Graafilise liidesega võrgurakendused

Võrguprotokoll, lõimed, klient, server, kuularid, dokumenteerimine, vektorgraafika

Trips-traps-trull

Kirjeldus

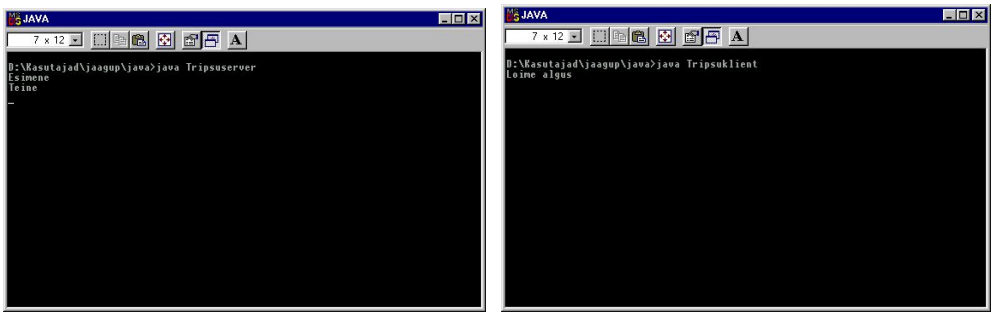
Võrguprogrammi näiteks on siin kokku pandud Trips-traps-trulli mäng. Serveripoole ülesanne on kaks mängijat kohale oodata, siis kordamööda teineteisele käiguõigus anda ning saadetud käsud edasi saata. Samuti on serveripoole otsustada, kumba mängijat tähistatakse nulli ja kumba ristiga. Otsus on tehtud lihtne: esimene saabunud mängija saab märgiks X, teine 0. Server ise on koostatud nii, et kui üks paar on omaette lõime abil mängima saadetud, siis asub serveri peaprogramm ise järgmist kasutajat ootama ning paari kokku saamisel paneb need taas omavahel mängima. Edasine klientide vaheline vestlus on juba nende otsustada. Server seda otseselt ei määra ega kontrolli. Vaid kirjutamise järg antakse mängijatele kordamööda. Kui keegi mängijatest saadab lõputunnuse, siis lõpetatakse neid mängijaid teenindava lõime töö.

Kliendi pool ühendab end serveriga ning asub sealt tulevaid teateid ootama. Kui öeldakse, millise sümboliga klient mängib, siis jäetakse see meelde. Kui teatatakse toimunud käigust, siis joonistatakse selle tulemus nuppudest moodustatud mängulauale. Kui oli tegemist mängu alguse teatega või vastase käiguga, siis muudetakse lubatud käigunupud aktiivseks, et kasutaja saaks sobiva valida. Vajutuse peale muudetakse nupud taas külmunuks, saadetakse nupule vastav käik serverisse ning edasi tegutsetakse vastavalt saabuvatele teadetele. Lõpetusnupule vajutades saadetakse lõpetusteade, mille peale server teab selle edasi saata mängupartnerile ning samas ka nende mängijatega suhtleva lõime sulgeda. Saadetavad käsud näevad välja järgnevad:

Kuju	Tähendus
.margiksX	Teate saanud klient teab edaspidi, et tema mängib ristidega. Kui olnuks .margiks0, tuleks nullidega mängida
.kaikX5	Mängija X käis nupu 5. .kaik03 tähendanuks, et mängija 0 vajutas nuppu 3.
.kaikVaba	Teate saanud klient võib oma käiguga alustada.
.ots	Mäng on lõppenud

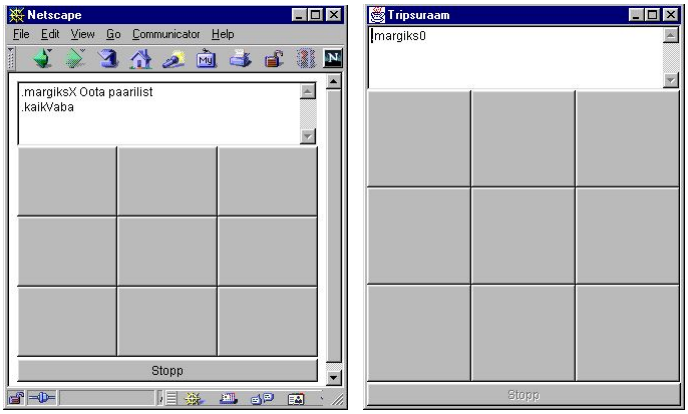
Tutvustavad pildid

Järgnevalt on mõned pildid ühest konkreetsest mängust. Masinas on käivitatud serverprogramm, üks klient käsurealt ning teine veebiseilurist. Serverisse on jõudnud mõlemad kliendid ühenduda (sellest teatavad serveri ekraanil sõnad Esimene ja Teine). Klient on serverist teateid kuulava lõime tööle saanud.

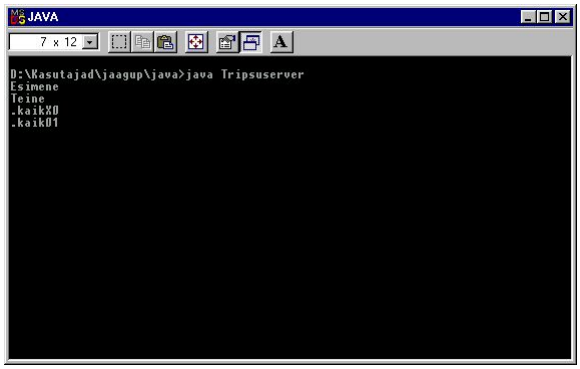


Seiluriaknas olev klient jõudis ühenduda esimesena. Temale tuli kõigepealt teade .margiksX Oota paarilist, mille peale inimene teab, et tal tuleb partnerit oodata. Partneri saabumisel antakse sellele teada .margiks0 ning seejärel esimesele ühendunule .kaikVaba. Selle peale vabastatakse esimese kliendi nupud lukust ning tal on vaba voli käia. Mängijal on vaba voli valida üheksa

mängunupu ning Stopp-nupu vahel.

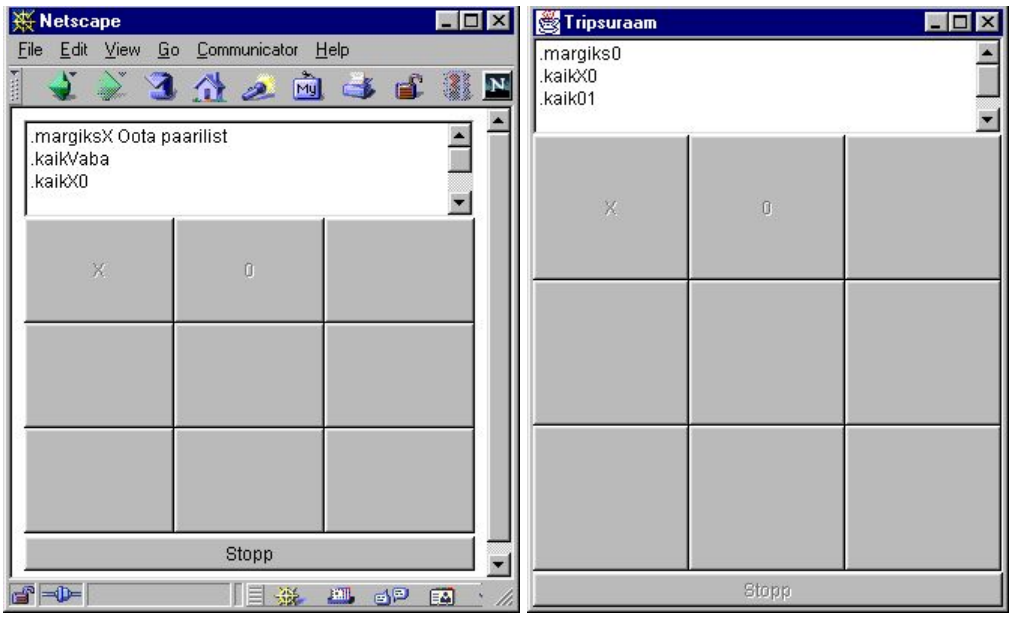


Nagu serveri konsoolilt näha, käis X-iga mängija kõigepealt nupule nr. 0 ning seejärel 0-ga mängija nupule 1.




```
D:\Kasutajad\jaagup\java>java Tripsuserver
Esimene
Teine
.kaikX0
.kaik01
```

Sama tulemus paistab ka mängulaudu vaadates. Saabuvad teated on lihtsalt kontrolli mõttes ülal asuvasse tekstivälja paigutatud. Nupud on aga samuti õigesti märgitud.



Nii võib mäng jätkuda kuniks mängijad seda soovivad. Praeguse näite puhul veel võite ei loeta ning tulemusi ei kontrollita. Kes leiab, et ta enam midagi arukat käia ei taha ega oska, võib vajutada stopp. Selle peale katkestatakse serveris selle mängu lõim ning ka kaaslasele saadetakse teade lõpetamisest. Nupule ilmub kiri Start ning kui sellele vajutada, algab mängijate kohtumine otsast peale.

Serveripoole lähtekood seletustega

```
import java.io.*;
import java.net.*;
```

Võrguprogrammide puhul alati vajalikud paketid

```
public class Tripsuserver{
```

Programmi töö käigus vajalikud konstandid. Kui nende väärtused ühte kohta kirjutada, siis on vajadusel nende väärtusi kergesti võimalik muuta. Näiteks kui juhtub vastava värati peal juba mõni programm jooksma, siis tuleb siia mõni vaba number määrata. Kui tegemist on käsurealt käivitatava kliendiga ning server jookseb kohalikust masinast väljaspool, siis tuleb vastava serveri nimi siia kirjutada, et klient teaks sinna ühenduda. Rakendi puhul pole siia märgitud masina nimi tähtis, sest rakend saab alati ühenduda vaid serverisse kust ta ise pärit ning selle aadress küsitakse töö käigus.

```
  static final int pordinr=3001;
  static final String masin="localhost";
  static final String lopp=".ots";
```

Serveri main-meetod ei tegele lihtsuse mõttes eriolukordade töötlemisega vaid laseb veateated lihtsalt konsoolile trükkida. Selleks on kiri `throws Exception` meetodi päises.

```
  public static void main(String argumendid[]) throws Exception{
```

Asutakse väratit kuulama

```
    ServerSocket ss=new ServerSocket(Tripsuserver.pordinr);
```

Jätkatakse igaveses tsüklis, st. senikaua kuni serverirakendus CTRL+C-ga kinni pannakse. Viisakam, kuid keerulisem võimalus oleks luua eraldi protokoll serveri juhtimiseks ning selle abil

saata serverile teateid sulgemise või muude toimingute kohta.

```
    while(true){
```

Oodatakse esimese kliendi saabumist ning teatatakse talle, et tema märgiks on X. Serveri administraatorile antakse teada, et paarist esimene klient on saabunud.

```
      Socket sc1=ss.accept();
       new PrintWriter(sc1.getOutputStream(), true).println(".margiksX "+
                                                  "Oota paarilist");
       System.out.println("Esimene");
```

Sama lugu teise kliendiga.

```
      Socket sc2=ss.accept();
      new PrintWriter(sc2.getOutputStream(), true).println(".margiks0");
      System.out.println("Teine");
```

Luuakse `Tripsuloim`'e nimelisest klassist uus eksemplar ning antakse sellele kaasa juurdepääsuvõimalus mõlema kliendi ühendusele, et loodud isendil oleks võimalus hakata nende omavahelise suhtluse üle hoolt kandma.

```
      new Tripsuloim(sc1, sc2);
    }
  }
```

Ning serveri peaklassil rohkem tööd polegi.

```
}
```

Tripsuloim'e ülesandeks on siis talle etteantud ühendustega suhtlema hakata ning hoolitseda, et nad omavahel saaksid mängu ilusti peetud.

```
class Tripsuloim extends Thread{
```

Klassi sisse luuakse koht ühenduste andmete hoidmiseks. Lühiduse mõttes on tehtud Socket tüüpi massiiv. Sellisel juhul pole vaja kahte eraldi muutujat ning ühised operatsioonid (näiteks sulgemine lõpus) on võimalik tsükli abil läbi viia nii, et pole tarvilik kummagi pistiku jaoks eraldi käske välja kirjutada. Eeldatakse, et esimese mängija ühendus paigutatakse elemendiks sc[0] ning teise oma sc [1].

```
  Socket[] sc=new Socket[2];
```

