Graphics2D

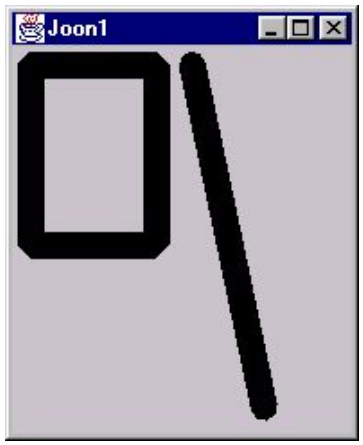
Graafiline kontekst, joone ja tausta omadused, kujundid.

Põhilised joonistamisfunktsioonid on klassis java.awt.Graphics. Alates versioonist 1.2 pakub laiendatud joonistamisvõimalusi eelmise alamklass Graphics2D. Kui esimesel juhul tuli alati arvutada ekraanikoordinaatides ning joone laiuseks oli üks punkt, siis siin võib valida omale sobiva taustsüsteemi ning ka joonistamisel saab enam parameetreid määrata. Kuna Graphics2D on klassi Graphics alamklass, siis ta oskab kõike mida eellanegi ning vajadusel saab temaga joonistada samade meetoditega, mis klassist Graphics omale sisse harjunud. Ühilduvuse huvides on komponendi meetodi paint parameetriks endiselt Graphics, kuid tegelikult antakse sellesse meetodisse joonistamiseks isend, kes suudab ka Graphics2D klassis kirjeldatud operatsioone täita. Juhul, kui spetsiifilisi omadusi vajatakse, tuleb muutuja tüüp enne teisendada.

**Joone omadused**

Joone tõmbamisel saab Graphics2D juures klassi BasicStroke abil määrata joone laiust, otsa kuju ning kahe joone ühendust (võimalused leiad API dokumentatsioonist).

```
import java.awt.*;
public class Joon1 extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    float laius=15;
    int jooneots=BasicStroke.CAP_ROUND;
    int uhendus=BasicStroke.JOIN_BEVEL;
    g.setStroke(new BasicStroke(
                    laius, jooneots, uhendus));
    g.drawRect(10, 10, 70, 100);
    g.drawLine(100, 10, 140, 200);
  }
}
```




**Punktiirjoon**

Punktiiri puhul tuleb määrata, millise pikkusega on punktiiri kriipsud, lisaks sellele hulk joontega seotud andmeid. Kuid kui korra on sulepea (Stroke) valmis tehtud, võib sellega rahumeeli jooni tõmmata nii palju kui vaid soovid – nii nagu eelmises näites. Kui vaadata sulepea koostamise käsklust

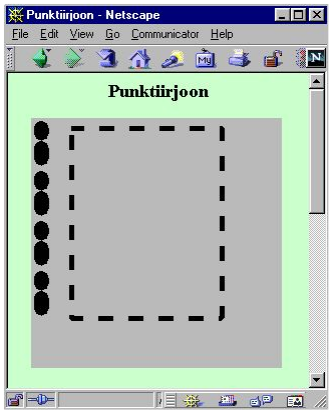
```
BasicStroke bs1=new BasicStroke(
                  15, ots, yhendus, yhendusemaxpikkus, punktiir, punktiirinihe);
```

siis 15 tähendab tõmmatava joone laiust. Ots näitab, millised (ümarad, kandilised) tuleb joone otsad teha. Muutuja yhendusemaxpikkus on enamasti tarbetu, vaja võib teda minna vaid siis, kui miskis kujundis ühinevad jooned väga väikese nurga all ning tekib oht pika väljaulatuva nurga moodustumiseks. Siis selle muutuja järgi vaadatakse, mitmest punktist vastav nurk pikemaks ei tohi minna. Eelnevalt loodud massiiv punktiir näitab, kuidas punktiirjoones vahelduvad kriipsud ja vahed. Nagu siin näites eespoolt võib piiluda, on esimese kriipsu pikkuseks määratud 5 punkti, sellele peaks järgnema viieteistpunktiline vahe. Edasi kümnepunktiline kriips ning selle järele kahekümnepunktine vahe. Siis taas algusest peale, et kahekümnepunktise vahe järele jälle viiene kriips, siis viieteistkümnene vahe ning nõnda edasi. Muutujast punktiirinihe vaadatakse, kus kohalt mustriga peale hakata. Kui nihe on 0, siis algab esimene kriips joone otsast. Kui mõni suurem arv, siis on esimese kriipsu algus vastavalt nihutatud.

Teine BasicStroke on loodud ühe käsuga, enamjaolt on väärtused otse konstruktorisse kirjutatud. `new float[]{15}` loob üheelemendilise massiivi otse kohapeal.

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

```
import java.awt.geom.*;
import java.applet.*;
public class Punktiir1 extends Applet{
  float laius=5;
  int ots=BasicStroke.CAP_ROUND, yhendus=BasicStroke.JOIN_MITER;
  float yhendusemaxpikkus=10;
  float[] punktiir={5, 15, 10, 20}; //kriips, vahe, kriips, vahe, ...
  float punktiirinihe=0;
  BasicStroke bs1=new BasicStroke(
                15, ots, yhendus, yhendusemaxpikkus, punktiir, punktiirinihe);
  BasicStroke bs2=new BasicStroke(laius, BasicStroke.CAP_BUTT, BasicStroke.JOIN_ROUND,
                   yhendusemaxpikkus, new float[]{15}, 2);
  public void paint(Graphics g){
    Graphics2D g2=(Graphics2D)g;
    Line2D l1=new Line2D.Float(10, 10, 10, 200);
    Rectangle2D r1=new Rectangle2D.Float(40, 10, 150, 190);
    g2.setStroke(bs1);
    g2.draw(l1);
    g2.setStroke(bs2);
    g2.draw(r1);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Punktiir");
    f.add(new Punktiir1());
    f.setSize(250, 250);
    f.setVisible(true);
  }
}
```




## Joonistusala piiramine

Joonistamisala on võimalik piirata käsuga clip, andes ette kujundi, mille piires tohib joonistada. Siin näites määratakse joonistamise alaks ellipsi pind ning seejärel joonestatakse seest täidetud ristkülik. Tulemusena tekib ekraanile vaid ellipsi ning ristküliku ühisosa.

```
import java.awt.*;
import java.awt.geom.*;
public class Kujund1 extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    Shape kujund=
        new Ellipse2D.Float(10, 10, 300, 100);
    g.clip(kujund);
    g.fillRect(20, 20, 400, 60);
  }
}
```


## Venitamine, keeramine

Joonistuspinda saab liigutada, venitada ja keerata. AffineTransform'i abil saab määrata, kuidas ja kui palju. Esimese näite puhul lükatakse koordinaatide alguspunkti saja ühiku võrra paremale ning alla.

```
import java.awt.*;
import java.awt.geom.AffineTransform;
public class Transform2 extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    g.setTransform(AffineTransform.getTranslateInstance(100, 100));
    g.fillRect(20, 20, 40, 20);
  }
}
```

rida

```
g.setTransform(AffineTransform.getScaleInstance(3, 1.5));
```

suurendab joonist x-telje suunas 3 ning y-suunas 1,5 korda.

