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

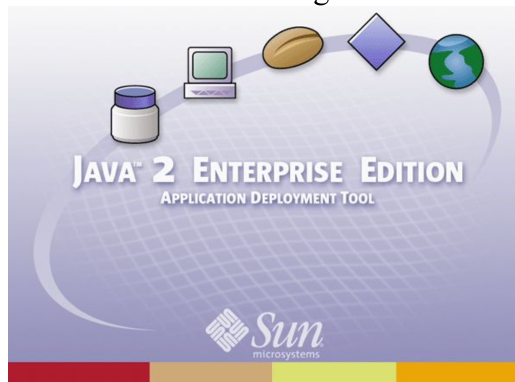
```
C:\jaagup\java\EJBArvutus>j2ee -verbose
J2EE server listen port: 1050
Naming service started:1050
Binding DataSource, name = jdbc/DB1, url =
jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = jdbc/DB2, url =
jdbc:cloudscape:rmi:CloudscapeDB;create=true
Binding DataSource, name = 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.
```

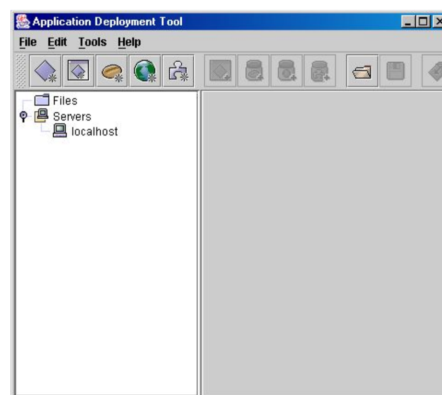
Ülespanek

Paljast serverist veel ei piisa. Loodud failidest tuleb rakendus kokku panna ning alles seejärel tööle lükata. Selle tarvis on loodud deploytool

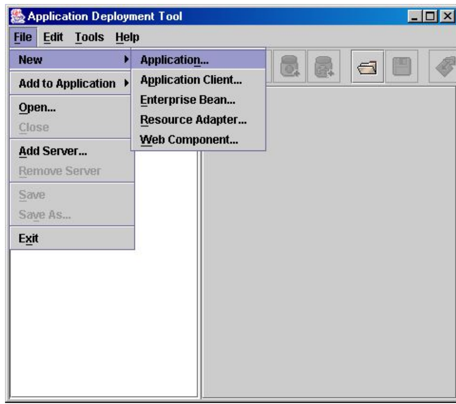
```
C:\jaagup\java\EJBArvutus>start cmd

C:\jaagup\java\EJBArvutus>deploytool
Starting Deployment tool, version 1.3.1
(Type 'deploytool -help' for command line options.)
```

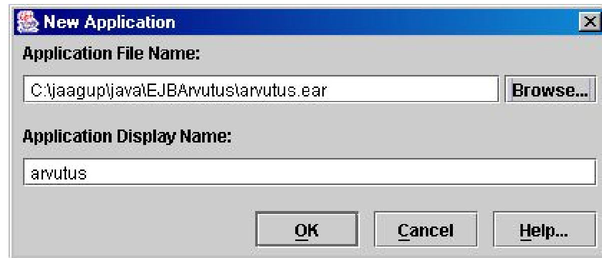
Mõningase mõttepausi järel avanebki esmalt ilus värviline pilt ning seejärel juba kasutatav deploytool. Esmasel käivitamisel pole seal küll veel töötavaid komponente, selle installeerimiseks aga me ta avasimegi.



