Graafikakomponendid

Graafikakomponendid aitavad programmeerijal hõlbustada programmi ja kasutaja suhtlemist. Samad võimalused saab luua ka joonistamisvahendite abil, kuid varem loodud komponentide puhul peab kasutaja nende käsitsemist õppima vaid korra, kasutada mõistab aga igal pool kus nad ette tulevad. Samuti piisab programmeerijal vaid paigutada komponent sobivale kohale, muuta tema omadusi vastavalt programmi vajadustele ning paluda programmil reageerida kasutaja tegevusele vastavalt. Komponendi sisemise ehituse üle on vaja pead murda vaid siis, kui soovida tema võimalustele midagi lisada, ise uut komponenti luua või olemasoleva vigu parandada.

Valmiskomponendid

Java keeles saab kasutada ligikaudu kümmet operatsioonisüsteemi juurde kuuluvat ning ligi seitsetkümmet java oma vahenditega loodud valmiskomponenti. Vajaduse korral aga saab neid täiendada ning ise juurde kombineerida ja luua. Operatsioonisüsteemi poolt pakutavad komponendid asuvad paketis `java.awt`, nad töötavad enamasti kiiremini ehk nõuavad arvutilt vähem ressursse. Nad näevad välja vastavalt operatsioonisüsteemile ning nende järgi vaadates ei pruugi väljagi paista, et rakendus on Java abil kirjutatud. Kui aga soovitakse (või on vajalik) et programmid erinevates keskkondades sarnaselt välja näeksid, siis tuleb kasutada "puhtaid" java komponente, mille värvus ning kuju operatsioonisüsteemist ei sõltu. Sellised valmiskomponendid asuvad enamjaolt klassis `javax.swing`.

Komponendid (nagu muudki objektid) saavad vastu võtta ning välja saata teateid. Saatja käivitab vastuvõtja meetodi. Komponendile teate saatmisel käivitatakse komponendi meetod (näiteks värvi muutmiseks). Teate saatmisel aga käivitab komponent (kui saatja) vastuvõtja meetodi. Näiteks kui nupule vajutamisel muutub tekst tekstikastis suuremaks, siis öeldakse, et nupp saatis tekstikastile teate. Sisuliselt tähendab see, et nupule vajutamise tulemusena käivitatakse tekstikasti meetod, millega saab muuta fondi suurust. Et nupul oleks võimalik tekstikasti meetodit käivitada, peab nupule olema antud osuti tekstikastile ning öeldud, et juhul kui nupule vajutatakse, tuleb tekstivälja vastav meetod käima panna.

Aken

Iseseisvalt saab ekraanil avada akent (Window) või raami (`Frame`, raamaken). Java keeles tähistab aken lihtsalt ekraanipiirkonda mille kasutamist saab programm juhtida, raami puhul on sel piirkonnal ümber raam ning üleval nupud suurusemuutmiseks ja sulgemiseks ning pealkirjariba. Enamasti kasutatakse kasutaja mugavuse huvides programmides raami, kuid näiteks täisekraaniefektide loomiseks on akna kasutamine paratamatu. Nende suuruse määramisel saab arvestada ekraani suurust. Dialog on eriline raam, mille avamisel võib jätta programmi töö seniks pooleli, kuni kasutaja on vastuse sisestanud. Sulgemisnupule vajutamine ei sule raami automaatselt. Vajutuse juurde on võimalik siduda programmilõik, kus kirjeldatakse programmi vajutusjärgset käitumist. Seal saab näiteks küsida, et "kas soovite salvestada" või muud taolist.

Tekstiväli

Tekstiväli (TextField) ning tekstiala (TextArea) võimaldavad ekraanile näidata ning kasutajal sisestada teksti. Esimesel neist võib olla vaid üks rida, tekstiala puhul aga saab määrata, mitut rida talle tahetakse. Värvi ning šrifti on võimalik muuta vaid kogu teksti juures korraga, keerulisemat kujundust nõudvate tekstide puhul tuleks kasutada `swing`i paketti kuuluvat `JEditorPane`. Redigeerimise jaoks aga lihttekstikomponentidest piisab. Lisaks komponendis oleva teksti küsimisele ja määramisele saab eraldi küsida ja määrata märgistatud teksti, samuti kursori asukohta.

Valik

Nimekiri (`List`) ning valik (`Choice`) lubavad kasutajal valida etteantud võimaluste seast. Nimekirjas on samaaegselt näha mitu rida. Kasutajal võib lasta märgistada samaaegselt ka mitu rida. Valiku puhul tuleb rippmenüü lahti vaid valimise ajaks, ülejäänud ajal on näha vaid üks, parasjagu valitud rida. Andmed ridadel on sõnedena, muul otstarbel kasutades tuleb neid vastavalt

intepreteerida. Et saaks valida näiteks pilte või värve, tuleb kasutada swingi komponente või siis vastav komponent ise kokku panna. Meetodite abil saab nendel komponentidel lisada ja eemaldada ridu, küsida, millise numbri või millise sisuga rea on kasutaja märgistanud. Saab lasta ka automaatselt rida märgistada. Veidi sarnane on ka `swing`i komponent `JTree`, kus kasutaja saab samuti ridu märgistada, seal aga on andmed paigutatud hierarhiliselt, puuna.

```
import java.applet.Applet;
import java.awt.List;
public class AwtList1 extends Applet{
  List list1=new List();
  public void init(){
    list1.add("Sinine");
    list1.add("Punane");
    list1.add("Kollane");
    list1.add("Valge");
    list1.add("Roheline");
    add(list1);
  }
}
```

```
import java.applet.Applet;
import java.awt.Choice;
public class AwtChoice1 extends Applet{
  Choice valik=new Choice();
  public void init(){
    valik.add("Sinine");
    valik.add("Punane");
    valik.add("Kollane");
    add(valik);
  }
}
```

Silt (`Label`) võimaldab endasse paigutada ühe rea teksti. Swingi analoogile saab panna ridu mitu ning soovi korral ka pilte sisse. Nupp (`Button`) reageerib hiirevajutusele. Ka temale on võimalik `awt`-variandis panna üks rida teksti, `swing`i juures aga enam kujundada.

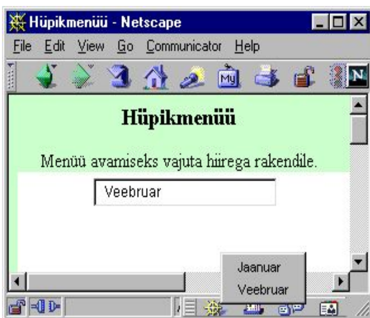
Märkeruudud

Märkeruut (`Checkbox`) laseb kasutajal valida kahe võimaluse vahel (märgitud või mitte). Raadionupp saab samuti olla kas sees või väljas, kuid neid kasutatakse enamasti juhul, kui saab valida vaid ühte mitme võimaluse seast. Selle eest hoolitsemiseks on vaja luua märkeruudugrupp (`CheckBoxGroup`), kes hoolitseb, et sinna gruppi lisatud nuppudest oleks vaid üks sisse lülitatud.

```
import java.awt.*;
import java.applet.Applet;
public class Raadionupud extends Applet{
  public Raadionupud(){
    CheckboxGroup grupp1=new CheckboxGroup();
    add(new Checkbox("Esimene", grupp1, false));
    add(new Checkbox("Teine",   grupp1, true));
    add(new Checkbox("Kolmas",  grupp1, false));
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Raadionupud");
    f.add(new Raadionupud());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

Menüü

Menüüriba (`MenuBar`) annab kinnitada raami külge. Temast saab panna hargnema menüüd ning neist omakorda alammenüüd. Hüpikmenüü (`Popup`) võib panna välja hüppama mis tahes koha pealt.




