Andmebaasid

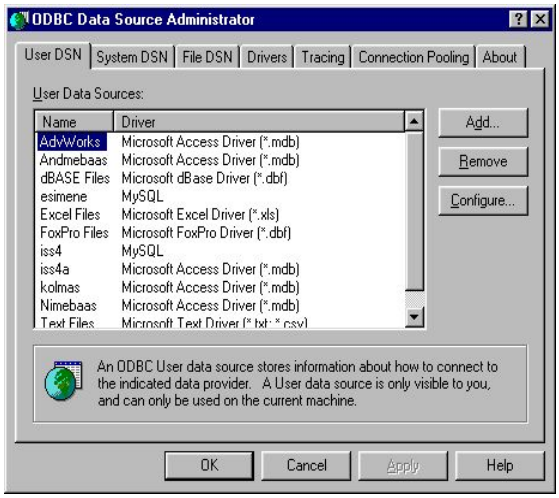
SQL, ODBC, Servlet, veebirakendus

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.

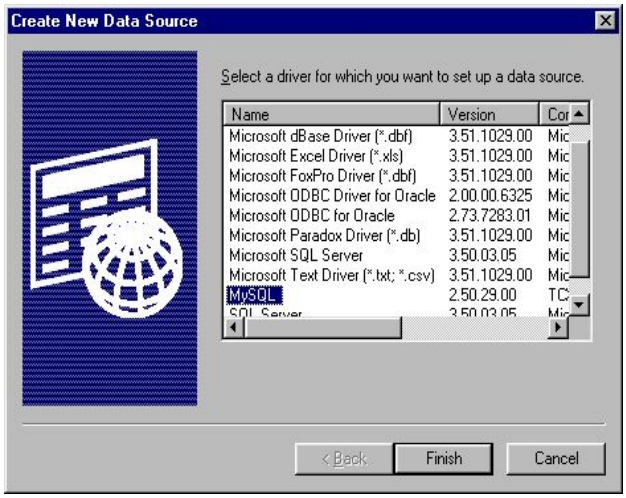
Andmebaasiühenduse loomine

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.

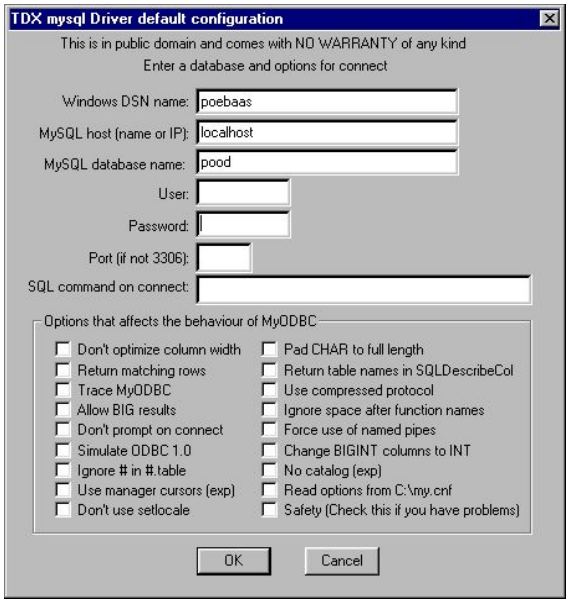


Kui aga soovida lihtsamalt läbi ajada, siis ei pea selleks mitte oma serverit püsti panema. Võib rahulikult toime tulla olemasolevate Accessi, Exceli või suisa tekstifaili draiveritega. Kui aga MySQL installeeritud, siis saab seda kasutada. Küllaltki universaalne koht andmebaasidele ligi pääsemiseks 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.

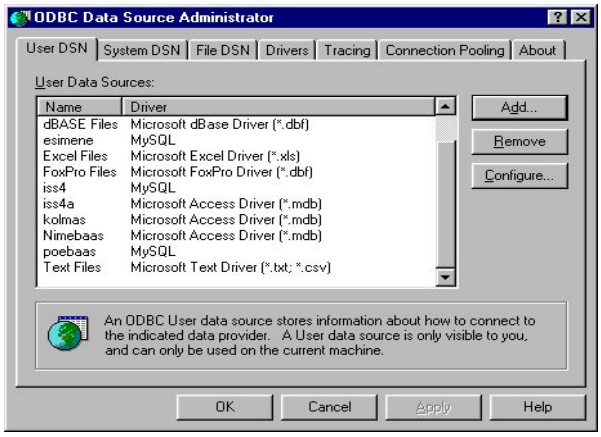


Nimetatud vahelüli (Open DataBase Connectivity) on lihtsalt ühine protokoll, mille kaudu saavad suhelda osapooled, kes üksteise keelt ei tunne (pole otseühenduseks vastavaid draivereid).

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



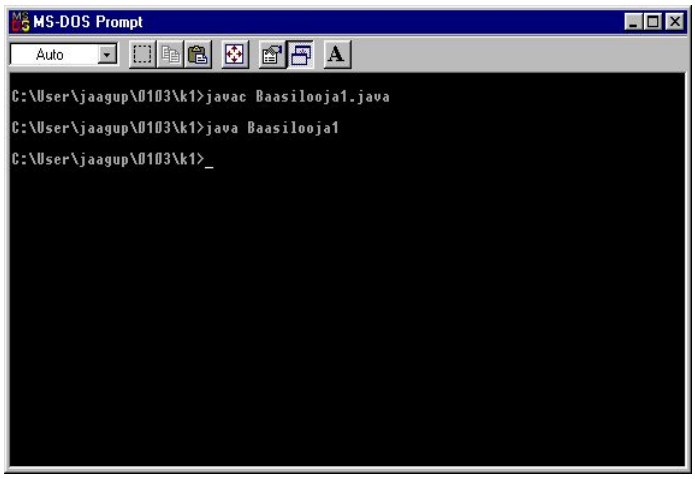
Kui kogu loomine läks õnnelikult, siis jõuame tagasi algse lehe juurde, kuhu on tekkinud ka vastloodud ühendus, siin näites nime all poebaas.

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.

