Vood, failid

Vood

Voogude kasutamise abil saab sarnaselt andmeid lugeda nii failist, Internetist, sõnest, baidimassiivist, teiselt lõimelt kui ka klaviatuurilt. Samuti saab voo abil andmeid ühtmoodi väljastada. Vaid voo loomisel piisab märkida, kuhu ta suunatakse ning meetodid lugemiseks ning kirjutamiseks on edaspidi sarnased. Kui on vaja andmete päritolu- või sihtkohta muuta, piisab vaid muutusest voo loomisel. Nii saab näiteks Internetiühenduseta arvutis katsetada võrgust lugevat programmi, andes talle tegelikult andmed ette failist.

Põhilised sisendi-väljundi klassid asuvad paketis `java.io`. Tähtede lugemiseks mõeldud klassid lõppevad sõnaga Reader, kirjutamiseks — Writer. Näiteks failist tähtede lugemiseks sobib `FileReader`, faili kirjutamiseks `FileWriter`. Baitide kirjutamiseks on OutputStream, lugemiseks InputStream. Faili puhul siis vastavalt `FileOutputStream` ning `FileInputStream`.

Sisimas käib andmevahetus baitide kaupa, kasutamismugavuse huvides on aga loodud selle ümber mitmesuguseid kesti. Nii aitab näiteks `DataOutputStream` muuta baitideks nii täisarvud, reaalarvud kui tõeväärtused.

Internetis paikneva faili lugemine

Baidivoo abil saab enese käsutusse soovikohase Internetti välja pandud faili. Luues aadressile vastava URLi objekti ning avades selle ühenduse, võib ühenduselt küsida sisendvoo, millelt saab baidi kaupa vastava faili andmeid lugeda. Siin näites kirjutan saabuvad baidid töökataloogis asuvasse faili, programmi töö tulemusena saan kaugel asuvast failist omale koopia.

```
import java.io.*;
import java.net.URL;
public class Voog3c{
  public static void main(String argumendid[]) throws IOException{
    String aadress="http://www.tpu.ee/plogo.GIF";
    InputStream sisse=
      new URL(aadress).openConnection().getInputStream();
    OutputStream valja=new FileOutputStream("pilt.gif");
    int nr=sisse.read();
    while(nr>=0){
      valja.write(nr);
      nr=sisse.read();
    }
    sisse.close();
    valja.close();
  }
}
```

Teksti lugemine

Teksti saab rea kaupa lugeda klassi `BufferedReader` abil. Selle konstruktor vajab parameetriks klassi `Reader` järglast, näiteks `FileReader`it, kui soovitakse failist lugeda. Voo lõpu puhul antakse lugemisel rea väärtuseks `null` (tühiväärtus).

```
import java.io.*;
public class Voog4{
  public static void main(String argumendid[]) throws IOException{
    BufferedReader sisse=new BufferedReader(
      new FileReader("andmed.txt")
    );
    String rida=sisse.readLine();
    System.out.println("Faili esimene rida on: "+rida);
    System.out.println("Nüüd järjestikku käik faili read:");
    while(rida!=null){
```

```
      System.out.println(rida);
      rida=sisse.readLine();
    }
  }
}
```

`InputStreami` (baidivoo) saab `Reader`iks (tähevoog) muundada klassi `InputStreamReader` abil. Näiteks võrgust lugemisel annab pistik vaid baidivoo, sellest teksti lugemisel on aga soovitav muuta see enne tekstivooks.

```
Socket sc=new Socket("madli.ut.ee", 13);
BufferedReader sisse=new BufferedReader(
  new InputStreamReader(sc.getInputStream())
);
```

Teksti kirjutamine

Teksti soovitatakse kirjutada klassi `PrintWriter` abil. Tema loomisel võib talle parameetriks anda nii `OutputStream`i kui `Writer`i. `PrintWriter` tunneb ise ära, millise vooga on tegemist ning vastavalt sellele saadab sinna andmeid.

```
import java.io.*;
public class Voog5{
  public static void main(String argumendid[]) throws IOException{
    PrintWriter valja=new PrintWriter(new FileWriter("ruudud.txt"));
    for(int nr=1; nr<=100; nr++){
      valja.println(nr*nr);
    }
    valja.close();
  }
}
```

Tavaliselt kogub `PrintWriter` välja saadetavad andmed kokku ning alles puhvri täitumisel või voo sulgemisel saadab andmed kohale. Nii kulub vähem ressursse, sest sageli (näiteks Internetis) kulub ühe baidi või terve bloki andmete saatmiseks ühepalju energiat, sest ülekantavad blokid on kindla pikkusega ning juhul kui saadetakse vähem andmeid kui blokki mahub, siis täidetakse ülejäänud bloki sisu "aherainega".

Andmete teele saatmiseks saab voole öelda `flush()`. Et teade alati kohe peale kirjutamist teele läheks, selleks tuleb `PrintWriter`i konstruktorisse lisada `true`.

Klaviatuur ning ekraan

Mitmetes operatsioonisüsteemides saab klahvistikult lugeda ning ekraanile saata andmeid voona, ka Java keelde on selline lähenemine üle võetud. Nendele voogudele pääseb ligi klassi `System` väljade vastavalt `in` ja `out` kaudu. Nii tuleb tulemuse tekstiekraanile väljastamiseks saata seal paiknevad tähed voogu `System.out`. Klaviatuurilt lugemiseks aga tuleb püüda baite voost `System.in`. Klaviatuurilt saab väärtusi sisestada vaid sõnena (erinevalt näiteks C-st või Pascalist, kus võib kohe ette määrata, millise tüübi sisse andmed loetakse). Juhul kui meil on vaja sisestatut interpreteerida mõne teise tüübina, siis tuleb tüüp vastavate meetodite abil muuta. Sõne püüab täisarvuliseks objektiks muuta klassi `Integer` meetod `parseInt`.

```
import java.io.*;
public class Sisse1{
  public static void main(String argumendid[]) throws Exception{
    int arv;
    double distants;
    String s;
    PrintWriter valja=new PrintWriter(System.out, true);
    BufferedReader sisse=new BufferedReader(
      new InputStreamReader(System.in)
    );
```

```
    valja.println("Osalejate arv:");
    s=sisse.readLine();
    arv=Integer.parseInt(s);
    valja.println("Vahemaa:");
    distants=Double.parseDouble(sisse.readLine());
    valja.println("Kokku läbiti "+arv*distants+" km");
  }
}
```

Väljund:

```
D:\Kasutajad\jaagup\java>javac Sisse1.java

D:\Kasutajad\jaagup\java>java Sisse1
Osalejate arv:
5
Vahemaa:
22.2
Kokku liiguti 111.0 km

D:\Kasutajad\jaagup\java>
```

Käsklus import `java.io.*` lubab kasutada kõiki paketti `java.io` kuuluvaid klasse. Siin programmis on neist kasutatud `PrintWriter`, `BufferedReader` ja `InputStreamReader`. `InputStreamReader`'it kasutatakse selles progammis sisendvoost `System.in` saabuva `BufferedReader`'ile "söödavaks muutmiseks".