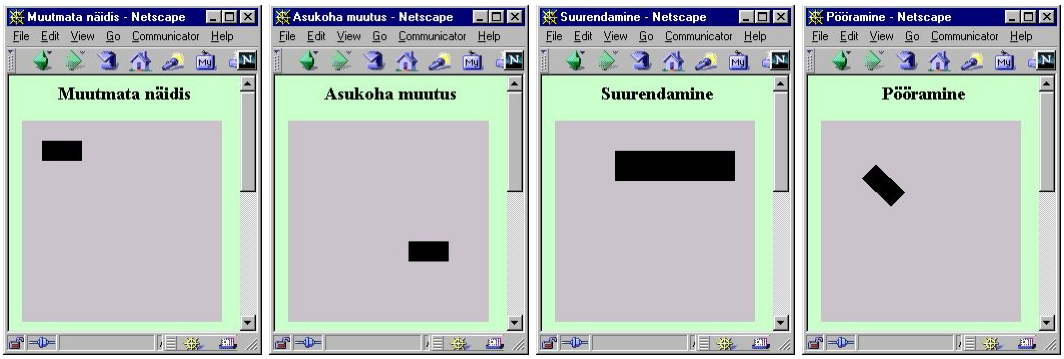
```
g.setTransform(AffineTransform.getRotateInstance(Math.PI/4, 100, 75));
```

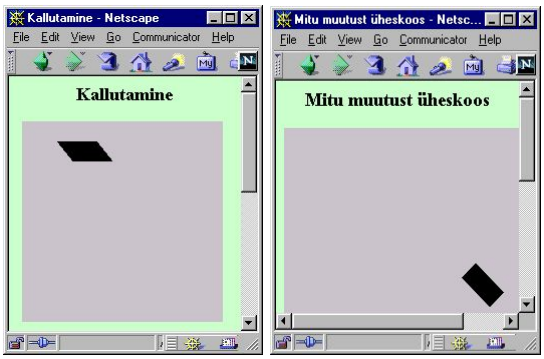
keerab joonist Pi/4 ehk 45 kraadi võrra ümber punkti 100, 75

```
g.setTransform(AffineTransform.getShearInstance(Math.PI/4, 0));
```
keerab püsttelge Pi/4 võrra.

Kui soovida mitut muutust üheskoos, siis võib luua AffineTransform tüüpi isendi ning talle soovitud muutusi järjekorras rakendada.

```
import java.awt.*;
import java.awt.geom.AffineTransform;
public class Transform6 extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    AffineTransform tr=new AffineTransform();
    tr.rotate(Math.PI/4, 50, 100);
    tr.translate(150, 0);
    g.setTransform(tr);
    g.fillRect(20, 20, 40, 20);
  }
}
```

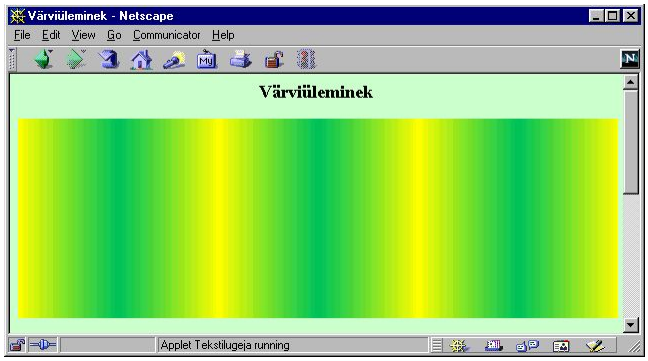



## Värviüleminek

Joonistamisel saab lisaks ühele värvile soovi korral pinda katta ka nii värviülemineku kui piltidest koostatud mustriga. Soovitud katmisstiil tuleb määrata Graphics2D meetodiga setPaint. Värviülemineku andmeid kannab GradientPaint. Konstruktoris tuleb määrata kaks punkti ning kummagi punkti juurde kuuluv värv. Nendes punktides vastab joonise värv sinna määratud värvile, punkte ühendaval sirget mööda muutub värv sujuvalt ühest värvist teiseks. Tavajuhul jääb kummagi punkti "selja taha" punktile vastav värv, kuid kui lisada konstruktorisse tõeväärtusmuutuja, siis saab panna värvi tsükliliselt lainetama nii, et laine pikkuseks jääb kahe punkti vahe.

```
import java.awt.*;
import java.awt.geom.*;
public class Kujund2a extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    g.setPaint(new GradientPaint(0, 100, Color.yellow,
               getSize().width, 100, new Color(0, 200, 100)
    ));
    g.fillRect(0, 0, getSize().width, getSize().height);
  }
}
```

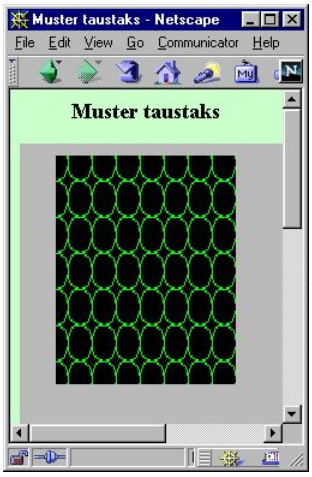



## Muster

Et taustaks saaks mustrit panna, selleks tuleb kõigepealt mustripilt koostada või välja otsida ning alles seejärel saab määrata, et joonistatava kujundi värviks on pidevalt korduv loodud muster. Järgnevas näites on mustritükiks lihtsalt roheline ring mustal taustal. `BufferedImage bi` hoiab eneses loodavat pilti, TexturePaint tp aga juba hoolitseb, kuidas pilt korralikult õigesse kohta paigutada. Konstruktoris luuakse pildile graafiline kontekst, mille abil joonistatakse pildile 20*20 punkti laiune roheline ring. Rida `tp=new TexturePaint(bi, new Rectangle(0, 0, 20, 30));` tähendab, et olemasolev pilt (algse suurusega 20*20 punkti) asub loodud `TexturePaint`'is olema 20 punkti lai ning 30 kõrge, seega pikkust pidi välja venitatud.

Joonistuskäsu `paint` sees võetakse pakutud `Graphics` vastu `Graphics2D`-na, et õnnestuks vajalikke käske (`setPaint`) kasutada. Edasi joonistatakse pinnale ristkülik. Kuna aga joonistusmustriks oli määratud pilt, siis loodud ristkülik näebki välja mustrilisena.

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.applet.*;
public class Mustritaust extends Applet{
  BufferedImage bi=new BufferedImage(
                  20, 20, BufferedImage.TYPE_INT_RGB);
  TexturePaint tp;
  public Mustritaust(){
    Graphics2D big=bi.createGraphics();
    big.setColor(Color.green);
    big.drawOval(0, 0, 20, 20);
    tp=new TexturePaint(bi, new Rectangle(0, 0, 20, 30));
  }
  public void paint(Graphics g){
    Graphics2D g2=(Graphics2D)g;
    Rectangle2D r1=new Rectangle2D.Float(30, 10, 150, 190);
    g2.setPaint(tp);
    g2.fill(r1);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Muster taustaks");
    f.add(new Mustritaust());
    f.setSize(250, 250);
    f.setVisible(true);
  }
}
```




## Värviüleminekuga tekst

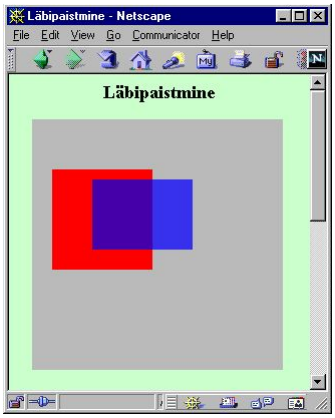
Joonistusala piiramist ning värviüleminekut kombineerides saab päris keeruka ja ilusa kujundusega pildi kokku panna.

