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.

Esimen

```
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

```
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.

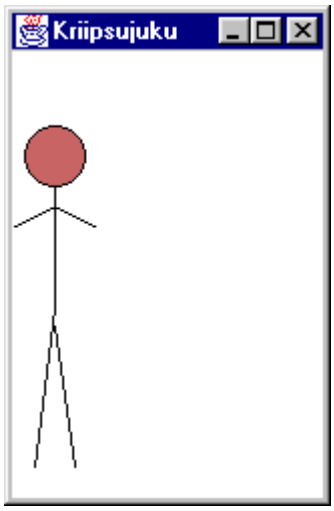
```
import java.awt.*;
public class Raamike5{
  public static void main(String argumendid[]) throws Exception{
    int punane=100, roheline=100, sinine=20;
    Frame f=new Frame("Esimene");
    f.setSize(200, 200);
    f.setBackground(new Color(punane, roheline, sinine));
    f.setVisible(true);
   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

Muutu v värv

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.

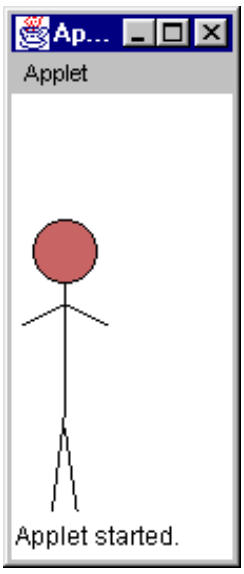
```
import java.awt.*;
import java.applet.Applet;

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.

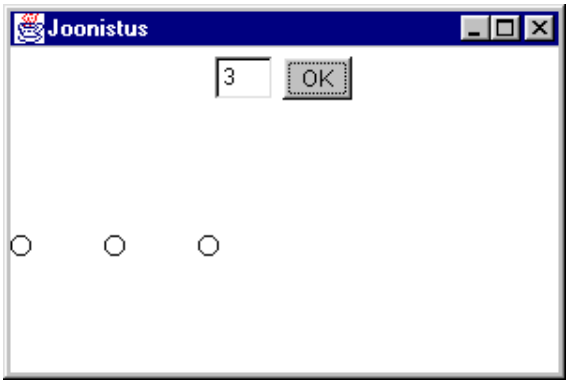


Kasutaja andmetele reageerimine

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);
  }
}
```

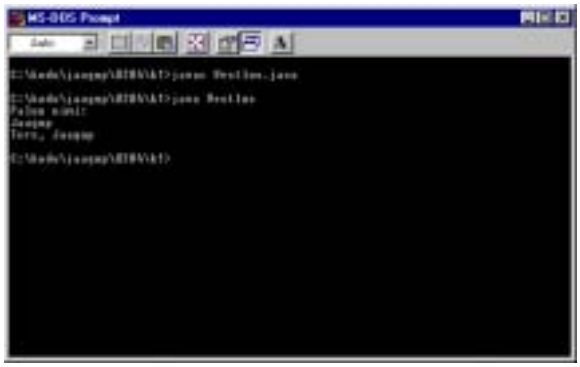
Vestlus tekstiekraanil

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 `integer`i 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',

`boolea`n - 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.