```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class Hypikmenyy extends Applet implements
ActionListener{
  TextField ta=new TextField(20);
  PopupMenu pm=new PopupMenu("Kuud");
  public Hypikmenyy(){
    MenuItem mi=new MenuItem("Jaanuar");
    mi.addActionListener(this);
    pm.add(mi);
    mi=new MenuItem("Veebruar");
    mi.addActionListener(this);
    pm.add(mi);
    add(pm);
    addMouseListener(
      new MouseAdapter(){
       public void mousePressed(MouseEvent e){
         pm.show(Hypikmenyy.this, e.getX(), e.getY());
       }
      }
    );
    add(ta);
  }
  public void actionPerformed(ActionEvent e){
    ta.setText(((MenuItem)e.getSource()).
                             getLabel()+"");
  }
}
```

Joonistamise jaoks on loodud lõuend (`Canvas`), kuid vajaduse korral saab ka teistele komponentidele (näiteks rakendile) joonistada. Lõuendi abil saame suhteliselt kergesti omale soovitud komponendi luua. Näiteks kui soovime pildiga nuppu, mille pilt vajutamise ajal muutuks, siis tuleks luua lõuendi alamklass. Sinna saab kirjeldada, millist pilti näidata nupu üleval oleku ajal ning millist siis, kui hiirega tema peale vajutatakse.

Kerimispaneel

Kui komponent on suurem kui tema jaoks eraldatud ekraanipind, siis saab ta paigutada `ScrollPane` sisse, mille tulemusena saab kerimisribade abil komponenti liigutada, vaadates teda läbi tema jaoks loodud "akna".

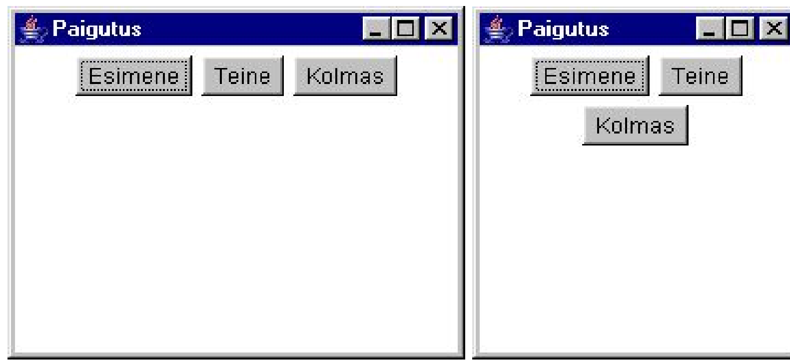
```
import java.awt.*;
import java.applet.Applet;
public class Paigutus10 extends Applet{
  public Paigutus10(){
    setLayout(new BorderLayout());
    Panel nupupaneel=new Panel(new GridLayout(10, 10));
    for(int i=0; i<10; i++){
      for(int j=0; j<10; j++){
        nupupaneel.add(new Button("Nupp "+i+""+j));
      }
    }
    ScrollPane sp=new ScrollPane();
    sp.add(nupupaneel);
    add(sp, BorderLayout.NORTH);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Paigutus");
    f.add(new Paigutus10());
    f.setSize(400, 200);
    f.setVisible(true);
  }
}
```

Paigutushaldurid

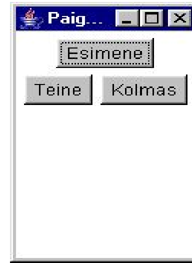
Graafikakomponent tuleb ekraanil näitamiseks paigutada konteinerisse (nt. raam, paneel, rakend). Konteineri sees komponente paigutada aitavad paigutushaldurid. Nad hoolitsevad, et näiteks raami suuruse muutmisel komponendid ekraanil mõistlikus suuruses näha jääksid. Kuna iga konteiner on samaaegselt ka komponent (ehk tema alamklass), siis saab ka konteinereid endid paigutushaldurite abil paigutada. Niimoodi paneele (või muid konteinereid) sobivalt üksteise sisse paigutades on võimalik saavutada peaaegu igasugune soovitud tulemus. Võimalik on ka täpselt ekraanipunktide järgi komponentide paigutus määrata, kuid see pole soovitav, sest näiteks raami suuruse või ekraani resolutsiooni muutmisel või uue komponendi lisamisel tuleks kogu kujundus uuesti kirjutada.

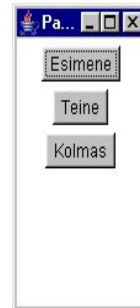
FlowLayout

Katsetamise ajal on lihtsaimaks paigustushalduriks `FlowLayout`. Seal pannakse komponendid üksteise järgi ritta ning kui rida täis saab, siis minnakse ekraanil järgmisesse ritta. Klassidel `Applet` ja `Panel` näiteks ongi `FlowLayout` vaikimisi paigutushalduriks. Sedasi ei pea paigutades arvestama komponetide suureustega. Haldur arvestab ise, et iga element nähtavale jääks, kui ekraanil vähegi ruumi on. Samas – vähegi keerukama kujunduse puhul ei saa siiski ainuüksi FlowLayouti oskustele lootma jääda.




```
import java.awt.*;
import java.applet.Applet;
public class Lihtpaigutus extends Applet{
  Button nupp1=new Button("Esimene");
  Button nupp2=new Button("Teine");
  Button nupp3=new Button("Kolmas");
  public Lihtpaigutus(){
    add(nupp1);
    add(nupp2);
    add(nupp3);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Paigutus");
    f.add(new Lihtpaigutus());
    f.setSize(250, 200);
    f.setVisible(true);
  }
}
```

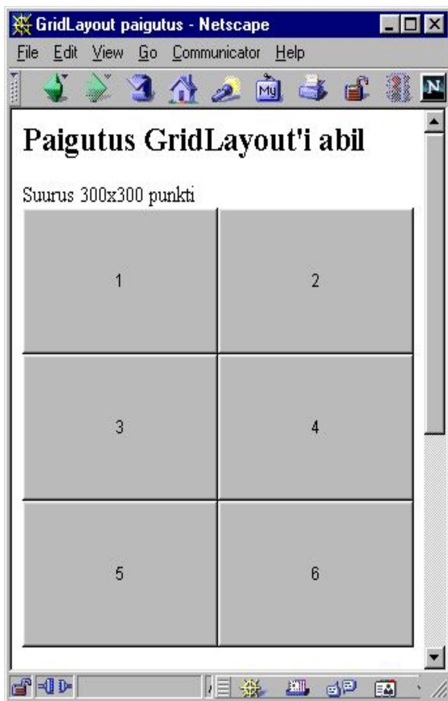




FlowLayout paigutuse näited.

GridLayout

`GridLayout` paigutuse puhul jagatakse konteineri juurde kuuluv piirkond ridadeks ja veergudeks, andes igale komponendile ühe lahtri. `GridBagLayout` on sarnane, kuid seal võib üks komponent katta ka mitu lahtrit. Swingi `BoxLayout` lubab komponendid panna kas ridadena või tulpadena, jättes nad loomulikku suurusse. GridLayouti on hea kasutada, kui on vajalik joondada komponendid tabelisse või muul puhul ühesuguse laiuse ja kõrgusega osadeks.

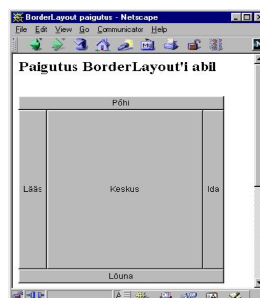



```
import java.awt.*;
import java.applet.Applet;

public class Paigutus2 extends Applet{
  public void init(){
    setLayout(new GridLayout(3, 2));
    add(new Button(" 1 "));
    add(new Button(" 2 "));
    add(new Button(" 3 "));
    add(new Button(" 4 "));
    add(new Button(" 5 "));
    add(new Button(" 6 "));
  }
}
```

BorderLayout

`BorderLayout` lubab komponendid paigutada nelja serva ning keskele. Servas olevad komponendid jäetakse servaga risti olevat mõõdet pidi nende loomulikku suurusse, keskele pandud komponent venitatakse kogu ülejäänud pinna ulatuses välja. Sugugi ei pea alati kõiki viit võimalust kasutama – piisab kui ühte serva on vaja panna nii, et komponent peaks terve servaala enda alla võtma ning muidu loomuliku suurusega välja paistma.

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

public class Paigutus1 extends Applet{
  public void init(){
    setLayout(new BorderLayout());
    add(new Button("Põhi"), BorderLayout.NORTH);
    add(new Button("Lõuna"), BorderLayout.SOUTH);
    add(new Button("Ida"), BorderLayout.EAST);
    add(new Button("Lääs"), BorderLayout.WEST);
    add(new Button("Keskus"),BorderLayout.CENTER);
  }
}
```



Paneel paigutamisel.

