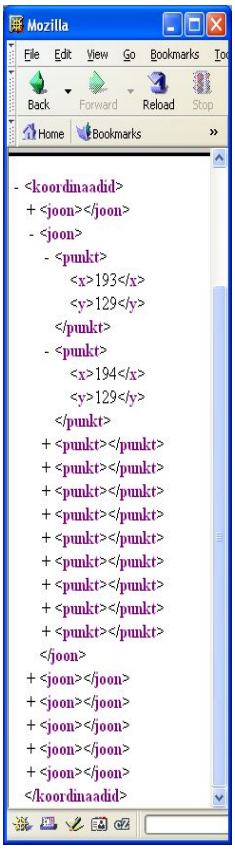
```
        int x1=Integer.parseInt(((Element)p1).getElementsByTagName("x").item(0).
                                                        getFirstChild().getNodeValue
());
```

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

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

Kui kõik neli koordinaati nõnda käes, võib tõmmata joone.

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



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

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

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

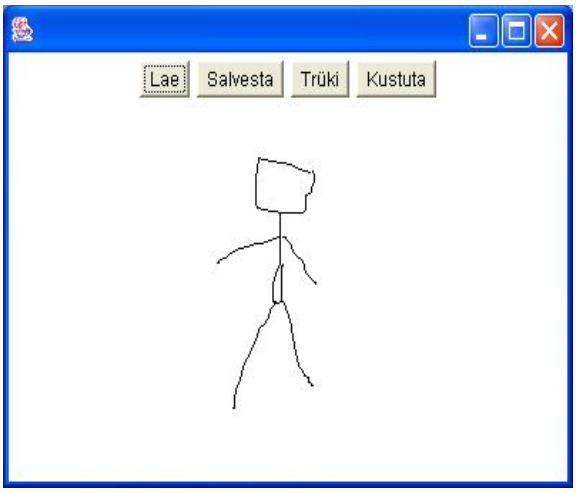
```
    if(e.getSource()==tryki){
       System.out.println(d.getFirstChild());}
```

Ning rakenduse kood tervikuna.

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

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

import org.w3c.dom.*;

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

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

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

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


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

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

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

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




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

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


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

SAX

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

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

Nimede loendur

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

```
import javax.xml.parsers.*;
```

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

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

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

Väljund nagu algandmete põhjal aimata oligi:

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

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

Elementide sisu

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

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

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

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

Iga kord eesnime elementi sisenedes luuakse uus tühi puhvri eksemplar. Meetodis characters liidetakse eesnimi tükkidest kokku, juhul kui andmed peaksid osade kaupa saabuma. Muidu pannakse puhvrisse lihtsalt üks terviklik tükk. Kui eesnime element lõpeb, siis trükitakse puhvri sisu ekraanile ning nõnda saabki kasutaja näha eesnimede loetelu.

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

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

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

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

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


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

*/
```

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

Turvalisus

Autentimine, krüptograafia, võtmed, õigused

Signeerimine

Lugedes teksti või käivitades programmi soovime mõnikord kindlad olla, et kellegi kuri käsi ega lihtsalt tehniline viperus pole algseid andmeid muutnud, et võime uskuda oma kettal olevaid andmeid samuti nagu usume oma käe ja tindiga telefoniraamatusse kirjutatud numbreid. Samuti soovime mõnikord kontrollida, et saadetis tuli ikka sellelt inimeselt, kust me arvasime, et see peaks tulema. Relvastatud valve ning mitmekordsete topelteksemplaride kõrval on üheks lahenduseks avaliku ja salajase võtme võtme krüptograafia, kus materjalide looja lisab andmete ning oma salajase võtme abil loodud digitaalallkirja. Hiljem saab autori avaliku võtme ning digitaalallkirja järgi kontrollida, kas andmed on muutumatul kujul säilinud. Nii võib kontrollida andmete terviklust oma kettal, sama moodusega võib ka kaugel asuv adressaat enamjaolt kindel olla, et kohale jõudnud saadetis ikka sellelt inimeselt pärit on, kellelt seda arvatakse olevat. Põhiehitus on pea kõigil sellistel kontrollivahenditel sarnane, erinevused on peamiselt krüptoalgoritmide ning pealisehituse osas. Levinumaid algoritme on kümmekond, kasutajaliideseid aga vähemalt nii palju kui ennast tähtsaks pidavaid tarkvaratootjaid ning eks aeg näitab, palju neid aja jooksul juurde tuleb või edasi areneb.

Java standardkomplekti kuuluvad keytool ning jarsigner. Esimese abil õnnestub võtmeid ja sertifikaate luua ning talletada, teise abil võib jar-arhiivi talletatud andmeid salajase võtmega signeerida ning avaliku võtmega kontrollida, kas arhiiv arvatud kohast tervena pärale jõudnud on.

Et õnnestuks andmeid turvama hakata, selleks peab omale võtmepaari looma. Keytoolile tuleb öelda, et soovime võtit luua (-genkey), määrata kuhu andmed panna (-keystore jaagup.store, muul juhul pandaks andmed faili .keystore) ning nimi mille juurde kuuluvaks võtmed luua. Tulemusena küsitakse veidi isikuandmeid ning loodud võtmed koos nendega väljastatakse sertifikaadina andmehoidlasse.

```
C:\User\jaagup\0104\turva\sert>keytool -genkey -keystore jaagup.store -alias jaagup
Enter keystore password:  123456
What is your first and last name?
  [Unknown]:  Jaagup Kippar
What is the name of your organizational unit?
  [Unknown]:  Informaatika oppetool
What is the name of your organization?
  [Unknown]:  TPU
What is the name of your City or Locality?
  [Unknown]:  Tallinn
What is the name of your State or Province?
  [Unknown]:  Harjumaa
What is the two-letter country code for this unit?
  [Unknown]:  EE
Is <CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE>
correct?
   [no]:  Y

Enter key password for <jaagup>
        (RETURN if same as keystore password):  123456
```

Kui soovin näha, kelle andmed võtmehoidlas on, siis piisab keytoolile anda käsklus list. Võtmehoidla teenusepakkujaks on SUN, Java looja ning hoidla tüübiks jks. Neid andmeid läheb vaja, kui hiljem soovida oma programmis hoidlast sertifikaate ning võtmeid lugeda. Sees on vaid üks sertifikaat, jaagupi-nimelisele isikule. Selge, sest sinna baasi pole rohkem kirjeid loodud. Kui teha või lugeda sertifikaate juurde, siis tuleb ka väljastatavaid ridu rohkem.

```
C:\User\jaagup\0104\turva\sert>keytool -list -keystore jaagup.store
Enter keystore password:  123456

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:

jaagup, Tue Apr 03 20:43:04 GMT+03:00 2001, keyEntry,
Certificate fingerprint (MD5): 0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
```

Soovides oma sertifikaati lähemalt uurida või sõbrale edasi anda, et too saaks hiljem kontrollida kas saadetis ikka minult on, tuleb hoidlast vastav sertifikaat eraldi faili kirjutada. Faili nimeks saab jaagup.cert.

```
C:\User\jaagup\0104\turva\sert>keytool -export -keystore jaagup.store -alias ja
gup -file jaagup.cert
Enter keystore password:  123456
Certificate stored in file <jaagup.cert>
```

Kui vaadata kataloogi, siis seal paistab kaks faili. .store-laiendiga sertifikaadihoidla ning .cert laiendiga üksik sertifikaat.

C:\User\jaagup\0104\turva\sert>dir

```
.              <DIR>        03.04.01  20:39 .
..             <DIR>        03.04.01  20:39 ..
JAAGUP~1 STO         1 278  03.04.01  20:43 jaagup.store
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
```

Soovides sertifikaadi sisuga lähemalt tutvuda, aitab võti -printcert .

```
C:\User\jaagup\0104\turva\sert>keytool -printcert -file jaagup.cert
Owner: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Issuer: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Serial number: 3aca0b9a
Valid from: Tue Apr 03 20:42:50 GMT+03:00 2001 until: Mon Jul 02 20:42:50 GMT+03:00
2001
Certificate fingerprints:
         MD5:  0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
         SHA1: AA:2B:73:28:A1:F9:B4:8F:71:D8:D8:C7:66:CC:37:14:36:8C:8F:EE
```

Kui tuttav (kelle kataloogiks on sert2) soovib edaspidi minult saadud teadete autentsust kontrollima hakata, siis ta muretseb enesele mu sertifikaadi. Kas lihtsalt kopeerib selle mu kataloogist või tähtsamal juhul saame pidulikult linnas kokku, kus talle disketil oma avalikku võtit sisaldava sertifikaadi ulatan.

```
C:\User\jaagup\0104\turva\sert2>copy ..\sert\jaagup.cert .
        1 file(s) copied
```

Saadud sertifikaati oma hoidlasse tõmmates peab Juku nüüd tõesti kindel olema, et tegemist oli kindlasti minu käest saadud kettaga ja keegi pole tema taskus vahepeal andmeid muutnud. Muul juhul võib see keegi tundmatu kergesti minu nime all esinema hakata ning vale tuleb välja alles siis, kui ise midagi Jukule saadan ning selle signatuur valeks loetakse.

```
C:\User\jaagup\0104\turva\sert2>keytool -import -keystore juku.store -alias jaagup
-file jaagup.cert
Enter keystore password:  123456
Owner: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Issuer: CN=Jaagup Kippar, OU=Informaatika oppetool, O=TPU, L=Tallinn, ST=Harjumaa, C=EE
Serial number: 3aca0b9a
Valid from: Tue Apr 03 20:42:50 GMT+03:00 2001 until: Mon Jul 02 20:42:50 GMT+03:00
2001
Certificate fingerprints:
         MD5:  0D:45:94:0E:55:A1:4F:70:D8:77:D2:ED:1F:1E:59:6E
         SHA1: AA:2B:73:28:A1:F9:B4:8F:71:D8:D8:C7:66:CC:37:14:36:8C:8F:EE
Trust this certificate? [no]:  y
Certificate was added to keystore
```

Igaks juhuks veel küsiti üle, et kas võtmehoidla omanik sellise sertifikaadi lisamisega nõus on. Kahtluste hajutamiseks võib üle kontrollida sertifikaadiga kaasnevad sõnumilühendid. Kui need on samasugused kui minu antud paberitükil, siis pole suurt põhjust enam kedagi sertifikaatide salajases vahetamises süüdistada, sest vähemalt lähiajal pole karta, et keegi suudaks ise samasuguse sõnumilühendiga andmeid luua.

Jukul niisiis praegu kataloogis kaks faili. Võtmehoidla ning minu sertifikaat. Kui viimane on hoidlasse imporditud, siis võib selle siit kustutada, sest edasised toimingud käivad hoidla kaudu.

```
C:\User\jaagup\0104\turva\sert2>dir
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
```

```
JUKU~1   STO           871  03.04.01  21:05 juku.store
```

Kui nüüd soovin Jukule saata signeeritud programmi, siis kõigepealt loon lähtekoodi
```
C:\User\jaagup\0104\turva\sert>edit Teretus.java
```
, kompileerin
```
C:\User\jaagup\0104\turva\sert>javac Teretus.java
```
ning arhiveerin selle jar-faili.
```
C:\User\jaagup\0104\turva\sert>jar cvf teretus.jar Teretus.class
added manifest
adding: Teretus.class(in = 429) (out= 297)(deflated 30%)
```

Edasi signeerin arhiivi oma salajase võtmega
```
C:\User\jaagup\0104\turva\sert>jarsigner -keystore jaagup.store teretus.jar jaagup
Enter Passphrase for keystore: 123456
```
ning saadan signeeritud arhiivi Jukule.
```
C:\User\jaagup\0104\turva\sert>copy teretus.jar ..\sert2
        1 file(s) copied
```
Avastades uue faili oma kataloogist või saades selle kirjaga võiks tal ikka huvi tekkida, kellelt saadetis on tulnud. Jarsigner pakub võimaluse
```
C:\User\jaagup\0104\turva\sert2>jarsigner -verify -keystore juku.store teretus.jar
jar verified.
```
ning teatab, et arhiiv sobis, st., et arhiivi oli signeerinud inimene, kelle sertifikaat asub Juku võtmehoidlas. Kui lisada käivitamisel võtmed -verbose ning -certs, siis on näha, kellelt fail tulnud on ning mis toiminguid kontrollimise ajal tehakse. Uskudes nüüd, et teade tuleb tuttavalt inimeselt kes talle halba ei soovi, võib Juku rahus arhiivi lahti pakkida,

```
C:\User\jaagup\0104\turva\sert2>jar xvf teretus.jar
extracted: META-INF/MANIFEST.MF
extracted: META-INF/JAAGUP.SF
extracted: META-INF/JAAGUP.DSA
  created: META-INF/
extracted: Teretus.class
```

vaadata huvi pärast, mis talle kataloogi tekkinud on

C:\User\jaagup\0104\turva\sert2>dir

```
JAAGUP~1 CER           808  03.04.01  20:59 jaagup.cert
JUKU~1   STO           871  03.04.01  21:05 juku.store
TERETUS  JAR         2 029  03.04.01  21:27 teretus.jar
META-INF       <DIR>        03.04.01  21:33 META-INF
TERETU~1 CLA           429  03.04.01  21:33 Teretus.class
```

ning saabunud programmi käima panna.

```
C:\User\jaagup\0104\turva\sert2>java Teretus
Soovin head lugemist!
```

Selgus, et oli tegemist lihtsa tervitusega.

Kui keegi muu sooviks Jukule minu nime alt kurja programmi saata,

```
class Suurtervitus{
  static void main(String argumendid[]){
   for(int i=1; i<10000; i++){
    System.out.println("Minge kõik kuu peale");
   }
  }
}
```

siis see tal ei õnnestu. Ta võib küll programmi kirjutada, kompileerida ja arhiveerida
```
C:\User\jaagup\0104\turva\sert>jar cvf suurtervitus.jar Suurtervitus.java
added manifest
adding: Suurtervitus.java(in = 158) (out= 130)(deflated 17%)
```

kuid minu privaatvõtmega seda naljalt signeerida ei õnnestu. Kui ta ka pääseks ligi mu kataloogi, siis seal on andmehoidlast võtme kätte saamiseks vaja lahti muukida sealne parool. Mujal aga samasugust võtit

välja mõtelda oleks peaaegu lootusetu. Kui õnnetu piraat otsustaks siiski signeerimata või mõne muu võtmega allkirjastatud faili Jukule saata,

```
C:\User\jaagup\0104\turva\sert>copy suurtervitus.jar ..\sert2
        1 file(s) copied
```

siis kohapeal kontrollides selguks, et tegemist pole õige asjaga.

```
C:\User\jaagup\0104\turva\sert2>jarsigner -verify -keystore juku.store suurtervitus.jar
jar is unsigned. (signatures missing or not parsable)
```

Digitaalallkiri

Mida keytool'i ning jarsigner'i abil saab kasutada valmis vahenditena, seda võib oma programmides ka ise teha, sest eks eelnimetatudki ole Java abil kokku kirjutatud programmid. Võtmepaaride loomiseks on KeyPairGenerator, sinna saab ette anda, millise algoritmi järgi võtmed genereerida. Käsud getPublic ning getPrivate annavad võtmepaarist vastavalt avaliku ning salajase võtme ning getEncoded neist annab võtme sisu baidijadana, mida edaspidi üle kanda või talletada saab. Järgnevas näites luuakse käivitajale failidesse teineteise juurde kuuluvad avalik ning sajalane võti.

```
import java.security.*;
import java.io.*;
public class Votmetelooja{
  public static void main(String argumendid[]) throws Exception{
    String avavotmefail="avavoti.key";
    String salavotmefail="salavoti.key";
    KeyPairGenerator votmepaarilooja=KeyPairGenerator.getInstance("DSA");
    votmepaarilooja.initialize(512); //jagamisja"a"k
    KeyPair votmepaar=votmepaarilooja.generateKeyPair();
    FileOutputStream valja1=new FileOutputStream(avavotmefail);
    valja1.write(votmepaar.getPublic().getEncoded());
    valja1.close();
    FileOutputStream valja2=new FileOutputStream(salavotmefail);
    valja2.write(votmepaar.getPrivate().getEncoded());
    valja2.close();
  }
}
```

Teate allkirjastamiseks tuleb failist lugeda või muul moel enesele kättesaadavaks teha salajane võti ning teate moodustavad andmebaidid. Salajast võtit hoitakse PKCS#8 standardi järgi, kus lisaks võtme väärtusele on kirjas ka andmed versiooni ning krüptimisalgoritmi kohta. Võtme (tüübist PrivateKey) väljastab KeyFactory ning failist saabuvad baidid aitab viiamasele suupäraseks teha `PKCS8EncodedKeySpec`. Kogu allkirjastamine ise toimub paari käsuga

```
    Signature allkirjastaja=Signature.getInstance("DSA");
    allkirjastaja.initSign(salavoti);
    allkirjastaja.update(andmebaidid);
    byte[] allkiri=allkirjastaja.sign();
```

, kus initSign salajase võtmega määrab, et järgnevalt update abil allkirjastajast läbi lastavad baidid muudavad allkirja ning sign väljastab baitidena allkirja, mis sõltub salajasest võtmest ning andmetest ning mida peaks pea võimatu olema salavõtme puudumisel järele teha.

```
import java.security.*;
import java.security.spec.*;
import java.io.*;
public class Allkirjastaja{
  public static void main(String argumendid[]) throws Exception{
    String salavotmefail="salavoti.key";
    String andmefail="andmed.txt";
    String allkirjafail="andmed.sig";
    byte[] salavotmebaidid=new byte[(int)new File(salavotmefail).length()];
    FileInputStream sisse=new FileInputStream(salavotmefail);
    sisse.read(salavotmebaidid);
    sisse.close();
    PrivateKey salavoti=KeyFactory.getInstance("DSA").generatePrivate(
      new PKCS8EncodedKeySpec(salavotmebaidid)
    );
    byte[] andmebaidid=new byte[(int)new File(andmefail).length()];
    sisse=new FileInputStream(andmefail);
    sisse.read(andmebaidid);
    sisse.close();
    Signature allkirjastaja=Signature.getInstance("DSA");
```

```
    allkirjastaja.initSign(salavoti);
    allkirjastaja.update(andmebaidid);
    byte[] allkiri=allkirjastaja.sign();
    FileOutputStream valja=new FileOutputStream(allkirjafail);
    valja.write(allkiri);
    valja.close();
  }
}
```

Kontrollimisel aitab samuti Signature. Käsu initVerify juures määratakse, millise avaliku võtme järele andmete allkirjale vastavust kontrollima hakatakse. Käsk verify väljastab "tõene", kui allkiri, andmed ja võti sobisid kokku, muul juhul loetakse kontroll ebaõnnestunuks.

```
import java.security.*;
import java.security.spec.*;
import java.io.*;
public class Allkirjakontrollija{
  public static void main(String argumendid[]) throws Exception{
    String avavotmefail="avavoti.key";
    String andmefail="andmed.txt";
    String allkirjafail="andmed.sig";
    byte[] avavotmebaidid=new byte[(int)new File(avavotmefail).length()];
    FileInputStream sisse=new FileInputStream(avavotmefail);
    sisse.read(avavotmebaidid);
    sisse.close();
    PublicKey avavoti=KeyFactory.getInstance("DSA").generatePublic(
      new X509EncodedKeySpec(avavotmebaidid)
    );
    byte[] andmebaidid=new byte[(int)new File(andmefail).length()];
    sisse=new FileInputStream(andmefail);
    sisse.read(andmebaidid);
    sisse.close();
    byte[] allkirjabaidid=new byte[(int)new File(allkirjafail).length()];
    sisse=new FileInputStream(allkirjafail);
    sisse.read(allkirjabaidid);
    sisse.close();
    Signature allkirjakontrollija=Signature.getInstance("DSA");
    allkirjakontrollija.initVerify(avavoti);
    allkirjakontrollija.update(andmebaidid);
    System.out.print("Andmed failist "+andmefail+
                     " ning allkiri failist "+allkirjafail+ " ");
    if(allkirjakontrollija.verify(allkirjabaidid)){
      System.out.println("sobivad.");
    } else {
      System.out.println("ei sobi.");
    }
  }
}
```

Kui programmide tööde tulemusi vaadata, siis kõigepealt loodi võtmefailid

```
AVAVOTI  KEY           244  05.04.01  14:51 avavoti.key
SALAVOTI KEY           202  05.04.01  14:51 salavoti.key
         2 file(s)            446 bytes
         0 dir(s)   1 603 133 440 bytes free
```

seejärel allkiri

```
C:\User\jaagup\0104\turva>java Allkirjastaja
```

ning soovides sobivust kontrollida, saime teada, et andmed ja allkiri sobisid kokku

```
C:\User\jaagup\0104\turva>java Allkirjakontrollija
```

Andmed failist andmed.txt ning allkiri failist andmed.sig sobivad.

```
C:\User\jaagup\0104\turva>type andmed.txt
Juku tuli koolist.

C:\User\jaagup\0104\turva>edit andmed.txt

C:\User\jaagup\0104\turva>type andmed.txt
Juku tuli koolist
```

Kui andmefaili kas või ühe punkti jagu muuta, siis saadakse teade, et

```
C:\User\jaagup\0104\turva>java Allkirjakontrollija
Andmed failist andmed.txt ning allkiri failist andmed.sig ei sobi.
```

Sõnumilühend

Kui piisab vaid tehnilisest teadmisest, et andmed pole tee peal kannatada saanud, siis üheks võimaluseks on saata andmeid mitu eksemplari ja pärast neid omavahel võrrelda. Mahukamate tekstide ja piltide korral aga võib teise eksemplarina kasutada sõnumilühendit. Tegemist on kavala baidijadaga, mida algsest tekstist on vastava algoritmi teel kerge luua, lühendi järgi teksti aga taastada pole võimalik. Lühendi eeliseks pikema teksti ees on lühidus. Sõltumata algandmete mahust on siinse algoritmi (SHA) järgi lühendiks ikkagi vaid 20 baiti, samas lühendi sisu muutub tundmatuseni, kui selle allikaks olnud kasvõi mitme gigabaidisest andmemassiivist vaid üks bitt muuta. Lühendit võib kasutada ka tervikluse ründe vastu. Kui näiteks tahame kindlad olla, et installeeritavasse tarkvarasse pole keegi pahalane pärast programmi ametlikku turule tulekut trooja hobust ega muud soovimatut lisanud, võime lasta installeerimisplaadist või installeeritud programmist sõnumilühendi arvutada ning tootjale helistada ja uurida, kas need ikka ilusti kokku langevad.

Java vahenditega käib sõnumilühendi loomine paari käsuga. Kõigepealt luuakse sobivat algoritmi oskav räsikoodi (sõnumilühendi) generaator. Käsule digest antakse ette andmebaidid, mille kohta tahetakse lühendit arvutada ning tulemusena väljastatakse baidimassiivina räsikood. Järgnev tsükkel lihtsalt trükib loodud koodi ekraanile.

```
import java.security.*;
public class Turva2{
  public static void main(String argumendid[]) throws Exception{
    String teade="Juku tuli koolist";
    byte[] teatebaidid=teade.getBytes();
    MessageDigest rasikoodigeneraator=MessageDigest.getInstance("SHA");
                    //Secure Hash Algorithm
    byte[] rasikood=rasikoodigeneraator.digest(teatebaidid);
    for(int i=0; i<rasikood.length; i++){
      System.out.print((rasikood[i] & 0xFF)+" ");
    }
  }
}
```

väljund:

```
233 166 35 250 225 13 241 133 131 60 82 76 215 146 95 66 163 89 142 64
```

Programmi õigused

Kui tekib oht, et käivitatav programm võimaldab meie andmetele või arvutile enesele mingil moel kurja teha, siis tuleks programmi õigusi nii piirata, et võiksime sel rahuliku südamega lasta tööd teha kartmata et salaja kas programmi kirjutaja pahatahtlikkuse või lihtsalt lohakuse tõttu võime millestki ilma jääda või et andmed, mida soovime vaid enese teada hoida, iseeneslikult võõraste silmade ette jõuavad. Rakendil näiteks on keelatud kohaliku failisüsteemiga suhelda ning võõrastesse masinatesse võrgu kaudu ühenduda, samuti mikrofonist häält lindistada. Nii võib kurjalt lehelt avatud programm küll hulga aknaid avada ning mälu ja võrguliiklust ummistada, kuid korralikult koostatud turvaaukudeta brauseri korral midagi tähtsat masinast kasutaja teadmata võrgu peale rändama või kaduma minna ei saa. Kui tekivad kahtlused rakendi kavatsuste osas, siis saame brauseri sulgeda ning kõik on vanaviisi nagu ennegi. Korraldatud on turvamine nii, et potentsiaalselt turvaohtlike käskude algusse on sisse kirjutatud pöördumine turvahalduri (SecurityManager) poole. Kui saadakse nõusolek, võib töö jätkuda, muul juhul heidetakse erind ning programm kas lõpetab töö või jätkab vastavalt erindi püüdmisega kaasnevatele korraldustele. Kuigi võimalikke käske, mis võiksid kasutajale liiga teha, on palju ning neid juurde kirjutades saaks kokku lugematu hulga, õnnestub kogu turvalisust siiski paarikümne õigusega juhtida, sest operatsioonisüsteemi ning muude turvaohtlike käskude poole pöördutakse vaid üksikutes kohtades ning kõik muud vastavat teenust vajavad käsud kasutavad sama "lüüsi". Näiteks kettalt lugemisel pöördutakse alati FileInputStream'i konstruktori poole, ükskõik kas on tegemist pildi avamise, teksti lugemise või kahendfaili analüüsiga. Kui selles konstruktoris lasta turvahaldurilt kontrollida, kas lugemine lubatud, siis saabki ketta kasutusõiguse ühes kohas määrata ning määrang kehtib igal pool kogu programmi ulatuses. Kui objektorienteerituse teoreetikud rõhutavad pea igal võimalikul juhul, et kui vähegi võimalik, siis ei tohi koodi kopeerida vaid tuleb terviku moodustavad käsud ühte meetodisse koondada ning see siis igas vajalikus kohas kas otse või teiste meetodite kaudu välja kutsuda. Nii võib programm mõnigikord pikemaks ja mõne intepretaatori korral ka aeglasemaks minna, kuid kui igal

toimingul on oma kindel koht, siis saab selle toiminguga seotud vea kergesti üles leida ning vajadusel käitumist ka muuta või keelata.

Kui iseseisval programmi laseksin ühendada end teise masinasse ning sealselt ajateenuselt kella küsida siis võrguühenduse olemasolul ning teise serveri korrasoleku puhul tõenäoliselt ilmub mulle ekraanile teises masinas olev kellaaeg.

```
C:\User\jaagup\0104\turva>type Kell1.java
import java.net.*;
import java.io.*;
public class Kell1{
  public static void main(String argumendid[]) throws Exception{
    Socket sc=new Socket("madli.ut.ee", 13);
    BufferedReader sisse=new BufferedReader(
      new InputStreamReader(sc.getInputStream())
    );
    System.out.println(sisse.readLine());
  }
}

C:\User\jaagup\0104\turva>java Kell1
Wed Apr  4 20:20:07 2001
```

Soovides aga hoolitseda, et käivitatav programm heast peast ei hakkaks ennast masinast väljapoole ühendama ega muid kahtlasi toiminguid tegema, võib virtuaalmasinale määrata uue kurja turvahalduri, kelle poolest vaid üksikuid toimingud on lubatud. Kui nüüd sama programm käivitada, siis teatatakse, et keelatud on juba madli.ut.ee-le vastava IP aadressi küsimine rääkimata sinna ühendumisest.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager Kell1
Exception in thread "main" java.security.AccessControlException: access denied (
java.net.SocketPermission madli.ut.ee resolve)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkConnect(SecurityManager.java:1042)
        at java.net.InetAddress.getAllByName0(InetAddress.java:559)
        at java.net.InetAddress.getAllByName0(InetAddress.java:540)
        at java.net.InetAddress.getByName(InetAddress.java:449)
        at java.net.Socket.<init>(Socket.java:100)
        at Kell1.main(Kell1.java:9)
```

Vaikimisi turvahaldur lubab vaid lugeda faile jooksvast ning sellele alanevatest kataloogidest. Pea kõik muu, mida vähegi keelata annab on keelatud. Kui soovida koostada programmi tarvis sobivat turvataset, võib hakata algsele peaaegu nulltasemele tasapisi õigusi juurde andma. Õiguste kirjeldused tuleb panna policy-failidesse, kust neid siis virtuaalmasinal programmi käivitamise ajal lugeda lasta.

```
C:\User\jaagup\0104\turva>type vork.policy
grant{
  permission java.net.SocketPermission "*.ut.ee", "connect";
};
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vork.policy Kell1
Wed Apr  4 20:21:15 2001
```

Käskluse grant abil määratakse, millises kohas (kataloogis, failis, serveris) asuvatele programmidele õigused kehtivad. Kui grant'i järel tulevad kohe loogilised sulud, siis kehtivad õigused kõikidele käivitavatele programmidele sõltumata asukohast. Kui ühendatava masina aadressis osa tärniga märkida, siis pääseb ligi kõikidele kirjeldusele vastavatele masinatele. Täpsemaks minnes võib aga piirduda ka ühe masina väratipiirkonna või sootuks ühe väratiga, kui soovime hoolitseda, et programm tõesti ainult vaid meile tuntud ja teatud kohaga piirdub.

```
C:\User\jaagup\0104\turva>type vork2.policy
grant{
  permission java.net.SocketPermission "madli.ut.ee:10-15", "connect";
};
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vork2.policy Kell1
Wed Apr  4 20:21:54 2001
```

Turvahalduri võib seada ka käsuga System.setSecurityManager. Kui eelmisel juhul pidi programmi käivitaja hoolitsema (lipuga -Djava.security.manager), et tööle lastud elukas midagi liialt kahtlast teha ei saaks, siis nii paneb programm seestpoolt omale ise päitsed pähe.

```
C:\User\jaagup\0104\turva>type Turva5.java
import java.io.*;
public class Turva5{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new SecurityManager());
    PrintWriter valja=new PrintWriter(new FileWriter("nimed.txt", true));
    valja.println("Siim");
    valja.close();
  }
}
```

Et vaikimisi õiguste juures faili kirjutamist ei lubata, siis antakse käivitamisel veateade.

```
C:\User\jaagup\0104\turva>java Turva5
Exception in thread "main" java.security.AccessControlException: access denied (
java.io.FilePermission nimed.txt write)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkWrite(SecurityManager.java:978)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:96)
        at java.io.FileWriter.<init>(FileWriter.java:52)
        at Turva5.main(Turva5.java:5)
```

Kui anda ennast kinni talitsenud programmile veidi õigusi, siis saab ta oma etteantud ülesannetega ilusti hakkama. Lisati õigus kirjutada kohaliku kataloogis faili nimed.txt ning võimegi näha kuidas pärast programmi töö lõppu seal Siimu nimi ilutseb.

```
C:\User\jaagup\0104\turva>type nimed.policy
grant{
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed.policy Turva5

C:\User\jaagup\0104\turva>type nimed.txt
Siim
```

Nagu ennist mainitud, võib codeBase abil määrata, kus asuvatele programmidele määratavad õigused laienevad. Nii on õigused kataloogile ning programm töötab.

```
C:\User\jaagup\0104\turva>type nimed_a.policy
grant codeBase "file:/C:/user/jaagup/0104/turva/" {
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_a.policy Turva5
```

Nõnda võivad faili "nimed.txt" kirjutada kõik programmid, mis asuvad kataloogis C:/user/jaagup/0104/ või sellest alanevates kataloogides. Kuna file:/C:/user/jaagup/0104/turva/Turva5 vastab sellele tingimusele, tuleb nime kirjutamine välja.

```
C:\User\jaagup\0104\turva>type nimed_b.policy
grant codeBase "file:/C:/user/jaagup/0104/-" {
 permission java.io.FilePermission "nimed.txt", "write";
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_b.policy Turva5
```

Kui õiguste kehtivuse piirkonnaks on aga vaid programmi ülemkataloog

```
C:\User\jaagup\0104\turva>type nimed_c.policy
grant codeBase "file:/C:/user/jaagup/0104/" {
 permission java.io.FilePermission "nimed.txt", "write";
};
```

, siis faili kirjutamine ei õnnestu.

```
C:\User\jaagup\0104\turva>java -Djava.security.policy=nimed_c.policy Turva5
Exception in thread "main" java.security.AccessControlException: access denied (
java.io.FilePermission nimed.txt write)
        at java.security.AccessControlContext.checkPermission(AccessControlConte
xt.java:272)
        at java.security.AccessController.checkPermission(AccessController.java:
399)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
        at java.lang.SecurityManager.checkWrite(SecurityManager.java:978)
        at java.io.FileOutputStream.<init>(FileOutputStream.java:96)
        at java.io.FileWriter.<init>(FileWriter.java:52)
        at Turva5.main(Turva5.java:5)
```

Omaloodud turvahaldur

Klassi SecurityManager võib laiendada ning seal olevaid meetodeid üle katta. Kõikide turvakontrollide puhul käivitatakse meetod checkPermission ning parameetrina antakse toiming, mille teostamiseks luba küsitakse. Juhul kui turvahaldur leiab, et küsitud toimingu tarvis ei tohi luba anda, heidetakse meetodist välja erind SecurityException. Allloodud turvahalduri laiendaja Lubaja1 on eriti sõbralik: siin vaid trükitakse välja, mille kohta õigust sooviti saada ning mingeid toiminguid ei piirata.

```
C:\User\jaagup\0104\turva>type Lubaja1.java
import java.security.*;
class Lubaja1 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
  }
}

C:\User\jaagup\0104\turva>type Turva5b.java
import java.io.*;
public class Turva5b{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new Lubaja1());
    PrintWriter valja=new PrintWriter(new FileWriter("nimed.txt", true));
    valja.println("Siim");
    valja.close();
  }
}
```

Väljatrükist on näta, et õigusi küsiti neljal korral. Lisaks faili kirjutamise õigusele käivad standardosa programmid küsimas õigusi ka jooksva kataloogi nime teada saamiseks ning uurivad enne järele, kas rea vahetamiseks vajalikku sümbolit tohib pärida.

```
C:\User\jaagup\0104\turva>java Turva5b
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Soovides teada, mida algne turvahaldur koos policy-failidest saadud vihjetega peab vajalikuks keelata, tuleb välja kutsuda ülemklassi samanimeline meetod ehk super.checkPermission. Et sealtkaudu tekkivad erindid siin näites programmi tööd ei katkestaks, selleks on käsule püünis ümber pandud ning trükitakse välja, milliseid toiminguid algne haldur ei luba.