Konstruktor saab kahe pistiku andmed, talletab need kohalikku pistikühenduste massiivi ning `start`-meetodi käivitamise abil palub virtuaalmasinal käivitada `run`-nimeline meetod eraldi lõimena, mis peaks hoolt kahe saabunud ühenduse vahelise suhtlemise eest. Kuni siiani ootab veel klassis `Tripsuserver` olev peaprogramm iga täidetava käsu taga enne kui oma järjega edasi läheb. Kui aga konstruktor on läbitud, siis võib peaprogramm käsu `new Tripsuloim(sc1, sc2);` edukalt lõppenuks lugeda ning järgmise juurde asuda. Järgmiseks tuleb aga peaprogrammis hüpe tsükli algusse ning siis asutakse juba uut klienti ootama. Loodud `Tripsuloim`'e isend aga toimetab tasapisi `run`-meetodis edasi.

```
  public Tripsuloim(Socket usc1, Socket usc2){
    sc[0]=usc1;
    sc[1]=usc2;
    start();
  }
```

Omaette lõimes töötav mängijapaarivahelist suhtlust korraldav meetod.

```
public void run(){
```

Veapüünis, kuna üle kaetud run-meetodist pole võimalik erindeid throws käsuga välja lasta, samas aga mitmed võrguga seotud käsud nõuavad erindite töötlemist. Samuti on viisakas tekkivatele probleemidele reageerida.

```
    try{
```

Nii sisend- kui väljundvoogude tarbeks luuakse massiivid, et pärastpoole oleks nende voogude poole mugavam pöörduda.

```
      BufferedReader sisse[]=new BufferedReader[2];
      PrintWriter valja[]=new PrintWriter[2];
      for(int i=0; i<2; i++){
        sisse[i]=new BufferedReader(
          new InputStreamReader(sc[i].getInputStream())
        );
        valja[i]=new PrintWriter(sc[i].getOutputStream(), true);
      }
```

Esimesele kliendile teatatakse, et too võib oma käiguga alustada.

```
      valja[0].println(".kaikVaba");
```

Muutuja veel näitab, kas vastav lõim peab oma tööd jätkama. Kui klient saadab lõpetamisteate, siis muutuja väärtus läheb vääraks ning enam uut ringi teateid kuulama ei minda.

```
      boolean veel=true;
      while(veel){
```

Esimeselt kliendilt saabunud käik või lõpetamisteade saadetakse mõlemale edasi. Klient on koostatud nii, et korrektseks toimimiseks peab ta ka ise serverilt oma saadetud teated kätte saama. See on justkui kontroll, et ühendus serveriga töötab.

```
        String vastus=sisse[0].readLine();
        kirjuta(valja, vastus);
```

Kui esimeselt kliendilt saadi lõpetamisteade, siis teise kliendi teadet enam ei oodata.

```
        if(!vastus.equals(Tripsuserver.lopp)){
```

Muul juhul kirjutatakse ka teiselt saabunud teade mõlemale.

```
          vastus=sisse[1].readLine();
          kirjuta(valja, vastus);
        }
        if(vastus.equals(Tripsuserver.lopp))veel=false;
      }
```

Jõudnud tsüklist välja, suletakse ühendused mõlema kliendiga.

```
      for(int i=0; i<2; i++){
        sc[i].close();
      }
    }catch(Exception e){
      System.out.println("Probleem:"+e);
    }
  }
```

Abimeetod teate välja saatmiseks. Teade jõuab nii mõlemale kliendile kui serveri konsoolile. Piiritleja private meetodi sees teatab, et seda võib kasutada ainult sama klassi (`Tripsuloim`) isendi seest.

```
  private void kirjuta(PrintWriter[] pw, String teade){
```

```
    for(int i=0; i<pw.length; i++){
      pw[i].println(teade);
    }
    System.out.println(teade);
  }
}
```

Kliendipoole lähtekood seletustega

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
```

Appleti (rakendi) alamlass, et oleks võimalik seda veebilehel tööle panna.

```
public class Tripsuklient extends Applet{
```

Serveri nimi küsitakse klassi Tripsuserver staatilisest muutujast. Andmeid käiakse sellepärast teisest klassist uurimas, et nii on võimalik väärtused ühte kohta kokku kirjutada ning konfigureerimine on lihtsam. Samas aga peab kliendi käivitamiseks sellisel juhul ka serveriklass kaasas olema.

```
  static String serverinimi=Tripsuserver.masin;
  public Tripsuklient(){
    try{serverinimi=getCodeBase().getHost();}catch(Exception e){}
      //rakendi puhul võetakse serveri nimi brauserilt
```

Rakendi vaikimisi paigutushalduriks on FlowLayout (elemendid vabas suuruses üksteise järel). BorderLayouti abil keskele paigutatuna on võimalik sisemine paneel üle kogu pinna välja venitada.

```
    setLayout(new BorderLayout());
    add(new Tripsupaneel(), BorderLayout.CENTER);
  }

  public static void main(String argumendid[]){
```

Käsurealt käivitades võimaldatakse serveri nimi käsurea parameetrina anda. Kui aga pole parameetreid antud, sel juhul jäetake vaikimisi nimi. Seilurilt klienti vaadates main meetod ei käivitu.

```
    if(argumendid.length>0)serverinimi=argumendid[0];
    Frame f=new Frame("Tripsuraam");
    f.add(new Tripsupaneel(), BorderLayout.CENTER);
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```

Kujundus on Tripsupaneeli nimelisse klassi kokku pandud. Klass nimega Tripsuklient on vaid käivitamiseks.

```
class Tripsupaneel extends Panel{
```

Kõik lehel nähtavad nupud on paigutatud ühisesse massiivi. Sellisena on neid kergem tsükli abil külmutada, sulatada või puhastada.

```
  Button nupp[]=new Button[10];
  String omanimi;  //X või 0, mille alt klient mängib
  TextArea suhtlus=new TextArea("",3, 60,TextArea.SCROLLBARS_VERTICAL_ONLY);
  public Tripsupaneel(){
    setLayout(new BorderLayout());
    add(suhtlus, BorderLayout.NORTH);
```

Nupud omaette paneeli, kus 3*3 elementi.

```
    Panel nupupaneel=new Panel();
```

```
    nupupaneel.setLayout(new GridLayout(3, 3));
```

Tsükli abil luuakse üheksa mängunuppu, igaüks neist lisatakse nii paneeli näitamiseks kui massiivi pärastiseks kergemaks ligipääsuks.

```
    for(int i=0; i<9; i++){
      nupp[i]=new Button("  ");
      nupupaneel.add(nupp[i]);
    }
    add(nupupaneel, BorderLayout.CENTER);
```

Massiivi viimane nupp mängu peatamiseks. Kui nupupaneel paigutati `Tripsupaneel`i keskele (kogu vabale alale), siis seiskamis/käivitusnupp pannakse `Tripsupaneel`i allserva, nupupaneeli alla.

```
    nupp[9]=new Button("Stopp");
    add(nupp[9], BorderLayout.SOUTH);
```

Kõik nupud külmutatakse.

```
    seaNupud(false);
```

Käivitatakse serveri teadete kuulamiseks ning nende töötlemiseks omaette lõim. Lõimele antakse ette osuti loodud Tripsupaneelile, mille kaudu on omakorda võimalik ligi pääseda seal paiknevatele nuppudele ning tekstialale.

```
    TripsukliendiLoim vahendaja=new TripsukliendiLoim(this);
  }
```

Alamprogramm kliendi nuppude külmutamiseks ning lahti sulatamiseks. Külmutamise korral külmutatakse kõik nupud. See toimub juhul, kui käigujärg läheb vastasele ning sel ajal pole server võimeline ühtki teadet vastu võtma. Lahti sulatamisel aga tehakse toimivaks vaid nupud, millel pole mingit kirja ehk mängunuppude puhul need, mis on veel vabad.

```
  void seaNupud(boolean kasutatav){
    if(!kasutatav)for(int i=0; i<nupp.length; i++)nupp[i].setEnabled(kasutatav);
    else{
      for(int i=0; i<9; i++)if(nupp[i].getLabel().trim().equals(""))
         nupp[i].setEnabled(kasutatav);
    }
```

Seiskamis/käivitusnupp muudetakse igal juhul vastavalt etteantud parameetrile, st. tehakse kasutatavaks ka siis kui sinna on midagi kirjutatud.

```
    nupp[9].setEnabled(kasutatav);
  }
}
```

Lõim, mille ülesandeks on sidepidamine kliendi ja serveri vahel: serverist tulevate teadete järgi nupu pealkirjade muutmine, samuti kasutajapoolsetest nupuvajutustest käikude koostamine ning serveri poole saatmine.

```
class TripsukliendiLoim implements ActionListener, Runnable{
  Tripsupaneel tp;
  PrintWriter valja;
  BufferedReader sisse;
  boolean veel=true;
  Socket sc;

  public TripsukliendiLoim(Tripsupaneel utp){
```

Jäetakse meelde osuti etteantud Tripsupaneelile

```
    tp=utp;
```

Kõik Tripsupaneeli nupud pannakse siinsele lõimeklassi isendile teateid saatma. St, et ükskõik millist nuppu paneelil vajutatakse, ikka käivitub siinse isendi actionPerformed.

```
    for(int i=0; i<10; i++)tp.nupp[i].addActionListener(this);
    alusta();
  }
```

Uue mängu alustamisel sooritatavad toimingud. Väljastatav tõeväärtus teatab, kas töö õnnestus.

```
  public boolean alusta(){
    boolean korras=true;
    try{
```

Mängunupud pealkirjatuks

```
     for(int i=0; i<9; i++)tp.nupp[i].setLabel(" ");
```

Käivitusnupule kiri Stopp

```
     tp.nupp[9].setLabel("Stopp");
```

Teatekastina kasutatav tekstiväli tühjaks

```
     tp.suhtlus.setText("");
```

Uus ühendus serverisse, sellest omakorda küsitakse sisend- ja väljundvoog.

```
     sc=new Socket(Tripsuklient.serverinimi, Tripsuserver.pordinr);
     sisse=new BufferedReader(
        new InputStreamReader(sc.getInputStream())
     );
     valja=new PrintWriter(sc.getOutputStream(), true);
     veel=true;
```

Luuakse uus lõim, mis run-meetodi (taas) käima paneb. Kui tähele panna, siis ülal klassi kirjelduses polnud mitte kirjas extends `Thread`, vaid oli implements `Runnable`. Seetõttu tuleb omaette `Thread` klassi isend luua ning sellele anda siinne isend käivitamiseks (ei saa lihtsalt siin samale isendile `start` ütelda). Samas on nii võimalik kergesti iga uue mängu algul `run` taas uue lõimena käima panna.

```
     new Thread(this).start();
    }catch(IOException e){
```

Veateade väljastatakse nii konsoolile kui tekstialasse.

```
      e.printStackTrace();
      tp.suhtlus.setText("Ühendus serveriga "+Tripsuklient.serverinimi+" puudub.");
      korras=false;
    }
    return korras;
  }

  public void run(){
    System.out.println("Loime algus");
    try{
       while(veel){
```

Serverist saabuvad read saadetakse tootle –nimelisele meetodile töötlemiseks.

```
         tootle(sisse.readLine());
       }
    } catch (Exception e){
      System.out.println("Probleem: "+e);
      e.printStackTrace();
    }
```

```
    System.out.println("Loime ots");
  }
```

Iga saabuva teatega käitutakse vastavalt selle sees olevale sisule.

```
  private void tootle(String teade) throws IOException{
```

Igal juhul lisatakse teade tekstialasse kontrolliks

```
    tp.suhtlus.setText(tp.suhtlus.getText()+teade+"\n");
```

Kui saabub teade kliendi kasutatava märgi kohta, siis jäetakse see meelde. Et vastav muutuja asub siinsele lõimeklassile etteantud Tripsupaneeli isendis, tuleb sellele Tripsupaneelile kõigepealt tp-nimelise muutuja kaudu ligi minna ning siis sealtkaudu märk muutujale omanimi omistada.

```
    if(teade.startsWith(".margiks"))tp.omanimi=teade.substring(8, 9);
```

Käigu puhul eeldatakse, et täht number 5 näitab käija nime ning täht nr. 6 nuppu, millele vajutati. Kui pakutud number on vigane, siis jäetakse käsk täitmata.

```
    if(teade.startsWith(".kaik")){
      try{
        int nr=Integer.parseInt(teade.substring(6, 7));
        tp.nupp[nr].setLabel(
          " "+teade.substring(5, 6)+" "
        );
      } catch(NumberFormatException e){} //sobimatu ruut
```

Vastase käigu puhul tehakse laual olevad nupud taas tundlikuks. Kontrollitakse, et käija nimi ei ühtiks kliendi enese märgiga.

```
      if(!teade.substring(5, 6).equals(tp.omanimi))tp.seaNupud(true);
    }
```

Lõputeate saabumise puhul antakse muutujale veel väärtuseks false, et enam uusi teateid ei kuulataks. Alumine suur nupp tehakse vajutatavaks ning sinna kirjutatakse Start. Suletakse ühendus serveriga.

```
    if(teade.equals(Tripsuserver.lopp)){
      veel=false;
      tp.nupp[9].setLabel("Start");
      tp.nupp[9].setEnabled(true);
      sc.close();
    }
  }
```