Paneel aitab kasutada oleva nelinurkse ala osadeks jaotada. Samuti nagu võib rakendile või aknale määrata paigutushalduri ning selle abil vastava konteineri sisse komponendid paigutada, saab nii olemasoleva ala jaotada paneeli abil komponetide vahel.

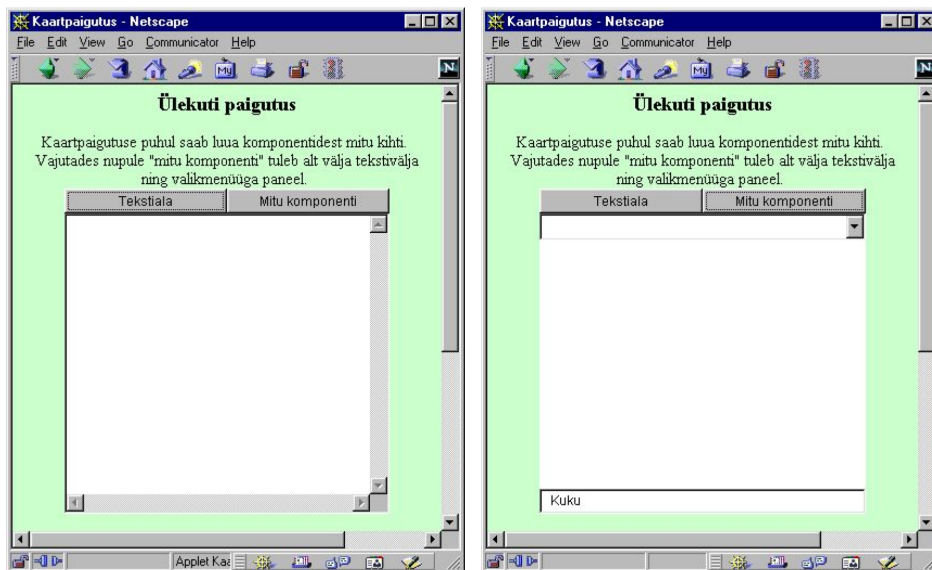
Järgnevas näites määratakse rakendi paigutushalduriks BorderLayout, mis lubab nagu ikka paigutada nii servadesse kui keskele. Kasutatakse vaid ülaserva, mis jagatakse GrigLayout-paigutusega paneeli abil üheks reaks ja kaheks veeruks ning siis lisatakse paneeli mõlemasse pessa nupp. Lõpuks pannakse paneel rakendi ülaserva.

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

public class Paneelpaigutus extends Applet{
  Button nupp1=new Button("Esimene");
  Button nupp2=new Button("Teine");
  public Paneelpaigutus(){
    setLayout(new BorderLayout());
    Panel p=new Panel(new GridLayout(1, 2));
    p.add(nupp1);
    p.add(nupp2);
    add(p, BorderLayout.NORTH);
  }
  public static void main(String[] argumendid){
    Frame f=new Frame();
    f.add(new Paneelpaigutus());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

Üldjuhul õnnestub paneele, BorderLayout'i ja GridLayout'i kombineerides kokku panna pea kõik võimalikud paigutusolukorrad, mis traditsioonilise „viisaka“ kujundusega rakenduse puhul ette tulevad.

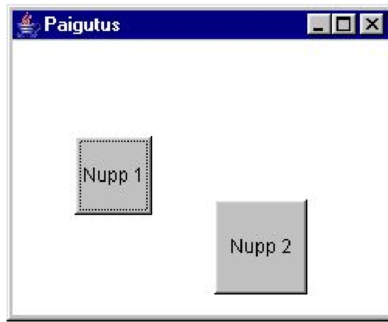
CardLayout

`CardLayout` võimaldab panna mitu kihti komponente üksteise peale, näidates välja pealmise kihi ning lubades kihte vahetada. Nõnda võib samal kohal vaadata kordamööda selliseid elemente nagu programmi looja parajasti tarvilikuks on pidanud. Swingi JTabbedPane eesmärk on sarnane, kuid seal on kohe automaatselt juurde lisatud võimalus kasutajal soovitud kihti välja kutsuda.



Absoluutsete koordinaatidega paigutus.

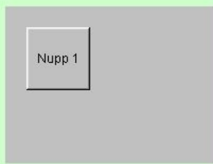
Pea alati tuleb esimese graafikakomponentidega tegelemise tunni jooksul kelleltki küsimus, et „kuidas ma saan täpselt määrata tekstiala/nupu koordinaadid“. On ju nii Visual Basicus, Multimedia Toolbook'is kui mõnes muuski keeles võimalik ekraanipunktide või muude numbrite abil määrata, kus miski komponent asub. Ning seletus, et „Java programmide juures peetakse sellist paigutust ebaviisakaks“ ei tundu kuigi usutavana. Võimalus on täiesti olemas, nii nagu järgmisest näitest paista võib. Koordinaatide järgi paigutatakse siis, kui paigutushaldur puudub, ehk selleks on seatud tühi osuti null. Enne konteineri sisse paigutamist määratakse komponentidele suurused ning siis lisamisel nad satuvadki määratud kohtadesse.




```
import java.awt.*;
import java.applet.Applet;
public class Paigutus9 extends Applet{
  public Paigutus9(){
    setLayout(null);
    Button nupp1=new Button("Nupp 1");
    Button nupp2=new Button("Nupp 2");
    nupp1.setBounds(40, 60, 50, 50);
    nupp2.setBounds(130, 100, 60, 60);
    add(nupp1);
    add(nupp2);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Paigutus");
    f.add(new Paigutus9());
    f.setSize(250, 200);
    f.setVisible(true);
  }
}
```

Omaloodud paigutushaldur

Kasutatud haldurid pole muud kui tavalised vastavate oskustega Java klassid. Kui mingil põhjusel selgub, et soovitakse täiesti erilist paigutust, siis võib sellise omale kirjutada. Tuleb lihtsalt koostada klassile java.awt.LayoutManager oma alamklass ning seal mõned meetodid üle katta. Tähtsam neist layoutContainer, mille sees igale konteineris asuvale komponendile määrata tema asukoht. Siin on piirdutud lihtsaima näitega, kus eeldatakse et tegemist on vaid ühe komponendiga ning sellelegi määratakse alati samad koordinaadid. Põhjalikuma paigutamise puhul aga tuleb arvestada konteinerile eraldatud ruumi, komponentide soovitud suurus ning muudki võimaluste järgi.




```
import java.awt.*;
import java.applet.Applet;
public class Paigutus11 extends Applet{
  public Paigutus11(){
    setLayout(new Ruutpaigutus());
    add(new Button("Nupp 1"));
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Paigutus");
    f.add(new Paigutus11());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```

```
import java.awt.*;
class Ruutpaigutus implements LayoutManager{
  public void layoutContainer(Container kest){
    if(kest.getComponentCount()>0){
      Component c=kest.getComponent(0);
      c.setBounds(20, 20, 60, 60);
    }
  }
  public void addLayoutComponent(String nimi, Component c){}
  public void removeLayoutComponent(Component c){}
  public Dimension preferredLayoutSize(Container kest){
    return new Dimension(100, 100);
  }
  public Dimension minimumLayoutSize(Container kest){
    return preferredLayoutSize(kest);
  }
}
```

Kuularid

Sündmuste tulemusena millegi juhtumiseks tuleb sündmuse registreerijalt (ehk allikalt) saata teade sündmuse vastuvõtjale. Ühe sündmuse (näiteks nupuvajutuse) teadet saab saata mitmele vastuvõtjale, muuhulgas ka iseendale. Et allikas vastuvõtjale teate saadaks, selleks peab olema vastuvõtja end allika juures registreerinud vastavat tüüpi sündmuse kuulajaks. Kuulajaks saab end registreerida isend, kelle klass realiseerib vastava sündmusetüübi kuulamiseks loodud liidest.

Näiteks hiirevajutuse kuulamiseks peab klass realiseerima liidest MouseListener. Liidese realiseerimine aga tähendab seda, et klassis oleks kirjeldatud kõigi liideses deklareeritud meetodite käivitamisel tehtavad tegevused. Näiteks MouseListeneri realiseerijal peavad olema kõik liideses kirjeldatud meetodid: nii hiire vajutamise, ülestõstmise, komponendi piirkonda sisenemise kui komponendist väljumise kohta. Kui soovitakse, et mõnel juhul ei reageeritaks, siis tuleb seegi arvutile selgeks teha, s.t. vastava meetodi sisuks kirjutada tühjad sulud.