```
import java.awt.*;
import java.awt.geom.*;
```

```
import java.awt.font.*;
public class Kujund2 extends Canvas{
  public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    TextLayout tl=new TextLayout("Kassikene",
      new Font("Arial", Font.PLAIN, 100),
      new FontRenderContext(null, false, false)
    );

    g.setPaint(new GradientPaint(200, 0, Color.yellow,
               200, getSize().height, new Color(255, 200, 0)
    ));
    g.fillRect(0, 0, getSize().width, getSize().height);

    g.clip(tl.getOutline(AffineTransform.getTranslateInstance(20, 120)));
    g.setPaint(new GradientPaint(0, 100, new Color(66, 66, 0),
               getSize().width, 100, new Color(0, 200, 100)
    ));
    g.fillRect(0, 0, getSize().width, getSize().height);
  }
}
```


Soovides jätta all olevat pilti joonistatava kujundi alt läbi paistma, tuleb enne peale joonistamist määrata `Composite`, mis paluks peale joonistataval alumine vaid osaliselt ära katta ning jätta tulemuseks vanaga segatud värvid.

```
import java.awt.*;
import java.applet.*;
public class Labipaistmine extends Applet{
  public void paint(Graphics g){
    Graphics2D g2=(Graphics2D)g;
    g2.setColor(Color.red);
    g2.fillRect(20, 50, 100, 100);
    g2.setComposite(AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, 0.7f));
           //70% joonistab, 30% paistab alt läbi
    g2.setColor(Color.blue);
    g2.fillRect(60, 60, 100, 70);
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Läbipaistmine");
    f.add(new Labipaistmine());
    f.setSize(250, 250);
    f.setVisible(true);
  }
}
```




## Kujundi äärejooned

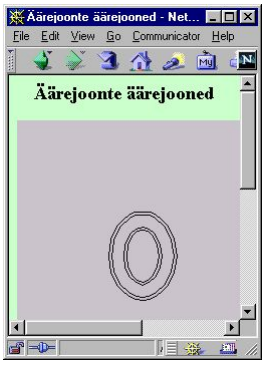
Kujundi (näiteks ellipsi) äärejooned annab BasicStroke meetod createStrokedShape. Nii näiteks on võimalik kujundist vaid äärejooned välja joonistada või siis teise värviga esile tuua.

```
public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    BasicStroke b1=new BasicStroke(15);
    Shape kujund=b1.createStrokedShape(
      new Ellipse2D.Float(100, 100, 50, 70)
    );
    g.fill(kujund);
    g.setColor(Color.green);
    g.draw(kujund);
  }
```


## Äärejoonte äärejooned

Kuna nii ellips on kujund ning ka jämedajoonelise ellipsi äärejooned on samuti kujundid, siis juhul, kui annan äärejoontele jämeduse, võin ka nendelt omakorda äärejooned küsida. Nii saan tulemuseks juba neli joont: kaks üksteisele lähedal asuvat ovaali sees ning teised kaks ovaali välisringis.

```
public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    BasicStroke b1=new BasicStroke(15);
    BasicStroke b2=new BasicStroke(3);
    Shape kujund=b1.createStrokedShape(
      new Ellipse2D.Float(100, 100, 50, 70)
    );
    Shape kujund2=b2.createStrokedShape(kujund);
    g.draw(kujund2);
 }
```

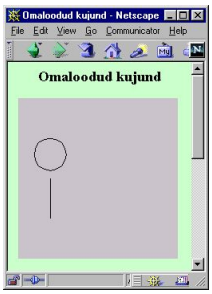



## Kujundi koostamine

Kui sooviksin nende joontega midagi eraldi teha, näiteks neid igaüht isevärvi värvida, siis on mul võimalik kujund joonteks jagada klassi PathIterator abil. Selle abil saan kätte iga joone andmed. Sirgjoonel piisab kahest punktist. Kolme punkti abil määratakse kõverjoon, kus kaks punkti on otspunktideks ning kolmas näitab, milline peab kaar tulema. Nelja punktiga määratud joone puhul on samuti kaks otspunktideks, ülejäänud kahe punkti abil aga määratakse joone suunda otspunktist väljumisel.

Uusi kujundeid aitab kombineerida klass GeneralPath. Talle tuleb lihtsalt öelda millise koha peale joon või ring või muu olemasolev kujund paigutada. Kriipsujuku saab tema abil küllalt kergesti valmis ning siis võib seda kasutada nagu iga muud kujundit.

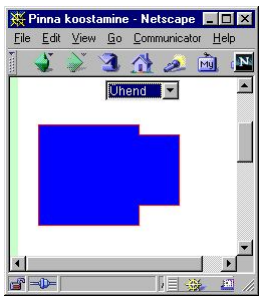
```
public void paint(Graphics alggr){
    Graphics2D g=(Graphics2D)alggr;
    GeneralPath gp=new GeneralPath();
    gp.append(new Ellipse2D.Float(20, 50, 40, 40), false);
    gp.append(new Line2D.Float(40, 100, 40, 150), false);
    g.draw(gp);
}
```

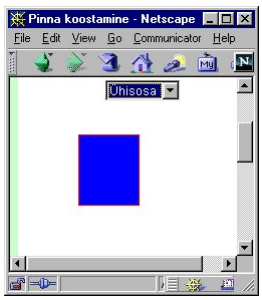


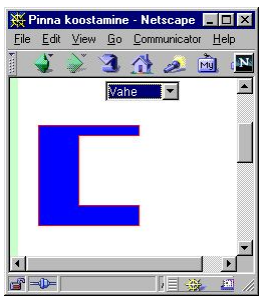
## Tehted kujunditega

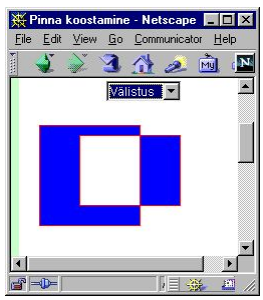
Kujundit luues on võimalik olemasolevatega ka keerukamaid tehteid teha. Näiteks kui soovida koostada kuuveerandikku, siis võib kõigepealt teha ühe väiksema ringi ning siis sellest lahutada maha suurem ring, nii et vaid väike osa algsest jääb alles. All olevas näites on võetud kas osaliselt kattuvat ristkülikut ning näidatud, mis tehteid nendega läbi viia saab. Liites esimesele juurde teise saame kujundite ühendi, kus mõlema kujundi pinnad on liidetud. Ühisosa tekitab käsk `intersect` – alles jääb vaid osa pinnast, mis kuulub mõlema kujundi koosseisu. Lahutamise (`subtract`) korral jääb esialgsest pinnast alles vaid osa, mis teise alla ei kuulu. Välistuse (exclusiveOr) puhul jääb mõlemast pinnast alles vaid osa, kus üks pind teisega ei kattu.

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.awt.event.*;
import java.applet.*;
public class Pind extends Applet implements ItemListener{
  Area r1=new Area(new Rectangle2D.Float(20, 50, 100, 100));
  Area r2=new Area(new Rectangle2D.Float(60, 60, 100, 70));
  Choice valik=new Choice();
  public Pind(){
    valik.add("Ühend");
    valik.add("Ühisosa");
    valik.add("Vahe");
    valik.add("Välistus");
    add(valik);
    valik.addItemListener(this);
  }
  public void paint(Graphics g){
    Area pind=new Area();
    pind.add(r1);
    String s=valik.getSelectedItem();
    if(s.equals("Ühend"))pind.add(r2);
    if(s.equals("Ühisosa"))pind.intersect(r2);
    if(s.equals("Vahe"))pind.subtract(r2);
    if(s.equals("Välistus"))pind.exclusiveOr(r2);
    Graphics2D g2=(Graphics2D)g;
    g2.setColor(Color.blue);
    g2.fill(pind);
    g2.setColor(Color.red);
    g2.draw(pind);
  }
  public void itemStateChanged(ItemEvent e){
    repaint();
  }
  public static void main(String argumendid[]){
    Frame f=new Frame("Muster taustaks");
    f.add(new Pind());
    f.setSize(250, 250);
    f.setVisible(true);
  }
}
```









