Tallinna Pedagoogikaülikool
Informaatika osakond

Jaagup Kippar

Java põhikursuse konspekt

Tallinn 2003

Sissejuhatus

Käesolev kirjutis on mõeldud nii iseõppijatele kui abimaterjaliks õpilastele ja õpetajatele. Kasulikke soovitusi või ideid peaksid siit leidma nii algajad kui ka kogenumad koodikirjutajad. Iseõppijailt eeldatakse varasemat kasvõi väikest kokkupuudet programmeerimisega, kuid lisamaterjalide või kõrvalise abi toel peaks ka ilma ettevalmistuseta hakkama saama. Kel on soovi kohe näpuga järele proovida, kuidas midagi tööle panna ning edaspidi juba töötavat programmi oma huvide järgi kohandada ning sealtkaudu õppida, siis võib julgesti alustada näidetest. Nende juures on alustatud lühikesest programmist ning lisatud võimalisi, püüdes siiski programm jätta lühikeseks, lihtsaks ja kergesti mõistetavaks. Kes on need näited mõttega läbi lugenud, läbi proovinud, väikseid ja suuremaidki muudatusi sisse viinud ja edukalt suutnud eri programmide osi ühendada, need peaksid hakkama saama märgatava osa tavaelus tarvilike lihtsate abiprogrammide koostamisega ning samas on suutnud enesele ehitada aluse, kindlustunde ja usu endasse, millele edasisi keerulisemaid teadmisi kinnitada. Ka koolitunnis või arvutiringides on soovitav alustada lihtsatest töötavatest näidetest, millega mängides tundub programmeerimine jõukohasena. Samas pakuvad näited õpetajale küllaldaselt võimalusi toimimise tagamaade ning annab võimaluse nii nõrgematele kui tugevamatele lahendada jõukohaseid ülesandeid ning tutvuda teooriateemadega. Mõistmise sügavus lihtsalt oleneb õppimisvõimest ning pühendumusest, kuid lihtsa valmisnäite juures suudab igaüks juhendaja juuresolekul midagi muuta või täiendada.

Objektide peatükis on püütud seletada ning näidete põhjal läbi proovida objektorienteeritud programmide koostamisel kasutatavad lõigud, pannes tähtsamad teemad ettepoole ning harvem ette tulevad taha. Objektinduse mõistmisel peaks olema kergem aru saada nii programmide ülesehitusest kui kuvatavatest veateadetest. Samas siiski ei pea laskma end liialt häirida, kui esimesel lugemisel osa peensustest arusaamatuks jääb. Tasapisi programme koostades tekivad ehk seosed ning vajadusel võib ju alati loetud peatüki juurde taas tagasi tulla.

Interneti peatükis on püütud lihtsate näidete varal seletada võrguprogrammide töötamispõhimõtteid, minnes tasapisi keerukamate lahenduste juurde. Peale teema läbitöötamist peaks kasutaja mõistma võrguprogrammide ehitust ja ettetulevaid probleeme ning suutma koostada interaktiivseid klient-server rakendusi. Järgnev lõik lõimedest aitab võrguprogrammides ette tulnud teemat veidi sügavamalt mõista.

Andmete peatükki on koondatud mitmesugused andmetöötluse, -ülekande ning -struktuuridega seotud teemad. Neist kindlasti olulisim, kuid tõenäoliselt mitte liialt keeruline on Java Collections Framework - Java vahendite kogu andmehulkade hoidmiseks, sorteerimiseks ning muudeks operatsioonideks. Edasi mõned read dokumenteerimise kohta ning siis juba tuttavad näited, mida tasuks ikka ja jälle kiigata, kui tundub, et programm muidu liialt keeruliseks läheb ja kusagilt mujalt alata ei oska.

Head lugemist!

Sisukord

Tutvus Java keelega

Java ajalugu ulatub aastasse 1990, kui seda hakati firma Sun poolt välja töötama. Eesmärgiks oli luua vahend, mille abil saaksid töötada paljud uued erinevad protsessorid nii magnetofonides, telefoniaparaatides kui mujal. Seni tuli iga uue protsessoritüübi puhul luua tema jaoks uuesti peaaegu kõik programmid. Ainult vahel õnnestus teise protsessori programmi emuleerida. Et protsessoritel ning nende programmidel olid sageli sees vaid nendele spetsiifilised käsud, siis tekkis nende käskude ülekandmisel raskusi. Sooviti luua lihtne baitkood, mida oleks hõlbus üle kanda. Nii saaks üldotstarbelisi toiminguid (nagu näiteks kellaaja väljastamist) kasutada muutmata kujul kõikjal ning eraldi tuleks luua vaid masinaspetsiifilised käsud. Näiteks kerimine magnetofonil ning vee välja laskmine pesumasinal.

Kuigi uut baitkoodi on võimalik kirjutada otse või kompileerida selle saamiseks kõrgkeeles kirjutatud programme, tehes vajalikud lihtsustused ja täiendused, otsustati luua omaette programmeerimiskeel. Et uus keel ei pea ühilduma eelnevatega, siis saab kõrvale jätta aja jooksul ebaotstarbekaks või veaohtlikuks osutunud kohad. Java on suure osa põhikonstruktsioone üle võtnud keelelt C, mistõttu selle keele oskajatele võivad Java programmilõigud esialgu tuttavamad tunduda.

Ka näiteks Basicu või Perli programmi saab lasta tõlkida mitme operatsioonisüsteemi intepretaatoril, kuid Javast on nad vähemalt pikemate programmide korral aeglasemad. Programmeerija kirjutatud teksti tõlkimine masina käsujadaks on aeglasem korralikult optimeeritud baitkoodi tõlkimisest. Märgatava aja võtab intepretaatori käivitamine, samuti võib lihtsate käskude tõlkimine võtta rohkem aega kui nende täitmine. Samas aga toimuvad aeganõudvad operatsioonid sageli ajal, kui arvuti muidu nagunii kasutajapoolset teadet (näiteks klahvivajutust) ootaks. Viimistletud tervikoperatsioonid, mille teostamist kasutaja sageli ootama peab, näiteks akna avamine, võtavad Javas ainult natukene rohkem aega kui operatsioonisüsteemispetsiifilistes programmides. Lisaks on loodud vahend, mis käivitamise ajal kompileerib Java baitkoodi ümber masinkoodiks ning sel juhul töökiiruses enam tõlkimisest tingitud vahet tavaliselt ei ole.

Suur osa Java programme aeglustavatest põhjustest tuleks muus keeles korralikult koostatavasse programmi nagunii sisse kirjutada. Java kompilaator paneb nad lihtsalt automaatselt sisse. Nii tuleb ka muidu kontrollida, et kasutaja sisestatud andmed sobiksid, et programmi kood on õigesti sisse loetud, et mittevajalikud andmed enam mälu ei raiskaks, et programm operatsioonisüsteemile liiga ei teeks. Muidu saab näiteks mälu vabastamise mõnikord välja jätta lootuses, et ka koos "surnud" andmetega ei ületata programmile eraldatud mälu mahtu, ning lastes programmil selle võrra kiiremini töötada, kuid Java kontrollib ikka üle, et kusagil midagi "ripakile" pole jäänud. Samuti vaadatakse igal massiivi poole pöördumisel üle, et vastava järjenumbriga element ikka massiivi kuulub.

Nagu eespool kirjas, võttis Java suure osa põhikonstruktsioone üle keelelt C, jättes samas kasutamata vahendid, milleta läbi saab ja mis kirjutamise keerulisemaks või veaohtlikumaks teevad. Välja jäeti "null terminated string", mille vääral kasutamisel võis valedesse mälupiirkondadesse sattuda. Ka pole Javas viita mäluaadressile. Siin pole üldse standardvahenditega võimalik otse masina mäluga tegelda, mis välistab suures osas võimaluse masinat või teisi programme kahjustada. Osuti isendile aga annab paljus samad võimalused mis viit. Vaid mäluaadressi asemel on objekti number tabelis. Klassid on nii kirjetüüpide kui objektitüüpide eest. Meetodeid võib kasutaja sinna soovi korral kas lisada või mitte.

Võrguprogrammeerimine pole Java eriomadus, kuid juba keele loomisel on arvestatud võimaluse ja vajadusega luua arvutivõrgus töötavaid programme. Selle keele abil saab luua nii serveri- kui kliendiprogramme. Saab luua näiteks www-serveri brauseritele HTML-lehekülgede saatmiseks kui ka mitme paralleelkasutatega andmebaasi, jututoa või üle võrgu mängitava mängu.

Iseseisvalt arvutis jooksvad programmid suudavad kasutajale pakkuda pea samasuguseid võimalusi nagu igas muus keeles kirjutatud programmid. Vaid masinaomaste käskude puhul tuleb need mõnes muus keeles luua ning siis Java programmist käivitada. Tavakasutaja jaoks aga, kel pole tarvis kõvaketast formaatida ega masinas mälu ümber jagada, peaks Java võimalustest täiesti piisama.

Rakendid on mõeldud käivitamiseks teise programmi (näiteks veebiseiluri) sees ning nendel on peal turvapiirangud. Rakendeid võib lasta sirvijal küllalt julgelt Internetist kohale laadida ja käivitada, ilma et peaks muretsema kohaliku masina võimaliku kahjustumise pärast.

Kuna tegemist on suhteliselt uue programmeerimiskeelega, siis tema võimalusi täiendatakse pidevalt ja märgatavalt. Algselt 1995.a. paiku avaldatud versioonis oli kuus paketitäit klasse arvutamiseks, süsteemiga suhtlemiseks ja ekraanil kujutamiseks. Pideva täiendamise tõttu on ettevalmistatud võimaluste arv kümnekonna aastaga vähemalt kümnekordistunud andmebaasiühenduse, komponenttehnoloogia, turvavahendite ning laiendatud graafika- ja muusikavõimaluste abil, kuid keele kallal töötatakse edasi. Pidevalt kasvades ja arenedes on keelel muidugi oht paisuda suureks ja raskesti haaratavaks, nagu juhtus juhtivaks programmeerimiskeeleks pürginud PL/1-ga ning praegugi on Java keeles valmis meetodeid ligi kakskümmend tuhat. Kuid nii nagu igas keeles on mõnisada tuhat sõna ja hädapärased jutud suudame ajada vähem kui tuhande sõnaga, ei tasu ka programmeerimiskeele puhul

lasta end heidutada suurtest sõnade (käskude) hulgast ning lihtsamate programmide juures on võimalik hakkama saada mõne klassi ning mõneteistkümne meetodiga. Kui üldpõhimõtted on selged, saab üksikuid käsklusi alati manuaalist juurde vaadata.

Koodi maht

Objektorienteeritud Java keeles peab kogu kirjutatav kood olema meetodis (ehk funktsioonis), mis omakorda kuulub mingi objektitüüpi ehk klassi koosseisu. Lihtsas programmis saab läbi ühe meetodi ning klassiga, kuid vähegi suuremas programmis on sageli otstarbekas iseseisvad osad eraldi kirjutada. Mõnikord see küll suurendab veidi töövaeva, kuid väiksemaid lõike on lihtsam kontrollida ning tulemus on töökindlam ja vajadusel kergemini täiendatav.

Lihtsaimale java-programmile:

```
public class Tervitus{
  public static void main(String argumendid[]){
    System.out.println("Tere");
  }
}
```

vastaks pascali-programm

```
begin
 writeln('Tere');
end.
```

või Basicu/Pythoni

```
print "Tere"
```

Nagu näha, kulub alustamise ja lihtsa väljundi jaoks märgatavalt enam ruumi. Ka mõnes muus kohas võib java kood suhteliselt palju ruumi võtta. Lisaks sellele tundub kompilaator algul mitmes kohas tüütu tähenärijana, teatades kümnetest vigadest, mida ollakse sisse tippimise ajal teinud, kirjutades näiteks suure tähe asemele väikese või ajades muutuja deklareerimisel midagi segamini. Pascali programm oleks selle aja peale juba ammu tööle hakanud, kui Java juures tuleb alles trükivigadega maadelda. Juba pisut suuremate, nii paarileheküljeliste programmide juures annab tunda, et tähenärimisest on ka kasu. Liiatigi veel siis, kui tuleb hakata kokku panema ammuunustatud ning vastvalminud programmilõike.

Tutvustusnäited

Et arvuti käituks vastavalt kasutaja soovidele, tuleb talle anda korrektseid käsklusi. Arvutile käskluste andmiseks on loodud programmeerimiskeeled. Iga käsklus palub arvutil midagi teha. Kui meie näeme mõistlikult töötavat programmi, siis tegelikult täidab arvuti üksteise järele otstarbekalt kirjutatud käsklusi. Programmeerija tööks on käsud niimoodi kirja panna, et nende täitmise tulemusena arvuti kasutajale soovitut teeb.

Java keel võimaldab kirjutada mahukaid (mitme tuhande leheküljelisi) programme. Käsud pannakse "kestadesse", et oleks võimalik pikas tekstis orienteeruda. Ka lühikesel programmil peab ümber olema vähemalt kaks kesta: meetod ja klass. Kui kasutatakse varem valmistatud klasse, tuleb mõnikord kirjeldada nende klasside asukoht, et nad üles leitaks. Järgnevalt mõned näiteprogrammid koos väikeste kirjeldustega. Kui mõni lause tundub võõrana, ärgu lugeja lasku end sellest suuremat häirida. Ülejäänud peatükkides püütakse kõigele lähemalt seletust anda.

Lihtne raamaken

Tutvustuseks väike programm, mis loob ekraanile pealkirjaribaga tühja akna. Esimene rida teatab, et klass `Frame` asub paketis `java.awt`. See klass suudab ekraanile tekitada pealkirjariba, nuppude ja servadega tühja akna. Kui klassi peaks vaja minema, teab arvuti seda sealt otsida. Rida `public class Raamike` teatab, et klassi nimi on `Raamike`. Samas kataloogis paiknevaid klasse saab eristada nime järgi. Loogeline sulg tähistab klassi algust ning klass lõpeb, kui selle sulg kinni läheb.

Selles klassis on vaid üks meetod, nimega `main`, mis hakkab tööle iseseisva programmi käivitamisel. Kui klassis `main`-meetodit pole (näiteks klassis `Frame`), siis saab seda kasutada vaid mõne teise klassi kaudu. Meetodi sees olevaid käske hakkab arvuti järjestikku täitma. Esimese käsuga loome raami ning määrame talle pealkirjaks Esimene. Siis määrame suuruse ekraanipunktides ning lõpuks palume raam nähtavaks teha. Ongi kogu programm.

Esimene raam

```
import java.awt.Frame;
public class Raamike{
  public static void main(String argumendid[]){
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

Roheline raamaken

Kui on vaja ühest paketist sisse tuua mitu klassi, siis võib klasside nime asemele panna tärni. Piiritleja `public` (avalik) tähendab, et vastavat klassi (või meetodit) on võimalik kasutada ka väljastpoolt kataloogi. Nii saab vajadusel väikestest klassidest midagi suuremat kokku lappida. Käsk `setBackground` määrab raami tausta ning `setLocation` raami asukoha ekraanil.

Asukoht ja värv

```
import java.awt.*;
public class Raamike2{
  public static void main(String argumendid[]){
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setBackground(Color.green);
    f.setLocation(200, 100);
    f.setVisible(true);
  }
}
```

Liikuv raamaken

Kui tahame raami ekraanil liigutada, siis tuleb tema asukohta mõne aja tagant vahetada, nii nagu filmis vahetuvad kaadrid. Et liikumine liialt kiire ei oleks, selleks tuleb vahepeal oodata. Ootamise käsuks on `Thread.sleep` ning sulgudes olev number näitab ootamise aega milli(ehk tuhandikes)sekundites. 1000 on siis parajasti terve sekund. Meetodi `main` sulgude järel on kirjas `throws Exception,` mis näitab, et oleme teadlikud eriolukorraohtliku meetodi `Thread.sleep` kasutamisest.

Asukoha muutus

```
import java.awt.*;
public class Raamike3{
  public static void main(String argumendid[]) throws Exception{
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setVisible(true);
    Thread.sleep(1000);
    f.setLocation(200, 100);
    Thread.sleep(1000);
    f.setLocation(400, 100);
  }
}
```

Tsükli abil liikumine

Kui sooviksime kõik kohad välja kirjutada, kus raam sujuva liikumise teel asub, läheks meie programm päris pikaks. Et kirjutusvaeva ning ka programmi mahtu vähendada, tuleb luua tsükkel.

Tsüklit kasutatakse, kui tahetakse mõnda korraldust mitu korda anda. Siin on mitmel korral arvutatud raami uus asukoht ning raam seejärel sinna joonistatud. Raami asukoha väärtuste hoidmiseks võetakse kasutusele muutujad `x` ja `y`. Tsükli while sees olevaid käske korratakse seni kuni `x`-i väärtus on kolmesajast väiksem. Kui raam on jõudnud juba kolmesajanda ekraanipunktini, siis väljub programm tsüklist ning enam raam ei liigu.

Liikuv raam

```
import java.awt.*;
public class Raamike4{
  public static void main(String argumendid[]) throws Exception{
    int x=0, y=150;
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setVisible(true);
    while(x<300){
      f.setLocation(x, y);
      Thread.sleep(100);
      x=x+5;
    }
  }
}
```

Värvide koostamine

Tuntumaid värve saab ette anda konstandina (`Color.red`, `Color.blue`). Punasest, rohelisest ja sinisest osast kokku aga saab segada meelepärase värvi. Siin näites muudetakse raami tausta sinise värvi osa.

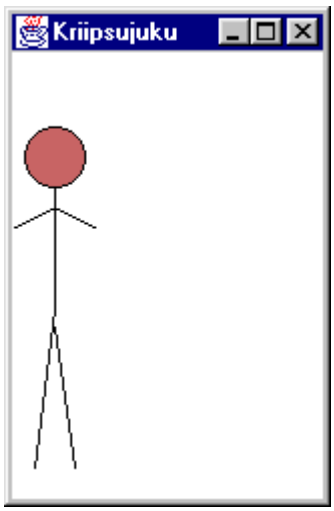
Muutuv värv

```
import java.awt.*;
public class Raamike5{
  public static void main(String argumendid[]) throws Exception{
    int punane=100, roheline=100, sinine=20;
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setBackground(new Color(punane, roheline, sinine));
    f.setVisible(true);
    Thread.sleep(1000);
    while(sinine<255){
      f.setBackground(new Color(punane, roheline, sinine));
      Thread.sleep(50);
      sinine=sinine+4;
    }
  }
}
```

Joonistamine

Klassi Graphics abil saab joonistada mitmesuguseid kujundeid. Jooni, ovaale, nelinurki ja muudki. Täpsemaid näiteid saab abiinfost (JDK API). Iseseisva programmi töö algab alati meetodist nimega `main`. `paint`-meetod kutsutakse välja, kui ekraanile on vaja joonistada. Ovaal joonistatakse ristküliku sisse ning neli koordinaati näitavadki selle ristküliku andmeid: esimesed kaks vasaku ülemise nurga asukohta, edasised laiust ja kõrgust. Kui viimased on võrdsed, siis on tegemist ringiga. Draw tähendab joone joonistamist, fill puhul värvitakse ka seest. `setColor` määrab värvi, millega edaspidi joonistatakse.

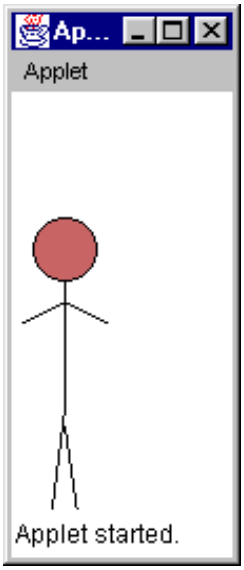
```
import java.awt.*;
import java.applet.Applet;

public class Joonis2a extends Applet{
  public void paint(Graphics g){
    g.setColor(new Color(200, 100, 100));
    g.fillOval(10, 60, 30, 30);
    g.setColor(Color.black);
    g.drawOval(10, 60, 30, 30);
    g.drawLine(25, 90, 25, 150);
    g.drawLine(25, 100, 5, 110);
    g.drawLine(25, 100, 45, 110);
    g.drawLine(25, 150, 15, 230);
    g.drawLine(25, 160, 35, 230);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Kriipsujuku");
    f.setSize(100, 250);
    f.add(new Joonis2a());
    f.setVisible(true);
  }
}
```



Veebilehel vastava programmi vaatamiseks tuleb koostada html-fail, mille sees märkida, kus ning kui suurena loodud graafilise sisuga klassi tuleks näidata.