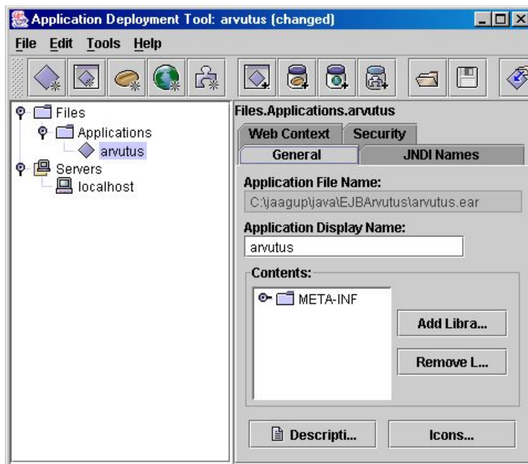


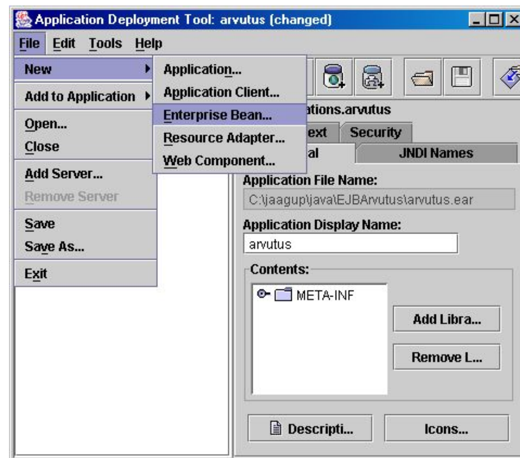
Kõigepealt tuleb luua uus rakendus, mille sisse on siis edaspidi võimalik asuda komponente lisama. Rakenduse puhul tuleb märkida fail, mis asub eneses teavet hoidma. Samuti tuleb määrata rakendusele nimi, mille kaudu teda kutsuda



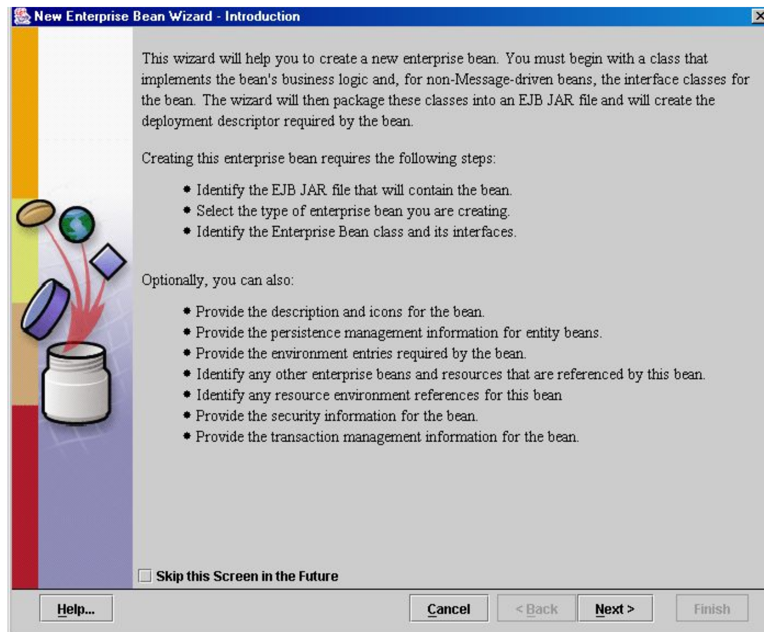


Kui tühi rakendus loodud, näeb pilt välja ligikaudu järgmine nagu vasakul. Tühjast rakendusest pole veel kasu kellelegi. Järgnevalt asume rakendusse lisama Enterprise Java Beani ehk äriuba.

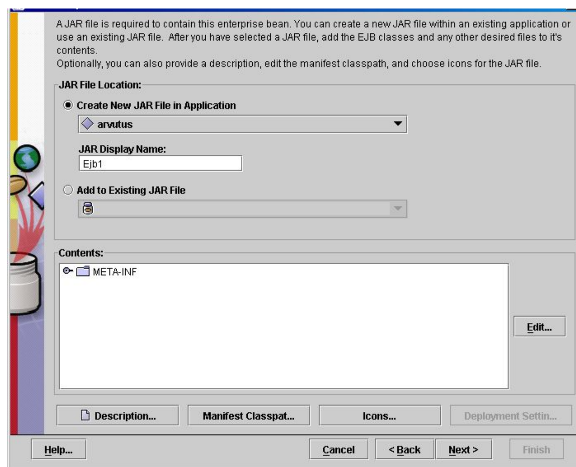


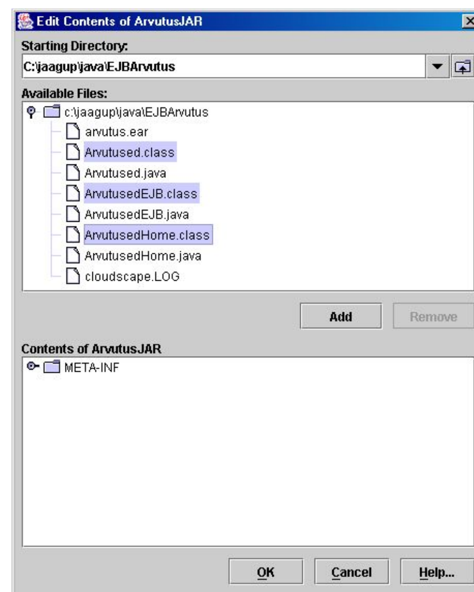


Asunud uba looma, antakse esimesel lehel kohe küllalt pikk seletus, millega tegemist.