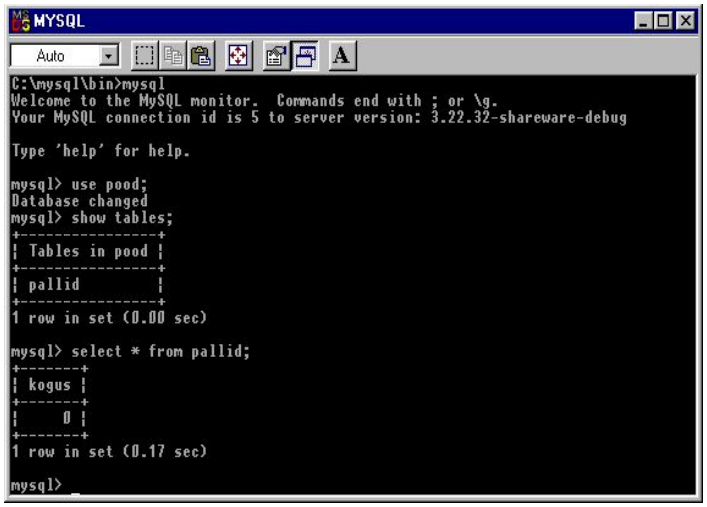
Andmebaasiga suhtlemiseks tuleb kõigepealt mällu laadida draiver. ODBC tarvis on Java starndardkomplektis kaasas `sun.jdbc.odbc.JdbcOdbcDriver`. Luuakse ühendus, jättes kasutajanime ja parooli koht tühjaks, kuna meie katsebaasi puhul neid ei nõuta. Saadetakse käsklaused teele ning suletakse ühendus.

```
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.

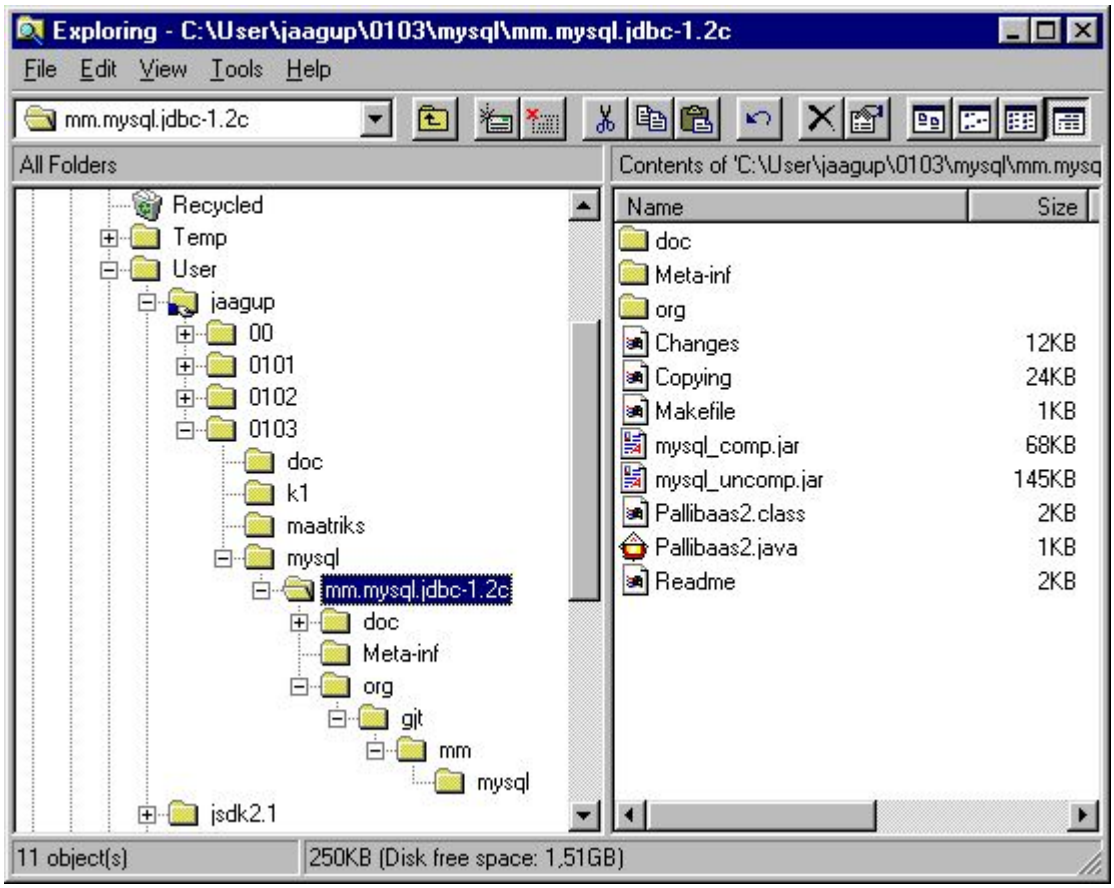
Otsene draiver

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

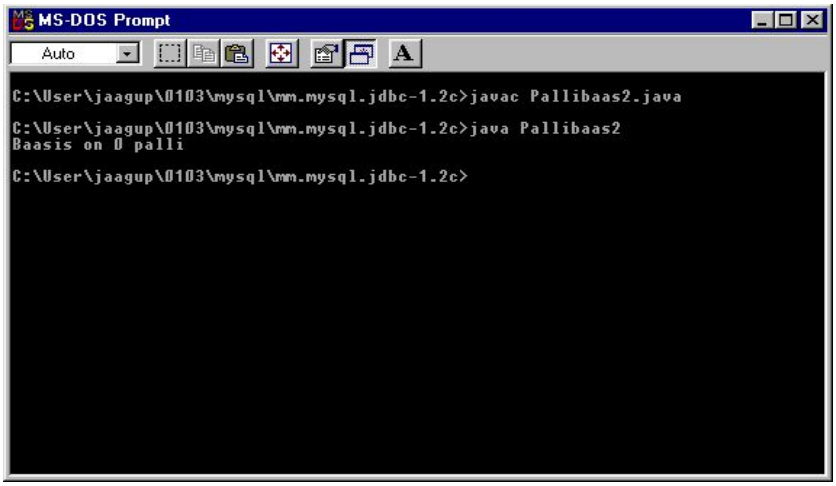
Kui sealne arhiiv maha laadida ning lahti pakkida, tekkis kataloog, milles nii draiver ise kui hulga õpetusi, kuidas temaga ümber käia.



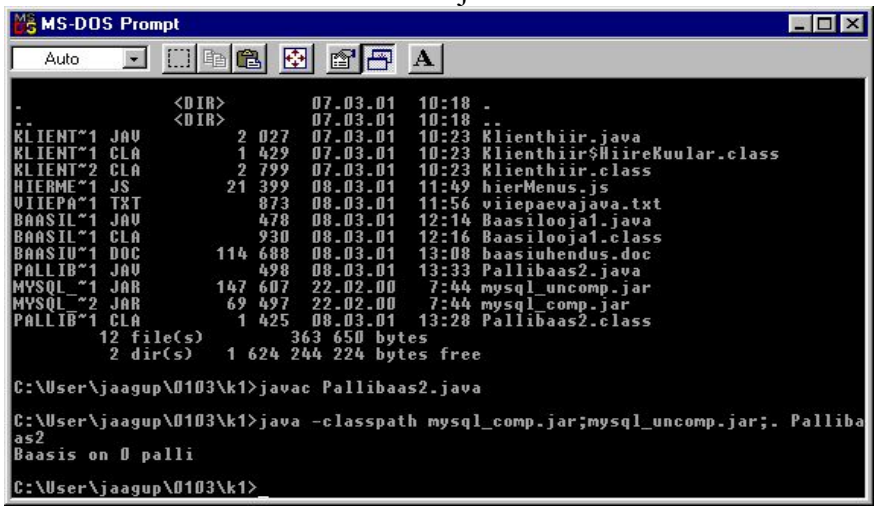
Kirjadest selgus, et muuta polegi vaja muud kui draiveri nime ning ühenduse URLi. Nüüd saab kirjutada otse jdbc:mysql: .