```
<html><body>
  <h2>Joonistusharjutus</h2>
  <applet code="Joonis2a" height="200" width="100">
  </applet>
</body></html>
```

Vasakul on näha rakend käivitatuna JDK-ga kaasas tuleva appletvieweri nimelise programmi abil, paremal pool veebiseiluris.



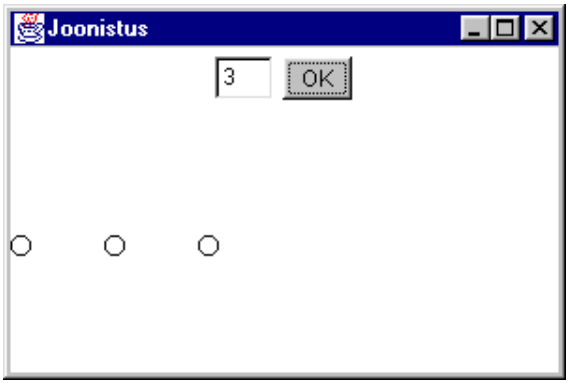
Kasutaja andmetele reageerimine

Programmilõike saab panna käivituma sündmuste peale. Sündmusteks võivad olla näiteks nupule vajutus, hiire liigutamine või teksti muutmine. Sündmusele reageerimiseks tuleb luua kuular – klassi eksemplar, milles on meetod(id) sündmusele reageerimiseks ning mis teatab oma kirjeldavas osas, et ta suudab vastavatele sündmustele reageerida.

Järgnevas näites luuakse tekstivälja ja nupuga raam. Tekstivälja kirjutatud numbrile joonistatakse ekraanile vastav arv ringe. `setLayout(new FlowLayout())` määrab paigutuse, kus elemendid saab panna järjest üksteise taha. Teade `implements ActionListener` klassi kirjelduses näitab, et meie loodud klass `Joonis4` suudab kuulata sündmusi (näiteks nupuvajutust). See kirjeldus on nagu tunnistus: et klassile saaks sellise kirjelduse panna, peab ta ka tegelikult sisaldama meetodit, mida käivitub sündmuse toimumise korral. Liidese `ActionListener` juurde kuulub meetod `actionPerformed`

nii nagu kooli matemaatikatunnistuse juurde kuuluvad läbitud õppetunnid. Meetod käivitatakse, kui sündmus toimub. Teadete saatjaid ja kuulareid võib olla mitu. Et programm teaks, millise sündmuse puhul milline kuular tuleb käivitada, tuleb kuular allika juures registreerida. Selleks on rida `nupp.addActionListener(this)`. Võtmesõna `this` tähendab tõlkes iseennast ehk praegusel juhul klassi `Joonis4` järgi loodud isendit. Nupule vajutamisel käivitatakse `Joonis4` meetod `actionPerformed`, parameetrina tuleva `ActionEvent`'i kaudu on võimalik allika kohta andmeid saada. See on tarvilik näiteks juhul, kui on võimalik valida mitme vajutatava nupu vahel. Vajutuse peale võetakse tekstiväljast tekst, muudetakse numbriks ja pannakse muutujasse `nr`. Meetod `repaint` käsib ekraani uuesti joonistada, s.t. käivitab meetodi `paint`. Selles joonistatakse ekraanile muutujas `nr` olev arv ringe.

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

public class Joonis4a extends Applet
             implements ActionListener{
  int nr=3;
  TextField tf=new TextField(""+nr);
  Button nupp=new Button(" OK ");
  public Joonis4a(){
    add(tf);
    add(nupp);
    nupp.addActionListener(this);
  }
  public void paint(Graphics g){
    for(int i=0; i<nr; i++){
      g.drawOval(50*i, 100, 10, 10);
    }
  }
  public void actionPerformed(ActionEvent e){
    nr=Integer.parseInt(tf.getText());
    repaint();
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Joonistus");
    f.add(new Joonis4a());
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```



Hiirevajutusele reageerimine

Ka hiirevajutusele oleks võimalik analoogiliselt reageerida, s.t. muuta loodav klass kuulariks ning käskida hiire teated sinna saata. Kuna hiirega on seotud palju (5) sündmusi, kuid meie soovime esialgu reageerida vaid ühele, tuleks ülejäänute kirjeldamiseks lisatööd teha. Selle asemel võib hiire teadete kuulamiseks luua `MouseAdapter`'i alamklassi. Sel juhul tuleb kirjeldada vaid vajalikke sündmusi. Ülejäänute puhul kasutatakse `MouseAdapter'i` sees paiknevaid tühje kirjeldusi. Kuna `HiireKuular` on klassi `Joonis3` sisemine klass, saab seal kasutada ka välimise klassi muutujaid (ning klassi ennast). Siin näites `Joonis3a.this.getGraphics()` annab isendi, mille abil on võimalik raami pinnale joonistada. MouseEvent annab andmed hiire kohta, nt. `e.getX()` on hiire x- ning `e.getY()` hiire y-koordinaat.

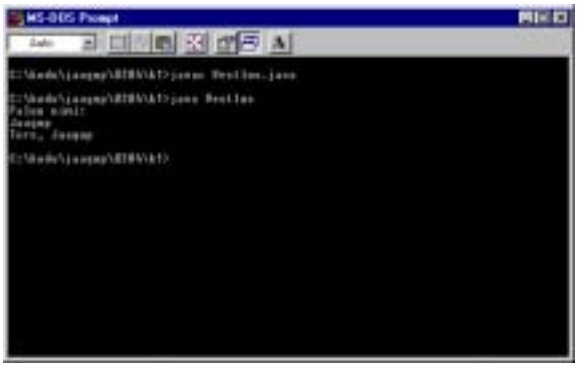
```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class Joonis3a extends Applet{
  public Joonis3a(){
    addMouseListener(new HiireKuular());
  }
  class HiireKuular extends MouseAdapter{
    public void mousePressed(MouseEvent e){
      Graphics g=Joonis3a.this.getGraphics();
      g.drawRect(e.getX(), e.getY(), 20, 10);
    }
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Vajuta hiirega");
    f.setSize(200, 200);
    f.add(new Joonis3a());
    f.setVisible(true);
  }
}
```

Vestlus tekstiekraanil

Vaid tekstiga tegelevad või arvutavad programmid ei vajagi graafilist kesta. Piisab sellest, kui kasutaja oma andmed sisse tipib ning rakendus talle mõne aja pärast vastuse väljastab.

Klaviatuurilt lugemiseks on vajalik importida pakett java.io (Input/Output). Peameetodi juures on kirjas throws IOException - samuti kui Thread.sleep on ka BufferedReaderi loomine veaohtlik käsklus ning tuleb seega kirjeldada. Peameetodi main esimese reaga loome vahendi, mille abil klaviatuurilt lugeda. Edaspidi võime readLine käsuga iga kord ühe rea inimese käest sisse lugeda. Head vestlemist!

```
import java.io.*;
public class Vestlus{
  public static void main(String[] argumendid) throws IOException{
    BufferedReader sisse=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Palun nimi:");
    String eesnimi=sisse.readLine();
    System.out.println("Tere, "+eesnimi);
  }
}
```



Tsüklid

Muutujad, tsüklid ja valikud on Javas olemas nagu Pascalis või mõnes muuski programmeerimiskeeles. Tüüpiline täisarv on `int`, reaalarv `double` ja sõne `String`. String on suure tähega, kuna ta on struktuurne andmetüüp(sisaldab mitut tähte), muud on lihttüübid(sisaldavad vaid üht väärtust). Näide:

```
public class Tyybid{
  public static void main(String argumendid[]){
    int rida=0, ridadearv;
    String ese="auto";
    ridadearv=5;
    while(rida<ridadearv){
      System.out.println(ese+"  nr "+rida);
```

```
      rida++;
    }
  }
}
```

Javas kirjutatakse kõik käsud meetodite sisse, kaasa arvatud põhiprogramm. Põhiprogrammi meetodi nimi on main ning ta saab parameetriks sõnemassiivi. Juhul, kui parameetrid puuduvad, on see massiiv tühi. Muutuja tüüp kirjutatakse muutuja esmakordsel mainimisel tema ette. Täisarvu `int` piirid on veidi enam kui +-2 miljardit (Pascali, Visual Basicu `integeri` 30000 vastu) ning võtab ruumi 32 bitti(4 baiti). Reaalarv `double` kasutab 64 bitti. Sõne pikkus on piiratud peaaegu ainult arvuti mälumahuga. Sõne on struktuurne tüüp, klassi String isend. Kui lihttüüpide (`int`, `double`) puhul asub muutujale vastaval mäluväljal väärtus, siis struktuurtüübi puhul asub seal osuti isendile. Sõne puhul näiteks sõltub sõneisendile eraldatud mälu maht pikkusest, osuti suurus jääb ikka samaks. Lihttüüpe on keelde sisse ehitatud 8 ning neid ise juurde luua ei saa. Struktuurtüüpide aluseks olevaid klasse saab ise vajadusel luua.

`while`-tsükkel töötab Javas samuti nagu Pascalis ja mitmes muuski keeles. Tsükli sisu korratakse senikaua, kuni tingimus on tõene. Väära tingimuse puhul sisu ei täideta. `for`-tsükkel Javas sarnaneb while-tsüklile, vaid mugavuse pärast on algväärtustamise ja tsüklimuutuja muutmise osad toodud sulgude sisse. Eelneva näite saaks `for`-tsükli abil kirjutada järgmiselt:

```
int ridadearv=5, rida;
String ese="auto";
for(rida=0; rida<ridadearv; rida++){
   System.out.println(ese+"  nr "+rida);
}
```

Esimene käsk (`rida=0`) täidetakse ainult üks kord tsüklisse sisenemisel. Iga korra algul kontrollitakse teisena asuvat tingimust (`rida<ridadearv`) ning kolmas käsk täidetakse pärast tsükli keha läbimist. Juhul, kui keha koosneb vaid ühest lausest, võib ka siin käsusulud ära jätta. Javas on käsusulgudeks loogelised sulud.

Täidetavad käsud on valmis olevate klasside meetodid. Näiteks `System.out.println()`, mille abil saab ekraanile trükkida, on lahtiseletatult klassi `System` juurde kuuluva trükkimisvoo nimega `out` meetod `println()`.

Valik

Programmi saab hargnema panna `if`-valikuga, kus tingimuse järel olev valik täidetakse vaid juhul, kui tingimus on tõene. Soovi korral saab lisada ka `else`-osa vastasel juhul toimimiseks.

```
public class ValikIf{
  public static void main(String argumendid[]){
    int marivanus=13;
    int jukuvanus=14;
    if(marivanus<jukuvanus){
      System.out.println("Mari on noorem kui Juku");
    }
  }
}
```

```
D:\Kasutajad\jaagup\java>java ValikIf
Mari on noorem kui Juku
```

```
public class ValikIf2{
  public static void main(String argumendid[]){
    int marivanus=13;
    int jukuvanus=14;
    if(marivanus<jukuvanus){
      System.out.println("Mari on noorem kui Juku");
    } else {
      System.out.println("Mari pole noorem kui Juku");
    }
  }
}
```

```
D:\Kasutajad\jaagup\java>java ValikIf2
Mari on noorem kui Juku
```

Sõne

Sõne hoidmiseks, uurimiseks ning nende võrdlemiseks kasutatakse klassi String.

```
public class Sone1{
  public static void main(String argumendid[]){
    String nimi="Juhan";
    System.out.println("Nimi="+nimi+" Pikkus="+nimi.length());
    System.out.println("h asub kohal "+nimi.indexOf("h"));
    System.out.println("Teine ja kolmas täht on "+
                       nimi.substring(2, 4)+"\n"+
                       "Lugemine algab nullist");
    if(nimi.equals("Juhan"))
      System.out.println("Nimi on endiselt Juhan");
    else
      System.out.println("Nimi pole Juhan");
  }
}
```

```
C:\kodu\jaagup\0204\k1>java Sone1
Nimi=Juhan Pikkus=5
h asub kohal 2
Teine ja kolmas tõht on ha
Lugemine algab nullist
Nimi on endiselt Juhan
```

Meetodi käivitamiseks kirjutatakse objekti nime ning tema meetodi vahele punkt.
Tähtsamad sõne juures kasutatavad meetodid:

```
meetodi nimi                  väljastab
length                       pikkuse
indexOf                      koha, kust alamsõne algas. Kui
                               ei leidu, siis väljastab -1
equals                       tõeväärtusena (boolean tõene/väär),
                               kas sõnede väärtused võrduvad
substring                    alamsõne. Kui meetodile antakse üks
                               täisarvuline parameeter siis väljastatakse
                               alamsõne alates määratud kohast kuni lõpuni.
                               Kui antakse kaks parameetrit, siis esimene
                               näitab algust (kaasaarvatud) ning teine lõppu
                               (väljaarvatud).
```

Meetodite täpsemad ingliskeelsed kirjeldused leiab dokumentatsioonist.

Sõne osadeks jagamisel aitab paketi java.util klass StringTokenizer. Vaikimisi tükeldab see iga sõna eraldi tükiks, kuid selle abil on võimalik ka näiteks pikk tekst lauseteks jagada, kusjuures lause eraldajaks on punkt, küsimärk ja hüüumärk. Allolevas näites trükitakse lausest välja vaid i-ga lõppevad sõnad.

```
import java.util.StringTokenizer;
public class Lauseuuring{
  public static void main(String[] argumendid){
    String lause="Juku tuli kooli";
    StringTokenizer tykeldaja=new StringTokenizer(lause);
    System.out.println("Lause tehti "+tykeldaja.countTokens()+". osaks.");
    System.out.println("i-ga loppevad:");
    while(tykeldaja.hasMoreTokens()){
      String sona=tykeldaja.nextToken();
      if(sona.endsWith("i")){
       System.out.println(sona);
      }
    }
  }
}
```

```
C:\kodu\jaagup\0204\k1>java Lauseuuring
```

```
Lause tehti 3. osaks.
i-ga loppevad:
tuli
kooli
```

Arvutamine

Tähtsamad matemaatikafunktsioonid asuvad klassis Math. Nagu mujalgi, nii ka siin tuleb meetodi välja kutsumiseks määrata meetodi omanik (siin klass Math) ning siis meetodi nimi.

```
public class Arvutus1{
  public static void main(String argumendid[]){
    double x=Math.PI/6;
    double siinus=Math.sin(x);
    System.out.println("Nurk x="+x+" sin(x)="+siinus);
    System.out.println(" cos(x)="+Math.cos(x)+
                       " tan(x)="+Math.tan(x));
    x=Math.random();
    int a=(int)(5*Math.random());
    double kuup=Math.pow(x, 3);
    System.out.println("x="+x+" a="+a+" x^3="+kuup);
  }
}
```

Meetod `Math.random()` väljastab juhusliku reaalarvu nulli ja ühe vahelt. Kui soovitakse täisarvulist juhuarvu, siis saab ühe võimalusena korrutada saadud reaalarv arvuga, kui suures vahemikus tahetakse juhuarvu saada ning siis võtta täisosa, nagu siin näites on tehtud. Uus tüüp on omistamisel vaja ette kirjutada juhul, kui tüübimuundamisega võib andmeid kaduma minna. Näiteks

```
int n;
n=(int)4.7;
```

annab muutuja `n` väärtuseks nelja.

```
n=4.7
```

aga kutsub esile veateate.

Tüübimuundamine on enam tähtis objektide ja pärimise juures.

Sisend käsurealt

Küllalt mugav on programmi tööks vajalikud andmed ette anda otse käivitamisel, kirjutades need programmi nime taha. Selliseid andmeid nimetatakse käsurea parameetriteks. Kõik nõnda kirjutatud sõnad on võimalik programmis kätte saada `main`-meetodile antavast sõnemassiivist. Iga etteantud sõna või lihtsalt tühikutega eraldatud sümbol pannakse sellesse elementide kogusse omaette isendina. Kui on massiivi nimeks on `argumendid`, siis argumendid.length tähistab seal paiknevate elementide arvu, `argumendid[0]` algelementi, `argumendid[1]` järgmist ning nõnda edasi. Kõik saabuvad andmed on tekstidena, vajadusel peame neid mõnele muule kujule muundama. Kui soovime etteantud sümbolite jada arvulist tähendust, siis `Integer.parseInt` püüab etteantud teksti arvuks tõlkida.

```
public class Tekstikorrutaja{
  public static void main(String[] argumendid){
    if(argumendid.length!=2){
      System.out.println(
          "Programm kordab kasutaja pakutud sona etteantud arv kordi. Kasuta:");
      System.out.println("java Tekstikorrutaja tekst arv");
      return; //katkestab meetodi täitmise
    }
    String sona=argumendid[0];
    int kordadearv=Integer.parseInt(argumendid[1]);
    for(int nr=0; nr<kordadearv; nr++){
      System.out.println(sona);
    }
  }
}
```

```
C:\kodu\jaagup\0204\k1>java Tekstikorrutaja
```

```
Programm kordab kasutaja pakutud sona etteantud arv kordi. Kasuta:
java Tekstikorrutaja tekst arv

C:\kodu\jaagup\0204\k1>java Tekstikorrutaja Tere 5
Tere
Tere
Tere
Tere
Tere
```

Alamprogramm

Kümnekonnast reast pikemate programmide puhul leidub ikka terviklikke toiminguid, mida on võimalik ning sageli ka vajalik ülejäänud programmist eraldada. Selline liigendamine aitab korraga kontrollimist vajavad osad muuta väiksemaks ning vead nende seest kergemini leitavaks. Samuti kasutatakse alamprogrammideks jagamist juhul, kui sama toimingut on tarvis välja kutsuda ülejäänud programmi mitmes kohas.

Alamprogrammi välja kutsudes saab talle soovi korral anda ette andmed. Alljärgnevas näites antakse ette täisarvuline arv. Alamprogramm võib soovi korral väljastada väärtuse. Siin näites väljastatakse samuti täisarv. Alamprogrammis etteantavate ja väljastatavate tüüpide kohta Java keeles piiranguid ei ole. Ette anda võib väärtusi piiramata arvu; väljastada tohib aga ainult ühe väärtuse. Kui etteantavaid väärtusi on mitu, eraldatakse nad komaga.

```
public class Alamprogramm{
  public static int liidaJuurde(int arv){
    int vastus=arv+1;
    return vastus;
  }
  public static void main(String argumendid[]){
    int nr=7;
    int tulemus=liidaJuurde(nr);
    System.out.println(tulemus);
  }
}

D:\arhiiv\naited\keel\muu>java Alamprogramm
8
```

Lihttüübid

`byte` - 8-bitine täisarv vahemikus -128 kuni 127.
`short` - 16-bitine täisarv -32768 kuni 32767.
`int` - 32-bitine täisarv -2147483648 kuni 2147483647.
`long` - 64-bitine täisarv -9223372036854775808 kuni 9223372036854775807.
`float` - 32-bitine reaalarv ligikaudses vahemikus -3,4x10^38 kuni 3,4x10^38 seitsme tüvekohaga.
`double` - 64-bitine ujukomaarv (reaalarv) ligikaudses vahemikus -1,7x10^308 kuni 1,7x10^308 15 tüvekohaga.
`char` - 16-bitine Unicode sümbol. Näit. 'a', '\n',
`boolean` - tõeväärtustüüp võimalike väärtustega true ja false.

Need kaheksa on "lihtsad" andmetüübid ilma riugasteta. Kui sinna tüüpi muutujasse midagi kirjutada, siis pole karta, et temaga "iseenesest" midagi juhtuks. Neid saab kasutada sarnaselt, nagu muutujaid Pascalis või mõnes muuski keeles. Tüüpiliselt kasutatavaks täisarvuks on `int` ning reaalarvuks `double`. Tüüp `byte` on ühebaidine nagu üks bait kettalgi. Tüüp `char` on javas 16-bitine (Pascali ja C 8 vastu) ning tal on üle 64 tuhande võimaliku väärtuse. Enamik Euroopa keeli saab väikeste mööndustega 256 tähega hakkama, mida 8 bitti pakuvad, kuid kuna ka hieroglüüfe sisaldavate ja muid paljutäheliste keelte edasiandmiseks on lihtsam kasutada igale sümbolile ühte tähte, siis ei pea imenippe rakendama tähtede surumiseks sinna, kuhu need ei mahu. Tähed kirjutatakse ühekordsete ülakomade vahele nagu näiteks 'm'. "m" on kompilaatori jaoks juba enam mitte lihttüüp `char`, vaid struktuurne tüüp String. Enamasti ongi programmides lihtsam kasutada sõnet kui tähte.

Lihttüüpide nimed kirjutatakse väikese tähega, struktuurtüüpide omad soovitatavalt suurega. Siis on kirjutamise käigus hea eristada, millega on tegu.

Arvusüsteemid

Täisarve saab lisaks kümnendsüsteemile kirjutada ka kaheksand- ja kuueteistkümnendsüsteemis. Kaheksandsüsteemis arvule tuleb ette kirjutada 0, kuueteistkümnendsüsteemis arvule 0x. Nii et 12, 014 ja 0xC on Java kompilaatori jaoks üks ja seesama asi. Enamasti piisab kasutamisel kümnendsüsteemist, kuid pildi joonistamisel kaheksat värvi kasutades on hea pildi punkte masinale kaheksandsüsteemis ette kirjutada või mälust lugeda.

Täisarvud loeb kompilaator automaatselt olevaks tüübist `int`. Kui soovime talle rõhutada, et tegu on nimelt tüübist `short`, siis tuleb tüübi nimi arvule sulgudes ette kirjutada nt. `(short)12` . Sama lugu ka reaalarvudega, kus automaatselt arvab kompilaator punkti esinemise korral olevat tegemist kahekordse täpsusega ujukomaarvuga (`double`). Tüüpi long saab rõhutada, lisades arvu lõppu tähe `l` nt. `12345l, float` puhul võib lõppu lisada tähe `f`.

Abiinfo

Java keele kasutamisel on abiks sõnastik ehk API spetsifikatsioon. Nagu (võõr)keele juures pole lootust kõiki sõnu ja väljendeid pähe õppida, nii pole ka programmeerimiskeele juures vaja sellega liigselt vaeva näha. Mis kulub pähe see kulub, kuid unustamise puhuks on alati manuaal olemas, tuleb vaid osata seda kasutada. Kui 1995 aastal välja tulnud Java-versioonis olid ametlikud kuus paketti kümnete klasside ning tuhatkonna meetodiga, siis 2002. aasta algul on SUN välja lasknud juba 50 paketti ligi tuhande klassi ning kahekümne tuhande meetodiga. Vaevalt nende loetelu võtab mitusada lehekülge, rääkimata lähematest kirjeldustest. Enamikus lühemates programmides saab läbi kuni kümne klassi ning saja meetodiga, kuid keerulisemate olukordade puhul on võimalus sõnastikust abi otsida. Hierarhia ja viidete abil on sealt täiesti lootust midagi leida.

Seletan siinkohal lahti manuaali seletused kahe meetodi kohta paketist java.lang.Math. Kopeeritult näevad nad välja järgmised:

```
static double  log(double a)
    Returns the natural logarithm (base e) of a double value.

static double  max(double a, double b)
    Returns the greater of two double values.
```

Esimese meetodi nimi on `log` ning ta saab omale parameetriks `double` tüüpi reaalarvu. (Et muutuja nimeks on `a`, see ei muuda kasutamisel midagi.) Meetod väljastab väärtuse samuti tüübist `double`. `Static` tähendab, et meetod kuulub klassi (mitte isendi) juurde. Sellest lähemalt edaspidi. Seletus juures teatab, et (meetod) tagastab naturaallogaritmi (alusel e) (parameetrina antud) `double` tüüpi väärtusest. Kasutada saab seda näiteks

```
double x=Math.log(3.5);
```

Siinkirjeldatud meetod `max` väljastab samuti reaalarvu (`double`), väljastades suurema etteantud parameetritest. Klassi `String` isendimeetod (`static` puudub) `length()` väljastab sõne pikkuse täisarvuna.