Käivitatakse, kui kliendiprogrammis on vajutatut ükskõik millist nuppu.

```
  public void actionPerformed(ActionEvent e){
```

Testiks teatatakse kliendi konsoolile, et nupuvajutus on kinni püütud

```
    System.out.println("Vajutus");
```

Nupud külmutatakse, et sama hooga poleks võimalik rohkem vajutada.

```
    tp.seaNupud(false);
```

Tehakse kindlaks, millisele nupule vajutati. Selleks vaatatakse läbi kõik olemasolevad nupud ning millise osuti on sama väärtusega kui vajutuse allika oma, sellele järelikult vajutati.

```
    int nr=0;
    Button allikas=(Button)e.getSource();
```

```
    for(int i=0; i<10; i++)if(tp.nupp[i]==allikas)nr=i;
```

Kui tegemist oli ühega ruudustiku nuppudest, siis saadetakse serverisse teade, millist nuppu kasutaja vajutas.

```
    if(nr<9)
      valja.println(".kaik"+tp.omanimi+nr);
```

Muul juhul peab olema tegemist seiskamis/käivitusnupuga

```
    else if(nr==9){
```

Kui sellel oli kiri stopp, siis saadetakse serverisse selle mängu lõpetusteade.

```
      if(tp.nupp[9].getLabel().equals("Stopp"))
         valja.println(Tripsuserver.lopp);
```

Muul juhul alustatakse uut ühendust ja mängu.

```
      else {
        tp.nupp[9].setLabel("Stopp");
        alusta();
      }
    }
  }
}
```

Edasiarendusvajadused

Näiteprogramm on püütud koostada võimalikult lihtsalt. Seetõttu on mitmed tarvilikud kohad välja jäetud. Võrguprogrammide koostamisel on üheks nõudeks, et kunagi ei tohi usaldada kliendi poolt saadetavaid andmeid, sest avalikus võrgus töötava serveri külge võib ühineda ükskõik kes ning miski ei takista pahatahtlikul sisenejal temale sobivaid baite teele saata. Praegu aga on näiteks mängija nimi meeles kliendipoolses programmis ning kui kliendi simuleerija otsustaks saata enese asemel kellegi teise nime, siis teda ei takistataks ning tal õnnestuks vastase eest käia. Kuna aga server siiski kindlalt otsustab, et käia saab kordamööda, siis õnnestub vaid teise käike rohkem teha, ülemääraseid oma märke kuhugile paigutada ei õnnestu.

Võrguprogrammide puhul on kombeks toimunud tegevused faili üles märkida ehk logida. Siis on selle järgi võimalik leida seletusi nii programmi töö käigus tekkinud probleemidele kui tagantjärele reageerida mängijatevahelistele vaidlustele.

Ka kliendiakna sulgemiseks peaks lihtsast töö katkestamisest viisakam võimalus olema, kus saadetakse serverile lõpetusteade. Rakendi puhul peaks see käivituma veebilehelt lahkumisel, rakenduse puhul aga akna sulgemisristile vajutamisel.

Programmitekst

Järgnevalt programmitekst tervikuna ilma vahele piktud kommentaarideta.

```
import java.io.*;
import java.net.*;
public class Tripsuserver{
  static final int pordinr=3001;
  static final String masin="localhost";
  static final String lopp=".ots";
  public static void main(String argumendid[]) throws Exception{
    ServerSocket ss=new ServerSocket(Tripsuserver.pordinr);
    while(true){
      Socket sc1=ss.accept();
       new PrintWriter(sc1.getOutputStream(), true).println(".margiksX "+
                                                  "Oota paarilist");
       System.out.println("Esimene");
      Socket sc2=ss.accept();
       new PrintWriter(sc2.getOutputStream(), true).println(".margiks0");
```

```
       System.out.println("Teine");
      new Tripsuloim(sc1, sc2);

    }
  }
}

class Tripsuloim extends Thread{
  Socket[] sc=new Socket[2];
  public Tripsuloim(Socket usc1, Socket usc2){
    sc[0]=usc1;
    sc[1]=usc2;
    start();
  }
  public void run(){
    try{
       BufferedReader sisse[]=new BufferedReader[2];
       PrintWriter valja[]=new PrintWriter[2];
       for(int i=0; i<2; i++){
         sisse[i]=new BufferedReader(
           new InputStreamReader(sc[i].getInputStream())
         );
         valja[i]=new PrintWriter(sc[i].getOutputStream(), true);
       }
       valja[0].println(".kaikVaba");
       boolean veel=true;
       while(veel){
         String vastus=sisse[0].readLine();
         kirjuta(valja, vastus);
         if(!vastus.equals(Tripsuserver.lopp)){
            vastus=sisse[1].readLine();
            kirjuta(valja, vastus);
         }
         if(vastus.equals(Tripsuserver.lopp))veel=false;
       }
       for(int i=0; i<2; i++){
         sc[i].close();
       }
    }catch(Exception e){
      System.out.println("Probleem:"+e);
    }
  }
  private void kirjuta(PrintWriter[] pw, String teade){
    for(int i=0; i<pw.length; i++){
      pw[i].println(teade);
    }
    System.out.println(teade);
  }
}
```

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

public class Tripsuklient extends Applet{
  static String serverinimi=Tripsuserver.masin;
  public Tripsuklient(){
    try{serverinimi=getCodeBase().getHost();}catch(Exception e){}
      //rakendi puhul võetakse serveri nimi brauserilt
    setLayout(new BorderLayout());
    add(new Tripsupaneel(), BorderLayout.CENTER);
  }

  public static void main(String argumendid[]){
    if(argumendid.length>0)serverinimi=argumendid[0];
    Frame f=new Frame("Tripsuraam");
    f.add(new Tripsupaneel(), BorderLayout.CENTER);
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```

```
class Tripsupaneel extends Panel{
  Button nupp[]=new Button[10];
  String omanimi;
  TextArea suhtlus=new TextArea("",3, 60,TextArea.SCROLLBARS_VERTICAL_ONLY);
  public Tripsupaneel(){
    setLayout(new BorderLayout());
    add(suhtlus, BorderLayout.NORTH);
    Panel nupupaneel=new Panel();
    nupupaneel.setLayout(new GridLayout(3, 3));
    for(int i=0; i<9; i++){
      nupp[i]=new Button("  ");
      nupupaneel.add(nupp[i]);
    }
    add(nupupaneel, BorderLayout.CENTER);
    nupp[9]=new Button("Stopp");
    add(nupp[9], BorderLayout.SOUTH);
    seaNupud(false);
    TripsukliendiLoim vahendaja=new TripsukliendiLoim(this);
  }


  void seaNupud(boolean kasutatav){
    if(!kasutatav)for(int i=0; i<nupp.length; i++)nupp[i].setEnabled(kasutatav);
    else{
      for(int i=0; i<9; i++)if(nupp[i].getLabel().trim().equals(""))
         nupp[i].setEnabled(kasutatav);
    }
    nupp[9].setEnabled(kasutatav);
  }
}


class TripsukliendiLoim implements ActionListener, Runnable{
  Tripsupaneel tp;
  PrintWriter valja;
  BufferedReader sisse;
  boolean veel=true;
  Socket sc;


  public TripsukliendiLoim(Tripsupaneel utp){
    tp=utp;
    for(int i=0; i<10; i++)tp.nupp[i].addActionListener(this);
    alusta();
  }


  public boolean alusta(){
    boolean korras=true;
    try{
     for(int i=0; i<9; i++)tp.nupp[i].setLabel(" ");
     tp.nupp[9].setLabel("Stopp");
     tp.suhtlus.setText("");
     sc=new Socket(Tripsuklient.serverinimi, Tripsuserver.pordinr);
     sisse=new BufferedReader(
        new InputStreamReader(sc.getInputStream())
     );
     valja=new PrintWriter(sc.getOutputStream(), true);
     veel=true;
     new Thread(this).start();
    }catch(IOException e){
      e.printStackTrace();
      tp.suhtlus.setText("Ühendus serveriga "+Tripsuklient.serverinimi+" puudub.");
      korras=false;
    }
    return korras;
  }


  public void run(){
    System.out.println("Loime algus");
    try{
       while(veel){
         tootle(sisse.readLine());
       }
    } catch (Exception e){
      System.out.println("Probleem: "+e);
      e.printStackTrace();
    }
    System.out.println("Loime ots");
  }


  private void tootle(String teade) throws IOException{
    tp.suhtlus.setText(tp.suhtlus.getText()+teade+"\n");
    if(teade.startsWith(".margiks"))tp.omanimi=teade.substring(8, 9);
    if(teade.startsWith(".kaik")){
      try{
```

```
        int nr=Integer.parseInt(teade.substring(6, 7));
        tp.nupp[nr].setLabel(
          " "+teade.substring(5, 6)+" "
        );
      } catch(NumberFormatException e){} //sobimatu ruut
      if(!teade.substring(5, 6).equals(tp.omanimi))tp.seaNupud(true);
    }
    if(teade.equals(Tripsuserver.lopp)){
      veel=false;
      tp.nupp[9].setLabel("Start");
      tp.nupp[9].setEnabled(true);
      sc.close();
    }
  }

  public void actionPerformed(ActionEvent e){
    System.out.println("Vajutus");
    tp.seaNupud(false);
    int nr=0;
    Button allikas=(Button)e.getSource();
    for(int i=0; i<10; i++)if(tp.nupp[i]==allikas)nr=i;
    if(nr<9)
      valja.println(".kaik"+tp.omanimi+nr);
    else if(nr==9){
      if(tp.nupp[9].getLabel().equals("Stopp"))
         valja.println(Tripsuserver.lopp);
      else {
        tp.nupp[9].setLabel("Stopp");
        alusta();
      }
    }
  }
}
```

Jututoa graafiline klient.

Märgatava osa graafiliste võrguprogrammide puhul tuleb rakenduse töölesaamiseks lahendada hulk sarnaseid probleeme. Jututoa näite põhjal püüame need lõigud läbi käia.

Lihtsaim komplekt.

Alustame lihtsast Java põhikursuse konspektis kirjeldatud serverist ning kirjutame sinna juurde graafilise kliendi andmete saatmiseks ja lugemiseks. Serveri pea ainsaks oskuseks on kõik kasutajate poolt tulnud andmed kõigile sisse meldinud klientidele laiali saata.

Klient seevastu võtab ühenduse serveriga. Iga kasutaja tipitud rea sisestuse järel saadetakse see serverisse. Kõik serverist saabunud read aga paigutatakse üksteise järel tekstialasse. Ühendus serveriga luuakse kliendi käivitamise ajal. Serveri nimi ja värat on koodi sees kirjas. Paigutuseks piisab kolmest reast.

```
    setLayout(new BorderLayout());
    add(tf1, BorderLayout.SOUTH);
    add(ta1, BorderLayout.CENTER);
```

BorderLayout lubab paigutada servadesse ja keskele. Alla serva paigutatakse tekstiväli teadete sisestamiseks. Keskele tekstiala andmete lugemiseks. Tekstivälja venitatakse laiust pidi vastavalt akna suurusele. Tekstiala katab akna sees ülejäänud keskele jääva vaba pinna.

```
       Socket sc=new Socket(serverinimi, 3001);
```

küsib pistikühenduse soovitud serveri ja väratiga. Kui ühenduse loomine õnnestus, siis küsitakse sinna pistikusse sisnd- ning väljundvoog. Parameeter true PrintWriteri konstruktoris tähendas, et andmed saadetakse iga println-käskusega teele. Vastasel juhul võiks juhtuda, et ennem kogutakse mitme kilobaidi jagu teateid, kui programm leiab vajaliku neid võrku mööda teele saata.

```
       pw1=new PrintWriter(sc.getOutputStream(), true);
```

```
br1=new BufferedReader(new
            InputStreamReader(sc.getInputStream()));
```

Kui ühendused loodud, lükatakse tööle eraldi lõim teadete püüdmiseks. Kui jääda lihtsalt miskis alamprogrammis ootama võrgust saabuvaid teateid, siis võib jääda kogu rakendus seniks hangunuks, kuni sealt rida saabub. On aga teadete vastuvõtt omaette lõimes, siis see ootamine muid toiminguid ei sega.

```
new Thread(this).start();
```

Meetodis run paiknev teadete vastuvõtt on kirja pandud küllalt lühidalt:

```
while(true){
  ta1.append(br1.readLine()+"\n");
}
```

Ehk siis igavene tsükkel, kus readLine muudkui ootab võrgu pealt saabuvat teadet. Saabumise järel lisatakse see tekstialasse koos järgneva reavahetusega ning asutakse taas uut rida ootama. Juhtub aga lugemisega probleeme olema, satutakse tsüklist välja katsendiplokki. Ning programmi sulgemiseks pole esiotsa viisakamat moodust kui protsessi töö kas siis Ctrl+C või mõne muu vahendi abil lõpetada. Omad puudused sel kliendil on, kuid lihtsaks ühenduse testimiseks peaks sobima küll.