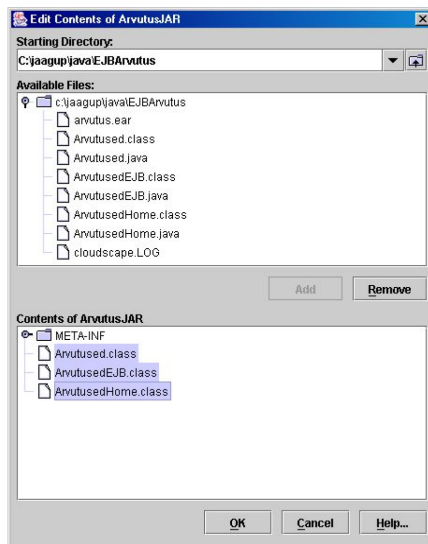


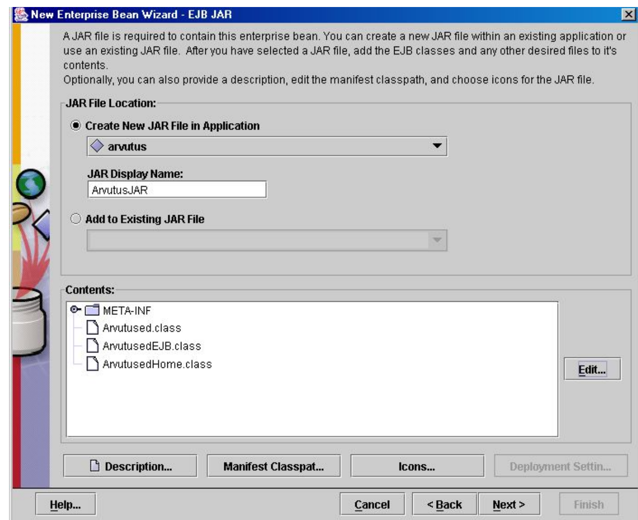
Liikunud Next-iga järgmisele lehele, tuleb asuda määrama nii oa nime kui sinna sisse kuuluvate failide loetelu. Contents – Edit alt tuleb välja järgmine dialoogiaken. Sealt tuleb siis valida oma rakenduse käivitamiseks tarvilikud failid, milleks on .class-id nii käskude kirjedamiseks, oa loomiseks kui ülesande tegelikuks lahendamiseks.



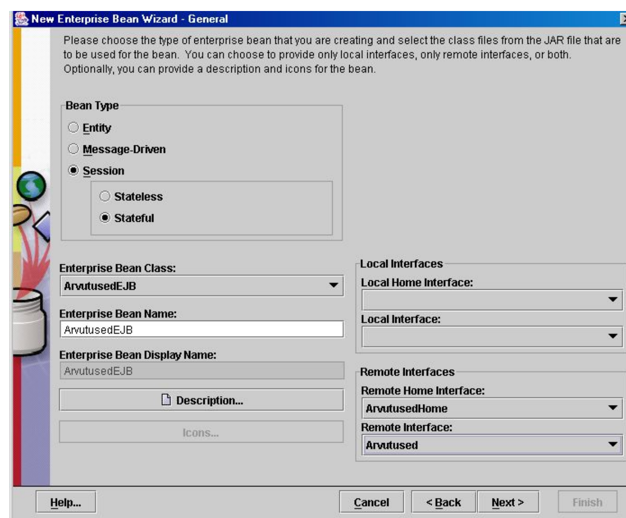


Add-nupule vajutades jõuavad nad oa JAR-faili. Seejärel OK-le vajutades võib faililoendit näha juba järgmises aknas. Jar-failile tuleb ka rakenduse sees kasutamiseks nimi anda. Siin on selleks pakutud ArvutusJAR.