```
C:\User\jaagup\0104\turva>type Lubaja2.java
import java.security.*;
class Lubaja2 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
    try{
      super.checkPermission(p);
    } catch(Exception e){
      System.out.println("Probleem: "+e);
    }
  }
}
C:\User\jaagup\0104\turva>java Turva5c
```

```
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
Probleem: java.security.AccessControlException: access denied (java.util.Propert
yPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
Probleem: java.security.AccessControlException: access denied (java.util.Propert
yPermission user.dir read)
(java.io.FilePermission nimed.txt write)
Probleem: java.security.AccessControlException: access denied (java.io.FilePermi
ssion nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Virtuaalmasina turvahalduri väga leplikuks muutumiseks tuleb määrata õiguseks java.security.AllPermission. Nii võib mõnest paigast või ka igalt poolt pärit programmidel lubada kõike ette võtta.

```
C:\User\jaagup\0104\turva>type koiklubatud.policy
grant{
  permission java.security.AllPermission;
};

C:\User\jaagup\0104\turva>java -Djava.security.policy=koiklubatud.policy Turva5c

(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed.txt write)
(java.util.PropertyPermission line.separator read)
```

Turvahalduri õiguste määramine

Ise lubades ja keelates tuleb meetodi ülekatmise juures uurida, millist tüüpi õigusesoov parameetrina anti ning vastavalt sellele reageerida sarnaselt nagu algnegi turvahaldur seda teeb. Kui tundub, et programm küsib liialt suuri õigusi, tuleb välja heita SecurityException ning soovitavalt konstruktori parameetrina antava teatega seletada mille vastu eksiti või millised tegevused lubatud on.

```
C:\User\jaagup\0104\turva>type Lubaja3.java
import java.security.*;
import java.io.FilePermission;
class Lubaja3 extends SecurityManager{
  public void checkPermission(Permission p){
    System.out.println(p);
    if(p instanceof FilePermission){
      if(!p.getName().toLowerCase().startsWith("t")){
        throw new SecurityException("Lugemiseks lubatud vaid t-ga algavad failinimed");
      }
    }
  }
}
```

Vaikimisi paigutushalduri puhul on kohaliku kataloogi failidest lugemine lubatud.

```
C:\User\jaagup\0104\turva>type Turva4a.java
import java.io.*;
public class Turva4a{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new SecurityManager());
    BufferedReader sisse=new BufferedReader(new FileReader("nimed2.txt"));
    System.out.println(sisse.readLine());
  }
}
```

Käivitamisel väljastab programm

```
C:\User\jaagup\0104\turva>java Turva4a
Katrin
```

, mis on ka loogiline, sest tekstifaili esimesel real asus nimi Katrin.

```
C:\User\jaagup\0104\turva>type nimed2.txt
Katrin
Kati
Kai
```

Kui aga määrata turvahalduriks isend, kes lubab tegelda vaid t-ga algavate failinimedega,

```
C:\User\jaagup\0104\turva>type Turva4c.java
import java.io.*;
public class Turva4c{
  public static void main(String argumendid[]) throws IOException{
    System.setSecurityManager(new Lubaja3());
    BufferedReader sisse=new BufferedReader(new FileReader(argumendid[0]));
    System.out.println(sisse.readLine());
  }
}
```

siis antakse faili lugemiseks avamisel veateade.

```
C:\User\jaagup\0104\turva>java Turva4c nimed2.txt
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission nimed2.txt read)
Exception in thread "main" java.lang.SecurityException: Lugemiseks lubatud vaid
t-ga algavad failinimed
        at Lubaja3.checkPermission(Lubaja3.java:8)
        at java.lang.SecurityManager.checkRead(SecurityManager.java:890)
        at java.io.FileInputStream.<init>(FileInputStream.java:61)
        at java.io.FileReader.<init>(FileReader.java:38)
        at Turva4c.main(Turva4c.java:5)
```

Kui aga lugeda sama sisuga faili, mille nimi algab t-ga, siis lugemine õnnestub. Kõigepealt kirjutab turvahaldur välja, mille kohta luba küsiti ning lõpuks on ilusasti näha nimi, mis leiti faili esimeset realt.

```
C:\User\jaagup\0104\turva>java Turva4c tydrukud.txt
(java.util.PropertyPermission sun.net.inetaddr.ttl read)
(java.util.PropertyPermission user.dir read)
(java.io.FilePermission tydrukud.txt read)
Katrin
```

Hoiatusribaga aken

Terasemal jälgimisel olete võinud märgata, et rakendi poolt avatud akende allservas on pea alati kirjas Java Applet Window, kohapeal käivitatud programmi akende puhul aga sellist riba ei leia. Riba on mõeldud turvahoiatusena, et kasutaja teaks arvestada salapärase veebilehelt avanenud aknaga, mis muidu sarnaneb kõigi teiste akendega ning võib ennast maskeerida suisa kohaliku parooli küsiva sisestusakna sarnaseks, kuid võib saabunud andmed ilma pikema jututa saata veebilehe omanikule kel edaspidi siis nende üle vaba voli on. Riba ei teki raami alla mitte mingi ime läbi vaid virtuaalmasinas on nii korraldatud, et ribata raami saamiseks peab eraldi luba olema. Tavalistel käivitatavatel programmidel on selline luba olemas kuid rakenditel ning uue vaikimisi turvahalduriga programmidel sellist õigust pole ning seetõttu surutakse nende poolt avatud raamidele kasutajat hoiatav tempel külge.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager Raam1

C:\User\jaagup\0104\turva>type Raam1.java
import java.awt.Frame;
public class Raam1{
  public static void main(String argumendid[]){
    Frame f=new Frame("Iseseisev raam");
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```

Et omaloodud või määratud turvahalduri korral raami all olevast hoiatusribast vabaneda, tuleb ribast vabanemise õigus programmile juurde anda.

```
C:\User\jaagup\0104\turva>type vabaraam.policy
grant {
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
};
```

ning siis näeme tavalist akent nagu ikka harjunud nägema oleme.

```
C:\User\jaagup\0104\turva>java -Djava.security.manager
-Djava.security.policy=vabaraam.policy Raam1
```

Valikuline õiguste loetelu

java.io.FilePermission	Asukoht, näit. /home/jaagup file://C:/temp/ file://C:/temp/- file://C:/temp/koolid.txt	Tegevus read, write, execute,delete
java.net.SocketPermission	masin[:värat] lin2.tpu.ee:79 *.ut.ee madli.ut.ee:7-80 * lin2.tpu.ee:80-	accept, connect, listen, resolve
java.util.PropertyPermission	* java.home java.*	read, write
java.lang.RuntimePermission	createClassLoader setSecurityManager exitVM setIO stopThread	
java.awt.AWTPermission	showWindowWithoutWarningBanner accessClipboard accessEventQueue listenToAllAWTEvents readDisplayPixels	
java.security.AllPermission		

Atribuudid (Properties)

Alati ühesuguse väärtuse (näiteks terve koera jalgade arv) saab kirjutada otse koodi sisse, kuigi pea alati oleks see viisakam koodi loetavuse huvides kusagil konstandina kirja panna ning edasi vastava nime järgi väärtust kasutada. Kui väärtus võib programmi käigus muutuda, siis enamasti võib vastava suuruse meelespidamiseks võtta muutuja ning sinna siis vajalikul hetkel andmeid talletada või sealt lugeda. Kui salvestatavate parameetrite arv pole kindlalt teada, siis on mõistlikum kasutada muutujate asemel muid vahendeid. java.util.Hashtable sisaldab võtmete ja väärtuste paare nagu ka muud liidest Map realiseerivad klassid. Võtmeid võib alati juurde lisada ning võtme järgi saab väärtuse kätte. Väärtused võivad korduda, võtmed aga mitte. Nii on kindlalt määratud, et igale võtmele vastab korraga vaid üks väärtus. Sõnede tarvis on loodud Hashtable alamklass java.util.Properties, kus saab stringe teineteisega vastavusse seada, vastavusi küsida, faili talletatada, voogu pidi edasi kanda või taas voost kasutamiseks välja lugeda.

```
import java.util.Properties;
public class Atribuudid1{
  public static void main(String argumendid[]){
    Properties p1=new Properties();
    p1.setProperty("kokk","Maali");
    p1.setProperty("kassapidaja", "Juuli");
    System.out.println("Täna keedab suppi "+p1.getProperty("kokk"));
  }
}
```

Väljundina teatatakse:
Täna keedab suppi Maali
, sest küsitud parameetrile (võtmele) kokk vastas nimi Maali. Kui polnuks kokka määratud ega teda ka kusagilt mujalt teada saadud, siis väljastataks tühiväärtus null.

Kui tahta andmeid failis säilitada ning hiljem tarvitada, siis tuleb määrata (faili)voog andmete saatmiseks, andmeid kirjeldav pealkiri ning käsuga store saata andmed tekstina teele.

```
import java.io.*;
import java.util.Properties;
public class Atribuudid2{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=new Properties();
    p1.setProperty("kokk","Maali");
    p1.setProperty("kassapidaja", "Krõõt");
    p1.store(new FileOutputStream("toitjad.properties"), "Toitev personal");
  }
}
```

Andmed jõudsid ilusti faili. Trellidega algav rida loetakse kommentaariks ning seda uuel sisselugemisel ei arvestata. ASCII sümbolid kuni 127-ni salvestatakse nii nagu nad on, ülejäänute puhul kasutatakse Unicode sümbolite salvestamiseks tehtud kuju, kus sümbol algab \u-ga ning järgnevad neli märki tähistavad sümbolile vastavat arvu kuueteistkümnendsüsteemis. Soovides andmeid meile tuttava kooditabeli järgi lugeda, tuleks kirjutada native2ascii toitjad.properties, mis loeb faili ning väljatrükil teisendab sümbolid võimaluse korral loetavamale kujule. Kui soovida sellist andmefaili parandada ning ei jõua pidevalt mõtelda tähekoodide peale, siis võib teksti rahumeeli täpitähti kasutades valmis kirjutada ning hiljem kirjutada native2ascii -reverse failinimi > muudetud_faili_nimi , tulemusena peaksid sinna jõudma suuremate koodinumbritega väärtused \uxxxx kujul, kust virtuaalmasin oma vahenditega taas andmeid lugeda suudab.

```
#Toitev personal
#Mon Apr 09 10:48:12 GMT+03:00 2001
kokk=Maali
kassapidaja=Kr\u00F5\u00F5t
```

Failist või mujalt voost loetakse andmed sisse käsuga load. Edasi võib juba atribuutide objekti kasutada nagu ennistki. Kui me pole kindlad, kas meie küsitud võti ja väärtus andmete hulgas leidub, siis võib käsklusele getProperty anda kaks argumenti: võtme ning vaikeväärtuse. Niimoodi ei väljastata väärtuse puudumise korral tühiväärtust null vaid võetakse vaikeväärtus-

```
import java.io.*;
import java.util.Properties;
public class Atribuudid3{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=new Properties();
    p1.load(new FileInputStream("toitjad.properties"));
    System.out.println("Täna keedab suppi "+p1.getProperty("kokk"));
    System.out.println("Veevärki hoiab korras "+
                                p1.getProperty("remondimees", "Ivan"));
  }
}
```

Väljund:

```
Täna keedab suppi Maali
Veevärki hoiab korras Ivan
```

Kui andmete hulka lisada ka remondimehe nimi, siis kuvatakse see ka väljundis sellisena nagu see failis kirjas on:

```
#Toitev personal
#Mon Apr 09 10:48:12 GMT+03:00 2001
kokk=Maali
kassapidaja=Kr\u00F5\u00F5t
```

```
remondimees=Maksim

Väljund
Täna keedab suppi Maali
Veevärki hoiab korras Maksim
```

Klassi Properties kasutab ka virtuaalmasin oma kasutatavate parameetrite hoidmiseks. Milliste parameetrite ja väärtustega on tegemist, selle saab järele uurida klassi System käsuga getProperties. Kui turvahaldur lubab, võib ükshaaval väärtusi lugeda ja muuta, samuti käsuga list soovitud voogu välja trükkida.

```
import java.io.*;
import java.util.Properties;
public class Atribuudid4{
  public static void main(String argumendid[]) throws IOException{
    Properties p1=System.getProperties();
    p1.list(System.out);
  }
}
```

Süsteemi juurde kuuluvaid parameetreid saab ka käsurealt määrata. Piisab vaid ette kirjutada -D ning võti=väärtus (appletvieweri puhul -J-Dvõti=väärtus) kui juba ongi meie antud andmed atribuudiloendis kirjas. Nagu alljärgnevast loetelust näha võib, asuvad käsurealt antud väärtused võrdsetena nende kõrval, mida virtuaalmasin oma südamest pidas sobivaks avaldada. Kala tuli sisse uuena, kasutaja endine nimi jaagup aga muudeti jukuks.

```
C:\User\jaagup\0104\k1>java -Dkala=angerjas -Duser.name=juku Atribuudid4
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=C:\PROGRAM FILES\JDK130\JRE\bin
java.vm.version=1.3.0-C
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\User\jaagup\0104\k1
java.runtime.version=1.3.0-C
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
os.arch=x86
java.io.tmpdir=C:\WINDOWS\TEMP\
line.separator=

java.vm.specification.vendor=Sun Microsystems Inc.
java.awt.fonts=
os.name=Windows 95
java.library.path=C:\PROGRAM FILES\JDK130\BIN;.;C:\WIND...
kala=angerjas
java.specification.name=Java Platform API Specification
java.class.version=47.0
os.version=4.0
user.home=C:\WINDOWS
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding=Cp1257
java.specification.version=1.3
user.name=juku
java.class.path=.
java.vm.specification.version=1.0
java.home=C:\PROGRAM FILES\JDK130\JRE
user.language=et
java.specification.vendor=Sun Microsystems Inc.
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode
java.version=1.3.0
java.ext.dirs=C:\PROGRAM FILES\JDK130\JRE\lib\ext
sun.boot.class.path=C:\PROGRAM FILES\JDK130\JRE\lib\rt.ja...
java.vendor=Sun Microsystems Inc.
file.separator=\
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport...
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
user.region=EE
sun.cpu.isalist=pentium i486 i386
```

Krüptimine

Andmeid võõrastele nähtamatuks muuta on püütud juba aastatuhandeid. Algselt tegeldi põhiliselt tähtede vahetamise ning järjekorra muutmisega. Arvutiajastul leitakse, et sobivam on kodeerida bitijada.

Üks plokk

Krüptimisalgoritme on mitmesuguseid. Märgatav osa neist kodeerivad andmeid plokkide kaupa. DES-i nimelisel algoritmil on ploki suuruseks 8 baiti, mis tähendab, et kodeeritavad andmete baitide arv peab jaguma kaheksaga. Nii andmed (avatekst) kui võti on bitijada kujul. Võtme pikkus on ühekordses DESis alati 8 baiti, andmete maht pole aga piiratud. Siin näites on andmeteks määratud plokk, kus kõik bitid on nullid ning võtmeks plokk, kus kõik bitid ühed.

Enne kodeerima asumist tuleb luua Javale kasutataval kujul võtmeobjekt (SecretKey). Edasi luua ja initsialiseerida kodeerija, määrates kasutatava algoritmi.Edasi tuleb hakata andmeid kodeerijast läbi laskma. Lühemate andmete puhul nagu siin, võib kohe anda käskluse doFinal ja saada lõpptulemuse kätte. Kui aga andmed ulatuvad megabaitidesse või gigabaitidesse ning nende korraga mällu lugemine pole mõistlik ega võimalik, siis on õnnestub šifreerida ka väiksemate osade kaupa ning ühelt poolt andmeid sisestada ja teiselt poolt väljastada ja talletada.

Viimane for-tsükkel loodi vaid tulemuste vaatamiseks, kõikide baitide väärtused trükitakse välja kuueteistkümnendsüsteemis.

```
import javax.crypto.*;
import javax.crypto.spec.*;
public class DESKrypt{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid=new byte[8];
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      avatekstibaidid[i]=(byte)0x00;
      votmebaidid[i]=(byte)0xFF;
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti);
    byte[] salatekst=kodeerija.doFinal(avatekstibaidid);
    for(int i=0; i<salatekst.length; i++){
      System.out.print(Integer.toString(salatekst[i] & 0xFF, 16));
    }
  }
}
```

Ning programmi töö tulemuseks siis väljatrükitud krüptogramm.

```
D:\Home\jaagup\rak>java DESKrypt
caaaaf4deaf1dbae6aec3f62f460a08d
```

Šifreereeritud voog

Et Javas õnnestub enamike andmetega voogude abil suhelda, siis on ka krüptimise tarbeks voog loodud. Sest kui üks voog suudab võtta vastu objekte, teine neid kokku pakkida ja kolmas faili kirjutada ning kõik need vood võib omavahel kokku liita, siis on ju mugav ka luua voog kokku- ja lahtikrüptimiseks, et vajadusel selline ühenduslüli lihtsalt ahelasse juurde panna. Seda ülesannet täidab CipherOutputStream. Nii nagu plokkidegi kaupa krüptides, tuleb ka siin võti luua ja initsialiseerida. Vaid andmete sisestamine ja edastamine sarnaneb voogudega seotud lähenemisele, kus write-käsu abil baidimassiivist andmeid saata võib.Ning et andmed ilusti kättesaadavad püsiksid, selleks saadab loodud krüptiv voog oma andmed FileOutputStream'i abil DES2.dat-nimelisse faili.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt2{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid="Juku tuli kooli.  ".getBytes();
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      votmebaidid[i]=(byte)0xFF;
```

```
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti);
    CipherOutputStream salavaljundvoog=new CipherOutputStream(
      new FileOutputStream("DES2.dat"), kodeerija
    );
    salavaljundvoog.write(avatekstibaidid);
    salavaljundvoog.close();
  }
}
```

Kui programm käima pandud

```
D:\Home\jaagup\rak>java DESKrypt2
```

siis tekib jooksvasse kataloogi vastava nimega fail, praegusel juhul 24 baidi pikkune, ehk sama pikk kui sisendandmed.

```
D:\Home\jaagup\rak>dir des2.dat
 Volume in drive D is DATA
 Volume Serial Number is 3842-03F3

 Directory of D:\Home\jaagup\rak

19.03.2004  18:13                24 DES2.dat
               1 File(s)             24 bytes
               0 Dir(s)   9 682 993 152 bytes free
```

Püüdes faili sisu tekstina vaadata, väljub sealt vaid hulk suhteliselt seosetuid sümboleid ning algset teksti sealt ilma võtit teadmata peaks olema küllalt lootusetu kätte saada

```
D:\Home\jaagup\rak>more DES2.dat
;├[↕q|Ñ}]Ax-s÷:Rw┼ÇП╬Ħ
```

Kui aga võti teada, siis võib kokku panna sarnase sisendvoo ning soovitud teate taas välja lugeda. Kõik muu näeb välja samasugune, vaid kodeerija ja voo suunad on vastupidised. Andmed loetakse baidimassiivi ning sealt omakorda muudetakse selgema väljatrüki huvides tekstiks.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt2a{
  public static void main(String argumendid[]) throws Exception{
    byte[] votmebaidid=new byte[8];
    for(int i=0; i<8; i++){
      votmebaidid[i]=(byte)0xFF;
    }
    SecretKey voti=SecretKeyFactory.getInstance("DES").generateSecret(
      new DESKeySpec(votmebaidid)
    );
    Cipher kodeerija=Cipher.getInstance("DES");
    kodeerija.init(Cipher.DECRYPT_MODE, voti);
    CipherInputStream salasisendvoog=new CipherInputStream(
      new FileInputStream("DES2.dat"), kodeerija
    );
    byte[] b =new byte[1000];
    int pikkus=salasisendvoog.read(b);
    System.out.println(new String(b, 0, pikkus));
  }
}
```

Kui kompileeritud kood tööle panna, võib rõõmsasti algset teadet imetlema jääda. Ja kõike vaid seetõttu, et teada oli krüptimisel kasutatud võti. Praegusel juhul kaheksa baiti, kõik bitid püsti.

```
D:\Home\jaagup\rak>java DESKrypt2a
Juku tuli kooli.
```

Parooliga krüptimine

Kaheksabaidine võti on ilus lühike küll, aga kuutkümmend nelja bitti lihtsalt pähe õppida on mõnevõrra tülikas. Olgugi, et juhuslikult genereeritud väärtused peaksid kokku andma kõige raskemini ära arvatava vastava pikkusega kombinatsiooni, on inimesed harjunud meeles pidama vähemasti veidigi loogilisema ülesehitusega paroole. Siin on nõnda kokku pandud räsikoodi ja krüptimise võimalused, et parool ei pea olema kindla pikkusega.

Parooli hoidmiseks ja edastamiseks kasutatakse Java juures tavalise Stringi asemel tähemassiivi. Nõnda on võimalik paranoilisemate juhtude korral masina mälus vastavad baidid pärast kasutamise lõppu ükshaaval üle kirjutada ning pole nii suurt muret, et keegi neid mälutõmmiste abil lugema pääseks.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt3{
  public static void main(String argumendid[]) throws Exception{
    byte[] avatekstibaidid=(loeRida("Teade:")+"       ").getBytes();
    byte[] sool=new byte[8];
    for(int i=0; i<8; i++){
      sool[i]=(byte)0xFF;
    }
    int segamiskordadearv=10;
    SecretKey voti=SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(
      new PBEKeySpec(loeRida("Parool:").toCharArray())
    );
    Cipher kodeerija=Cipher.getInstance("PBEWithMD5AndDES");
    kodeerija.init(Cipher.ENCRYPT_MODE, voti, new PBEParameterSpec(sool,
segamiskordadearv));
    CipherOutputStream salavaljundvoog=new CipherOutputStream(
      new FileOutputStream("DES3.dat"), kodeerija
    );
    salavaljundvoog.write(avatekstibaidid);
    salavaljundvoog.close();
  }
  public static String loeRida(String kysimus) throws IOException{
    System.out.println(kysimus);
    return new BufferedReader(new InputStreamReader(System.in)).readLine();
  }
}
```

Käivitamisel küsitakse nii teadet kui parooli:

```
D:\Home\jaagup\rak>java DESKrypt3
Teade:
Tere tulemast
Parool:
kalake
```

Tulemuseks on krüptogramm, mis nagu nõutud, ei näe kuigivõrd eelnevate järgi välja.

```
D:\Home\jaagup\rak>more DES3.dat
ôÆjce═·σ?►Rß∞ΣêiíûΘé■ß
```

Soovides algseid andmeid kätte saada, tuleb kogu toiming tagurpidi läbi käia. Sool ja segamiskordade arv peavad kattuma. Loodud nad selleks, et ka krüptitud paroolifailide väljatuleku korral poleks liialt lihtne võrrelda ja leida ühesuguseid paroole.

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.io.*;
public class DESKrypt3a{
  public static void main(String argumendid[]) throws Exception{
    byte[] sool=new byte[8];
    for(int i=0; i<8; i++){
      sool[i]=(byte)0xFF;
    }
    int segamiskordadearv=10;
    SecretKey voti=SecretKeyFactory.getInstance("PBEWithMD5AndDES").generateSecret(
      new PBEKeySpec(DESKrypt3.loeRida("Parool:").toCharArray())
    );
    Cipher kodeerija=Cipher.getInstance("PBEWithMD5AndDES");
    kodeerija.init(Cipher.DECRYPT_MODE, voti,
                   new PBEParameterSpec(sool, segamiskordadearv));
    CipherInputStream salasisendvoog=new CipherInputStream(
```

```
      new FileInputStream("DES3.dat"), kodeerija
    );
    int nr=salasisendvoog.read();
    while(nr != -1){
      System.out.print(((char)nr));
      nr=salasisendvoog.read();
    }
    System.out.println();
  }
}
```

Kui õige parool ette anda, siis võib rõõmsasti teadet lugeda.

```
D:\Home\jaagup\rak>java DESKrypt3a
Parool:
kalake
Tere tulemast
```

Juhtus aga parool ununema, tuleb selleks korraks teatest suu puhtaks pühkida.

```
D:\Home\jaagup\rak>java DESKrypt3a
Parool:
kass
&■e?k?X↔4FkNHª♥?
```

Ülesandeid

Signeerimine

```
* Koosta enesele keytool'i abil sertifikaat
* Loo lihtne programm, paki see jar-faili.
* Signeeri loodud arhiiv.
* Anna oma avalikku võtit sisaldav sertifikaat pinginaabrile
  ja võta tema oma vastu.
* Saada signeeritud arhiivifail naabrile ning kontrolli ise naabrilt
  saabunud arhiivifaili õigsust.
* Kontrolli, et signeerimata või tundmata võtmega signeeritud arhiivi
  puhul antaks veateade.
```

Krüptograafia

```
* Tutvu näidetega
* Sifreeri arvude 0-15 ruute tähistavad baidid (0, 1, 4, 9, 16 ...
225) tähistavad baidid DES-i abil võtmega, mille 8 baiti on väärtused
alates  kahest (2, 4, 6 ...).
* Kontrolli tulemust desifreerimise abil.
* Sama võtmega krüpti kasutaja poolt määratud failis asuvad andmed.
* Loo failide krüptimiseks/dekrüptimiseks graafiline liides. Parool
ning  failinimi küsi kasutaja käest.
```

Digitaalallkiri

```
* Tutvu näidetega.
* Loo enesele võtmepaar.
* Signeeri tekst salajase võtmega
* Kontrolli allkirja sobivust.
* Loo graafiline liides, kus saab luua võtmepaari, signeerimisel ning
  kontrollimisel valida võtmefaili , allkirjafaili ja andmefaili.

* Lisa vahend sõnumilühendi loomiseks ning etteantud faili baitide
  esitamiseks kuueteistkümnendsüsteemis.
```

Hajusrakendused

Protokollid, liidesed, sünkroniseerimine, J2EE, EJB

RMI

Rakenduste suuremaks kasvamisel ei pruugita enam kõiki toiminguid ühes masinas teha. Tüüpiliselt võib eraldi paikneda näiteks andmebaas. Kuid ka ressursinõudlikumad arvutused või eripärast riistvara nõudvad toimingud paigutatakse suuremate süsteemide puhul eraldi masinatesse. Lihtsamaks ja levinumaks ühenduseks selliste masinate vahel on TCP või UDP põhine protokoll, kus programmeerijale teadaoleval kujul andmed ühes või teises suunas saadetakse. Selline süsteem on platvormist küllaltki sõltumatu ning andmeid kannatab vahetada mitmesuguste operatsioonisüsteemide ja programmeerimiskeelte vahel. Kui on tegemist üksiku lihtsakoelise käsklusega - nagu näiteks kellaaja küsimine - siis võib taolise teenusega täiesti leppida. Keerulisematel juhtudel aga võib programmeerijal mugavam olla kasutada samamoodi objekte ja käsklusi nii nagu muu programmiosa väljakutsetel. Ning jätta tüübiteisendused ja erinditöötluse mõne muu valmistüki hooleks. Üht sellist abikomplekti tuntaksegi RMI nime all.

Lihtsaim näide

Ka lühima näite töölesaamiseks tuleb üldjuhul luua vähemasti neli faili. Liideses kirjeldatakse, millised käsud võrgu kaudu väljajagamisele lähevad. Liidesed võiksid olla kõige püsivamad. Ehk kui kord on käsklus kokku lepitud, siis võiksid liideses paiknevat käsku mujal kasutavad programmeerijad loota, et vastavat käsku ka tulevikus kasutada saab. Paketina tasub importida java.rmi.*; selle kaudu saab kätte nii RMI kaudu teenuse pakkumiseks vajaliku ülemliidese Remote kui ühendusprobleemidest teatava RemoteExceptioni. Kõik väljajagatavad käsklused tuleks liidesesse riburadapidi kirja panna.

```
import java.rmi.*;
public interface Arvutaja extends Remote{
  public int liida(int a, int b) throws RemoteException;
}
```

Nagu liideste puhul ikka, vajatakse tegelikuks realisatsiooniks klassi, kus toiming masinale arusaadavas keeles lähemalt kirjas. Siin liitmise puhul pääsetakse lihtsalt, sest näide meelega lihtne tehtud. Muul juhul aga kandub programmeerimise märgatav raskus just siia, et kõik soovitu võimalikult hästi kirja saaks pandud. Samas - välispoolsete komponentide tarbeks on tähtsamad just liidesed, sest nende käskluste nimedest ja parameetrite väärtustest sõltuvad muud programmikoodid. Kui realiseerivas klassis leitakse, et sama ülesanne suudetakse mõne muu algoritmiga paremini lahendada, siis võib liidest realiseeriva klassi sisu suhteliselt väiksema vaevaga vajadusel välja vahetada.

```
public class Koolilaps implements Arvutaja{
  public int liida(int a, int b){
    return a+b;
  }
}
```

Nagu võrguprogrammide puhul enamasti, saab ka siin eristada serveri- ja kliendipoolt. Esimese ülesandeks teenus välja pakkuda ning teisel võimalik väljajagatud teenust kasutama tulla. Võimalikult lühidalt väljendades saab tolle väljajagamise vaid ühte lausesse paigutada. Tuleb luua realiseerivast objektist eksemplar ning see miski nime all võrku välja jagada. 1099 on levinud värati number RMI teenuste jaoks. Eeldatakse, et jooksvas masinas on käivitatud RMI register ning sinna seotakse praegu nime alla "arvutamine" loodud Koolilapse eksemplar.

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class RMIServer1{
  public static void main(String argumendid[]) throws Exception{
    LocateRegistry.getRegistry("localhost", 1099).
     rebind("arvutamine", UnicastRemoteObject.exportObject(new Koolilaps()));
  }
}
```

Kliendi poolel tuleb kõigepealt küsida ligipääs kaugel asuvale objektile ning siis saab selle objekti käsklusi käivitada. Kliendi juures piisab liidesetüübi arvestamisest, tegeliku realiseeriva klassi ülesehitus on suurelt jaolt serveri siseasi.

```
import java.rmi.*;
public class RMIKlient1{
  public static void main(String argumendid[]) throws Exception{
    Arvutaja a=(Arvutaja)Naming.lookup("arvutamine");
    System.out.println(a.liida(3, 2));
  }
}
```

Käivitamise juhend

Kopeeri failid Arvutaja.java, Koolilaps.java, RMIServer1.java ning RMIKlient1.java ühte kataloogi. Kompileeri failid

Kirjuta

```
rmic Koolilaps
```

Selle tulemusena luuakse klassid Koolilaps_Stub.class ning Koolilaps_Skel.class andmaks nimehaldurile teavet liideste tööst.

Sama kataloogi ühte aknasse käivita

```
rmiregistry
```

teise aknasse

```
java RMIServer1
```

ning kolmandasse

```
java RMIKlient1
```

Tulemusena peaks saama klient registri kaudu serveriga ühendust ning väljastama serveri poolt kokku arvutatud tehte tulemuse.

Seiskumisvõimeline server

Esimene RMI serverprogramm oli lihtsuse huvides koostatud võimalikult lühike. Et tegelikud andmetüübid välja paistaksid ning serveri tööd ka viisakamal moel lõpetada saaks, selleks koostati järgnev näide. Nagu lõpus näha, tuleb mälu vabaksandmiseks nii objekt registrist maha võtta kui anda ka käsklus unexportObject.