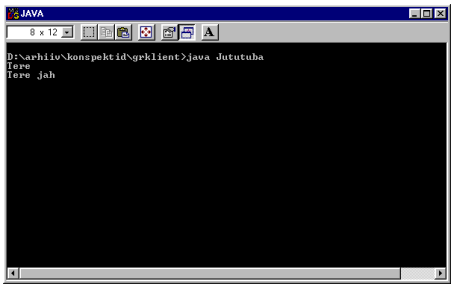

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;


public class GrKlient1 extends Applet
                implements ActionListener, Runnable{
  TextField tf1=new TextField(15); //suurus
  TextArea ta1=new TextArea(5, 20);
  PrintWriter pw1;
  BufferedReader br1;
  final String serverinimi="localhost";
  public GrKlient1(){
    setLayout(new BorderLayout());
    add(tf1, BorderLayout.SOUTH);
    add(ta1, BorderLayout.CENTER);
    tf1.addActionListener(this);
    try{
       Socket sc=new Socket(serverinimi, 3001);
       pw1=new PrintWriter(
                 sc.getOutputStream(), true);
       br1=new BufferedReader(new
         InputStreamReader(sc.getInputStream()));
       new Thread(this).start();
    }catch(Exception viga){
      ta1.setText(viga.getMessage());
    }
  }


  public void actionPerformed(ActionEvent e){
     pw1.println(tf1.getText());
     tf1.setText("");
  }

  public void run(){
   try{
    while(true){
      ta1.append(br1.readLine()+"\n");
    }
   }catch(Exception viga){
     viga.printStackTrace();
     ta1.append("Oled väljas");
   }
  }


  public static void main(String[] argumendid){
     Frame f=new Frame();
     f.add(new GrKlient1());
     f.setSize(300, 300);
     f.setVisible(true);
  }
}
```

Lihtne server

Lühidalt taas toodud koodilõik, mille abil võimalik käivitada serverprogramm, kuhu huvilised liituma pääsevad ning mis igalt kasutajalt saabuvad teated kõigile edasi annab. Kuna siinne programm ei tea koodi sisust midagi, siis võib sama "mootori" külge ehitada rakendusi vastavalt kirjutaja fantaasiale.

```
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);
  }
}
```

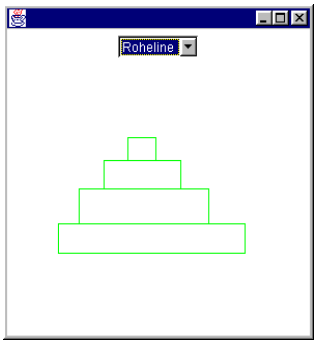


Tahvel

Soovides jututoale külge ehitada tahvlit, on enne hea järele proovida, kuidas lihtne joonistusvahend eraldi elama panna. Esiotsa joonistatakse kujund mouseReleased käskluse sees, ehk kohe, kui soovitud andmed teada on. paint-meetodi puudumise tõttu akna suurendamisel või korraks teise alla peitmisel lähevad andmed kaduma - see lihtsustusest tulenev viga parandatakse järgmises näites. Värv valitakse vastavalt rippmenüüle. Kuna värv määratakse elemendi järjekorranumbri ja mitte teksti abil, siis oleks vajadusel võimalik rakendus tõlkida mõnda muusse keelde ilma, et sõnade muutmine toimimist takistaks.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Tahvel1 extends Applet
  implements MouseListener{
  int hax, hay, hyx, hyy;
  String[] varvid={"Sinine", "Punane", "Kollane", "Roheline"};
  Color[] c={Color.blue, Color.red, Color.yellow, Color.green};
  Choice varvivalik=new Choice();
  public Tahvel1(){
    addMouseListener(this);
    for(int i=0; i<varvid.length; i++){
      varvivalik.addItem(varvid[i]);
    }
    add(varvivalik);
  }
```

```
  public void mousePressed(MouseEvent e){
    hax=e.getX();
    hay=e.getY();
  }
  public void mouseReleased(MouseEvent e){
    hyx=e.getX();
    hyy=e.getY();
    int laius=hyx-hax;
    int korgus=hyy-hay;
    Graphics g=getGraphics();
    g.setColor(c[varvivalik.getSelectedIndex()]);
    g.drawRect(hax, hay, laius, korgus);
  }
  public void mouseClicked(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
  public static void main(String[] argumendid){
    Frame f=new Frame();
    f.add(new Tahvel1());
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```



Täiendatud tahvel

Eelnevalt sai meelde tuletada, kuidas midagi pinnale joonistada õnnestub. Selline tahvel võib olla muu rakenduse juures küll niisama joonistusharjutuseks, kuid kuigi lihtsat tahvli pilti mõne muu programmi osaga ühendada ei saa.

Järgnevalt on kirjeldatud liides JooniseKuular ning seal sees käsklus, mis mõeldud kujundi andmete ühest komponendist teise saatmiseks.

```
interface JooniseKuular{
  public void kujund(String rida);
}
```

Kompilaator kontrollib vaid vastava käsu olemasolu ning lubab liidesetüüpi muutuja kaudu vastavaid käsklusi välja kutsuda. Rea formaat tuleb aga ise välja mõelda ja pärast sellest kinni pidada, et liidest kasutavad objektid suudaksid teineteisele käsklusi ühemõtteliselt edasi anda. Määran siis oma mõttes (ja vajadusel kirjutan ka liidese juurde kommentaarina üles), et rea abil saab edasi anda joont, ristkülikut või ovaali. Esimesel juhul algab rida tähekombinatsiooniga .pj, teisel . pr ning kolmandal .po. Edasi järgnevad juba neli koordinaati, mis Javas vastavate kujundite joonistamisel tarvilikud on. Joon ekraanipunktidest 10, 50 punktideni 100, 50 ehk horisontaaljoon peaks siis käsuna välja nägema

```
.pj 10 50 100 50
```

Hiire sündmuste vastuvõtt ning joonistamine ekraanil on teineteisest nõnda lahku viidud, et ainsaks ühenduslüliks nende vahel on loodud kuularliides. Hiire ülesliikumisel käivitatakse käsklus saada, mis siis meelde jäetud koordinaatide abil paneb kokku kujundi määrava rea ning kuulaja olemasolu korral edastab rea kuulajale liidesest võetud käsu kujund abil. Kas kuulajaks on teine tahvel, jututuba või kohalik tahvel ise, sellest ei tea joonistusrida kokkupanev kood midagi. Tahvel2 konstruktoris on rida

```
    kuulaja=this; //Vaikimisi joonistatakse iseenesele.
```

mis nagu juurdelisatud kommentaarigi tähendab, teatab, et kui hiljem pole midagi ümber määratud, siis tahvli sündmuste kuulajaks on tahvel ise, s.t. tahvlile joonistatud kujundid ilmuvad tahvlile enesele.

Väljapoolt tahvli kuulari määramiseks on loodud käsklus

```
public void seaJooniseKuular(JooniseKuular j){
  kuulaja=j;
}
```

Nagu näha, saab käsule ette anda JooniseKuular-tüüpi liidest realiseeriva objekti, kuhu siis edaspidi tahvlil hiirega toimetamistest tekkinud sündmuste põhjal kokku pandud kujundeid kirjeldavad read edasi saadetakse. Erinevalt järgmisest näitest saab siin korraga vaid üks objekt olla tahvlil joonistamise kuulariks. Selline "ühe kuulaja tava" on rohkem kasutusel Java mobiilirakenduste juures, kus programmid väiksemad ning ressursse usinamini kokku hoitakse.

Saatmisel kontrollitakse kõigepealt, kas üldse keegi joonistatavate andmete vastu huvi tunneb. Kui kuulaja on null, siis pole keegi saabuvatest andmetest huvitatud ning teksti pole vaja ka kokku panna. Sarnaseid kokkuhoidmiskohti õnnestub mõnigikord koodi sisse paigutada ning keerulisemate arvutuste ja joonistuste korral võib nii märgatavalt hoida kokku arvuti tööaega.

Edasi leitakse kujundi laius ja kõrgus ehk hiire alla- ja ülesliikumise koordinaatide vahe. Siis saadetakse vastavalt valitud kujundile andmed kuulaja poole teele.

```
void saada(){
  if(kuulaja==null){return;}
  int laius=hyx-hax;
  int korgus=hyy-hay;
  if(kujundivalik.getSelectedItem().equals("Joon")||
     kujundivalik.getSelectedItem().equals("Vabakäejoon")){
    kuulaja.kujund(".pj "+hax+" "+hay+" "+hyx+" "+hyy);
  }
  if(kujundivalik.getSelectedItem().equals("Ristkülik")){
    kuulaja.kujund(".pr "+hax+" "+hay+" "+laius+" "+korgus);
  }
  if(kujundivalik.getSelectedItem().equals("Ovaal")){
    kuulaja.kujund(".po "+hax+" "+hay+" "+laius+" "+korgus);
  }
}
```

Kuna ka tahvel ise realiseerib liidest JooniseKuular, et ta saaks jututoa kliendilt, teiselt tahvlilt või iseeneselt sama liidese kaudu jooniste teateid vastu võtta, siis peab ka tahvli sees olema meetod kujund sõnelise parameetriga. Nagu näha, palutakse vastuvõetud kujundi puhul kõigepealt see ekraanile joonistada ning siis lisatakse saabunud kujundi andmed hoidlasse, et oleks paint-meetodis võimalik ekraaniseis taastada.

```
public void kujund(String andmed){
  joonista(andmed);
  hoidla.add(andmed);
}
```

Joonistuskäskluses lõigatakse andmeid hoidev rida StringTokenizeri abil lõikudeks, leitakse andmereast joonistamiseks vastavad koordinaadid ning esimene jupp reast näitab, millise kujundiga on tegemist. Katsendiplokk on käskudele ümber pandud selleks, et üksik hulka sattunud vigane käsklus ei takistaks kogu pildi joonistamist. Praegusel juhul jääb lihtsalt konkreetne joonistuskäsk täitmata, väljakutsuvale funktsioonile aga sellekohast teadet edasi ei anta.

```
void joonista(String andmed){
 try{
  Graphics g=getGraphics();
  StringTokenizer stk=new StringTokenizer(andmed);
  String tyyp=stk.nextToken();
  int a[]=new int[4];
  for(int i=0; i<4; i++){
     a[i]=Integer.parseInt(stk.nextToken());
  }
  if(tyyp.equals(".pj")){
    g.drawLine(a[0], a[1], a[2], a[3]);
  }
  if(tyyp.equals(".pr")){
    g.drawRect(a[0], a[1], a[2], a[3]);
  }
  if(tyyp.equals(".po")){
    g.drawOval(a[0], a[1], a[2], a[3]);
  }
```

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

Meetodis paint on küllalt lühidalt hakkama saadud. Käiakse läbi kõik hoidlas olevad elemendid ning iga rea puhul palutakse sellele vastavad andmed komponendi pinnale joonistada. Listi käsklus iterator väljastab objekti, mille abil võimalik mööda ahelat üha järgmisi elemente küsida. Käsklus hasNext kontrollib, kas veel midagi tulemas on ning next võtab siis järgmise elemendi. Et hoidlas püsivad kõik read ülemklassi Object isendina, siis et saaks andmeid omistada String-tüüpi muutujale, selleks tuleb soovitud tüüp sulgudes ette kirjutada. Edasi juba käsklus joonista andmereale vastava kujundi ekraanile kuvamiseks.

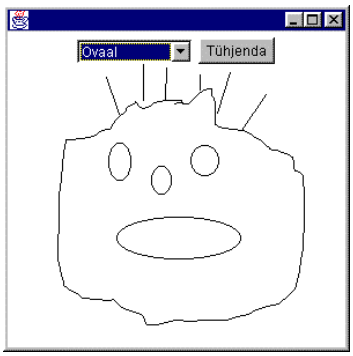
```
  public void paint(Graphics g){
     for(Iterator it=hoidla.iterator(); it.hasNext();){
       String s=(String)it.next();
       joonista(s);
     }
  }
```

Kood

Ning edasi rakenduse kood tervikuna.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class Tahvel2 extends Applet
  implements MouseListener, MouseMotionListener, JooniseKuular, ActionListener{
  int hax, hay, hyx, hyy;
  String[] kujundid={"Joon", "Ristkülik", "Ovaal", "Vabakäejoon"};
  Choice kujundivalik=new Choice();
  JooniseKuular kuulaja=null;
  LinkedList hoidla=new LinkedList();
  Button tyhjenda=new Button("Tühjenda");

  //objekt, kellele saadetakse tahvlil toimunud sündmused.
  public Tahvel2(){
    addMouseListener(this);
    addMouseMotionListener(this);
    for(int i=0; i<kujundid.length; i++){
      kujundivalik.addItem(kujundid[i]);
    }
    add(kujundivalik);
    add(tyhjenda);
    tyhjenda.addActionListener(this);
    kuulaja=this; //Vaikimisi joonistatakse iseenesele.
  }
  public void seaJooniseKuular(JooniseKuular j){
    kuulaja=j;
  }
  public void paint(Graphics g){
     for(Iterator it=hoidla.iterator(); it.hasNext();){
       String s=(String)it.next();
       joonista(s);
     }
  }

  void joonista(String andmed){
   try{
    Graphics g=getGraphics();
    StringTokenizer stk=new StringTokenizer(andmed);
    String tyyp=stk.nextToken();
    int a[]=new int[4];
    for(int i=0; i<4; i++){
       a[i]=Integer.parseInt(stk.nextToken());
    }
    if(tyyp.equals(".pj")){
      g.drawLine(a[0], a[1], a[2], a[3]);
    }
    if(tyyp.equals(".pr")){
      g.drawRect(a[0], a[1], a[2], a[3]);
    }
    if(tyyp.equals(".po")){
      g.drawOval(a[0], a[1], a[2], a[3]);
    }
```