```
        int length()
             Returns the length of this string.
```

Struktuursed andmetüübid

Massiiv

Hulga ühetüübiliste andmete tarvis vajatakse massiive.

```
public class Massiiv1{
  public static void main(String argumendid[]){
    int[] ruudud = new int[5];
    for(int i=0; i<5; i++){
      ruudud[i]=i*i;
    }
    System.out.println("Arvu 2 ruut on "+ruudud[2]);
  }
}
```

Massiivi elemendid hakkavad lugema numbrist 0. Korraldus `new int[5]` loob viieelemendilise täisarvumassiivi, mille esimeseks elemendiks on element järjenumbriga 0 ning

viimase elemendi järjenumbriks on 4. Esimeses kümnes võib selline lähenemine paista harjumatuna, kuid alustades nullist, algab järgmine kümme arvust 10 (mitte 11). Tegemist on sarnase probleemiga, et kas uut aastatuhandet hakata lugema aastast 2000 või 2001. Java (ning ka C ja mõnes muuski) keeles loetakse numbreid alates nullist nagu sündinud lapsel, kes saab aastaseks alles pärast ühe aasta möödumist sünnist.

Massiivi elemendid võib ka massiivi loomisel algväärtustada.

```
public class Massiiv2{
  public static void main(String argumendid[]){
    String nimed[]={"Juku", "Kati", "Siim"};
    System.out.println("Nimekirja alguses on "+nimed[0]);
    System.out.println("Kogu nimekiri koosneb nimedest:");
    for(int nr=0; nr<nimed.length; nr++){
      System.out.println(nr+1+". "+nimed[nr]);
    }
  }
}
```

```
D:\Kasutajad\jaagup\java>java Massiiv2
Nimekirja alguses on Juku
Kogu nimekiri koosneb nimedest:
1. Juku
2. Kati
3. Siim
```

Massiivi juurde kuuluv väli `length` näitab massiivi elementide arvu. Siinses näites oli elementide arvuks kolm. Massiivi tunnus võib kirjeldamisel olla nii muutuja tüübi kui nime taga. `String nimed[]` ning `String[] nimed` tähendavad sama.

Massiivid võivad olla ka mitmemõõtmelised. Näiteks annab nii meeles pidada õpilaste paiknemise klassiruumis. Tuleb algul öelda, mitme rea ning mitme veeru jagu andmeid tarvis hoida on ning edaspidi saabki nendele kohtadele väärtused paigutada. Järgnevas näites võib ette kujutada kolme lauda, kusjuures keskmine (number 1) on tüdrukute oma.

```
public class Massiiv3{
  public static void main(String argumendid[]){
     String[][] klass=new String[3][2];
     klass[0][0]="Juku";
     klass[0][1]="Mati";
     klass[1][0]="Kati";
     klass[1][1]="Killu";
     klass[2][0]="Siim";
     klass[2][1]="Sass";

     System.out.println(klass[1][0]);
  }
}

/*
  Istekohad:

  Juku   Mati
  Kati   Killu
  Siim   Sass
*/
```

```
D:\Kasutajad\jaagup\java>java Massiiv3
Kati
```

Nii nagu ühe mõõtme puhul, nii ka siin saab massiivile anda sulgudes ette algväärtused. Et kõik elemendid saaks läbi käia, tuleb kaks tsüklit üksteise sisse panna. Muutuja rida näitab, et mitmenda rea peal massiivis ollakse. Kui soovime kõikide elementidega midagi ette võtta (näiteks ekraanile kirjutada), siis tuleb iga rea peal kõik veerud läbi käia. See on sisemise tsükli ülesanne. Iseenesest ei pruugi igas reas sugugi ühepalju inimesi istuda. Seetõttu kontrollitakse nimede kirjutamisel, palju vastavas reas veerge on (`klass[rida].length`) ning senikaua jätkatakse selle rea veergude läbimist.

```
public class Massiiv4{
  public static void main(String argumendid[]){
     String[][] klass={
       {"Juku", "Mati"},
```

```
      {"Kati", "Killu"},
      {"Siim", "Sass"}
    };
    for(int rida=0; rida<klass.length; rida++){
      for(int veerg=0; veerg<klass[rida].length; veerg++){
        System.out.print(klass[rida][veerg]+" ");
      }
      System.out.println();
    }
  }
}
```

```
D:\Kasutajad\jaagup\java>java Massiiv4
Juku Mati
Kati Killu
Siim Sass
```

Massiivi mõõtmeid võib olla ka julgesti enam kui kaks. Näiteks, kui laos on konteinerid ridade ja veergudena üle põranda ning lisaks sellele veel mitmes kihis, siis võib massiivis esimene number tähendada rea, teine veeru ning kolmas kihi numbrit. Täisarvulise elemendi väärtust võib ette kujutada kui konteineris oleva kauba kilode arvu. Samuti pole võimatu ka neljamõõtmeline massiiv: neljas number võib näiteks tähendada päeva koodi, mil vastava konteineri mass on mõõdetud.

Järgnevas näites on koostatud korrutustabel, kus kirjas kolme arvu korrutised. Kui massiiv on valmis tehtud, siis võib sealt hakata tehetele vastuseid pärima. Nagu näha, on 3*4*2 vastuseks 24.

```
public class Massiiv5{
  public static void main(String[] argumendid){
    int pikkus=10, laius=7, korgus=12;
    int[][][] korrutised=new int[pikkus][laius][korgus];
    for(int x=0; x<pikkus; x++){
      for(int y=0; y<laius; y++){
        for(int z=0; z<laius; z++){
          korrutised[x][y][z]=x*y*z;
        }
      }
    }
    System.out.println(korrutised[3][4][2]);
  }
}
```

```
D:\Kasutajad\jaagup\java>java Massiiv5
24
```

Omakoostatud tüüp

Kaheksa lihttüüpi on keelde sisse ehitatud, programmeerija neid muuta ei saa. Vajadusel saab lihttüüpidest koostada struktuurse andmetüübi. Kui lihttüüpide võimalused meid ei rahulda, siis struktuurtüübis saab neid omavahel kombineerida, et luua oma soovidele vastav tüüp. Näide:

```
class Punkt{
  int x, y;
}
```

Niimoodi vaid kirjeldame tüübi. Loodud tüübi omaduste kasutamiseks tuleb sellest luua vähemalt üks isend.

```
public class Punktid1{
  public static void main(String argumendid[]){
    Punkt a=new Punkt();
    a.x=5;
    a.y=3;
    System.out.println("a="+a+" a.x="+a.x+" a.y="+a.y);
  }
}
```

annab oma töö tulemuseks rea

```
a=Punkt@1fa4d40b a.x=5 a.y=3
```

Nagu näha, peitub meie jaoks mõistlik info Punkti a väljadel x ja y, a enese väljatrükkimisel näeme vaid tema räsikoodi. Struktuurse tüübi muutuja näitab kohale, kust leida tema välju. See on sarnane Pascali ja C viidatüüpi muutujale, mille väärtuseks oli mäluaadress. Kuna Java-programm töötab virtuaalmasinas ning ta pole otseselt seotud arvuti enese füüsilise mäluga, siis käib ka tehniline pool teisiti. Struktuurse tüübi muutujat nimetatakse osutitüüpi muutujaks ehk osutiks. Java keeles saab nii väärtus- kui osutimuutjaid kasutada ilma probleemideta ka meetodite parameetritena ning tagastusväärtustena.

Näite juures võib imelikuna tunduda rida `Punkt a=new Punkt();` mille juures luuakse uus isend tüübist `Punkt` ning pannakse talle osutama muutuja `a`. Kui kirjutaksime vaid `Punkt a;` , siis kirjeldatakse ainult muutuja `a` ilma tema jaoks mälu eraldamata. Sõna `Punkt` kaks korda kirjutamine võib tunduda iiasus, kuid ei ole. Edaspidi selgub, et juhul, kui oleme kirjeldanud `PuutePunkti` erinevused Punktist, siis on tähendus ka lausel

```
Punkt ristmik=new PuutePunkt(asfalttee, kruusatee);
```

Konstruktor

Loodud isendile aitab väärtusi sisestada konstruktor ehk alamprogramm, mis käivitub vaid üks kord ning seda isendi loomise algul.

```
class Punkt2{
  int x, y;
  public Punkt2(int uus_x, int uus_y){
    x=uus_x;
    y=uus_y;
  }
}
```

Konstruktoril peab olema klassiga sama nimi. Ta käivitatakse isendi loomisel käsu `new` abil. Piiritleja `public` näitab, et konstruktorit on võimalik käivitada ka väljastpoolt klassi.

```
public class Punktid2{
  public static void main(String argumendid[]){
    Punkt2 a=new Punkt2(5, 3);
    Punkt2 b=new Punkt2(1, 1);
    System.out.println(b.x-a.x);
  }
}
```

annab käivitamisel vastuseks -4 (ehk 1-5).

Nagu ennist kirjas oli, "sisaldavad" vaid lihttüüpi muutujad neisse pandud väärtusi. Ülejäänud muutujad osutavad vastavat tüüpi isendile (või ei osuta kuhugi, sellisel juhul on selle muutuja väärtuseks `null` (sõnana)). Järgnevas näites pannakse kaks osutit osutama ühele isendile.

```
public class Punktid3{
  public static void main(String argumendid[]){
    Punkt2 a=new Punkt2(5, 3);
    Punkt2 b=new Punkt2(1, 1);
    a=b; // ka a hakkab osutama b jaoks loodud kohale
    b.x=7;
    System.out.println(a.x+"  "+a.y);
  }
}
```

väljastab a väljade väärtusteks 7 ja 1.

Esialgu luuakse kaks isendit. Ühe väljade väärtusteks saavad 5 ja 3, teisele 1 ja 1. Esimesele pannakse osutama `a`, teisele `b`. Siis pannakse ka `a` osutama teisele isendile, esimene isend jääb sootuks tähelepanuta ning ta koristatakse mälust. Teise isendi väljad on endiselt 1 ja 1. Kui nüüd kirjutatakse `b.x=7`, siis pannakse teise isendi x-väljale 7. Kuna ka `a` viitab nüüd teisele isendile, siis kirjutatakse välja 7 ja 1.

Et saaks omistamisega andmeid kopeerida nii nagu lihttüüpide korral, selleks tuleb kõik lihttüüpide väärtused eraldi kopeerida või siis luua selle tarbeks vastav meetod.

Kokkuvõte

Java baitkood koosneb lihtsalt erinevatele protsessoritele tõlgitavatest käskudest ning vajab käivitamiseks intepretaatorit. Java programmeerimiskeele loomisel on võetud aluseks keel C (ja C++),

püüdes sealsetest kogemustest õppides muuta Java keel kergemini õpitavaks ning vead programmi seest kergemini leitavaks.

Programmi käskudeks on valmis olevate klasside meetodid. Käskude jada moodustab meetodi, meetodid klassi. Programmi käivitamisel asutakse täitma meetodit nimega main.

Tsükleid kasutatakse tegevuste kordamiseks. `while`-tsükli sisu täidetakse senikaua kui tingimus on tõene. `for`-tsüklis on lisatud koht eelväärtustamiseks enne tsüklisse sisenemist ning tsüklimuutuja väärtuse muutmiseks pärast tsükli sisu läbimist, kuna need tööd tulevad tsüklite puhul sageli ette. Käsusulgudeks on loogelised sulud {}.

If-valikut täidetakse juhul, kui tingimus on tõene. Vajadusel võib lisada ka else-osa.

Sõne pikkusel on piiriks vaid mälumaht. Ta on objekt erinevalt lihttüüpidena olevast täis- ning reaalarvust. Temalt saab küsida tema omadusi (nt. pikkust, alamsõnet) meetodi käivitamise teel. Põhilised arvutusoperatsioonid paiknevad klassis Math. Vanu võimalusi meelde tuletada ning uusi leida saab API spetsifikatsioonist.

Massiivis hoitakse ja töödeldakse suuremat hulka sarnast tüüpi andmeid. Kaheksa lihttüüpi on Java keelde sisse ehitatud ning neid kasutaja muuta ei saa. Soovi korral saab nende abil struktuurtüüpe koostada.

Ülesandeid

Püüa loetu põhjal võrrelda Java keelt nende programmeerimiskeeltega, mis enesele tuttavamad on. Leia eeliseid ja puudusi.

Raamiga aken

- Loo ekraanile raam
- Liiguta raami ekraanil vasakult paremale
- Liiguta raami ülalt alla ja tagasi.
- Pane raam vasakule-paremale pendeldama.
- Suurenda ning vähenda raami laiust.
- Alusta väikesest raamist ekraani üleval vasakul nurgas. Pane raam kasvamaüle ekraani. Jäta parem alumine nurk paigale ning vähenda raam paremasse alla nurka pisikeseks tagasi.

Aknad

- Ava korraga kaks akent.
- Määra ühe akna taust roheliseks, teise oma punaseks.
- Pane üks aken liikuma paremalt vasakule ning teine vasakult paremale.
- Näita akna pealkirjaribal koordinaate.
- Aken liigub üle ekraani vasakult paremale. Kuni keskkohani akna kõrgus suureneb, edasi hakkab vähenema.
- Aken liigub üle ekraani vasakult paremale. Kuni keskkohani akna liikumisekiirus suureneb, edasi hakkab vähenema.
- Aken liigub üle ekraani paraboolikujulist trajektoori pidi.

Aken ja käsurida

- Käsurealt saadud sõna paigutatakse akna pealkirjaks.
- Lisaks eelmisele koosneb pealkiri kahest sõnast
- Pealkirjaribale paigutatakse kahe käsureale kirjutatud arvu summa.
- Akna kaugus vasakust servast määratakse käsurealt
- Käsurealt teatatakse akna asukoht ning suurus.
- Luuakse käsurealt määratud arv aknaid.

Juhuarvud

- Väljasta akna pealkirjaribale juhuslik arv.
- Määra akna kõrguseks juhuslik arv.
- Loo juhusliku asukoha ja suurusega aken.
- Loo viis juhusliku asukoha ja suurusega akent.
- Loo juhuslik arv aknaid.
- Määra akna taustaks juhuslik värv.
- Määra akna taustaks juhuslik halltoon.
- Määra akna taustaks juhusliku heledusega sinine.
- Määra akna pealkirjaks juhuslikult üks käsurea parameeter.

Maja joonis

- Joonista raamile maja.
- Lase kasutajal sisestada maja korruste ning trepikodade arv ning joonista nende andmete põhjal maja.
- Arvesta maja joonistamisel raami kõrgust ja laiust (vastavad suurused annavad `getHeight()` ja `getWidth()`)

Hiiremäng

- Hiirega vajutamise kohale joonistatakse ristkülik
- Hiirevajutuse tulemusena hüppab ristkülik suvalisse kohta
- Hiirega ristküliku tabamisel hüppab viimane suvalisse kohta.
- Tekstiväljades loetakse, mitu tabamust on pihta, mitu mööda läinud.
- Kasutajal on võimalik valida, kas tal tuleb püüda ruutu või ringi.

Arvutaja

- Koosta programm kahe kasutaja sisestatud arvu korrutamiseks.
- Luba lisaks valida, millist tehet sooritada.

Arvamismäng

- Arvuti mõtleb juhusliku arvu. Testiks teata arv.
- Teata, kas kasutaja pakutav arv ühtib sellega, on suurem või väiksem.
- Luba pakkuda sinikaua, kuni pihta saadakse.
- Loetakse kokku, mitmendal korral õige vastus saadi.
- Massiivis hoitakse meeles kasutaja pakutud vastused. Mängu lõppedes teatatakse pakkumised.

Objektid programmeerimisel

"Arvutimaailm lendab lõhki! Ja suurimaks probleemiks pole mitte viirused. Pole ka häkkerid ja kräkkerid. Suurimaks probleemideks programmides on vead!"

Suurelt jaolt vigade leidmise tarvis hakatigi otsima programmeerimisse uusi vahendeid. Objektorienteeritud programmeerimine on viimase paarikümne aasta jooksul aidanud vigu leida ning tal on ka muid kasulikke omadusi.

Suuremate projektide korral aitab objektideks jagamine tõsta abstraktsiooni taset ning valmistada ja kontrollida iseseisvaid programmi osi eraldi. Samuti nagu alamprogrammide loomine aitab orienteeruda suures käskude jadas, aitab objektide loomine orienteeruda alamprogrammide rägastikus.

Valmis programmilõike saab korduvalt kasutada vaid siis, kui on võimalik nad arhiivist üles

leida. Ka siin on kasu kõrgemast abstraktsioonitasemest.

Mõnikord on lihtsalt mugav reaalset elu jäljendavaid programme panna kokku objektidest, sest ka tegelikus elus on meil tegemist suhteliselt iseseisvate üksustega (auto, inimene ...).

Java programmis peab olema kogu kirjutatav kood meetodi sees. Meetod omakorda klassi ehk objektitüübi sees. Klassid omakorda moodustavad paketi.

Lihtsama programmi puhul

```
public class Tervitus{
  public static void main(String argumendid[]){
    System.out.println("Tere");
  }
}
```

on koodiks `System.out.println("Tere");`
meetodiks `main` ning klassiks `Tervitus`.

Lühikeste programmide korral saab kogu programmi teksti kirjutada `main`- meetodi sisse ning objektitüüp tema ümber ei ole millekski kasulik. Kui aga programmis tuleb luua uus tüüp ning sellest tüübist isendiga suhtlema hakata, siis saab programmi teksti muuta tavakeelele lähemaks ning isendeid kergemini eraldi kontrollida.

```
class Punkt4{
  int x, y;
  public Punkt4(){
    this(0, 0);
  }
  public Punkt4(int uus_x, int uus_y){
    x=uus_x;
    y=uus_y;
  }
  public String kirjutaAndmed(){
    return "x="+x+" y="+y;
  }
  public double kaugusNullist(){
    return Math.sqrt(Math.pow(x, 2)+Math.pow(y, 2));
  }
}

public class Punktid4{
  public static void main(String argumendid[]){
    Punkt4 p=new Punkt4(3, 4);
    System.out.println(p.kirjutaAndmed());
    System.out.println("Kaugus koordinaatide alguspunktist= "+
                       p.kaugusNullist());
  }
}
```

```
D:\Kasutajad\jaagup\java>java Punktid4
x=3 y=4
Kaugus koordinaatide alguspunktist= 5.0
```

Kas või selle näite koostamisel oli klassideks jagamine abiks. Kirjutasin programmi valmis ning panin käima. Selgus, et käivitamisel pakkus ta `x` ja `y` väärtuste 3 ja 4 juures kauguse nullist 2.2 juures. Kuna peaprogrammis on vaid palve saata oma kaugus, siis seal polnud arvutusviga teha võimalik. Seetõttu nägin, et punkt "käitub imelikult" ning teadsin kohe tema juurest viga otsima minna. Selgus, et olin ruutjuure lihtsalt kaks korda võtnud.

Pärimine

Kui soovida klassile `Punkt4` omadusi lisada, siis pole selleks vaja terve klassi koodi uuesti ümber kopeerida. Saab luua laiendatud võimalustega klassi, mis pärib klassist `Punkt4` tema omadused ning lisaks saab talle meetodeid juurde kirjutada.