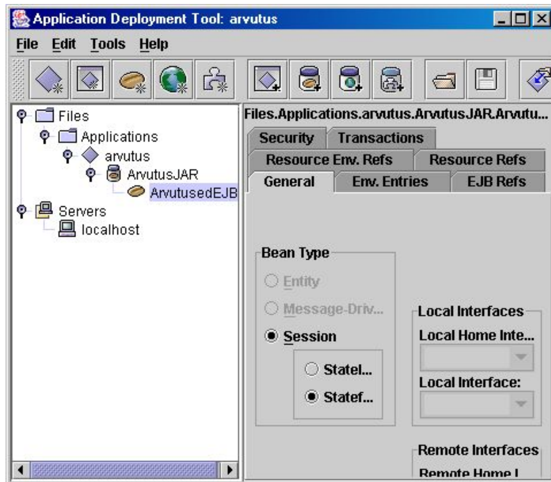


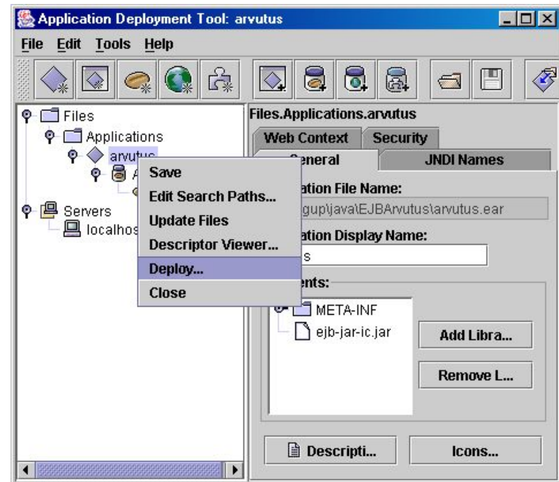


Rakendusserver ei pruugi teada, millist klassi või liidest kavatseme kasutada oa loomiseks, millist oskuste kirjeldamiseks ning millist tegelikuks arvutamiseks. Need tuleb sinna kõik ükshaaval ette ütelda. RemoteHome juurde tuleb oa create-meetodiga liides, Remote juurde käskude loetelu ning Enterprise Bean klassiks saab loodud SessionBean alamklass.

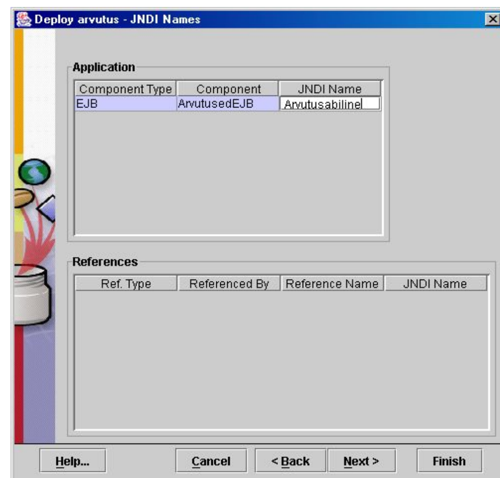


Kui määrangud seatud, võib next-i ja finishiga lõppu minna ning deploytoolis imetleda omavalmistatud uba. Et loodu ka teistele kättesaadavaks saaks, tuleb rakendus serverisse üles panna. Selleks lähen arvutus-nimelise rakenduse peale, vajutan paremat klahvi ning valin deploy.