```
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.

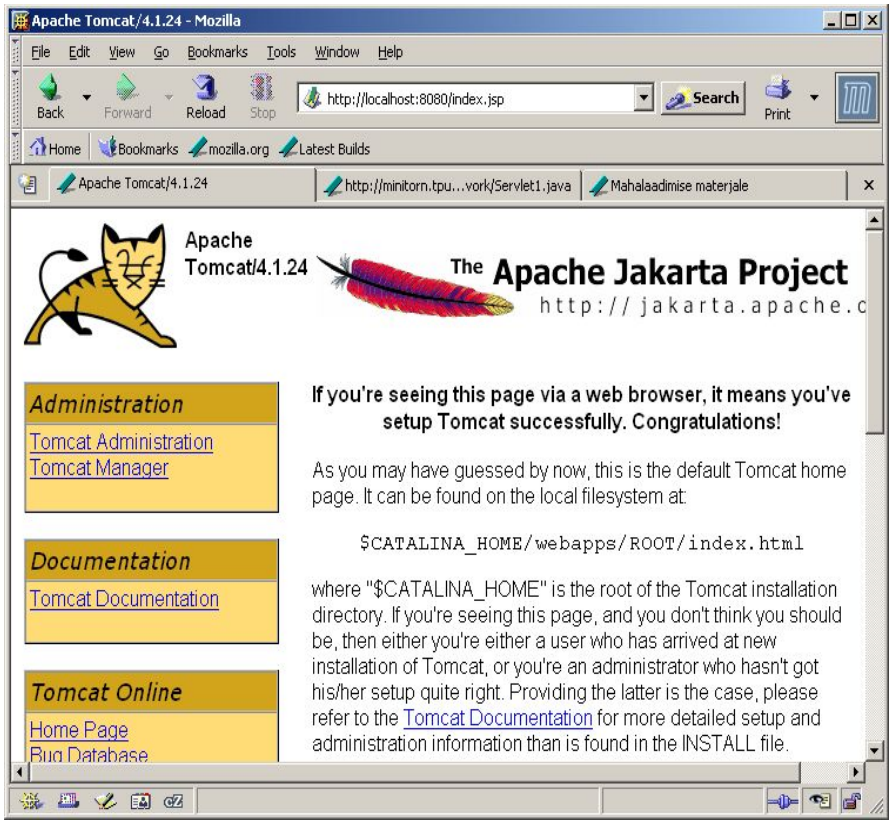
HTTP-päringute puhul on võimalike toiminguid vähemasti kuus, kuid servlettide puhul levinumateks GET ning POST, mõlemal juhul väljastatakse üldjuhul veebileht. GET-päringu puhul antakse parameetrid kaasa URLi real, nende pikkus on piiratum ning loodud leht võidakse kergemini puhverdada. Tüüpiline kasutusvaldkond 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.

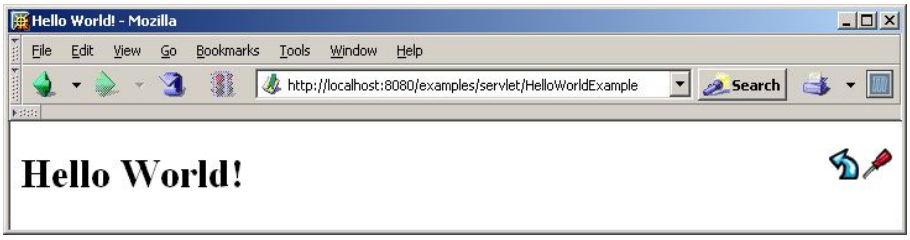
Järgnevalt näha võimalikult lihtne tervitav servlet.

```
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!");
  }
}
```

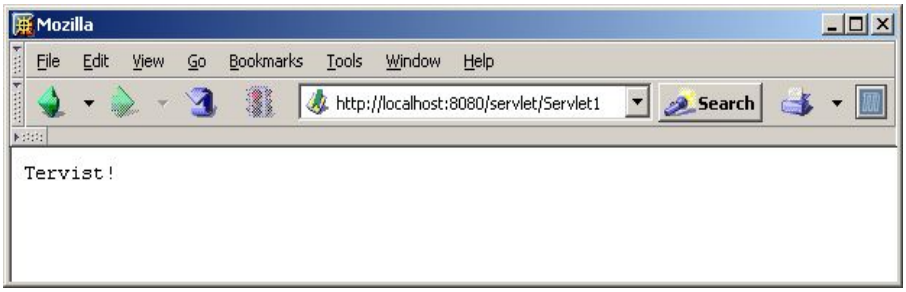
Enne tulemuse nägemist tuleb veel pingutada servletile sobiva keskkonna loomise nimel. Kel juba sobiv käivitusserver eelnevalt püsti, sel võib piisata faili kompileerimisest ning sobivasse kataloogi paigutamisest. Kel aga mitte, siis tuleb veidi installeerimisega pead vaevata. 2004. aastal tundub levinud 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.