Sõnevoog

Kuna sõne pikkusel Java keeles pole piirangut (2 miljardit tähte mida neljabaidine täisarv lubab on tunduvalt rohkem, kui tavaliste tekstide puhul võib ette tulla), siis saab ka suuremad andmed (näiteks failitäie teksti) sõnesse paigutada. Sõnesse kirjutamiseks sobib `StringWriter`, kuhu saab kirjutusvoo suunata samuti nagu mõne muu `Writer`i (näiteks `FileWriter`i) sisse. Sõne saab sellest kätte meetodiga `toString()`. `StringReader`i abil saab pikast sõnest lugeda nagu voost, konstruktorina tuleb talle anda sõne, mille andmeid soovitakse voona lugema hakata.

```
import java.io.*;
public class SonevooLugeja{
  public static void main(String[] argumendid) throws IOException{
    StringReader sisendvoog=new StringReader(
      "Tekst,\nmis simuleerib\nsisestust failist");
    BufferedReader sisse=new BufferedReader(sisendvoog);
    String rida=sisse.readLine();
    while(rida!=null){
       System.out.println(rida);
       rida=sisse.readLine();
    }
    sisse.close();
  }
}
```

```
D:\Kasutajad\jaagup\java>java SonevooLugeja
Tekst,
mis simuleerib
sisestust failist
```

zip-faili loomine

Ka zip-faile saab Java abil luua ilma, et peaks ise faili struktuuri uurima. `ZipOutputStream`ile tuleb anda väljundvoog pakitud andmete väljasaatmiseks, teda ennast võib aga kasutada nagu iga muud väljundvoogu, näiteks `PrintWriter`i abil temasse andmeid kirjutades. Meetod `putNextEntry` teatab, et nüüd hakkavad tulema

järgmise arhiivi lisatava faili andmed. Parameetriks tuleb anda `ZipEntry`, mis sisaldab andmeid lisatava faili kohta, lihtsamal juhul vaid selle nime.

```
import java.io.*;
import java.util.zip.*;
public class Voog6{
  public static void main(String argumendid[]) throws IOException{
    ZipOutputStream zo=new ZipOutputStream(
      new FileOutputStream("nimed.zip")
    );
    PrintWriter valja=new PrintWriter(zo, true);
    zo.putNextEntry(new ZipEntry("nimed.txt"));
    valja.println("Juku");
    valja.println("Kaarel");
    zo.putNextEntry(new ZipEntry("kirjeldus.txt"));
    valja.println("Korvpallimeeskonna varumängijad");
    valja.close();
  }
}
```

Sarnasel põhimõttel on võimalik kirjutada ka jar-formaadis arhiivifaile. Samuti saab ise luua filtri, mis andmed meile sobivalt muudab. Kui kirjutada filtrit, mis saadaks tekstist edasi vaid numbrid, tuleb lihtsalt iga tähe puhul vaadata, kas tegemist on numbriga ning siis vastavalt ta kas edasi saata või mitte.

Voogude kokkuliitmine

SequenceInputStream aitab kokku liita mitmest voost tulevad andmed. Senikaua võetakse esimesest voost, kuni see on tühi, siis minnakse järgmise voo juurde. Viimase voo ammendumisega saab ka SequenceInputStream tühjaks.

Isendite lugemine ja kirjutamine

Voogu on võimalik kirjutada ja sealt lugeda ka terveid objekte (ehk isendeid) klasside ObjectInputStream ning ObjectOutputStream abil. Nii saab säilitada või mujale mööda voogu edasi anda näiteks punkte, pilte või ka keerulisemate komponentide olekuid ilma, et peaks teadmagi, kuidas komponendid tehtud on.

```
import java.io.*;
import java.awt.Point;
import java.util.Date;
public class Voog7{
  public static void main(String argumendid[]) throws IOException{
    ObjectOutputStream valja=new ObjectOutputStream(
      new FileOutputStream("objektid.dat")
    );
    valja.writeObject(new Point(3, 2));
    valja.writeObject(new String("Kirjutamise aeg"));
    Date praegu=new Date();
    valja.writeObject(praegu);
    valja.close();
  }
}
```

```
import java.io.*;
import java.awt.Point;
import java.util.Date;
public class Voog7a{
  public static void main(String argumendid[]) throws Exception{
    ObjectInputStream sisse=new ObjectInputStream(
      new FileInputStream("objektid.dat")
    );
    Point p=(Point)sisse.readObject();
    String s=(String)sisse.readObject();
    Date aeg=(Date)sisse.readObject();
    sisse.close();
    System.out.println(p+" "+s+" "+aeg);
  }
}
```

Ka klasse (ning koos nendega alamprogramme) on võimalik voogu mööda transportida. Nii on võimalik omale võrku mööda kohale laadida meetodid, mida kohalikus masinas olemas pole.

Failid ja kataloogid

Nii failide kui kataloogidega tegelemiseks on Java keeles klass `File`. Selle abil saab kontrollida faili pikkust, loomise ning muutmise aega. Samuti faile luua, kustutada ning ümber nimetada. Saab kontrollida, kas fail on olemas, kas sinna saab lugeda või kirjutada. Kataloogi puhul saab küsida samas kataloogis asuvate failide nimesid, luua alamkatalooge.