```
class Punkt4Laiend extends Punkt4{
  public void liiguParemale(){
    x++;
  }
```

```
}


class Punktid4a{
  public static void main(String argumendid[]){
    Punkt4Laiend p=new Punkt4Laiend();
    System.out.println(p.kirjutaAndmed());
    p.liiguParemale();
    System.out.println(p.kirjutaAndmed());
  }
}
```

```
D:\Kasutajad\jaagup\java>java  Punktid4a
x=0 y=0
x=1 y=0
```

Lisaks ümberkirjutamise vaeva vähenemisele näitab selline toimimine ka paremini välja, millise klassi isend mida oskab ning millised oskused tal võrreldes eelnevaga on juurde tulnud. Kuna sama koodi on vaid kord, siis muutmise vajaduse puhul tuleb muutus sisse viia samuti vaid ühes kohas. Kui soovin muuta väljatrükki ilmekamaks, siis piisab, kui muudan klassi `Punkt4` meetodi `kirjutaAndmed()`. Uut koodi kasutatakse siis ka klassi `Punkt4Laiend` isendi andmete välja kirjutamisel. Samuti piisab ka vea leidmisel vaid ühes kohas parandamisest.

Kuna isend klassist `Punkt4Laiend` oskab teha kõike, mida klassi `Punkt4` isend, siis teda saab kasutada kõikjal, kus `Punkt4`-gi. Analoogia tavaelust võiks olla, et näiteks lendur suudab kõike, mis tavaline inimenegi (süüa, juua, magada, kellaaega öelda jne.), kuid lisaks sellele suudab ta veel lennukit juhtida. Kellaaega võime küsida ükskõik millise inimese käest, kuid lennukijuhtimist saame paluda vaid lendurilt. Seetõttu on omistamine inimene=lendur ehk `Punkt4=Punkt4Laiend` täiesti lubatud ja võime levinumat tegevust (nt. `kirjutaAndmed()`) küsida `Punkt4` tüüpi muutuja käest teadmata, et tegemist on tegelikult isendiga tüübist Punkt4Laiend sarnaselt nagu võime kella küsida vastutulevalt inimeselt, teadmata, et vastutulija on ametilt lendur. Kui aga on vaja paluda ülemklassile ainuomast tegevust, siis tuleb enne veenduda, et meil kasutada olev isend ka selleks võimeline on. Samuti nagu lennujuhti vajades teeme kindlaks, et rooliasuja tõepoolest ikka lendur on. Et ülemklassi meetod välja kutsuda, tuleb muutujas peituva isendi tüüp muundada vastavaks. Kui kirjutaksime `(Punkt4Laiend)p.liiguParemale()`, siis saaks kompilaator aru, et tuleks kõigepealt muutuja p poolt osutatud isendil paluda paremale liikuda ning seejärel saadud tulemus muundada tüübiks `Punkt4Laiend`. Sel juhul kompilaator vaatab, et `Punkt4`-l palutakse paremale liikuda, mida ta ei oska ning tekib veateade. Kui aga lisada veel ühed sulud, siis on tulemuseks `((Punkt4Laiend)p).liiguParemale()` ehk muutujas `p` olnud isend `Punkt4Laiend`'iks muundatuna ning temal juba on võimalik lasta paremale liikuda. Juhul, kui muutujas `p` siiski aga pole `Punkt4Laiend`i tüüpi isendit, tekib viga. Samuti nagu inimesel, kes pole lendur, pole võimalik lenduritunnistust näidata. Kui tüüp vastab oodatule, on kõik korras.

```
class Punktid4b{
  public static void main(String argumendid[]){
    Punkt4 p=new Punkt4Laiend();
    System.out.println(p.kirjutaAndmed());
    ((Punkt4Laiend)p).liiguParemale();
    System.out.println(p.kirjutaAndmed());
  }
}
```

```
D:\Kasutajad\jaagup\java>java Punktid4b
x=0 y=0
x=1 y=0
```

Üksikjuhul paistab säärasest muundamisest vähe kasu olema, kuid sellise omistamisvõimaluse tõttu saab ühest allikast põlvnevate isendite andmeid ühes massiivis koos hoida. Lisaks saab neilt paluda seda, mida nad kõik oskavad ning kui oleme kindlaks teinud, et vastaval isendil rohkem oskusi on, saame temalt ka nende rakendamist küsida. Nii võime inimeste andmed üheskoos hoida, sõltumata sellest, kas mõned neist on lendurid või mitte. Kui oleme aga inimese käest teada saanud, et ta lendur on, siis saame tal paluda lennuki rooli asuda. Operaator `instanceof` kontrollib, kas kontrollitav isend suudab vastavale klassile seatud ülesandeid täita, s.t. kas ta on sellest klassist või tema alamklassist andes vastuseks true või false.

```
class Punktid4c{
  public static void main(String argumendid[]){
    int pArv=3;
```

```
    Punkt4 pd[] = new Punkt4[pArv];
    pd[0]=new Punkt4(2, 1);
    pd[1]=new Punkt4Laiend();
    pd[2]=new Punkt4(5, 5);
    for(int nr=0; nr<pArv; nr++){
      if(pd[nr] instanceof Punkt4Laiend)
        ((Punkt4Laiend)pd[nr]).liiguParemale();
    }
    for(int nr=0; nr<pArv; nr++){
      System.out.println(pd[nr].kirjutaAndmed());
    }

  }
}
```

```
D:\Kasutajad\jaagup\java>java Punktid4c
x=2 y=1
x=1 y=0
x=5 y=5
```

Kuna kõik klassid on ühtlasi klassi `Object` alamklassid (et see on üldine, siis võib kirjutamata jätta), siis saab kõiki struktuurtüüpide andmeid panna kokku objektimassiivi.

Ülekate

Pärimisel saab luua meetodeid juurde. Samuti on võimalik panna alamklassis ülemklassi meetod teisiti toimima. Selleks tuleb lihtsalt kirjutada samanimeline ning samade parameetritega meetod.

```
class Inimene{
  int vanus;
  public void ytleVanus(){
     System.out.println("Mu vanus on "+vanus);
  }
}

class Daam extends Inimene{
  public void ytleVanus(){
    System.out.println("Sain just "+(vanus-5)+" aastat vanaks.");
  }
}

public class Tutvustus{
  public static void main(String argumendid[]){
    Inimene naine1=new Inimene();
    Inimene naine2=new Daam();
    naine1.vanus=40;
    naine2.vanus=40;
    naine1.ytleVanus();
    naine2.ytleVanus();
  }
}
```

```
D:\Kasutajad\jaagup\java>java Tutvustus
Mu vanus on 40
Sain just 35 aastat vanaks.
```

Liidesed

Iga klassi isend suudab käituda nii selle klassi kui ka tema ülemklasside jaoks loodud olukordades. Nagu siin näiteks on iga Daam samal ajal ka Inimene, sest Daam pärineb inimesest. Ning samal ajal on nii Inimene kui Daam ka Object, sest Java hierarhias on juhul, kui ülemklassi pole määratud, ülemklassiks vaikimisi Object. Kui aga soovime, et meie loodava klassi isend sobiks peale oma ülemklasside ka muudesse kategooriatesse, tuleb kasutada liideseid.

```
interface Viisakas{
  public void tervita(String partner);
}
```

Kui loodav klass realiseerib liidest `Viisakas`, peab temas olema meetod `tervita`, mis saab

parameetriks sõne. Liidese realiseerimine annab klassile kohustuse osata liideses ette antud tegevusi (ehk meetodeid). Samas annab aga selle klassi isendile asuda igal pool, kus on lubatud olla vastavat liidest realiseerival klassil.

```
class Laps extends Inimene implements Viisakas{
  public void tervita(String tuttav){
    System.out.println("Tere "+tuttav);
  }
}

class Koer implements Viisakas{
  public void tervita(String nimi){
     System.out.println("Auh!");
  }
}

public class Tutvustus2{
  public static void main(String argumendid[]){
    Laps juku=new Laps();
    juku.vanus=6;
    juku.ytleVanus();
    Viisakas kylaline1=juku;
    Viisakas kylaline2=new Koer();
    kylaline1.tervita("vanaema");
    kylaline2.tervita("Juku vanaema");
  }
}
```

```
D:\Kasutajad\jaagup\java>java Tutvustus2
Mu vanus on 6
Tere vanaema
Auh!
```

Mõnes programmeerimiskeeles (näiteks C++) eraldi liideseid ei ole. Seal lubatakse pärida korraga mitmest ülemklassist ning loodud klassi isend on samuti võimeline paiknema nii ühe kui teise ülemklassi jaoks eraldatud kohas. Kuna aga võib juhtuda, et mõlemal ülemklassil on sama nimega meetod, siis sellest võib tekkida probleeme. Selleks on javas loodud liidesed, mida realiseerides tekib vaid kohustus liideses kirjeldatud tegevusi osata. Oskusi eneseid ehk programmikoodi saab aga pärida vaid ülemklassidest. Kui on vaja mõnele klassile lisada teise klassi oskusi, siis on võimalik klassi üheks andmeväljaks võtta teise klassi isend, kellel paluda soovitud tegevusi sooritada. Ülemklasse saab olla igal klassil vaid üks, liideseid aga võib realiseerida iga klass piiramatul hulgal.

Piiritlejad

Objektorienteeritud programmides püütakse objekti kasutaja eest talle mittevajalikud tunnused ära peita, et need teda ei häiriks ning et ta ka ei saaks neid muuta ning sellega objekti toimimist häirida. Vaikimisi on võimalik nii klasse, meetodeid kui muutujaid kasutada vaid sama paketi (kataloogi) seest. Piiritleja `public` tähendab, et ligi pääseb ka väljastpoolt, `private` lubab kasutada vaid sama klassi piires ning `protected` lubab lisaks oma klassile ka alamklassides. Seda ka juhul, kui alamklassid juhtuvad teises paketis olema; `final` tähendab, et enam ei lubata muuta. Kui muutujal on ees piiritleja `final`, siis tema väärtus omistamisjärgselt enam muutuda ei saa, s.t. on konstant. Struktuurtüübi muutuja jääb `final` piiritleja korral kuni oma eluea lõpuni osutama samale isendile. Kui meetod on `final`, siis ei saa teda üle katta. Kui klass on `final`, siis ei saa talle luua alamklasse. Võtmesõna `static` meetodi või muutuja ees tähendab, et meetod või muutuja kuulub klassile, mitte isendile. Tema kasutamiseks pole vaja luua eraldi isendit. Staatilisi meetodeid ei saa üle katta. Kui harilikud muutujaid ehk väljad luuakse iga isendi jaoks uued (nt. kolmel isendil tüübist `Inimene` on igaühel oma `vanus`), siis staatilist muutujat on vaid üks (näiteks muutuja klassi `Inimene` juures, mis näitab inimeste arvu). Sellisel muutujal on väärtus siis, kui ühtki inimest pole loodud. Sellisel juhul on vastava muutuja väärtuseks 0. Samuti on tal üks ja konkreetne väärtus juhul kui inimesi on rohkem.

Lisaoskused

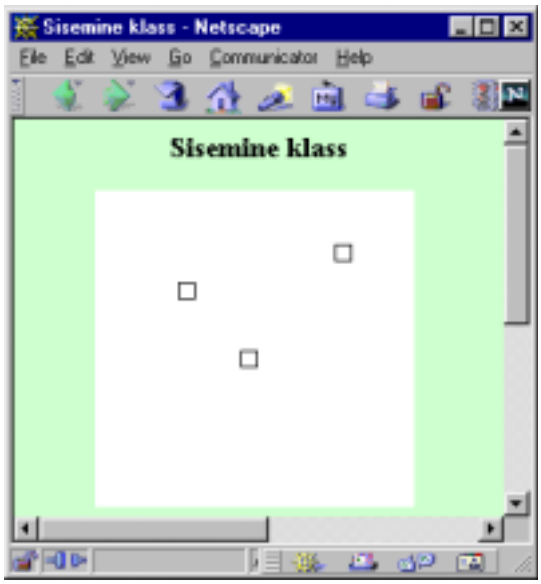
Java keeles tohib olla igal klassil vaid üks ülemklass. Kui tahta oma loodud klassis kasutada muude klasside oskusi, siis on mõistlik oma klassi luua muutuja, mille abil pöörduda vastava klassi isendi poole. Näiteks kui soovin, et rakendi pinnale tekiks tekstiväli, siis lasen rakendil lihtsalt luua isendi klassist TextField ning paigutada viimase oma pinnale. Märkimisväärne hulk olukordi õnnestub niimoodi lahendada, kus puuduvate võimaluste hankimiseks kasutatakse muude klasside isendeid.

Loodud isendile saab ligi ja tunnuseid pärida, sest meie loodud klassis paiknev muutuja osutab sinna. Nii on näiteks kerge rakendist tekstivälja sisu muuta või pärida. Samas on raske panna tekstivälja muutuse peale rakendil midagi toimuma, sest loodud tekstiväljal me rakendi kohta andmed puuduvad. Et saaks tekstiväljalt rakendile teateid saata, selleks peab esimesel teisele ligipääsuks osuti olema. Kui kirjutame rakendis `tf.addActionListener(this),` sel juhul annamegi loodud tekstiväljale võimaluse rakendile ligi pääseda ning tema meetodeid käivitada. Muul juhul peaks rakend pidevalt iga natukese aja tagant tekstivälja kontrollima, et kas seal midagi muutunud on ning siis seepeale muutuse avastama ning reageerima. Lisaks saab kõikjalt kasutada teiste klasside poolt välja pakutud staatilisi meetodeid, selleks pole isegi vastava klassi isendit vaja luua.

Sisemine klass

Sisemised klassid võimaldavad ka teisiti klassi võimalusi laiendada. Sisemisel klassil on kasutada lähteklassi eksemplari kõik muutujad ja meetodid, lisaks sellele veel need, mis sisemises klassis eneses juurde kirjeldatud on. Kuna sisemised klassid võivad olla samaaegselt ka mõne muu klassi alamklassid, siis nõnda saab ühendada kahe klassi omadused. Järgnevas näites kuulub rakendiklassi `SisemiseKlassigaRakend` sisse klass `SisemineKlass`. Hiirevajutustele reageerimiseks luuakse `init`-meetodis sisemisest klassist isend ning pannakse ta rakendile saabuvaid hiire teateid kuulama. Kuna sisemine klass pääseb ligi rakendi meetoditele ja muutujatele, saab ta ka ekraanile joonistada. Joonistusvahendi küsimiseks tuleb siiski kirjutada `SisemiseKlassigaRakend.this.getGraphics()`, sest lihtsalt `this` tähendaks sisemise klassi seest kutsutuna hoopis sisemise klassi eksemplari, kelle valduses aga ekraani pole, millele joonistada.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class SisemiseKlassigaRakend extends Applet{
  class SisemineKlass extends MouseAdapter{
    public void mousePressed(MouseEvent e){
      SisemiseKlassigaRakend.this.getGraphics().
        drawRect(e.getX(), e.getY(), 10, 10);
    }
  }
  public void init(){
   addMouseListener(new SisemineKlass());
  }
}
```



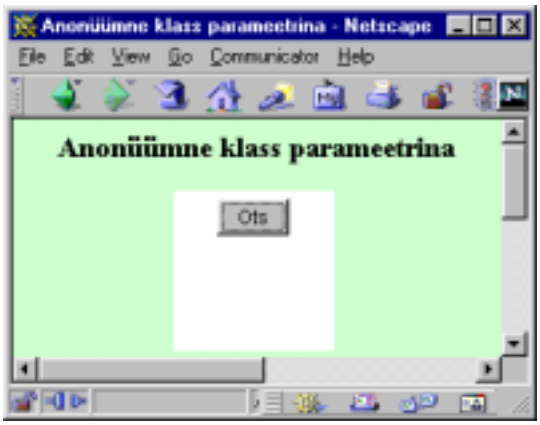
Sisemise klassi saab luua ka isendi loomise ajal.

```
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;
public class AnonyymneKlass extends Applet{
  final Button b=new Button("Algus");
```

```
  ActionListener anonyymneKuular=new ActionListener(){
    public void actionPerformed(ActionEvent e){
      b.setLabel("Ots");
    }
  };

  public void init(){
    add(b);
    b.addActionListener(anonyymneKuular);
  }
}
```

Veel lühem võimalus on toodud allpool. Siin luuakse sündmusekuularit realiseeriv isend otse `addActionListeneri` sulgude sees.

```
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;
public class AnonyymneKlass2 extends Applet{
  Button b=new Button("Algus");
  public void init(){
    add(b);
    b.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        b.setLabel("Ots");
      }
    });
  }
}
```



Arengusuunad

Arvutusvõimsuste ja programmimahtude kasvades on programmide arengut pidurdama jäänud programmeerija tööaeg ning programmides leiduvad vead. Loodeti, et tupikust päästab objektitehnika, kuid ka see ei osutunud imevahendiks. Nii nagu protseduurid aitasid liigendada jadaprogramme, aitasid objektid süstematiseerida protseduure. Kui aga klasse saab palju, on nendega samuti raske toime tulla. Javas on loodud lisatasemeks paketid, kuhu koondatakse sarnaste omadustega klassid. See aitab sobivat klassi leida. Samuti saavad ühes paketis paiknevad klassid lubada vastastikku üksteise omadusi kasutada.

Järgmiseks lahenduseks on pakutud komponenttehnoloogia. Selle järgi pannakse programm kokku kasutamisvalmis tükkidest. Näiteks võivad tükkideks olla kalender, liikuv auto, draiverite komplekt. Programmeerijal tuleb nad sobivalt ühendada ning loota, et tulemusena valmib kasutajasõbralik programm. Ühte komponenti võivad kuuluvad klassid eri pakettidest, samuti võib võrguprogrammide puhul üks komponent asetseda laiali mitmes masinas. Komponente saab luua mitmetes programmeerimiskeeltes. Nii on Java komponentideks oad (Bean), Microsoftil ActiveX komponendid. Et nad omavahel suhelda saaksid, on kirjeldatud sillad, millisel kujul komponendid andmeid vahetada saavad.

Mõningaid Java abil loodud komponente nimetatakse ubadeks (bean). Et neid võiks julgelt kõikjal kasutada, on neil samasugused turvapiirangud nagu rakenditelegi. Vabalt saab omale laadida BDK, mille abil saab hiirega komponente kokku lohistada ning rakendiks muuta. Seal saab kasutada ka neid komponente, mille loomisel pole BDK võimalusi silmas peetud, kuid järgides

komponentidevahelise suhtlemise reegleid on võimalik oma komponenti teistega paremini suhtlema panna.

Kokkuvõte.

Objektorienteeritud programmeerimine loodi vigade paremaks avastamiseks, suurte programmide lihtsamaks kirjutamiseks, programmikoodi korduvkasutuseks ning reaalse maailma objektide paremaks kirjeldamiseks.

Enne isendi loomist peavad olema tema omadused ja oskused kirjeldatud klassis muutujate ja meetodite abil. Klassi isendi oskusi saab suurendada, luues klassile alamklassi. Alamklassis saab meetodeid juurde kirjeldada, samuti kasutada ülemklassi meetodeid. Kui soovida ülemklassi meetodi tööd muuta, tuleb alamklassis kirjeldada sama nime ja parameetritega, kuid alamklassi jaoks vajaliku koodiga meetod. Alamklasside isendit saab kasutada kõikjal, kuhu sobib ülemklassi isend, sest alamklasside omad mõistavad kõiki neid tegevusi sooritada, mis ülemklassis kirjeldatud on. Nad kas pärivad need oskused muutumatuna, või muudavad nende sisu meetodi ülekatmise abil. Kui soovida, et loodava klassi isend saaks tegutseda ka mujal kui oma ülemklassile ettenähtud kohas, tuleb kasutada liidest. Liidese realiseerimiseks peavad loodavad klassis olema kirjeldatud kõik oskused, mis liideses kirjas on.

Piiritlejate `public`, `protected` ja `private` abil saab määrata ligipääsetavust. Piiritleja `static` näitab, et temaga märgitu kuulub klassi, mitte isendi juurde.