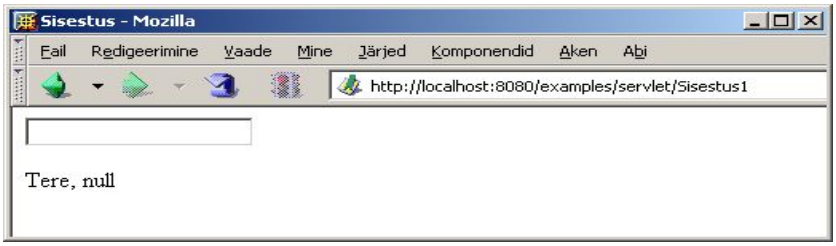
Sisestus

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.

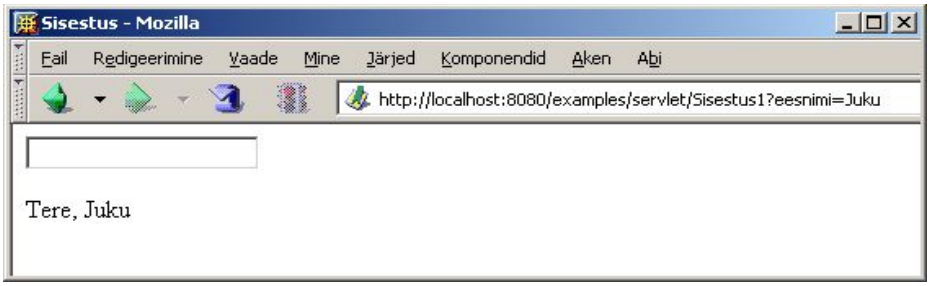
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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

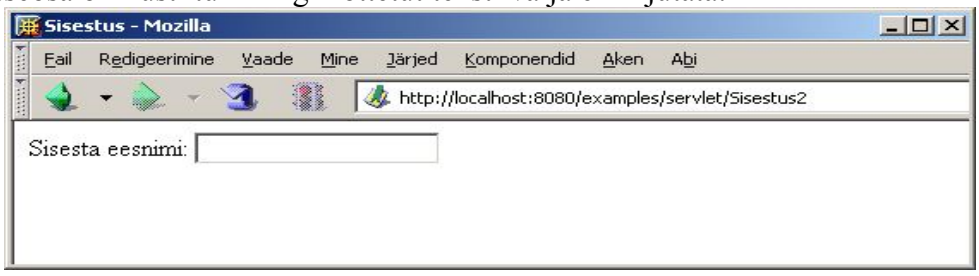
programmeerijale ilusti näha ning tal võimalus kontrollida, mis ja millise nime all serverisse saadeti.