```
import java.awt.*;
import java.awt.event.*;
public class Hiir1{
  public static void main(String argumendid[]){
    Frame f=new Frame("Hiireraam");
    f.setSize(200, 300);
    f.setVisible(true);
    f.addMouseListener(new HiireKuular1(f));
  }
}

class HiireKuular1 implements MouseListener{
  Frame raam;
  public HiireKuular1(Frame uusraam){
    raam=uusraam;
  }
  public void mousePressed(MouseEvent e){
    //suurendab raami laiust 10 ühiku võrra
    Dimension suurus=raam.getSize();
    raam.setSize(suurus.width+10, suurus.height);
  }
  public void mouseReleased(MouseEvent e){}
  public void mouseClicked(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
}
```

Et tühje sulge peaks vähem kirjutama, selleks on loodud adapterklassid liideste jaoks, mis defineerivad enam kui ühe meetodi. Vastava liidese adapterklass realiseerib liidest, jättes meetodi kehaks tühjad sulud, s.t. tehes meetodi käivitamisel mitte midagi. Kui me nüüd oma kuulari loome vastava adapteri alamklassina, siis piisab meil vaid üle katta meile vajalik meetod. Näiteks oma hiirekuulari loomisel katame üle vaid hiire vajutamisel käivitatava meetodi. Muude hiiresündmuste puhul kasutatakse adapterklassi meetodeid ning kuna seal midagi teha ei paluta, siis meile näib, nagu ei reageeritakski nendele sündmustele.

```
import java.awt.*;
import java.awt.event.*;
public class Hiir2{
  public static void main(String argumendid[]){
    Frame f=new Frame("Hiireraam");
    f.setSize(200, 300);
    f.setVisible(true);
```

```
    f.addMouseListener(new HiireKuular2(f));
  }
}

class HiireKuular2 extends MouseAdapter{
  Frame raam;
  public HiireKuular2(Frame uusraam){
     raam=uusraam;
  }
  public void mousePressed(MouseEvent e){
    Dimension suurus=raam.getSize();
    raam.setSize(suurus.width+10, suurus.height);
  }
}
```

Komponent võib ka ise saata teateid enese juures juhtunud sündmustest ning neid siis töödelda. Sel juhul pole eraldi klassi (ega isendit) kuulari jaoks vaja luua. Piisab vaid, kui komponendi alamklass ise realiseerib vastava sündmuse kuulamiseks vajalikku liidest. Liidese realiseerimiseks peab aga meetodi keha olema kõikidel liideses kirjeldatud meetoditel. Sellest siin need tühjad meetodid, et programm teaks, et näiteks hiire sisenemise korral ei tule tal midagi teha.

```
import java.awt.*;
import java.awt.event.*;
public class Hiir3 extends Frame implements MouseListener{
   public Hiir3(){
     addMouseListener(this);
   }
   public void mousePressed(MouseEvent e){
     Dimension suurus=getSize();
     setSize(suurus.width+10, suurus.height);
   }
   public void mouseReleased(MouseEvent e){}
   public void mouseClicked(MouseEvent e){}
   public void mouseEntered(MouseEvent e){}
   public void mouseExited(MouseEvent e){}

   public static void main(String argumendid[]){
     Frame f=new Hiir3();
     f.setSize(200, 300);
     f.setVisible(true);
   }
}
```

Kui aga ka komponendi alamklassis tahetakse kasutada adapteri võimalusi ning leitakse, et tühjade meetodite kirjutamine on aja ja ruumi raiskamine, siis tuleb veidi keerulisemalt hakkama saada. Kuna java keeles on võimalik pärida ainult ühelt eellaselt, siis üheaegselt nii Frame kui adapteri klassis paiknevat koodi pole võimalik pärida. Selgem ning kindlam on kirjutada eraldi adapteri alamklass ning ta konstruktorile anda parameetriks osuti komponendile nagu esimestes näidetes. Siis saab selle osuti kaudu komponendi tegevust juhtida. Versioonist 1.1 alates aga loodi võimalus sama probleemi ka lühemalt lahendada. Alljärgnevas näites luuakse adaptrile nimetu alamklass, kus kaetakse üle tema meetod, siis luuakse isend ning pannakse ta teateid kuulama. Meetod windowClosing kutsutakse välja siis, kui kasutaja vajutab raami sulgemise nupule. Selle tulemusena lõpetatakse programmi töö (System.exit).

```
import java.awt.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.event.*;
public class Raam4 extends Frame {
   public Raam4(){
     addWindowListener(
       new WindowAdapter(){
        public void windowClosing(WindowEvent e){
          System.exit(0);
        }
       }
     );
   }

   public static void main(String argumendid[]){
     Frame f=new Raam4();
     f.setSize(200, 300);
     f.setVisible(true);
```

```
    }
}
```

Tekstikuular

Tekstikuulari liideses on kirjeldatud vaid üks meetod: textValueChanged. See meetod käivitatakse kuulajatel siis, kui allika (ehk tekstivälja või tekstiala) sees olev tekst on muutunud. Siin näites on rakend ühtlasi esimese tekstiala kuulariks. Kui esimeses tekstialas ehk allikas teksti muudetakse, siis selle tulemusena pannakse teise tekstiala sisuks esimese sisu koopia, kusjuures kõik tähed muudetakse väiketähtedeks.

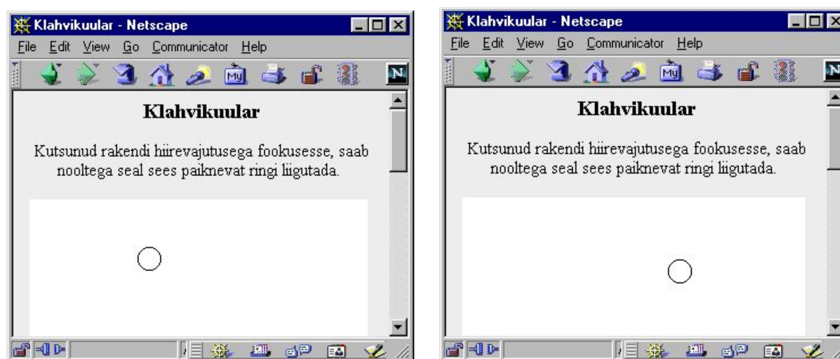
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class Tekstikuular extends Applet
       implements TextListener{
  TextArea tekstiala1=new TextArea(3, 30);
  TextArea tekstiala2=new TextArea(3, 30);
  public Tekstikuular(){
    add(tekstiala1);
    add(tekstiala2);
    tekstiala1.addTextListener(this);
  }
  public void textValueChanged(TextEvent e){
    tekstiala2.setText(tekstiala1.getText().toLowerCase());
  }
}
```

Klahvikuular

Klahvikuular peab realiseerima liidest KeyListener ning temale saadetakse teated keyPressed, keyReleased ning keyTyped. Siin näites pööratakse tähelepanu vaid allavajutamise juhule. KeyEvent isendi meetod getKeyCode annab tulemuseks täisarvu, mis vastab klahvi koodile. Siis kontrollitakse, millise klahviga on tegemist ning toimitakse vastavalt sellele. Konstant KeyEvent.VK_LEFT tähendab näiteks noolt vasakule. Käsklus repaint() keyPressed meetodi lõpus käsib ekraani üle joonistada, s.t. kustutab vaja joonise ning käivitab siis eraldi lõimena meetodi paint. Sellise toimimise korral on ring õiges kohas ekraanil ka näiteks pärast rakendi nihutamist või teiste programmide alla sattumist.