### *Ülesandeid*

- Joonista ring joone laiusega 25 punkti.
- Määra joonistusvärviks üleminek punaselt kollasele
- Määra taustamustriks väikesed kriipsujukud.
- Muuda joon punktiiriks.
- Määra joonistusala nii, et sinna jääb vaid osa ringjoont
- Jäta taust ringi alt veidi läbi paistma.

## Swing

Operatsioonisüsteemist sõltumatud komponendid, kujundus

Lisaks kümnekonnale paketis `java.awt` asuvale komponendile saab kasutajaga suhtlemiseks tarvitada ka paketi `javax.swing` graafilisi komponente. Nagu kirjeldatud, palutakse `awt`-komponendid joonistada operatsioonisüsteemil, `swing`-komponente joonistatakse Java vahenditega. Sellest tulenevalt näevad esimesed välja nii nagu vastavas operatsioonsiüsteemis tavaks, `swing`i nupp või silt aga on igal pool tavajuhul peaaegu ühesugune. `UIManager`i abil aga on võimalik panna ka `swing`-komponente vastavalt operatsioonisüsteemile välja nägema.

Swingi graafikakomponendid algavad tähega J. Tõenäoliselt seetõttu, et neid oleks kerge eristada analoogilistest awt komponentidest. Järgnevas näiteks on raamiks `JFrame`, selle sees on silt `JLabel` ning nupp `JButton`. Nii nupule kui sildile (ja ka mitmetele muudele komponentidele) saab tema ilmestamiseks määrata ikooni. `ImageIcon` loob ikooni kasutades aluseks pildifaili, kuid vajadusel saab ikooni ka käskude abil joonistada. Kõikidele `swing`i komponentidele saab määrata `ToolTipText`'i. Seda näidatakse ekraanile juhul, kui kasutaja on hiirega vastava komponendi peale liikunud. Enamasti vastav tekst seletab komponendi otstarvet või annab kasutajale tegutsemissoovitusi. Korraldus `setMnemonic` lubab klahvikombinatsiooni (Alt + täht) võrdsustada hiirega nupule vajutamisele.

Komponentide raami lisamisel tuleb swingi puhul määrata, millisesse kihti ta paigutada. Harilikult kasutatava alumise kihi saab kätte `getContentPane()` abil. Pealmist kihti nimetatakse `GlassPane` ning vahepealsetesse kihtidesse paigutamiseks saab kasutada `LayeredPane` vahendeid. Kihtidega mängides saab komponente mitmesse kihti paigutada.

```
import java.awt.*;
import javax.swing.*;
public class Pildid{
  public static void main(String argumendid[]){
    JLabel silt=new JLabel("Maja silt");
    Font suurkiri=new Font("Serif", Font.BOLD+Font.ITALIC, 30);
    Icon majapilt=new ImageIcon("maja.gif");
    silt.setFont(suurkiri);
    silt.setIcon(majapilt);

    JButton nupp=new JButton("Maja nupp", majapilt);
    nupp.setToolTipText("Head vajutamist!");
    nupp.setMnemonic(java.awt.event.KeyEvent.VK_M);
   JFrame f=new JFrame("Sildiraam");
   Container p=f.getContentPane();
   p.setLayout(new GridLayout(2, 1));
   p.add(silt);
   p.add(nupp);
   f.pack();
   f.setVisible(true);
  }
}
```

### *HTML-kujundus*

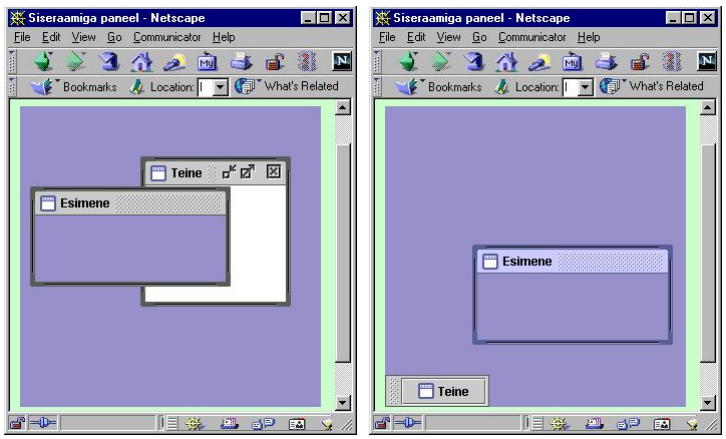
Swingi komponentidel näidatavat teksti saab HTMLi abil kujundada. Täpne väljanägemine võib sõltuda intepretaatorist, kuid selliselt on programmi väljanägemist lihtsam pilkupüüdvaks muuta kui awt vahenditega kujundades. Nagu lihtsat teksti kandva sildi puhul, nii ka siin on võimalik programmi töö käigus sildi sisu muuta.

```
import javax.swing.*;
public class HtmlLabel{
  public static void main(String argumendid[]){
    JFrame f=new JFrame("Kujundatud silt");
    JLabel silt=new JLabel(
      "<html><h2>Pealkirja</h2>\n"+
      "ja <font color=red>punase tekstiga</font> silt</html>"
    );
   f.getContentPane().add(silt);
   f.pack();
   f.setVisible(true);
  }
}
```


### *Sisemised raamid*

Mõnes programmis on näha, et pearaami sees on omaette väiksemad raamid. Nii näiteks Wordi puhul võib iga tekst olla lahti omaette raamis, need aga omakorda suure Wordi raami sees. Sellist olukorda Java keskkonnas saab tekitada `JInternalFrame` abil. Nemad käituvad `JDesktopPane` sees samuti nagu harilikud raamid suure ekraani sees. Tema sees paiknevasse paneeli saab lisada komponente nagu ikka. Ka sisemisi raame saab muuta ikooniks alla serva, suurendada ja vähendada. Kui suure raami teateid püüab `WindowListener`, siis siseraamiga toimuva teada saamiseks aitab `InternalFrameListener`. Vorm on küll erinev, kuid võimalused samad. Näiteks koostab JDesktopPane tüüpi komponendi eraldi staatiline meetod. Rakendis paigutatakse komponent ekraanile `init` meetodi sees, käsurealt käivitades luuakse raam ning siis paigutatakse komponent raami sisse.