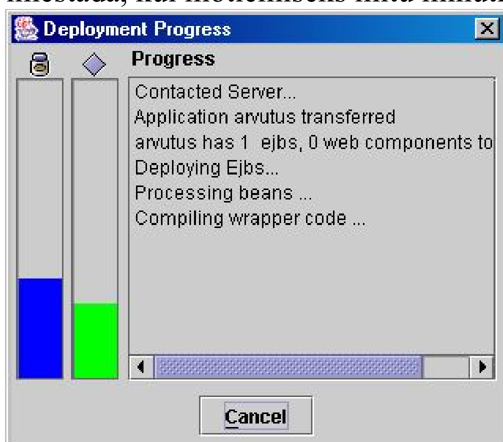


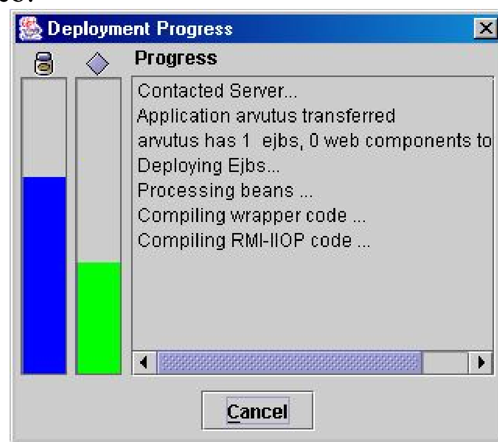


Edasi küsitakse faili nime, millesse andmed kanda. Et õnnestuks pärast rakendust käsurealt käivitada, selleks peab sees olema ristike ruudus “Return Client Jar”. Samuti peab serveris välja pakutud komponendi kohta ütlema, millise nime alt seda tellida saab. Siin on JNDI nimeks pandud Arvutusabiline.

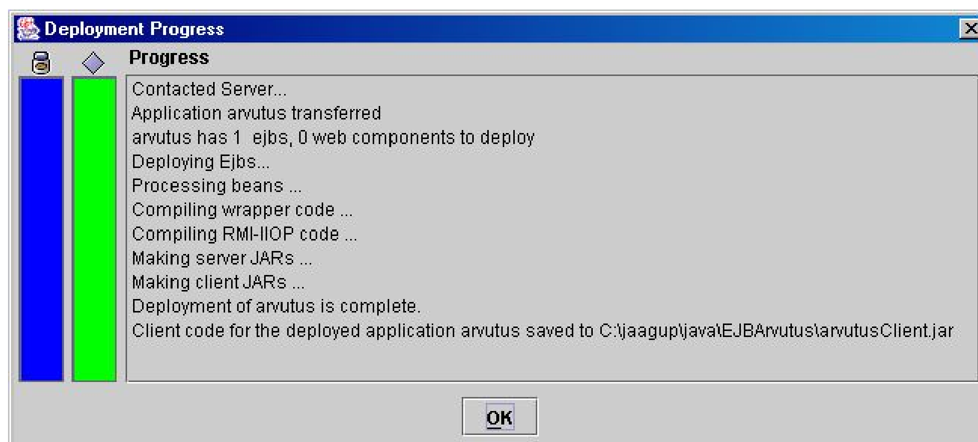


Edasi pole muud kui finish ning kompileerima. Tööd on masinal kõvasti, nii et pole imestada, kui mõtlemiseks mitu minutit läheb.





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>"
      );
    }
}
```

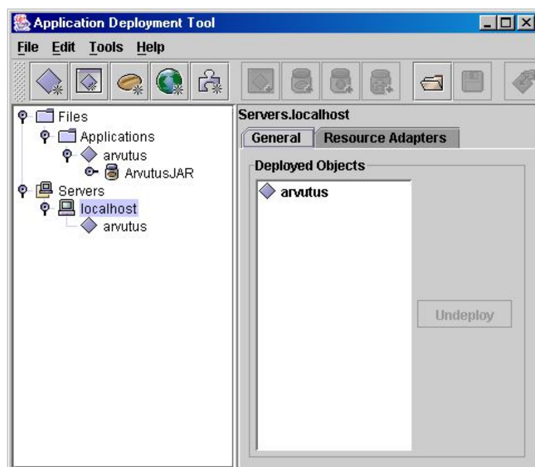
Nagu iga kood, tuleb ka see kompileerida.

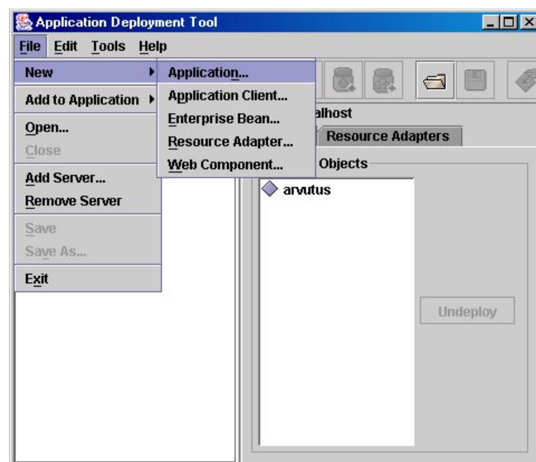
```
C:\jaagup\java\servlet>javac Servlet1.java
```

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
```