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

public class Klahvikuular2 extends Applet implements KeyListener{
  int x=100, y=100;
  public Klahvikuular2(){
    addKeyListener(this);
  }
  public void paint(Graphics g){
```

```
    g.drawOval(x-10, y-10, 20, 20);
  }
  public void keyPressed(KeyEvent e){
    int kood=e.getKeyCode();
    if(kood==KeyEvent.VK_LEFT)x--;
    if(kood==KeyEvent.VK_RIGHT)x++;
    if(kood==KeyEvent.VK_UP)y--;
    if(kood==KeyEvent.VK_DOWN)y++;
    repaint();
  }

  public void keyReleased(KeyEvent e){}

  public void keyTyped(KeyEvent e){}


  public static void main(String argumendid[]){
    Frame f=new Frame("Klahvikuular");
    f.add(new Klahvikuular2());
    f.setSize(300, 300);
    f.setVisible(true);
  }

}
```



Fookusekuular

Fookusekuular registreerib fookuse saabumise ning eemaldumise teated, s.t. nende sündmuste toimumisel käivitatakse vastavad meetodid. Siin näites lihtsalt omistatakse tõeväärtusmuutujale väärtus tõene või väär vastavalt sellele, kas rakend on fookuses või mitte. Seejärel palutakse ekraan üle joonistada. Ülejoonistamine sõltub muutuja väärtusest. Kui komponent on fookuses, siis joonistatakse esiplaanivärviga rakend üle. Kui aga komponent fookuses pole, siis jäetakse pind värvimata.

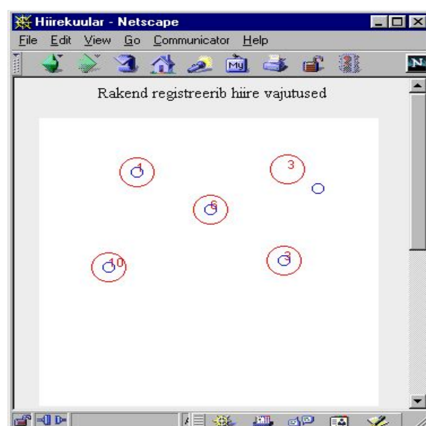
```
import java.applet.Applet;
import java.awt.event.*;
import java.awt.Graphics;
public class Fookusekuular extends Applet implements FocusListener{
  boolean fookuses=true;
  public Fookusekuular(){
    addFocusListener(this);
  }

  public void paint(Graphics g){
    if(fookuses){
      g.drawRect(0, 0, getSize().width-1, getSize().height-1);
    }
  }
  public void focusGained(FocusEvent e){
    fookuses=true;
    repaint();
  }
  public void focusLost(FocusEvent e){
    fookuses=false;
    repaint();
  }

  public static void main(String[] argumendid){
    Frame f=new Frame("Fookuseraam");
    f.add(new Fookusekuular());
    f.setSize(200, 200);
```

```
    f.setVisible(true);
  }
}
```

Hiirekuular

Hiire vajutuste ning hiire liikumise registreerimiseks on kummagi jaoks omaette kuular. Nii on kasulik seetõttu, et hiire liigutamisel tuleb teateid tunduvalt rohkem kui vajutamisel. Liigutamisel tuleb registreerida piisavalt palju punkte, et nende abil oleks võimalik kõverjoonelist liikumisteed ette kujutada. Järgnevas näites joonistatakse hiire vajutamise kohale punane ring, ülestõstmise kohale sinine ning kuid hiir väljub komponendi piirkonnast, siis joonistatakse viimase pind valge värviga üle. Vajutamise juures kirjutatakse ka, mitu korda selles punktis on hiireklahvi lühikeste vahedega vajutatud. Andmed selle kohta saab hiirevajutussündmuse puhul väljakutsutava meetodi MouseEvent tüüpi parameetrist.

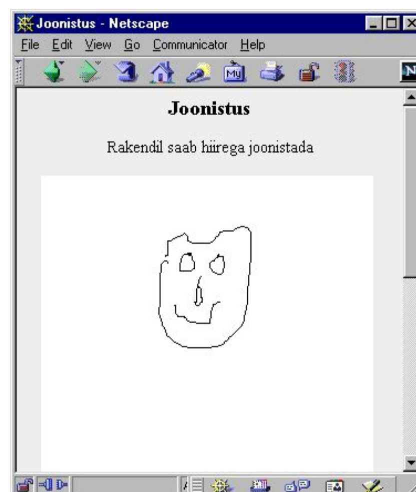
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class Hiirekuular extends Applet implements MouseListener{
  public Hiirekuular(){
    addMouseListener(this);
  }
  public void mousePressed(MouseEvent e){
    Graphics g=getGraphics();
    g.setColor(Color.white);
    g.fillRect(e.getX()-15, e.getY()-15, 30, 30);
    g.setColor(Color.red);
    g.drawOval(e.getX()-15, e.getY()-15, 30, 30);
    g.drawString(e.getClickCount()+"",e.getX(), e.getY());
  }
  public void mouseReleased(MouseEvent e){
    Graphics g=getGraphics();
    g.setColor(Color.blue);
    g.drawOval(e.getX()-5, e.getY()-5, 10, 10);
  }
  public void mouseExited(MouseEvent e){
    Graphics g=getGraphics();
    g.setColor(Color.white);
    g.fillRect(0, 0, getSize().width, getSize().height);
  }
  public void mouseEntered(MouseEvent e){}
  public void mouseClicked(MouseEvent e){}
}
```



Hiire liikumise kuular

Pildi joonistamiseks on vaja registreerida nii hiire vajutusi kui liikumist. Et pääseda hiirekuulari praeguses programmis mitte vaja minevate sündmuste töötlemisest, selleks loon oma kuulari klassi MouseAdapter alamklassina, mis samaaegselt realiseerib MouseMotionListener liidest (meeldetuletuseks: korraga võib laiendada vaid ühte klassi, kuid liideseid realiseerida kuitahes mitu). Hiire allavajutamisel jäetakse meelde vajutuse asukoht. Liigutamisel aga tõmmatakse joon eelmise punkti ning hetkel hiire asukohaks oleva punkti vahele.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class Joonistus extends Applet{
  public Joonistus(){
    JoonistuseKuular kuular=new JoonistuseKuular(this);
    addMouseListener(kuular);
    addMouseMotionListener(kuular);
  }
}
class JoonistuseKuular extends MouseAdapter
      implements MouseMotionListener{
  Joonistus peremees;
  int vanax, vanay, uusx, uusy;
  public JoonistuseKuular(Joonistus j){
    peremees=j;
  }
  public void mousePressed(MouseEvent e){
    vanax=e.getX();
    vanay=e.getY();
  }
  public void mouseDragged(MouseEvent e){
    uusx=e.getX();
    uusy=e.getY();
    Graphics g=peremees.getGraphics();
    g.drawLine(vanax, vanay, uusx, uusy);
    vanax=uusx;
    vanay=uusy;
  }
  public void mouseMoved(MouseEvent e){}
}
```



Järgnev näide erineb eelmisest selle poolest, et pilt joonistatakse enne mällu ning alles sealt ekraanile. Sellisel juhul saab pildi ka pärast selle ekraanilt kadumist uuesti sinna tekitada. Pilt luuakse mällu paint-meetodi esmakordsel väljakutsel. Meetod update on üle kaetud, et joonistamine ilusamini välja näeks. Vaikimisi update enne joonistab komponendi taustavärviga üle ning alles siis joonistab sinna peale pildi. Kui aga paluda update'l kohe pilt joonistada, siis aitab see vältida vilkumist.

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