```
    register.unbind("arvutamine");
    UnicastRemoteObject.unexportObject(juku, true);
```

Serverprogramm tervikuna.

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.io.*;
public class RMIServer2{
  public static void main(String argumendid[]) throws Exception{
    Koolilaps juku=new Koolilaps();
    Arvutaja kooliesindaja=(Arvutaja)UnicastRemoteObject.exportObject(juku);
    Registry register=LocateRegistry.getRegistry("localhost", 1099);
    register.rebind("arvutamine", kooliesindaja);
```

```
    System.out.println("Server korras");
    new BufferedReader(new InputStreamReader(System.in)).readLine();
                                                  //oodatakse reavahetust
    register.unbind("arvutamine");
    UnicastRemoteObject.unexportObject(juku, true);
  }
}
```

Klient ei pruugi samuti piirduda sama masina pakutavate teenustega. RMI mõtegi on ju meetodeid kaugemalt välja kutsuda. Objekti otsingul tuleb ette määrata masina aadress ning objektiga seotud nimi vastavas masinas.

```
    Arvutaja a=(Arvutaja)Naming.lookup("//jaagup.cs.tpu.ee:1099/arvutamine");
```

Veel tuleb hoolitseda, et rmic-i abil loodud abifailid oleksid mõlemal pool kättesaadavad. Ning nagu siinse lõigu juures katsetati, toimis täiesti olukord, kus klient asus Haapsalus ning server Tallinnas.

```
import java.rmi.*;
public class RMIKlient2{
  public static void main(String argumendid[]) throws Exception{
    Arvutaja a=(Arvutaja)Naming.lookup("//jaagup.cs.tpu.ee:1099/arvutamine");
    System.out.println(a.liida(Integer.parseInt(argumendid[0]),
        Integer.parseInt(argumendid[1])));
  }
}
```

Nime hoidmine serveris

RMI eripäraks lihtsalt meetodi kaugväljakutsega on, et serverist ei jagata välja mitte üksikuid käsklusi, vaid terve objekt. Nõnda on võimalik jätta andmeid käskluste vahel meelde ning järgnevate käskude tulemus võib sõltuda eelnevate käskude abil seatud väärtustest. Väärtuse püsimise näiteks on koostatud lihtne liides ja klass, mille abil võimalik inimese nime serveris meeles pidada.

```
import java.rmi.*;
public interface NimeHoiuLiides extends Remote{
  public void paneNimi(String nimi) throws RemoteException;
  public String annaNimi() throws RemoteException;
}
```

```
import java.rmi.*;
public class NimeHoiuKlass implements NimeHoiuLiides{
  String nimi="Katrin";
  public void paneNimi(String uusnimi){
    nimi=uusnimi;
  }
  public String annaNimi(){
    return nimi;
  }
}
```

Server nagu ikka. Ehk siinse programmi ülesandeks on klassist eksemplar luua ning kasutamiseks välja jagada.

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class NimeHoiuServer{
  public static void main(String argumendid[]) throws Exception{
    LocateRegistry.getRegistry("localhost", 1099).rebind("nimehoidla",
                    UnicastRemoteObject.exportObject(new NimeHoiuKlass()));
  }
}
```

Klient juba suhtleb nii kasutajaga kui väljajagatud objektiga. Ning kui kasutaja juhtus nime sisestama, siis jäetakse see ootele ja nähtavale, kuni mõni muu klient taas nime ära asendab.

```
import java.rmi.*;
import java.io.*;
public class NimeHoiuKlient{
  public static void main(String argumendid[]) throws Exception{
    NimeHoiuLiides n=(NimeHoiuLiides)Naming.lookup("nimehoidla");
    System.out.println("Viimati loeti parimaks "+n.annaNimi()+
      ". Keda sina soovitad?");
    String nimi=new BufferedReader(new InputStreamReader(System.in)).readLine();
    if(nimi.length()>0)n.paneNimi(nimi);
  }
}
```

Tagasisidega ühendus

Siiamaani toiminud näited põhinesid kliendi aktiivsusel: kliendi poolt käivitati meetodeid ning saadi vastuseid. Serveri ülesandeks oli vaid oodata ja vaadata mis toimub ning serverisse saadetud palvetele reageerida. Kui klient soovis serveris toimuvatest muutustest teada, siis tuli tal iga natukese aja tagant uurimas käia, et kas olukord on muutunud. Sellised "küsimaskäimised" on programmide juures küllalt levinud, sest mõnigikord on teistpidiste väljakutsete korraldamine küllalt tülikas. Näiteks POP3-protokolli järgi kirju lugedes saab samuti uuest kirjast teada alles siis, kui klient on serverist kirjade loetelu küsinud. Kirja serverisse jõudmine seda iseenesest kliendile teada ei anna.

Siin aga on serverist välja jagatud objektile peale nime seadmise ja küsimise käskluste olemas meetod jätmaks serveri juurde meelde kliendi osuti. Nõnda on serveril võimalik nime muutusest näiteks mõne teise kliendi tegevuse tõttu kõikidele serveriga ühinenud klientidele teada anda.

```
import java.rmi.*;
public interface NimeHoiuLiides3 extends Remote{
  public void paneNimi(String nimi) throws RemoteException;
  public String annaNimi() throws RemoteException;
  public void lisaKlient(NimeHoiuKliendiLiides3 uusklient)
     throws RemoteException;
}
```

NimeHoiuKliendiLiides3 on eraldi liides ning seal kirjas käsklus, mida soovitakse tagurpidises suunas ehk serveri poolt kliendi poole käivitada.

```
public interface NimeHoiuKliendiLiides3 extends java.rmi.Remote{
  void uusNimi(String uusnimi)
    throws java.rmi.RemoteException;
}
```

Serveris toimival objektil on nüüd siis lisaks nime hoidmise kohustusele ka ühendunud klientide teavitamise kohustus. Selleks kasutatakse Vector-tüüpi kogumit, kus pole vaja programmi käivitamise algul teada tegelike ühendujate maksimumarvu.

Iga kliendi lisandumisel paigutatakse tema andmed vektorisse.

```
  Vector kliendid=new Vector();
  public void lisaKlient(NimeHoiuKliendiLiides3 uusklient){
    kliendid.add(uusklient);
  }
```

Igal nimemuutusel käiakse läbi kõik registreeritud kliendid ning antakse kõigile teada, et serveris on hoitav nimi muutunud.

```
    for(int i=0; i<kliendid.size(); i++){
      ((NimeHoiuKliendiLiides3)kliendid.elementAt(i)).uusNimi(nimi);
    }
```

Klass tervikuna:

```
import java.rmi.*;
import java.util.Vector;
public class NimeHoiuKlass3 implements NimeHoiuLiides3{
  String nimi="Katrin";
  Vector kliendid=new Vector();
  public void paneNimi(String uusnimi) throws RemoteException{
    nimi=uusnimi;
    for(int i=0; i<kliendid.size(); i++){
      ((NimeHoiuKliendiLiides3)kliendid.elementAt(i)).uusNimi(nimi);
```

```
    }
  }
  public String annaNimi(){
    return nimi;
  }
  public void lisaKlient(NimeHoiuKliendiLiides3 uusklient){
    kliendid.add(uusklient);
  }
}
```

Server sama lihtne ja lühike nagu mujalgi

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class NimeHoiuServer3{
  public static void main(String argumendid[]) throws Exception{
    LocateRegistry.getRegistry("localhost", 1099).rebind("nimehoidla3",
                    UnicastRemoteObject.exportObject(new NimeHoiuKlass3()));
  }
}
```

Klient peab aga eelmiste näidetega võrreldes mõnevõrra põhjalikumalt kokku pandud olema. Kuna serveri pool saab meelde jätta ligipääsu terve klassi eksemplarile ja mitte lihtsalt üksikule meetodile, siis peab klient ka nõnda kokku pandud olema, et oleks võimalik klassi isendile nime muutmise teateid meetodi kaudu saata. Graafilise kliendi puhul eraldi püsivalt nähtav graafikakomponendi eksemplar on tavaline. Sarnase objekti saab aga luua ka tekstipõhise rakenduse korral. Sel juhul jääb taas main-meetodi ülesandeks vaid eksemplari loomine, ülejäänud töö tehakse juba sealsetes käskudes.

Et kliendipoolset isendit oleks võimalik serveri poolt välja kutsuda, peab ka kliendiisendi eksemplar olema exportObject-i abil avalikuks tehtud.

```
    UnicastRemoteObject.exportObject(this);
    n.lisaKlient(this);
```

Samuti tuleb töö lõpetamisel siis kliendiisend "arvelt maha" võtta.

```
    UnicastRemoteObject.unexportObject(this, true );
```

Nagu aga katsetustel paistis, jõuavad serveripoolsed teated sealsetest nimemuutustustest ilusti kohale. Edasine on juba kliendi mure, mis saadud teabega peale hakatakse.

```
  public void uusNimi(String uusnimi){
    System.out.println(uusnimi);
  }
```

Ning kliendi kood tervikuna.

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class NimeHoiuKlient3 implements NimeHoiuKliendiLiides3{
  public static void main(String argumendid[]) throws Exception{
    new NimeHoiuKlient3();
  }
  public NimeHoiuKlient3() throws Exception{
    NimeHoiuLiides3 n=(NimeHoiuLiides3)Naming.lookup("nimehoidla3");
    System.out.println("Viimati loeti parimaks "+n.annaNimi()+".");
    UnicastRemoteObject.exportObject(this);
    n.lisaKlient(this);
    BufferedReader sisse=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Keda sina soovitad?");
    String nimi=sisse.readLine();
    while(nimi.length()>0){
       n.paneNimi(nimi);
       nimi=sisse.readLine();
    }
    UnicastRemoteObject.unexportObject(this, true );
  }
  public void uusNimi(String uusnimi){
    System.out.println(uusnimi);
  }
}
```

Sünkroniseeritud lisamine ja eemaldamine

Viisakatel programmidel peaks lisaks alustamisele olema sees ka lõpetamise võimalus. Ning lisaks lisamisele ka eemaldamise võimalus. Sellisel juhul paraneb programmi jätkusuutlikus - end külge haakinud klientidel on võimalik end ka eemaldada nõnda, et server suudab nende jaoks eraldatud vahendid vabaks anda.

```
import java.rmi.*;
public interface NimeHoiuLiides3a extends Remote{
  public void paneNimi(String nimi) throws RemoteException;
  public String annaNimi() throws RemoteException;
  public void lisaKlient(NimeHoiuKliendiLiides3 uusklient)
     throws RemoteException;
  public void eemaldaKlient(NimeHoiuKliendiLiides3 klient)
      throws RemoteException;
}
```

Kui üheaegseid kasutajaid tuleb rohkem, siis on tähtis, et nad üksteise toimetusi segama ei juhtuks. Muidu võib üheaegsete toimingute puhul kergesti juhtuda, et samal ajal kui klientide jadale antakse teada serveris paikneva nime muutumisest, juhtub end mõni klient lahti ühendama. Ning tulemusena võib tekkida olukord, kus server üritab olematule kliendile teateid jagada ning tulemusena tekib veateade ning ka järgnevad kliendid jäävad teavitamata.

Kui aga hoolitseda, et ühiste andmete poole pöördumisel tehakse seda ühekaupa, siis õnnestub mitmetest üheaegse pöördumisega seotud probleemidest hoiduda. Javas on loodud synchronized-plokk, kuhu pääseb korraga vaid üks ploki päises kirjas oleva monitorobjektiga seotud lõim. Siinsetes näidetes on monitoriks klientide vektor. Et kõik klientidega seotud operatsioonid - lisamine, eemaldamine ning teadete laiali saatmine on sama monitoriga sünkroniseeritud plokkides, siis pole karta, et need toimingud üksteist häirima võiksid hakata, sest neid korraga teha ei saa. Kui aga ikkagi peaks mõne kliendi juurde andmete saatmisega muresid tekkima, siis see eemaldatakse loendist.

```
import java.rmi.*;
import java.util.Vector;
public class NimeHoiuKlass3a implements NimeHoiuLiides3a{
  String nimi="Katrin";
  Vector kliendid=new Vector();
  public void paneNimi(String uusnimi){
    nimi=uusnimi;
    synchronized(kliendid){
     for(int i=0; i<kliendid.size(); i++){
      try{
       ((NimeHoiuKliendiLiides3)kliendid.elementAt(i)).uusNimi(nimi);
      } catch(RemoteException e){
         kliendid.removeElementAt(i);
         System.out.println("Klient eemaldatud. "+e);
      }
     }
    }
  }
  public String annaNimi(){
    return nimi;
  }
  public void lisaKlient(NimeHoiuKliendiLiides3 uusklient){
   synchronized(kliendid){
    kliendid.add(uusklient);
   }
  }
  public void eemaldaKlient(NimeHoiuKliendiLiides3 klient){
   synchronized(kliendid){
    kliendid.remove(klient);
   }
  }
}
```

Serveril küljes tavapäraselt vahendid nii käivitumise kui seiskumise tarvis

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.io.*;
public class NimeHoiuServer3a{
  public static void main(String argumendid[]) throws Exception{
```

```
    NimeHoiuKlass3a yhine=new NimeHoiuKlass3a();
    LocateRegistry.getRegistry("localhost", 1099).rebind("nimehoidla3",
                    UnicastRemoteObject.exportObject(yhine));
    System.out.println("Server püsti");
    new BufferedReader(new InputStreamReader(System.in)).readLine();
    System.out.println("Hakkame maanduma");
    UnicastRemoteObject.unexportObject(yhine, true);
  }
}
```

Kliendil võrreldes eelmise näitega muu hulgas kirjas ka enese serveris paiknevast loendist maha võtmise käsklus, et server ei peaks ise avastamas käima, millise kliendiga on veel võimalik suhelda ja millisega mitte.

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class NimeHoiuKlient3a implements NimeHoiuKliendiLiides3{
  public static void main(String argumendid[]) throws Exception{
    new NimeHoiuKlient3a();
  }
  public NimeHoiuKlient3a() throws Exception{
    NimeHoiuLiides3a n=(NimeHoiuLiides3a)Naming.lookup("nimehoidla3");
    System.out.println("Viimati loeti parimaks "+n.annaNimi()+".");
    UnicastRemoteObject.exportObject(this);
    n.lisaKlient(this);
    BufferedReader sisse=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Keda sina soovitad?");
    String nimi=sisse.readLine();
    while(nimi.length()>0){
       n.paneNimi(nimi);
       nimi=sisse.readLine();
    }
    n.eemaldaKlient(this);
    UnicastRemoteObject.unexportObject(this, true );
  }
  public void uusNimi(String uusnimi){
    System.out.println(uusnimi);
  }
}
```

Ülesandeid

RMI tutvus

* Loo juhuslikke paarisarve väljastav RMI objekt. Katseta selle tööd kliendi abil.
* Luba kliendil küsida ja muuta väljajagatud objektis paiknevat arvu.

Oksjon

* Muuda kliendi kasutajaliides graafiliseks, kohanda see oksjonil osalemiseks.
* Panust saab ainult suurendada, iga kliendi ekraanil on jooksvalt näha väärtuse kasvamine.

* Lisa oksjonile administraator, kel on õigus klientidele teateid saata, pakkumine lõppenuks kuulutada ning uus ese müüki panna.
* Müüdud esemed, kliendi andmed ning hinnad jäävad kirja andmebaasi.

EJB

Sadade üheaegsete klientidega toimetulekuks kasutatakse Java-maailmas J2EE ehk Java Enterprise Editioni vahendeid. Microsofti analoogiks selle juures on COM+ ning DCOM oma võimalustega. Taolise küllalt suure ja kohmaka süsteemi kasutamine võimaldab peita programmeerija eest osa paralleelprogrammeerimisega seotud keerukusi. Samuti tuuakse märgatav osa administreerimisest programmikoodist välja. Nõnda peaks programmeerija saama paremini keskenduda rakenduse sisulise külje ja vajalike funktsioonide kokkupanekule. Tal ei ole põhjust ja vahel ka võimalust kasutajate ja ressursside haldusega seotud toiminguid oma koodi sisse kirjutada. See omakorda võimaldab koodis vähem vigu teha. Samuti õnnestub taolist koodi loodetavasti tulevikus kergemini vajadusel mujal kasutada.

Üheks osaks J2EE juures on "ärioad" ehk Enterprise Java Beans. Nende kaudu pannakse tööle keskserveris toimivad teenused, mille külge saab siis kliendiga näiteks RMI kaudu ühendust võtta. Ressursside haldus paljude kasutajate korral jääb nõnda serveri hooleks. Võrreldes eraldi töötava programmi loomisega on EJB juures nikerdamist üllatavalt palju. Väikese tervituse välja meelitamiseks tuleb koostada vähemasti neli faili. Ainsaks lohutuseks võib öelda, et rakenduse suuremaks kasvamisel koodi enam nii suures koguses ei lisandu.

Liides

Ühe liidesega tuleb määrata, mida meie loodav uba peab oskama. Liides pärineb EJBObject'ist ning iga meetod peab lubama heita RemoteExceptioni.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Arvutused extends EJBObject{
  public int liida(int a, int b) throws RemoteException;
}
```

Eraldi liideses kirjeldatakse, kuidas uba luuakse. Lihtsamal juhul piirdutakse create-nimelise meetodiga.

Koduliides

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.*;
public interface ArvutusedHome extends EJBHome{
   Arvutused create() throws RemoteException, CreateException;
}
```

Realiseeriv objekt

Nagu arvata võis, tuleb kusagil ka kirja panna, kuidas ülal deklareeritud meetod käituma peab. Ülejäänud meetodid on liidese realiseerimiseks vaja üle katta, sisu neile on põhjust lisada alles oa olekut arvestama hakates.

```
import java.rmi.RemoteException;
import javax.ejb.*;
public class ArvutusedEJB implements SessionBean{
  public int liida(int a, int b){
     return a+b;
  }
  public void ejbCreate(){}
  public void ejbRemove(){}
  public void ejbActivate(){}
  public void ejbPassivate(){}
  public void setSessionContext(SessionContext sc){}
}
```

Kompileerimine

Kui failid olemas, tuleb need kompileerida – nii nagu muudegi rakenduste puhul.

Liigun käsureal kataloogi kus failid asuvad

```
C:\>cd jaagup\java\EJBArvutus
```

Kontrollin igaks juhuks järele, et nad seal ikka olemas on

```
C:\jaagup\java\EJBArvutus>dir
 Volume in drive C has no label.
 Volume Serial Number is 8459-0195

 Directory of C:\jaagup\java\EJBArvutus

29.07.2002  14:46       <DIR>          .
29.07.2002  14:46       <DIR>          ..
29.07.2002  14:41                   169 Arvutused.java
29.07.2002  14:46                   340 ArvutusedEJB.java
29.07.2002  14:43                   199 ArvutusedHome.java
               3 File(s)            708 bytes
               2 Dir(s)   2 140 848 128 bytes free
```

Ning võibki kõik julgesti kompileerima panna.

```
C:\jaagup\java\EJBArvutus>javac *.java
```

Võrreldes tavaprogrammidega võib EJB rakenduse failide kompileerimine tunduvalt kauem aega võtta. Masinal lihtsalt palju tööd.

Keskkonna käivitus

Loodud kood ei tee iseenesest veel midagi. Peab töötama ka tema tarbeks sobiv keskkond. Avan uue käsureaakna

```
C:\jaagup\java\EJBArvutus>start cmd
```

Ning seal käivitan J2EE serveri. Käivitamine sõltub serveri tüübist. Siin on tegemist SUNi poolt vabalt kaasaantava testserveriga. Võti –verbose palub teated välja näidata, nii on kergem mõista mis toimub ning vajadusel vigu otsida

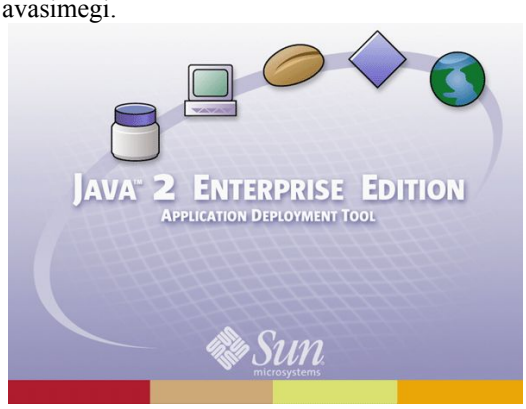
```
C:\jaagup\java\EJBArvutus>j2ee -verbose
J2EE server listen port: 1050
Naming service started:1050
Binding DataSource, name = jdbc/DB1, url = jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/DB2, url = jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/InventoryDB,
 url = jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/EstoreDB, url = jdbc:cloudscape:rmi:CloudscapeDB;
 create=true
Binding DataSource, name = jdbc/Cloudscape, url =
jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/XACloudscape, url = jdbc/XACloudscape__xa
Binding DataSource, name = jdbc/XACloudscape__xa, dataSource =
COM.cloudscape.core.RemoteXaDataSource@44cbbe
Starting JMS service...
Initialization complete - waiting for client requests
Binding: < JMS Destination : jms/Queue , javax.jms.Queue >
Binding: < JMS Destination : jms/Topic , javax.jms.Topic >
Binding: < JMS Cnx Factory : TopicConnectionFactory , Topic , No properties >
Binding: < JMS Cnx Factory : jms/TopicConnectionFactory , Topic , No properties >
Binding: < JMS Cnx Factory : jms/QueueConnectionFactory , Queue , No properties >
Binding: < JMS Cnx Factory : QueueConnectionFactory , Queue , No properties >
Starting web service at port: 8000
Starting secure web service at port: 7000
J2EE SDK/1.3.1
Starting web service at port: 9191
J2EE SDK/1.3.1
Loading jar:/c:/j2sdkee1.3.1/repository/don/applications/converterapp10274889297
71Server.jar
J2EE server startup complete.
```

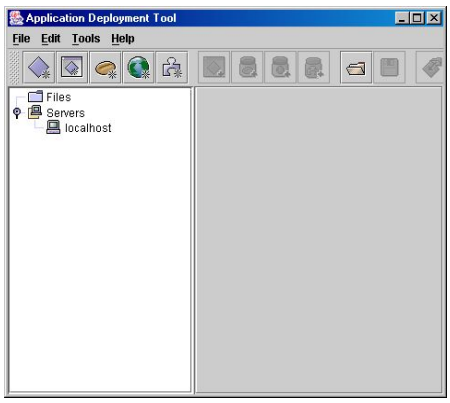
Ülespanek

Paljast serverist veel ei piisa. Loodud failidest tuleb rakendus kokku panna ning alles seejärel tööle lükata. Selle tarvis on loodud deploytool

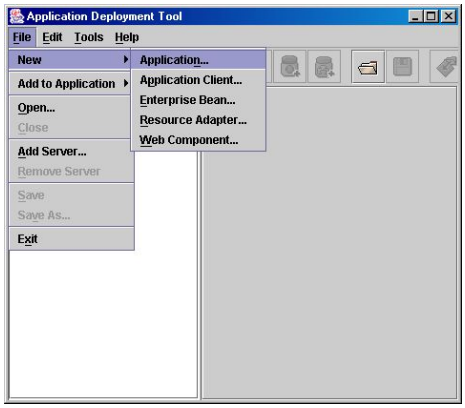
```
C:\jaagup\java\EJBArvutus>start cmd