Lihtandmetüüpideks Java-s on `byte`, `short`, `int`, `long`, `float`, `double`, `char` ja `boolean`. Nendest saab kombineerida struktuurtüüpe. Lihttüübi muutujas on väärtus, struktuurtüübi omas osuti isendile. Struktuurtüübist muutuja tühiväärusek on `null`. Nii liht- kui struktuurtüüpi andmeid saab paigutada
massiivi.

Ülesandeid

Objektid

- Loo klass Inimene väljadega eesnimi ja synniaasta
- Koosta klassist kaks eksemplari, anna väljadele väärtused
- Koosta inimeste andmetest massiiv. Trüki välja inimesed, kes on sündinud varem kui 1980.

- Koosta Inimese alamklass Kodanik. Lisa väli "perekonnanimi"
- Koosta massiiv, kus on nii Inimesed kui Kodanikud
- Trüki massiivist välja kõik eesnimed, kodanikel ka perekonnanimed.

- Loo Kodanikule konstruktor andmete sisestamiseks.
- Lisa kodanikule ka sünniaastata konstruktor. Sel juhul
- tuleb sünniaasta väärtuseks -1.
- Lisa ka Inimesele konstruktor andmete sisestamiseks.
- Kutsu Kodaniku Konstruktorist välja Inimese konstruktor.

Klassid ja isendid

- Loo klass Loom väljaga "nimi" ning meetodid "tutvusta" ning "nimepikkus".
- Tee loomale alamklassid "Koer" ja "Lehm", kus tuleb katta üle meetod "tutvusta" nii, et selles väljastataks ka looma liik ning looma nime pikkus. Koosta kestprogramm loodud klasside katsetamiseks.
- Lisa klassile Loom staatiline väli loomadeArv, mille väärtus suureneb iga looma (ka koera ja lehma) lisamisel (st klassi Loom konstruktori käivitamisel). Muuda klassi Loom meetodit "tutvusta" nii, et see väljastaks loomade arvu. Kutsu klassi "Koer" meetodist "tutvusta" välja klassi "Loom" meetod "tutvusta".
- Muuda konstruktoreid nii, et isendi loomisel saaks talle koheselt ka nime omistada. Loo staatiline meetod loomade arvu väljastamiseks ning käivita see ilma isendit kasutamata.

Punktid

- Loo klass Punkt väljadega x ja y.
- Loo sellest klassist kaks isendit ja anna nende väljadele väärtused.
- Trüki mõlema punkti väljade väärtused ekraanile.

- Kopeeri ühe punkti andmed teise punkti andmeteks.
- Paiguta loodud kaks punkti massiivi.
- Loo massiiv saja punkti ning juhuslike andmetega.
- Määra viiekümnenda punkti x-koordinaadi väärtuseks 243.
- Kirjuta välja punkti andmed, mille x-koordinaadi väärtus on vähim.

Erindid, veatöötlus

Iga vähegi suurema programmi juures tekib mittestandardseid olukordi, mis tuleb programmi sujuva töö jätkumiseks lahendada. Näiteks sisestab kasutaja sobimatu numbri või pole kettal piisavalt ruumi vaheandmete salvestamiseks. Üksikjuhtudel võib probleemi lahenduse kirjutada kahtlase olukorra juurde ning tillukeste programmide juures on see täiesti kasutatav. Kui aga sarnaseid veaohtlikke kohti on palju, siis tuleb sellise lähenemise juures ka kontrollimehhanisme palju kirjutada. See võib omakorda programmi kohmakaks, väheülevaatlikuks ning raskesti muudetavaks teha. Ka on olukord keerulisem, kui vastavalt olukorrale tuleks samale eksitusele erinevalt reageerida. Näiteks sünniaasta ekslikul sisestamisel võib kasutaja tähelepanu sellele juhtida ning aastat kohe uuesti küsida. Kui aga parooli sisestamisel eksitakse, siis tuleks enne uut sisestusluba mõnda aega oodata, et poleks võimalik lühikese ajaga suurt hulka paroole läbi proovida. Samuti võib juhtuda, et veale on kasulik reageerida selle tekkimise kohast koodis küllalt kaugel. Traditsiooniliselt on sellistest vigadest teada andmiseks kasutatud muutujaid, mis informeerivad paranduslõiku vajaliku paranduse iseloomust või saadavad vea kirjelduse mingit teed pidi tema jaoks loodud töötlemise koha juurde.

Java keelde loodi ootamatustega toime tulemiseks omaette mehhanism aitamaks probleemiinfot transportida lahenduskohani. Probleemi tekkimisel saab luua ning "lendu lasta" probleemiinfot kandva isendi. Sellest kohast alates programmi täitmine katkestatakse. Kui programmilõigu ümber on pandud neid teateid töötlev püünis, siis jätkatakse programmi täitmist püünisele järgnevalt käsurealt. Püünise puudumisel lendab probleemisend programmist välja, jättes kogu teel töö tegemata. Lihtprogrammide puhul tähendab see programmi töö lõppu. Kui erind aga tekkis näiteks rakendi ülejoonistamisel meetodis paint(), siis jääb see joonistuskord lõpetamata. Näide.

Probleem tüübimuundel:

```
public class Erind1{
  public static void main(String argumendid[]){
    String s="aabits";
    int nr=Integer.parseInt(s);
    System.out.println(nr);
  }
}
```

annab töö tulemuseks:

```
C:\TEMP\java>java Erind1
Exception in thread "main" java.lang.NumberFormatException: aabits
        at java.lang.Integer.parseInt(Integer.java:409)
        at java.lang.Integer.parseInt(Integer.java:458)
        at Erind1.main(Erind1.java:4)
```

Lahtiseletatult tähendab arvuti poolt tulnud vastus seda, et lõimes nimega main tekkis erind java.lang.NumberFormatException, kuna sõna aabits ei sobi arvuks. Allpool on näha, et viga tekkis klassi Integer meetodis parseInt failis Integer.java koodireal nr. 409, see meetod oli omakorda välja kutsutud samast failist realt 458 ning see omakorda minu loodud klassi Erind main-meetodis, real nr. 4.

Kuna sõna "aabits" pole võimalik kümnendsüsteemis arvuks muundada, siis kohas, kus seda üritatakse (`Integer.parseInt(s)`) lastakse lendu erind. Kuna siin näites pole erindipüünist, siis jäävad read alates tekkinud probleemist täitmata ning numbri väärtust välja ei trükita.

Järgneva näite

```
import java.io.*;
public class Erind2{
  public static void main(String argumendid[]){
    String s="aabits";
    try{
      int nr=Integer.parseInt(s);
      System.out.println(nr);
    } catch(NumberFormatException ex){
      System.out.println("Vigane number");
    }
    System.out.println("Programmi ots");
  }
```

```
}
```

Käivitamisel paistab:

```
C:\TEMP\java>java Erind2
Vigane number
Programmi ots
```

Siinses näites satub tekkinud erind püünisesse (`try{ ... } catch ...`) ning töödeldakse. Kasutajale teatatakse, et number on vigane. Kui erind on kinni püütud, jätkub programmi töö oma tavalist rada pidi järjekorras mööda käsklauseid.

Järgnevas näites palutakse kasutajal senikaua numbrit sisestada, kuni ta sellega arvuti jaoks loetavalt hakkama saab.

```
import java.io.*;
public class Erind3{
  public static void main(String argumendid[]){
    boolean korrata=true;
    while(korrata){
      try{
        BufferedReader sisse=new BufferedReader(
          new InputStreamReader(System.in)
        );
        System.out.println("Palun number:");
        String rida=sisse.readLine();
        int nr=Integer.parseInt(rida);
        System.out.println("Number "+nr+" sisestati korralikult.");
        korrata=false;
      } catch (IOException sisendierind){
        System.out.println("Probleem klaviatuuriga");
        korrata=false;
      } catch (NumberFormatException numbriformaadierind){
        System.out.println("Valesti sisestatud number. "+
            numbriformaadierind.getMessage()+
            "\n Proovi veel!");
      }
    }
  }
}
```

Eeltoodud näites võib erind tekkida nii klaviatuurilt lugemisel kui sisestatud sõne numbriks muundamisel. Katsendilause `try{ ... }` kogub tekkinud erindid kokku ning seejärel saab neid püünistes töötlema hakata. Iga püünis püüab kinni erindi sellest tüübist, mis on kirjutatud püünise parameetriks, või tema alatüübist. Muud liiki erindi püüdmiseks peab olema talle vastav püünis.

Erindiklasside hierarhia

Erindiklassid on samuti hierarhilised nagu muudki klassid Javas. Kõige üldisem on ehk kõige kõrgemal asub klass `Throwable` (kuid seegi on `Objecti` alamklass nagu muudki), tema alamklassideks on `Error` ning `Exception`. Esimene enamjaolt parandamatute vigade jaoks, teine eriolukordade jaoks, mida on lootust lahendada. Erindi suurematest alamklassides on `IOException` ning `RuntimeException`. Esimese alla kuuluvad kõik sisendi-väljundi (Input-Output) erindid. `RuntimeException`i alla kuuluvad erindid, mille tekkimise võimalusi on programmis palju, peaaegu igas lauses, kus omistatakse või arvutatakse midagi. Näiteks ruutjuur negatiivsest arvust annab erindi tüübist `ArithmeticException` Kui viietähelisest sõnast püütakse leida seitsmendat tähte, siis tekib `IndexOutOfBoundsException`. Need mõlemad on `RuntimeException`i alamklassid. Kui muud eriolukorra tekkimise võimalused tuleb alati katsendibloki abil kinni püüda või kirjutada meetodi päisesse teade, et meetodist võib väljuda vastav erind, siis klassi `RuntimeException` või ükskõik millist tema alamklassi erindit püüda pole kohustuslik. Selline kohustus lihtsalt muudaks programmi kohmakamaks. Vajadusel saab neid aga püüda nagu muidki erindeid nagu eelpool toodud näiteprogrammides Erind1 ja Erind2.

Kui tahta ühe `catch`-püünisega püüda kinni näiteks nii massiivi rajaerindit (`ArrayIndexOutOfBoundsException`) kui ka tüübimuunduserindit (`ClassCastException`), siis piisab kui püünise parameetriks kirjutada `RuntimeException`. Kuna mõlemad esimesed on viimase alaliigid, siis sobivad nad `RuntimeException`i kohale samuti nagu inimese kohale sobis nii vanaema, lendur kui insener. Kui kirjutame püüniseks `catch(Exception erind)`, siis jäävad sinna kinni kõik erindid ning `catch(Throwable probleem)` püüab kinni kõik erindid ja vead. Kui tahame, et mõne erindiklassi puhul töödeldaks selle isendit ühtemoodi, kõiki muid erindeid aga teisiti, siis tuleb spetsiifilisem püünis ettepoole panna. Näiteks:

```
try{
```

```
  lisaMassiivi(); //varem valmis olev meetod
} catch (ArrayIndexOutOfBoundsException  me){
  System.out.println("Massiiv täis");
} catch(Exception e){
  System.out.println("Eriolukord: "+e.getMessage());
  e.printStackTrace();
}
```

Erindi meetod `printStackTrace()` kirjutab välja, millises alamprogrammis (või ka omakorda alamprogrammi alamprogrammis) probleem tekkis.

Järgnevalt valik sagedamini kasutust leidvatest vea- ning erindiklassidest koos hierarhilise struktuuri ning kommentaaridega.

Klass	Seletus
Throwable	Igasugune probleem
Exception	Erind ehk parandatav eriolukord
ClassNotFoundException	Programmi ühte klassi ei leita
IOException	Sisendi-väljundi erind
FileNotFoundException	Faili ei leita
SocketException	Interneti ühenduse erind
RuntimeException	Igasugu erind, mida ei pea töötlema
ArithmeticException	Arvutuserind (nt. jagamine nulliga)
ClassCastException	Tüübimuunduserind
IllegalArgumentException	Lubamatu argument
NumberFormatException	Vigane numbriformaat
IndexOutOfBoundsException	Rajaerind (nt. masiivi olematu element)
NullPointerException	Muutuja ei osuta isendile, kuigi peaks
NoSuchElementException	Otsitud elementi ei leita
Error	Tõsisem viga
LinkageError	Viga programmi käivitamisel
ClassFormatError	Klassifail vigane
VerifyError	Programmi sees lubamatud operatsioonid
VirtualMachineError	Viga intepreteerimisel
OutOfMemoryError	Mälu otsas

Erindeid saab programmi töö käigus ka ise lendu lasta. Siis saab eriolukorra teate saata töötleva püüniseni ilma selle jaoks muid vahendeid kasutamata. Näiteks:

```
if(nr>100) throw new ArithmeticException("Liiga suur number");
```

Erindi konstruktori parameetriks antud tekst kuulub loodud erindi juurde kogu tema "eluajaks" ning selle teksti sisu saab kätte erindile saadetava teatega `getMessage()`. Seda sisu saab vajaduse korral töötlemise juures arvestada.

Nagu eelnevalt kirjas, peab peale `RuntimeException`i alla kuuluvate erinditüüpide kõikidele muudele erindi tekkevõimalustele tähelepanu pöörama. Niimoodi kompilaator ühtlasi hoolitseb, et kahtlased programmilõigud ei jääks tähelepanuta. Nende ümber peab olema katsendiblokk koos püünisega, kuhu vastavat tüüpi erind sobib, või peab meetod lubama vastavat tüüpi erindi enesest välja:

```
public void trykiKriipse(int hulk) throws Exception{
  if(hulk<0)throw new Exception("negatiivne arv");
  else for(int i=0; i<hulk; i++)System.out.print("-");
}
```

Sel juhul tuleb loodud erindiga tegelda siinset meetodit välja kutsuvas meetodis.

Soovi või vajaduse korral saab ka ise luua uusi erinditüüpe, s.t. klassi `Exception` alamklasse.

Kokkuvõte

Erindid aitavad probleemiteavet transportida tekkekohast töötlemiskohta. Erindid püüab kokku try{} katsendiblokk ning töödelda saab neid catch-püünistes. Töötlema ei pea vaid `RuntimeException`i ning tema alamklasside erindeid.

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();
    }
  }
}
```

`InputStream`i (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 `Writeri` (näiteks `FileWriteri`) sisse. Sõne saab sellest kätte meetodiga `toString()`. `StringReaderi` 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 `PrintWriteri` 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();
    }
  }
}
```

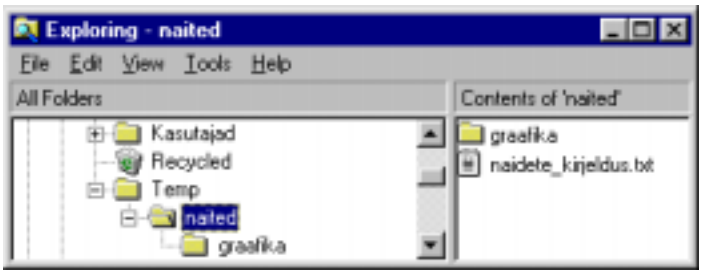
Alamkataloogide andmed

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.

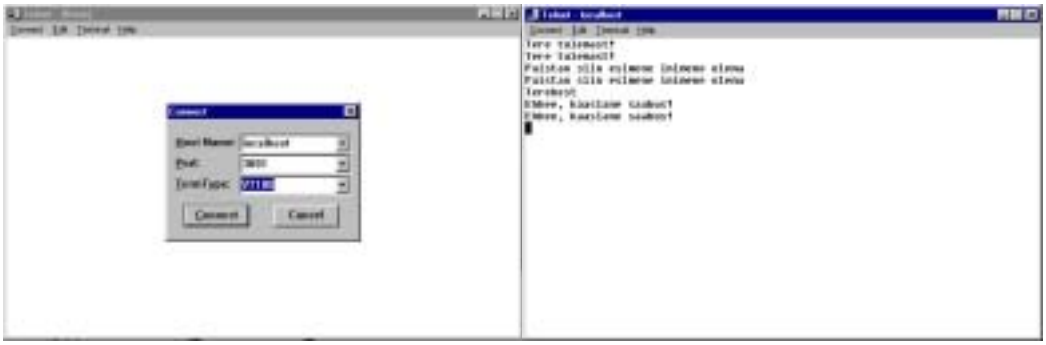
`try-catch` püünis run meetodi sees töötleb tekkivaid eriolukordi, praegusel juhul lihtsalt trükib välja sõna "probleem" ning seejärel kirjelduse, mis erindiga kaasa anti. Erind tekib näiteks juhul, kui ühendus ära kaob. Kinni püütakse erindid sellepärast, et nad programmi tööd ei hakkaks segama. Kuna 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

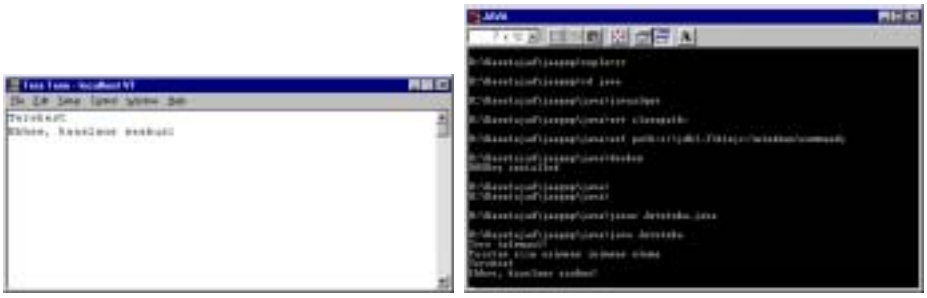
Järgnev programm on väike Telneti-jututuba:

```
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.

Andmehaldusvahendid

Massiiv

Andmeid hoitakse Javas lihttüüpidena, vajadusel grupeeritakse neid selguse huvides struktuurtüüpidesse. Ühetüübiliste või ühest ülemklassist alanevat tüüpi andmeid saab hoida massiivis. Massiivi andmete töötlemiseks on loodud paketti `java.util` klass `Arrays`. Seal olevate meetodite abil on võimalik massiivi sortida, massiivist sobivat elementi leida, massiivi määratud väärtusega täita ning kaht massiivi võrrelda. Järgnev näide kirjutab sorteeritutena välja käsurea parameetritena antud sõnad. Lisaks teatatakse sõna "kass" asukoht massiivis või selle puudumine.

```
import java.util.*;
public class Sort2 {
    public static void main(String args[]) {
        Arrays.sort(args);
        for(int i=0; i<args.length; i++)
          System.out.println(args[i]);

        String otsitav="kass";
        int koht=Arrays.binarySearch(args, otsitav);
        if(koht>=0){
          System.out.println("Massiivis asub "+otsitav+
            " kohal nr. "+koht);
        } else{
          System.out.println(otsitav+" puudub massiivist");
        }
    }
}

D:\arhiiv\naited\io\muu>java Sort2 koer ahv kass lammas kits
ahv
kass
kits
koer
lammas
Massiiviis asub kass kohal nr. 1

D:\arhiiv\naited\io\muu>java Sort2 leevike tihane kurg
kurg
leevike
tihane
kass puudub massivist
```

Sorteerimiseks on käsk `sort`, otsimiseks on meetod `binarySearch`, mis eeldab massiivi eelnevat sorditust. Massiivi andmetega täitmiseks on meetod `fill` ning kaht massiivi aitab võrrelda `equals`. Samanimelise meetodiga võib sorteerida nii sõnesid, täisarve, reaalarve, kui igasugu muid objekte, st. nii liht- kui struktuurtüüpe. Sorteerides tõstetakse massiivi sees elemendid ringi. Sorteerimisalgoritmiks on kiirsortimise ning shell-sortimise vahepealne algoritm, kus sorteerimata algandmete korral ei kulu sortimiseks aega enam kui n*log(n), samas aga eelnevalt sorditud andmete korral töötab tunduvalt kiiremini. Sama väärtusega elementide järjekord jääb ka pärast sorteerimist samaks. Lihttüüpide korral võrreldakse andmete väärtusi, liittüüpide korral kasutatakse `compareTo` meetodit, eeldades, et massiivis asuvaid elemente on võimalik omavahel võrrelda. Probleemi tekkimisel heidetakse meetodist välja erind. Kui andmeid ei saa omavahel `compareTo` abil võrrelda või annab see kasutaja jaoks soovimatu tulemuse, siis saab kirjutada ise liidest `Comparator` realiseeriva klassi mille abil massiivis olevaid elemente võrrelda. Nii tuleks näiteks teha, kui soovime, et nimede tähestiku järjekorda seadmisel arvestataks täpitähti õigesti.

Järgnevas näites on loodud oma lihtne `Comparator`, mis võrdleb sõnade järjestust tähestikus, jättes esimese tähe arvestamata. Nii on Lammas eespool kui Koer, sest esimesel juhul on teiseks täheks a, teisel juhul o. Võrdleja kirjutamisel on pääsetud lihtsalt, sest on kasutatud kahe sõne võrdlemiseks tarvitatavat meetodit compareTo. Üle kaetud meetodile compare antakse ette kaks sorteeritava massiivi elementi, millede vahelist omavahelist järjestust on vaja sorteerimisel teada. Ehkki meetodi päises on mõlema elemendi tüübiks Object, võime praegu päris kindlad olla, et tegelikult saabuvad andmed on tüübist String. Seda lihtsalt sellepärast, et me ise anname sorteerida String-tüüpi massiivi ning mujalt pole pakutavaid elemente kusagilt võtta. Et aga kõiki struktuurtüüpe võib Javas `Object`iks ning tagasi muuta, sel juhul on mugav kirjutada liideses `Comparator` meetodi compare parameetriteks kaks `Object`i ning lasta juba edaspidi programmeerijal sealt omale vajalikku tüüpi andmed tagasi välja võtta. Enamasti tuleb andmeid omale sobivaks muundada tüübimuunduse ehk `cast`'iga ehk `String s1=(String)o1;` , kuid kuna klassis `Object` (ning ka kõigis tema alamklassides ehk

kokkuvõttes üldse kõigis Javas ette tulevates klassides) on meetod `toString` ning `String`i puhul väljastab see lihtsalt tema väärtuse, siis saab sõne väärtuse ka lihtsalt kätte, kirjutades `o1.toString()`.