```
public class Joonistus2 extends Applet{
  Image pilt;
  public Joonistus2(){
    Joonistuse2Kuular kuular=new Joonistuse2Kuular(this);
    addMouseListener(kuular);
    addMouseMotionListener(kuular);
  }
  public void paint(Graphics g){
    if(pilt==null)pilt=createImage(getSize().height, getSize().width);
    g.drawImage(pilt, 0, 0, this);
  }
  public void update(Graphics g){
    paint(g);
  }
  public static void main(String[] argumendid){
     Frame f=new Frame("Joonistus");
     f.add(new Joonistus2());
     f.setSize(300, 300);
     f.setVisible(true);
  }
}

class Joonistuse2Kuular extends MouseAdapter
      implements MouseMotionListener{
  Joonistus2 peremees;
  int vanax, vanay, uusx, uusy;
  public Joonistuse2Kuular(Joonistus2 j){
    peremees=j;
  }
  public void mousePressed(MouseEvent e){
    vanax=e.getX();
    vanay=e.getY();
  }
  public void mouseDragged(MouseEvent e){
    uusx=e.getX();
    uusy=e.getY();
    Graphics g=peremees.pilt.getGraphics();
    g.drawLine(vanax, vanay, uusx, uusy);
    vanax=uusx;
    vanay=uusy;
    peremees.repaint();
  }
  public void mouseMoved(MouseEvent e){}
}
```

Graafikakomponendi loomine

Omatehtud jooniste ning komponentide aluseks sobib lõuend (Canvas). Ta sobib lihtsaks joonistamiseks, kuid samas saab ta panna ka teateid vastu võtma, s.t. näiteks hiirevajutusele reageerima. Kui oled kord komponendi loonud, siis saad seda tervikuna kasutada seal kus parajasti vaja on. Kui oled komponendi tööga rahul, siis võid tarvitada teda ilma sisemisse ehitusse süvenemata. Alati ei pea komponendi loomisel kõike otsast tegema, vaid võib kasutada juba varem olemas olevaid tükke. Samuti võib luua varemvalmistatud komponendile alamklassi ning seal soovitud meetodid muuta. Nii võib kerge vaevaga lisada tekstialale võimaluse, et ta väljastaks oma sees oleva ridade arvu või paneks lisatavatele ridadele tühikud ette. Kui aga tahetakse sündmused ja kujundused täiesti ise määrata, siis tuleb aluseks võtta tühi pind ehk lõuend.

Hulknurk

Siin näites joonistatakse lõuendile soovitud nurkade arvuga hulknurk. Nurkade arvu saavad väljapoolsed muuta vaid meetodi abil. Iga muutmisega kaasneb uus joonistamine.

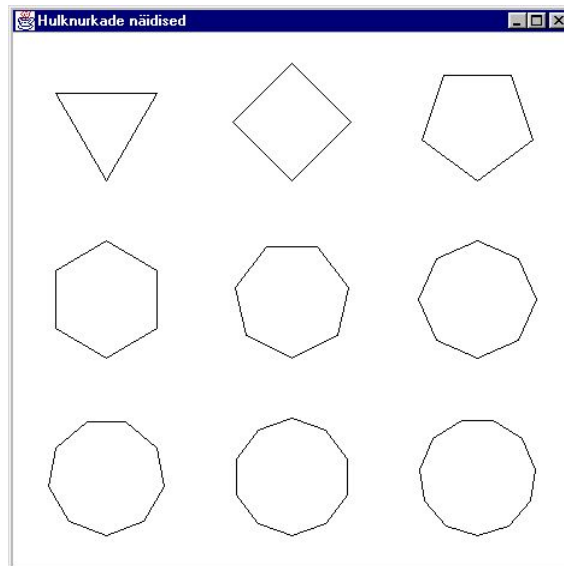
```
import java.awt.*;
public class Nurgad extends Canvas{
  protected int nurkadearv;
  public Nurgad(){
    nurkadearv=3;
  }
  public Nurgad(int uusarv){
    nurkadearv=uusarv;
  }

  public void muudaNurkadeArv(int uusarv){
```

```
    nurkadearv=uusarv;
    repaint();
  }

  public void paint(Graphics g){
    int korgus=getSize().height;
    int laius=getSize().width;
    double nurgavahe=2*Math.PI/(double)nurkadearv;
    int raadius=Math.min(korgus, laius)/3;
    int keskx=laius/2;
    int kesky=korgus/2;
    int vanax=keskx;
    int vanay=kesky+raadius;
    int uusx, uusy;
    for(int i=1; i<=nurkadearv; i++){
      uusx=keskx+(int)(raadius*Math.sin(i*nurgavahe));
      uusy=kesky+(int)(raadius*Math.cos(i*nurgavahe));
      g.drawLine(vanax, vanay, uusx, uusy);
      vanax=uusx;
      vanay=uusy;
    }
  }
}
```

Komponendi kasutamine

Loodud komponenti saab kasutada seal, kus vaja hulknurki joonistada. Siin näites pannakse rakendi ekraanile üheksa hulknurka, nurkade arvuga kolmest üheteistkümneni.

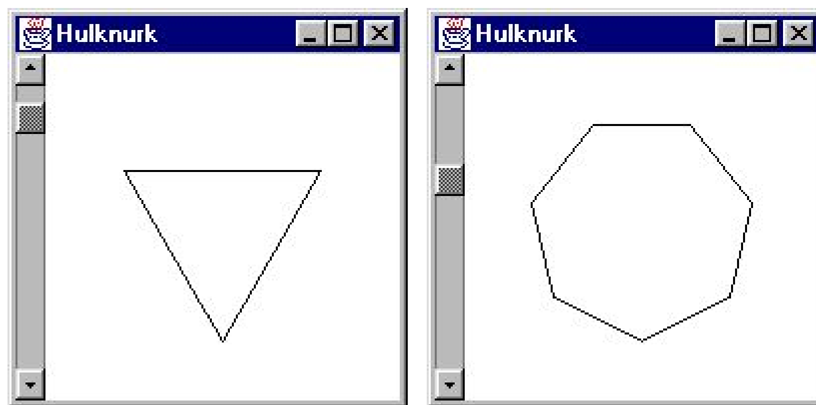
```
import java.applet.Applet;
import java.awt.Frame;
public class Nurgarakend extends Applet{
  public Nurgarakend(){
    setLayout(new java.awt.GridLayout(3, 3));
    for(int nr=3; nr<12; nr++){
      add(new Nurgad(nr));
    }
  }
  public static void main(String[] argumendid){
    Frame f=new Frame("Hulknurkade näidised");
    f.add(new Nurgarakend());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```



Samuti saab loodud komponendi abil lasta kasutajal valida, mitme nurgaga hulknurka soovib. Selleks panin rakendile kerimisriba ning loodud komponendi. Rakendi panin kerimisriba kuulajaks (AdjustmentListener). Kui kerimisriba määratud koha väärtust muudetakse, siis saadetakse uus väärtus rakendile, kes selle omakorda saadab hulknurka joonistavale komponendile.

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

```
import java.awt.event.*;
public class Nurgarakend2 extends Applet implements AdjustmentListener{
  Nurgad ng=new Nurgad();
  Scrollbar sb=new Scrollbar(
    Scrollbar.VERTICAL, 3, 2, 2, 20
  );
  public Nurgarakend2(){
    setLayout(new BorderLayout());
    add(sb, BorderLayout.WEST);
    add(ng, BorderLayout.CENTER);
    sb.addAdjustmentListener(this);
  }

  public void adjustmentValueChanged(AdjustmentEvent e){
    ng.muudaNurkadeArv(e.getValue());
  }

  public static void main(String[] argumendid){
    Frame f=new Frame("Hulknurk");
    f.add(new Nurgarakend2());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```



Andmete ülekanne

Kopeerimine

Programmide sees ning ka programmide vahel kasutatakse andmete vahetamiseks mälupuhvrit (`clipboard`). Sinna saab andmeid paigutada ning sealt vajadusel kopeerida. Kui puhver on operatsioonisüsteemi juures ning sinna pääsevad ligi mitmed programmid, siis saab selle abil nende vahel andmeid vahetada. Et andmeid sobiksid, peavad osapooled aru saama andmete formaadist. Kõige lihtsamaks formaadiks on lihtne tekst, kuid selle kaudu saab kõike vahetada, mida on võimalik baitideks muundada.

Järgnevast näitest suurem osa kulub kujundusele, kus luuakse raam, pannakse sinna sisse tekstiväli, tekstiala ning menüü. Kopeerimine ja kleepimine asub meetodis actionPerformed, mis käivitub menüüst valiku tegemisel. Vastavalt menüürea nimele käivitatakse tegevus. Meetodi algul küsitakse juurdepääs operatsioonisüsteemi mälupuhvrile.

```
Clipboard malu = getToolkit().getSystemClipboard();
```

Kui antakse korraldus Kopeeri, siis võetakse tekstiväljast tekst ning muudetakse `StringSelection`'iks. Viimatinimetatud klassis on tekst kujul, mida saab mälupuhvrisse panna ning mida teised programmid lugeda mõistavad.