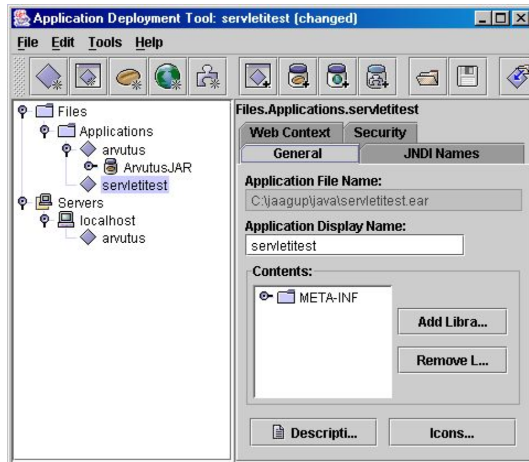
Loodud servlet peaks töötama mitmesugustes konteinerites, kaasa arvatud tolles, mis SUNi poolt EJB tarvis tasuta kaasa antakse. Esialgu on näha vaid üks installeeritud Arvutuse-nimeline rakendus. Iseenesest on võimalik loodud servletti ka olemasolevale rakendusele lisada, kuid siin näites teeme eraldi rakenduse. Sellisel juhul on osi kergem lisada ja eemaldada.

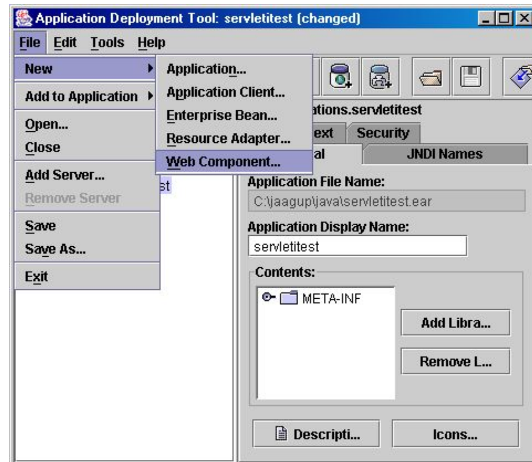




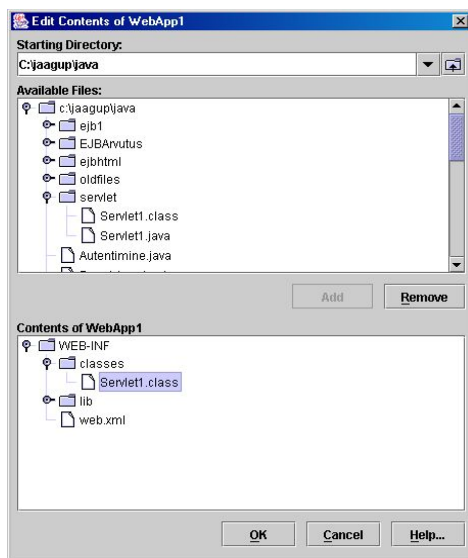
Rakenduse tarvis küsitakse loodava faili nime, samuti nime rakendusele enesele.

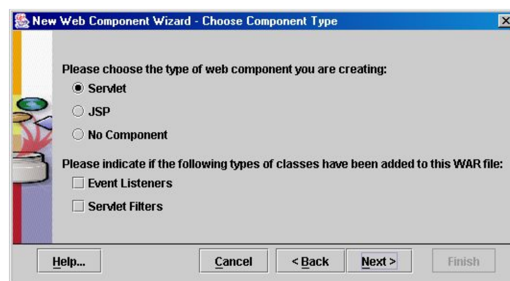
Uue tühja rakenduse loomine õnnestus kergesti. Servleti töö nägemiseks tuleb luua uus Web Component.





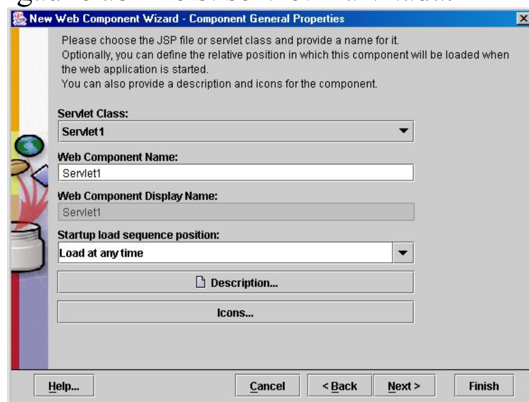
Sinna sisse valida servleti kompileeritud fail. Juhul, kui rakendus vajaks rohkem faile, annab need kõik sobivasse kataloogi paigutada.