C:\jaagup\java\EJBArvutus>deploytool
Starting Deployment tool, version 1.3.1
(Type 'deploytool -help' for command line options.)
```

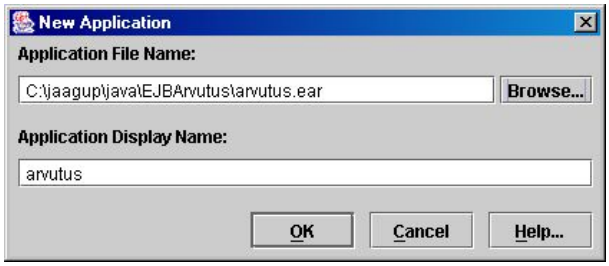
Mõningase mõttepausi järel avanebki esmalt ilus värviline pilt ning seejärel juba kasutatav deploytool. Esmasel käivitamisel pole seal küll veel töötavaid komponente, selle installeerimiseks aga me ta avasimegi.



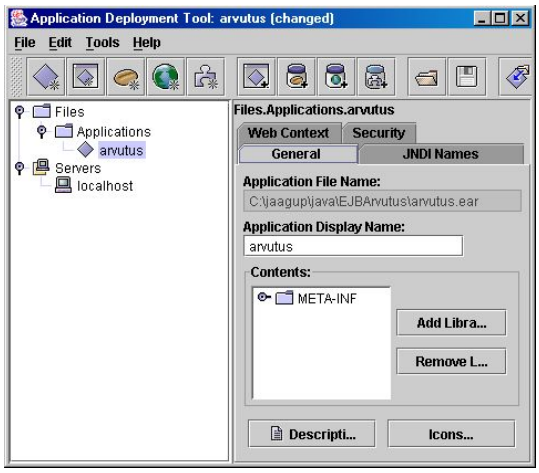


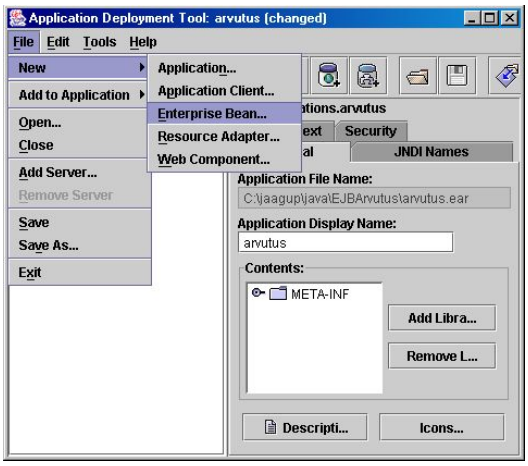
Kõigepealt tuleb luua uus rakendus, mille sisse on siis edaspidi võimalik asuda komponente lisama. Rakenduse puhul tuleb märkida fail, mis asub eneses teavet hoidma. Samuti tuleb määrata rakendusele nimi, mille kaudu teda kutsuda



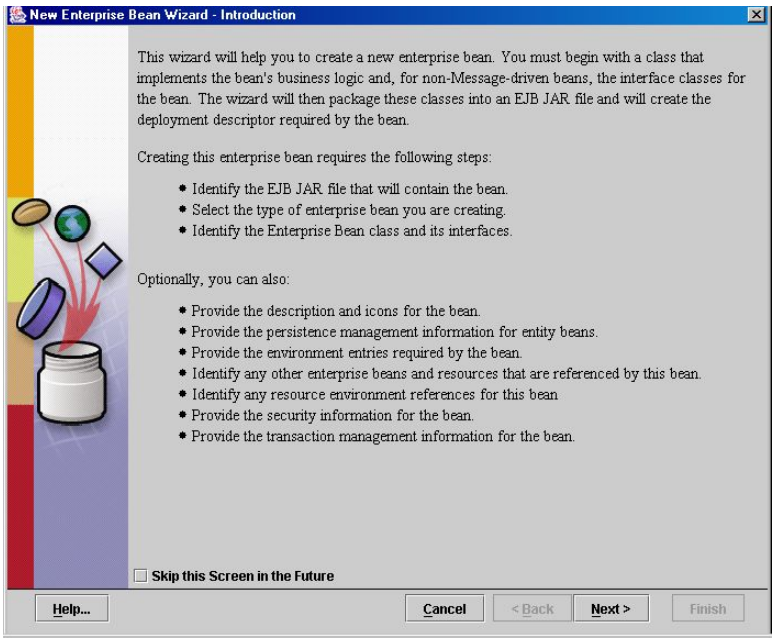


Kui tühi rakendus loodud, näeb pilt välja ligikaudu järgmine nagu vasakul. Tühjast rakendusest pole veel kasu kellelegi. Järgnevalt asume rakendusse lisama Enterprise Java Beani ehk äriuba.

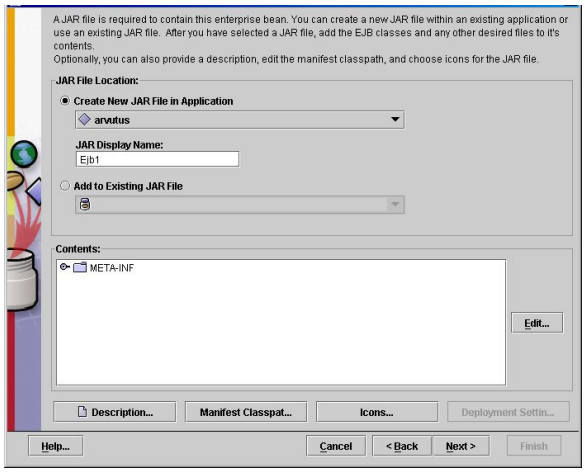


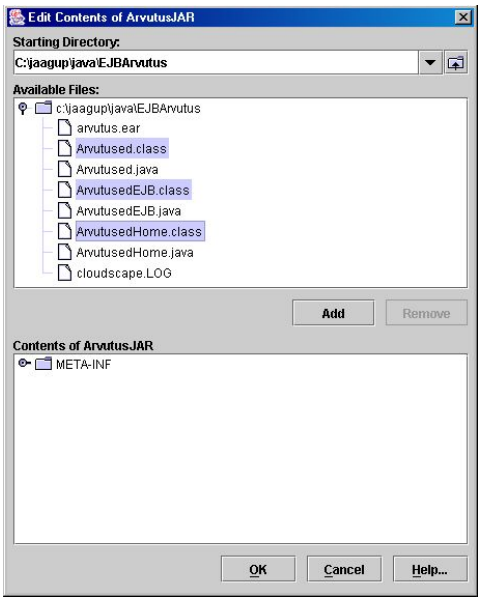


Asunud uba looma, antakse esimesel lehel kohe küllalt pikk seletus, millega tegemist.

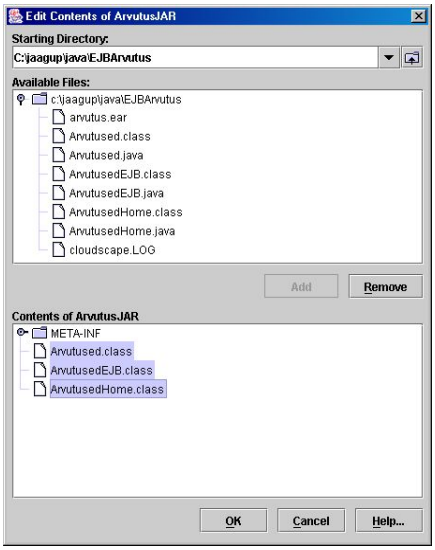


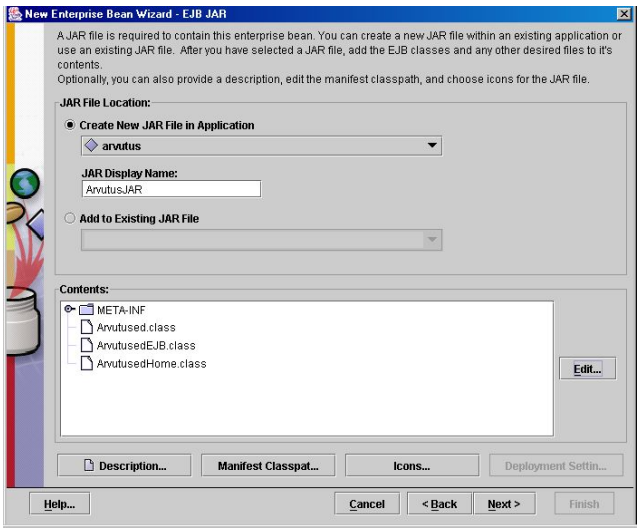
Liikunud Next-iga järgmisele lehele, tuleb asuda määrama nii oa nime kui sinna sisse kuuluvate failide loetelu. Contents – Edit alt tuleb välja järgmine dialoogiaken. Sealt tuleb siis valida oma rakenduse käivitamiseks tarvilikud failid, milleks on .class-id nii käskude kirjedamiseks, oa loomiseks kui ülesande tegelikuks lahendamiseks.



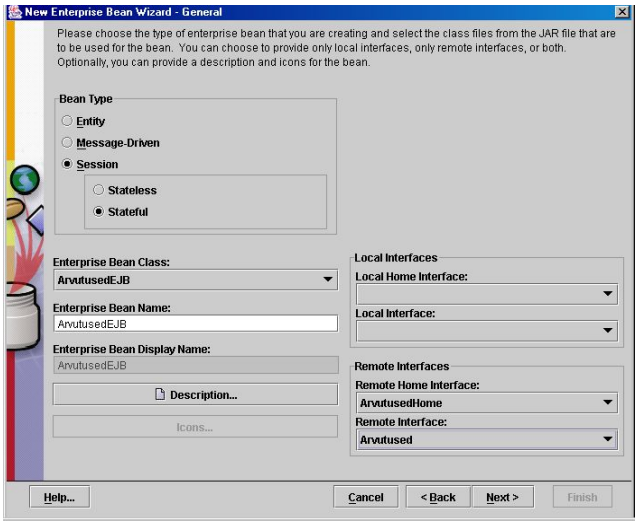


Add-nupule vajutades jõuavad nad oa JAR-faili. Seejärel OK-le vajutades võib faililoendit näha juba järgmises aknas. Jar-failile tuleb ka rakenduse sees kasutamiseks nimi anda. Siin on selleks pakutud ArvutusJAR.

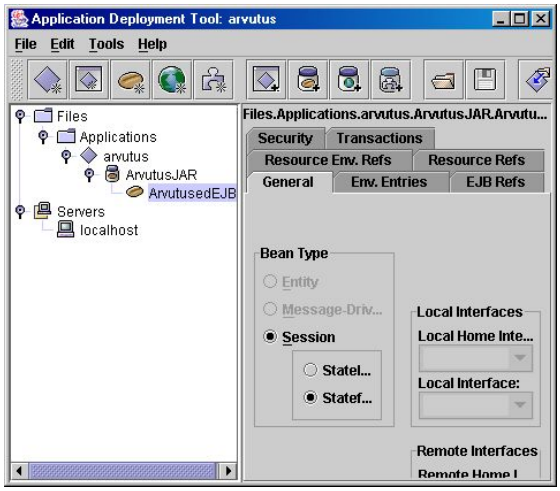


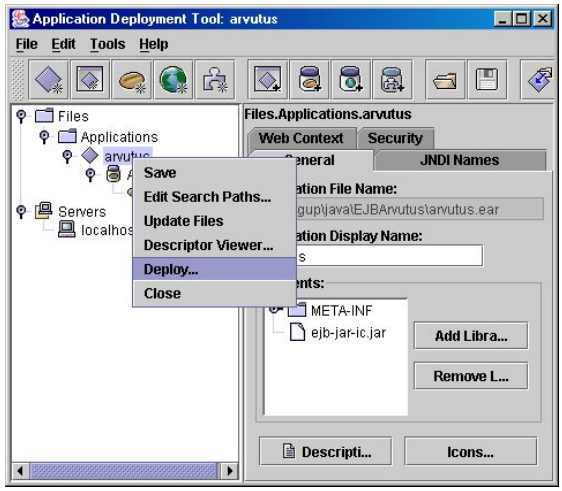


Rakendusserver ei pruugi teada, millist klassi või liidest kavatseme kasutada oa loomiseks, millist oskuste kirjeldamiseks ning millist tegelikuks arvutamiseks. Need tuleb sinna kõik ükshaaval ette ütelda. RemoteHome juurde tuleb oa create-meetodiga liides, Remote juurde käskude loetelu ning Enterprise Bean klassiks saab loodud SessionBean alamklass.

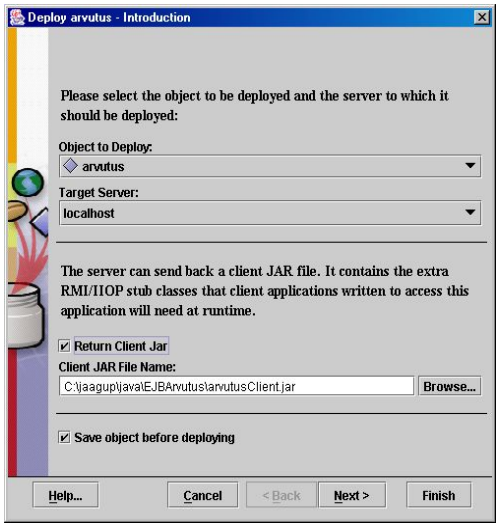


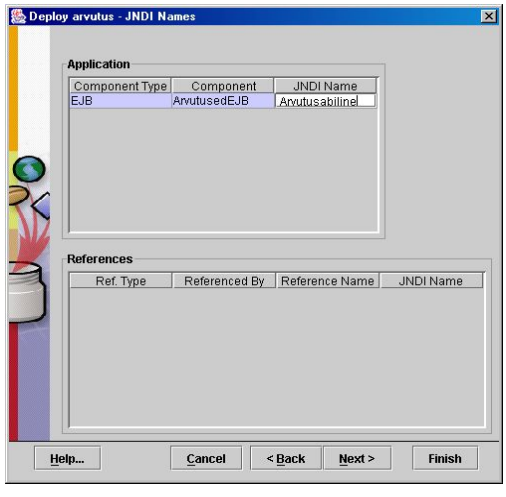
Kui määrangud seatud, võib next-i ja finishiga lõppu minna ning deploytoolis imetleda omavalmistatud uba. Et loodu ka teistele kättesaadavaks saaks, tuleb rakendus serverisse üles panna. Selleks lähen arvutus-nimelise rakenduse peale, vajutan paremat klahvi ning valin deploy.



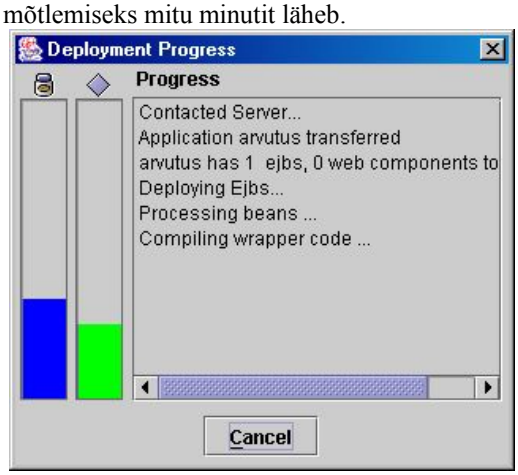


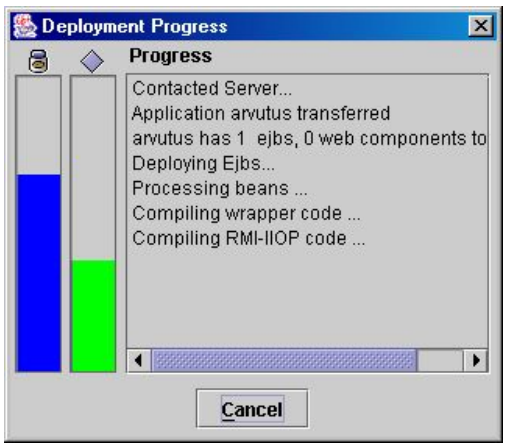
Edasi küsitakse faili nime, millesse andmed kanda. Et õnnestuks pärast rakendust käsurealt käivitada, selleks peab sees olema ristike ruudus “Return Client Jar”. Samuti peab serveris välja pakutud komponendi kohta ütlema, millise nime alt seda tellida saab. Siin on JNDI nimeks pandud Arvutusabiline.



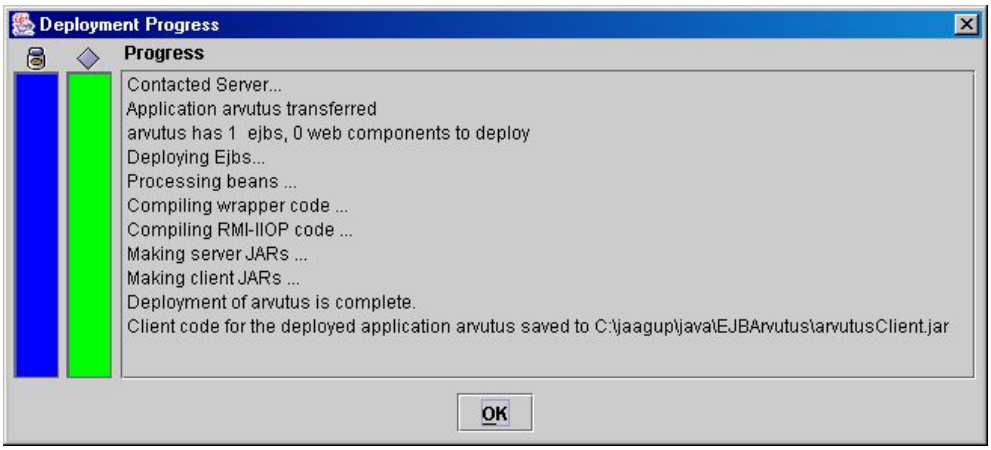


Edasi pole muud kui finish ning kompileerima. Tööd on masinal kõvasti, nii et pole imestada, kui mõtlemiseks mitu minutit läheb.





Kui kõik õnnestub, siis võib lõpus oodata ligikaudu taolist teadet:



Edasi võib loodud oa oskusi testida.

Klient

Kui järgnev koodilõik käsurealt käivitada,

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
public class ArvutusKlient{
  public static void main(String[] argumendid) throws Exception{
    Object osuti=new InitialContext().lookup("Arvutusabiline");
    ArvutusedHome kodu=(ArvutusedHome)PortableRemoteObject.narrow(osuti,
ArvutusedHome.class);
    Arvutused a=kodu.create();
    System.out.println(a.liida(3, 2));
  }
}

C:\jaagup\java\EJBArvutus>javac ArvutusKlient.java
```

Siis tulemus oli järgmine.

```
C:\jaagup\java\EJBArvutus>java -classpath arvutusClient.jar;%classpath% ArvutusKlient
5
```

Sellega võib esimese EJB loomise õnnestunuks kuulutada.

Servleti installeerimine J2EE serverisse

Kõigepealt tuleb servleti käivitamiseks selle kood leida või kokku panna. Testiks peaks sobima järgnev võimalikult lihtne ja lühike servlet.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Servlet1 extends HttpServlet{
  public void doGet(HttpServletRequest kysimus,
                    HttpServletResponse vastus)
              throws IOException, ServletException{
    vastus.setContentType("text/html");
    PrintWriter valja=vastus.getWriter();
    valja.println(
      "<html><body>\n"+
      "  <h2> Tervist! </h2>\n"+
      "</body></html>"
    );
  }
}
```

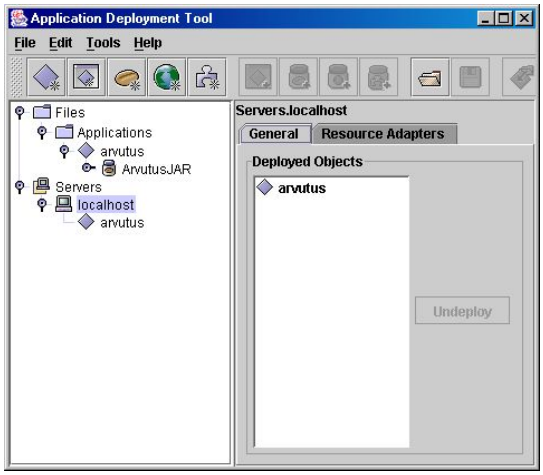
Nagu iga kood, tuleb ka see kompileerida.

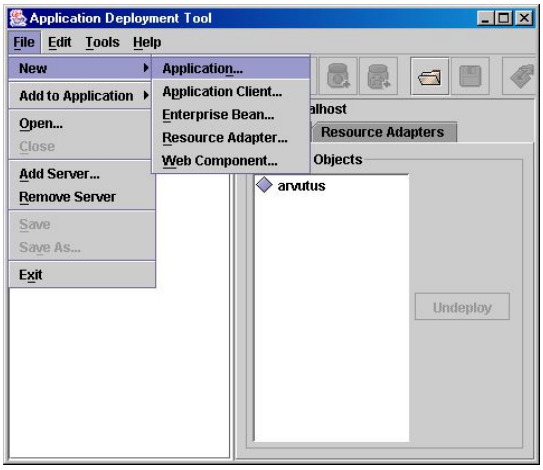
```
C:\jaagup\java\servlet>javac Servlet1.java
```

Kui servleti kompileerimiseks tarvilikud klassid on kättesaadavad, sel juhul võiks ettevõtmine õnnestuda. J2EE-ga peaks sobiv arhiiv kaasa tulema, muul juhul tasub aga näiteks Tomcati-nimelise veebiserveri juurest otsida servlet.jar nimelist faili ning see kas classpath-i või jre\lib\ext-nimelise kataloogi kaudu kompilaatorile kättesaadavaks teha. Tundub et siin kompileerimine õnnestus, sest tekksi Servlet1.class fail.

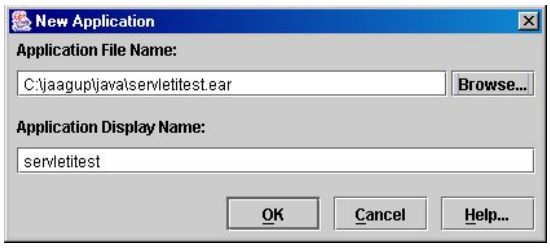
```
C:\jaagup\java\servlet>dir

Directory of C:\jaagup\java\servlet

30.07.2002  09:59       <DIR>          .
30.07.2002  09:59       <DIR>          ..
30.07.2002  09:59                   713 Servlet1.class
30.07.2002  09:59                   472 Servlet1.java
```

Loodud servlet peaks töötama mitmesugustes konteinerites, kaasa arvatud tolles, mis SUNi poolt EJB tarvis tasuta kaasa antakse. Esialgu on näha vaid üks installeeritud Arvutuse-nimeline rakendus. Iseenesest on võimalik loodud servletti ka olemasolevale rakendusele lisada, kuid siin näites teeme eraldi rakenduse. Sellisel juhul on osi kergem lisada ja eemaldada.

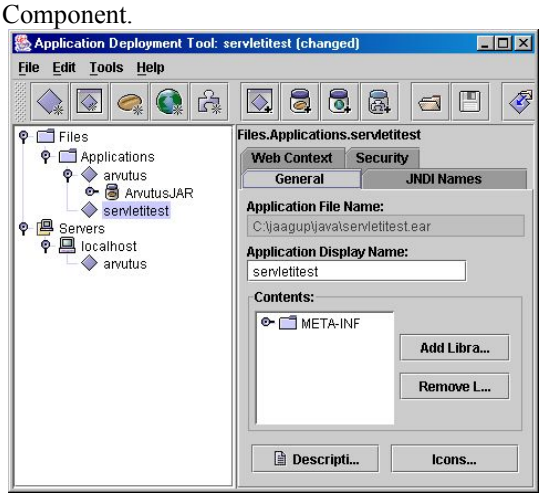


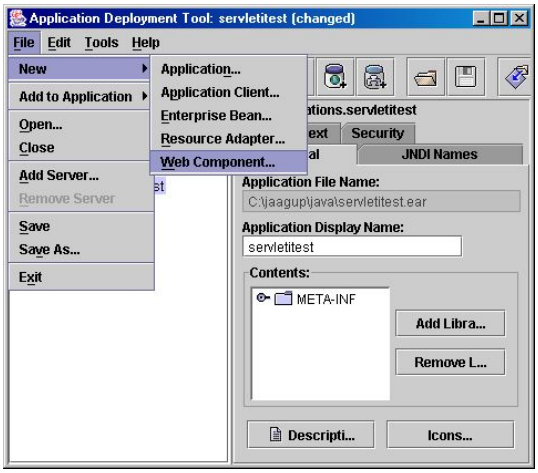


Rakenduse tarvis küsitakse loodava faili nime, samuti nime rakendusele enesele.

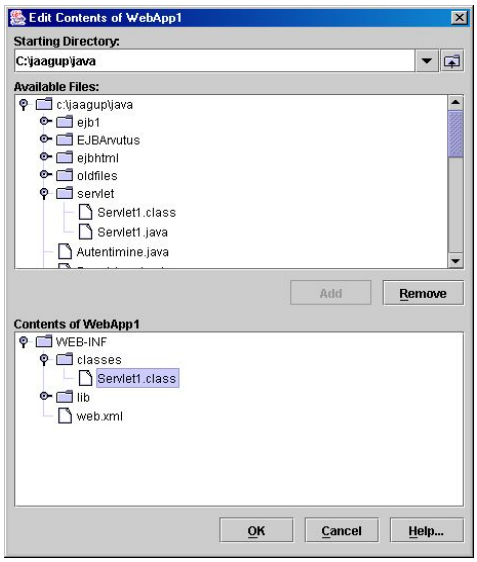


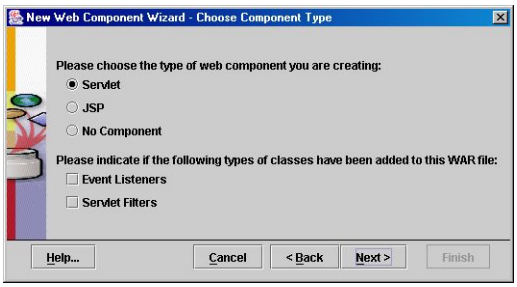
Uue tühja rakenduse loomine õnnestus kergesti. Servleti töö nägemiseks tuleb luua uus Web Component.





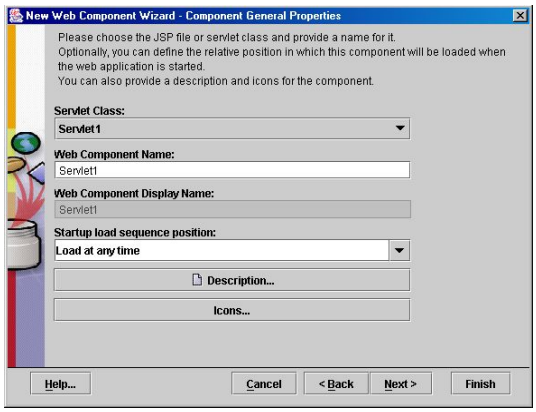
Sinna sisse valida servleti kompileeritud fail. Juhul, kui rakendus vajaks rohkem faile, annab need kõik sobivasse kataloogi paigutada.

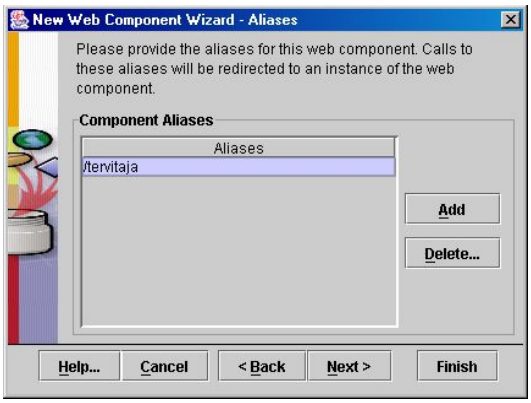




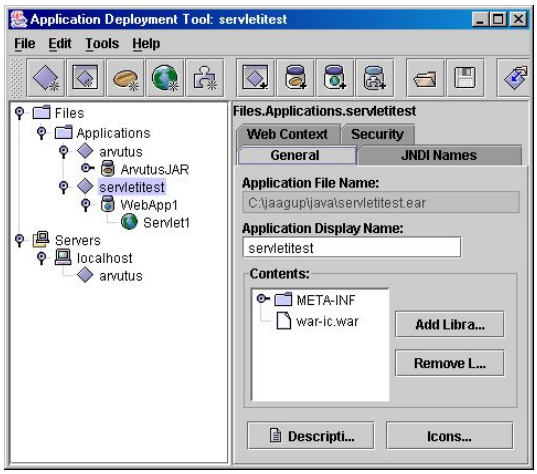
Edasi liikudes tuleb jälgida, et komponendi tüübiks oleks servlet.

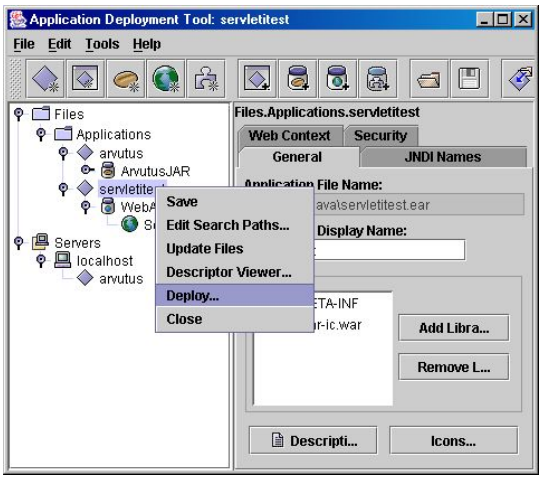
Failide hulgast tuleb valida, milline neist on käivitatav. Kui next-ile vajutades jõutakse Aliases-aknani, tuleb servletile vähemasti üks nimi anda, mille järgi see veebikataloogist üles leitaks. Servleti faili nimi ning URL-il näidatav nimi pole teineteisega seotud. Ühele servletile võib ka mitu aliast panna, sellisel juhul võib igaühe abil neist servleti käivitada.



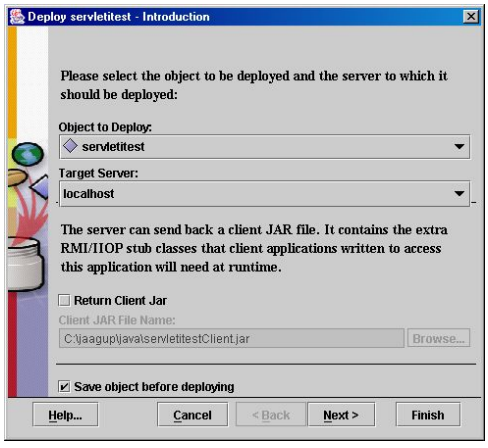


Kui järgnevalt finish vajutada, siis võib näha servlettest'i küljes olevat WebApp1'te, mille all omakorda Servlet1. Rakenduse serveris käivitamiseks tuleb öelda deploy.

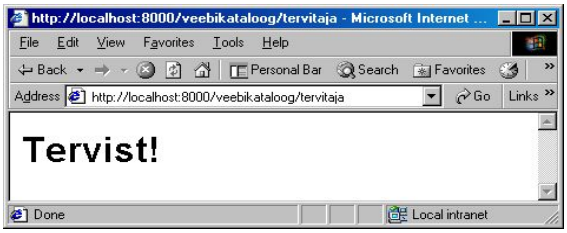




Esimesest ekraanist võib rahulikult edasi jalutada, lihtsalt igaks juhuks kontrollides, et soovitud rakendust soovitud serverisse installeeritakse. Teisel ekraanil tuleb määrata kataloogi nimi, mille all installeeritav rakendus veebibrauseris paistab.



Kui kõik õnneks läks, siis võib avada veebibrauseri, tippida sisse http://localhost:8000, selle järele kataloogi ja alias-nime ning imetleda servleti töö tulemust.



Andmehaldus

Bititöötlus, omaloodud voog, kirjed, puu

Bitid

Ehkki kõrgkeeltes programme kirjutades ei pea me liialt pead vaevama arvuti sisemuses toimetava kahendsüsteemi üle, võib mõnel pool vastavatest teadmistest ikkagi tulu tõusta. Eriti olukordades, kus korraga on vaja üle kanda või salvestada hulgem tõeväärtusi. Nõnda näiteks hiirevajutuse sündmuse juures saab käskluse getModifiersEx kaudu küsida int-tüüpi väärtuse, mille igast bitist saab välja lugeda kas hiirenupu või mõne klaviatuurinupu asukoha. Samuti ei pääse bititehetest pakkimisalgoritmide või muude baidi sisu lahkavate tegevuste puhul.

Bititehetel kasutatakse operaatoreid &, |, <<, >> ning >>> ning neid saab kasutada täisarvude puhul. Esimene neist käitub sarnaselt kui &-tehe tavalistegi tõeväärtuste puhul, st., et tehte tulemus loetakse tõeseks ainult siis, kui mõlemad osapooled on tõesed. Ainult, et arvude puhul võetakse sellesse tehtesse kõik bitid eraldi ning tulemuse arvutamisel pannakse kõik bitid jälle üksteise järgi ritta. Näiteks

1100 ehk 12 ning
0110 ehk 6 annavad & tehte tulemusena kokku
0100 ehk 4.

Kui tarvis arvus kontollida üksiku biti asendit, siis on & hea mugav tehe. Kui &-tehe teha väärtusega, kus vaid üks bitt püsti, siis tulemus saab olla kas null või siis seesama ühe püstise bitiga väärtus. Esimesel juhul ei sattunud vastav bitt kohakuti, teisel juhul sattus.

Sarnaselt võib kontrollida, kas tegemist on paarisarvuga. Kui arvu viimane bitt on 0, siis arv jagub kahega, sest kõik vasakpoolsemad bitid on kahendsüsteemi ülesehituse tõttu paratamatult arvu 2 kordsed. Kui aga parempoolseim bitt on püsti, siis peab arv olema paaritu.

```
public class Bitid1{
  public static void main(String[] argumendid){
    int arv=6;
    if((arv & 1) == 0){
      System.out.println("Paarisarv");
    } else {
      System.out.println("Paaritu arv");
    }
  }
}
```

Märk << väljastab oma vasaku operandi väärtuse nihutatuna vasakule paremal pool kirjutatud arvu jagu kohti. Üldjuhul see annab sama tulemuse kui arvu kahega korrutamine. Kuna kahendsüsteemi bittide väärtused alates paremalt on 1, 2, 4, 8, 16 jne., siis bitte ühe võrra vasakule lükates justkui korrutataks iga moodustavat arvu kahega. Sarnaselt nagu kümnendsüsteemis suurendatakse nulli lõppu lisamisega arvu väärtust kümme korda.

```
public class Bitid2{
  public static void main(String[] argumendid){
    int arv=6;
    arv = arv << 1;
    System.out.println("Kahega korrutatult: "+arv);
  }
}
```

Tahtes arvu lähemalt uurida, tasub vaadata enamasti rohkem kui ühe biti väärtusi. Kui vaadata arvu viimast bitti ning igal korral bitid paremale nihutada, sellisel juhul trükitakse arvu bitid tagurpidises järjekorras. Kui aga vaadata igal korral arvu parempoolseima baidi vasakpoolseima biti väärtust (eraldi võetuna kaks astmes seitse ehk 128) ning siis iga võrdluse järel arvu ühe biti jagu vasakule nihutada, siis saab tulemusena kätte parempoolseima baidi kõik bitid vasakult paremale.

```
public class Bitid3{
  public static void main(String[] argumendid){
    int arv=67;
    System.out.println("Viimased kaheksa bitti:");
    for(int i=0; i<8; i++){
     if((arv & 128) !=0){System.out.print("1");}
```

```
     else {System.out.print(0);}
     arv = arv << 1;
    }
  }
}
/*
D:\Kasutajad\jaagup\java>java Bitid3
Viimased kaheksa bitti:
01000011
*/
```

Bitinihutuskrüptograafia

Kui soovitakse tekst silma ees mõnevõrra loetamatumaks muuta, siis selleks on välja töötatud hulgem võimalusi. Aastatuhandete jooksul on levinumateks võteteks olnud tähtede järjekorra muutmine ning tähtede asendamine. Nüüdisaegsemas raalipõhises krüpteerimises on tähtede asemele tulnud baidid ja bitid ning mitmed päris põhjalikud meetodid. Kuid nii nagu näiteks EditPad Lite-nimelises ja mõnes muuski tekstiredaktoris kasutatakse ROT-13 nimelist täheasendust nihutades inglise tähestiku tähti kolmeteistkümne koha võrra, nii on siin näites liigutatud baidi bitte ringlevalt ühe koha võrra vasakule ning vasakpoolseim bitt paigutatakse parempoolseks.

```
public class Bitid4{
  /**
  * Väljastatakse arvu viimase kaheksa biti väärtused.
  */
  public static void kirjutaBitid(int arv){
    for(int i=0; i<8; i++){
     if((arv & 128) !=0){System.out.print("1");}
     else {System.out.print(0);}
     arv = arv << 1;
    }
    System.out.println();
  }
  /**
  * Nihutab arvu parempoolsed seitse bitti vasakule ning
  * kaheksanda paneb parempoolseks bitiks.
  */
  public static int keeraVasakule(int arv){
    int abi=arv & 128;
    arv = arv << 1;
    arv = arv | (abi >> 7);
    return arv;
  }
  public static void main(String[] argumendid){
    int arv=67;
    kirjutaBitid(arv);
    arv=keeraVasakule(arv);
    kirjutaBitid(arv);
    arv=keeraVasakule(arv);
    kirjutaBitid(arv);
  }
}

/*
D:\Kasutajad\jaagup\java>java Bitid4
01000011
10000110
00001101
*/
```

Baidi bitid failist.

Suurema bitimahuga on pistmist binaarfailide puhul - olgu siis lugemise või kirjutamise juures. Madalama taseme vood ongi loodud arvestusega, et seal on käsud vaid baitide lugemiseks ja kirjutamiseks. Nõnda on lihtsam keskenduda konkreetse sihtseadme jaoks voo loomisele. Vajalikud andmetüüpide muundused saab korda ajada mähisklasside abil. Baidi lugemiseks tuleb avada voog -

praeguse näite puhul failist. Iga read-käsklus loeb ühe baidi mille väärtus talletatakse int-tüüpi muutujasse parempoolseks baidiks ehk muutuja saab väärtuse vahemikus 0-255.

```
    FileInputStream fis=new FileInputStream("ataht.txt");
    int arv=fis.read();
```

Ning töötav kood tervikuna. Eeldatakse, et fail nimega ataht.txt on olemas.

```
import java.io.*;
public class Bitid5{
  public static void kirjutaBitid(int arv){
    for(int i=0; i<8; i++){
     if((arv & 128) !=0){System.out.print("1");}
     else {System.out.print(0);}
     arv = arv << 1;
    }
    System.out.println();
  }
  public static void main(String[] argumendid) throws IOException{
    FileInputStream fis=new FileInputStream("ataht.txt");
    int arv=fis.read();
    kirjutaBitid(arv);
    fis.close();
  }
}
/*
D:\Kasutajad\jaagup\java>java Bitid5
01100001
*/
```

Bitikaupa failikirjutus

Faili bittide kirjutamiseks on mugav teha omaette alamprogramm, mille ülesandeks on hoolitseda, et bitid ilusti ükshaaval baidiks kokku loetaks ning sobival hetkel faili kirjutataks. Et faili saab korraga kirjutada terve baidi, siis peab pidevalt arvet pidama, et õige arv bitte enne kokku saaks kui andmeid kirjutama hakatakse. Samuti peab kirjutamisel ja lugemisel teadma, kummast baidi otsast bitte paigutama hakatakse. Ning üldjuhul on bitte faili mõistlik kirjutada kaheksa kaupa, sest muul juhul pole kuigi lihtne kindlaks teha, kas ülejäänud bittidesse kirjutati nullid või on need väärtused lihtsalt täitmata.

Siin näites on püütud läbi ajada staatiliste funktsioonidega, ilma oma objekte loomata. Et aga faili väljundvoo loomisel võib tekkida erind, siis vastavat voo isendit ei sa muutujate deklareerimise juures luua. Välja aitab staatiline initsialiseerimisplokk, milles võimalik tekkiv erind ka kinni püüda ja töödelda.

Edaspidi piisab programmeerijal vaid sobiva parameetriga välja kutsuda funktsioon nimega kirjutaBittFaili. Muutujas nr loetakse, mitmenda bitiga on tegemist, muutujas malu hoitakse bittide lisamisel tekkivat poolfabrikaati enne tulemuse faili kirjutamist. Kui lisatakse püstine bitt, siis tõstetakse mälumuutuja viimane bitt üheks, muul juhul jäetakse nulliks. Püstitõstmistehe paistab järgnevalt:

```
malu |=1; //viimane bitt pannakse üheks
```

Samuti võiks kirjutada malu=malu|1. Arvu 1 puhul on ainuke püstine bitt viimane. Tehte | puhul loetakse tulemus tõeseks kui vähemalt üks osalevatest pooltest on tõene ning nõnda ükshaaval iga kahe arvu bitipaari korral. Nõnda jäävad arvuga 1 | (või) tehte puhul algse arvu kõik muud bitid paika, vaid viimane tõstetakse püsti sõltumata selle algsest väärtusest.

Edasi: kui kokku on kaheksa bitti täis kirjutatud, siis paigutatakse valmissaanud bait failivoogu ning asutakse taas tühja baidi bitte täitma. Kui aga kaheksat veel täis pole, siis nihutatakse mälumuutujas olevaid kõiki bitte ühe biti jagu vasakule nagu allpool käsus näha on.

```
malu <<=1;
```

oleks pikalt välja kirjutatuna
malu = malu << 1, mis siis tähistabki väärtuse kõigi bittide ühe jagu vasemale lükkamist nõnda, et parempoolseks bitiks tekib 0.