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

  public void kujund(String andmed){
    joonista(andmed);
    hoidla.add(andmed);
  }

  public void mousePressed(MouseEvent e){
    hax=e.getX();
    hay=e.getY();
  }
  public void mouseReleased(MouseEvent e){
    hyx=e.getX();
    hyy=e.getY();
    saada();
  }
  public void actionPerformed(ActionEvent e){
    if(e.getSource()==tyhjenda){
      hoidla.clear();
      repaint();
    }
  }
  public void mouseDragged(MouseEvent e)
  {
    if(kujundivalik.getSelectedItem().equals("Vabakäejoon")){
        hyx = e.getX();
        hyy = e.getY();
        saada();
        hax = hyx;
        hay = hyy;
    }
  }
  public void mouseMoved(MouseEvent e){}


  void saada(){
    if(kuulaja==null){return;}
    int laius=hyx-hax;
    int korgus=hyy-hay;
    if(kujundivalik.getSelectedItem().equals("Joon")||
       kujundivalik.getSelectedItem().equals("Vabakäejoon")){
      kuulaja.kujund(".pj "+hax+" "+hay+" "+hyx+" "+hyy);
    }
    if(kujundivalik.getSelectedItem().equals("Ristkülik")){
      kuulaja.kujund(".pr "+hax+" "+hay+" "+laius+" "+korgus);
    }
    if(kujundivalik.getSelectedItem().equals("Ovaal")){
      kuulaja.kujund(".po "+hax+" "+hay+" "+laius+" "+korgus);
    }
  }

 public void mouseClicked(MouseEvent e){}
 public void mouseEntered(MouseEvent e){}
 public void mouseExited(MouseEvent e){}
 public static void main(String[] argumendid){
   Frame f=new Frame();
   f.add(new Tahvel2());
   f.setSize(300, 300);
   f.setVisible(true);
 }
}
```



Jututoa tahvliga klient.

Kui andmete sisselugemise ja väljasaatmise võimega tahvel olemas, siis tuleb see vaid jututoa kliendile sobivalt külge ühendada ning inimesed saavadki üksteisele pilte ja jooniseid saatma hakata. Kujundite andmeid on vaja saata mõlemas suunas. Võrku ühendunud klient edastab võrgust saabunud kujundikäsklused tahvlile joonistamiseks. Samas tahvlil toimunud hiiretoimingute põhjal saadetakse teated võrku ühendunud kliendile. Et see omakorda saadab teated serverisse, server aga

kõigile klientidele laiali, siis jõuab nõnda ringiga ka kohalikus masinas hiirega määratud kujund ekraanile. Kuna tahvlile jõuavad kõik teated ühtviisi võrgu kaudu sõltumata sellest, kas teade pandi teele kohalikust või mõnest muust masinast, siis on kujundi ekraanile ilmumine ka tõend selle kohta, et andmed on serverisse läbi läinud ning sealt tagasi jõudnud.

Et klient ning tahvel on pandud vastastikku teineteisele kujundite teateid saatma, on näha kliendi konstruktori viimastest ridadest.

```
tahvel.seaJooniseKuular(this);
seaJooniseKuular(tahvel);
```

Väike mõtlemisülesanne: mis juhtuks siis, kui klient oleks enesele ning tahvel enesele jooniskuulariks pandud. Ning ülesande lahendus:

Tahvliga ei juhtuks midagi erilist, sest kui tahvel iseenesele teateid saadaks, siis tahvli peal hiirega joonistatud kujundid tekiksid tahvlile enesele, just nii nagu kasutaja seda ootabki. Kui aga klient asuks võrgust saabunud teateid enesele saatma, siis oleks muresid rohkem. Algul ei pruugiks midagi hullu juhtuda - seni, kuni keegi pole veel ühtki kujundit teele saatnud. Samuti ei satu kohalikul tahvlil tehtud kujundid võrku juhul, kui tahvel teateid vaid iseenesele saadab. Kui nüüd aga mõne teise kliendi kaudu või lihtsalt kasutaja tippimise peale satuks kasvõi üks kujundi joonistamist nõudev käsk võrku, siis edasi tekiks tsükkel. Serveri ülesandeks on klientidelt saabuvad teated kõikidele klientidele edasi saata. Kui nüüd juhtuks selline klient tekkima, kes võrgust tulnud pilditeated enesele saadaks nii nagu tuleksid need tahvlilt, siis jääks pildikäsk võrku tiirlema. Ikka kliendilt serverile ja tagasi. Ning kui sarnase omadusega kliendi eksemplare oleks serveri küljes mitu, tekiks ahelreaktsioon: iga serveri poolt saadetud pildisoovi peale tuleks igalt kujundit tagasisaatvalt kliendilt teade kujundi kohta. Need saadaks server jälle kõikidele laiali ning mõne aja pärast oleks võrgus korralik kaos.

Et aga tahvel ja klient vastastikku andmeid vahetavad, siis kirjeldatud probleem oli vaid uitmõte ning siinne näide peaks korralikult töötama.

Võrreldes eelmise klientprogrammiga on näha muutujat seisund, näiteks:

```
String seisund="algus";
```

Selle abil määratakse, millises staadiumis rakendus parajasti on. Seisunditeks on veel nimesisestus, paroolisisestus ja tavatekst. Siinne rakendus eeldab, et kasutajalt küsitaks serveri pool nime ja parooli ning alles nende sobivuse korral lastaks võrku suhtlema nagu allpool kirjeldatud serverprogramm teeb. Samas aga on rakendus võimeline ühendust pidama ka eelpool kirjeldatud lihtsama serverprogrammiga, kes kõik ühendujad kohe jutule võtab ning neilt tulnud andmed kogu kuulajaskonnale laiali paiskab. Nagu koodi piiluda ja toimingute järjekordadele mõelda, siis esiotsa eeldatakse, et kasutaja vajutab nupule ühenda, edasi sisestab nime, siis parooli ning ühenduse õnnestumise korral asub teateid saatma ja vastu võtma. Parooli sisestamise ajaks määratakse tekstiväljas nähtavad tähed tärnideks käsu setEchoChar abil. Kui tahta taas tekstiväljas näha kirjutatavaid tähti endid, siis aitab, kui näidatavaks määrata täht koodina 0.

Meetodis run määratakse, et kõik .p-ga algavad read saadetakse tahvlile joonistamiseks, ülejäänud read paigutatakse tekstialasse.

```
String rida=br1.readLine();
if(rida.startsWith(".p")){
  saada(rida);
}else{
  ta1.append(rida+"\n");
}
```

Saatmine iseenesest lihtne. Igaks juhuks kontroll, et tahvel ikka ühendatud on ning edasi lihtsalt kuulaja vastava meetodi väljakutse.

```
void saada(String andmed){
  if(kuulaja==null){return;}
  kuulaja.kujund(andmed);
}
```

Samuti käib lühidalt tahvlilt saabunud teadete edasisaatmine. Ilma mingi täiendava

kontrollita saadetakse need lihtsalt PrintWriteri abil serveri poole teele.

```
  public void kujund(String andmed){
    pw1.println(andmed);
  }
```

main-meetodis veel eelmise kliendiga võrreldes juures käsklus, mis akna sündmustele kuulari lisab.

```
     f.addWindowListener(new Raamikuular());
```

Kuular ise paar rida allpool, staatilise sisemise klassina. Ning ainsaks toiminguks tal kogu virtuaalmasina julm sulgemine akna sulgemisristile vajutamise puhul. Tahtes akna sulgemisristi kaudu võrgukliendil paluda serveri küljest viisakamalt väljuda, tuleks teha mõningane ring. Üheks võimaluseks oleks aknakuulamisoskused siduda otse kliendiklassi külge. Sel puhul võiks juba kliendis eneses otsustada, mis sulgemisteate peale teha - küsida kasutajalt täiendavat nõusolekut, saata serverile lõpetusteade või lihtsalt ühendus sulgeda. Selline lähenemine aga eeldaks kliendiklassis kõikide aknaga seotud käskluste realiseerimist olgu siis või tühjade meetoditena.

```
  static class Raamikuular extends WindowAdapter{
    public void windowClosing(WindowEvent e){
      System.out.println("Programmi ots");
      System.exit(0);
    }
  }
```

Ning kood tervikuna.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;


public class GrKlient2 extends Applet
     implements ActionListener, Runnable, JooniseKuular{
  TextField tf1=new TextField(15); //suurus
  Button nupp1=new Button("Ühenda");
  TextArea ta1=new TextArea(5, 20);
  Panel p1=new Panel(new GridLayout(2, 1)); //alumine
   //2 rida ja 1 veerg tabelis
  Panel p2=new Panel(new GridLayout(2, 1)); //tekst, tahvel
  Tahvel2 tahvel=new Tahvel2();
  Label silt1=new Label();
  PrintWriter pw1;
  BufferedReader br1;
  String seisund="algus";
  String serverinimi="localhost";
  JooniseKuular kuulaja=null;
  public GrKlient2(){
    setLayout(new BorderLayout());
    p1.add(tf1);
    p1.add(silt1);
    add(p1, BorderLayout.SOUTH);
    add(nupp1, BorderLayout.NORTH);
    p2.add(ta1);
    p2.add(tahvel);
    add(p2, BorderLayout.CENTER);
    tf1.addActionListener(this);
    nupp1.addActionListener(this);
    tahvel.seaJooniseKuular(this);
    seaJooniseKuular(tahvel);
  }
  public void seaJooniseKuular(JooniseKuular j){
    kuulaja=j;
  }
  public void actionPerformed(ActionEvent e){
    if(e.getSource()==tf1){
       pw1.println(tf1.getText());
       tf1.setText("");
       if(seisund.equals("nimesisestus")){
         silt1.setText("Palun parool");
         tf1.setEchoChar('*');
         seisund="paroolisisestus";
```

```
      } else if(seisund.equals("paroolisisestus")){
        tf1.setEchoChar((char)0);
        silt1.setText("Kirjuta rahus");
        seisund="tavatekst";
      }

    }
    if(e.getSource()==nupp1){
      try{
         Socket sc=new Socket(serverinimi, 3001);
         pw1=new PrintWriter(sc.getOutputStream(), true);
         br1=new BufferedReader(new
           InputStreamReader(sc.getInputStream()));
         new Thread(this).start();
         silt1.setText("Palun nimi: ");
         seisund="nimesisestus";
      }catch(Exception viga){
        ta1.setText(viga.getMessage());
      }
    }
  }

  public void run(){
   try{
    while(true){
      String rida=br1.readLine();
      if(rida.startsWith(".p")){
        saada(rida);
      }else{
        ta1.append(rida+"\n");
      }
    }
   }catch(Exception viga){
     viga.printStackTrace();
     ta1.append("Oled väljas");
   }
  }
  void saada(String andmed){
    if(kuulaja==null){return;}
    kuulaja.kujund(andmed);
  }
  public void kujund(String andmed){
    pw1.println(andmed);
  }


  public static void main(String[] argumendid){
     Frame f=new Frame();
     f.add(new GrKlient2());
     f.setSize(300, 300);
     f.setVisible(true);
     f.addWindowListener(new Raamikuular());
  }
  static class Raamikuular extends WindowAdapter{
    public void windowClosing(WindowEvent e){
      System.out.println("Programmi ots");
      System.exit(0);
    }
  }
}
```

Registreeritud kasutajatega server.