Andmed faili kohta

Klassi `Fail1` `main`-meetodis uuritakse, kas fail nimega “nimed.txt” leidub. Juhul kui jah, siis kirjutatakse välja ta nimi, pikkus ning viimane muutmisaeg.

```
import java.io.*;
import java.util.Date;
public class Fail1{
  public static void main(String argumendid[]) throws IOException{
    File fail=new File("nimed.txt");
    if(fail.exists()){
      System.out.println(
        "Faili "+fail.getName()+" pikkus on "+fail.length()+
        " baiti. Viimati muudeti seda "+new Date(fail.lastModified())
      );
    }
  }
}
```

Kataloogi sisu päring

Kataloogiosuti luuakse nagu failiosuti, s.t. antakse konstruktorisse vastava faili või kataloogi nimi. Ühe punktiga tähistatakse jooksvat kataloogi ning kahe punktiga ülemkataloogi. Alles spetsiifiliste meetodite rakendamisel kontrollitakse, kas tegemist on faili või kataloogiga, s.t. `list()` saab öelda vaid kataloogidele, meetod väljastab selles kataloogis asuvate failide nimed sõnemassiivina.

```
import java.io.*;
public class Kataloog1{
 public static void main(String argumendid[]){
   File kataloog=new File("."); // . on jooksev kataloog
   String failid[]=kataloog.list();
   System.out.println("Kodukataloogis asuvad failid on:");
   for(int i=0;i<failid.length; i++){
     System.out.println(failid[i]);
   }
   System.out.println("Laiendiga .txt on neist:");
   for(int i=0; i<failid.length; i++){
     if(failid[i].endsWith(".txt"))
        System.out.println(failid[i]);
   }
 }
}
```

Kataloogi loomine

Uue kataloogi aitab luua käsk `mkdir()`. Vastloodud kataloogi saab kasutada nagu iga muud juba olemas olevat kataloogi, s.t. sinna faile kirjutada ning sealt lugeda. File.separator annab faili otsingutee eraldaja vastavalt operatsioonisüsteemile (s.t. "\" Dos/Windowsi ning "/" Unixi puhul). Kui soovitakse korraga luua mitu (rohkem kui üks) üksteise sees asuvat kataloogi, siis selleks on käsk `mkdirs()`.

```
import java.io.*;
public class Kataloog2{
  public static void main(String argumendid[]) throws IOException{
```

```
    File kataloog=new File("uus");
    kataloog.mkdir();
    PrintWriter valja=new PrintWriter(
      new FileWriter("uus"+File.separator+"katse.txt")
    );
    valja.println("Fail katsetamiseks");
    valja.close();
  }
}
```

Kataloogi kustutamine

Nii faile kui katalooge saab ka kustutada. Kataloogi kustutamiseks peab ta enne olema seest tühi. Kõigepealt kustutatakse kataloogis uus asuvad failid ning seejärel kataloog ise.

```
import java.io.*;
public class Kataloog3{
  public static void main(String argumendid[]){
    File kataloog=new File("uus");
    if(kataloog.isDirectory()){
      String[] failid=kataloog.list();
      for(int i=0; i<failid.length; i++){
        new File(kataloog+File.separator+failid[i]).delete();
        System.out.println(failid[i]);
      }
      kataloog.delete();
    }
  }
}
```

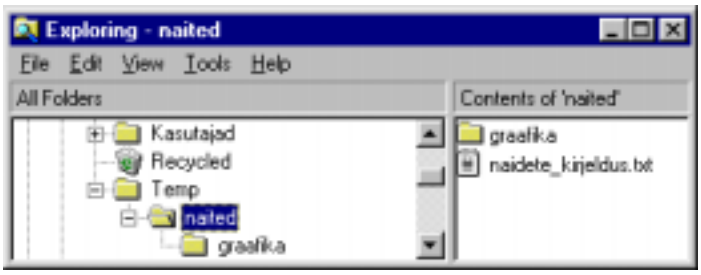
Alamkataloogide andmed

Rekursioon

Kui soovida teada, palju on mõne kataloogi all andmeid ning kui palju saaks kettale katakoogi kustutamisel vaba ruumi juurde, siis ei piisa vaid kataloogi sees olevate failide suuruste kokku liitmisest. Ka ei piisa vaid sellest kui otsida üles vaid kataloogis olevad alamkataloogid ning nende seest failide pikkused kokku liita. Ka alamkataloogides võivad omakorda olla alamkataloogid ja nii päris mitu taset edasi. Et kõigile neile ligi pääseda, tuleb kasutada mõnd kavalamat võtet. Üheks võimaluseks on teha alamprogramm, millele antakse ette kataloogi nimi. Alamprogrammi ülesandeks oleks teha etteantud kataloogi failidega soovitud töö (näiteks lugeda kokku failide maht) ning juhul, kui töö käigus selgub, et etteantud kataloogis on omakorda alamkatalooge, siis paluda uuesti seesama alamprogramm välja kutsuda ning paluda tal vahepeal leitud alamkataloogiga sama töö ära teha. Kui alamprogrammile antakse ette vaid faile sisaldav kataloog, siis pole tal põhjust kusagile sügavamale minna. Kui kataloogi sisse on sattunud vaid faile (ning mitte teisi katalooge) sisaldavaid alamkatalooge, siis juhtub selle alamprogrammi täitmine olema kõige rohkem kahes kohas korraga: üks eksemplar uurimas alamkataloogi faile ning teine ootamas alamkataloogi nime juures, et saaks ülemkataloogi uurimisega edasi minna. Kui aga üksteise sees olevaid katalooge on rohkem, siis peab ka rohkem alamprogrammi eksemplare olema käiku pandud – iga taseme jaoks üks.

```
import java.io.*;
class Faililoetelu1{
  static void trykiKataloog(String kataloginimi){
     String failid[]=new File(kataloginimi).list();
     for(int i=0; i<failid.length; i++){
       String failinimi=kataloginimi+File.separator+failid[i];
```

```
      System.out.println(failinimi);
      if(new File(failinimi).isDirectory()){
        trykiKataloog(failinimi);
      }
    }
  }
  public static void main(String argumendid[]){
    trykiKataloog("d:\\temp");
  }
}
```




```
D:\Kasutajad\jaagup\java>java Faililoetelu1
d:\temp\1art-a.html
d:\temp\naited
d:\temp\naited\graafika
d:\temp\naited\graafika\hulknurk.txt
d:\temp\naited\naidete_kirjeldus.txt
```