```
    if (nr==8){
      fos.write(malu);
      malu=0;
      nr=0;
    } else {
      malu <<=1;
    }
```

Tervikuna kogu bitikaupa andmeid faili kirjutav rakendus.

```
import java.io.*;
public class Bitid6{
  static FileOutputStream fos;
  static int nr=0;
  static int malu=0;
  static{  //staatiline initsialiseerimisplokk
    try{
      fos=new FileOutputStream("bitijada.dat");
    } catch(IOException e){
      System.out.println(e);
    }
  }
  public static void kirjutaBittFaili(boolean bitt) throws IOException{
    if(bitt){
      malu |=1; //viimane bitt pannakse üheks
    }
    nr++;
    if (nr==8){
      fos.write(malu);
      malu=0;
      nr=0;
    } else {
      malu <<=1;
    }
  }
  public static void main(String[] argumendid) throws IOException{
     kirjutaBittFaili(false);
     kirjutaBittFaili(true);
     kirjutaBittFaili(true);
     kirjutaBittFaili(false);
     kirjutaBittFaili(false);
     kirjutaBittFaili(false);
     kirjutaBittFaili(true);
     kirjutaBittFaili(false); //kokku 98 ehk täht b
     fos.close();
  }
}
```

Bitiväljundvoog

Et muundatud vooge tuleb mitmel pool ette, siis võib selle kirja panna ka omapoolse alamklassina. Nõnda võiks loodud klassi olla hiljem kergem soovitud muudes rakendustes kasutada. Päris standardseks vahendiks selliste voogude loomisel on FilterOutputStreami laiendamine. Siin on lihtsuse mõttes aluseks võetud FileOutputStream. Loodud uue klassi eksemplarile võib konstruktoris öelda soovitud failinime ning juba võibki asuda sinna bitte kirjutama. Bitti kirjutava funktsiooni ülesehitus on jäetud samaks kui eelmises näites. Kuid FileOutputStreami ülekatmise tõttu on siin võimalik otse kasutada kaasatulnud käsklust write ühe baidi faili kirjutamiseks. Sulgemise juures on lisatud kontroll, et andmed ikka terve baidi kaupa korraga kirjutataks. Muul juhul võiks kergesti juhtuda, et viimane bait jõuaks faili loetamatult. Voo sulgemiskäskluse edasisaatmine ülemklassile on samuti vajalik hoolitsemaks, et andmed mälupuhvrist ikka füüsilisele andmekandjale jõuaksid.

```
import java.io.*;

public class Bitid7{
  public static void main(String[] argumendid) throws IOException{
     BitiValjundVoog bvv=new BitiValjundVoog("bitijada.dat");
     bvv.kirjutaBitt(false);
     bvv.kirjutaBitt(true);
     bvv.kirjutaBitt(true);
     bvv.kirjutaBitt(false);
     bvv.kirjutaBitt(false);
     bvv.kirjutaBitt(false);
     bvv.kirjutaBitt(true);
     bvv.kirjutaBitt(false); //kokku 98 ehk täht b
     bvv.close();
  }
  static class BitiValjundVoog extends FileOutputStream{
    int nr=0;
    int malu=0;
```

```
    public BitiValjundVoog(String failinimi) throws IOException{
      super(failinimi);
    }
    public void kirjutaBitt(boolean bitt) throws IOException{
     if(bitt){
      malu |=1; //viimane bitt pannakse üheks
     }
     nr++;
     if (nr==8){
      write(malu);
      malu=0;
      nr=0;
     } else {
      malu <<=1;
     }
    }
    public int mitmesBitt(){
      return nr;
    }
    public void close() throws IOException{
      if(nr!=0){
        throw new IOException("Viimane bait pole valmis "+nr);
      }
      super.close();
    }
  }
}
```

Bittide sisendvoog failist.

Mis kord faili kirjutatud, on kasulik sealt ka välja lugeda. Sarnaselt eelmisele näitele on voog koondatud omaette klassi, et vajadusel saaks seda sarnasena ka mõnes teises rakenduses kasutada. Voole on lisatud käsklus veel() kontrollimaks, et kas on veel bitte võimalik lugeda. Käsklus loeBitt väljastab voost tõeväärtusena järgneva biti.

Käsuga loeJargmine võetakse algandmeid sisaldavast failist ette järgmine bait. Seda nii voo avamisel kui olukorras, kus baidist kaheksa bitti juba loetud on. Kui failist lugemisel saabub -1 ehk voo lõputunnus, jäetakse siingi meelde, et enam midagi lugeda pole. Selle järgi funktsioon veel() teabki küsijale teada anda, et kas midagi lugeda on. Et bitid jääksid lugemisel näha ilusti vasakult paremale, selleks võrreldakse siingi vasakult seitsmenda biti väärtust ning iga küsimise järel liigutatakse kogu mälus oleva muutuja sisu ühe biti jagu vasakule. Ning nõnda võibki omaloodud voost lugeda bitte sarnaselt nagu FileInputStreamist loetakse baite, DataInputStreamist reaalarve või BufferedReaderist ridu. Lihtsalt kontrollitakse, kas midagi lugeda onn. Ning vastavalt biti väärtusele trükitakse välja praegu lihtsalt üks või null.

```
     while(bsv.veel()){
      System.out.print((bsv.loeBitt())?"1":"0");
     }
```

Näitrakenduse kood tervikuna.

```
import java.io.*;

public class Bitid8{
  public static void main(String[] argumendid) throws IOException{
     BitiSisendVoog bsv=new BitiSisendVoog("bitijada.dat");
     while(bsv.veel()){
       System.out.print((bsv.loeBitt())?"1":"0");
     }
     bsv.close();
  }
  static class BitiSisendVoog extends FileInputStream{
    int nr=0;
    boolean veel=true;
    int malu=0;
    public BitiSisendVoog(String failinimi) throws IOException{
      super(failinimi);
      loeJargmine();
    }

    public boolean veel(){
      return veel;
```

```
    }

    private void loeJargmine() throws IOException{
        malu=read();
        nr=0;
        if(malu==-1){veel=false;}
    }

    public boolean loeBitt() throws IOException{
      if(!veel){throw new IOException("Bitid otsas");}
      boolean vastus=(malu&128)!=0; //kas seitsmes bitt on püsti
      malu<<=1;
      nr++;
      if(nr==8){loeJargmine();}
      return vastus;
    }
    public int mitmesBitt(){
      return nr;
    }
  }
}
```

Kokkuvõtteks

Binaarfailidega ümber käimisel on bitioperatsioonid paratamatud. Enamik tulemusi õnnestub leida nii harilike arvutuste kui spetsiaalsete bititehete abil. Osa lõpus näidati, kuidas bittide lugemiseks ja kirjutamiseks võib koostada vooklassid nagu need mitmete muude andmetüüpidega ümber käimiseks juba olemas on.

Ülesandeid

Bitid

* Teata, kas arvu parempoolne bitt on 1
* Teata arvu kõik bitid
* Küsi kahe inimese vanused, talleta nende
  väärtused ühes int-muutujas. Väljasta väärtused

Bitimuster

* Trüki ekraanile kasutaja antud arvu bitid.
* Kasutaja tipitud nullide ja ühtede jada salvesta täisarvuna tekstifaili.
  Hiljem loe see arv failist ning väljasta taas ekraanile.
* Ühes tekstifailis on tühikutest ja ristidest koosnev muster (32 rida, 64
  tähte reas). Talleta see muster arvudena biti kaupa teise faili. Hiljem loe
  arvud failist ning taasta muster.

DNA ahela pakkimine

DNA-ahel koosneb neljast eri tüüpi osast. Tähistame iga desoksüribonukleotiidi järgneva bitikombinatsiooniga.

G - 00
C - 01

A - 10
T - 11

* Talleta ahel CGCACGCTCACTCACG sõnena.
* Trüki tähed tsükli abil ükshaaval, tühikutega eraldatult ekraanile
* Trüki tähed välja neile vastavate koodidena
* Talleta ahel vastavate bittide abil ühes neljabaidises täisarvus
* Loe ahel sellest arvust taas välja ning trüki tulemus.

* Luba ahel sisestada klaviatuurilt (max 256 sümbolit).
  Andmed talletatakse bittidena arvumassiivis. Ahela tegelik pikkus
  salvestatakse massiivi esimeses baidis.
* Loo arvule biti lisamiseks alamprogramm.
* Hoolitse, et programm töötab viisakalt ka vigase sisestuse korral.

Bitinihutus

* Küsi kasutajalt arv (<256) ning väljasta see kujul, kus vasakpoolseim bitt on tõstetud paremale ning ülejäänud ühe võrra vasakule.
* Lisaks eelmisele küsi kasutajalt, mitme biti võrra tahetakse arvu bitte keerata.
* Võrreldes eelmisega keera kasutaja etteantud arvu võrra terve faili iga baidi bitte.

Andmestruktuurid

Andmestruktuuridest räägitakse enamike programmeerimiskeelte juures. Pole siis põhjust ka Java puhul sellest hoiduda - eriti kui täiesti kõlbulikud vahendid keele juurde loodud on.

Esmapilgul võib nimistu käsitlemine mõttetunagi tunduda, sest Java juures on ju olemas Java Collections Framework, mille puhul igaühel õigus priipärast kasutada LinkedList-tüüpi objekti ning selle kaudu elemente hoida, järjestada ja muidki meelde tulevaid toiminguid ette võtta. Tegemist mõnevõrra samalaadse küsimusega, et kas mul on põhjust teada ruutjuure leidmise algoritmi. Iseenesest on vastav käsklus nii enamikes programmeerimiskeeltes kui muude arvutamist võimaldavate seadmete juures olemas. Samas aga, kui kätte juhtub Felixi-nimeline mehhaaniline arvutusmasin millel küll peal korralikud vahendid liitmiseks-lahutamiseks, kuid ruutjuurest mitte märkigi, võib algoritmi teadmine või tuletusoskus päris ilusti tee kätte näidata. Samuti aitab algoritmi aimamine mõista, milliste väärtuste puhul ülesanne arvutile raskem ja millal kergem lahendada on.

Andmestruktuuridele "käsitsi" lähenemiseks ei pruugi sugugi liialt ebatavalist kohta otsida. Kui näiteks asuda programmi abil XMLi andmepuud uurima või muutma, ei pääse enamasti mööda elemente ringi liikumisest. Kui püüda failisüsteemis toimetada, siis seal on tegemist samuti hierarhiliste andmetega, milles liikumisel tuleb arvestada nii enese asukohta, ülemkatalooge kui alanevaid faile ja katalooge. Kolmandaks sarnaseks näiteks sobiks ehk Swingi puu: üksikute okste ja lehtede abil tuleb kõik küllaltki nullist valmis ehitada. Ning nagu õppejõud Isotamm Tartu Ülikooli kompilaatorite kursuse esimeses loengus ütles: kui andmed on õnnestunud mälus kahendpuusse ajada, siis sageli on programmeerija jaoks pool tööd valmis.

Järgnevalt püüame võimalikult lihtsate näidete varal levinumad andmete mälus paiknemisega seotud võimalused läbi mängida, et hiljem oleks oskust ja julgust omal jõul vajadusel ka keerukamaid struktuure ehitada.

Nimistu

Tõenäoliselt levinuimaks andmestruktuuriks on ahel. Ehk jada, kus igal elemendil on oma väärtus ning siis ka osuti järgmisele isendile ahelas või tühiväärtus tähistamaks ahela lõppu. Siinsetes

näidetes on andmetüübiks võetud lihtsuse mõttes int, kuid andmeteks võib iseenesest olla ükskõik milline liht- või struktuurtüüp. C++is kasutatakse malle võimaldamaks kord valmis ehitatud andmestruktuurivahendeid kasutamaks igasuguste tüüpidega. Javas tavatsetakse andmetüübiks panna Object, siis on võimalik sellesse muutujasse paigutada igasugu struktuurtüüpide eksemplare sõltumata nende tegelikust andmetüübist. Siin näidetes aga on piirdutud int-väärtusega.

Üksik rakk

```
public class Rakk{
  int sisu;
  Rakk jargmine;
}
```

Seotud rakud

Sarnaselt loodud Rakkude eksemplare võib rahus üksteisega siduda. Siin näites koostatakse kaks Raku eksemplari. Ühele antakse väärtuseks 13, teisele 20. Ning kuna esimese raku muutuja jargmine pannakse osutama teisele rakule, siis avaldise r1.jargmine.sisu väärtuseks on teise raku sisu ehk 20.

```
public class Rakukatse1{
  public static void main(String argumendid[]){
    Rakk r1=new Rakk();
    r1.sisu=13;
    System.out.println(r1.sisu);
    Rakk r2=new Rakk();
    r2.sisu=20;
    r1.jargmine=r2;
    System.out.println(r1.jargmine.sisu);
  }
}
```

Pikem ahel

Rakke võib nõnda ka rohkem üksteisele sabasse lükkida. Keele poolt piirangut ei ole, vaid suurtel arvudel võib lihtsalt mälumahuga probleeme tekkida. Siinne r1.jargmine.jargmine.sisu ja sealt seest paistev r3-e väärtus lihtsalt tutvustab võimalikku jadastamist.

```
public class Rakukatse1{
  public static void main(String argumendid[]){
    Rakk r1=new Rakk();
    r1.sisu=13;
    System.out.println(r1.sisu);
    Rakk r2=new Rakk();
    r2.sisu=20;
    r1.jargmine=r2;
    System.out.println(r1.jargmine.sisu);
    Rakk r3=new Rakk();
    r3.sisu=7;
    r2.jargmine=r3;
    System.out.println(r1.jargmine.jargmine.sisu);
  }
}
```

Vähem muutujaid

Kui ahelas on lülisid rohkem, siis pole mõistlik igale elemendile omaette muutujat luua. Piisab, kui osuteid mööda on võimalik sobivasse kohta liikuda. Nõnda võib muutuja uus abiga luua pika rivi ilma, et oleks rohkem muutujaid juurde tarvis. Alguse tarvis on muutuja ikka vajalik, sest sealtkaudu pääseb viiteid pidi ahela sees olevatele elementidele ligi.

```
public class Rakukatse2{
  public static void main(String argumendid[]){
```

```
    Rakk algus=new Rakk();
    algus.sisu=13;
    Rakk uus=new Rakk();
    uus.sisu=20;
    algus.jargmine=uus;
    uus.jargmine=new Rakk();
    uus=uus.jargmine;
    uus.sisu=7;
    System.out.println(algus.sisu);
    System.out.println(algus.jargmine.sisu);
    System.out.println(algus.jargmine.jargmine.sisu);
  }
}
```

Ahela läbimine tsükliga

Võrreldes eelmisega ongi siin sama muutujate arvu juures tsükli abil hulga pikem ahel kokku pandud. Iga ringi juures lihtsalt luuakse uus eksemplar, antakse sellele sisu ning ollakse taas valmis uue elemendi loomiseks.

```
public class Rakukatse3{
  public static void main(String argumendid[]){
    Rakk algus=new Rakk();
    algus.sisu=10;
    Rakk uus=algus;
    for(int i=20; i<=100; i=i+10){
      uus.jargmine=new Rakk();
      uus=uus.jargmine;
      uus.sisu=i;
    }
    System.out.println(algus.sisu);
    System.out.println(algus.jargmine.sisu);
    System.out.println(algus.jargmine.jargmine.sisu);
  }
}
```

Väljatrükk

Mis kord sisse kirjutatud, see ka hea välja lugeda. Olgu siis kontrolli eesmärgil või pärastpoole ka tunduvalt asjalikumate ettevõtmiste tarvis. Jada lõppu saab null-tunnuse järgi kontrollida, sest õnnelikult on Java keele juures määratud, et väärtustamata isendimuutujad on nullilise väärtusega algväärtustatud. Nõnda, et kui ahela viimasel rakul on küll andmete juures väärtus, kuid järgmise elemendi tarvis pole midagi juurde pandud, siis võib uskuda, et muutuja jargmine väärtuseks on null.

Jada trükkimisel piisab ühest muutujast juhul, kui piisab vaid ühekordsest läbimisest. Ning kuna peaprogrammis on muutuja "algus" kindlalt paigas, siis pole karta, et ahela algus võiks kogemata käest lipsata.

Näidatud trükkimiskäsklust läheb olukorra kontrollimiseks vaja mitmes järgmiseski näites.

```
public class Rakukatse4{
  public static void trykiJada(Rakk jooksev){
    while(jooksev!=null){
      System.out.println(jooksev.sisu);
      jooksev=jooksev.jargmine;
    }
  }

  public static void main(String argumendid[]){
    Rakk algus=new Rakk();
    algus.sisu=10;
    Rakk uus=algus;
    for(int i=20; i<=100; i=i+10){
      uus.jargmine=new Rakk();
      uus=uus.jargmine;
      uus.sisu=i;
    }
    trykiJada(algus.jargmine.jargmine);
  }
}
```

Vahelepanek

Võrreldes eelnenud näitega lisatakse siin vahelepaneku võimalus. Ahela puhul on vahelepanek märkimisväärselt tähtis omadus. Kui andmeid hoitakse massiivis ning on vaja midagi lisada, siis üldjuhul ei õnnestu toiming muidu, kui tuleb märgatav osa massiivi andmetest ümber tõsta. Ahela puhul aga piisab vaid uue raku juures viited sobivalt paika sättida. Nii et uue raku järgmine näitaks tegeliku järgmise peale. Ning et vahelepandud rakule eelneva raku edasi-viide näitaks uuele rakule.

```
    Rakk uus=new Rakk();
    uus.sisu=vaartus;
    uus.jargmine=eelmine.jargmine;
    eelmine.jargmine=uus;
```

Näide tervikuna:

```
public class Rakukatse5{
  public static void trykiJada(Rakk jooksev){
    while(jooksev!=null){
      System.out.println(jooksev.sisu);
      jooksev=jooksev.jargmine;
    }
  }

  public static void lisaVahele(Rakk eelmine, int vaartus){
    Rakk uus=new Rakk();
    uus.sisu=vaartus;
    uus.jargmine=eelmine.jargmine;
    eelmine.jargmine=uus;
  }

  public static Rakk looJada(int vahim, int suurim, int vahe){
    Rakk algus=new Rakk();
    algus.sisu=vahim;
    Rakk uus=algus;
    for(int i=vahim+vahe; i<=suurim; i=i+vahe){
      uus.jargmine=new Rakk();
      uus=uus.jargmine;
      uus.sisu=i;
    }
    return algus;
  }

  public static void main(String argumendid[]){
    Rakk algus=looJada(20, 100, 10);
    lisaVahele(algus.jargmine, 35);
    trykiJada(algus);
  }
}

/*
D:\arhiiv\naited\io\muu>java Rakukatse5
20
30
35
40
50
60
70
80
90
100
*/
```

Järjestamine

Märgatava osa maailma arvutite jõudlusest pidada võtma mitmesuguste andmete järjestamine ja otsimine. Heaks lihtsaks sorteerimisalgoritmiks peaks olema võimalus elemente üksteise järgi väärtuste järjekorras sobivasse kohta vahele panna. Lisaks tavalisele sobivasse kohta paigutamisele tekib kaks erandlikku olukorda: lisatav on kõigist andmetest kas suurem või väiksem. Kui soovitakse ahela lõppu panna, siis võib seda mõnes mõttes ette kujutada kui elemendi lisamist viimase elemendi ja tema poolt viidatava tühiväärtuse vahele. Ette lisamine aga mõnevõrra keerulisem: paigast nihkub ka muidu

muutumatuna tunduv ahela algus. Siinses näites ei pruugi sellest midagi suuremat hullu olla. Aga kohtades, kus ahela algus on kirjas mitmes paigas, et oleks mugav andmetele ligi pääseda, võib alguse asukoha muutmine muresid põhjustada. Et algus võib lisamisel paigast nihkuda, selleks siis jäetakse iga käigu juures meelde ka uus algus pärast vastavat sammu. Toiming, mida eelnenud näidete juures tarvis polnud.

```
    algus=lisaVaartus(algus, 5);
```

Kuna vahele lisamisel on vaja ligi pääseda nii uuest sõlmest ette- kui tahapoole jäävatele sõlmedele, siis sobiva koha otsimisel tuleb meeles pidada nii jooksev element, mille väärtust võrreldakse vahelepandava väärtusega kui ka jooksvale elemendile eelneva elemendi osuti, et oleks vahelepanekul võimalus kõik osutid korralikult kätte saada ja paika sättida.

```
   while(jooksev!=null && vaartus>jooksev.sisu){
     eelmine=jooksev;
     jooksev=jooksev.jargmine;
   }
```

Ning vahelepaigutamistega näide kogu pikkuses.

```
public class Rakukatse6{
  public static void trykiJada(Rakk jooksev){
    while(jooksev!=null){
      System.out.println(jooksev.sisu);
      jooksev=jooksev.jargmine;
    }
  }

  public static void lisaVahele(Rakk eelmine, int vaartus){
    Rakk uus=new Rakk();
    uus.sisu=vaartus;
    uus.jargmine=eelmine.jargmine;
    eelmine.jargmine=uus;
  }

  public static Rakk lisaEtte(Rakk algus, int vaartus){
    Rakk uusAlgus=new Rakk();
    uusAlgus.sisu=vaartus;
    uusAlgus.jargmine=algus;
    return uusAlgus;
  }

  public static Rakk lisaVaartus(Rakk algus, int vaartus){
   Rakk jooksev=algus;
   if(vaartus<jooksev.sisu){
     return lisaEtte(jooksev, vaartus);
   }
   Rakk eelmine=jooksev;
   jooksev=jooksev.jargmine;
   while(jooksev!=null && vaartus>jooksev.sisu){
     eelmine=jooksev;
     jooksev=jooksev.jargmine;
   }
   lisaVahele(eelmine, vaartus);
   return algus;
  }

  public static Rakk looJada(int vahim, int suurim, int vahe){
    Rakk algus=new Rakk();
    algus.sisu=vahim;
    Rakk uus=algus;
    for(int i=vahim+vahe; i<=suurim; i=i+vahe){
      uus.jargmine=new Rakk();
      uus=uus.jargmine;
      uus.sisu=i;
    }
    return algus;
  }

  public static void main(String argumendid[]){
    Rakk algus=looJada(20, 100, 10);
    algus=lisaVaartus(algus, 45);
    algus=lisaVaartus(algus, 48);
    algus=lisaVaartus(algus, 5);
    algus=lisaVaartus(algus, 500);
    trykiJada(algus);
```

```
  }
}
/*
D:\arhiiv\naited\io\muu>java Rakukatse6
5
20
30
40
45
48
50
60
70
80
90
100
500

*/
```

Jadasse elementide lisamiseks ei pea sugugi tervet jada valmis kirjutama. Kui tahetakse olemasolevaid väärtusi jadasse ja ilusti ritta paigutada, siis piisab alustuseks tühjast jadast, ehk lihtsalt Rakutüüpi muutujast millel väärtuseks null. Nõnda, et jada algus ja lõpp on samas kohas ning sees polegi midagi kahtlast. Vaid väärtuse lisamisel peab arvestama, et alguseks võib olla ka tühiväärtus ning sellisel juhul tuleb täiendav kontroll lisada. Et kui jada juhtub tühi olema, siis lisatav element saab jada ainukeseks elemendiks ja ühtlasi ka jada alguseks.

```
  public static Rakk lisaVaartus(Rakk algus, int vaartus){
   Rakk jooksev=algus;
   if(algus==null){
     Rakk uus=new Rakk();
     uus.sisu=vaartus;
     return uus;
   }
...
}
```

Kui nüüd tühja jadasse asutakse ükshaaval elemente lisama ning iga lisamise puhul hoolitsetakse, et elemendid ikka soovitud järjekorda jääksid, siis tulemuseks ongi soovitult sorditud jada.

```
public class Rakukatse7{
  public static void trykiJada(Rakk jooksev){
    while(jooksev!=null){
      System.out.println(jooksev.sisu);
      jooksev=jooksev.jargmine;
    }
  }

  public static void lisaVahele(Rakk eelmine, int vaartus){
    Rakk uus=new Rakk();
    uus.sisu=vaartus;
    uus.jargmine=eelmine.jargmine;
    eelmine.jargmine=uus;
  }

  public static Rakk lisaEtte(Rakk algus, int vaartus){
    Rakk uusAlgus=new Rakk();
    uusAlgus.sisu=vaartus;
    uusAlgus.jargmine=algus;
    return uusAlgus;
  }

  public static Rakk lisaVaartus(Rakk algus, int vaartus){
   Rakk jooksev=algus;
   if(algus==null){
     Rakk uus=new Rakk();
     uus.sisu=vaartus;
     return uus;
   }
   if(vaartus<jooksev.sisu){
     return lisaEtte(jooksev, vaartus);
   }
   Rakk eelmine=jooksev;
   jooksev=jooksev.jargmine;
   while(jooksev!=null && vaartus>jooksev.sisu){
     eelmine=jooksev;
     jooksev=jooksev.jargmine;
   }
```

```
   lisaVahele(eelmine, vaartus);
   return algus;
  }

  public static void main(String argumendid[]){
     int[] arvud={4,2,56,2,3,6,4};
     Rakk algus=null;
     for(int i=0; i<arvud.length; i++){
       algus=lisaVaartus(algus, arvud[i]);
     }
     trykiJada(algus);
  }
}

/*
D:\arhiiv\naited\io\muu>java Rakukatse7
2
2
3
4
4
6
56
*/
```

Ülesandeid

Pinu

* Kasutajalt küsitakse kaks arvu ning väljastatakse need vastupidises järjekorras.
* Iga kord, kui kasutaja kirjutab positiivse arvu, lisatakse see pinu lõppu (peale). Kui kasutaja kirjutab nulli, siis väljastatakse pinu pealmine element ning eemaldatakse see pinust.
* Realiseeri sarnane pinu nii massiivi kui viitadega ühendatud elementide abil.

Järjekord

* Küsi kasutajalt kaks arvu ning väljasta neist esimene.
* Iga käsuga saab kasutaja määrata, kas ta soovib arvu lisada või küsida. Sisestatud arv lisatakse lõppu, eemaldamise puhul väljastatakse arv ja eemaldatakse algusest. Sisestatud arvude hulk võib olla piiratud massiivi maksimaalse pikkusega.
* Realiseeri järjekord nii massiivi kui viitadega ühendatud elementide abil. Massiivi puhul võib samaaegne järjekorras olevate elementide arv olla piiratud massiivi pikkusega, pideval lisamisel ja eemaldamisel aga piiranguid ei seata.

Osutiring

* Loo andmetüüp, mis koosneks täisarvust ning osutist samatüübilisele rakule. Loo kaks eksemplari ning pane nad üksteisele näitama. Testi tulemust.
* Loo alamprogramm, mis kontrolliks, kas etteantud osutiahel moodustab silmuse ning viimasel juhul teata, mitmest elemendist ringahel koosneb.
* Loo alamprogramm ringikujulisse osutiahelasse elemendi lisamiseks ning teine sellisest ahelast elemendi eemaldamiseks. Nende kahe abil ühenda kaks ringikujulist osutiahelat.

Kahendpuu

Suuremate andmemahtude korral lähevad ahelad pikaks ning soovitud kohta jõudmine võtab palju samme. Liikumist konkreetse sõlmeni aitavad lühendada otseteed ehk viited sõlmedeni ahela keskosas.

Levinud andmestruktuuriks on kahendpuu: iga sõlme juures on andmeplokk ning osutid kahele sõlmele, mida tinglikult võib nimetada vasakuks ja paremaks haruks.

Üksik sõlm

```
public class Solm{
  int sisu;
  Solm vasak;
  Solm parem;
}
```

Nõnda andmeid paigutades võib õnnestuda tulemus, kus suhteliselt lühikese teega on võimalik jõuda iga väärtuseni. Kaks astmes 10 on 1024, astmes 20 juba üle miljoni. Siit paistab, et kui andmepuu on tasakaalustatud, st., et puu üksikud harud on ligikaudu ühepikkused, siis ka miljoni väärtuse salvestamisel ei pruugi iga väärtuse leidmiseks kuluda oluliselt üle kahekümne sammu. Lineaarse ahela puhul tulnuks aga läbi käia keskeltläbi pool miljonit sõlme.

Osutid, millele pole väärtust omistatud, sisaldavad tühiväärtust null. Nii on kindlalt võimalik määrata, kus miski haru lõpeb. Puu võib koosneda ka vaid ühest sõlmest.

```
public class Solmetest1{
  public static void main(String[] argumendid){
    Solm s1=new Solm();
    s1.sisu=7;
    System.out.println(s1.sisu+" "+s1.vasak+" "+s1.parem);
  }
}
```

Kahe haruga puu

Kui sõlmesid loodud rohkem, siis saab nad omavahel ühendada. Praeguses näites esialgu igale sõlmele oma väärtus ning siis paigutatakse esimese sõlme vasaku haru külge teine sõlm ning parema haru külge kolmas sõlm. Avaldis s1.parem.sisu annab nõnda kolmanda sõlme väärtuse.

```
public class Solmetest2{
  public static void main(String[] argumendid){
    Solm s1=new Solm();
    Solm s2=new Solm();
    Solm s3=new Solm();
    s1.sisu=4;
    s2.sisu=3;
    s3.sisu=8;
    s1.vasak=s2;
    s1.parem=s3;
    System.out.println(s1.sisu+" "+s1.vasak.sisu+" "+s1.parem.sisu);
  }
}
```

Sõlmede ahela saab ka pikema ehitada. Ning enamasti ongi andmepuus märgatavalt enam kui paar-kolm väärtust. Nagu näha, luuakse siin algul sõlmed, määratakse neile väärtused ning siis asutakse sõlmesid ühendama. Pärast on võimalik asukohtade järgi väärtused välja küsida.

```
public class Solmetest3{
  public static void main(String[] argumendid){
    Solm s1=new Solm();
    Solm s2=new Solm();
    Solm s3=new Solm();
    Solm s4=new Solm();
    Solm s5=new Solm();
    Solm s6=new Solm();
```

```
    s1.sisu=4;
    s2.sisu=3;
    s3.sisu=8;
    s4.sisu=5;
    s5.sisu=7;
    s6.sisu=10;
    s1.vasak=s2;
    s1.parem=s3;
    s2.vasak=s4;
    s3.vasak=s5;
    s3.parem=s6;
    System.out.println(
       s1.sisu+" "+s1.vasak.sisu+" "+s1.vasak.vasak.sisu+" "+
       s1.parem.sisu+" "+s1.parem.vasak.sisu+" "+s1.parem.parem.sisu);
  }
}
```

Rekursioon

Kui sõlmesid on rohkem, siis peab nendega toimetamiseks miski üldisema mooduse leidma. Ahela elementide läbimiseks sobib tsükkel, puu puhul on aga tsükliga vähem ette võtta. Iseenesest on võimalik puu läbimiseks koostada abiahel kohtadest, millises puu juureni ulatuvas harus parajasti ollakse ning nõnda meeles pidades püüda kogu puu järjestikku läbi käia. Teiseks võimaluseks on rekursioon, ehk alamprogrammid oma väljakutsete kaudu peavad meeles, millises harus parajasti ollakse. Ning et korraga võib samast alamprogrammist töös olla rohkem kui üks eksemplar. Toimingud liiguvad kõige viimati käivitatud eksemplaris. Ülejäänud on ootel, kuniks välja kutsutud eksemplar oma tööga ühele poole saab.

Siin näites ongi väjatrükk rekursiooni hooleks jäetud. Alamprogrammile antakse ette sõlm ning alamprogrammi ülesandeks on trükkida välja selle sõlme ning kõigi alanevate sõlmede sisu. Enese sees olevate sõlmede väljatrükiks on lubatud kasutada alamprogrammi ennast. Rekursiooni baasiks ehk olukorraks, kus enam uuesti samasse alamprogrammi ei siseneta on juhtum, kui trükitav sõlm puudub, ehk muutuja väärtuseks on null. Nõnda siis juure ning kahe alaneva elemendiga puus kutsutakse alamprogrammi välja päris mitu korda. Kõigepealt juure jaoks. Kuna juure muutuja pole null, siis liigutakse edasi. Trükitakse juurelemendi andmeploki väärtus ning edasi kutsutakse sama funktsioon välja vasaku haru jaoks. Juure trükkimist alustanud funktsiooni eksemplar jääb vasaku haru trükkimise ajaks ootele. Vasaku haru trükkimise eksemplar väljastab kõigepealt andmesegmendi väärtuse ning siis proovib vasakust harust omakorda vasakut haru trükkida. Et sealne osuti on tühi, siis kolmandast trükkimisfunktsiooni eksemplarist jõutakse kiiresti tagasi ning proovitakse vasaku haru paremat haru välja trükkida. Et ka sealne pool on tühi, siis väljutakse juure vasakut haru trükkivast funktsioonist ning alustatakse parema poole trükkimist. Ning parema poole puhul tulevad samasugused kolm väljakutset. Kõigepealt sõlme enese oma ning siis kaks lühikest käivitust tühjade osutite tarbeks. Samasugust juhtumist võimegi näha s3-e ehk sõlme väärtusega 8 väljatrükil. Hilisem kogu puu väljatrükk paigutab kõik väärtused ekraanile.