Järgnevalt algsest näitest mõnevõrra arukam serverprogramm. Sisenejatelt küsitakse kasutajanime ja parooli. Uue nime puhul teatatakse uuest kasutajast, tuttava kombinatsiooni puhul rõõmustatakse jällenägemise puhul ning olemasoleva kasutajanime kuid vigase parooliga meldimise korral sisse ei lasta. Andmed püsivad mälus küll vaid serveri tööajal, kuid faili kirjutamine ja sealt lugemine on siit puudu vaid näite lühiduse ettekäändel. Soovides andmeid püsivalt kettal talletada, oleks mõistlik need programmi käivitumisel mällu (HashMap-i) lugeda ning iga uue kasutaja lisandumisel vastav rida faili juurde kirjutada. Failioperatsioonid võiksid olla sünkroniseeritud nii nagu kasutajate loetelu puhul näha on. Sel juhul pole vaja peljata, et mitme lõime üheaegne failikirjutussoov andmeid rikkuda võiks.

Andmete hoidmiseks siis kaks kogu programmi piires kasutatavat andmestruktuuri. Üks

kasutajate lõimede jaoks, teine nimede ja paroolide tarbeks. Mõlema operatsioonid on määratud sünkroniseerituks, et lõimed korraga samu andmeid muutma ei asuks ning et ei asutaks küsima kasutaja andmeid, keda enam loetelus pole.

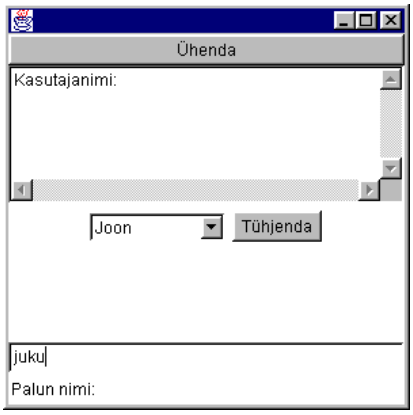
```
static List kasutajad=Collections.synchronizedList(new LinkedList());
static Map paroolid=Collections.synchronizedMap(new HashMap());
```

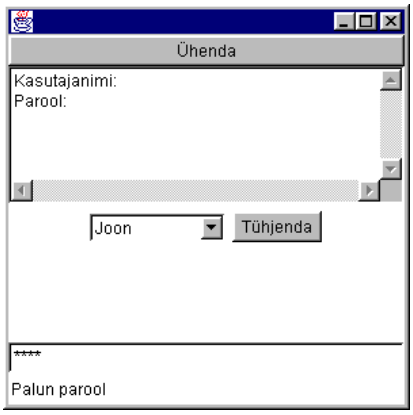
Võrreldes eelpool oleva serverinäitega ei hoita siin loetelus mitte kasutajate pistikuid, vaid eraldi objekte, kuhu on koondatud mitmed andmed kasutaja kohta. Et igal kasutajal on PrintWriter valja temale andmete saatmiseks, siis pole enam kasutajale andmete saatmiseks vaja igal korral pistikust väljundvoogu küsida, vaid sellele võib vastava muutuja kaudu kohe juurde pääseda.

```
synchronized(kasutajad){
    Iterator loend=kasutajad.iterator();
    while(loend.hasNext()){
      Kasutaja k=(Kasutaja)loend.next();
      k.valja.println(rida);
    }
  }
}
```

Kuna kasutajate loend on sünkroniseeritud ning kasutajatele andmete trükkimise tsükkel vastava loendi järgi sünkroniseeritud plokis, siis ei peaks saama kasutajaid trükkimise ajal lisada ning eemaldada.

Et kasutajale loodud klassi eksemplar lisatakse vektorisse alles pärast nime ja parooli küsimist, siis ei hakka ta ka enne muudelt inimestelt saabuvaid teateid saama, kui nime ja parooli sobivus kontrollitud.






```
import java.io.*;
import java.net.*;
import java.util.*;
public class ParooligaJututuba{
  static List kasutajad=
     Collections.synchronizedList(new LinkedList());
  static Map paroolid=
     Collections.synchronizedMap(new HashMap());
  public static void main(String argumendid[])
                                 throws IOException{
    ServerSocket ss=new ServerSocket(3001);
    while(true){
      new Kasutaja(ss.accept());
    }
  }



  static class Kasutaja extends Thread{
    Socket sc;
    PrintWriter valja;
    String kasutajanimi="";
    public Kasutaja(Socket uus_sc){
      sc=uus_sc;
      start();
    }
    public void run(){
     try{
      BufferedReader sisse=new BufferedReader(
        new InputStreamReader(sc.getInputStream())
      );
      valja=new PrintWriter(
                    sc.getOutputStream(), true);
      valja.println("Kasutajanimi: ");
      kasutajanimi=sisse.readLine();
      valja.println("Parool: ");
      String parool=sisse.readLine();
        //tavaline readLine ei suuda märke peita.
      if(paroolid.get(kasutajanimi)==null){
        valja.println(
           "Tere tulemast, uus kasutaja "+
            kasutajanimi);
        paroolid.put(kasutajanimi, parool);
      } else if(paroolid.get(kasutajanimi).
                                 equals(parool)){
        valja.println("Tere taas, "+kasutajanimi);
      } else {
        valja.println("Vigane meldimine.");
        sc.close();
        return;
      }
      kasutajad.add(this);
      boolean veel=true;
      while(veel){
         String rida=sisse.readLine();
         System.out.println(rida);
             //administraatori tarbeks
         if(rida.startsWith(".ots")){veel=false;}
         synchronized(kasutajad){
           Iterator loend=kasutajad.iterator();
           while(loend.hasNext()){
             Kasutaja k=(Kasutaja)loend.next();
             k.valja.println(rida);
           }
         }
      }
      sc.close();
    } catch(Exception e){
      System.out.println("Probleem: "+e);
    }
    kasutajad.remove(this);
  }
 }
}
```

```
D:\arhiiv\konspektid\grklient>java ParooligaJututuba
.pr 94 62 33 21
.pr 113 48 8 14
.pr 112 70 12 13
```

.pr 99 70 7 7
Joonistasin sulle maja pildi
Väljund serveriaknas

Kolmas arendusring.

Järgnevalt on tahvli koodi täiendatud javadoc-ile sobivate kommentaaridega. Nii õnnestub ülevaatlikkuse tarbeks genereerida koodi kohta mõningane dokumentatsioon ilma selle jaoks eraldiseisvat juttu kirjutamata. Joonistussündmustega ümber käimiseks on lihtne JooniseKuular-liidese tüüpi muutuja asemele paigutatud nimistu jooniseKuularid, mis võib eneses hoida kuulajaid nõnda palju kui parajasti tarvilik on.

```
LinkedList jooniseKuularid=new LinkedList();
```

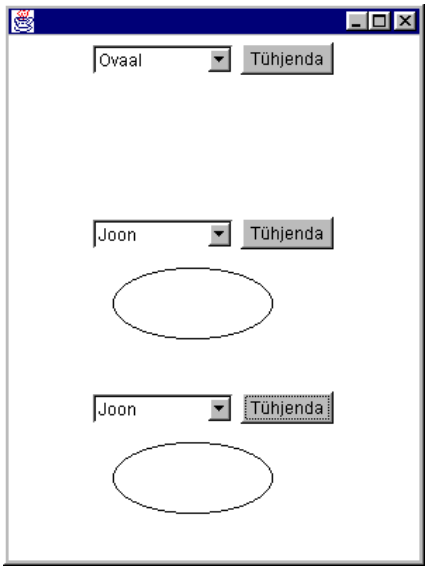
Et kannataks tahvlile väljapoolt kuulajaid külge panna, selleks alljärgnev meetod. Töötab teine samal põhimõttel, kui näiteks nupu või tekstivälja puhul addActionListener. Ehk sündmuse allikaks olev objekt jätab enese juurde meelde, kuhu tuleb sündmuse toimumise puhul teated välja saata.

```
public void lisaJooniseKuular(JooniseKuular j){
  jooniseKuularid.add(j);
}
```

Saatmise puhul tuleb hoolitseda, et teade kõikide sooviavaldanuteni jõuaks. Kui ennist oli valida kas nulli või ühe kuulaja vahel, siis nüüd alampiiriks endiselt null, ülempiiri pole aga seatud. Tõsi küll, mõni klass võib liiga suure kuulajaks registreerunute arvu puhul anda TooManyListenersException'i. Nagu ikka, on korduvate tegevuste puhul abiks tsükkel. Nimistust soovitud järjekorranumbriga elemendi aitab kätte saada get. Et oleme nimistusse paigutanud vaid JooniseKuulareid, siis võime ka kindlad olla, et sama tüüpi elemente sealt kätte saame.

```
void saada(){
 if(jooniseKuularid.size()==0){return;}
 int laius=hyx-hax;
 int korgus=hyy-hay;
 for(int i=0; i<jooniseKuularid.size(); i++){
  JooniseKuular kuulaja=(JooniseKuular)jooniseKuularid.get(i);
....
  if(kujundivalik.getSelectedItem().equals("Ovaal")){
    kuulaja.kujund(".po "+hax+" "+hay+" "+laius+" "+korgus);
  }
 }
}
```

Ehkki jututoa kliendi rakenduses piisab siinsele tahvlile vaid ühest kuulajast - kliendist, mille kaudu andmed võrku saadetakse kannatab tahvlile külge panna näiteks teisi tahvleid, nagu allpool olevas näites näha on. Tahvli t1 sündmustele reageerivad nii t2 kui t3.




```
import java.awt.*;
public class Tahvel3KuulariDemo{
  public static void main(String[] argumendid){
    Tahvel3 t1=new Tahvel3();
    Tahvel3 t2=new Tahvel3();
    Tahvel3 t3=new Tahvel3();
    Frame f=new Frame();
    f.setLayout(new GridLayout(3, 1));
    f.add(t1);
    f.add(t2);
    f.add(t3);
    t1.lisaJooniseKuular(t2);
    t1.lisaJooniseKuular(t3);
    f.setSize(300, 400);
    f.setVisible(true);
  }
}
```

Tahvlile lisati nupp.

```
Button tyhjenda=new Button("Tühjenda");
```

Nagu nimigi näitab, on see loodud ala puhastuseks, et õnnestuks soovi korral taas valgelt lehelt alustada. Tühjenduseks piisab vaid hoidlas olevate teadete kaotamisest - siis järgnev repaint vaid kustutab platsi taustaga ühte värvi, kuid midagi vaadatavat ei lisa.

```
public void actionPerformed(ActionEvent e){
  if(e.getSource()==tyhjenda){
    hoidla.clear();
    repaint();
  }
}
```

Et õnnestuks tervet pildi sisu kas kopeerida või arhiveerida, selleks juures vastav meetod. Hoidla enese osutit ei väljastata seetõttu, et kui osuti kätte saanud klass asuks hoidla sisu muutma, siis võiks see kergesti siinse pildi segamini keerata. Kui aga väljastatakse koopia, siis vastavat ohtu pole. Ehkki ka koopiahoidlasse jäävad osutid algsetele sõnedele ja mitte nende koopiatele, siis kuna klassi String eksemplari väärtust pole võimalik pärast selle loomist muuta, pole karta andmete muutumist algses hoidas.

```
public LinkedList hoidlaKoopia(){
  return new LinkedList(hoidla);
}
```

Eelmisega võrreldes vastandlik käsk: pildile saab ette anda uue sisu kollektsioonina. Nagu käskudest näha, tehakse uue sisu määramisel tahvel vanadest andmetest puhtaks ning lisatakse kõik, mis etteantust võtta on.

```
/**
* Uus hoidla sisu ja ekraanipilt.
*/
public void hoidlaUusSisu(Collection c){
  hoidla.clear();
  hoidla.addAll(c);
  repaint();
}
```

Eelnevad kaks käsku üheskoos võimaldavad ühe pildi sisu teisele kopeerida. Ühe tahvlilt küsitud andmed antakse ilusti teisele ette nagu järgnevas kahe tahvliga jututoa kliendi näites.

```
    if(e.getSource()==kopeeriPilt){
      tahvelOma.hoidlaUusSisu(tahvel.hoidlaKoopia());
    }
```

Ning laiendatud tahvli kood tervikuna.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;


public class Tahvel3 extends Applet
  implements MouseListener, MouseMotionListener, JooniseKuular, ActionListener{
  /**
  *  Kujundi alguse x (hiir alla)
  */
  int hax;
  /**
  *  Kujundi alguse y
  */
  int hay;
  /**
  * Kujundi lõpu x
  */
  int hyx;
  /**
  * Kujundi lõpu y
  */
  int hyy;
  /**
  * Joonistatavate kujundite loetelu.
  */
  String[] kujundid={"Joon", "Ristkülik", "Ovaal", "Vabakäejoon"};
  /**
  * Valikukombo.
  */
  Choice kujundivalik=new Choice();
  /**
  * Kujunditeadete püüdjate loend. Võivad olla nii tahvel ise, teine
  * tahvel, GrKlient kui mõni muu JooniseKuularit realiseeriva
  * klassi eksemplar.
  */
  LinkedList jooniseKuularid=new LinkedList();
  /**
  * Joonistamiseks meeles peetavad kujundid
  */
  LinkedList hoidla=new LinkedList();
  /**
  * Hoidla tühjendus, pildi puhastus.
  */
  Button tyhjenda=new Button("Tühjenda");