Kataloogide nimistu

Alamkataloogide andmete kätte saamiseks on ka põhimõtte poolest küllalt teistsugune võimalus olemas. Siin tehakse mällu loetelu kataloogide nimedest, mida pole veel jõutud läbi vaadata. Iga kord, kui jõutakse läbi vaatamata kataloogini, pannakse selle nimi meelde. Kui parasjagu uuritud kataloog läbi saab, võetakse meelest järgmise kataloogi nimi ning asutakse selle sisu uurima. Esimest lähenemist nimetatakse rekursiooniks. Seda saab enamasti küllalt lühidalt kirja panna, samas võib mõnikord aga masinale raskeks käivitada osutuda, kui andmeid palju ning samast alamprogrammist palju eksemplare tööle pannakse. Tuhatkond korda kipub Pentium-100 arvutile piir olema, millest alates juba uute alamprogrammi eksemplaride väljakutse raskeks kipub minema. Kui aga otsitavate kataloogide nimed lihtsalt meeles hoida, siis ei saa arvuti ressursid mitte nii kergesti otsa.

Nimede meeles pidamiseks on appi võetud klass LinkedList. Tegu on kui massiiviga, kuid siin pole vaja ette öelda, palju elemente loetellu tuleb ning lisamisel või eemaldamisel muudetakse nimistu suurust automaatselt. Käsuga `add` lisatakse leitud kataloogi nimi listi lõppu, käsuga `removeFirst` küsitakse välja esimene ning eemaldatakse see loetelust.

```
import java.io.*;
import java.util.LinkedList;
class Faililoetelu2{
  static void trykiKataloog(String kataloginimi){
    LinkedList l=new LinkedList();
    l.add(kataloginimi);
    while(l.size()>0){
      String failinimi=(String)l.removeFirst();
      System.out.println(failinimi);
      if(new File(failinimi).isDirectory()){
        String[] failid=new File(failinimi).list();
        for(int i=0; i<failid.length; i++){
          l.add(failinimi+File.separator+failid[i]);
        }
      }
    }
```

```
  }
  public static void main(String argumendid[]){
    trykiKataloog("d:\\temp");
  }
}
```

```
D:\Kasutajad\jaagup\java>java Faililoetelu2
d:\temp
d:\temp\1art-a.html
d:\temp\naited
d:\temp\naited\graafika
d:\temp\naited\naidete_kirjeldus.txt
d:\temp\naited\graafika\hulknurk.txt
```

Kui kahe katalooge uuriva programmi väljundeid võrrelda, siis on näha, et esimesel juhul trükitakse iga alamkataloogi sisu sinna järele, kus kataloogi enesegi nimi on. Teisel juhul aga pannakse kataloogi nimi meelde ning alles pärast hakatakse selle sisu uurima. Sellepärast siin näites kataloogis graafika paikneva faili hulknurk.txt nimi ongi alles pärast eelmist faili. Kui andmepuu juhtub suurem olema, siis võib pealtnäha kokkukuuluvate andmete kaugus üksteisest päris suureks osutuda. Kui andmete vaatamise järjekorral pole tähtsust (näiteks failisuuruste summa arvutamise juures), siis pole vaja lasta end häirida. Samuti on niimoodi ühe kataloogi seest leitud failid ilusti koos.

Kokkuvõte

Voogude abil saab andmeid vahetada ühtmoodi mitmesuguste lähte- ning sihtkohtade vahel. Põhiliselt on tegemist faili, klaviatuuri/ekraani, mälu, toru ning võrguga, kuid vajadusel saab ka muid ühendusi (näiteks printeriga) luua. Voos liiguvad andmed jadana. Voo sisimas on tegemist baidijadaga, kuid neid saab interpreteerida vastavalt kasutaja vajadustele. `Reader`- ja `Writer` klassid on tähtede (sõnede) lugemiseks ja kirjutamiseks. `DataInputStream`i ja `DataOutputStream`i abil saab lugeda ja kirjutada lihttüüpe. Sõnede ridade kaupa lugemisel on lõputunnuseks tühiväärtus `null`. Baitide puhul väljastatakse lõpu puhul –1. `ObjectInputStream`i ning `ObjectOutputStream`i abil saab lugeda ja kirjutada objekte. Filtrite abil saab kasutada näiteks pakitud faile, samuti sidet kodeerida. Ka filtreid saab ise luua.

Voogude abil saab vaid järjest lugeda või kirjutada. Andmevahetuse mitmed meetodid võivad heita erindeid. Seetõttu tuleb tegelda nende püüdmise või edasisaatmisega.

Failide ja kataloogide kohta andmete küsimiseks, nende kustutamiseks ning ümber nimetamiseks on klass File. Üksteises alanevaid katalooge aitab uurida rekursioon.

Ülesanded

Liitmistehted

- Väljasta tervitus tekstifaili.
- Väljasta tekstifaili arvud ühest sajani.
- Väljasta tekstifaili arvude ruudud ühest sajani.
- Väljasta tekstifaili viis juhuslike liidetavatega liitmisülesannet.

- Väljasta tekstifaili kasutaja pool määratud arv juhuslike liidetavatega liitmisülesannet, mille vastus ei ületaks kümmet.

Hinnetetabel

- Programm loob kasutaja poolt määratud värvi taustaga tühja HTML-tekstifaili.
- Tekstifailis on õpilaste nimed ning nende hinded. Programm loob nende põhjal HTML-faili, kus nimed ning nendele vastavad hinded on tabelis.
- Tekstifailis on igal real õpilase nimi ning hinded ainete kaupa. Ainete nimed on faili esimesel real. Igale õpilasele luuakse HTML leht, kus pealkirjaks on õpilase nimi ning tabelis ained ning nendes ainetes olevad hinded. Lisaks juhtleht viitega iga õpilase lehele.