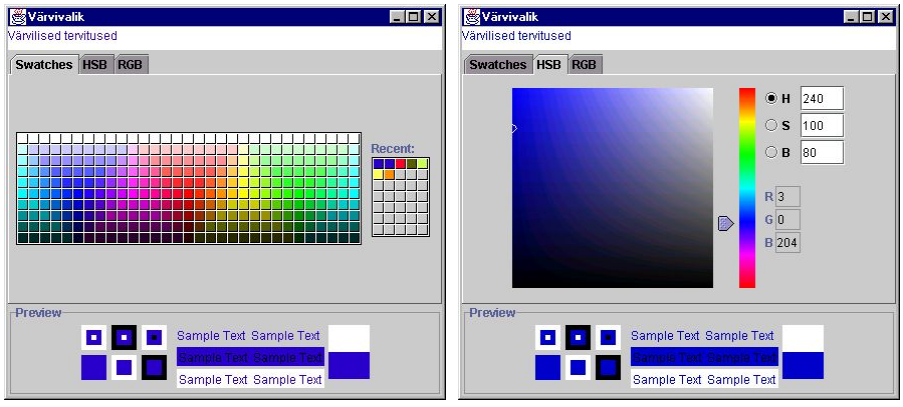
```
import javax.swing.*;
public class SiseraamigaRaam extends JApplet{
   static JDesktopPane looRaamiPaneel(){
    JInternalFrame siseraam1=new JInternalFrame("Esimene");
    JInternalFrame siseraam2=new JInternalFrame("Teine",
                                 true, true, true, true);
    siseraam2.getContentPane().add(new JTextArea());
    JDesktopPane paneel=new JDesktopPane();
    siseraam1.setSize(200, 100);
    siseraam1.setLocation(10, 80);
    siseraam1.setVisible(true);
    paneel.add(siseraam1);
    try{siseraam1.setSelected(true);}catch (Exception e){}
    siseraam2.setVisible(true);
    paneel.add(siseraam2);
    siseraam2.setSize(150, 150);
    siseraam2.setLocation(120, 50);
    return paneel;
  }
  public void init(){
    getContentPane().add(looRaamiPaneel());
  }
  public static void main(String argumendid[]) throws Exception{
    JFrame f=new JFrame("Kest");
    f.setContentPane(looRaamiPaneel());
    f.setSize(300, 300);
    f.setVisible(true);
  }
}
```

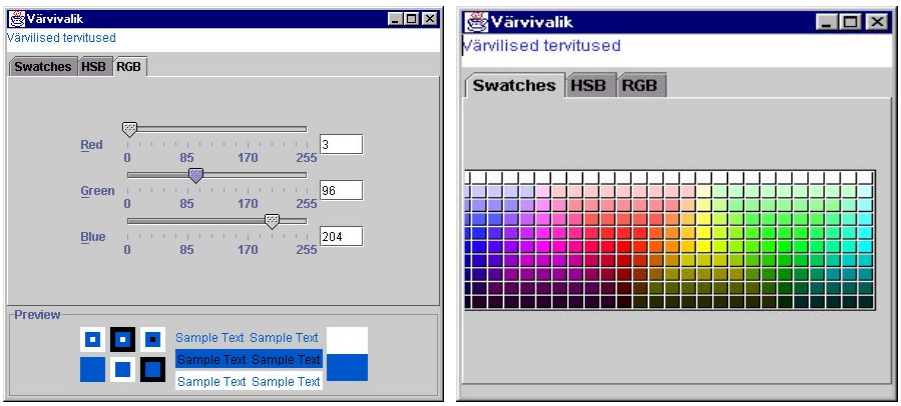



### *Värvivalija*

Kasutajapoolseks värvi valimiseks saab vajadusel kirjutada ise dialoogiakna, kust hiirega omale sobiv värv leida võimalik on. `Swing`i all aga on programmeerimisvaeva vähendamiseks loodud komponent `JColorChooser`, mille abil kasutaja võib pakutavate hulgast sobiva värvi välja valida. Värvivalikuks pakub komponent kolme võimalust: esimesel juhul saab hiirega vajutada sobivat värvi ruudule. Teisel puhul saab valida värvi ning teiselt skaalalt värvile vastava tumeduse. Kolmandas valikuaknas lastakse kasutajal määrata punase, rohelise ning sinise vahekord loodavas värvis. Vaikimisi kujul näitab komponent otsitavat värvi mitmesuguste kujundite ning ingliskeelse teksti peal. Selle kujundi saab aga vajadusel programmi ilmele sobivama või hoopis tühja pisikese paneeli vastu välja vahetada (järgnevas näites vastav rida välja kommenteeritud). Värvivaliku registreerimiseks ning temale reageerimiseks saab kasutada kuularit. Värvivaliku klassi juurde on loodud meetodid ka dialoogiakna kaudu värvi valimiseks.

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.BorderLayout;
public class Varvivalik{
  public static void main(String argumendid[]){
    final JTextArea tekstiala=new JTextArea("Värvilised tervitused");
    final JColorChooser valija=new JColorChooser();
//    valija.setPreviewPanel(new JPanel());
    valija.getSelectionModel().addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){
          tekstiala.setForeground(valija.getColor());
        }
      }
    );
    JFrame f=new JFrame("Värvivalik");
    java.awt.Container p=f.getContentPane();
    p.add(tekstiala, BorderLayout.CENTER);
    p.add(valija, BorderLayout.SOUTH);
    f.setSize(300, 500);
    f.setVisible(true);
  }
}
```

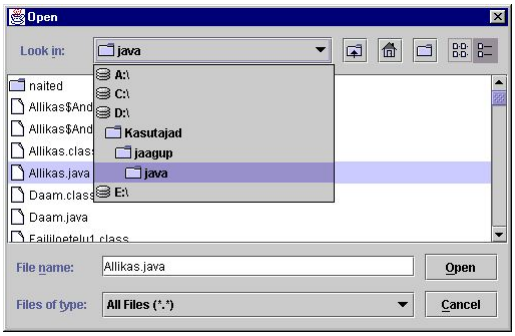





## *Failinime valija*

Ka failinime valimise dialoogi saab ise suurema vaevata kirjutada, kuid swingi klass JFileChooser on juba vastavaks otstarbeks loodud ilusasti kujundatud komponent. Saab määrata, millisest kataloogist alates faili otsima saab hakata. Meetod `showOpenDialog` avab dialoogi ning programm jääb kasutajapoolset valikut ootama. Kui fail on valitud või valimine tühistatud, läheb programm edasi ning komponendi meetodite kaudu saab teada kasutaja vastuse. Kui soovitud fail on käes, saab temaga samuti toimida nagu failiga ikka, s.t. tema kohta infot küsida, sealt lugeda või sinna kirjutada.