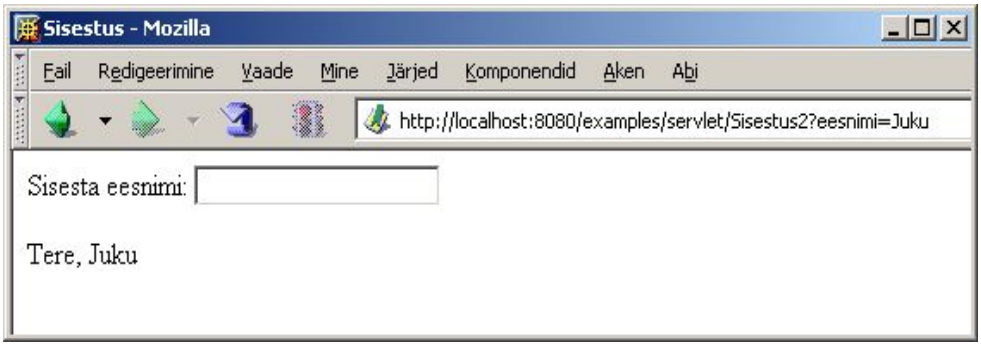
Esmakordselt näidatavast tühiväärtusest on täiesti võimalik hoiduda. Selleks tuleb enne nime välja trükkimist kontrollida, kas ikka midagi teele saadeti. Andmed loeti eesnime-nimelisse muutujasse kahel põhjusel. Lühema nimega muutujaga on lihtsalt kergem ümber käia kui pidevalt parameetrit käskluse kaudu 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>");
    }
}
```

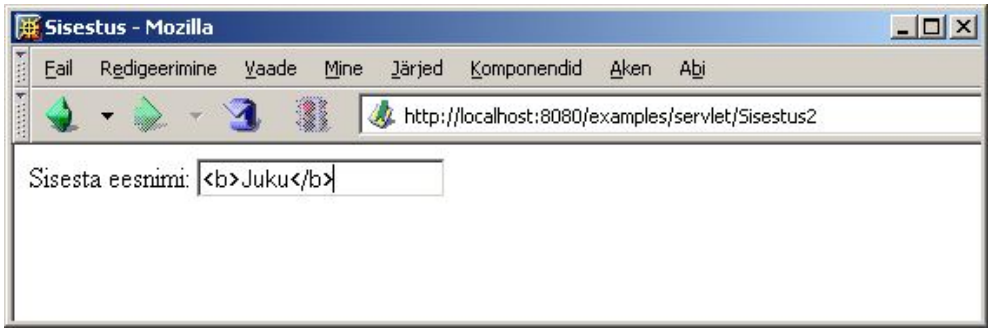
Kui nüüd kontrollitakse enne väljatrükki tühiväärtust, siis võib näha, et lehe vastuseosa on ilusti tühi ning mõttetut teksti välja ei kirjutata.



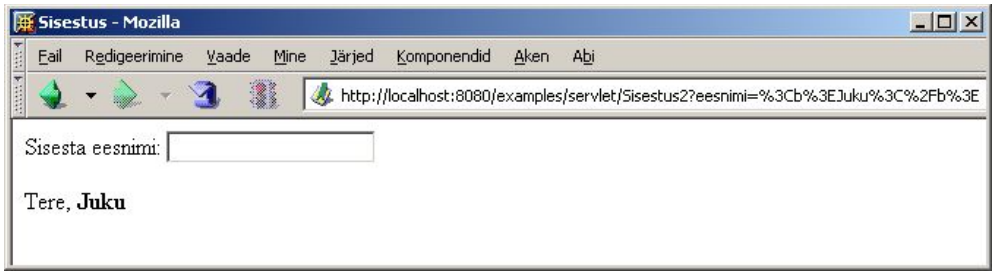
Viisakalt sisse kirjutatud nime puhul aga tervitatakse sama rõõmsasti vastu.



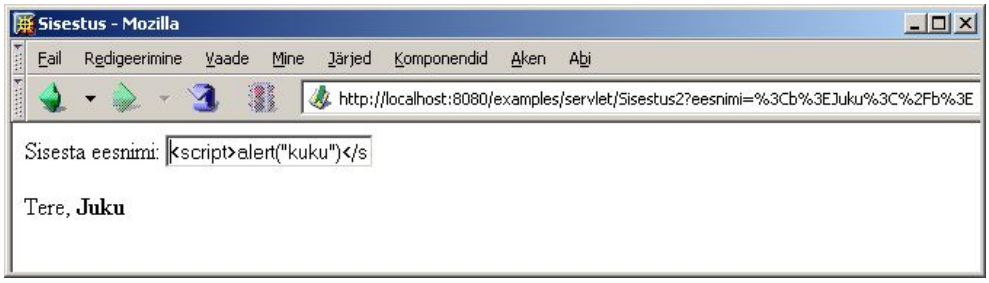
Räägitakse, et veebilehtede koostamisel tuleb arvestada igasuguste turvaprobleemidega. Ning et üheks märgatavaks ohuks on kasutajate nii kogemata kui meelega sisestatud erisümbolid. Lihtsamatel juhtudel võidakse kirjutada HTML-i kujunduskäsklusi oma teksti ilmestamiseks.



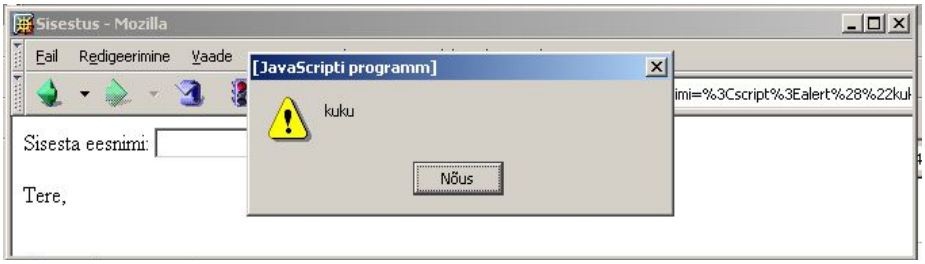
Tulemus võib sellisel juhul päris meediv olla.



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.

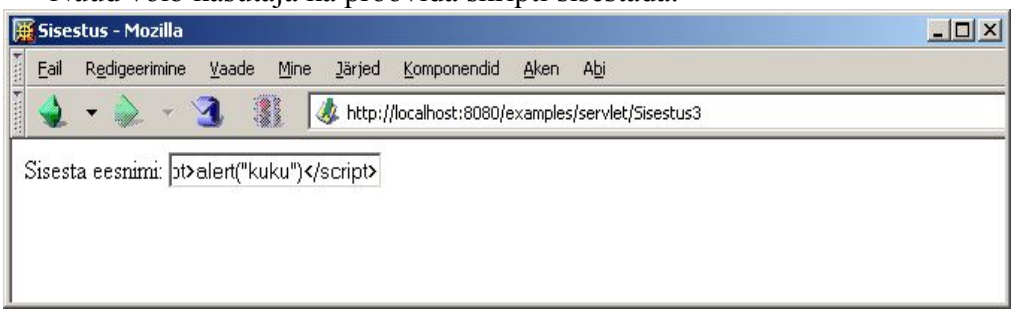
```
String eesnimi=Abi.filtreeriHTML(kysimus.getParameter("eesnimi"));
```

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

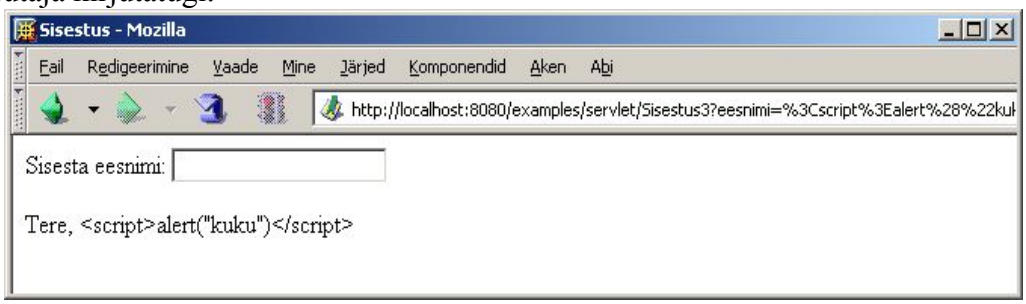
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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.

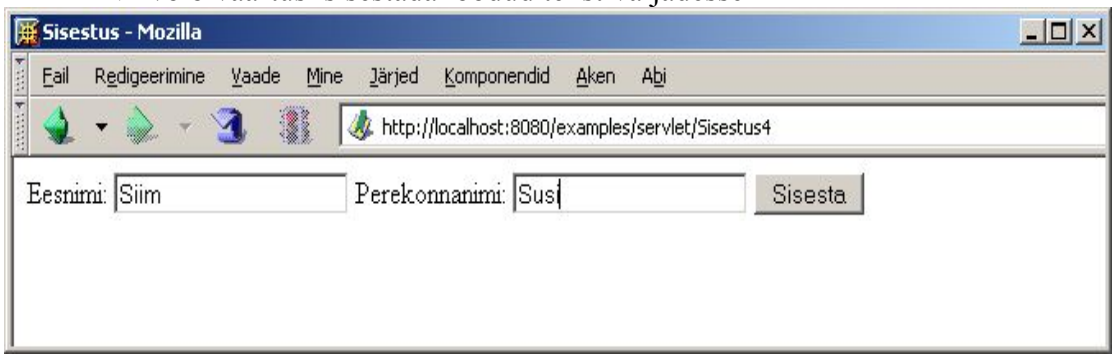
```
<html><head><title>Sisestus</title></head>
<body><form>Sisesta eesnimi: <input type='text' name='eesnimi' /></form>
Tere, &lt;script&gt;alert(&quot;kuku&quot;)&lt;/script&gt;
</body></html>
```

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”.

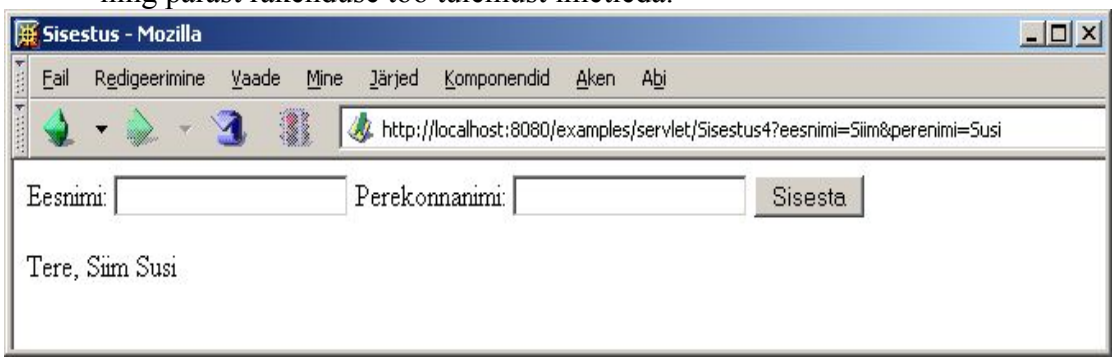
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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);
```