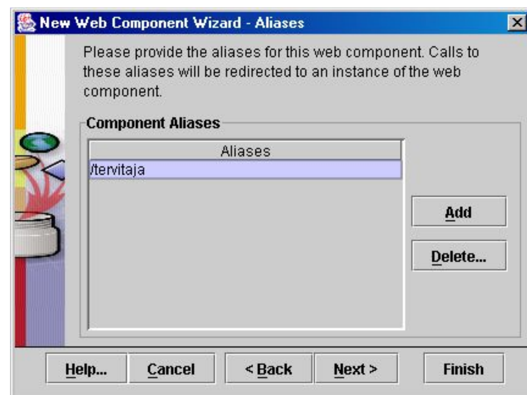




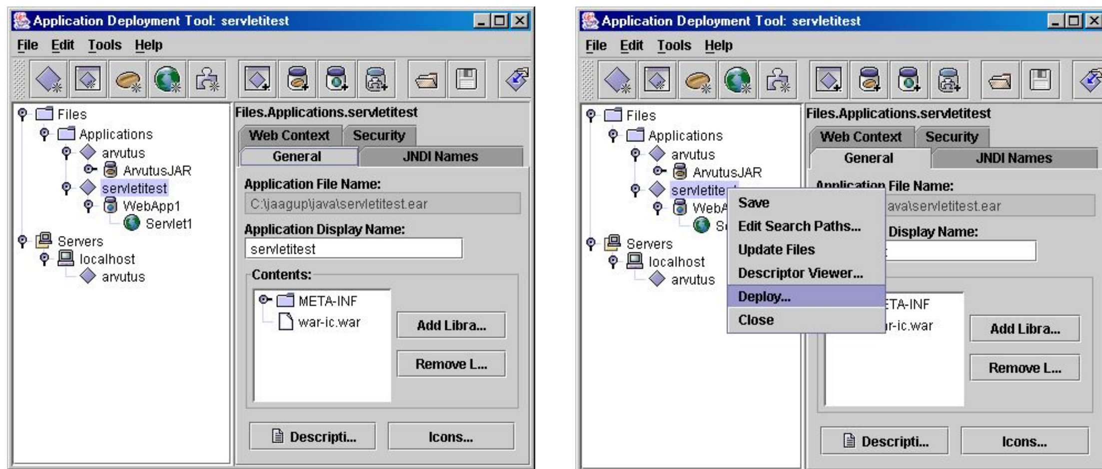
Edasi liikudes tuleb jälgida, et komponendi tüübiks oleks servlet.

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.

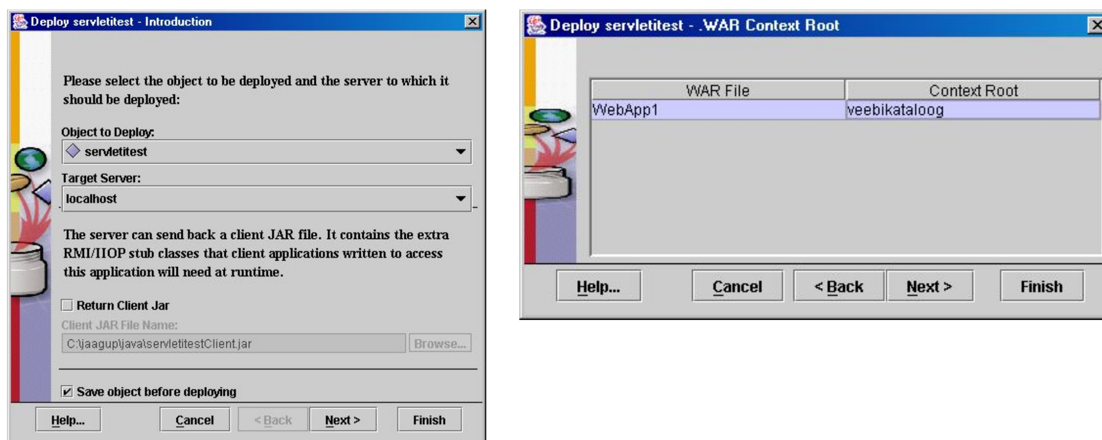




Kui järgnevalt finish vajutada, siis võib näha servlettest'i küljes olevat WebApp1'te, mille all omakorda Servlet1. Rakenduse serveris käivitamiseks tuleb öelda deploy.



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.