```
import java.util.*;
public class Sort3{
  public static void main(String[] argumendid){
    String[] loomad={"Koer", "Kass", "Lammas", "Lehm", "Elevant", "Kits"};
    Arrays.sort(loomad, new TeisestTahestVordleja());
    for(int nr=0; nr<loomad.length; nr++){
      System.out.println(loomad[nr]);
    }
  }
}

class TeisestTahestVordleja implements Comparator{
  public int compare(Object o1, Object o2){
    return o1.toString().substring(1).compareTo(o2.toString().substring(1));
  }
}
```

```
C:\kodu\jaagup\0204\k1>java Sort3
Lammas
Kass
Lehm
Kits
Elevant
Koer
```

Java Collections Framework

Lisaks massiivile on Java keeles ka muid vahendeid andmete hoidmiseks ja töötlemiseks. Mitmetes masinalähedasemates programmeerimiskeeltes aitavad andmetega kiiremini ja paindlikumalt ümber käia viidad, siin asendavad neid osutid. Et programmeerijad saaksid paremini ja paremaid programme kirjutada, selleks on loodud andmehulkade hoidmist ning nendega manipuleerimist kirjeldav liideste kogum, Java Collections Framework. Liidesed on kaetud klassidega, kuid neid saab vajadusel ise luua ning tõenäoliselt luuakse ka Java arendajate poolt valmis liidestele konkreetsetesse oludesse klasse juurde. Java loojad pakuvad välja Collections Framework'i kasutamisel välja järgmised eelised.

Programmeerimise kiirus kasvab, kuna ei tule pidevalt luua uusi klasse eri tüüpi andmega tegelemiseks ega andmete muundamiseks. Ka programmi enese kiirus peaks kasvama, kuna optimeeritud valmisklasside kasutamisel ei pea nende kallal enam vaeva nägema, samas aga on kindlasti võimalik vabaneva ajaga ülejäänud programmi paremaks teha. Selge süsteem võimaldab kergemini andmeid vahetada teiste programmidega. Õppimine ning ajaga kaasas käimine peaks minema lihtsamaks, kuna põhilised osad jäävad samade liideste kasutamisel ka erinevatel andmetel samaks. Uue andmetöötlussüsteemi välja töötamisel saab olemasolevad liidesed aluseks võtta, kuna neis on juba läbikaalutud meetodid olemas. Suuremas plaanis suurendab ühtne liideste komplekt programmi taaskasutust, sest siis on kergem ennustada, mida ühelt või teiselt programmijupilt oodata võib.

Loomulikult on aga ühtsel süsteemil omad puudused. Mõnes olukorras on ta kohmakam, sest harva on võimalik optimiseerida sama hästi erinevate ootuste jaoks kui seda saaks ühe kindla eesmärgi jaoks teha. Samas aga on ehk korralik kombain parem kui ülejala tehtud ning läbi kontrollimata vahend. Tavanäiteks võiks olla, et kärbsepiits-klopper-pudeliavajaga on raskem kärbest kinni püüda kui lihtsa viimistletud kärbsepiitsaga, kuid tõenäoliselt hulga tulemuslikum kui tavalise tolmulapiga, mis meil enamasti juhtub käepärast olema.

Kollektsioon

Andmeliideste juureks on `Collection`, igasugune andmekogum. Kirjeldatud on mõniteist meetodit, mida nende andmetega teha saab. Meedodi `add` abiga saab elemendi lisada, `remove` võtab vastava elemendi kollektsioonist ära; `addAll` lisab terve teise kollektsiooni, `clear` teeb platsi puhtaks; saab kontrollida, kas kollektsioon on tühi, kui palju temas on elemente, kas temas leidub määratud objekt, kas temas sisaldub teine kollektsioon või on ta hoopistükkis teisega samaväärne. Vastavad meetodid oleksid `isEmpty`, `size`, `contains`, `containsAll` ja `equals`. Kollektsioonist saab andmeid kätte, muundades ta massiiviks või küsides temalt objektijada tüübist `Iterator`. Selle abil saab ükshaaval läbi vaadata kõik kollektsiooni elemendid, vajadusel küsides osuti vajalikule elemendile või eemaldades ta kollektsioonist.

Nimistu

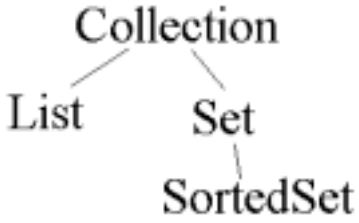
`List` on järjestatud kollektsioon, ehk selline andmekogum, kus igal elemendil on järjekorranumber. Ta on kollektsiooni alamliideseks. Juurde tulevad meetodid, mis on seotud järjekorranumbriga. Näiteks saab lisada elementi määratud kohale, samuti eemaldada või küsida teda sealt. Elemendi lisamisel temast paremale jääjate järjekorranumber suureneb ühe võrra, eemaldamisel väheneb. Kui sooviksime massiivi keskele lisada, peaksime vastava algoritmi ise kirjutama, `List`i ehk nimekirja puhul on see juba olemas. Massiivi saab nimekirjaks muuta klassi `Arrays` käsuga `asList`, vastupidi saab aga nimekirjale öeldes `toArray()`. Kui massiivis asuvad elemendid enamjaolt üksteise järel ning nende asetsemist mälus saab muuta üksnes intepretaator või operatsioonisüsteem, siis `List` on vaid liides ning tema realisatsioonis saab vabalt määrata, kas elemente hoitakse osutite abil teineteisele viidates, massiivis või hoopis mõnel muul moel (kas või näiteks faili abi kasutades). Liides `List` on vaid kirjeldus, kuidas andmetele ligi pääseb.

Hulk

`Set` on samuti kollektsiooni alamliides. Temas võib iga väärtusega elementi olla vaid üks nagu matemaatilises hulgas kohane. Liides `Set` ei ütle aga midagi elementide järjekorra kohta. Temasse lisades jäävad alles vaid erinevad elemendid. Nii saab tema abil kergesti suurest komplektist erinevad välja sõeluda.

`SortedSet` on `Set`'i alamliides. Nagu nimigi ütleb, on temas elemendid järjestatud. Lisaks muule `Set`'i võimalustele saab temalt küsida esimest ja viimast elementi, samuti alamhulki alates või kuni mingi väärtuseni ning hulka kahe väärtuse vahelt.

Nagu klass `Arrays` sisaldab meetodeid massivi töötlemiseks, nii klassi `Collections` meetodid aitavad ümber käia kollektsiooni ning tema alanejatega. Saab leida kollektsioonist vähimat ning suurimat elementi, muuta kollektsioon sünkroniseerituks (s.t. et tema poole saab korraga pöörduda vaid üks lõim) või keelata temasse kirjutamine. Klass aitab sorteerida `list`i, samuti sorteeritud `list`is elemente segi paisata (meetod `shuffle`). Võib keerata järjekorra vastupidiseks, täita nimekirja määratud elemendiga või otsida elementi.



Tegelikud realisatsioonid

Et programmeerija ei peaks vaid ilusaid kirjeldusi vaatama, vaid saaks ka välja pakutud hüvesid ise neid loomata maitsta, selleks on loodud liidesed ka realiseeritud. Kollektsiooni realiseerijaks on küll vaid `AbstractCollection`, mis mõeldud aitamaks ise kollektsiooni luua, kuid muudele liidestele on kasutatavad katted olemas. Listi realiseerijateks on määratud `LinkedList`, `Vector` ning `ArrayList`.

Kasutaja jaoks käituvad nad ühtmoodi nii, nagu liideses kirjeldatud, sisimas aga on erinevad ning suuremate andmehulkade korral on kasulik nende eripära arvestada. `LinkedList` on jadas paiknevate elementide kogum, igalt elemendil (v.a. viimane) on osuti järgmisele. Sellise lahenduse juures on suhteliselt kerge elemente keskele vahele panna, sest piisab vaid kahe (pandava ning temale eelneva) elemendi ümber tõstmisest. Samas on pika jada keskele jõudmine suhteliselt vaevarikas, sest tuleb kõik vahepeal asuvad elemendid läbi käia, et otsitu juurde jõuda.

`Vector` ning `ArrayList` hoiavad oma andmeid massiivis. Andmetele on lihtne ligi pääseda, kuid eemaldamiseks või lisamiseks tuleb kõiki muudetavast taga pool paiknejaid ühe võrra liigutada, mis jällegi suuremate andmete puhul päris palju operatsioone nõuab. Samuti saab järjest andmeid lisades ükskord massiiv lihtsalt täis. Sellisel puhul luuakse uus suurem massiiv ning kopeeritakse

andmed sinna. Et andmete lisamisel ümber kopeerimist liialt tihti ette ei tuleks, selleks luuakse igal korral vähemalt `Vectori` puhul eelmisest kaks korda suurem massiiv.

Set-liidese realiseerijaks on `HashSet` ning `SortedSet` võimalusi aitab kasutada `TreeSet`. Viimane vähemasti standardrealisatsioonis kasutab kahendpuud ning selle algoritmi järgi on soovitud väärtusi küllalt kiire leida ka suuremate andmehulkade puhul.

Järgnevas näites luuakse alustuseks loomanimedest sõnemassiiv. Massiiv muudetakse `Vector`'iks, mis realiseerib liidest `List`. Nagu nimistule kohane, saab temasse andmeid soovitud kohta lisada ning sealt eemaldada. Siin paigutatakse kohale nr. 2 (ehk kolmandaks) rebane. Väljatrükil on näha, et parempoolsed elemendid on ühe võrra edasi nihkunud. Nimistu muudetakse `HashSeti` abil hulgaks. Väljatrükil on näha, et korduv koer on kadunud. Ka muud elemendid on oma kohta vahetanud, kuid see on lubatud, sest hulgas pole elementide järjekord määratud. Lisades kitse ja lamba, tuleb juurde vaid kits, sest lammas oli hulgas juba olemas. Hulga sorteerimiseks paigutame andmed isendisse tüübist `TreeSet`. Selles klassis jäävad andmed sordituks ka pärast uute elementide lisamist.

```
import java.util.*;
public class Andmed{
  public static void main(String argumendid[]){
    String loomamassiiv[]={"kass", "koer", "lammas", "koer", "ahv"};
    List loomalist=new Vector(Arrays.asList(loomamassiiv));
    loomalist.add(2, "rebane");
    System.out.println(loomalist);
    Set loomahulk=new HashSet(loomalist);
    System.out.println(loomahulk);
    loomahulk.add("kits");
    loomahulk.add("lammas");
    System.out.println(loomahulk);
    SortedSet sorteeritudLoomahulk=new TreeSet(loomahulk);
    System.out.println(sorteeritudLoomahulk);
    sorteeritudLoomahulk.add("antiloop");
    System.out.println(sorteeritudLoomahulk);
  }
}
```

Ning programmi töö tulemus:

```
[kass, koer, rebane, lammas, koer, ahv]
[ahv, lammas, kass, rebane, koer]
[ahv, kits, lammas, kass, rebane, koer]
[ahv, kass, kits, koer, lammas, rebane]
[ahv, antiloop, kass, kits, koer, lammas, rebane]
```

Paisktabel

Objektipaaride jaoks on loodud liidesed `Map` ning `SortedMap`. Neid realiseerivate klasside `HashMap, Hashtable` ning `TreeMap` abil saab võtmetele panna vastama väärtused, samuti küsida võtmetele vastavaid väärtusi. Neid saab kasutada nagu sõnaraamatut, kuid seletuste asemel ei pruugi olla vaid sõnad, vaid sobivad igasugu objektid. All näites on võtmeteks valvamiskohad ning väärtusteks inimeste nimed, kes vastaval kohal olema peavad.

```
import java.util.*;
public class Paisktabel{
  public static void main(String[] argumendid){
    Hashtable korrapidajad=new Hashtable();
    korrapidajad.put("koridor", "Juku");
    korrapidajad.put("klass", "Kati");
    System.out.println(korrapidajad);
    korrapidajad.put("klass", "Mari");  //asendab Kati
    System.out.println("Koridoris valvab:"+korrapidajad.get("koridor"));
    System.out.println("Valvekohti kokku: "+korrapidajad.size());
    Enumeration kohaloend=korrapidajad.keys();
    while(kohaloend.hasMoreElements()){
      String koht=(String)kohaloend.nextElement();
      System.out.println("Valvekoht: "+koht+", valvaja: "+korrapidajad.get(koht));
    }
    Collection valvajad=korrapidajad.values();
    Iterator valvajaloend=valvajad.iterator();
```

```
    System.out.println("Valvajad:");
    while(valvajaloend.hasNext()){
      System.out.println(valvajaloend.next());
    }
  }
}
```

```
D:\kodu\0309\oma>java Paisktabel
{klass=Kati, koridor=Juku}
Koridoris valvab:Juku
Valvekohti kokku: 2
Valvekoht: klass, valvaja: Mari
Valvekoht: koridor, valvaja: Juku
Valvajad:
Mari
Juku
```

Ülesanded

Sünniaastad

Koosta tekstifail, milles on hulk sünniaastaid.

- Leia, mitu korda esineb nende seas aasta 1959
- Loe tekstifaili aastaarvud LinkedList-i
- Väljasta need aastaarvud sorteerituna teise tekstifaili
- Otsi, kas loetelus leidub kasutajalt küsitud aastaarv
- Väljasta teise tekstifaili vaid erinevad aastaarvud
- Sega aastaarvude järjekord

Dokumenteerimine

Paarist reast pikemate programmide mõistmiseks ei pruugi põgusast pealevaatamisest piisata. Meetodite ja klasside nimed veidi selgitavad olukorda, kuid peetakse sobivaks ka pikemaid seletusi programmilõikude töö ning kasutusvõimaluste kohta anda. Enesel on nii lihtsam koodi meelde tuletada ning annab grupitöö korral võimaluse teistel programmeerijatel tehtut kasutada, ilma et peaks autorilt selgitusi pärima või iga programmirea eesmärki eraldi uurima. Java lähtekoodi saab kommentaare lisada kahel moel. Tekst kahest järjestikusest kaldkriipsust rea lõpuni loetakse kommentaariks näiteks:

```
int arv=3 // kassi poegade arv
// poegade arvu järgi tuleb arvestada toiduvajadust
double toidukulu=arv*koefitsient
```

Pikemat programmi osa saab välja kommenteerida märkides selle osa alguse ja lõpu. Algust tähistab `/*` ning lõppu `*/` . Valikuvõimaluse puhul soovitakse kasutada ridade kaupa väljakommenteerimist, pidada vähem vigu tekkima.

Programmifaile iseloomustavate HTML-lehtede loomiseks kuulub JDK programmide koosseisu `javadoc`, mis koondab programmidest meetodite kirjeldused koos vastavalt märgistatud kommentaaridega. Vastavalt sihtgrupile võib lasta javadocil välja tuua mitmesuguse tähtsusastmega programmiosi. Näiteks kaasprogrammeerijatele tuleks näidata ka vaid paketi või klassi piires kasutatavaid muutujaid, klassi väljapoolt kasutajale pole neid tarvis. Ka loob `javadoc` võimaluse korral viited muude klasside kirjeldustele. Lisaks sellele on siiski sageli viisakas lisada "kirjeldav" ülevaade ning kasutamisõpetus. `Javadoc`i abil on koostatud ka JDK API dokumentatsioon, mida abiinfona kasutame. Selliselt automaatsena loodud dokumentatsiooni puhul on eelis, et andmed ei saa inimlike eksituste tõttu valeks minna. Näiteks C-keele puhul võis kergemini ette tulla olukordi, kus käsud on küll valmis kirjutatud ja olemas, kuid neid ei tea kasutada, kuna on unustatud dokumentatsiooni lisada või on selle kirjutamise juures viga tekkinud.

Klassi ja meetodite kirjelduste lisamiseks javadoci abil loodud HTML-faili tuleb need kirjeldused paigutada `javadoc`ile mõistetavalt kujundatuna klassi või meetdi ette. Äratundmiseks peab kommentaar algama reaga, millel kaldkriips ja kaks tärni. Iga rea algul on üks tärn ning kommentaari lõpus tärn ja kaldkriips. Teksti esimene rida peab olema kokkuvõttev - see tõstetakse meetodite omaduste kokkuvõtete tabelisse. Edasine pikem kirjeldus on näha vaid tagapool. Sellisena on hea kiiresti ülevaade saada ning suurest meetodite hulgast omale sobivad välja leida. Kirjelduse sees võib kasutada HTML kujundust. Erikirjelduste abil saab määrata tunnuste väärtusi. `@author` tähendab programmifaili autorit, `@since` versiooni, millest alates seda programmi kompileerida ning kaivitada saab. `@see` loob viite olemasolevale meetodile, mis kommentaari kirjutaja selles kohas tähsaks viidata peab.

Järgnevalt on Javadoci kommenteerimise näitena toodud klass punkti koordinaatide hoidmiseks ning sealt väärtuste pärimiseks.

```
/**
* Klass abistab <b>kasutajat</b>
* punkti andmete hoidja ning analüüsijana.
* Koostatud <a href="http://www.tpu.ee">Tallinna Pedagoogikaülikoolis</a>.
* @see #kaugusNullist
* @author Jaagup Kippar
* @since JDK1.0
*/
public class DokumenteeritudPunkt{
  /**
  * x-koordinaat
  */
  public int x;

  /**
  * y-koodinaat
  */
  public int y;

  /**
  * Klassimuutuja loodud punktieksemplaride arvu loendamiseks.
  * @see #teataPunktideArv
  */
  static int punktideArv;

  /**
  * Parameetriteta konstruktor, jätab muutujate väärtused nulliks.
  */
  public DokumenteeritudPunkt(){
    x=y=0;
  }

  /**
  * Konstruktor lähteandmete sisestamiseks.
  * <p>
  * Konstruktoris määratud parameetrid jäävad
  * vastavate muutjate väätusteks
  * @param x loodava punkti x-koordinaat
  * @param y loodava punkti y-koordinaat
  */
  public DokumenteeritudPunkt(int x, int y){
    this.x=x;
    this.y=y;
  }

  /**
  * Leitakse vahemaa tasandil oleva punkti asukoha ning
  * koordinaatide alguspunkti vahel.
  * Vahemaa leitakse ruutjuure abil koordinaatide ruutude summast.
  * @return kaugus koordinaatide alguspunktist reaalarvuna.
  */

  public double kaugusNullist(){
    return Math.sqrt(x*x+y*y);
  }

  /**
  * Väljastatakse loodud punktide arv.
  * @return programmi töö keskel loodud vastavat tüüpi isendite arv.
  */

  public static int teataPunktideArv(){
    return punktideArv;
  }
}
```

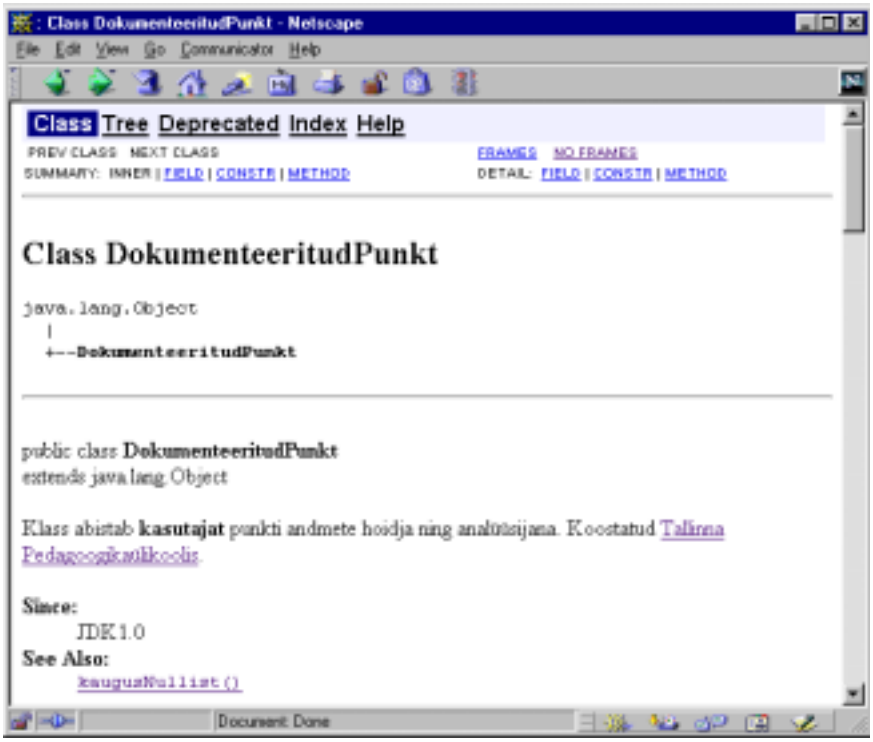
Dokumentatsiooni loomine käivitatakse, kirjutades javadoc ning dokumenteeritava faili nimi. Soovi korral saab mitmesuguste võtmetega päris palju dokumentatsiooni loomise juures määrata. Aeglasema masina puhul võib javadoci töö päris hulga sekundeid aega võtta. Samuti venib aeg pikemaks, kui korraga koostatakse dokumentatsiooni mitte ühele lähtekoodifailile vaid rohkematele. Samuti luuakse töö käigus päris hulk HTML-faile, mis võtab ka oma aja.