```
public class Solmetest4{
  //trükitakse väärtused vastavast sõlmest alates
  public static void trykiSolm(Solm s){
    if(s==null){return;}
    System.out.print(s.sisu+" ");
    trykiSolm(s.vasak);
    trykiSolm(s.parem);
  }

  public static void main(String[] argumendid){
    Solm s1=new Solm();
    Solm s2=new Solm();
    Solm s3=new Solm();
    Solm s4=new Solm();
    Solm s5=new Solm();
    Solm s6=new Solm();
    s1.sisu=4;
    s2.sisu=3;
    s3.sisu=8;
    s4.sisu=5;
    s5.sisu=7;
    s6.sisu=10;
    s1.vasak=s2;
    s1.parem=s3;
    s2.vasak=s4;
    s3.vasak=s5;
    s3.parem=s6;
```

```
    trykiSolm(s3);
    System.out.println();
    trykiSolm(s1);
  }
}
```

Sõlmede paiknemine ja väljatrüki tulemused.

```
/*
     4
    / \
   3   8
  /   / \
 5   7  10

C:\jaagup>java Solmetest4
8 7 10
4 3 5 8 7 10

*/
```

Järjestamine

Kahendpuud rakendatakse sageli väärtuste järjestamisel. Eeliseks ahelate ees just võimalus, et loetelu keskel paiknevate liikmeteni jõuab tunduvalt vähesema sammude arvuga. Kui andmete puusse paigutades hoolitseda, et elemendist ühele poole jääks alati temast väiksem ning teisele poole temast suurem väärtus, siis lõpuks ongi võimalik kätte saada nii järjestatud väärtused kui vajadusel suhteliselt vähese sammude arvuga kindlaks teha, kas soovitud väärtus puus leidub. Siinses näites korduvaid väärtusi puusse ei lisata. Suuremad väärtused paigutatakse paremale ning väiksemad vasakule.

Alamprogrammile antakse ette sõlm, kuhu juurde panna ning uus väärtus mida panna. Alamprogrammil on õigus iseennast teiste parameetritega välja kutsuda. Nõnda võib vajadusel otsida sisule sobivat kohta algsele sõlmele alanejate hulgas.

```
public static void lisa(Solm s, int sisu){
```

Kui vastav väärtus juba andmepuus olemas on, siis teist samasugust ei lisata, vaid toiming katkestatakse.

```
    if(s.sisu==sisu){return;}
```

Kui uus väärtus on jooksva sõlme väärtusest suurem, siis püütakse uus väärtus paigutada paremale.

```
    if(sisu>s.sisu){
```

Kui jooksva elemendi parempoolne haru on tühi

```
      if(s.parem==null){
```

Siis luuaks sinna lihtsalt uus sõlm ning pannakse uus väärtus selle sisse.

```
        s.parem=new Solm();
        s.parem.sisu=sisu;
      } else {
```

Kui aga element juhtus paremal juba olemas olema, siis käivitatakse sama lisamise alamprogramm juba selle paremal pool asuva elemendi suhtes. Hakatakse siis võrdlema, kas vastav väärtus on tolle parempoolse elemendi oma ning võib puusse lisama jätta või tuleb omakorda järgnevat paika otsima asuda.

```
        lisa(s.parem, sisu);
      }
    }
```

Ning vasaku poolega sama lugu.

```
    if(sisu<s.sisu){
```

```
      if(s.vasak==null){
        s.vasak=new Solm();
        s.vasak.sisu=sisu;
      } else {
        lisa(s.vasak, sisu);
      }
    }
  }
```

Ning näitrakendus tervikuna.

```
public class Solmetest5{
  public static void tryki(Solm s){
    if(s==null){return;}
    tryki(s.vasak);
    System.out.print(s.sisu+" ");
    tryki(s.parem);
  }

  public static void lisa(Solm s, int sisu){
    if(s.sisu==sisu){return;}
    if(sisu>s.sisu){
      if(s.parem==null){
        s.parem=new Solm();
        s.parem.sisu=sisu;
      } else {
        lisa(s.parem, sisu);
      }
    }
    if(sisu<s.sisu){
      if(s.vasak==null){
        s.vasak=new Solm();
        s.vasak.sisu=sisu;
      } else {
        lisa(s.vasak, sisu);
      }
    }
  }

  public static void main(String[] argumendid){
    Solm s=new Solm();
    s.sisu=3;
    lisa(s, 2);
    lisa(s, 6);
//    tryki(s);
    int[] arvud={4, 3, 3, 1, 9, 8, 0};
    for(int nr=0; nr<arvud.length; nr++){
      lisa(s, arvud[nr]);
    }
    tryki(s);
  }
}
/*
D:\arhiiv\naited\io\muu>java Solmetest5
0 1 2 3 4 6 8 9
*/
```

Otsimine

Kui andmed on kindla korra järgi puusse paigutatud, siis saab selle sama korra järgi neid ka otsida. Või kui andmed on lihtsalt puus, siis saab tulemuse kätte kogu puu läbi vaadates. Siin vaadatakse otsimisel läbi jooksev element ning mõlema poole alampuud ning kui kusagilt otsitav leiti, siis väljastatakse selle kohta "jah".

```
  public static boolean otsi(Solm s, int sisu){
    if(s==null)return false;
    if(s.sisu==sisu)return true;
    if(otsi(s.vasak, sisu))return true;
    if(otsi(s.parem, sisu))return true;
    return false;
  }
```

Otsimist annaks optimeerida, eeldades, et suuremad väärtused asuvad paremal ja väiksemad vasakul. Näiliselt tunduks tegemist olema ainult paari väikese võrdluse lisamisega, kuid suuremate

andmemahtude korral võib kiiruse võit olla tohutu. Miljoni väärtuse korral kuluks tasakaalustatud puus sobiva väärtuse leidmiseni paarkümmend sammu, täisläbivaatluse korral aga keskeltläbi pool miljonit.

Järgnevalt siiski otsingunäide täisläbivaatluse kohta. Otsingut kiirendavad võrdlused võiks lugeja mõttes siia juurde lisada.

```
public class Solmetest6{
  public static void tryki(Solm s){
    if(s==null){return;}
    tryki(s.vasak);
    System.out.print(s.sisu+" ");
    tryki(s.parem);
  }

  public static void lisa(Solm s, int sisu){
    if(s.sisu==sisu){return;}
    if(sisu>s.sisu){
      if(s.parem==null){
        s.parem=new Solm();
        s.parem.sisu=sisu;
      } else {
        lisa(s.parem, sisu);
      }
    }
    if(sisu<s.sisu){
      if(s.vasak==null){
        s.vasak=new Solm();
        s.vasak.sisu=sisu;
      } else {
        lisa(s.vasak, sisu);
      }
    }
  }

  public static boolean otsi(Solm s, int sisu){
    if(s==null)return false;
    if(s.sisu==sisu)return true;
    if(otsi(s.vasak, sisu))return true;
    if(otsi(s.parem, sisu))return true;
    return false;
  }

  public static void main(String[] argumendid){
    Solm s=new Solm();
    s.sisu=3;
    int[] arvud={4, 3, 3, 1, 9, 8, 0};
    for(int nr=0; nr<arvud.length; nr++){
      lisa(s, arvud[nr]);
    }
    tryki(s);
    if(otsi(s, 7)){
      System.out.println("Leiti");
    } else {
      System.out.println("Ei leitud");
    }
  }
}
```

Kokkuvõtteks

Kahendpuu võimaldab andmeid mällu järjestatult paigutada ning soovitud väärtusi kiiremini lugeda kui ahelas see võimalik oleks. Puu läbimiseks või andmete lisamiseks kasutatakse rekursiivseid algoritme.

Ülesandeid

Andmepuu

* Loo andmetüüp, mis koosneks kuni 30 tähe pikkusest tekstist ning viidast samatüübilisele rakule. Loo sellisest tüübist eksemplar, väärtusta ning trüki tulemus.
* Koosta selle abil lõik ülikooli struktuurist, kusjuures iga raku tekst on vastava üksuse nimi ning viit näitab tase kõrgemal olevale üksusele. Samaaegselt hoia viitu kõigile rakkudele massiivis ning trüki iga raku kohta välja, mis ta nimi ning millistesse kõrgematesse üksustesse ta kuulub.
* Lisaks eelmisele lisa võimalused andmete ja seoste lisamiseks ja muutmiseks klaviatuuri abil.

Trepitud kahendpuu

* Loo andmetüüp, mis koosneks täisarvust ning osutist kahele samatüübilisele rakule. Loo kolm eksemplari, nii et kaks järgmist oleksid esimese küljes. Trüki tulemused esimese kaudu.
* Loo alamprogramm, millele antakse ette osuti vastavat tüüpi rakule. Kui osuti väärtus pole null, siis trüki väärtus ning käivita sama alamprogramm mõlema alaneva osuti puhul.
* Hoolitse trükkimisel, et treppimise abil oleks näha, milline väärtus millise alla kuulub. Väljasta puus leiduvate väärtuste summa.

Morse

* Koosta pliiatsi ja paberi abil morsemärkidest kahendpuu.
* Koosta sarnane puu arvutisse omaloodud andmetüübi ja osutite abil.
  Väljasta oma puu abil jada ... --- ... tulemus.
* Loe oma andmepuust taas välja massiivi igale tähele vastava jada tarvis. Koosta arhiveerimisprogramm tekstifaili morsestamiseks ning taastekstistamiseks.

```
a  .-                 w  .--
b  -...               ä  .-.-
c  -.-.               öõ ---.
d  -..                ü  ..--
e  .                  x  -..-
f  ..-.               y  -.--
g  --.
h  ....               1  .----
i  ..                 2  ..---
j  .---               3  ...--
k  -.-                4  ....-
l  .-..               5  .....
m  --                 6  -....
n  -.                 7  --...
o  ---                8  ---..
p  .--.               9  ----.
q  --.-               0  -----
r  .-.
s  ...                .  .-.-.-
z  --..               ,  --..--
t  -                  ?  ..--..
u  ..-
v  ...-
```

Keele võimalused

Arhiivid, programne koodi uuring ja testimine.

Jar-arhiivid

Kuude ja aastatega koguneb programmilõike, millest ka uute rakenduste koostamisel kasu on. Lühematel juhtudel saab need uue valmiskoodi sisse kopeerida, kuid nõnda lähevad kapseldumisest saadavad hüved kaduma. Avastanud kord lähtefailis vea, tuleb ka kõik muud lõigud läbi käia, kus sama koodijupiga tegemist on. Kui kõik programmid kirjutatakse ühes kataloogis, siis pole suurt muret – klassi nime kaudu saab otse ka koodile ligi. Vähegi suurema programmeerimise korral tuleb koodifaile aga nii hulganisti, et neid üheskoos hoides läheks pilt väga kirjuks ning samuti ei õnnestuks eri ülesannete tarvis loodud koodilõike lahus hoida.

Edaspidi mitmel pool kasutamist leidvate klasside failid võib kokku pakkida ühte Jar-arhiivi, mida siis vajadusel mujalgi tarvitada annab. Näitena klass meetodiga nimega tühja raami avamiseks:

```
C:\kodu\jaagup\0108\k1>type ArhiiviKlass.java
import java.awt.*;
public class ArhiiviKlass{
  public static void avaRaam(String nimi){
    Frame f=new Frame(nimi);
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

ning teine klass loodud meetodi katsetamiseks.

```
public class ArhiiviKasutaja{
  public static void main(String argumendid[]){
    ArhiiviKlass.avaRaam("Tervitusaken");
  }
}
```

Traditsioonilisel moel kompileerides ja käivitades ongi tulemuseks avatud aken.

```
C:\kodu\jaagup\0108\k1>javac Arhiivi*.java

C:\kodu\jaagup\0108\k1>java ArhiiviKasutaja
```

Soovides aknaavamismeetodiga klass arhiivi paigutada, tuleb anda korraldus

```
C:\kodu\jaagup\0108\k1>jar cf ArhiiviKlassid.jar ArhiiviKlass.class
```

Selle tulemusena luuakse uus arhiiv nimega ArhiiviKlassid.jar. Võti c tähendab uue arhiivifaili loomist (create), f – failinime. Kui sellenimeline arhiiv oleks olemas olnud, siis see kirjutataks üle.

Nüüd võib kompileeritud klassi lahtipakitud kuju maha kustutada,

```
C:\kodu\jaagup\0108\k1>del ArhiiviKlass.class
```

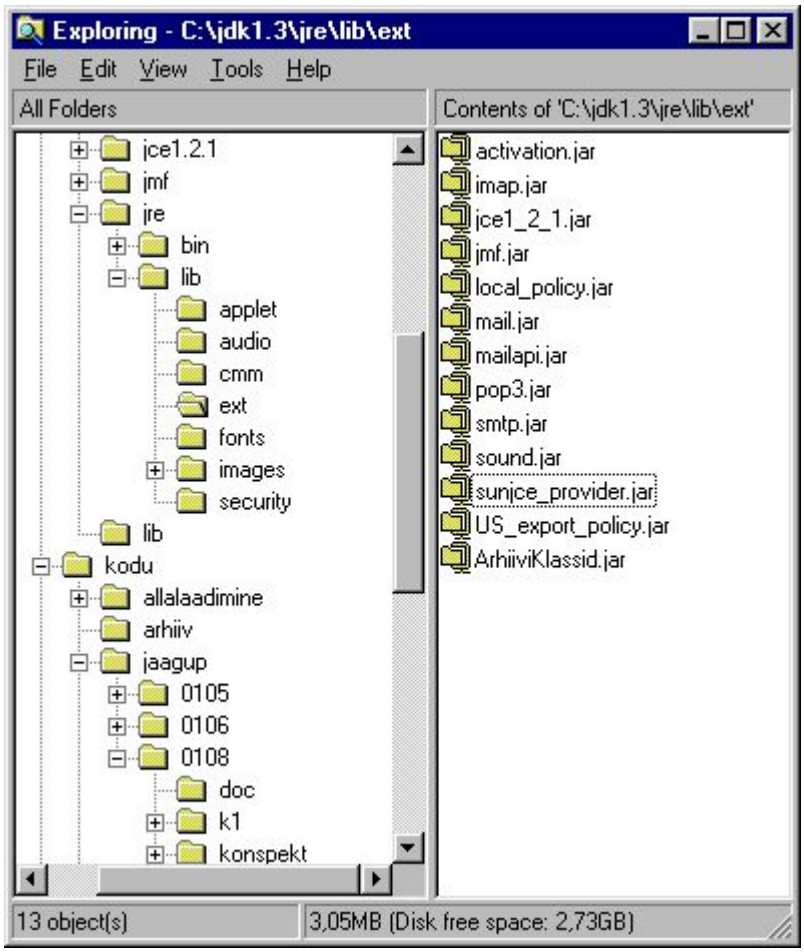
sest andmeid saab ka failist lugeda.

```
C:\kodu\jaagup\0108\k1>java -cp ArhiiviKlassid.jar;. ArhiiviKasutaja
```

Semikoolon ning punkt arhiivi nime järel tähendab, et klasside otsinguteele pannakse lisaks arhiivile ka jooksev kataloog. Muidu jääks käivitamiseks tarvilik ArhiiviKasutaja leidmata, sest seda arhiivis pole.

```
C:\kodu\jaagup\0108\k1>java -cp ArhiiviKlassid.jar ArhiiviKasutaja
Exception in thread "main" java.lang.NoClassDefFoundError: ArhiiviKasutaja
```

Kui loodud arhiivi läheb mitmel pool vaja, siis võib selle paigutada kohta, kust otsitav klass kogu virtuaalmasina piires üles leitakse. Kohaks on Java installeerimiskohast alanev kataloogipuu jre\lib\ext.



Kui nüüd raamiloomiskäsku vajatakse

```
C:\kodu\jaagup\0108\k1>java ArhiiviKasutaja
```

, siis leitakse see vabalt ligi pääsetavast kataloogist paiknevast arhiivist välja ning tulemusena võib jälle loodud raami näha.

Arhiivi võib panna ka käivitatava klassi.

```
C:\kodu\jaagup\0108\k1>jar cf ArhiiviKlassid.jar Arhiivi*.class
```

Nii võib vajaduse korral ka võõras masinas oma arhiivi sees paikneva programmi välja kutsuda.

```
C:\kodu\jaagup\0108\k1>java -cp ArhiiviKlassid.jar ArhiiviKasutaja
```

Arhiivist on kasu ka veebilehe puhul. Atribuudiga archive saab määrata, millisest failist andmed tõmmata. Sel juhul pääseb brauser hulga üksikute klasside kohale sikutamisest ning lehe laadimine võib vähem aega võtta.

```
<html><head><title>Arhiivirakend</title></head>
<body>
 <h2>Rakend arhiivist</h2>

 <applet code="ArhiiviRakend" archive="ArhiiviKlassid.jar"
   height="10" width="10">
 </applet>

</body></html>
```

```
import java.applet.Applet;
public class ArhiiviRakend extends Applet{
  public ArhiiviRakend(){
    ArhiiviKlass.avaRaam("Rakendi aken");
  }
}
```

Paketid

Üle maailma luuakse Java klasse hulgaliselt ning üsna kindlasti satuvad mõnede nimed nendest kokku. Mõistlikke arusaadavaid nimetusi on lihtsalt piiratud hulk. Selle tarvis on Java keelde loodud objektidest/klassidest veel ühe taseme võrra kõrgem grupp: paketid. Standardpakettidest oleme kindlasti kasutanud java.lang-i, java.io või java.awt vahendeid. Selliseid klasside komplekte aga saab rahus ise juurde luua. Põhikomplekti kuuluvad paketid on pandud algama nimega java, põhikomplekti laiendused sõnaga javax. Ülejäänud pakkettide nime algusse aga soovitatakse paigutada neid välja töötava kompanii veebiaadress. Pea igal pakette looval asutusel või isikul on oma väljund veebis ning juba nimede jagamisel hoolitsetakse selle eest, et sama nime alla mitut omanikku ei satuks. Domeeni sees aga tuleb loojatel juba isekeskis hoolitseda, et nimed kattuma ei hakkaks. Nõnda panen pedagoogikaülikoolis loodud paketi nimeks veebiaadressi järgi ee.tpu. Failisüsteemis paketi nimed kattuvad kataloogi nimega, nõnda tuleb siis loodud failid paigutada kompileerimis/käivituskataloogi alamkataloogi ee\tpu.

```
C:\kodu\jaagup\0108\k1>type ee\tpu\Tervitaja.java
package ee.tpu;

public class Tervitaja{
  public static void tervita(){
    System.out.println("Tervist");
  }
}
```

Faili algusse tuleb kirjutada, millise paketi klassiga tegemist on. Et paketist kasutatakse sageli klasse mõne muu programmi töö tarbeks, siis ka siinses näites pole main-meetodit, kust töö käima võiks minna. Nii nagu java.awt.Button'it võime oma töösse lisada, nii saab ka vastloodud klassi oskusi oma hüvanguks kulutada.

```
C:\kodu\jaagup\0108\k1>type Paketikatse.java
import ee.tpu.*;

public class Paketikatse{
  public static void main(String argumendid[]){
    Tervitaja.tervita();
  }
}
```

Faili alguses import-lause hoolitseb, et seal paketis asuvatele klassidele mugavalt ligi pääseks. Programm läheb tööle nagu tavaline muugi Java rakendus.

```
C:\kodu\jaagup\0108\k1>javac Paketikatse.java

C:\kodu\jaagup\0108\k1>java Paketikatse
Tervist
```

Käivituva klassi võib ka paketi sisse panna.

```
C:\kodu\jaagup\0108\k1> type ee\tpu\Alustus.java
package ee.tpu;

public class Alustus{
  public static void main(String argumendid[]){
    Tervitaja.tervita();
  }
}
```

Sellisel juhul saab paketile ligi pääsedes mugavalt seal paikneva programmi käivitada.

```
C:\kodu\jaagup\0108\k1>java ee.tpu.Alustus
Tervist
```

Paketi klassid võib lahedasti üheks arhiiviks kokku pakkida.

```
C:\kodu\jaagup\0108\k1>jar cf tekstipakett.jar ee\tpu\*.class
```

Sellisena piisab installeerimiseks vaid ühe faili sobivasse kohta kopeerimisest ning töö võibki alata.

```
C:\kodu\jaagup\0108\k1>java -cp tekstipakett.jar ee.tpu.Alustus
Tervist
```

Kui tahta arhiivile üle kogu virtuaalmasina ligi pääseda, siis võib selle kopeerida kättesaadavasse jre\lib\ext kataloogi nagu eelmiseski näites.

Soovides oma Java-programmi võõrasse masinasse paigutada, peab omanik enamasti kopeerima sinna hulga faile ning lisaks teadma, millise klassi käivitamisel kogu lugu tööle hakkab. Jar-faili mainfest-osas saab määrata, millise klassi main-meetodist programmi käivitamist alustada tuleb.

```
C:\kodu\jaagup\0108\k1>type lisateave.txt
Main-Class: ee.tpu.Alustus
```

Kui arhiivi loomisel manifest tekstifailist lisada,

```
C:\kodu\jaagup\0108\k1>jar cmf lisateave.txt tekstipakett.jar ee\tpu\*.class
```

siis käivitamisel pannaksegi arhiiv niimoodi tööle, kuidas programmi kirjutaja seda soovinud on.

```
C:\kodu\jaagup\0108\k1>java -jar tekstipakett.jar
Tervist
```

Kes on Jar-failile assotsiatsiooni loonud (või on see vaikimisi tehtud), et käivitamisel lükatakse tööle java intepretaator jar-võtmega ning parameetriks antakse jar-arhiivi nimi, siis tundubki, et tegemist on isekäivituva jar-failiga.

Erindid

Veidi seletusi erindite loomise ja kasutamise kohta.

Probleemist teada andmiseks võime soovitud kohas välja heita erindi. Järgnevad käsud jäetakse täitmata kuni erind lendab virtuaalmasinast välja või püütakse kinni. Üldjuhul tuleb meetodi päises näidata throws-teatega, kui meetodist võib erindeid välja tulla.

```
public class Erind5{
  public static void main(String argumendid[]) throws Exception {
    int vanus=8;
    if(vanus<10)throw new Exception("Liiga noor");
    System.out.println("Tere tulemast pikamaajooksule");
  }
}
```

väljund

```
Exception in thread "main" java.lang.Exception: Liiga noor
        at Erind5.main(Erind5.java:4)
```

vastab täiesti ootustele. Klassi Erind4 neljandal koodireal saadeti lendu erind ning edaspidised käsud jäid täitmata.

Kui soovitakse heita erind, mille tekkimist ei pea deklareerima, siis tuleb kasutada RuntimeExceptioni või selle alamklassi. Enamjaolt kuuluvad RuntimeExceptioni alla eriolukorrad, mis võivad ette tulla väga paljudes kohtades (nt. jagamine nulliga, massiivi piiride ületamine) ning mida deklareerides peaks peaks siis pea kõikide meetodite juurde deklaratsioonid kirjutama. Nagu näha, siin kasutatakse RuntimeExceptioni ning main-meetodis pole tarvis üles tähendada, et throws Exception või throws RuntimeException. Samas kui sinna see siiski kirjutada, siis probleeme sellest ei tekiks.

```
public class Erind6{
  public static void main(String argumendid[]) {
    int vanus=8;
    if(vanus<10)throw new RuntimeException("Liiga noor");
    System.out.println("Tere tulemast pikamaajooksule");
  }
}
```

Jällegi väljund vastavalt ootustele

```
Exception in thread "main" java.lang.RuntimeException: Liiga noor
        at Erind6.main(Erind6.java:4)
```

, neljandal real avastati, et pikema jooksmise tarvis on vanust liialt vähe.

Omaloodud erind

Kui soovime selliseid eriolukordi teada anda ja nendele reageerida, milliseid standardvahendites kirjas pole, siis võime luua oma erindialamklassi. Sellisel võime kergemini reageerida vastavalt tekkinud probleemile. Ka erindi väljatrükil näidatakse, millisest klassist erind pärit on.

```
class VanuseErind extends Exception{}

public class Erind7{
  public static void main(String argumendid[]) throws VanuseErind{
    int vanus=8;
    if(vanus<10)throw new VanuseErind();
    System.out.println("Tere tulemast pikamaajooksule");
  }
}

/* väljund:
 Exception in thread "main" VanuseErind
        at Erind7.main(Erind7.java:4)
*/
```

Soovides erindi kinnipüüdjale anda selgituse probleemi kohta, on üheks võimaluseks katta üle erindi meetod getMessage().

```
class VanuseErind2 extends Exception{
  public String getMessage(){
```

```
    return "Vanus ei sobinud";
  }
}

public class Erind7a{
  public static void main(String argumendid[]) throws Exception{
    int vanus=8;
    if(vanus<10)throw new VanuseErind2();
    System.out.println("Tere tulemast pikamaajooksule");
  }
}
```

Nõnda jõuab koodi sisse kirjutatud selgitus veateatena ekraanile või võidakse seda muul moel veatöötluses arvestada.

```
Exception in thread "main" VanuseErind2: Vanus ei sobinud
         at Erind7a.main(Erind7a.java:4)
```

Kõige viisakam võimalus programmi sees omaloodud erindile teadet kaasa panna on luua erindile uus sõneparameetriga konstruktor mis omakorda ülemklassi vastava konstruktori välja kutsub. Ülemklass hooliseb, et getMessage teate välja annaks. Et loodud klassi veel omakorda ilusti laiendada annaks, tuleks siia ka parameetriteta konstruktor kirjutada, mis ülemklassi parameetriteta konstruktori välja kutsuks. Põhjus selles, et vaikimisi kutsutakse uue isendi loomisel alati välja ülemklassi parameetriteta konstruktor, kui parajasti käivitatava konstruktori esimese käsuna pole välja kutsutud mõni muu konstruktor. Kui juhtub aga, et ülemklassil parameetriteta konstruktorit pole ning mõnda muud ka välja ei kutsuta, siis antakse veateade. Siinse näite VanuseErind3 juures on parameetriteta konstruktori kirjutamisest loobutud, arvates et sellest erindist võib luua vaid programmeerija määratud teatega isendeid ning et teateta alamklasse siia enam ei looda.

```
class VanuseErind3 extends Exception{
  VanuseErind3(String teade){
    super(teade);
  }
}

class Erind7b{
  static void main(String argumendid[]) throws Exception{
    int vanus=8;
    if(vanus<10)throw new VanuseErind3("Pole veel 10. aastane");
    System.out.println("Tere tulemast pikamaajooksule");
  }
}

/* väljund:
Exception in thread "main" VanuseErind3: Pole veel 10. aastane
        at Erind7b.main(Erind7b.java:4)
*/
```

Lõpuplokk finally

Lisaks try-le ning cactch-i(de)le võib katsendile lisada finally-ploki, mis täidetakse sõltumata sellest, kas uuritavas piirkonnas tekkis probleeme või kas neid töödeldi. Sinna kirjutatakse enamasti käsud, mis tuleb alati täita. Näiteks voogude sulgemine või andmebaasiühenduse katkestamine, mis tuleb ressursside vabastamiseks ikka läbi viia, ükskõik, kas saabuvad andmed vastasid ootustele või mitte. Järgnevates näidetes saadetakse lõpuplokis tervitused kõigile kohalolijatele sõltumata sellest kui noored või vanad nad parajasti on. Samuti tegevus, mis oleks patt ära jätta.

```
class Erind8{
  static void main(String argumendid[]) throws VanuseErind{
    int vanus=8;
    try{
     if(vanus<10)throw new VanuseErind();
     System.out.println("Tere tulemast pikamaajooksule");
    } catch(VanuseErind v){
      v.printStackTrace();
    }
    finally {
      System.out.println("Tervitus igale kohalolijale");
    }
  }
}
```

Kõigepealt püüti catch-osas erind kinni ning trükiti selle andmed, edasi tervitati finally-plokis kohalolijaid.

```
/* väljund:
VanuseErind
         at Erind8.main(Erind8.java:5)
Tervitus igale kohalolijale
*/
```

Ka juhul, kui katsendiplokis püütakse return-i abil meetodist väljuda, ei pääse lõpuploki käivitamisest.

```
public class Erind8a{
  public static void main(String argumendid[]){
    int vanus=78;
    try{
     if(vanus>70)return;  //püütakse meetodist väljuda;
     if(vanus<10)throw new VanuseErind();
     System.out.println("Tere tulemast pikamaajooksule");
    } catch(VanuseErind v){
      v.printStackTrace();
    }
    finally {
      System.out.println("Tervitus igale kohalolijale");
    }
  }
}

/* väljund:
Tervitus igale kohalolijale
*/
```

Kui kõik tingimused sobivad ning probleeme ei teki, ka siis täidetakse lõpuplokk.

```
class Erind8b{
  static void main(String argumendid[]){
    int vanus=27;
    try{
     if(vanus<10)throw new VanuseErind();
     System.out.println("Tere tulemast pikamaajooksule");
    } catch(VanuseErind v){
      v.printStackTrace();
    }
    finally {
      System.out.println("Tervitus igale kohalolijale");
    }
  }
}

/* väljund:
Tere tulemast pikamaajooksule
Tervitus igale kohalolijale

*/
```

Erindi võib pärast töötlemist edasi saata sarnaselt throw käsuga nagu algselgi juhul. Sellist moodust läheb vaja, kui samale probleemile tuleb reageerida mitmes kohas.

```
public class Erind9{
  public static void main(String argumendid[]) throws VanuseErind{
    int vanus=8;
    try{
     if(vanus<10)throw new VanuseErind();
     System.out.println("Tere tulemast pikamaajooksule");
    } catch(VanuseErind v){
      v.printStackTrace();
      throw v;
    }
  }
}

/* väljund:
VanuseErind
        at Erind9.main(Erind9.java:5)
Exception in thread "main" VanuseErind
        at Erind9.main(Erind9.java:5)

*/
```

Ka siis ei pääse finally-ploki täitmisest.

```
public class Erind9a{
  public static void main(String argumendid[]) throws VanuseErind{
    int vanus=8;
    try{
     if(vanus<10)throw new VanuseErind();
     System.out.println("Tere tulemast pikamaajooksule");
    } catch(VanuseErind v){
      v.printStackTrace();
      throw v;
    } finally {
      System.out.println("Tervitus igale kohalolijale");
    }
  }
}

/* väljund:
VanuseErind
        at Erind9a.main(Erind9a.java:5)
Tervitus igale kohalolijale
Exception in thread "main" VanuseErind
        at Erind9a.main(Erind9a.java:5)
*/
```

Kloonimine

Objektide puhul rõhutatakse kolme eristatavat omadust: tüüp, andmed ja identiteet. Tüübi alla kuuluvad klassi kirjeldamisel loodud käsklused ehk meetodid ning väljade ehk muutujate loetelu. Andmed on igal objektil omad: olgugi et mõlemad isendid võivad olla tüübist Punkt, nende x-i ja y-i väärtused on üldjuhul erinevad. Ning isegi, kui mõne punkti koordinaadid peaksid teise punkti koordinaatidega kattuma, ei ole tegemist sama objektiga, sest neil on erinev identiteet.

Soovides isendist luua koopiat, tuleb teha koopia kõigist tema väljadest. Üheks võimaluseks on koostada koopia loomiseks eraldi käsklus. Seal luua konstruktori abil uus sama tüüpi isend. Anda käskude ja parameetrite abil uuele isendile soovitud väärtused ning siis returni abil tagastada osuti uuele objektile.

Vaeva vähendamiseks on loodud liides Cloneable, mille abil on lubatud objektidest luua "mehhaaniline koopia". See tähendab, et uude koostatavasse objekti kantakse üle lihtsalt kõik vana objekti väljade väärtused. Vastava toimingu läbiviijana saab kasutada kõikide klasside ülemklassi Object protected-juurdepääsuga käsklust nimega clone, mis loobki just uue objekti ja kopeerib sinna kõikide väljade väärtused.

```
class Kloon1 implements Cloneable{
  String nimi;
  int mass;
  public Object clone(){
    Object o=null;
    try{
      o=super.clone();
    }catch(CloneNotSupportedException e){}
    return o;
  }
}
```

Et liides Cloneable määrab meetodi clone tagastustüübiks Object, siis tegelikuks kasutamiseks tuleb uus eksemplar tüübimuundusega sobivasse tüüpi ehk algse isendiga samasse tüüpi määrata.