Interneti vahendid Javas

Võrgu võimalused

Arvutid saab ühendada võrguks. Kohalik võrk võib koosneda kasvõi vaid kahest arvutist, mis omavahel ühendatud on ning mille abil saab teineteisele teateid ja dokumente saata. Võrgu kaudu saab transportida kõiksugu infot, mida on võimalik elektrisignaalideks muundada. Näiteks teksti, pilti ja heli. Üle võrgu transportides on tähtis ka ülekandmise aeg, et teine pool oleks valmis saadetavat infot vastu võtma.

Kahe kasutaja omavahelise reaalajas suhtlemise puhul on tähtis, et nad üheaegselt masina taha satuksid. Enamike teenuste korral aga töötab programm serveris pidevalt, kasutaja saab end vaid sobival ajal sinna ühendada ning siis andmeid saata ja küsida.

Ühenduse põhimõte

Füüsiliselt on arvutid ühendatud mitmesuguste juhtmetega (Ethernet, “keerupaar”, valguskaabel), juhtmed omakorda “sõlmede” (hub, switch) abil. Ühendusprotokollide abil on korraldatud nii, et programmidel andmete transpordiks piisab vaid teada arvutile antud aadressi (IP numbrit) või nime. Arvuti saab enese poole pöördumiseks kasutada nime localhost. Nii saab võrguprogramme katsetada ka ilma masinat võrku ühendamata.

Andmekoguse edasiandmiseks piisab andmeportsioni teele saatmisest koos sihtarvuti aadressiga, kus sihtarvuti programm selle kinni püüab (ühenduseta protokoll UDP datagrammide saatmiseks). Kui soovitakse andmete kohalejõudmist kontrollida või soovitakse muul põhjusel kahepoolset andmesidet, siis tuleks kasutada ühendusega protokolli (TCP), kus arvutite vahel luuakse ettekujutatav ühendusliin, mille kaudu saab andmeid saata ja lugeda.

Ühest arvutist võib samaaegselt olla selliseid ühendusi mitu. Näiteks FTP abil kopeeritakse faili teise arvutisse, Telneti aknast loetakse kirju ning jututoas suheldakse. Sel juhul on FTP programmil ühendus arvutiga kuhu kopeeritakse, Telneti programmil ühendus kirju hoidva masina vastava programmiga ning seiluri rakendil või muul jututoa klientprogrammil ühendus jututoa serveriga, kes kasutajatelt saabuvad teated kinni püüab ning teistele teateid kuulama määratud kasutajatele edasi saadab. Ka ühel programmil võib samaaegselt olla mitu ühendust teiste programmidega. Nii on jututoa keskuseks oleva masina programmil eraldi ühendus samaaegselt kõigi jututuppa meldinud kasutajate programmidega. Ta saab neilt saabuvaid andmeid lugeda ning omalt poolt kasutajatele saata. Serveripoolse programmi kood peab olema nii kirjutatud, et ta suudab samaaegselt mitmelt kasutajalt teateid lugeda ning otsustada, millisele liinile mida saata ning mida näiteks kettale logifailidesse kirjutada.

Ühenduse loomiseks seab ühes arvutis olev programm end ootele ning teisest arvutist saab ootajaga ühendust võtta. Kui nad ilusti ühele ajale sattusid, nii et ootaja jõudis ära oodata, millal teine ühendust palub, siis luuakse ilus kahepoolne sidekanal, mille kaudu saavad programmid andmeid vahetada.

Ühes masinas võib korraga suhtluspartnerit oodata mitu programmi. Masinas on ligikaudu 64000 väratit ehk porti, mille abil saab ühendust pidada. Tavalises lauaarvutis sellist ühenduste hulka harilikult vaja ei lähe, kuid näiteks asutuse serveris võib ühekorraga töös olevate ühenduste arv päris suureks minna.

Serverarvutis on harilikult alati töös mõned standardsed programmid, mis pakuvad kasutajatele teenuseid. Põhiliste teenuste juurde on välja kujunenud väratid, mille juures nad kasutajat ootavad. Serverarvutis kehtivat kellaaega näiteks teatab programm, mis peab väljastpoolt tuleva kasutajaga ühendust värati nr. 13 abil. Kasutajate kohta andmeid jagav finger ootab tavaliselt väratis number 79 ning www lehekülgi näidatakse värati 80 kaudu. Neid numbreid saab vajadusel muuta, kuid sel juhul peab kasutajale eraldi ütlema, kus kohast millist programmi otsida. Ühe värati kaudu saab teenust pakkuda (ehk partnerit oodata) üldjuhul vaid üks programm. Üks programm aga võib vajadusel kasutada ka mitut väratit. Näiteks FTP ühenduses kasutatakse kahte väratit: ühte teadete ja käskluste ning teist andmefailide edastamiseks.

Kellaaja küsimine eemal asuvast arvutist - näide

Järgnev programm küsib masina madli.ut.ee kellaaega.

```
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());
  }
}
```

ning töö tulemusena vastati `Tue Sep 21 17:38:45 1999` , mis oli parasjagu täiesti õige aeg.

Sideliini mängib klass nimega Socket ehk maakeeli pistik. `Socket sc` loob muutuja nimega `sc` tüübist `Socket` ning

```
new Socket("madli.ut.ee", 13);
```

loob uue pistikliidese kohalikus masinas töötava programmi ning masina `madli.ut.ee` vahel. Ühendus luuakse Tartu Ülikooli masinas väratis nr. 13 ootava teenust pakkuva programmiga. Pistikliidesest lugemiseks luuakse isend nimega `sisse` tüübist `BufferedReader`. Lause `sc.getInputStream()` annab pistiku `sc` voo, kust saab lugeda. `InputStreamReader` muudab voost saabuvad baidid tähtedeks ning `BufferedReader`i abil hakkatakse ridu lugema. Eemal masinas ootav programm on loodud nii, et kui keegi temaga ühendust võtab, siis pärast ühenduse moodustamist saadab ta ühendust võtnud programmile oma kellaaega teatava lause ning lõpetab ühenduse. Käsu `sisse.readLine()` tulemusena loetakse sisendvoost üks rida ning see satub System.out.println meetodi parameetriks, kust kaudu see ekraanile trükitakse.