```
import javax.swing.*;
import java.io.*;
public class Failivalik{
  public static void main(String argumendid[]){
    JFileChooser valija=new JFileChooser(new File("."));
    valija.showOpenDialog(new JFrame());
    System.out.println("Valiti "+valija.getSelectedFile());
  }
}
```




```
D:\Kasutajad\jaagup\java\naited\gr\swing>java Failivalik
Valiti D:\Kasutajad\jaagup\java\Allikas.java
```

Failivalikut saab veidi programmiga kohandada. Näiteks võib määrata avamisnupu peal olevat kirja. Samuti võib lubada korraga mitut faili valida. Järgnevas näites lisatakse filtri abil võimalus eraldi vaid pildifailide hulgast kasutajal sobivat otsida. Filtri loomisel tuleb üle katta meetodid accept ning getDescription. Esimesele antakse järjekorras ette kõik failid, mida parasjagu oleks võimalik valida. See meetod peab igaühe kohta neist ütlema, kas vastavat faili kasutajale näidata või mitte. Meetod getDescription väljastab filtrile sobivate failide ühisnimetaja, siin näites "Pildifailid".

```
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import java.io.*;
public class Failivalik2{
  public static void main(String argumendid[]){
    JFileChooser valija=new JFileChooser(new File("."));
    valija.addChoosableFileFilter(new Pildifilter());
    valija.showDialog(new JFrame(), "Vali fail");
    System.out.println("Valiti "+valija.getSelectedFile());
```

```
  }
}

class Pildifilter extends FileFilter{
  public boolean accept(File f){
    String failinimi=f.getName();
    if(failinimi.endsWith(".gif")|failinimi.endsWith(".jpg"))
      return true;
    else return false;
  }
  public String getDescription(){
    return "Pildifailid ";
  }
}
```


## *Puud*

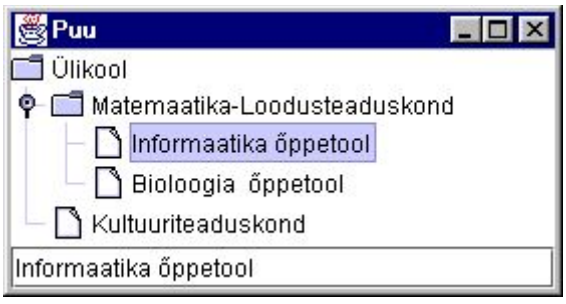
Infohulgas orienteerumiseks saab andmeid esitada puuna. Siis on kasutajal võimalik ekraanilt kasutu kõrvaldada ning hierarhia abil enesele sobiv üles leida. Puu abil on näiteks esitatud failid ja kataloogid WindowsExploreris. Puusse saab lisada kõiki objekte. Vaikimisi juhul määrab `JTree` ise okstele ja lehtedele sobivad ikoonid ning objekti kirjelduseks kasutab sõnet mille väljastab selle `toString` meetod. Vajadusel aga võib nii ikooni kui kirjeldust muuta.

Puu hierarhia saab kokku panna `DefaultMutableTreeNode` abil. Kui element on reas viimane, on ta leht, keskel oks ning algul juur. Puu ekraanile kuvamiseks tuleb luua `JTree`, kellele määrata juur, millest alates elemente näidata tuleb. Siin näites on pandud juureks ülikool, tema alla paar teaduskonda ning teaduskonna alla mõni õppetool. Võib jääda mulje, nagu tuleks puu loomiseks hirmus palju kirjutada, kuid suuremate andmehulkade korral saab luua või kasutada olemasolevaid alamprogramme ning puu tegemine polegi kuigi keeruline.

```
import javax.swing.*;
import javax.swing.tree.*;
public class Puu{
  public static void main(String argumendid[]){
    DefaultMutableTreeNode juur=new DefaultMutableTreeNode("Ülikool");
    DefaultMutableTreeNode teaduskond=new DefaultMutableTreeNode(
       "Matemaatika-Loodusteaduskond");
    teaduskond.add(new DefaultMutableTreeNode("Informaatika õppetool"));
    teaduskond.add(new DefaultMutableTreeNode("Bioloogia  õppetool"));
    juur.add(teaduskond);
    juur.add(new DefaultMutableTreeNode("Kultuuriteaduskond"));
    JTree puu=new JTree(juur);
    JFrame f=new JFrame("Puu");
    f.getContentPane().add(puu);
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```


Kasutaja tegevuse registreerimiseks saab tarvitada mitmesuguseid kuulareid. Saab teada, millal ta puu nähtavat osa suurendas või vähendas, mis osa puust nähtav on, millal mõni element valiti. Ka pärast puu loomist saab temasse elemente lisada ja sealt eemaldada. Käsklus puu.putClientProperty("JTree.lineStyle", "Angled"); palub puu osad omavahel joontega ühendada. Siin näites trükitakse igal valikul välja valitud komponent. TreeSelectionEvent'i meetod getPath() annab tulemuseks kogu rea alates juurest kuni märgitud leheni. Selle kaudu oleks võimalik ükskõik millise tee peale jääva komponendi poole pöörduda. Meetod getLastPathComponent() annab tulemuseks tee viimase elemendi, ehk selle, millele kasutaja vajutas.

```
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.BorderLayout;
public class Puu2a extends JApplet{
   static JTextField tekstikast=new JTextField();
   static JPanel looPuu(){
    DefaultMutableTreeNode juur=new DefaultMutableTreeNode("Ülikool");
    DefaultMutableTreeNode teaduskond=new DefaultMutableTreeNode(
       "Matemaatika-Loodusteaduskond");
    teaduskond.add(new DefaultMutableTreeNode("Informaatika õppetool"));
    teaduskond.add(new DefaultMutableTreeNode("Bioloogia  õppetool"));
    juur.add(teaduskond);
    juur.add(new DefaultMutableTreeNode("Kultuuriteaduskond"));
    JTree puu=new JTree(juur);
    puu.addTreeSelectionListener(
      new PuuKuular()
    );
    puu.putClientProperty("JTree.lineStyle", "Angled");
    JPanel paneel=new JPanel(new BorderLayout());
    paneel.add(puu);
    paneel.add(tekstikast, java.awt.BorderLayout.SOUTH);
    return paneel;
   }
   public void init(){
     getContentPane().add(looPuu());
   }
   public static void main(String argumendid[]){
    JFrame f=new JFrame("Puu");
    f.getContentPane().add(looPuu());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}

class PuuKuular implements TreeSelectionListener{
  public void valueChanged(TreeSelectionEvent e){
    Puu2a.tekstikast.setText(e.getPath().getLastPathComponent()+"");
  }
}
```



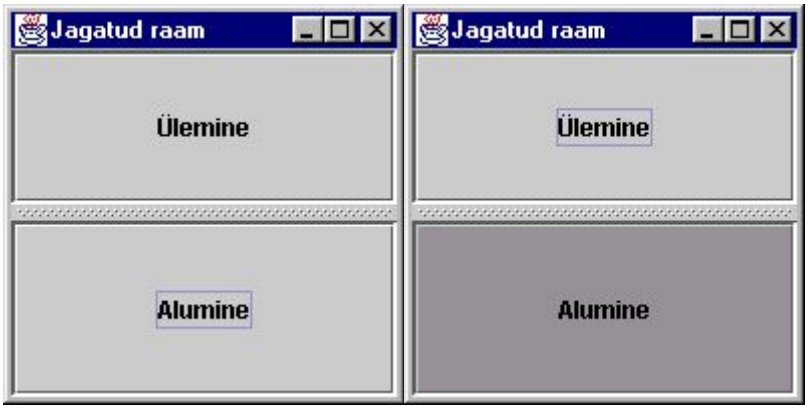
## *Paigutus*

### Jaotuspaneel

`JSplitPane` võimaldab temale eraldatud pinna jaotada kahe komponendi vahel. Tuleb vaid määrata, kas pind jaotatakse kaheks horisontaalselt või vertikaalselt ning komponendid, mis kummasegi ossa panna. Lisameetoditega saab määrata ja muuta jagamise kohta, samuti kasutajapoolset vahepiiri nihutamise võimalusi. Jagatud paneelidena näevad välja mitut lehte sisaldavad brauseri aknad, kus vasakul näiteks sisukord ja paremal sisu.