Järgnevalt võibki näha, kuidas algsest eksemplarist kloon luuakse nõnda, et kõik väärtused samaks jäävad, kuid nendele eraldi mälupesad eraldatakse. Ning kuna pärast kloonimist on tegemist eraldi objektidega, siis massi muutmine ühe isendi juures ei muuda teise isendi massi.

```
public class Kloon1Test{
   public static void main(String[] argumendid){
     Kloon1 k1=new Kloon1();
     k1.nimi="Dolly";
     k1.mass=80;
     Kloon1 k2=(Kloon1)k1.clone();
     k2.mass=90;
     System.out.println(k1.mass+" "+k2.mass);
```

```
   }
}

/*
D:\kodu\0312>java Kloon1Test
80 90
*/
```

Kui klassil on Cloneable-liidese kaudu kord juba kloonimine lubatud, siis edasi on ka kõik vastava klassi alamklasside isendid kloonitavad. Et Kloon1 sai eelmises näites kloonitavaks muudetud, siis vastav õigus kehtib ka Kloon3-e isendite kohta.

```
class Kloon3 extends Kloon1{
  int vanus;
}
```

Ehkki nüüd kloonitakse isendit tüübist Kloon3, osatakse ülemklassi meetodite väljakutse abil ikkagi kõikidest väljadest koopiad teha.

```
public class Kloon3Test{
  public static void main(String[] argumendid){
    Kloon3 k=new Kloon3();
    k.mass=60;
    k.vanus=2;
    Kloon3 k2=(Kloon3)k.clone();
    System.out.println(k2.mass+" "+k2.vanus);
  }
}

/*
D:\kodu\0312>java Kloon3Test
60 2
*/
```

Süviti kloonimine.

Lihttüüpidega on asi selge: kui isendil olid väljad ning nendel väärtused, siis kloonimise puhul loodi uus isend, kus iga välja jaoks oli loodud uus mälupesa. Välja mälupesal paiknev väärtus ongi selle välja väärtus.

Struktuurtüüpidega on lugu keerulisem. Taoline väli võib viidata andmebaasiühendusele, avatud aknale või näiteks failile kusagil kaugel Internetis. Kui isendil ka on vastav väli küljes, siis välja kloonimine ei tee veel eraldi asetsevat objekti juurde. Tulemuseks on lihtsalt mõlema isendi küljes paiknevad osutid, mis näitavad samale tegelikule objektile. Kui nüüd ühe kloonitud isendi osuti kaudu taolise väljaspool asuva objekti parameetreid muuta, siis muutus paistab mõlemast kloonimisega seotud isendist. Samuti nagu ühest aknast puuoksa külge seotud pekitükk talvel lindude söötmiseks paistab ka teisest aknast, sest tegemist on ikka ühe ja sama oksa ning pekiga. Ning kui pekk hakkab otsa lõppema, siis paistab see samuti mõlemast aknast kätte.

Mõnikord aga soovitakse, et kloonimise järel oleksid ka kummastki objektist viidatavad objektid teineteisest sõltumatud. Selleks tuleb ka alanevad objektid kloonida, mis mõnikord on võimalik, teinekord mitte. Siin näites püütakse kirjeldada sugupuud. Et isik ning vanemad on sama liiki ning vastav klass ise kloonitav, siis õnnestub vajadusel ka kogu sugupuu andmed kloonida. Kloonimisel tehakse kõigepealt koopia enesest ning siis ka kummastki esivanemast nende olemasolul.

```
      if(isa!=null){
        k.isa=(Kloon2)isa.clone();
      }
```

Et esivanem on samuti tüübist Kloon2, siis ka tema kloonimisel minnakse omakorda alanejaid esivanemaid kloonima.

Sarnaselt liigutakse alamelementide sisse ka väärtusi välja trükkides toString meetodi abil.

```
class Kloon2 implements Cloneable{
  String nimi;
  Kloon2 isa;
  Kloon2 ema;
  public Object clone(){
```

```
    Object o=null;
    try{
      o=super.clone();
      Kloon2 k=(Kloon2)o;
      if(isa!=null){
        k.isa=(Kloon2)isa.clone();
      }
      if(ema!=null){
        k.ema=(Kloon2)ema.clone();
      }
    }catch(CloneNotSupportedException e){}
    return o;
  }
  public String toString(){
    String s=nimi+"(";
    if(isa!=null){ s+=isa.toString(); }
    s+=", ";
    if(ema!=null){ s+=ema.toString(); }
    s+=")";
    return s;
  }
}
```

Testimisel luuakse kõigepealt väike sugupuu, kus peale Maasu leiduvad ka tema isa, ema ning isaisa. Edasi sugupuu kloonitakse ning uue puu puhul määratakse isaisa nimeks Puhvik.

```
public class Kloon2Test{
   public static void main(String[] argumendid){
     Kloon2 k1=new Kloon2();
     k1.nimi="Maasu";
     k1.isa=new Kloon2();
     k1.isa.nimi="Punik";
     k1.ema=new Kloon2();
     k1.ema.nimi="Maara";
     k1.isa.isa=new Kloon2();
     k1.isa.isa.nimi="Puhvel";
     System.out.println(k1);
     Kloon2 k2=(Kloon2)k1.clone();
     k2.isa.isa.nimi="Puhvik";
     System.out.println(k1);
     System.out.println(k2);
   }
}
```

Kui süviti kloonimist poleks toimunud, siis oleks mõlemal kloonil isaisa objekt sama ning isaisa nime muutmine ühe isendi kaudu muutnuks ta nime ka teise isendi poolt vaadates. Et aga kloonimine toimus süviti, siis klooniti ka vanavanemad ning nimemuutus ühe muutuja kaudu jättis algse vanaisa nime muutmata.

```
/*
D:\kodu\0312>java Kloon2Test
Maasu(Punik(Puhvel(, ), ), Maara(, ))
Maasu(Punik(Puhvel(, ), ), Maara(, ))
Maasu(Punik(Puhvik(, ), ), Maara(, ))
*/
```

Ülesandeid

* Koosta klass Punkt kahe koordinaadiga.
* Loo eksemplar, testi.
* Loo teine muutuja näitamaks samale eksemplarile. Muuda andmeid, testi väljatrükki mõlema muutuja kaudu.
* Realiseeri Punktil liides Cloneable, kata üle meetod clone loomaks koopia.
* Loo muutuja, mis näitaks esialgse punkti koopiale. Testi väljade muutmist, trüki tulemused ekraanile.

* Loo klass Kujund, mis sisaldab eneses nime ning Punktide nimistu. Klassile käsklused punktide lisamiseks ja muutmisks. Testi.
* Muuda Kujund Kloonitavaks. Kloonitakse nimistu, kuid veel mitte üksikuid punkte (nimistu käsk clone). Testi lisamist ja muutmist.
* Paranda koodi nõnda, et ka üksikud punktid kujundi nimistu sees kloonitaks.

Klasside uuring koodiga

Programmeerija üldjuhul teab, millised käsklused milliste parameetritega ta on oma koodi kirjutanud. Või kui täpselt ei tea, siis vaatab koodist või dokumentatsioonist järele. Kui aga pole lähtekoodi või dokumentatsiooni käepärast, samuti juhtudel, kui soovitakse automaatselt statistikat teha või mõningaid käsklusi testida, aitab välja võimalus programmikäskude abil olemasolevat klassi uurida. Järgnevas näites on võetud ette klass String ning küsitud välja kõikide sealsete meetodite nimed.

```
import java.lang.reflect.Method;
public class Klassiuuring1{
  public static void main(String[] argumendid){
    String s="Tere";
    Class c=s.getClass();
    System.out.println("Klassi nimi: "+c.getName());
    Method[] m=c.getMethods();
    System.out.println("Meetodid:");
    for(int i=0; i<m.length; i++){
      System.out.println("  "+m[i].getName());
    }
  }
}

/*
Klassi nimi:java.lang.String
Meetodid:
  valueOf
  valueOf
  valueOf
  valueOf
  valueOf
  valueOf
  valueOf
  valueOf
  valueOf
  copyValueOf
  copyValueOf
  wait
  wait
  wait
  getClass
  notify
  notifyAll
  hashCode
  compareTo
  compareTo
  equals
  toString
  length
  charAt
  getChars
  getBytes
  getBytes
  getBytes
  equalsIgnoreCase
  compareToIgnoreCase
  regionMatches
  regionMatches
  startsWith
  startsWith
  endsWith
  indexOf
  indexOf
  indexOf
  indexOf
  lastIndexOf
  lastIndexOf
  lastIndexOf
  lastIndexOf
  substring
  substring
  concat
  replace
  toLowerCase
  toLowerCase
  toUpperCase
```

```
  toUpperCase
  trim
  toCharArray
  intern
*/
```

Iga meetodi sisse saab ka mõnevõrra põhjalikumalt piiluda. Siin on vaadatud lisaks meetodite nimedele ka nende parameetrid koos tüüpidega. Samuti meetoditest välja heidetavate erinditüüpide loetelu.

```
import java.lang.reflect.Method;
public class Klassiuuring2{
  public static void main(String[] argumendid){
    String s="Tere";
    Class c=s.getClass();
    Method[] m=c.getMethods();
    System.out.println("Meetodid:");
    for(int i=0; i<m.length; i++){
      Class[] parameetrid=m[i].getParameterTypes();
      Class[] erindid=m[i].getExceptionTypes();
      Class tagastus=m[i].getReturnType();
      System.out.print(tagastus.getName()+" "+m[i].getName());
      for(int j=0; j<parameetrid.length; j++){
        System.out.print(" "+parameetrid[j].getName());
        //[I parameetri nimena tähistab näiteks täisarvude massiivi
      }
      for(int j=0; j<erindid.length; j++){
        System.out.println(" *"+erindid[j].getName());
      }
      System.out.println();
    }
  }
}
```

Väljund võib esiotsa ebatavaline tunduda, kuid kui dokumentatsioonist klassi Class meetodit getName() uurida, leia sealt, et massiivide tähistamiseks on omaette moodus olemas. Ning

```
java.lang.String valueOf [C int int
```

tähendab lihtsalt, et klassis leidub java.lang.String tüüpi objekti väljastav käsklus valueOf, mis saab parameetriteks tähemassiivi ning kaks täisarvu. Nii nagu [C tähistab tähemassiivi, nii tähistaks [[I kahemõõtmelist täisarvumassiivi.

```
/*
Meetodid:
java.lang.String valueOf [C int int
java.lang.String valueOf [C
java.lang.String valueOf java.lang.Object
java.lang.String valueOf long
java.lang.String valueOf boolean
java.lang.String valueOf char
java.lang.String valueOf int
java.lang.String valueOf float
java.lang.String valueOf double
java.lang.String copyValueOf [C int int
java.lang.String copyValueOf [C
void wait *java.lang.InterruptedException

void wait long int *java.lang.InterruptedException

void wait long *java.lang.InterruptedException

java.lang.Class getClass
void notify
void notifyAll
int hashCode
int compareTo java.lang.String
int compareTo java.lang.Object
boolean equals java.lang.Object
java.lang.String toString
int length
char charAt int
void getChars int int [C int
[B getBytes
[B getBytes java.lang.String *java.io.UnsupportedEncodingException

void getBytes int int [B int
boolean equalsIgnoreCase java.lang.String
```

```
int compareToIgnoreCase java.lang.String
boolean regionMatches int java.lang.String int int
boolean regionMatches boolean int java.lang.String int int
boolean startsWith java.lang.String
boolean startsWith java.lang.String int
boolean endsWith java.lang.String
int indexOf int int
int indexOf int
int indexOf java.lang.String
int indexOf java.lang.String int
int lastIndexOf java.lang.String int
int lastIndexOf int int
int lastIndexOf java.lang.String
int lastIndexOf int
java.lang.String substring int int
java.lang.String substring int
java.lang.String concat java.lang.String
java.lang.String replace char char
java.lang.String toLowerCase java.util.Locale
java.lang.String toLowerCase
java.lang.String toUpperCase
java.lang.String toUpperCase java.util.Locale
java.lang.String trim
[C toCharArray
java.lang.String intern

*/
```

Et Javas tulevad pärimisel käsklused kaasa, siis saab eristada klassis eneses loodud käsklusi ning kaasatulnud käsklusi. Esimesed neist saab kätte käsuga getDeclaredMethods(). Tahtes kätte saada kogu pärimispuu jooksul loodud käsklused, võetakse järgnevas näites pärast konkreetse klassi käskluste uurimist ette tema ülemklass ning nõnda kuni juureni välja. Ehk kuni enam ülemklassi võtta ei ole ning meetod getSuperclass väljastab tühitunnuse null.

```
    Class c=p.getClass();
    while(c!=null){
      ...
      c=c.getSuperclass();
    }
```

Kui tegemist pikema pärimishierarhiaga, siis võib käskude loend õige pikaks minna. Siin näidatud lihtsalt kood ja mõned esimesed käsud.

```
import java.lang.reflect.Method;
import java.awt.Panel;
public class Klassiuuring3{
  public static void main(String[] argumendid){
    Panel p=new Panel();
    Class c=p.getClass();
    while(c!=null){
      System.out.println("Klassi nimi: "+c.getName());
      Method[] m=c.getDeclaredMethods();
      System.out.println("Meetodid:");
      for(int i=0; i<m.length; i++){
        System.out.println("  "+m[i].getName());
      }
      c=c.getSuperclass();
    }
  }
}

/*
Klassi nimi: java.awt.Panel
Meetodid:
  constructComponentName
  addNotify
  getAccessibleContext
Klassi nimi: java.awt.Container
Meetodid:
  add
  ...
*/
```

Käivitamine nime järgi

Ehkki harva, kuid vahel siiski tuleb ette olukord, kus tahetakse loodava objekti tüüp määrata alles programmi käivitamise ajal. Üheks taoliseks näiteks on juhtum, kus vanema interpretaatoriversiooniga tahetakse kasutada üht klassi, uuema puhul teist. Kui aga uuem klass koodi sisse kirjutada, siis võib verifeerija teatada tundmatust klassist ning kogu koodi käivitamise keelata.

Samuti läks nime järgi eksemplari loomist vaja olukorras, kus sooviti funktsioon joonistada mitmesuguste kasutaja sisestatud valemite järgi. Et aga Javas eval-käsklus koodilõigu käivitamiseks puudub, siis aitas hädast välja kompilaator. Kasutaja sisestatud avaldise põhjal kompileeriti kokku uus klass. Edasi tuli joonistamise tarbeks klassist eksemplar luua. Vana klassi nime ei saanud kasutada, sest see võis olla juba puhvrisse laetud ning nõnda oleks graafik tulnud vana avaldise alusel. Sobis aga lahendus, kus igal korral kompileeriti kokku uue nimega klass ning loodi vastava nimega klassist eksemplar.

Teine võimalus olnuks kasutada ClassLoaderit ning kompilaatori pakutud baitkoodist sealtkaudu klassi eksemplar luua. Rakendis aga polnud turvanõuete tõttu ClassLoaderi kasutamine võimalik, nime järgi klassi eksemplaride loomine aga küll.

Järgnevas näites paistab, kuidas võib nime järgi luua klassi eksemplari. Samuti küsitakse klassilt nime järgi välja meetod ning selle kaudu sisestatakse loodud isendile väärtus.

```
import java.lang.reflect.*;
public class MeetodiTest{
  public static void main(String[] argumendid) throws Exception{
    Class c=Class.forName("Arv");
    Arv a1=(Arv)c.newInstance();
    Method m=c.getMethod("paneSisu", new Class[]{int.class});
    m.invoke(a1, new Object[]{new Integer(7)});   System.out.println(a1.kysiSisu());
  }
}

class Arv{
  int a;
  public void paneSisu(int uusArv){
    a=uusArv;
  }
  public int kysiSisu(){
    return a;
  }
}
```

Ülesandeid

Klasside ja objektide uuring.

* Tutvu klassiuuringu näidetega.
* Trüki välja klassi java.lang.Integer meetodid.
* Käivita meetod parseInt käskluse invoke abil.

JUnit, automaattestimine

Kord valmiskirjutatud koodi haldamisele - muudatuste tegemisele ja vigade parandamisele pidada minema hulga enam ressursse kui koodi enese loomisele. Programmi töö korrektsuse kontrolliks on välja mõeldud hulgem tehnikaid, üks nendest on automaattestimine. Automaattestidega kontrollitakse üldjuhul üksikute alamprogrammide või väiksemate koodilõikude tööd, kuid on võimalik ka suuremate toimingute kontroll. Ekstreemprogrammeerimise nimelise tehnika juures peetakse programmi tööd kontrollivaid teste vähemasti sama tähtsaks kui programmi ennast, kuid oma roll on neil muudelgi puhkudel. Kui automaattestid kirjutatakse enne kui kood, siis on testid heaks abivahendiks soovitavate funktsionaalsuste kavandamisel ja kirja panemisel. Samuti saab nende kaudu küllalt hästi kontrollida näiteks kusagil eemal kolmanda osapoole loodud tükkide sobivust süsteemi.

Teise tähtsa omadusena saab automaattestide abil kontrollida soovitud funktsionaalsuse säilimist süsteemis tehtud muudatuste järel. Vähegi suuremas rakenduses tekivad soovimatud kõrvalnähud lubamatult kergesti. Automaattestide abil aga õnnestub pärast uue mooduli lisamist või vana muutmist ülejäänud süsteemi toimimine kontrollida vähemasti testidesse kirjutatud tavajuhtumite puhul.

Java puhul on tõenäoliselt levinuimaks automaattestimisvahendiks tasuta kasutatav JUnit. Programmeerija ülesandeks on kirjutada testitava rakenduse funktsionaalsuste kontrolliks hulk terviklikke alamprogramme. JUniti keskkond käivitab need ning peab statistikat, milliste töö toimus edukalt milliste oma mitte. Kontrollimiseks võib olla koostatud kümneid, sadu või isegi tuhandeid koodilõike kuid rakendus loetakse toimivaks juhul, kui ta kõikide testide nõudmistele vastab.

JUniti testprogrammid kirjutatakse tavaliste Java klassidega, vaid ülemklassiks peab olema paketist junit.framework kaasa tulnud TestCase. Tavalisi meetodeid võib klassi lisada nõnda nagu igaüks soovib. Testolukordi tähistavad meetodid peavad algama sõnaga test. Meetodi tagumise poole nimed võib programmeerija ise määrata. Tõenäoliselt eelpool kirjeldatud klassi meetodite vaatamise ja käivitamise vahendi abil suudab JUniti testimissüsteem panna meetodid tööle ilma, et nende nimed peaksid kuhugile mujale olema sisse kirjutatud.

Olukorda testiv funktsioon üldjuhul teeb oma toimingud ning viimaste käskude hulgas assert-lausega kontrollitakse, kas saavutati eeldatav olukord. JUnit-kestprogrammi ülesandeks on meetodid käivitada ning lõpus kokku lugeda, milliste testkäskluste käivitamisel probleemid tekkisid. Siin esimeses lihtsamas näites küll vaid kontrollitakse, kas kolm pluss kaks on ikkagi viis.

Testkeskkonna vaatamiseks ja juhtimiseks on loodud mitu kasutajaliidest. Siin näites pannakse tööle kõige vähem ressursse nõudev neist ehk tekstipõhine. Olemas on veel nii awt kui Swingi graafikal põhinevad liidesed.

```
import junit.framework.*;

public class Minitest extends TestCase{
  public void testArvutus(){
    assertEquals(3+2, 5);
  }
  public static void main(String[] argumendid){
    junit.textui.TestRunner.run(new TestSuite(Minitest.class));
  }
}
```

JUniti kompileerimiseks tuleb nende kodulehelt http://www.junit.org/ alla laadida arhiiv, mille ühe osana paiknev fail junit.jar sisaldab tarvilikke klasse ning on hädavajalik kompileerimisel ja käivitamisel. Nagu õpetuses öeldud, ei tohi junit.jar-i paigutada lisamoodulite juurde jre/lib/ext kataloogi, vaid peab parameetriga käsureal kaasa andma.

Väljundist võib näha, et üks ja ainuke testitoiming sooritati edukalt.

```
/*
C:\java\jaagup\testid>javac -classpath junit.jar;. Minitest.java

C:\java\jaagup\testid>java -cp junit.jar;. Minitest
.
Time: 0,01

OK (1 test)
*/
```

Üldjuhul koostatakse testimoodulid konkreetsete moodulite, klasside, alamprogrammide või muul viisil piiritletud tervikute kontrolliks. Enne testfunktsioonide käivitamist võib mõnikord tarvilik olla rakenduse osa testimiseks ette valmistamine ning pärast testimise lõppu taas seiskamine. Selle tarvist võib testmoodulis üle katta funktsioonid setUp ning tearDown. Siis pole igas testkäskluses eraldi vaja nende toimetustega tegelda.

Järgnevas näites kontrollitakse bittväärtuste hoidlana toimiva java.util.BitSeti tööd. BitSeti puhul saab iga elemendi puhul määrata, kas sealne bitt on püsti või mitte. Algseadistamise juures määratakse bitt number 2 püstiseks. Edasiste funktsioonide juures kontrollitakse, kas bitid paiknevad ettearvatult.

```
import junit.framework.*;
import java.util.BitSet;

public class Bitihulgatest extends TestCase{
  BitSet b=new BitSet();
  public void setUp(){
    b.set(2, true);
    System.out.println("Testi algus");
  }
  public void tearDown(){
    System.out.println("Testi ots");
  }
  public void testPysti(){
    assertTrue(b.get(2));
  }
```

```
  public void testPikali(){
    assertFalse(b.get(1));
  }
  public static void main(String[] argumendid){
    junit.textui.TestRunner.run(new TestSuite(Bitihulgatest.class));
  }
}
/*
C:\java\jaagup\testid>javac -classpath junit.jar;. Bitihulgatest.java

C:\java\jaagup\testid>java -cp junit.jar;. Bitihulgatest
.Testi algus
Testi ots
.Testi algus
Testi ots

Time: 0,01

OK (2 tests)

*/
```

Testide kogum

Suurema rakenduse puhul on lihtsam kirjutada iga tervikliku osa test omaette faili. Rakenduse täiskontrolli ajal tuleb aga kõik selle kohta käivad testid käivitada. Nõnda on võimalik koostada testidest kogum ning see tervikuna käivitada.

```
import junit.framework.TestSuite;
public class TestiKogum{
  public static void main(String[] argumendid){
    TestSuite kogum=new TestSuite();
    kogum.addTest(new TestSuite(Minitest.class));
    kogum.addTest(new TestSuite(Bitihulgatest.class));
    junit.textui.TestRunner.run(kogum);
  }
}

/*
C:\java\jaagup\testid>javac -classpath junit.jar;. TestiKogum.java

C:\java\jaagup\testid>java -cp junit.jar;. TestiKogum
..Testi algus
Testi ots
.Testi algus
Testi ots

Time: 0,02

OK (3 tests)
*/
```

Ülesandeid

* Tutvu näidetega. Lae ja paki lahti JUnit. http://www.junit.org
* Lisa Minitesti testfunktsioon, mis kontrolliks, et nimes "Juku" on 4 tähte.
* Loo omaette testklass, mille initsialiseerimisel luuakse fail ning kirjutatakse sinna kümme juhuslikku arvu; arvude summa jäetakse meelde. Lisa testfunktsioon kontrollimaks, kas failis on ikka kümme rida ning teine jälgimaks, et arvude summa on muutumatu.
* Testi sulgemisel kustutatakse fail.

* Loo liides nimede lisamiseks, otsimiseks ja kustutamiseks.
* Loo liidest realiseeriv klass.
* Loo test klassi oskuste mitmekülgseks kontrolliks.

* Loo liidese põhjal klass, mis kasutaks faili asemel andmebaasi.
* Hoolitse, et loodud testid töötaksid ka uue klassi puhul.

Tarkvara hindamine.

Maksumus, töökindlus, testimise korraldus

Mõõdetavad suurused

Mõõdetakse päris paljusid tarkvaraga seotud suurusi, kuid eesmärgist lähtuvalt võiks eristada järgmisi alamhulki:

* vastavus spetsifikatsioonile
* sobivus kasutajatele
* veakindlus
* kulud loomiseks ja haldamiseks.

Ehkki "hea" programmi loomisel tuleb kõiki neid aspekte arvestada, võib mõnel juhul ühe või teise koha pealt silma veidi kinni pigistada. Nagunii pole enamasti üheski valdkonnas võimalik täielikku täpsust saavutada, tuleb piirduda hägusa loogika ning leiduvate vahenditega. Spetsifikatsioonile pea täpne vastavus on näiteks vajalik, kuid loodav moodul peab ilma muudatusteta sobituma teiste omasuguste hulka. Siis tuleb kasvõi töökiiruses ja kasutusmugavuses järele anda, et loodud tarkvaratükk standardile vastaks. Niigi on näiteid standardi järgi loodud draiveritest mis mõnes kombinatsioonis töötavad ja teises mitte ning vaesel kasutajal pole sageli muud teha kui meelehärmist kukalt kratsida. Parem kui me midagi sama hullu enam juurde ei loo. Selle kontrollimiseks tuleb mõõta loodud tarkvara vastavust standarditele.

Iseseisvalt töötava programmi puhul on pigem tähtis, et kasutaja selle abil oma töö võimalikult hästi ja kiiresti valmis saaks. Nii võib mõnikord sellisel juhul pigem eelnevast detailsest spetsifikatsioonist loobuda ning töö käigus tekkinud häid ideid juurde lisada. Loomulikult tuleb seejärel testida nii tehtud muudatusi kui kogu programmi kasutaja(te) peal ning hoolitseda, et tulemus võimalikult hästi tarvitatud oleks.

Mida raskemad on võimaliku vea tagajärjed, seda hoolikamalt tuleb neist hoiduda. Kui suurest mälust ühekordselt paar baiti rohkem kulub või kasutaja mõne operatsiooni juures sekundi murdosa jagu rohkem ootama peab, siis pole veel asi väga hull. Valesid tulemusi andev funktsioon võib juba rohkem muret valmistada ning kui selline koodilõik juhtub sattuma lennuki juhtimissüsteemi, siis võib seda mõne aja pärast juba ajalehe õnnetusteveerust lugeda. Järelikult sellistel juhtudel on vajalik veakindlust põhjalikult testida.

Tarkvara loomise kulu on võimalik samuti ennustada ning mõõta. Sama ülesande lahendamisel võib konkureerivate võimaluste puhul projekti maksumusega võrreldes väike osa otsustada, millises suunas on mõistlik liikuda.

Valikuline loetelu tarkvara mõõdetavatest suurustest

Maksumus

* Kulud projekteerimisele, kirjutamisele, testimisele ja muule (õpe, sõidud, kohtumised).
* Keskmine koodirea maksumus
* Valmimisjärgse koodihoolduse kulud
* Dokumenteerimisele kulutatud aeg
* Arvutipargi hind
* Ümbertegemiseks kulunud vahendid

Vead

* Tarkvara loomisel ning testimisel ilmnenud vigade ning veatüüpide hulk
* Millal ja kuidas vead leiti
* Spetsifikatsioonist leitud vead
* Moodulite õnnestunud ning vigaste ühendamiste suhe.

Võrdlus eelmise tarkvaraprojektiga

* Lähtekoodi kasvukõver
* Koodimuutuste jaotus arendamise ning haldamise käigus
* Maht koodiridades/keerukuspunktides
* Ajakava sõltuvus mahust
* Hinna sõltuvus mahust
* Töötajate hulk
* Dokumentatsiooni maht

Maksumus jagati vanemates käsitlustes enamjaolt kolme võrdsesse ossa projekteerimise, kirjutamise ning testi vahel muutes mõnikord suhteid sõltuvalt tarkvara eripärast. Nüüd aga on leitud, et teistega vähemalt samaväärne on juurdekuuluv "muu", kuhu kuuluvad nii konteksti uurimine, uute oskuste õppimine kui enese tutvustamine. Ilma selleta võib tehiskeskkonnas mõnda aega hakkama saama, kuid iseseisvana tänapäeva muutuvas maailmas ei tule hästi välja.

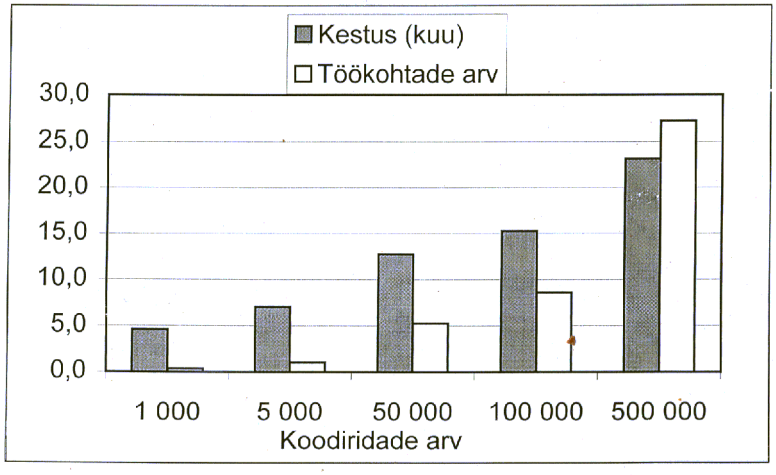
Projekte uurides ja võrreldes on leitud mõnesugused seosed. Küll nõrgad ning eri valdkondade ja meeskondade puhul tublisti erinevad, kuid siiski märgatavad seosed. Näiteks, et sama tüüpi rakenduste puhul on valminud koodiridade arv ning tööjõukulu ligikaudu võrdelised. Ning, et mida mahukam projekt, seda enam on sinna põhjust ja võimalust inimesi tööle võtta. Väiksema ülesande puhul pole suurel inimeste hulgal mõtet, sest toiminguid ei anna nõnda kergelt üksteisest sõltumatuks jagada ning algse ülesannete jagamise ning pärastise tulemust kokku ühendamise peale võib rohkem aega ja jõudu kuluda kui kogu lugu üksinda tehes oleks võtnud.

Kuna mahukamatel projektidel on võimalik rohkem inimesi tööle võtta, siis nende kestus kasvab tunduvalt aeglasemalt kui maht – siin tabelis on pakutu suisa neljandat juurt. Samas dokumentatsioon tuleb suuremate projektide juures ka märgatav – selle hulk on ligikaudu võrdeline töötavate inimeste arvuga.