Voog nii kirjutamiseks kui lugemiseks

Kui soovida serverist kasutaja kohta teavet saada, siis aitab programm nimega finger. Kui ta on serverisse tööle pandud, siis talle kasutaja nime ütlemisel vastab ta näiteks, millal kasutaja viimati kirju lugenud on ning millal üldse masinasse loginud. Järgnevalt programmilõik, mis küsib serverilt lin2.tpu.ee infot jaagupi-nimelise kasutaja kohta.

```
import java.net.*;
import java.io.*;
public class Fingerinfo{
```

```
  public static void main(String argumendid[]) throws Exception{
    Socket sc=new Socket("lin2.tpu.ee", 79);
    BufferedReader sisse=new BufferedReader(
      new InputStreamReader(sc.getInputStream())
    );
    PrintWriter valja=new PrintWriter(sc.getOutputStream(), true);
    valja.println("jaagup");
    String rida=sisse.readLine();
    while(rida!=null){
     System.out.println(rida);
     rida=sisse.readLine();
    }
  }
```

Ning tulemus nägi ekraanil välja järgmine:

```
C:\jaagup\vork>java Fingerinfo
Login: jaagup                           Name: Jaagup Kippar
Directory: /home2/jaagup                Shell: /bin/bash
Office: L51
On since Tue Sep 21 19:06 (EEST) on ttypa from math6
   4 minutes 3 seconds idle
Mail last read Tue Sep 21 19:08 1999 (EEST)
Plan:
Loodusteaduslike ainete eriala tudeng.
```

Muu programmiosa on kellaaja küsimisega sarnane. Andmete saatmiseks kasutati klassi `PrintWriter`, kes saadab andmed pistiku väljundvoo kaudu teisele osapoolele. Sõna `true` konstruktori teise parameetrina tähendab, et teade saadetakse kohe välja. Võrdlus `!=null` kontrollib, et seal rida ikka olemas oli ning järgnev käsk trükib selle rea ekraanile. Kui voog on suletud, siis pole sealt enam midagi tulemas ning meie programm läheb sellest while tsüklist edasi ning lõpetab oma töö.

Telnet-ühendus

Nõnda väratisse kirjutada ning lugeda saab ka programmi Telnet abil. Teda saab kasutada nii serverisse (nt. lin2.tpu.ee) sisse loginuna (samuti Telneti abil) kui ka kohalikust masinast otse mingi masina mingisse väratisse ühendust võttes. Kella küsimine näeb telneti abil välja järgmine:

```
[jaagup@minitorn jaagup]$ telnet madli.ut.ee 13
Trying 193.40.5.124...
Connected to madli.ut.ee.
Escape character is '^]'.
Wed Oct 11 19:16:07 2000
Connection closed by foreign host.
[jaagup@minitorn jaagup]$
```

Esimesel real on ees operatsioonisüsteemi viip ning selle järele kirjutati `telnet madli.ut.ee 13`. Järgnev rida teatab, et programm püüab ühendust saada masinaga 130.40.5.124. Tegemist on sama Tartu masinaga. Igal masinal on lisaks nimele veel number. Nimega saab lihtsalt elu mugavamaks teha, numbreid on keerulisem pähe õppida. Masinatel aga omavahel numbritest kergem aru saada. Connected rida teatab, et ühendus on saadud ning veel järgmine, et ühenduse katkestamiseks tuleb üheaegselt vajutada CTRL ning]. Seejärel tuleb sealtpoolselt programmilt saabunud vastus. Connection closed on jällegi telneti programmi poolne teade ühenduse lõppemise kohta. Ka fingerit saab telneti abil kasutada, siis tuleb vaid väratiks valida 79 ning pärast ühenduse loomist tippida sisse inimese nimi, kelle kohta infot soovitakse saada. Nii saab telneti-nimelist programmi kasutada oma elu mugavamaks muutmisel ja programmide töö kontrollimisel, kuid soovides lasta omatehtud programmidel

ühendust pidada ja mujalt andmeid saada, peame paratamatult ühenduse loomise ning teadete vahetamise ise programmi sisse kirjutama.

Omatehtud serveriprogramm

Ka ise saame luua väratit kuulavaid programme, mis sinna uudistama tulnuga suhtlevad. Järgnev programm kirjutab kõigile, kes ennast selle masina, kus programm töötab, väratisse nr. 3001 ühendavad, et arvuti on korras.

```
import java.io.*;
import java.net.*;
public class Teade1{
  public static void main(String argumendid[]) throws IOException{
    ServerSocket ss=new ServerSocket(3001);
    while(true){
      Socket sc=ss.accept();
      PrintWriter valja=new PrintWriter(sc.getOutputStream(), true);
      valja.println("Arvuti on korras");
      sc.close();
    }
  }
}
```

Ning nagu teisest masinast proovides näha, programm töötab.

```
[jaagup@minitorn tarkvara]$ telnet math6.tpu.ee 3001
Trying 193.40.238.56...
Connected to math6.tpu.ee.
Escape character is '^]'.
Arvuti on korras
Connection closed by foreign host.
```

Proovida saab ka tegelikult samast masinast, kui masina nimeks panna `localhost` ning sellisel juhul ei pea arvuti isegi mitte võrgus olema.

Kui ühenduse pidamiseks oli klass `Socket` (pistik), mille sisend- ning väljundvoo abil sai teateid lugeda ning saata, siis ühenduse loomiseks peab ühes otsas olema klass `ServerSocket`. Konstruktoris antakse ette, millist väratit ta kuulab ning siis, kui programmis saadetakse talle teade `accept()`, jätab ta programmi täitmise seniks seisma, kuni väljapoolt keegi selle väratiga ühendust tahab võtta (n.ö. kuulab kuni keegi uksele koputab). Siis väljastab `ServerSocket` pistiku, mille abil saab koputanuga ühendust pidada. Edaspidine pistikust lugemine ning sinna kirjutamine on sarnane kui eelmiste programmidegi puhul.