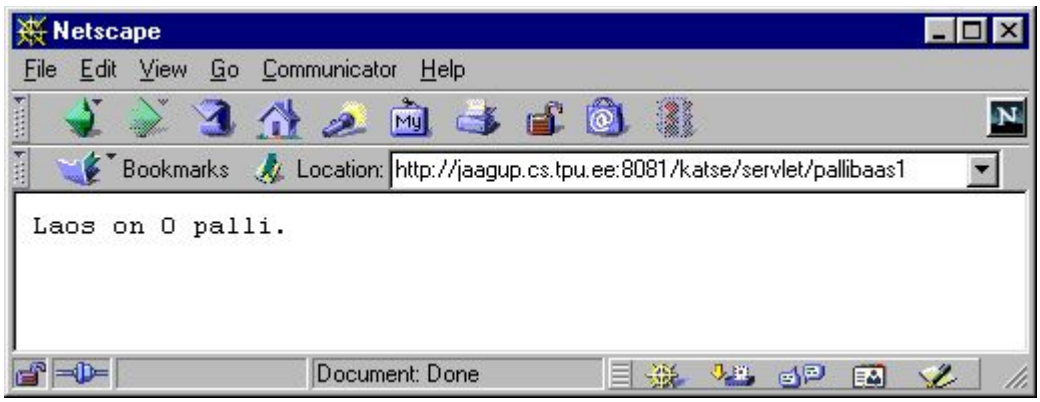
```
  }
}
```

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.

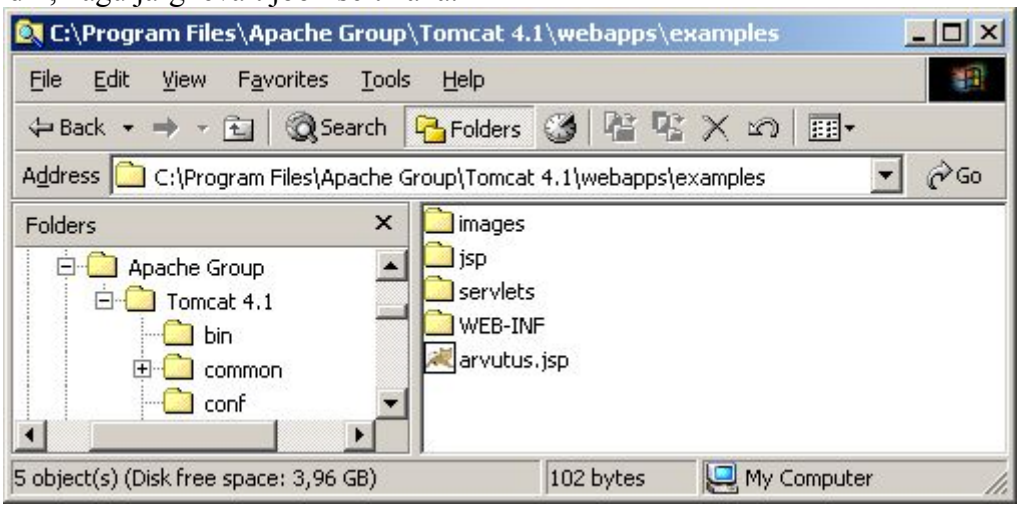
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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.

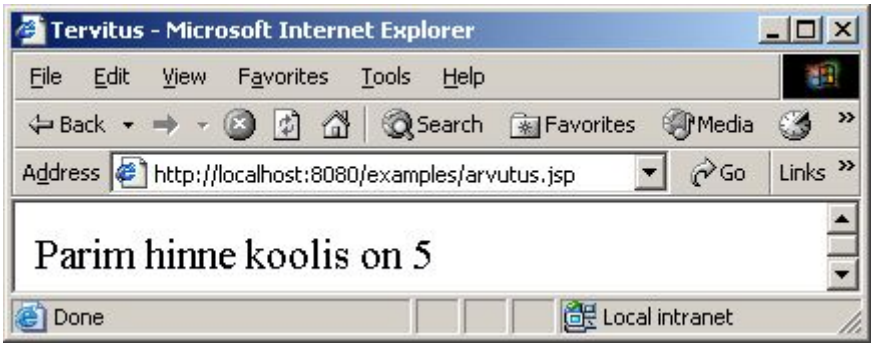
Järgnevalt on tegemist võimalikult lihtsa näitega, kus lihtsalt arvutatase kokku kahe arvu summa.

```
<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

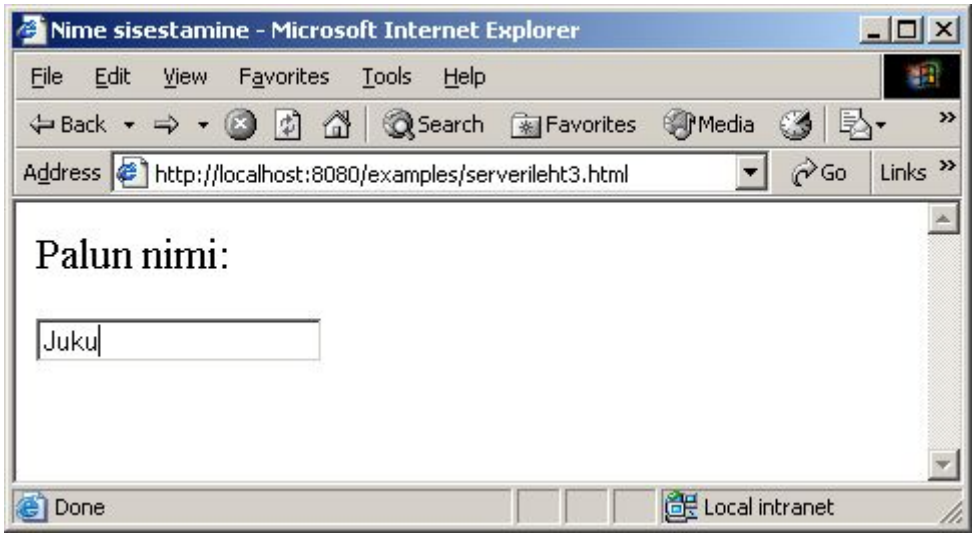
JSP lehe sisse saab Java koodi täiesti rahumeeles kirjutada. Et lõppkokkuvõttes muudetakse JSP-leht ikkagi servletiks, siis käivitamisel polegi nende vahel kuigivõrd vahet. Vaid kasutaja mugavuse mõttes kannatab JSP-lehel tavalist teksti kergemini edasi anda. Nõnda töötab harilik tsükkel