  /**
  * Initsialiseerimine
  */
  public Tahvel3(){
    addMouseListener(this);
    addMouseMotionListener(this);
    for(int i=0; i<kujundid.length; i++){
      kujundivalik.addItem(kujundid[i]);
    }
    add(kujundivalik);
    add(tyhjenda);
    tyhjenda.addActionListener(this);
  }

  /**
  * Kujundite kuulaja lisamine. Korraga suudab teateid
  * saata vaid ühele kuulajale.
  */
  public void lisaJooniseKuular(JooniseKuular j){
    jooniseKuularid.add(j);
  }

  /**
  * Joonis hoidla põhjal.
  */
  public void paint(Graphics g){
     for(Iterator it=hoidla.iterator(); it.hasNext();){
       String s=(String)it.next();
       joonista(s);
```

```
   }
}

/**
* Ühe saabunud kujundi lahkamine sõnest ja joonistus.
*/
void joonista(String andmed){
 try{
  Graphics g=getGraphics();
  StringTokenizer stk=new StringTokenizer(andmed);
  String tyyp=stk.nextToken();
  int a[]=new int[4];
  for(int i=0; i<4; i++){
     a[i]=Integer.parseInt(stk.nextToken());
  }
  if(tyyp.equals(".pj")){
    g.drawLine(a[0], a[1], a[2], a[3]);
  }
  if(tyyp.equals(".pr")){
    g.drawRect(a[0], a[1], a[2], a[3]);
  }
  if(tyyp.equals(".po")){
    g.drawOval(a[0], a[1], a[2], a[3]);
  }
 }catch(Exception viga){viga.printStackTrace();}
}

/**
* Saabuv kujund
*/
public void kujund(String andmed){
  joonista(andmed);
  hoidla.add(andmed);
}

/**
* Salvestatakse hiire allavajutuse koordinaadid.
*/
public void mousePressed(MouseEvent e){
  hax=e.getX();
  hay=e.getY();
}

/**
* Salvestatakse hiire üleslaskmise koordinaadid. Vajadusel
* luuakse kujund.
*/
public void mouseReleased(MouseEvent e){
  hyx=e.getX();
  hyy=e.getY();
  saada();
}

/**
* Tühjendusnupule reageerimine.
*/
public void actionPerformed(ActionEvent e){
  if(e.getSource()==tyhjenda){
    hoidla.clear();
    repaint();
  }
}

/**
* Lohistamine. Vähemasti vabakäejoone tarbeks.
*/
public void mouseDragged(MouseEvent e)
{
  if(kujundivalik.getSelectedItem().equals("Vabakäejoon")){
      hyx = e.getX();
      hyy = e.getY();
      saada();
      hax = hyx;
      hay = hyy;
  }
}

/**
* Tühi meetod, vajalik liikumisliidese realiseerimiseks.
*/
public void mouseMoved(MouseEvent e){}



/**
```

```
  * Andmete väljastus jooniseKuulajale. Kuulaja puudumise
  * korral ei väljastata midagi.
  */
  void saada(){
   if(jooniseKuularid.size()==0){return;}
   int laius=hyx-hax;
   int korgus=hyy-hay;
   for(int i=0; i<jooniseKuularid.size(); i++){
    JooniseKuular kuulaja=(JooniseKuular)jooniseKuularid.get(i);
    if(kujundivalik.getSelectedItem().equals("Joon")||
       kujundivalik.getSelectedItem().equals("Vabakäejoon")){
      kuulaja.kujund(".pj "+hax+" "+hay+" "+hyx+" "+hyy);
    }
    if(kujundivalik.getSelectedItem().equals("Ristkülik")){
      kuulaja.kujund(".pr "+hax+" "+hay+" "+laius+" "+korgus);
    }
    if(kujundivalik.getSelectedItem().equals("Ovaal")){
      kuulaja.kujund(".po "+hax+" "+hay+" "+laius+" "+korgus);
    }
   }
  }

  /**
  * Olemasolevate kujundite koopia. Muutumatu sisuga loetelu
  * joonistusaja tarbeks, samuti kogu pildi teele saatmiseks.
  */
  public LinkedList hoidlaKoopia(){
    return new LinkedList(hoidla);
  }

  /**
  * Uus hoidla sisu ja ekraanipilt.
  */
  public void hoidlaUusSisu(Collection c){
    hoidla.clear();
    hoidla.addAll(c);
    repaint();
  }

  /**
  * Tühi meetod, vajalik hiireliidese realiseerimiseks.
  */
  public void mouseClicked(MouseEvent e){}
  /**
  * Tühi meetod, vajalik hiireliidese realiseerimiseks.
  */
  public void mouseEntered(MouseEvent e){}
  /**
  * Tühi meetod, vajalik hiireliidese realiseerimiseks.
  */
  public void mouseExited(MouseEvent e){}
  /**
  * Võimaldab tahvlit iseseisvalt joonistuslõuendina käivitada.
  * Joonistab iseendale.
  */
  public static void main(String[] argumendid){
    Frame f=new Frame();
    Tahvel3 t=new Tahvel3();
    t.lisaJooniseKuular(t);
    f.add(t);
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```

Lühidokumentatsioon

Javadoci abil vormistatult näeb tahvli väljade ja meetodite dokumentatsioon välja järgmine. Esmasel vaatamisel peaks siit olema kergem klassi käsklused üles leida.

Väljad	
(package private) int	**hax** Kujundi alguse x (hiir alla)
(package private) int	**hay** Kujundi alguse y

(package private) java.util.LinkedList	**hoidla** Joonistamiseks meeles peetavad kujundid
(package private) int	**hyx** Kujundi lõpu x
(package private) int	**hyy** Kujundi lõpu y
(package private) java.util.LinkedList	**jooniseKuularid** Kujunditeadete püüdjate loend.
(package private) java.lang.String[]	**kujundid** Joonistatavate kujundite loetelu.
(package private) java.awt.Choice	**kujundivalik** Valikukombo.
(package private) java.awt.Button	**tyhjenda** Hoidla tühjendus, pildi puhastus.

Konstruktor	
Tahvel3() Initsialiseerimine	

Meetodid	
void	actionPerformed(java.awt.event.ActionEvent e) Tühjendusnupule reageerimine.
java.util.LinkedList	hoidlaKoopia() Olemasolevate kujundite koopia.
void	hoidlaUusSisu(java.util.Collection c) Uus hoidla sisu ja ekraanipilt.
(package private) void	joonista(java.lang.String andmed) Ühe saabunud kujundi lahkamine sõnest ja joonistus.
void	kujund(java.lang.String andmed) Saabuv kujund
void	lisaJooniseKuular(JooniseKuular j) Kujundite kuulaja lisamine.
static void	main(java.lang.String[] argumendid) Võimaldab tahvlit iseseisvalt joonistuslõuendina käivitada.
void	mouseClicked(java.awt.event.MouseEvent e) Tühi meetod, vajalik hiireliidese realiseerimiseks.
void	mouseDragged(java.awt.event.MouseEvent e) Lohistamine.
void	mouseEntered(java.awt.event.MouseEvent e) Tühi meetod, vajalik hiireliidese realiseerimiseks.
void	mouseExited(java.awt.event.MouseEvent e) Tühi meetod, vajalik hiireliidese realiseerimiseks.
void	mouseMoved(java.awt.event.MouseEvent e) Tühi meetod, vajalik liikumisliidese realiseerimiseks.
void	mousePressed(java.awt.event.MouseEvent e) Salvestatakse hiire allavajutuse koordinaadid.

void	mouseReleased(java.awt.event.MouseEvent e) Salvestatakse hiire üleslaskmise koordinaadid.
void	paint(java.awt.Graphics g) Joonis hoidla põhjal.
(package private) void	saada() Andmete väljastus jooniseKuulajale.

Kahe tahvliga klient.

Võrdlusseeria lõpetuseks juba rohkem ametliku väljanägemisega klient. Kasutajal tuleb määrata ühendumiseks vajalik server ja värat, samuti kasutajanimi ja parool. Edasi ekraan tühjendatakse ning sinna paigutatakse juba suhtlemiseks vajalikud vahendid: tekstiväli kirjutamiseks, tekstiala teadete vastuvõtmiseks ning kaks tahvlit: üks joonistamiseks ja andmete teele saatmiseks, teine võrgu pealt saabuva vaatamiseks.

Nagu koodist näha, näitab nupule nupp1 vajutus, et tuleb asuda ühendust võtma ning seejärel ekraan ümber kujundada. Serverist saabuvaid andmeid kontrollitakse ning kui esimesena ei küsita kasutajanime, siis pole satutud oodatud serveri otsa, on tegemist vigase protokolliga ning väljastatakse erind.

```
new Thread(this).start();
```

Lükkab tööle teadete püüdmiseks mõeldud lõime.

Elementide vahetamiseks eemaldatakse kõigepealt kõik ekraanile paigutatu käsuga

```
removeAll();
```

Edasi säetakse nii tahvlid, nupud kui tekstikastid paika ning lõpetuseks öeldakse

```
validate();
```

mis peaks hoolitsema, et ekraanile paigutatu ka kõik ilusti välja näidataks.

```
if(e.getSource()==nupp1){
  try{
     Socket sc=new Socket(tf3.getText(), Integer.parseInt(tf4.getText()));
     pw1=new PrintWriter(sc.getOutputStream(), true);
     br1=new BufferedReader(new
       InputStreamReader(sc.getInputStream()));
     if(!br1.readLine().equals("Kasutajanimi: ")){
       throw new IOException("Vigane protokoll");
     }
     pw1.println(nimesisestus.getText());
     br1.readLine(); //eeldatavasti küsitakse parooli
     pw1.println(tf2.getText());
     tf2.setText("");
     new Thread(this).start();
     removeAll();
     //Aken puhtaks ning uus paigutus
     setLayout(new BorderLayout());

  ...
     validate();
   }catch(Exception viga){
     nimesisestus.setText(viga.getMessage());
   }
}
```

Andmete võrku saatmise nupule on antud programmis kaks ülesannet. Kui ühendus olemas, siis nupule vajutades küsitakse tahvli peal asuv joonis ning sealsed teated saadetakse ükshaaval võrku, et ka ülejäänud vestlusel osalejad saaksid pilti näha.

Kui aga ühendus juhtub katkema, siis määratakse nupu peal olevaks tekstiks "Alusta" ning nupule vajutusel manatakse kasutaja ette algpaigutus, kus kasutajal on võimalik määrata, millise serveriga ja millisesse väratisse ühendust võtta. Nõnda ühe nupuga piirdudes pole vaja kujundust muutma asuda. Mõnel sarnasel klientprogrammil kipub kombeks olema kohe ühenduse katkemisel

sissemeldimisaken ette visata, kuid see ei tundu meeldiva lahendusena, sest nõnda pole võimalik enam eelnenud teksti üle vaadata. Nupu pealkirja järgi reageerimise miinuseks oleks aga võimatus kergesti silti muuta või tõlkida. Eraldi muutuja kasutamisel sellist probleemi pole.

```
    if(e.getSource()==piltVorku){
     if(yhendusOlemas){
      Iterator it=tahvelOma.hoidlaKoopia().iterator();
      while(it.hasNext()){
        pw1.println(it.next());
      }
     } else {
       removeAll();
       algpaigutus();
       validate();
       ta1.setText("");
       nimesisestus.setText("");
       piltVorku.setLabel("Võrku");
     }
    }
```

Kui võrgust lugemisel tekib viga või kui ühendus katkestatakse, sel juhul satub kood while-tsükli seest katsendiploki veatöötlusossa, kus jäetakse meelde ühenduse katmine ning vahetakse nupul pealkiri et kasutaja teaks sellel vajutades otsast alustada.

```
  public void run(){
   try{
    while(true){
      yhendusOlemas=true;
      String rida=br1.readLine();
      if(rida==null){
        throw new IOException("Ühenduse lõpp");
      }
      if(rida.startsWith(".p")){
        saada(rida);
      }else{
        ta1.append(rida+"\n");
      }
    }
   }catch(Exception viga){
     yhendusOlemas=false;
     piltVorku.setLabel("Alusta");
     ta1.append("Oled väljas");
   }
  }
```

Muud vahendid sarnased kui eelneval kliendil.

Vahepalaks Javadociga välja eraldatud käsklused koos kommentaaridega.

Meetodid	
void	actionPerformed(java.awt.event.ActionEvent e) Sündmustele reageerimise keskus.
void	algpaigutus() Paigutus, kus on näha sisenemiseks tarvilikud tekstiväljad.
void	kujund(java.lang.String andmed) Parameetrina saadavad andmed saadetakse võrku edasi.
static void	main(java.lang.String[] argumendid) Käivitus.
void	run() Teateid püüdva lõime käsud.
(package private) void	saada(java.lang.String andmed) Käsklus pildikujundi (edasi) saatmiseks (eeldatavalt tahvlile) juhul kui vastuvõtja on olemas.