```
C:\kodu\jaagup\0204\k1>javadoc DokumenteeritudPunkt.java
Loading source file DokumenteeritudPunkt.java...
Constructing Javadoc information...
```

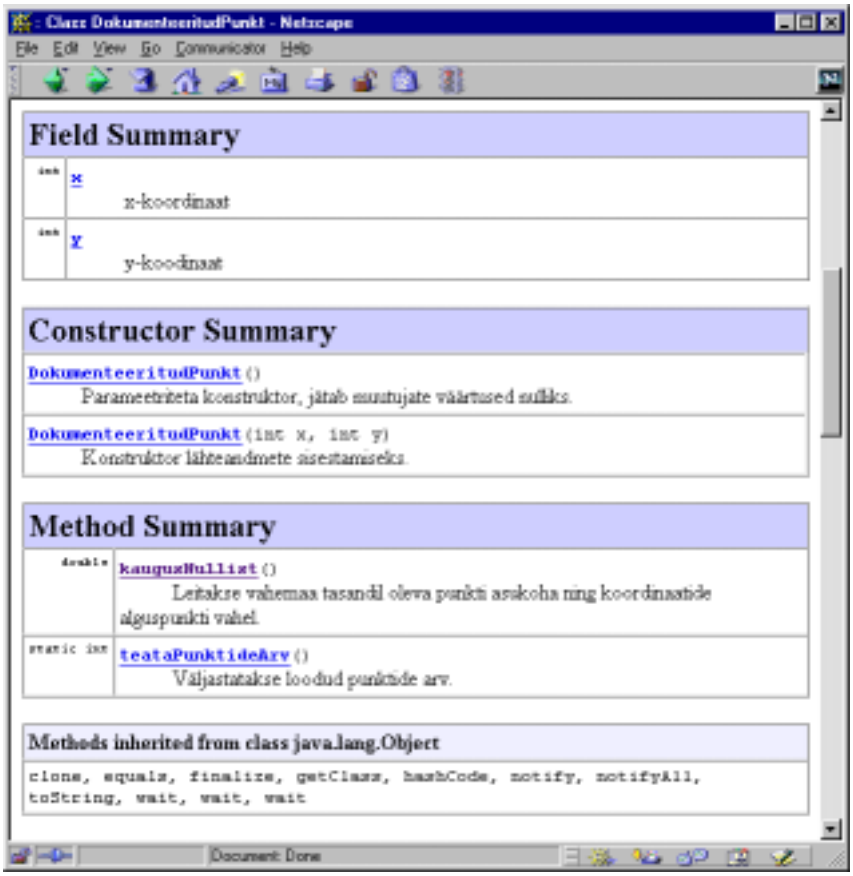
```
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating DokumenteeritudPunkt.html...
Generating serialized-form.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...
```

Javadoci töö tulemuseks on loodetavasti juba tuttava väljanägemisega abiinfo leht. Ülaservas olevate viidetega on rohekem peale hakata, kui on tegemist rohkemate klasside ja meetoditega (nagu näiteks standarpakettide puhul). Tree näitab puuna välja kõik dokumentatsioonis leiduvad klassid. Deprecated-loetelus on kirjas vananenud ning ebasoovitavad meetodid. Index-ist võib tähestiku järjekorras leida soovitud klassi või meetodi - hea abiline juhul, kui mäletad küll käsu ligikaudset nime, aga ei tule meelde, kust kandist seda otsida võiks.

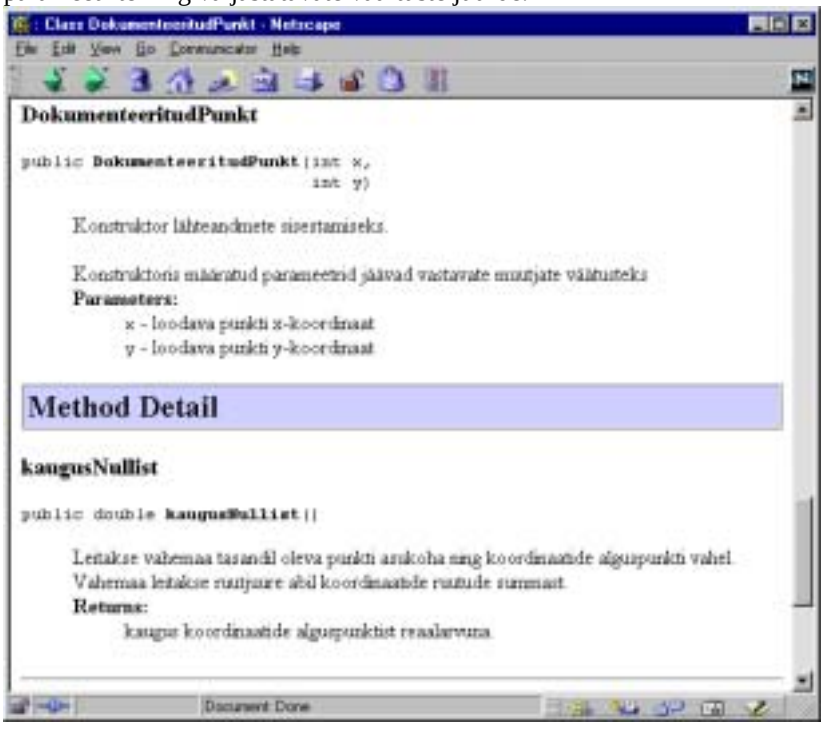
Edasi allpool tulevad konkreetse klassi andmed. Nimi ning joonis, kuidas vastav klass asub objektide hierarhias. Et loodud DokumenteeritudPunkt'il eraldi ülemklassi pole, on tema ülemklassiks kõige juur Javas ehk java.lang.Object. Edasi tulevad juba lähtekoodi seest välja korjatud kommentaarid.



Siis nii väljade, konstruktorite, meetodite lühikirjeldused ning päritud meetodite nimed. Viimaste kirjelduste vaatamiseks tuleks juba üles otsida vastava klassi (praegusel juhul java.lang.Object) dokumentatsioon.



Allapoole minnes juba leiab iga käsu detailsema kirjelduse, samuti seletused, mis on määratud parameetrite ning väljastatavate väärtuste juurde.



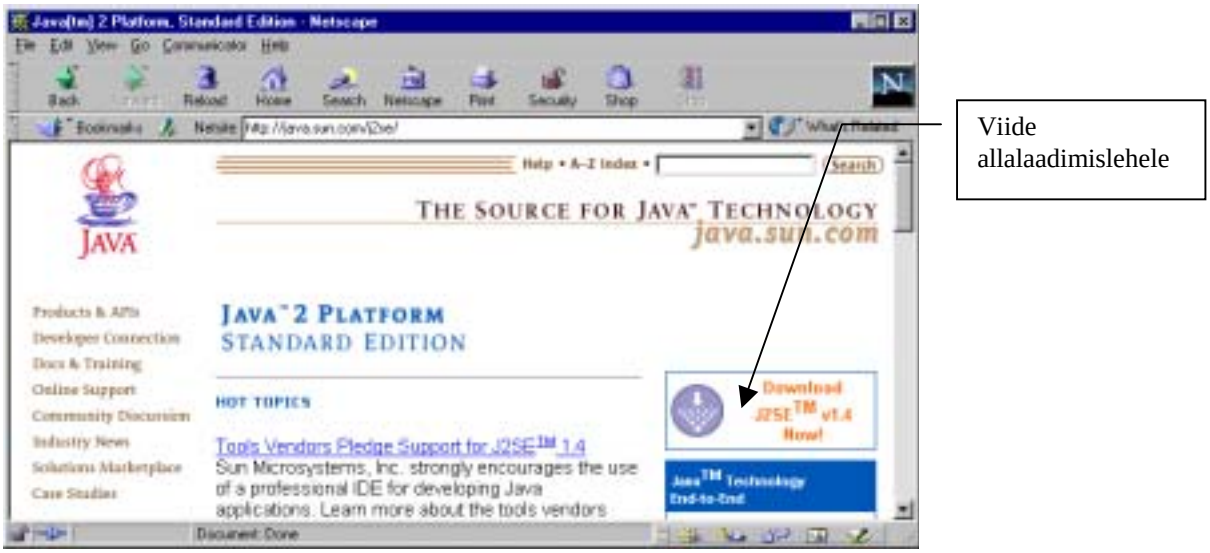
Installeerimine

Järgnevalt on püütud “puust ette ja punaseks” selgeks teha, kuidas Windows operatsioonisüsteemiga arvuti peal laadida ja installeerida enesele tööks vajalik J2SDK ning selle abil pisike programm koostada ja käima panna.

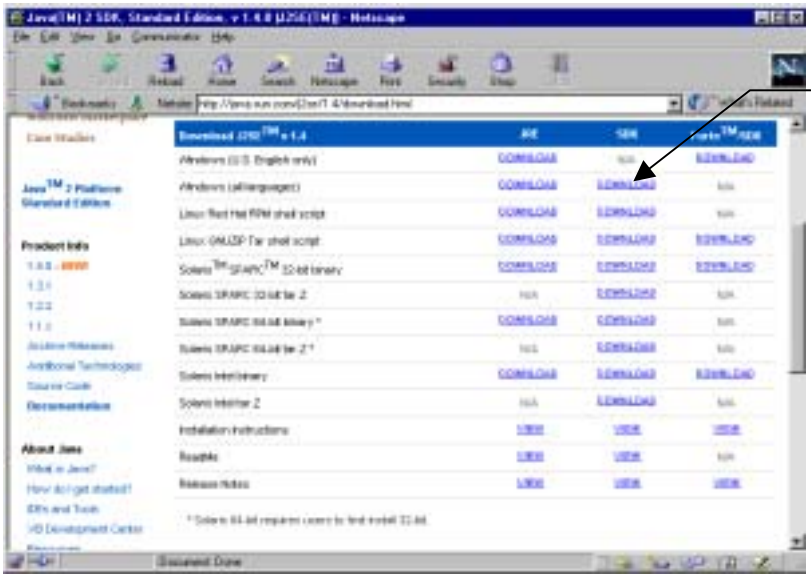
Java vahendeid tasub otsima hakata SUNi Java-teemaliselt leheküljelt http://java.sun.com/. Sellelt lehelt leiab Java-valla uudiseid ning ohtralt viiteid. Tarkvara laadimiseks tuleb valida vasakult

menüüst "Products & APIs". Jõutakse lehele, kus on kirjas kümnete kaupa kirjas Javaga seotud alateemasid. Küllalt tähtsal kohal peaks silma hakkama Java 2 SDK Standard Edition mõne oma versiooniga.

Ringi vaadates võib leida lehekülgede kaupa tutvustavat juttu, kuid kui eesmärgiks on vahend oma arvutisse katsetamiseks tööle saada ning mitte infotulva ära uppuda, siis võiks pigem otsida viite, kust platvormi enesele alla laadida saaks. Vastav paik peaks olema tähistatud sõnaga Download. 2002. aasta kevadel, mil seda kirjatükki luuakse, oli just välja tulnud J2SDK versioon 1.4.0. Selle põhjal viiakse läbi ka kogu installeerimistoiming. Seekord läks allalaadimiskoha leidmine kergelt, sest lehe paremasse ülaserva oli välja toodud eraldi reklaamnupp "Download J2SE v1.4 now!", kuid ka muidu ei tohiks vastav viide eriti peidetud olla.

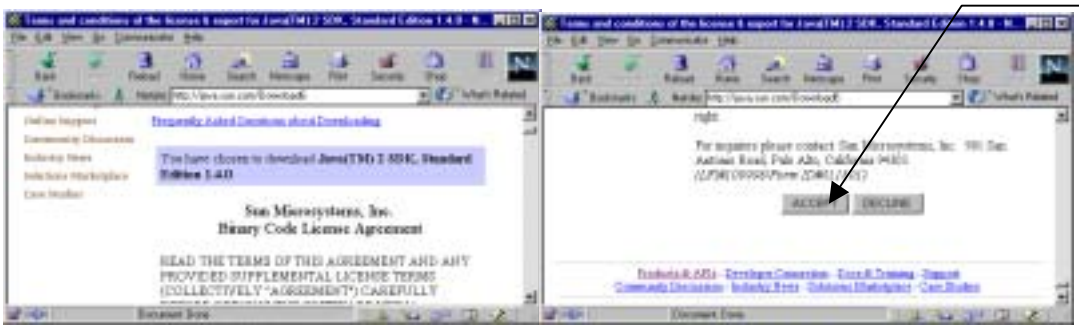


Laadimise juures tuleb silmas pidada, millise operatsioonisüsteemi jaoks keskkonda vajatakse ning kas soovitakse vaid valmisprogramme käivitada või soovitakse neid ka ise kokku panna. Meil on kavas oma programme luua, seetõttu ei piisa JRE-st (Java Runtime Environment), vaid tuleb otsida SDK (Standard Development Kit). Kolmandas tulbas lahkesti välja pakutav Forte-nimeline programmide koostamise keskkond aitab küll mõnikord suuremate programmide puhul kujundust korrastada ning veidi vigu mööda programmi püüda, kuid esiotsa on tavalise kompilaatori ja intepretaatoriga lihtsam hakkama saada ning masina ressursse kulub ka vähem.



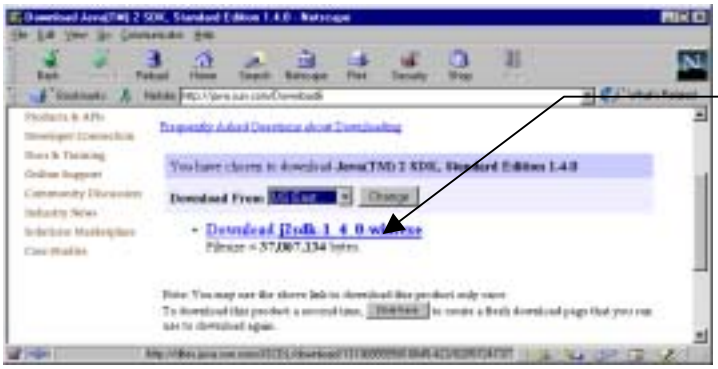
Meile sobib rahvuskeelte toetusega SDK

Edasi liikudes palutakse meil läbi lugeda litsents, kust saame teada, et Java väljatöötaja ei vastuta selle programmi kasutamise juures tekkivate probleemide ja õnnetuste pärast ning antakse hulga muud teavet. Sellelt lehelt lastakse edasi vaid juhul, kui vajutada nupule ACCEPT.



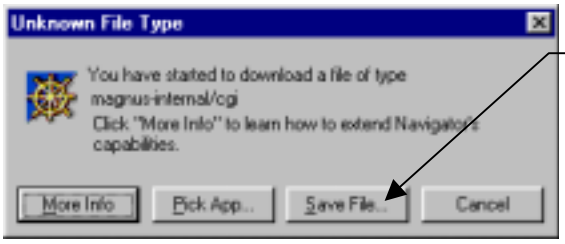
Vaid tingimustega nõustudes pääseb edasi

Litsensilugemisefaasi edukalt läbinuna pakutakse meile võimalust hakata suurt faili tegelikult alla laadima. Juures praegusel juhul teade, et andmete maht on 37 MB - kogus mida vähemalt aeglasema ühenduse omanikud peaksid kindlasti silmas pidama ning kaalutlema, kas otsustada mitmetunnise ootamise ja kopsaka telefoniarve kasuks või püüdma minna mõne kiirema ühendusega masina juurde ning sealt tarkvara omale plaadiga koju kopeerida.



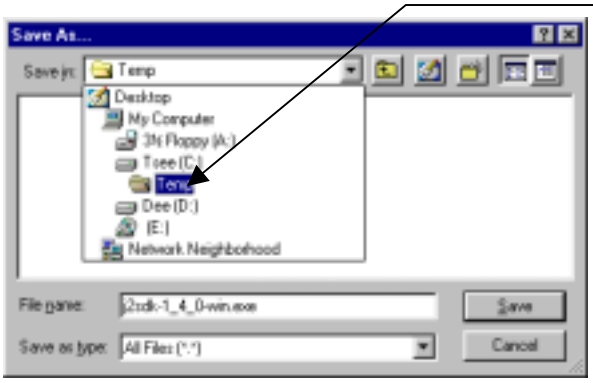
Siit võib lõpuks laadima hakata

Kui arvutile juhtub pakutav andmete tüüp tundmatu olema, siis võib ette tekkida ligikaudu taoline dialoogiaken. Et meie eesmärgiks on esialgu installeerimisfail kettale salvestada ning alles siis sellega tasapisi edasi toimetama hakata, võime üsna julgesti valida "Save file".



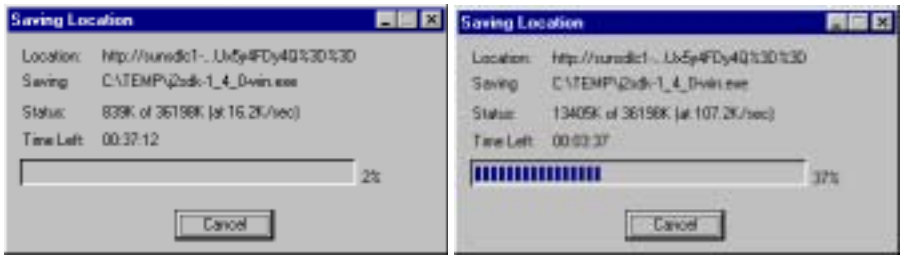
Käskluse Save alt peaksime saama oma faili salvestada

Muidugi võib juhtuda, et vahepealset dialoogiakent ette ei tõstetagi, sellisel juhul võime kohe asuda määrama paika, kus soovime, et salvestatav fail me arvutis asuks. Siin paigutatakse ta C-kettale temp-kataloogi ehk kohta, kuhu võib küllalt julgesti mitmeid asju ajutiselt panna, kui nende tarvis oma masinasse paremat kohta mõeldud pole. Igaks juhuks tuleb jälgida, et faili laiendiks ikka .exe pannakse. Muul juhul võib programmi käivitamisega raskusi tekkida.



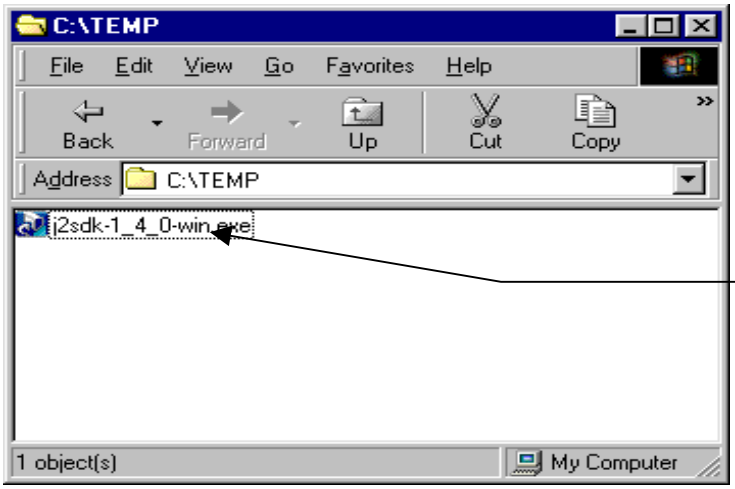
C:\temp sobib ajutiste failide hoidmise kohaks täiesti, kui oma masinasse muud süsteemi loodud pole

Edasi järgneb pikk ja väsitav ootamine, et saabuv fail ükskord ometi kohale jõuaks. Kiire ühenduse korral võib kohale jõudmine päris ruttu minna, aeglasemal juhul aga võivad kilobaidid tilkuda ja tilkuda, lisaks pidevalt väike mure kripeldamas kui ühendus sootuks suvatseks ära kaduda ning kogu vaev oleks seega luhta läinud.



Hea õnne korral aga võime mõne aja pärast tõdeda, et kõik andmed on ilusti kohale jõudnud ning võime installeerimise kallale asuda. Juhul kui on õnnestunud kusagilt installeerimisplaat hankida, kus sama fail peal, siis need inimesed võivad ka siinsest punktist liituda.

Oma arvutist tuleb soovitud installeerimisfail üles otsida ning käivitada.



Topeltklõps õigel failil ning installeerimine algabki.

Veidi ragistamist, mõnigased ettevalmistustööd ning juba hakataksegi teateid jagama ning valikuid küsima. Esialgu pole muud teha, kui pakutuga nõus olla ning Next vajutada.

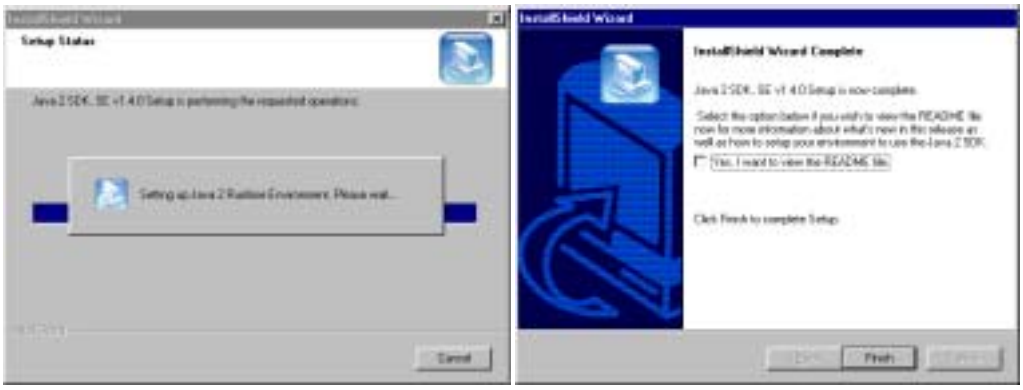
Pakutakse veelkord litsentsitingimusi lugeda ning seejärel küsitakse, kuhu kasutaja soovib programmi oma masinas installeerida. Vaikimisi kataloogiks pakutakse c:\j2sdk1.4.0 ning kui omal oluliselt paremaid ettepanekuid ei ole, siis võib see ju jääda. Mõne JDK lisapaketi ja Windowsi versiooni korral võib tekkida probleeme tühikuid sisaldavate katalooginimedega (nt. Program Files), kuid üldjuhul laseb programm end masinasse küllalt vabalt paigutada.