```
StringSelection ss = new StringSelection(tf.getText());
```

Seejärel öeldakse, et mingu see tekst mälupuhvrisse. Meetodi teiseks parameetriks on `ClipboardOwner`, kellele saadetakse teade puhvri sisu vahetumisest. Siin näites tegelikult nende teadetega midagi ette ei võeta.

```
      clipboard.setContents(ss, ss);
```

Kleepimise puhul võetakse teade mälupuhvrist välja ning pannakse ta tekstialasse. Puhvrist

saadakse andmed kätte esialgu tüübina `Transferable`, mis tuleb seejärel sobivaks tüübiks muundada. Transferable käest on võimalik küsida millisel kujul ta andmeid kannab. Siin aga eeldame, et tegemist on sõnega ning palume tal sellisena need andmed ka välja anda.

```
Transferable andmed = clipboard.getContents(this);
String s = (String)(andmed.getTransferData(DataFlavor.stringFlavor));
```

Tulemuseks on programm, mille abil saab andmeid tekstina programmide vahel vahetada. Selgitust vajab ka ehk menüü loomine. Algul luuakse menüüriba (`MenuBar`), sinna külge pannakse menüü(d) (`Menu`) ning viimasesse menüüread (`MenuItem`). Menüüridadele öeldakse (`addActionListener`), kellele nende peale vajutamisel teateid saata.

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.datatransfer.*;

public class Tekstikopeering extends Frame implements ActionListener {
  TextField tf=new TextField();
  TextArea ta=new TextArea();
  public Tekstikopeering() {
    super("Kopeerimine");
    add(tf, BorderLayout.NORTH);
    add(ta, BorderLayout.CENTER);
    MenuBar mb = new MenuBar();
    mb.add(looMenyy());
    setMenuBar(mb);
  }

Menu looMenyy() {
    Menu m = new Menu("Parandused");
    MenuItem mi = new MenuItem("Lõika");
    mi.addActionListener(this);
    m.add(mi);
    mi = new MenuItem("Kopeeri");
    mi.addActionListener(this);
    m.add(mi);
    mi = new MenuItem("Kleebi");
    mi.addActionListener(this);
    m.add(mi);
    m.addSeparator();
    mi = new MenuItem("Puhasta");
    mi.addActionListener(this);
    m.add(mi);
    return m;
  }
```

```
  public void actionPerformed (ActionEvent e) {
    Clipboard malu = getToolkit().getSystemClipboard();
    String kask = e.getActionCommand();
    if (kask.equals("Kopeeri")) {
       StringSelection data = new StringSelection(tf.getText());
       malu.setContents(data, data);
    } else if (kask.equals("Puhasta")) {
       tf.setText("");
    } else if (kask.equals("Kleebi")) {
       Transferable andmed = malu.getContents(this);
       String s;
       try {
        s = (String)(andmed.getTransferData(DataFlavor.stringFlavor));
       } catch (Exception viga) {
        s = viga.getMessage();
       }
       ta.setText(s);
    } else if (kask.equals("Lõika")) {
       StringSelection ss = new StringSelection(tf.getText());
       malu.setContents(ss, ss);
       tf.setText("");
    }
  }
  public static void main (String argumendid[]) {
    Frame f=new Tekstikopeering();
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```

Andmete vedamine (Drag and Drop)

Lisaks mälupuhvri abil kopeerimisele püütakse andmete ülekannet ka hiirega vedamise abil kasutajale intuitiivsemaks muuta. Enamik meist on tõenäoliselt hiirega Word'i redaktoris sõnu lauses ringi tõstnud või Windows Exploreri aknas faile ühest kataloogist teise lohistanud. Andmete allikaks või suudmeks saab määrata ükskõik millise komponendi, kes on võimeline hiire teateid vastu võtma. Komponendile tuleb määrata sündmus, mille peale ta end andmete allikaks loeb. Sageli on selleks näiteks hiire vajutus ning lohisemine tema peal vähemalt viie punkti ulatuses. Kui andmed on kord liikuma pandud, saab nende "käekäigu" üle teateid andmeveo kuulari abil, kes teatab hiire sattumisest võimaliku vastuvõtja alasse.

Andmete vastuvõtjalgi on kuular. Tema saab teateid enesele sattunud andmetega varustatud hiirest ning on võimeline vastavalt nendele teadetele käituma. Ta saab võrrelda pakutavat andmete tüüpi enese poolt vastu võtta suudetavate andmetüüpidega ning sellest kasutajale teadma andma. Kui hiire klahv lastakse vastuvõtja kohal lahti, siis saabub teade drop ning andmed võib vastu võtta.

Siin näites luuakse raam kolme sildiga. Ülemised kaks on andmete allikaks ning nende siltide pealt vedama hakkamisel kaasneb andmetena sildi peal olev kiri. Kolmas silt on vastuvõtja. Kui selle peal vabastatakse andmeid kandva kursoriga hiire klahv, siis jääb saabunud tekst sildi sisse.

Nii allika kui suudme olen loonud sildi alamklassina. AndmeveoAlguseKuularis on kirjas, mida tuleb teha, kui `DragSource` poolt loodud `DefaultDragGestureRecognizer` on märganud, et sildi pealt hakatakse andmeid vedama. Sel puhul võetakse sildi tekst, muutetakse ta `Transferable` tingimustele vastavaks `StringSelection`'iks, et teda saaks üle kanda ning siis käivitatakse vedu käsuga `startDrag`. Parameetriteks on vedamise ajal näidatav kursor, kantavad andmed ning kuular, kellele saadetakse teated andmetega teel toimuva kohta.

Suudmel on isend `DropTarget`, kelle poolt loodud `AndmeteSaabumiseKuular` saabuvate andmetega tegeleb. Kui suudme kohal lastakse lahti andmehulk, saab selle tüüpide sobivuse korral vastu võtta. Esialgu küsitakse meetodi parameetrilt `Transferable`-tüüpi andmed, sealt oodatud kujul `Object`ina ning lõpuks tuleb nad kasutatavale kujule muudada. Siis võib nendega edasi toimida, siin näites andmete sees paiknev tekst oma sildile paigutada.

```
import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;

public class Allikas extends Label {
  private DragSource dragSource;
  public Allikas(String s) {
    setText(s);
    dragSource = DragSource.getDefaultDragSource();
    dragSource.createDefaultDragGestureRecognizer(
      this, DnDConstants.ACTION_COPY,  new AndmeveoAlguseKuular());
    }

  class AndmeveoAlguseKuular implements DragGestureListener {
    public void dragGestureRecognized(DragGestureEvent e) {
      Transferable andmed = new StringSelection( getText() );
        e.startDrag(DragSource.DefaultCopyDrop,
          andmed, new AndmeveoKuular());
    }
  }


  class AndmeveoKuular implements DragSourceListener {
    public void dragDropEnd(DragSourceDropEvent e) { }
    public void dragEnter(DragSourceDragEvent e) {  }
    public void dragOver(DragSourceDragEvent e) {  }
    public void dragExit(DragSourceEvent e) { }
    public void dropActionChanged (DragSourceDragEvent e) { }
  }
}
```

```
import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;
```

```
import java.io.*;
public class Suue extends Label {
  private DropTarget dropTarget;
  public Suue(String s) {
    setText(s);
    dropTarget = new DropTarget(this,
      DnDConstants.ACTION_COPY,
      new AndmeteSaabumiseKuular(), true);
  }

  class AndmeteSaabumiseKuular implements DropTargetListener {
    public void dragOver(DropTargetDragEvent e) { }
    public void dropActionChanged(DropTargetDragEvent e) { }
    public void dragExit(DropTargetEvent e) { }
    public void dragEnter(DropTargetDragEvent e) { }
    public void drop(DropTargetDropEvent e) {
     try{
        e.acceptDrop(DnDConstants.ACTION_COPY);
        Object data = e.getTransferable().
                      getTransferData(DataFlavor.stringFlavor);
        setText(data.toString());
     }catch(Exception ex){ ex.printStackTrace(); }
    }
  }
}
```