Kui koodi vaadata, siis üles on pandud JButton, alla JBoggleButton. Viimase omapäraks on, et esimest korda vajutades jääb nupp sisse (tumedaks), alles teisel korral tuleb välja tagasi.

```
import javax.swing.*;
public class Jaotuspaneel{
  public static void main(String argumendid[]){
    JSplitPane paneel=new JSplitPane(
      JSplitPane.VERTICAL_SPLIT,
      new JButton("Ülemine"),
      new JToggleButton("Alumine")
    );
    JFrame f=new JFrame("Jagatud raam");
    f.getContentPane().add(paneel);
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```


### Valikupaneel

Kui määranguaknas on võimalusi rohkem kui kasutajale korraga mõistlik näidata on, siis `JTabbedPane` abil saab muud paneelid ülekuti paigutada. Sarnaselt on loodud näiteks Exceli lahtrimäärangute dialoogiaken, kus ühel paneelil saab määrata andmete tüüpi, teisel kujundust jne. Soovi korral saab paneelivaliku nuppudele lisada ikoonid, eemaldada, vahetada ja lisada paneele, mõne valiku tegemist keelata, automaatselt soovitud paneel esile tuua ning mitmel moel kasutaja tegevuse kohta teateid saada.

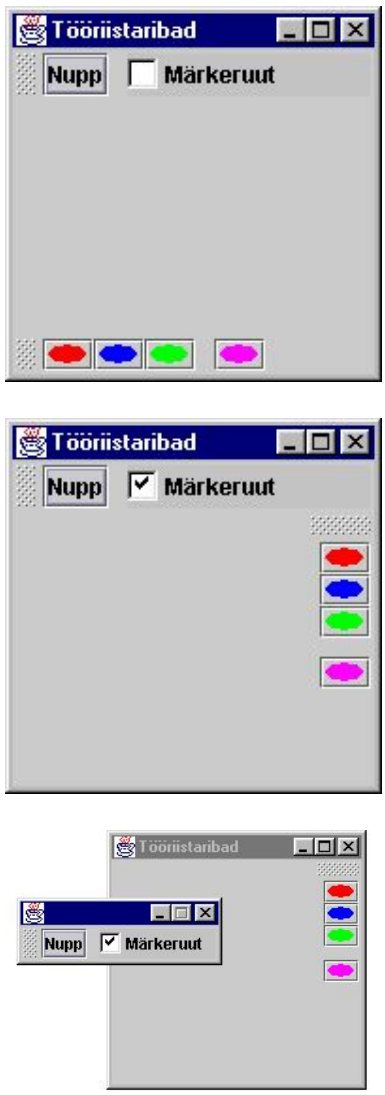
```
import javax.swing.*;
public class Valikupaneel{
  public static void main(String argumendid[]){
    JPanel p1=new JPanel(new java.awt.GridLayout(2,1));
    p1.add(new JButton("Ülemine"));
    p1.add(new JButton("Alumine"));

    JTabbedPane paneel=new JTabbedPane();
    paneel.add("Esimene", p1);
    paneel.add("Teine", new JLabel("Suur silt"));

    JFrame f=new JFrame("Valikupaneeli näide");
    f.getContentPane().add(paneel);
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```


### *Tööriistariba*

Ka sellenimeline abivahend on `Swing`i all täiesti olemas. Kui on soovi nuppe ja muid tööriistu oma silma järgi ümber paigutada, siis tööriistariba peaks selleks parim valik olema. Kokku saab selle panna nagu tavalise paneeli, kuid edaspidi on loodud paneeli võimalik pea vabalt paigutada. Riba saab liigutada nii omaette raamaknana kui paigutada vabalt iga BorderLayout'i vaba serva peale. Piisab vaid riba lohistamisest õige koha lähedusse, kui see juba haakub. Paigalt eemale saab ka täiesti rahumeeli lohistada. Et JAppleti vaikimisi paigutushalduriks ongi BorderLayout, siis neli külge on tööriistaribade jaoks kohe kasutatavad.

Riba ennast annab koostada ning valmis komponente sinna peale panna paari käsuga. Pea pool näiteprogrammist on kulunud ovaali kujutava pildiga ikoone loova klassi valmistamiseks, mille abil on hõlbus äratuntava pildiga nuppe toota. Ikooni loomiseks tuleb teha liidest Icon realiseeriv klass. Siin on see paigutatud Tooriistariba sisemiseks klassiks, et oleks viimasele alati kättesaadav ning kopeerides kaduma ei läheks. Iseenesest aga võib loodav ikoone tootev klass olla rahumeeli eraldi klassina, sel juhul on võimalik ovaalseid ikoone ka teiste programmide kasuks luua. Meetodis `paintIcon` tuleb kirja panna, milline ikoon välja näeb, `getIconWidth()` ning `getIconHeight()` annavad ikooni soovitud suuruse.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Tooriistariba extends JApplet {
   public Tooriistariba() {
    JToolBar riba = new JToolBar();
    JButton nupp = new JButton("Nupp");
    riba.add(nupp);
    riba.addSeparator();
    riba.add (new Checkbox ("Märkeruut"));
    getContentPane().add (riba, BorderLayout.NORTH);


    riba = new JToolBar();
    Icon icon = new OvaalneIkoon(Color.red);
    nupp = new JButton(icon);
    riba.add(nupp);
    icon = new OvaalneIkoon(Color.blue);
    nupp = new JButton(icon);
    riba.add(nupp);
    icon = new OvaalneIkoon(Color.green);
    nupp = new JButton(icon);
    riba.add(nupp);
    riba.addSeparator();
    icon = new OvaalneIkoon(Color.magenta);
    nupp = new JButton(icon);
    riba.add(nupp);
    getContentPane().add (riba, BorderLayout.SOUTH);
  }
  class OvaalneIkoon implements Icon {
    Color varv;
    public OvaalneIkoon (Color c) {
      varv = c;
    }
    public void paintIcon (Component c, Graphics g,
                           int x, int y) {
      g.setColor(varv);
      g.fillOval (
        x, y, getIconWidth(), getIconHeight());
    }
    public int getIconWidth() {
      return 20;
    }
    public int getIconHeight() {
      return 10;
    }
  }
  public static void main(String[] argumendid){
    JFrame f=new JFrame("Tööriistaribad");
    f.getContentPane().add(new Tooriistariba());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```




### *Ennistamine*

Kui üheksakümnendate aastate algul Word 2-te käsu tagasi võtmise võimalus sisse pandi, siis tundus olevat tegemist uue tähelepanuväärse ja omapärase lahendusega. Nüüd on kasutajad programmide juures pea alatise tagasivõtmise võimalusega nii harjunud, et panevad imeks, kui mõne käsu tagajärgi pole võimalik olematuks teha ning peab enne näpuliigutust hoolikalt läbi mõtlema, mis tehtu tulemuseks võib olla. Isegi terveid kataloogitäisi andmed saab Windows Exploreris paigast nihutada ning tagasi panna ilma, et selle peale suuremaid raskusi tekiks. Samuti kannatab mitmeski kohas mitte ainult üht või paari käsku tagasi võtta, vaid annab pea sammhaaval kogu töö algusesse minna.