```
<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.

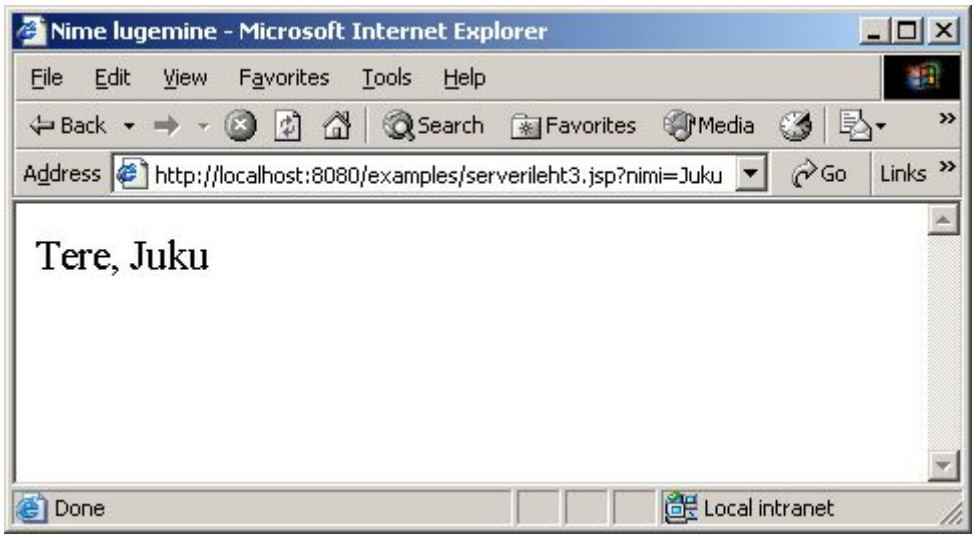
```
<html><head><title>Nime sisestamine</title></head>
<body>
 Palun nimi:
 <form action="serverileht3.jsp">
   <input type="text" name="nimi">
 </form>
</body></html>
```



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>
```

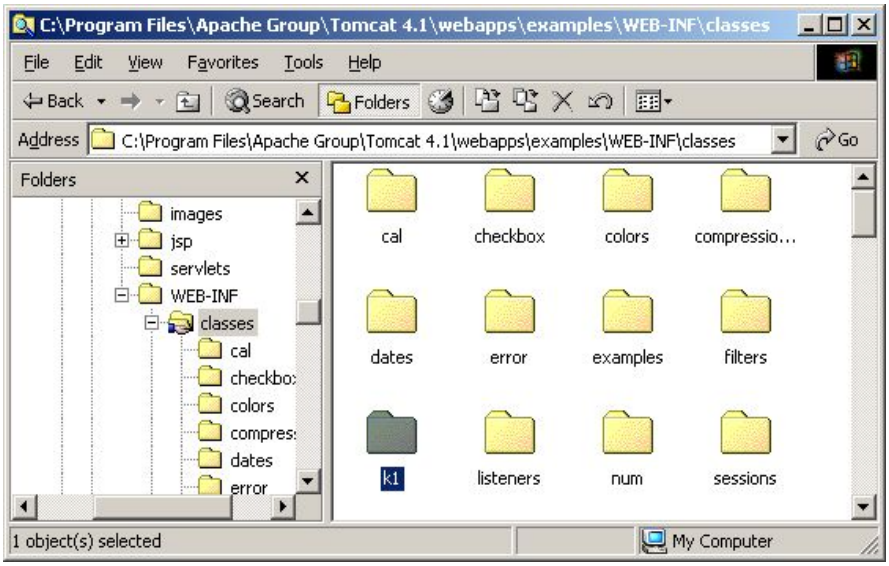
Uba

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.

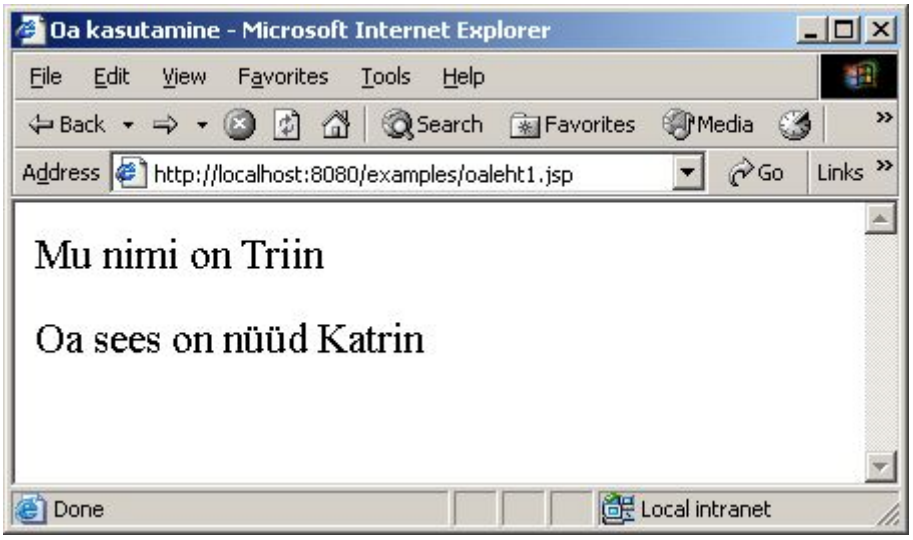
C:\Program Files\Apache Group\Tomcat 4.1\webapps\examples\WEB-INF\classes>javac k1\Uba1.java

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.



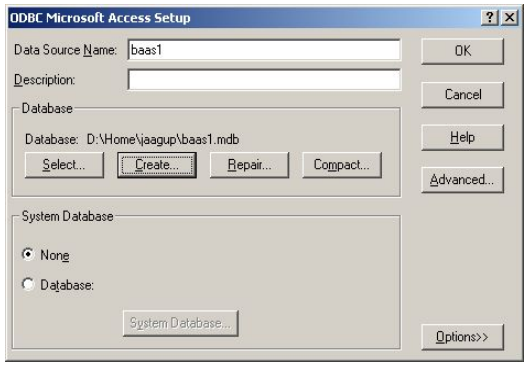
Vaikimisi on oa poole võimalik pöörduda vaid sama lehe avamise jooksul. Kui soovida aga andmeid pikemks talletada, võib määrata skoobiks sessiooni. Nõnda püsivad andmed paigal sama kasutaja mitme järjestikuse pöördumise ajal ning seal võib meeles pidada näiteks teadet, et kasutaja on end juba sisse meldinud.