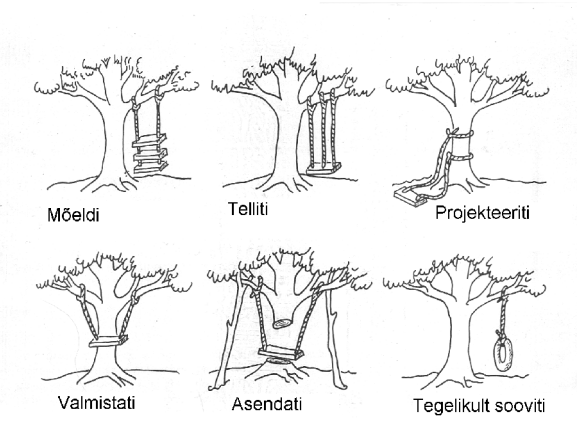
Tarkvara maksumuse ligikaudne sõltuvus mahust

Koodiridade arv	1 000	5 000	50 000	100 000	500 000	Valem
Tööjõukulu (inimkuu)	1,5	7,2	68,4	135,0	653,5	$1{,}48*K^{0{,}98}$
Kestus (kuu)	4,6	7,0	12,7	15,2	23,1	$4{,}6*K^{0{,}26}$
Töökohtade arv	0,3	1,0	5,2	8,6	27,2	$0{,}24*\text{Tööjõukulu}^{0{,}73}$
Dokumentatsioon (lk)	34,7	112,4	603,4	1000,8	3240,2	$34{,}7*K^{0{,}73}$

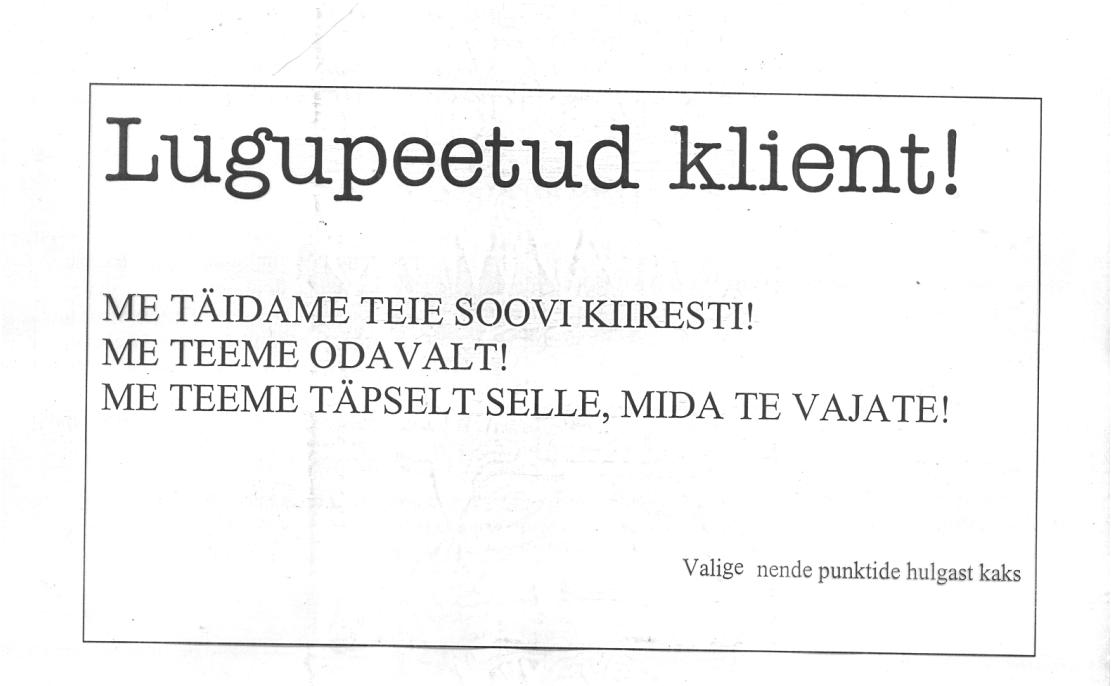
K=Tuhat koodirida



Kui suuremat tarkvaraprojekti põhjalikumalt ette ei kavanda, siis juhtub kergesti nõnda, nagu näha järgnevas Soome huumorikogumikust pärit koomiksis. Et kui müügimees, arendaja ning klient oma vajadustest selgesti aru ei saa, siis võib kuluda palju aega ja vahendeid saavutamaks mingilgi moel rahuldavat tulemust – selle asemel, et lihtsate vahenditega soovitud funktsionaalsus saavutada. Tõsi küll – esimesel juhul on põhjust ja võimalust iga sammu jaoks vahendeid küsida ning täiesti uues valdkonnas tegutsedes on harva lootust paljalt oma ideedele tuginedes lihtsale ja toimivale lahendusele jõuda. Kui siinset pilti aga silma ees hoida, siis tasub teada, et peaaegu alati on mõni lihtsam lahendus olemas. Nii nagu malemänguski.



Igal asjal pidada olema oma head ja halvad küljed. Ning kui kusagilt võita, siis tuleb teisest kohast järele anda. Programmeerimisfirmadele olen kuulnud soovitus panna ukse peale järgnev plakat.



Nagu näha, saab kliendi meelitada kohale, sest kõik pakutud punktid on üldjuhul täidetavad. Ainult, et korraga nende täitmiseks peab lihtsalt õnnelik juhus olema. Kahe punkti täitmine on aga palju jõukohasem. Kui soovitakse kiiret ja odavat lahendust ning pole nõnda tähtis, et kõik tehnilised nõuded täpselt täidetud oleks, siis võib mõningase otsimise tulemusena leida sobivad valmiskomponendid olgu siis oma firma seest või võrgu pealt ning nende abil sobiva valdkonna rakendus kokku klopsida. Kui hind pole tellijale probleem, siis võib mitmete tegijate, pikkade tööpäevade ja ostetud komponentidega täiesti kõlbuliku tulemuse saavutada. Ning kui pole kiiret, sel juhul võib mõnigi tarkvaraprojekt valmida küllatki "muu seas", pannes pikema aja jooksul tallele sobivaid koodilõike ja komponente ning siis mõne päevaga lõpptulemuse viimistleda.

Vead

Traditsiooniline ning kõige selgem ja arvutilähedasem tarkvara kvaliteedi mõõtmise valdkond. Kui pidevalt saadakse sisestatud õigete andmete abil valesid tulemusi, siis ei saa seda tarkvara mitte heaks pidada. NASA-s tehtud 5-aastase uuringu käigus ilmnenud ligi 10000 viga jagati tähtsuse järjekorras järgmistesse gruppidesse:

* Valeandmed
* Vigased liidesed programmide vahel
* Programmi tööjärje suunamisvead
* Väljade algväärtustus, mälu eraldamine ja vabastamine
* Arvutusvead

Liidesed tulevad mängu suuremate programmide puhul, kus väikesest ebaklapist võib järgneda hulk probleeme, sest sama ühendust kasutatakse paljudes kohtades. Algväärtustamise ja mäluga seotud vead sõltuvad mõningal määral kasutatavast programmeerimiskeelest ning -tehnikast, ülejäänuid aga tuleb kindlasti iga programmi koostamisel arvestada, mõõta ja parandada.

Testid

Eriti algajal kirjutajal, kuid pea alati ka kogenumal koodiloojal ei hakka programm pärast lähtekoodi valmistippimist kohe õigesti tööle. Ikka õnnestub sisse teha mõni süntaksiviga, et selle tulemusena translaator loodud tekstist aru ei saa. Või kui puuduvad semikoolonid või sulud on paika saanud, siis ikka juhtub, et kõik vastused ei tundu inimliku loogikaga järele kontrollides kuidagi mõistlikud olema. Ühele lehele mahtuva käsujada puhul jõuab silmadega koodi üle vaadata, mitmesuguste andmete ja väljatrükkidega katsetada kus viga leidub ning see rida muuta või välja kommenteerida ja paremaga asendada. Nii saab enamasti mõne(kümne) minutiga programmilõigu tööle ning võib seda rahumeeli enese huvides kasutama hakata. Kasutasime intuitiivselt testimise võtteid ning tõenäoliselt edukalt. Kui aga programm on suurem ja mitme inimese hallata või on igal leiduval veal kõrge hind, siis sellisest lähenemisest ei piisa ning tuleb süstemaatiliselt kontrollida, kas masin etteantud käskude peale teeb seda mida vaja ning mitte lihtsalt seda, mis me talle ette kirjutasime. Siinsele lühikokkuvõttele võib pikemaid seletusi kõrvale leida raamatust Tarkvara kvaliteet ja standardid, Tepandi 1997.

Laused ja harud

Süstemaatiliseks testimiseks on hulk võimalusi mitmesuguse veakindluse ning ressursikuluga. Üks võimalus on võtta ette programmi tekst ning proovida programmil lasta läbida kõik kirjutatud laused, soovitavalt vähemalt kaks korda. Juhuslike andmetega katsetades võib kergesti juhtuda, et läbitakse korduvalt samu teid ning eriolukordadel töötlemiseks loodud programmilõigud (mis korralikult tingimusi kontrollival programmil võivad võtta küllalt suure osa mahust) võivad paljus jääda läbimata. Selline test ei avasta küll samas lauses lähteandmetest tingitud eriolukordi, kuid võimaldab leida mõned suuremad kahe silma vahele jäänud apsakad ning näitab kätte ka lõigud, kuhu polegi võimalik jõuda.

Eelkirjeldatud lauseadekvaatsuseks nimetatud meetodi täiendusena on loodud haru- ning elementaartingimuste adekvaatsuse meetodid. Niimoodi tuleb lisaks kõikidele käsulausetele läbida ka tingimuste tühjad pooled ning liittingimuste puhul proovida iga osa töötamist. Näiteks

```
if (vanus<10) or (vanus>20) then

writeln('pole teismeline');
```

juures piisaks esimesel juhul kontrollida väärtusega nt. 11, teisel 7 ja 11 ning kolmandal 7, 11 ja 25.

Lähtekoodita test, piirjuhud

Ka lähtekoodita saab süstemaatiliselt testida, siis tuleb leida andmevahemikud, mille juures programm peab erinevalt käituma. Alustatakse aga nii, kuidas ka tavainimene võiks programmi tööd proovima hakata, umbes sellises järjekorras:

* Kas programm üldse midagi teeb, tööle läheb
* Kas tähtsamad kohad töötavad.
* Kuidas õnnestub masinalt viga välja meelitada
* Kas tehakse kõike mis tarvis
* Kuidas kõik kättesaadavad vahendid toimivad
* Andmed piirjuhtudel
* Suur koormus (palju andmeid, kasutajaid)

Kui ühtemoodi töödedavad andmed asuvad mingis vahemikus, siis tasub proovida sisestada väärtusi nii vahemiku seest, eest kui tagant. Piiri juures tuleks teha mitu katset. Testi sisendandmete komplekt tuleks koostada nii, et kahtlane väärtus oleks vaid üks, ülejäänud asugu võimalikult probleemidevabas piirkonnas. Nii on kergem vigu üles leida. Testida tuleks nii sisendite kui väljundite piirolukorras. Mõned näited piiride leidmisekst.

Reaalarvu puhul proovida täpselt piirjuhuga, samuti üles- ning allapoole võimalikult vähe, kuid siiski eristatava arvuga. Kui väärtusel pole ülemist piiri tasub proovida väga suure arvuga. Hulkade puhul tuleks võtta arve nii seest kui väljast. Samuti võib piirolukordadena arvestada nii täis- kui reaalarvu lubatud maksimaalset või minimaalset suurust ja massiivide varutud pikkusi.

Juhuslikud andmed

Lihtsalt juhuslike andmete väljamõtlemisega kaasneb oht, et kasutaja pakub peast samu alateadlikult tuttavaks saanud väärtusi ning mitmed vahemikud jäävad proovimata. Lihtsalt matemaatilisi juhuarve pakkudes läheb skaala laiaks ning vajalikku piirkonda võib suhteliselt vähe katseid langeda. Kui aga eelnevalt kindlaks teha ligikaudsed enam tarvilikud piirkonnad ning siis seal juhuarvudega testida võib tulemus päris tõhus olla. Lisaks on siiski vajalikud piirolukordade testid.

Koodi analüüs, tüüpvead

Mida selgemini kirjutatud kood, seda lihtsam on vigu leida. Kehtib pea alati, kui teine inimene peab lähteteksti uurima, kuid ka ise on võimalik mõne ajaga loodud käske ja ideoloogiat nii palju unustada, et omakirjutatud koodist läbinärimine võib pea sama raske töö olla kui ise sama koodilõigu uuesti kirjutamine. Eri keelte ja valdkondade jaoks on koostatud loetelud sagedamini esinevate vigade kohta, mõned näited:

* Segased muutujanimed
* Kas pöördumisel muutujal väärtus olemas
* Andmete lugemise järjekord.
* Kas samu andmeid talletatakse/loetakse ühtmoodi
* Lubamatud tüübid, tüübimuundus
* Üle/alatäitumine
* Jagamine nulliga
* Muutujatel kahtlased väärtused (vanus=200)
* Ebatäpsuste kuhjumine (tsüklis liidad 1, lahutad 0.99)
* Sulud
* Kas spetsifikatsioon õieti tõlgitud
* Tsükli kordamine 0 või väga palju kordi
* Alamprogrammi sisendparameetrite muutmine
* Globaalsed ja lokaalsed muutujad

Moodulite ühendamine

Lihtsamate programmide korral võib valminud eraldi testitud moodulid ühte panna ning seejärel katsetada, kas ja kus vigu leidub. Suure hulga moodulite puhul on aga vea allikat raske üles leida. Vaheetapina saab kasutada üheskoos töötavate/testitavate moodulite kogumeid. Nii saab sellele mooduliplokile andmeid saata ning uurida, kas nad tagastavad selliseid väärtusi, nagu neid kirjeldav liides ette näeb.

Süsteemist terviklikuma pildi saamiseks võib osa mooduleid asendada väiksemate programmilõikudega või suisa inimpoolse vastamisega.

Vigade hulga hindamine

Üks võimalus vigade hulga mõõtmiseks on jälgida vigade avastamist sama intensiivse töö korral. Esialgu on vigade osakaal koodis suurem ning neid leitakse kergemini. Tasapisi avastamine väheneb ning seda kõverat pikendades võib ligikaudu hinnata allesjäänud vigade arvu.

Alternatiivina saab kasutada lisatud vigu. Kui tegijaid on mitu, siis saab üks lisada teise poolt uuritavasse moodulisse tahtlikke vigu. Uurides leitud tegelike ning lisatud vigade suhet saab aimata allesjäänud vigade arvu.

Testimise maksumus

Kõiki vigu on lootust harva kätte saada. Kui valida konkreetse programmi jaoks sobivat testi, siis vigade ohtlikkuse ning parandamiseks kuluva hinna suhted oleksid järjekorras:

* proov kasutajalt saadavate andmetega (tõenäolisemad enamkasutatavad koodilõigud)
* läbivaatused (avastavad varaseid kalleid vigu)
* vea oletus (ekspert)
* piirjuhud
* sisendandmete grupid
* topeltprogramm
* lisatud vead
* tõestamine

Vigade põhjalikum püüdmine

Väga veakindlate programmide testimiseks tavalistest meetoditest ei piisa. Ka ressursside põhjalikumal kulutamisel ei õnnestu naljalt viia vigade arvu alla viie ühe tuhande koodirea kohta. Edasi iga järgneva vea avastamiseks tehtavad kulutused suurenevad tohutult. Kalleid seadmeid juhtiv programm peaks aga töötama suhteliselt laitmatult, sest koodiapsaka tõttu tekkinud katastroof või suur varaline kahju joonistab tarkvara tootjale paratamatult suure pleki. Tehiskaaslase juhtimiseks kasutatakse aga ligi pooltteist miljonit koodirida nii et selle töökindlana loomine ning hoidmine ei pruugi kuigi kerge ülesanne olla. Liiatigi kui tegelikes oludes katsetamist kuigi palju lubada ei saa. Mõned vahendid kitsaskohast ülesaamiseks.

Sama ülesannet täitev programm luuakse mitmel moel ning saadud vastuseid võrreldakse. Nii leiab võimalikke vigu testimisel ning käivitades võib häda korral ka eeldada, et kui mitu algoritmi näitavad sama tulemust, üks aga erinevat, siis esimestel on tõenäoliselt õigus. Selliseid lahendusi kasutati paarkümmend aastat tagasi suurarvutite puhul, kus lambi läbipõlemine võis kergesti tulemusi muuta. Tarkvara puhul aga kipub inimene samu vigu tegema ning sellega meetodi töökindlus väheneb.

Kaitsemehhanismid ning veapuu analüüs. Suurte vigade (näiteks kokkupõrke) vältimiseks saab mõnikord piirata nende vigade eeltingimusi. Tulemusena võib muutuda toimimine aeglasemaks kuid turvalisemaks. Eks kõigil ole oma hind.

Matemaatiline tõestamine, et programm vastab spetsifikatsioonile. Töömahukas, kuid kriitilistel puhkudel vajalik.

Lühikesed alamprogrammid, lihtsad moodulid. Mida väiksem on lõik, seda enam on sealt lootust kõik vead üles leida.

Muid töökindluse parandamise vahendeid

Autori analüüs

... ehk lihtsalt programmi koostaja või koostajate grupp vaatab oma töö pärast enese arvates valmimist veel korra terase pilguga üle. Hea tahte puhul küllalt tulemusrikas meetod, lihtne korraldada ning nõuab vaid koostajalt lisatööaega, kuid muude kulutustega seotud ei ole. Miinuseks on, et autori mõte käib vana rada, ta ei suuda jälgida kasutaja loogikat ning seetõttu jäävad võimalikud vead avastamata. Käsud, mis autorile une pealt selged on, ei pruugi võõrast programmi kasutavale inimesele sugugi mitte tuttavad olla ning seetõttu võib viimane päris kergesti imelikesse olukordadesse sattuda. Samuti on autori eesmärk lõpetada töö, mitte leida vigu ning ta ei soovi oma valmistehtud loogikat lõhkuda olgugi, et sellest võivad mitmed arusaamatused alguse saada.

Läbivaatus

Tarkvara tootev seltskond istub kokku ning igaüks saab ülevaate ka teiste tegemisest. Teades, et ka teised sinu tööd jälgivad on ka omal alateadlik soov mõnigi asi ilusamini ja selgemalt teha, et see paremini arusaadav oleks ning teistele vigasena ei paistaks. Mitmekesi koos uurides on mitu mõttekäiku ning harvemini jääb mõni tähtis detail tähelepanuta. Nii suurendatakse inimestevahelisi kontakte ning vajaduse korral on ühte liiget võimalik kergemini asendada. Kasulikuks korraldamiseks peavad sellised läbivaatused olema ettevalmistatud, muul juhul võib koosistumine lihtsalt lõbusaks mokalaadaks kujuneda. Samuti peab leiduma juhataja, kes aitab ajakavast kinni pidada ning hoolitseb, et osavõtjad teaksid ettevõtmise eesmärke ning oma kohustusi. Isiklike vastuolude või temperamentsete isikute puhul võib tekkida probleeme.

Audit

Väline hinnang valmivale tarkvarale või valmistamise protsessile. Üheks eesmärgiks on välistele osapooltele (näiteks tellijale) näidata tootja usaldatavust. Samuti aitab asutuse juhtkonnal otsida ilmnenud probleemide põhjusi või avastada võimalikke ohuallikaid. Võõra inimese käest on mõnel kriitikat kuulda või lugeda kergem kui omade seast. Samas kui pole sisemist valmisolekut lasta oma vigu välja paista või nendest õppida, siis pole loota ka auditi abil oma töö tulemuste paranemist.

Testimise korraldus

Kui kogu programmi loob ja kasutab sama inimene ning ilmnenud vead ei saa tuua suurt kahju, siis piisab kui testitakse ja parandatakse vigu töö käigus.

Ühe looja ja ühe tellija puhul kirjutab tellija programmist leitud vead üles, arendaja parandab need ning ei tohiks ka muid keerulisi protseduure sinna juurde tekkida. Kui tarkvara täidab vastutusrikast ülesannet ja/või on sellel palju kasutajaid, siis peab teste olema mitu: nii tootja, testijate kui väikse kasutajate grupi juures. Hoolimata sellest kipub märgatav osa vigu siiski lõppkasutajate juurde jõudma, kes ei oska nende parandamiseks suurt midagi ette võtta. Mõnel puhul aga pole võimalik eeltoodud süsteemi kasutada, siis tuleb leida abivahendid.

Keeruka toote puhul tuleb alustada kontrolliga juba algusest ning hallatavate osadena vaadata üle nii spetsifikatsioon kui kood. Siis on lootust pärastise osade ühendamise juures ka tekkivate vigade põhjustele kergemini jälile saada. Sama käib ka lihtsama kuid ülisuure töökindlusvajadusega programmi kohta. Seal peab iga rida olema mitmeid kordi läbi uuritud ja põhjendatud.

Kui projekteerijad ja programmeerijad asuvad lahus, siis peab arvestama vajadust programmi kirjutamisel spetsifikatsiooni täiendada, kui selgub, et midagi olulist on tähelepanuta jäänud. Hajusalt asuva kasutaja puhul on sealtpoolne vigade teada saamine ning pärastine muudatuste tegemine keeruline. Ikka leidub keegi, kelle masinasse on vana versioon sisse jäänud ning kus mõned tehtud parandused ei tööta.

Muutmiste juures tuleb kontrollida, kas endised testid töötavad. Kergesti võib juhtuda, et ühte viga parandades luuakse teine või rohkemgi juurde.

Mõned tarkvarameetrika reeglid
* Tarkvarakontroll pole imevahend

Kontroll ja mõõtmine ilma lähema eesmärgita toob lihtsalt muret ja vaeva. Kasu võib sellest olla vaid ühe abinõuna tarkvaratöö korraldamisel

* Keskendu tulemuste kasutamisele, mitte andmete kogumisele.

Kui liialt palju energiat kulub esimese peale, siis on raske kasuliku tulemuseni jõuda.

* Selgita eesmärgid.

Tarkvaratootja püüab enamasti suurendada toodangu hulka või kvaliteeti, vähendada kulusid, püsida graafikus. Enne, kui asuda mõõtma ja kontrollima, tuleks selgeks teha, millises järjekorras ja valdkonnas oleks mõistlik muudatusi teha. Kus nagunii kõik esialgu samamoodi jääb, seal pole suurt mõtet ka mõõtmisele vahendeid kulutada.

* Uuri, kuidas mõõtmistulemusi rakendada.

Tippjuhid tõenäoliselt loodavad uuringust teavet põhiprintsiipide kohta. Projektijuhid soovivad andmeid jooksva projekti käigushoidmiseks ning uute plaanimiseks. Programmeerijatel on lootust saada teada, millele on mõistlik enam ja millele vähem rõhku panna. Hea meetrika puhul peaks pea samade andmete analüüsist kõik osapooled kasu saama.

* Uuri rakendamisvalmidust.

Mõõtmistulemuste kasuliku tarvitamisega kaasneb paratamatult vajadus inimeste töös muudatusi teha. Enne mõõtmise algust on mõistlik kindlaks teha, kes, kas ja kui palju on valmis oma tööharjumusi muutma ning kui kulukaks selle sisseviimine läheks ettevõttele. Head süsti võib mõõtmiselt oodata vaid siis, kui nii juhtkond kui tehniline personal on valmis uuringu tulemusi oma töös arvesse võtma.

* Kavanda kiire edu

Esimene mõõdetav projekt tuleks valida selline, kus on loota juba lühikese aja jooksul märgatavaid tulemusi. Siis tekib osalejatel usk, et mõõtmise ning tulemuste rakendamise abil on võimalik omale ja teistele kasu saada. Mitmete muidu heade uuringute hädaks kipub olema, et mõõtmine ning tulemuste rakendamine võtavad nii kaua aega, et töötajad on selleks ajaks suurest andmete kogumisest tüdinud ning ei arva mõõtmisest enam kuigi hästi. Sellises olukorras aga soovituste andmine, et tee nii või teisiti, kuigi see võiks pikemas plaanis kasu tuua tekitab pigem trotsi kui valmisolekut uuendustega kaasa minna. Kiiresti võib uurimisest meeldiva laengu saada näiteks vastava ala programmidest tüüpvigu leides. Kui mõne nädalaga või rutemgi on võimalik teha analüüs ning näidata levinumate vigade jaotus, siis saab seda kirjutamisel varsti arvestama hakata ning kirjutaja suhtub uurijasse juba tunduvalt väiksema ettevaatusega, võibolla suisa poolehoiuga. Soovides aga kogu protsessi suunamiseks tarvilikke vihjeid saada, selleks tuleb siiski pikema ajaga ja põhjalikumate arvutustega leppida.

* Keskendu seotud grupile

Kui kavatsetakse tarkvaratöö tulemust mõõtma hakata, siis tuleb paratamatult otsustada, milliseid projekte, millises arengujärgus ning mis tüüpi tööd seal tuleb mõõtma hakata. Edukat uuringut koos tulemuste rakendamisega on tunduvalt lihtsam läbi viia seotud grupis, näiteks ühes hoones asuvas ettevõtteüksuses. Suuremate maastaapidega saab üldisemaid andmed ning neid võib kasutada teadustöös või muudes teoreetilistes aruteludes, kuid nende abil pole kuigi palju lootust konkreetsete inimeste tööd tegelikult paremaks muuta. Uuringu tulemuseks saab olla olemasoleva töö analüüs ning oma vahendite abil selle paremaks muutmine, mitte võrdlus teiste gruppidega.

* Alusta väikselt

Mõõtmise sisseviimisel alusta väheste inimeste, tarkvaratootmise üksikute selgepiiriliste etapide ning üksikute projektidega. Väiksest õnnestunud mõõtmisest on kergem suurema peale edasi minna.

* Tootjad ning mõõtjad olgu eraldi

Neil on erinevad huvid ning ka hea tahtmise korral kipuvad ühed teisi segama.

* Kindlusta, et mõõtmised vastaksid eesmärkidele

Pole mõtet koguda andmeid, mille analüüsi tulemusi pole plaanis realiseerida. Iseenesest võivad mitmed väärtused tunduda huvitavad, aga kui neid pole kavas selle uuringu käigus kasutada, siis kokkuvõttes ikkagi raisatakse ressursse.

* Kogu võimalikult vähe andmeid.

Kui lisamõõtmised järelduste täpsust oluliselt ei suurenda, pole mõtet ilmaasjata andmeid kirja panna

* Hoidu liigsete tulemuste esitamisest.

Lugemisel parajasti kasutu kraam on müra. Kui uuringu käigus selgus huvitavaid tulemusi, tuleks neid näidata neile, kel selliste tulemustega midagi mõistlikku teha on. Üldraportis sellised tabelid ja graafikud lihtsalt raiskavad kasutajate aega ning riiulipinda. Tüüpilised andmed, mis liiaga raportitesse lisatakse, sest nad on olemas: testide arv, arvutite kasutusgraafikud, kohtumistele kulunud aeg, koodi keerukusaste.

* Arvesta uuringu hinda

Uuringul on mõtet, kui temasse kulutatud vahendid end tasa teenivad. Märkimisväärne osa kuludest tuleb kanda uuringu alustamisel, edaspidine ei pruugigi enam nii hull olla. Uuringu suuremad kulud: andmete kogumine - ankeetide täitmine, andmete saatmine (tarkvaratööliste lisatöö); tehniline personal - raamatupidamine, tulemuste transport õigesse kohta; analüüs, tulemuste esitamine. Uuring võib suures organisatsioonis võtta ~6% selle aja jaoks eraldatud ressurssidest, väiksemas aga kuni viiendiku. Andmete kogumine ei tohiks võtta üle paari protsendi projekti eelarvest, tehniline abi peaks piirduma nelja-viiega ning analüüsile võib veidi rohkem kuluda.

* Kavanda andmete analüüsile vähemalt kolm korda rohkem ressursse kui kogumisele

Maailmas on kogutud juba väga palju andmeid kuid suur osa neist vananeb andmekandjatel täiesti kasutuna või on neist kätte saadud tühine osa võimalikust teabest. Püüa seda viga vältida. Kogu ainult tarvilikke andmeid, võimalikult vähe ning püüa nende põhjal järeldused kätte saada.

* Kogu tööjõukulu andmeid vähemalt kord kuus

Kui vaja siis ka tihemini. Pikema aja peale oskab inimene harva tagantjärele oma pingutusi täpsemalt hinnata. Loodud kogumissüsteemi tihedamini tööle panna on lihtsam kui süsteemi käivitada.

* Uuri, millise tööjõukulu kohta on mõistlik andmeid koguda

Kas näiteks programmeerija suudab eristada, palju tal kulus aega koodi kirjutamiseks palju testimiseks. Mõnikord need kipuvad omavahel nii ühenduses olema, et eristada on küllalt raske.

* Kogu vaid töötavaks kuulutatud tarkvara vigu.

Konfigureerimata ja juhuslikult kokku pandud tükkide juurest leitud vigu ei õnnestu kergesti süstematiseerida. Samuti on selliste vigade ülesmärkimine paratamatult juhuslik ega anna pilti tegelikust vigade jaotusest. Muidu tuleks vigade juurde märkida: leidmise kuupäev; parandamise kuupäev; avastamiseks ning parandamiseks kulunud tööaeg; vea allikas (spetsifikatsioon, kood, eelmine muudatus); vea tüüp.

* Vea parandamiseks minevat aega on raske mõõta

Programmeerija saab tunduvalt kergemini öelda, palju kulus tal aega leidmiseks ja parandamiseks kokku.

* Tarkvaraprotsessi on raske mõõta

Samale tulemusele võib jõuda mitmeti.

* Jaga programmi loomine selgepiirilistesse etappidesse

Siis saab kergemini jälgida ajakavast kinnipidamist ning osalistel on selge hinnata oma edukust.

* Kasuta programmi mahu mõõtmisel ka koodiridu

Kuigi see mõõtühik on küllalt laialivalguv ning sõltuvalt keelest, kirjutajast ja keskkonnast võib samasuguse programmi puhul ligi suurusjärgu võrra erineda, on tegemist siiski lihtsalt edastatava suurusega, mis samades tingimustes aitab programme omavahel võrrelda. Lisaks tavalisele numbrile, kus loetakse kokku kõik tarkvara koosseisu kuuluvad read sõltumata sellest kas seal ka midagi kasulikku tehakse, eristatakse ka käskluste arvu (mõnel real võib neid olla mitu), kommentaaride arvu, korduvkasutamiseks loodud ridu. Kui programmis kasutatakse varem valminud lõike, siis saab programmi või tema osi jagada: uus; tugevasti muudetud (üle 25% ridadest); nõrgalt muudetud ning muutmata. Teisteks programmi mahu mõõtühikuteks võivad olla näiteks tsüklomaatiline keerukus (erinevate võimalike harude arv) või punktid kus arvestatakse nii projekteerimise, kirjutamise kui testimisega seotud tööd.

* Ära looda automaatsele andmekogumisele liialt

Sellise süsteemi ehitamine võib minna tunduvalt kallimaks kui inimjõul kogumine. Hea korralduse puhul võtab andmete edastamine vaid kuni paar protsenti programmeerijate tööajast.

* Tee andmete saatmine lihtsaks

Kui sellega probleeme ei tule, siis on andmete kogumine palju meeldivam. Hoolitse, et oleks kergesti võimalik kontakti võtta andmete kogujaga ning temalt vajadusel nõu saada.

* Kasuta andmete hoidmiseks olemasolevaid vastavaid programme

Ise selliste välja töötamine läheb liialt kulukaks.

* Andmed on nagunii vigased ja puudulikud

Sest ikka juhtub, et kusagil läks mõni number valesti, võeti ajapuuduses või pettekujutluses huupi või jäeti lihtsalt saatmata. Osalt sellepärast, et andmete kogumist ei pea programmeerijad oma põhitööks ega vaevu saadetavaid teateid üle kontrollima. Ning inimesed teevad ikka ja alati vigu.

Ülesandeid

- Koosta ruutvõrrandit lahendav programm.
- Hoolitse, et see töötaks korralikult ka kõikvõimaliku vigase sisestuse korral.
- Paiguta võimalikult iga toiming eraldi alamprogrammi (kokku neid vähemalt 5)
- Testi iga alamprogrammi tööd nõnda, et iga koodirida saaks vähemalt korra läbitud.
- Leia algandmed võimalike piirjuhtude tarbeks ning kontrolli programmi tööd nende puhul, samuti mõlemal pool piiri.

- Kommenteeri kõik oma funktsiooni, nende ülesanded, sisendparameetrid ja väljundväärtused.
- Lisa oma koodi paar apsakat, märgi need üles.
- Vahetage oma koodid pinginaabritega. Märkige üles kõik koodis leiduvad parandamist vajada võivad kohad koos ettepanekutega.
- Võrrelge koos kummagi koode ja ettepanekuid. Jälgige, kas lisatud vead leiti üles. Hinnake, kui suure osa moodustasid lisatud vead kogu paranduste hulgast.

Järelsõna

Kes end siia maani välja lugeda jõudis, sel jäi paratamatult eelnenust midagi külge. Kõik kopeeritud ekraanipildid ei pruugi aastate pärast sarnaseid programme käivitades enam päris samasugused välja näha, kuid enamik siin kirja pandud põhimõtteid peaksid loodetavasti püsima kauem kui aasta või paar. Või kui mitte, siis jõutakse nende asemele midagi uut ja paremat leiutada ning selle üle võib ainult rõõmu tunda.

Eestikeelseid programmeerimisraamatuid paistab ilmuma nagu seeni pärast vihma, mis on ainult hea. Eriti seetõttu, et igaüks neist on omamoodi. Vahel täiendavad sama valdkonna raamatud üksteist, kuid kümneid kattuva sisu, mõttelaadi ja raskustasemega raamatuid nagu näiteks inglisekeelseid Java algõpetusi pole ma eesti keeles mitte kohanud. Samas võib mõningase otsimise peale olgu siis võrgust või paberilt leida pea iga teema tarvise eestikeelse aluse, mille külge oma juurdeotsitavaid teadmisi kinnitama hakata. Seda nii informaatika, loodusteaduste kui ehk muudegi valdkondade puhul. Siinsetest kolmest kirjutisest võiks algusniidi kätte saada tunduvalt rohkem kui poolte tavaprogrammeerijale Java keele juures ette tulevate teemade puhul – et siis hiljem oleks kergem manuaalidest ja ringlevatelt näidetelt lisateavet hankida. Ning kes leiab sellise vajaliku ning enesele tuttavama valdkonna kus maakeelset seletust veel kirjas pole, siis pangu julgesti oma mõtted ja näited võrku üles, et ühisest keeleruumist kõigile kaasmaalastele tuge oleks. Kui algne seeme lihtsate vahenditega enesele selgeks tehtud, siis on edasi tunduvalt kergem mujalt tarkust juurde ammutada ja olemasolevale lisada.

Soovin lugejale ilusaid ideid ning pikka meelt oma rakenduste loomisel.