Edasi lastakse valida, millised osad installeerida, millised mitte. Täispaketi puhul kulub veidi üle 100 MB ruumi, miinimumvaliku puhul võib aga paarkümmend megabaiti kokku hoida. Kui tahta suuremat kokkuhoidu, siis tuleks võtta mõni eelmine versioon: JDK 1.2.2 näiteks sisaldab pea kõiki tavaliste programmide koostamisel vaja minevaid võimalusi, samas piirdudes vähem kui kolmekümne megabaidiga. Kui aga pole kavatsust lähemal ajal sügavamale Java standardklasside saladustesse tungida, siis võib lähtekoodi (Java Sources) rahumeeli installeerimata jätta. Samuti saab edukalt hakkama ilma standardseid demoprogramme kopeerimata. Eks neid jõuab hiljem otsida ja proovida, kui selleks peaks vajadus tekkima.

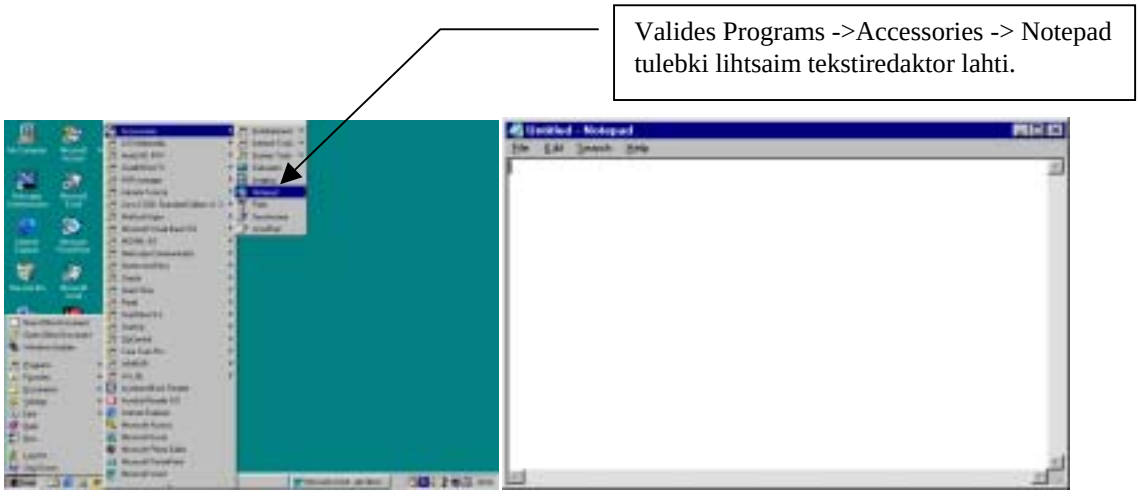
Samuti küsitakse, millise seiluri vaikimisi rakendikäituriks end sättida. Siingi võib pakutuga rahul olla ning pärast Control Paneli alt vajaduse korral oma meele järgi seadinguid muuta.

Taas veidi ootamist, kuni installeerimisprogramm töötab. Kiiremagi masina peal võib see päris mitu minutit võtta rääkimata aeglasemast. Mida rohkem vahendeid on palutud peale panna, seda kauem kopeerimine võtab. Eriti aeganõudvad on sajad pisikesed näiteprogrammid.

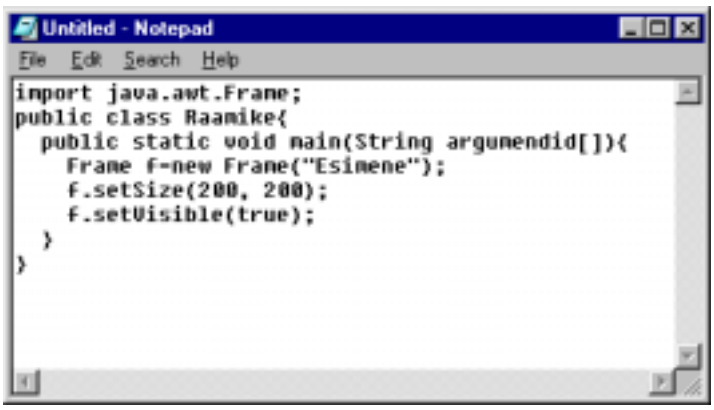
Kui failid kopeeritud, siis kulub veel veidi aega operatsioonisüsteemisiseste sätingute paika panemiseks. Veel natuke ning võime rahuldustundega vajutada nupule Finish.



Pealtnäha ei reeda justkui miski, et masin oleks mõne näpuliigutuse tulemusena kvalitatiivselt uuele tasemele tõusnud - siiamaani tavalisest valmisprogrammide käivitajast abivahendiks uute rakenduste loomisel. Veidi terasemal uurimisel võib programs menüüst mõne uue viite leida, kuid need ei anna veel kuigivõrd uusi oskusi. Meile pakutud kõvaketta kataloogis c:\j2sdk1.4.0 on aga tallel hulganisti salapäraseid vahendeid, millega on põhjust programmeerimise õppimise käigus lähemat tutvust teha. Kõige lihtsam tee eduelamuse saavutamiseks oleks üks lihtne programm käima panna ning mõista, millega on tegu ning kuidas loodut oma kasuks muuta annab. Selleks tuleb programmi tekst valmis kirjutada, kompilaatorile baitkoodi tõlkimiseks kätte anda ning interpretaatoril paluda baitkood käivitada. Koodi kirjutamiseks on Interneti peal sadade kaupa tekstiredaktoreid, kes igaüks oma häid omadusi esile püüavad tõsta, kuid lihtsa teksti kirjutamiseks sobib tegelikult pea iga programm, milles tähti tippida annab. Windowsiga tuleb kaasa Notepad - selle abil püüamegi oma esimese programmi tööle saada.

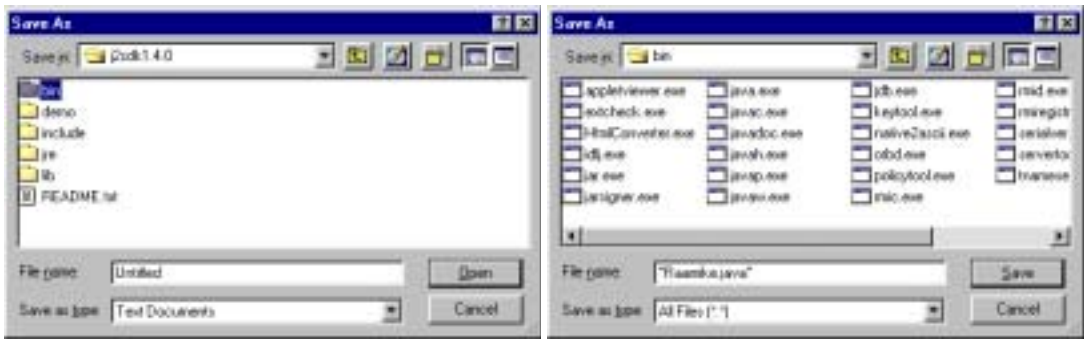


Väikese otsimise peale leiab ehk kusagilt mõne koodinäite, mis kirjade järgi peaks käima minema. Selle peal saabki proovida



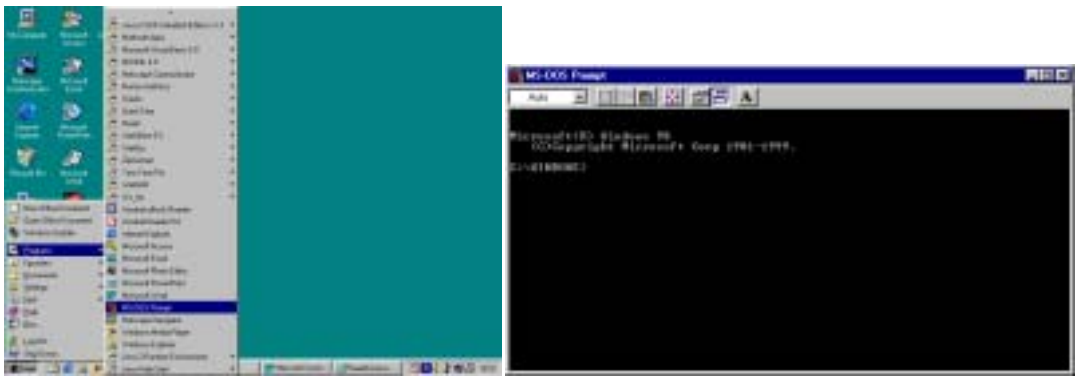
Veidi tuleb mõelda kuhu salvestada, et installeeritud kompilaator selle kätte saaks. Vaikimisi pakutud MyDocuments'ist ei osata esialgu me lähtekoodi otsida. See tuleb paigutada kompilaatorile otse nina alla, c:\j2sdk1.4.0\bin\ kataloogi, ehk samasse kataloogi, kus kompilaator (javac.exe) ise asub.

Faili nimi peab ühtima klassi nimega, ehk kui koodis oli kirjas class Raamike, siis faili nimeks saab Raamike.java . Salvestamisel tuleks tüübiks valida All Files ning faili nimele igaks juhuks jutumärgid ümber kirjutada - sel juhul pole nii palju karta, et Notepad suvatseks failinime ja laiendiga omaloomingut teha.

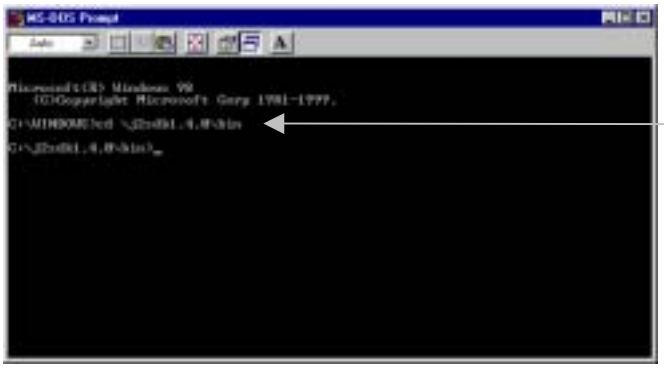


Kui salvestamine õnnestus, siis on faili nimi ilusti näha Notepadi päises.

Edasi tuleb loodud faili kompileerima asuda. Selleks peaks kõige mugavam olema minna käsureale, ehk MS-DOS prompti. Kui juhtub õnneks minema, siis peaks avanema ligikaudu taoline musta taustaga aken.

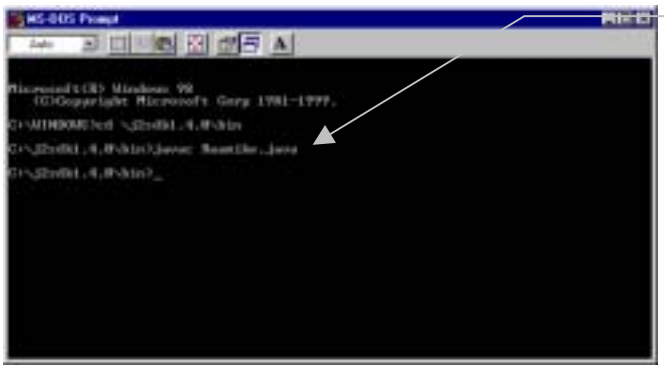


Selles aknas tuleks liikuda käsureaga samasse kataloogi, kus kompilaator ning meie lähtekoodki asuvad. Kui kirjutada cd \j2sdk1.4.0\bin ja vajutada sisestusklahvile (Enter), siis peakski jooksev kataloog olema sinna liikunud.



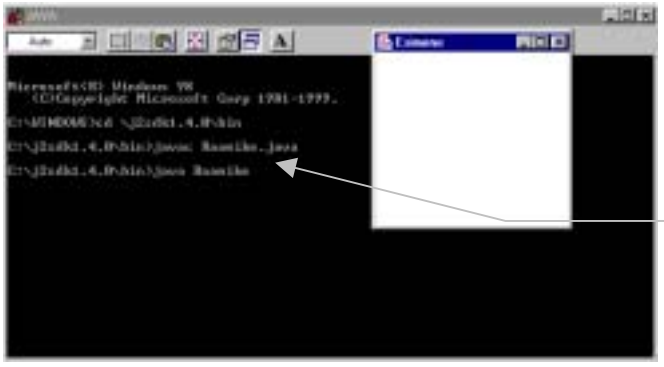
Käsklus cd (change directory) kataloogi vahetamiseks

Kompilaatori nimeks on javac. Kui kirjutada javac Raamike.java, peaks kompilaator aru saama, et sellenimeline fail on vaja ära kompileerida. Kui kõik õnnestub ning trükivigu ei leitud, siis minnakse käsureal pärast kompileerimistöö tegemist rahumeeli järgmisele reale.



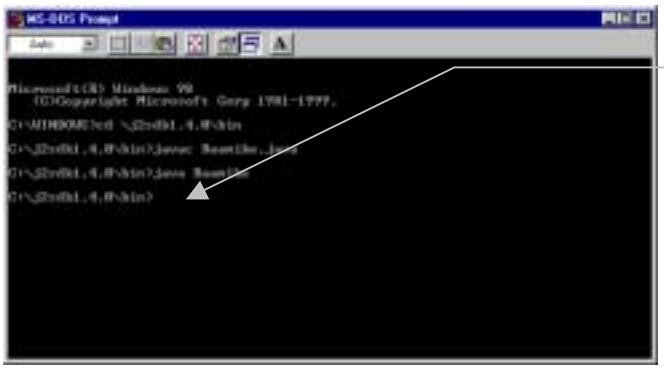
javac ehk Java Compiler tõlgib inimese kirjutatud lähtekoodi masinloetavaks baitkoodiks

Javakeelsete programmide eripäraks on, et neid tuleb käivitada interpretaatoriga. Tolle nimeks on lihtsalt java ning kui kirjutada java Raamike, siis peaks me loodud programmi töö tulemust ekraanil näha olema.



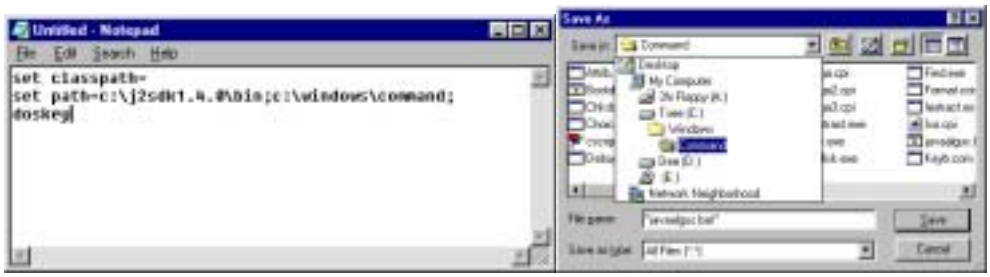
java Raamike ning õnnelikul juhul ongi väike raamaken nähtaval

Oma programmile me veel viisakat sulgemisvõimalust loonud pole. Võime küll akna ülaservas oleval ristikesel klõpsida, kuid selle peale ei juhtu midagi. Esiotsa aitab vaid CTRL + ALT + DEL vajutamine ning programmide loetelust meie esimese raami töö katkestamine või siis viisakam variant CTRL+C. Sel juhul samuti katkestatakse programmi töö ning käsurida jääb järgmist käsklust ootama.

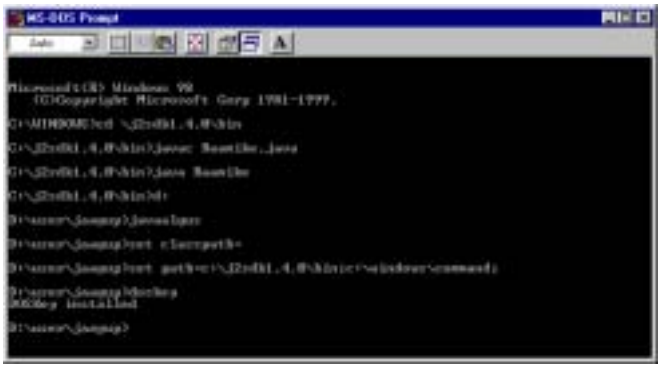


CTRL ja C korraga klaviatuuril alla ning saamegi oma töö lõpetamise oskuseta krati tegemised seisma

Niiviisi olemegi oma esimese programmi töösse saanud ja võiksime rahus üha uusi ja uusi näiteid ette võtta ning läbi proovida ja veidi oma kasuks muuta. Veidi kuiv aga tundub, kui me suudame oma programme vaid ühes kataloogis luua ja käivitada. Liiatigi hakkaksid loodud lähte- ja baitkoodid Java tööriistade kataloogi liialt risustama ning samuti ei õnnestu varsti suure hulga oma programmide hulgast enam sobivaid üles leida. Selle puhuks on olemas rohi - tuleb masinale seletada, et ta kompilaatori ka muudes kataloogides üles leiaks. Kui alltoodud kolmerealine fail salvestada nt. javaalgus.bat nime all c:\windows\command kataloogi, siis on meil võimalik alati soovi korral see käima tõmmata, et suudetaks igal pool kogu masina ulatuses me loodud lähtekoode kompileerida.



Kuna c:\windows\command on üldjuhul juba algselt otsinguteel, siis võime käsureal kirjutada javaalgus ning kolmerealine abiskript pannaksegi käima. Esimene rida set classpath= teatab, et kustutaks keskkonnamuutuja nimega classpath. Enamasti võib see käsk mõttetu olla, kuid kui mõne teise programmi installeerimise käigus on vastavanimeline keskkonnamuutuja seatud tolle programmi soovide järgi, siis meie Java intepretaator ei pruugi soovitud klasse üles leida. Järgmine on tähtsaim rida, kus otsinguteeks määratakse nii Java kompilaatori asukoht kui c:\windows\command. Viimane on kasulik, kuna seal asuvad mitmed abikäsklused, mida siis soovi korral käsurealt käivitada annab. Viimane rida on puhtalt mugavuse pärast. Kui doskey nimeline programm on käivitatud, siis jäetakse käsureale tipitud käsklused meelde ning ülespidi näitavale nooleklahvile vajutamisel saab eelmise rea juurde ilma, et seda peaks uuesti sisse tippima hakkama.



Ka progammi lähtekood tuleks siis kataloogi salvestada, kus on kavas edaspidi Java programme koostada ja katsetada. Siis veelkord kompileerida, et ka sinna kataloogi soovitud baitkood tekiks ning võibki oma koostatud programmi käima tõmmata.