```
<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");
  }
}
```

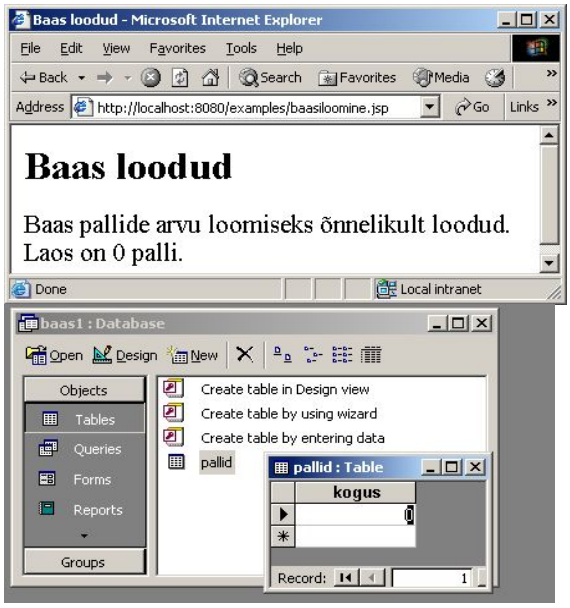
Kui paketi sees olev uba loodud, tuleb ta kompileerida nagu Java-fail ikka.

```
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.

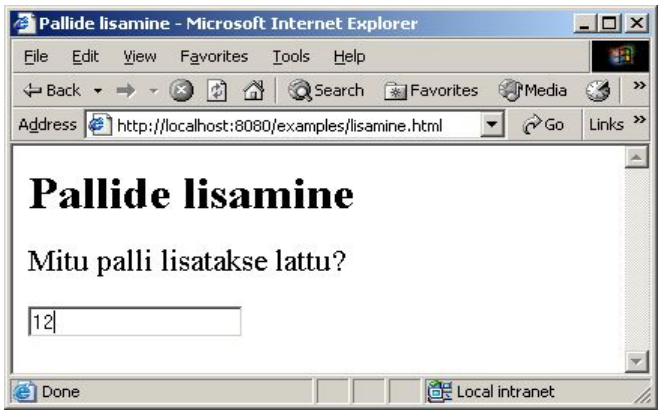
```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<% pallibaas.looBaas(); %>
<html><head><title>Baas loodud</title></head>
<body><h2>Baas loodud</h2>
 Baas pallide arvu loomiseks õnnelikult loodud.
 Laos on <jsp:getProperty name="pallibaas" property="palliarv"/> palli.
</body></html>
```

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.

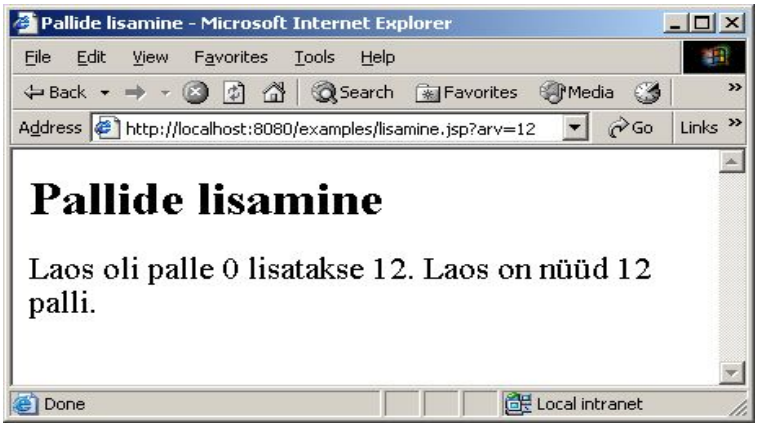
```
<html><head><title>Pallide lisamine</title></head>
<body><h2>Pallide lisamine</h2>
 Mitu palli lisatakse lattu? <br>
 <form name="vorm1" action="lisamine.jsp">
   <input type="text" name="arv">
 </form>
</body></html>
```



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

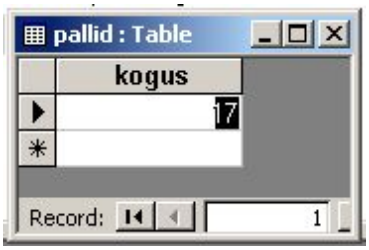
```
<jsp:useBean id="pallibaas" class="k1.Baasiuba1" />
<html><head><title>Pallide lisamine</title></head>
<body><h2>Pallide lisamine</h2>
 <%
  int olemas=pallibaas.getPalliarv();
  int juurde=Integer.parseInt(request.getParameter("arv"));
  int kokku=olemas+juurde;
  out.println("Laos oli palle "+olemas+"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