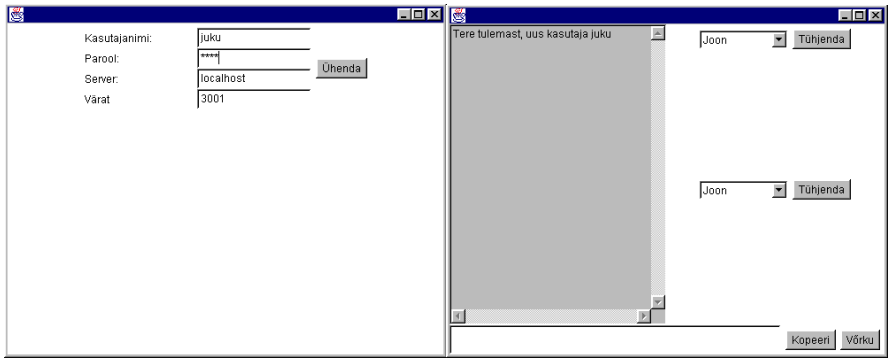
void	seaJooniseKuular(JooniseKuular j) Määratakse, millisele objektile saadetakse edasi kliendile saabunud teated pildile paigutatavate kujundite kohta.

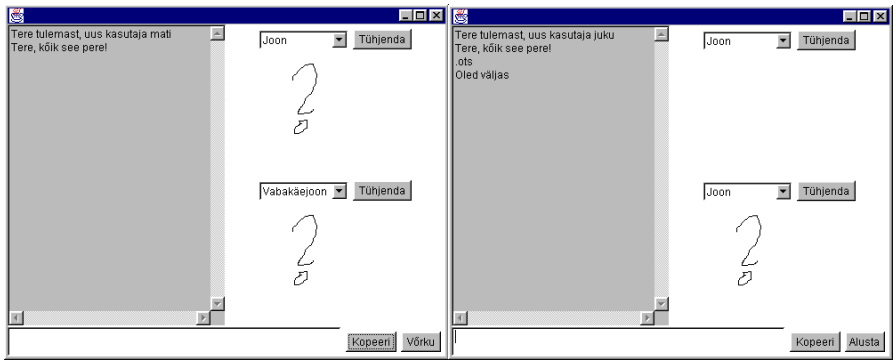
Ning edasi kahe tahvliga kliendi kood.

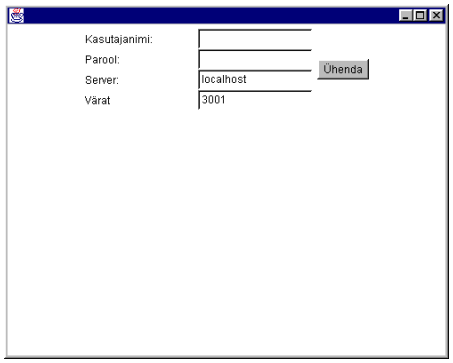
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.*;


public class GrKlient3 extends Applet
     implements ActionListener, Runnable, JooniseKuular{
  /**
  * Tekstiväli andmete võrku saatmiseks.
  */
  TextField tf1=new TextField(15);
  /**
  * Nimesisestuskoht.
  */
  TextField nimesisestus=new TextField(15);
  /**
  * Paroolisisestusväli.
  */
  TextField tf2=new TextField(15);
  /**
  * Serveri nime tarvis.
  */
  TextField tf3=new TextField("localhost", 15);
  /**
  * Serveri värati number.
  */
  TextField tf4=new TextField("3001", 15);
  /**
  * Vajutuse peale ühendatakse klient serveriga.
  */
  Button nupp1=new Button("Ühenda");
  /**
  * Serverist saabuvad andmed.
  */
  TextArea ta1=new TextArea(5, 20);
  /**
  * Võrgust saabuv joonis, võimalus otse võrku joonistada.
  */
  Tahvel3 tahvel=new Tahvel3();
  /**
  * Rahus omaette joonistamiseks. Hiljem võimalik kopeerida.
  */
  Tahvel3 tahvelOma=new Tahvel3();
  /**
  * Pilt võrgutahvlilt oma tahvlile.
  */
  Button kopeeriPilt=new Button("Kopeeri");
  /**
  * Pilt oma tahvlilt võrku.
  */
  Button piltVorku=new Button("Võrku");
  /**
  * Voog võrku saatmiseks.
  */
  PrintWriter pw1;
  /**
  * Voog võrgust lugemiseks.
  */
  BufferedReader br1;
  /**
  * Kuular kujundite saatmiseks. Siin juhul liiguvad
  * võrgust saabuvad andmed avalikule tahvlile.
  */
  JooniseKuular kuulaja=null;
  /**
  * Konstruktor kujunduse sättimiseks ning teadete saatmise seadmiseks.
  */
  boolean yhendusOlemas=false;
  /**
  * Kuularite paika sättimine.
  */
  public GrKlient3(){
    algpaigutus();
```

```
  nupp1.addActionListener(this);
  tf2.setEchoChar('*');
  tf1.addActionListener(this);
  kopeeriPilt.addActionListener(this);
  piltVorku.addActionListener(this);
  tahvel.lisaJooniseKuular(this);
  tahvelOma.lisaJooniseKuular(tahvelOma);
  seaJooniseKuular(tahvel);
  ta1.setEditable(false);
}


/**
* Paigutus, kus on näha sisenemiseks tarvilikud tekstiväljad.
*/
public void algpaigutus(){
  setLayout(new FlowLayout());
  Panel p1=new Panel(new GridLayout(4, 2));
  p1.add(new Label("Kasutajanimi:"));
  p1.add(nimesisestus);
  p1.add(new Label("Parool:"));
  p1.add(tf2);
  p1.add(new Label("Server:"));
  p1.add(tf3);
  p1.add(new Label("Värat"));
  p1.add(tf4);
  add(p1);
  add(nupp1);
}
/**
*  Määratakse, millisele objektile saadetakse edasi kliendile saabunud
*  teated pildile paigutatavate kujundite kohta.
*/
public void seaJooniseKuular(JooniseKuular j){
  kuulaja=j;
}
/**
* Sündmustele reageerimise keskus.
*/
public void actionPerformed(ActionEvent e){
  if(e.getSource()==tf1){
     pw1.println(tf1.getText());
     tf1.setText("");
  }
  if(e.getSource()==nupp1){
    try{
       Socket sc=new Socket(tf3.getText(), Integer.parseInt(tf4.getText()));
       pw1=new PrintWriter(sc.getOutputStream(), true);
       br1=new BufferedReader(new
         InputStreamReader(sc.getInputStream()));
       if(!br1.readLine().equals("Kasutajanimi: ")){
         throw new IOException("Vigane protokoll");
       }
       pw1.println(nimesisestus.getText());
       br1.readLine(); //eeldatavasti küsitakse parooli
       pw1.println(tf2.getText());
       tf2.setText("");
       new Thread(this).start();
       removeAll();
       //Aken puhtaks ning uus paigutus
       setLayout(new BorderLayout());
       Panel kesk=new Panel(new GridLayout(1, 2));
       kesk.add(ta1);
       Panel tahvlid=new Panel(new GridLayout(2, 1));
       tahvlid.add(tahvelOma);
       tahvlid.add(tahvel);
       kesk.add(tahvlid);
       add(kesk, BorderLayout.CENTER);
       Panel nupud=new Panel();
       nupud.add(kopeeriPilt);
       nupud.add(piltVorku);
       Panel alumine=new Panel(new BorderLayout());
       alumine.add(tf1, BorderLayout.CENTER);
       alumine.add(nupud, BorderLayout.EAST);
       add(alumine, BorderLayout.SOUTH);
       tf1.setText("");
       validate();
    }catch(Exception viga){
      nimesisestus.setText(viga.getMessage());
    }
  }
  if(e.getSource()==kopeeriPilt){
    tahvelOma.hoidlaUusSisu(tahvel.hoidlaKoopia());
  }
  if(e.getSource()==piltVorku){
   if(yhendusOlemas){
```

```
      Iterator it=tahvelOma.hoidlaKoopia().iterator();
      while(it.hasNext()){
        pw1.println(it.next());
      }
     } else {
       removeAll();
       algpaigutus();
       validate();
       ta1.setText("");
       nimesisestus.setText("");
       piltVorku.setLabel("Võrku");
     }
    }
  }

  /**
  *  Teateid püüdva lõime käsud. Võrgust saabuvad andmed paigutatakse
  *  vastavalt algusele kas teatena tekstialasse või saadetakse
  *  kujundina pildile.
  */
  public void run(){
   try{
    while(true){
      yhendusOlemas=true;
      String rida=br1.readLine();
      if(rida==null){
        throw new IOException("Ühenduse lõpp");
      }
      if(rida.startsWith(".p")){
        saada(rida);
      }else{
        ta1.append(rida+"\n");
      }
    }
   }catch(Exception viga){
     yhendusOlemas=false;
     piltVorku.setLabel("Alusta");
     ta1.append("Oled väljas");
   }
  }

  /**
  * Käsklus pildikujundi (edasi) saatmiseks (eeldatavalt tahvlile) juhul kui vastuvõtja on olemas.
  */
  void saada(String andmed){
    if(kuulaja==null){return;}
    kuulaja.kujund(andmed);
  }

  /**
  *  Parameetrina saadavad andmed saadetakse võrku edasi. Vajalik
  *  ka JooniseKuulari liidese realiseerimiseks.
  */
  public void kujund(String andmed){
    pw1.println(andmed);
  }


  /**
  * Käivitus. Luuakse raamaken ning paigutatakse kliendi eksemplar sellesse.
  */
  public static void main(String[] argumendid){
     Frame f=new Frame();
     f.add(new GrKlient3());
     f.setSize(500, 400);
     f.setVisible(true);
     f.addWindowListener(new Raamikuular());
  }


  /**
  * Akna sulgemine ristile vajutusel.
  */
  static class Raamikuular extends WindowAdapter{
    public void windowClosing(WindowEvent e){
      System.out.println("Programmi ots");
      System.exit(0);
    }
  }
}
```







Loetud kolme keerukustaseme juures pole läbi proovitud sootukski kõik võrgurakendustega seotud võimalused, kuid siinse aluse peale peaks andma ehitada enamiku lahendustest, mis tellijal või programmeerija mõttesse saaksid tulla. Ikka tuleb välja mõelda mõningane protokoll andmete saatmiseks ning kanali kummassegi otsa rakendus saabuvate teadete tekitamiseks ja töötlemiseks. Täiesti lubatud on andmeid saata binaarformaadis, ainult et sel juhul on liikluse kontroll veidi keerulisem. Ka liikumise või muusika kannatab võrgurakendusele külge ehitada. Võrk on lihtsalt üks täiendav vahend programmile andmete saatmiseks ja sealt vastuvõtuks.

Ülesandeid

Paigutatud kasutajatega jututuba

- Hiirevajutuse koordinaadid saadetakse jututoa serverisse kujul kasutajanimi x y
- Programmis paigutatakse ekraanile saabunud kasutajanimi sellega kaasas olevatele koordinaatidele.
- Iga kasutaja viimase hiirevajutuse asukohal on klientprogrammis näha tema nimi.
- Lisaks oma asukohale saab üle kanda ka tavalist teksti. Võrguliikluses on sellise rea ees hüüumärk.

Laevade pommitamine

- Serverprogramm mõtleb numbri ühest kümneni. Kasutaja võtab ühendust ning pakub numbri. Programm teatab, kas number pakuti õigesti või valesti.
- Serveriga saavad ühendust võtta kaks kasutajat. Nad võivad hakata teineteisele kordamööda andmeid saatma (nagu käike males või kuule laevade pommitamisel)
- Kaks kasutajat saavad graafilise liidese abil laevade pommitamist mängida. Kumbki kasutaja saab algul hiire abil oma laevade asukohad märkida. Siis saavad kasutajad näidata, kuhu hiirega lasta, ülejäänud töö teeb arvuti mängijate eest ära.

Rändurid võrgumaastikul

- Kaks kasutajat saavad võrgu kaudu saata teineteisele oma koordinaate.
- Kummagi kasutaja asukoht on näha ekraanil. Kumbki saab klahvide abil oma asukohta muuta.
- Mängu alustamisel on mõlemad kasutajad vasakul alumises nurgas. Võidab see, kes jõuab rutem ümber akna keskel paiknevate tõkete paremasse ülemisse nurka.

Liiklus võrgus

- Iga kasutaja saab oma sõiduvahendit liigutada.
- Lisaks eelmisele peavad sõidukid liikuma mööda teid ning ei tohi sattuda üksteise peale.
- Teedel on eesõigusmärgid ning valgusfoorid eesõigusi seadmas, eeskirju rikkuda ei saa.