```
import java.awt.*;
public class Vedamine{
  public static void main(String argumendid[]){
    Frame f=new Frame("Andmeveo raam");
    f.setLayout(new GridLayout(3, 1));
    f.add(new Allikas("Karu"));
    f.add(new Allikas("Rebane"));
    f.add(new Suue("Vea siia!"));
    f.setSize(200, 100);
    f.setVisible(true);
  }
}
```

Trükkimine

Lühike näide

Lihtsaks trükkimiseks tuleb luua liidest `Printable` realiseeriv klass, kus meetodis `print` öeldakse, kuidas tuleb trükkida. Printerisse joonistamine käib samuti graafilise konteksti `Graphics` abil nagu ekraanile või mällugi joonistamine. Meetodi `print` parameetrina tuleva `PageFormat`'i abil saab teada, kui suure trükitava lehega tegemist on ning millisele osale lehest on võimalik joonistada. Enamasti on lehe serva määratud vaikimisi selline osa, kuhu joonistada ei saa. Joonistuskõlbuliku osa alguse x-koordinaadi annab `getImageableX`(). Samuti saab `PageFormat`'i käest küsida y-koordinaati ning joonistatava ala kõrgust ja laiust. Siin näites transleeritakse graafilise konteksti nullkoht joonistatava ala algusesse. Kolmanda parameetrina tulev leheküljenumber näitab, millist lehekülge soovitakse trükkida. Isendil on täiesti võimalik mitmele leheküljele trükkida. Lihtsalt tuleb meetodis print igale leheküljenumbrile vastavalt reageerida. Kui vastava numbriga lehekülge ei soovita trükkida, peab meetod tagastama väärtuse `Printable.NO_SUCH_PAGE`, muul juhul `Printable.PAGE_EXISTS`.

Trükkimise käivitamiseks luuakse isend tüübist `PrinterJob`, määratakse, milline `Printable` oskusega isend trükitöö ära teeb ning siis palutakse trükkima hakata.

```
import java.awt.print.*;
import java.awt.*;

public class Trykk1{
  public static void main(String argumendid[])
                      throws PrinterException{
    PrinterJob pj=PrinterJob.getPrinterJob();
    pj.setPrintable(new Trykitoo1());
    pj.print();
  }
}

class Trykitoo1 implements Printable{
  public int print(Graphics g, PageFormat pf, int lk)
                               throws PrinterException{
    if(lk>0) return Printable.NO_SUCH_PAGE;
    g.translate((int)pf.getImageableX(), (int)pf.getImageableY());
    g.drawOval(10, 10, 200, 200);
    return Printable.PAGE_EXISTS;
  }
}
```

Komponendi trükkimine

Kuna nii ekraanile kui printerisse joonistab klassi `Graphics` järglane, siis saab joonistamisel kasutada sama meetodit. Siin näites joonistatakse mõlemasse meetodi `paint` abil. Programm loob ekraanile nupu ning omaloodud komponendi `Kiri2`. Nupule vajutades trükitakse `Kiri2` printerisse. Trükkimine on korraldatud nii, et vajutamise peale trükitakse `Kiri2` tüüpi isendit 2 lehekülge, kummalegi lehele joonistatakse tema kujutis ning lehe alla kirjutatakse lehekülje number. Klassi `PrinterJob` meetod `printDialog` kutsub välja dialoogiakna, kust kasutaja saab määrata printerit ning väljastatavate lehekülgede numbreid ja koopiate arvu.

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

class Kiri2 extends Canvas implements Printable {

  public void paint(Graphics g) {
    g.setColor(Color.black);
    int W = (int)getSize().getWidth();
    int H = (int)getSize().getHeight();
    g.drawRect(1, 1, W-3, H-3);
    g.drawString("Tere!", W/2, H/2);
  }

  public int print(Graphics g, PageFormat pf, int lk)
                             throws PrinterException {
    if (lk >= 2) {
```

```
      return Printable.NO_SUCH_PAGE;
    }
    g.translate((int)pf.getImageableX(), (int)pf.getImageableY());
    g.setColor(Color.black);
    paint(g);
    g.drawString("lk nr. "+(lk+1), 100,
                          (int)pf.getImageableHeight()-50);
    return Printable.PAGE_EXISTS;
  }
}

public class Trykk2 extends Panel implements ActionListener {
        Kiri2 kiri=new Kiri2();
        Button b = new Button("Tryki");

    public Trykk2() {
        b.addActionListener(this);
        add(b);
        kiri.setSize(100, 50);
        add(kiri);
    }

    public void actionPerformed(ActionEvent e) {
        PrinterJob pj = PrinterJob.getPrinterJob();
        pj.setPrintable(kiri);
        try {
          if(pj.printDialog())
                  pj.print();
        } catch (Exception PrintException) { }
    }

    public static void main(String s[]) {
        Frame f = new Frame("Trykkimisraam");
        f.add("Center", new Trykk2());
        f.pack();
        f.setSize(400,300);
        f.show();
    }
}
```

Trükitava ala suuruse muutmine

Trükitava ala suurust lehel saab ka ise muuta. Sel juhul tuleb `PageFormat`'i käest küsida `Paper` tüüpi isend, ja siis sinna määrata soovitud suurusega joonistusala. Edasi määrata PageFormat'ile vastav paber ning paluda `PrinterJob`'il vastava `PageFormat`'i järgi trükkida. Siin näites lubatakse trükkida lehel servast serva.

```
import java.awt.print.*;
import java.awt.*;

public class Trykk3{
  public static void main(String argumendid[])
                       throws PrinterException{
    PrinterJob pj=PrinterJob.getPrinterJob();
    PageFormat pf=pj.defaultPage();
    Paper p=pf.getPaper();
    p.setImageableArea(0, 0, p.getWidth(), p.getHeight());
    pf.setPaper(p);
    pj.setPrintable(new Trykitoo3(), pf);
    pj.print();
  }
}

class Trykitoo3 implements Printable{
  public int print(Graphics g, PageFormat pf, int lk)throws PrinterException{
    if(lk>0) return Printable.NO_SUCH_PAGE;
    g.drawOval(0, 0, 300, 200);
    return Printable.PAGE_EXISTS;
  }
}
```

Trükitava ala suurust saab lasta ka kasutajal dialoogiakna abil määrata. Sellise akna manab ekraanile `pageDialog`.

```
public class Trykk3a{
  public static void main(String argumendid[])
   throws PrinterException{
    PrinterJob pj=PrinterJob.getPrinterJob();
    pj.setPrintable(new Trykitoo3a(),
            pj.pageDialog(pj.defaultPage()));
```

```
    pj.print();
  }
}
```

Lisavõimalused

`Printable` liidese abil saab trükkida ühesuguse suurusega lehekülgi. Kui peaks aga vaja olema ühte trükitavasse dokumenti kokku panna mitmesuguseid (näiteks püsti- ning põikiformaadis) lehti, siis tuleb algul panna kirjutatavatest lehtedest kokku `Book` ning seda trükkima hakata.

Kokkuvõte

Graafikakomponendid aitavad lihtsustada kasutajaga suhtlemist. Kümmekonda awt-paketis olevat komponenti juhitakse operatsioonisüsteemi poolt, nad näevad välja nii nagu vastavas operatsioonisüsteemis tavaks. Mõnikümmend swing-komponenti lisavad võimalusi. Need näevad välja igal pool ühtemoodi, töötavad suhteliselt aeglasemalt, kuid neid on kergem pilkupüüdvaks kujundada.

Komponentidega juhtunud sündmuste töötlemiseks tuleb luua vastava sündmuse kuular ning kuularil lasta end sündmuse allika juures registreerida. Komponent võib ka ise olla enesega juhtunud sündmuste kuulariks. Lisaks kuularitele saab vajadusel ka uusi sündmusi ja komponente ise luua.

Ülesandeid

Paigutamine

- Paiguta BorderLayout-i abiga nupp ülaserva ning tekstiala keskele.
- Jaga ülaserv võrdselt kolme nupu vahel.
- Paiguta allserva ühte ritta valik (choice) ja kerimisriba nii, et valik võtab tema jaoks hädavajaliku ruumi, riba aga ülejäänu.
- Allserva teise ritta lisa märkeruut ning kolm üheskoos töötavat raadionuppu.

Püüdmine

- 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.

Diagrammikomponent

- Loodavale komponendile joonistatakse etteantud arvu kõrgune tulp.
- Tulpade kõrgused antakse komponendile ette massiiviga.
- Joonistamisel leitakse koefitsiendid nii, et suurima tulba pikkus oleks 80% komponendi kõrgusest.