Nüüd näide programmist, mis vastab sõltuvalt kasutaja nimele:

```
import java.io.*;
import java.net.*;
public class Teade2{
  public static void main(String argumendid[]) throws IOException{
    ServerSocket ss=new ServerSocket(3001);
    while(true){
      Socket sc=ss.accept();
      PrintWriter valja=new PrintWriter(sc.getOutputStream(), true);
      BufferedReader sisse=new BufferedReader(
        new InputStreamReader(sc.getInputStream())
      );
      valja.println("Mis su nimi on?");
      String nimi=sisse.readLine();
      if(nimi.equals("Mari")){
        valja.println("Tule homme minu juurde!");
      } else{
        valja.println("Mind pole homme kodus.");
      }
      sc.close();
    }
  }
```

```
}
```

Selle programmi juures aga tekib probleem: ajal, kui keegi kasutaja oma nime sisse tipib, on programm selle koha peal paigal ning samaaegselt ei saa teise kasutajaga suhelda. Kui aga juhtuks sel ajal Mari ühendust võtma, siis peaks ju ka tema ootama ning sellest oleks ju ometi kahju. Analoogiliselt: kui säärase põhimõttega töötaks ka pangaautomaatide programm, siis saaks korraga vaid üks kasutaja oma rahaga opereerida ning teised peaksid tema järele ootama. Olgugi, et kasutaja on võibolla Tartus ning ootaja Raplas, kuid kui nad kasutavad võrgu kaudu sama programmi teenuseid, peab programm suutma nende mõlemaga tegelda.

Eraldi lõim

Järgnev programm peaks hakkama saama ka juhul, kui mõni kasutaja end külge ühendab ning siis paigale unustab.

```
import java.io.*;
import java.net.*;
public class Teade3{
  public static void main(String argumendid[]) throws IOException{
    ServerSocket ss=new ServerSocket(3001);
    while(true){
      new Teate3loim(ss.accept());
    }
  }
}

class Teate3loim extends Thread{
   Socket sc;
   public Teate3loim(Socket uus_sc){
     sc=uus_sc;
     start();
   }
   public void run(){
     try{
       PrintWriter valja=new PrintWriter(sc.getOutputStream(), true);
       BufferedReader sisse=new BufferedReader(
         new InputStreamReader(sc.getInputStream())
       );
       valja.println("Mis su nimi on?");
       String nimi=sisse.readLine();
       if(nimi.equals("Mari")){
         valja.println("Tule homme minu juurde!");
       } else{
         valja.println("Mind pole homme kodus.");
       }
       sc.close();
     }catch(Exception e){
       System.out.println("Probleem: "+e);
     }
   }
}
```

Kui üldjuhul käivitatakse programmi käske ükshaaval ning järgmist käsku alustades võib kindel olla, et eelmisega seotud toimingud on lõpetatud, siis mitme kasutaja üheaegseks teenindamiseks tuleb sellest rahulolust loobuda. Kuigi iga kasutaja puhul täidetakse tema jaoks tarvilikke toiminguid ükshaaval, tegeldakse samal ajal ka teise kasutajaga. Lihtsamate programmide puhul võib eeldada ja loota, et samaaegsed toimingud üksteist ei sega, kuid veidi hiljem tuleb hakata neid omadusi arvesse võtma.

Siin luuakse iga kasutaja jaoks omaette lõim(`Thread`) ehk iseseisvalt töötav käsujada. Selle abil saab programm korraga nagu "mitmes kohas" olla. Peaprogrammi (`Teade3`) osaks jääb vaid lasta `ServerSocket`'il väratit kuulata ning kui keegi

ühendust võtab, siis luua uus lõime (`Teate3Loim`) eksemplar ehk isend temaga suhtlemiseks.

Lõimeklassi meetodisse `run()` kirjutatud koodi saab käima panna sõltumatult programmi muudest tegemistest. Selleks tuleb lõimeklassi isendil käivitada meetod `start()` ning siis juba Java virtuaalmasin ise hoolitseb selle eest, et loodud uus lõim saaks oma `run()` meetodisse kirjutatud koodi iseseisvalt täita. Lõimeklassi isend luuakse peaprogrammis `new Teate3loim(ss.accept());` Selle käsu peale eraldatakse uuele isendile mälu ning käivitatakse konstruktor. Viimases jäetakse lõime isendisse meelde accept-käsuga loodud ühendus ning edasi kutsutakse välja lõimeklassi sisene meetod `start()`, et Java virtuaalmasin teaks hakata selle isendi sees olevat `run` meetodit käivitama.

Muutuja `sc` on kirjeldatud väljapool meetodeid selle pärast, et ta kehtiks kogu isendi ulatuses, s.t. nii konstruktoris kui meetodis `run`. Konstruktoris hakkab see osutama peaprogrammilt antud pistikule ning run-meetodis loetakse ning kirjutatakse sc kaudu viidatud ühenduse abil.

`try-catch` püünis run meetodi sees töötleb tekkivaid eriolukordi, praegusel juhul lihtsalt trükib välja sõna "probleem" ning seejärel kirjelduse, mis erindiga kaasa anti. Erind tekib näiteks juhul, kui ühendus ära kaob. Kinni püütakse erindid sellepärast, et nad programmi tööd ei hakkaks segama. Kuna run meetod kaetakse algse Thread klassiga võrreldes üle ning algses meetodis pole lubatud erindeid välja heita, siis ei tohi ka üle kaetud meetodisse erindit välja lubavat throws-klauslit kirjutada. Vähemasti seetõttu tulebki püünist kasutada.

Lihtne jututuba

Järgnev programm on väike Telneti-jututuba:

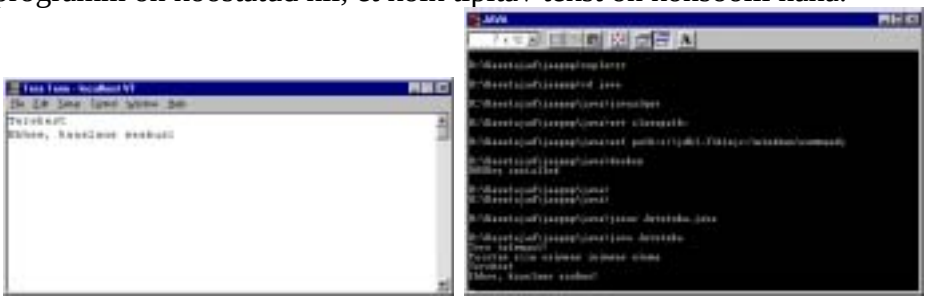
```
import java.io.*;
import java.net.*;
import java.util.Vector;
public class Jututuba{
  public static void main(String argumendid[]) throws IOException{
    ServerSocket ss=new ServerSocket(3001);
    Vector uhendused=new Vector();
    while(true){
      Socket sc=ss.accept();
      uhendused.add(sc);
      new JututoaLoim(sc, uhendused);
    }
  }
}

class JututoaLoim extends Thread{
  Vector v;
  Socket sc;
  public JututoaLoim(Socket uus_sc, Vector uus_v){
    v=uus_v;
    sc=uus_sc;
    start();
  }
  public void run(){
    try{
      BufferedReader sisse=new BufferedReader(
        new InputStreamReader(sc.getInputStream())
      );
      boolean veel=true;
      while(veel){
         String rida=sisse.readLine();
         System.out.println(rida);
         if(rida.startsWith(".ots"))veel=false;
         for(int i=0; i<v.size(); i++){
           Socket skt=(Socket)v.elementAt(i);
           PrintWriter valja=new PrintWriter(skt.getOutputStream(), true);
           valja.println(rida);
```

```
          }
        }
        sc.close();
      } catch(Exception e){
        System.out.println("Probleem: "+e);
      }
      v.remove(sc);
    }
  }
```

Jututoa töö nägemiseks tuleb kõigepealt serveripoolne programm tööle panna ning seejärel võib klient end külge ühendada. Windowsi Telneti puhul saab Connect menüüst valida Remote System ning sealt sättida, millise nimega masinasse ning väratisse ühenduda soovitakse. Sama masina nimi on localhost, väratis 3001 ootab me loodud serverprogramm. Kui Terminal menüü alt lülitada Local Echo, siis on ka tippimise ajal näha, mida kirjutatakse. Muul juhul tulevad nähtavale vaid serveri poolt saadetud teated.

Järgmisena külge ühendunud klient hakkab teksti nägema alates hetkest, kui ühendus on loodud. Siin paistab teise kliendi esimene saadetud teade olema "Terekest". Esimese kliendi aknas paistab see erinevalt muust tekstist olema ühekordselt. Serverprogramm on koostatud nii, et kõik tipitav tekst on konsoolil näha.



Võrreldes eelmise programmiga on juurde tulnud Vector ühenduste andmete hoidmiseks. Programmeerija jaoks käitub see samamoodi kui eelpool kataloogi nimede meelespidamiseks kasutatud LinkedList. Tegemist on muutuva suurusega massiiviga, kuhu saame elemente lisada ning eemaldada. Vector hoiab andmeid teadmises, et need on kõige ülema klassi ehk `Object` isendid. Massiivi lisades muudab ta kõik muud automaatselt `Object`iks, välja võttes peame ütlema, milliseks tüübiks me teda muundada soovime. Juhul kui muundamine on võimalik, toimib kõik ilusti. Kui kusagilt kasutaja juurest tuleb uus rida teksti, saadetakse see tekst kõikidele kasutajatele, kelle pistik on massiivis. Kui kasutaja kirjutab .ots, siis pääseb ta tsüklist välja, ühendus katkestatakse ning tema pistik eemaldatakse massiivist.

Turvalisus

Ka rakendid ehk appletid ehk rakendikäituri sees töötavad programmikesed saavad Interneti kaudu andmeid liigutada. Et aga neid võiks julgelt käivitada, selleks on neile lisaks muudele piirangutele (nad ei pääse ligi failidele ega arvuti seadmetele) ka võrgu kasutamise juures piirang: nad saavad ühendust pidada vaid selle masina väratitega, kust rakendi enese programm rakendikäiturisse tõmmatud on. Kui on vaja muude masinatega suhelda, siis tuleb rakendi klasse hoidvasse masinasse tööle panna programm, mis siis juba vajaliku masinaga ühendust peab. Kui taolist turvapiirangut

poleks, siis saaks programmeerija kirjutada rakendi, mis kasutajale teadmata näiteks võõrasse masinasse sisse murrab. Samas võõrast masinast sissetungijat otsima hakates jõutaks rakendi kasutajani, kes aga milleski süüdi pole. Kuna brauserid on arvutites levinud, siis on kerge programme kasutada rakenditena. Siis võib kasutaja asuda ükskõik millise Internetti ühendatud arvuti taha ja programmi kasutada. Kui ka andmed salvestatakse serveris, siis saab tööd rahumeeli jätkata samast kohast, kus eelmisel korral pooleli jäi ning andmed tulevad võrku pidi kohale.

Teinud eraldiasuvas arvutis programmi valmis ning soovides loodud serverprogrammi avalikku serverisse tööle panna, ei pruugi tolle serveri administraator nimetatud soovi kuigi rõõmsalt vastu võtta. Isegi siis, kui serverprogramm pole kirjutatud pahatahtlikult ning on küllalt tähelepanelikult vigade suhtes üle vaadatud, võib seal küllalt kergesti leiduda apsakaid, mis võivad Interneti pealt tulevatel huvilistel serveri tööd tublisti häirida või mõnel juhul suisa suuta serverarvuti administreerimine üle võtta.

Ülesandeid

Meldimine

- Väratit kuulav programm teatab ühendusevõtjale tema järjekorranumbri
- Programm küsib ühendusevõtjalt nime. Juhul, kui sellist nime veel kirjas ei ole, siis lisatakse see nimi nimede faili.
- Programm küsib ühendusevõtjalt kasutajanime ning parooli. Kui sellise kasutajanime ning parooliga kasutaja leidub, siis väljastatakse serverarvuti kellaaeg. Olemasoleva kasutajanime kuid vale parooli puhul antakse veateade. Puuduva kasutajanime puhul lisatakse kasutaja soovi korral koos sisestatava parooliga andmebaasi. Järgmisel korral selle nime ja parooliga sisse meldides väljastatakse talle kellaaeg.

Jututoa graafiline klient.

Töö serveripooleks eelpool vaadeldud programm, mis kasutajatelt saabuvad teated kõigile edasi saadab.

- Loo tekstiväljaga aken sinna teadete saatmiseks.
- Loo tekstialaga aken jututoast tulevate teadete lugemiseks
- Ühenda need programmid kokku, pannes teadete lugemise omaette lõime.
- Jututoas on näha kasutaja nimi.