Sellised võimalused ei teki töösse iseenesest, selle loomiseks tuleb programmi kirjutamisel kõvasti hoolt kanda. Samuti peab arvestama, et igat asja pole siiski võimalik tagasi võtta. Kui kord on soovimatu sisuga kirjad teistele inimestele laiali saadetud, siis on nad teel ja kohal ning meie loodud programmil pole nende kaotamiseks võimalik enam midagi ette võtta. Parimal juhul annab tagasivõtmise käsu juures sihtkohta uus kiri saata, et ärgu vastuvõtja eelmise saabunud kirja sisu liialt südamesse võtku.

Tagasivõtmist annab koodi sisse mitut moodi ehitada, kuid `Swing`i vahendite juures on eraldi

selle tarbeks loodud `UndoManager`, mis peaks aitama suuremate tööde puhul toimingule süsteemsemalt läheneda. Et tööd saaks pärast ilusti tagasi võtta, tuleb iga tehtud samm lisada ennistushaldurisse ning iga sammu juures peab olema kirjas, kuidas samm teha ning kuidas tagasi võtta. Niimoodi tekib sammude ahel, mida mööda on pärastpoole mugav edasi ja tagasi käia.

Allpool olevas näites on tagasivõetava programmi koostamine läbi mängitud lihtsa pildiredaktori peal, millele saab joonistada vaid ringe. Lisatud on nupud käskude tagasi võtmiseks ning sama teed pidi edasi liikumiseks.

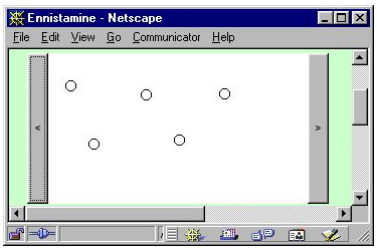
Iga hiirevajutusega lisatakse näidatavate ringide nimistusse (`Vector`) (muutuva pikkusega massiiv) punkti andmed, mis tähistavad hiirevajutuse asukohta. Nii on mälus kirjas, kuhu joonistusvajaduse korral ringid tekitada ning vähemasti iga operatsiooni järel palutakse ekraanipilt uuendada.

Ennistuse huvides ei lisata punkti andmeid nimistusse otseselt, vaid tehakse veidi pikem ring. Ringi lisamise kirjeldamiseks on loodud eraldi klass `LisatavRing`, mis laiendab klassi `AbstractUndoableEdit`. Klassis on käsud `undo` ja `redo`, kuhu tuleb kirja panna tegevused, mis tuleb sooritada vastavalt tagasi või edasi liikumiseks. Klassis on isendimuutujaks `Point` (paketist `java.awt`) parasjagu lisatava punkti andmete hoidmiseks. Töö lihtsustamiseks on kohe klassi LisatavRing isendi loomisel palutud tal kohe teha läbi edasi liikumisega seotud töö ehk lisada punkti andmed. Sel juhul piisab peaprogrammis sammu tegemisel vaid soovitud koordinaatidega LisatavaRingi loomisest. Edaspidise tagasivõtu sujumiseks tuleb vastav isend UndoableEditEvent'i koosseisus ennistushaldurisse lisada. Nuppude abil tööjärjes edasi-tagasi liikumiseks tuleb vaid ennistushaldurile anda käske undo või redo vastavalt soovitud suunale. Samuti on viisakas enne kontrollida, kas vastavas suunas üldse võimalik liikuda on.

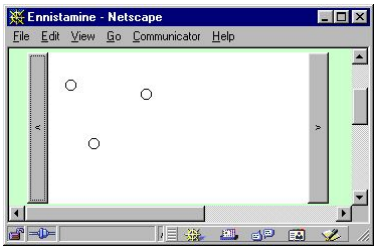
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.undo.*;
import javax.swing.event.*;
import java.util.Vector;


public class Ennistus extends JApplet
                       implements ActionListener{
  static Vector ringid=new Vector();
  UndoManager ennistushaldur=new UndoManager();
  Button edasi=new Button(">");
  Button tagasi=new Button("<");


  public Ennistus(){
    getContentPane().add(tagasi, BorderLayout.WEST);
    getContentPane().add(edasi, BorderLayout.EAST);
    tagasi.addActionListener(this);
    edasi.addActionListener(this);
    addMouseListener(
      new MouseAdapter(){
        public void mousePressed(MouseEvent e){
          ennistushaldur.undoableEditHappened(
              new UndoableEditEvent(
                Ennistus.this,
                new LisatavRing(e.getX(), e.getY())
              )
          );
          repaint();
        }
      }
    );
  }
```

```
  public void paint(Graphics g){
    g.setColor(Color.white);
    g.fillRect(0, 0, getWidth(), getHeight());
    g.setColor(Color.black);
    for(int i=0; i<ringid.size(); i++){
      Point p=(Point)ringid.elementAt(i);
      g.drawOval(p.x-5, p.y-5, 10, 10);
    }
  }

  public void actionPerformed(ActionEvent e){
   try{
    if(e.getSource()==tagasi&&ennistushaldur.canUndo()){
      ennistushaldur.undo();
    }
    if(e.getSource()==edasi&&ennistushaldur.canRedo()){
      ennistushaldur.redo();
    }
   }catch(Exception ex){ex.printStackTrace();}
   repaint();
  }

  public static void main(String argumendid[]){
    JFrame f=new JFrame("Ennistamine");
    f.setSize(250, 200);
    f.setLocation(200, 100);
    f.getContentPane().add(new Ennistus());
    f.setVisible(true);
  }
}


class LisatavRing extends AbstractUndoableEdit{
  Point p;
  public LisatavRing(int x, int y){
    p=new Point(x, y);
    edasi();
  }

  void edasi(){
    Ennistus.ringid.add(p);
  }

  public void redo(){
    super.redo();
    edasi();
  }

  public void undo(){
    super.undo();
    Ennistus.ringid.remove(p);
  }
}
```

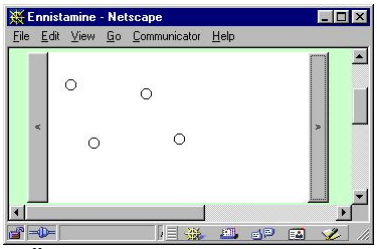
Tühi väli



Lisatud ringid



Kaks ringi tagasi võetud



Üks endine taas ekraanile pandud

## *Dialoogiaknad*

Enamike graafiliste programmeerimisvahendite juures on esimeseks näiteks lihtne teateaken. Java keeles ei kipu see lihtne käsk kohe silma alla jääma, kuid olemas on sellegipoolest. `Swing`i vahendite hulgas on `JOptionPane`, mille abil suhtlemise tarvis dialoogiaknaid luua annab. Kaks lihtsamat näidet kohe allpool. `System.exit(0)` on koodi viimaseks käsuks pandud, et virtuaalmasin oma töö rahus lõpetaks. Nii nagu muude graafika- ning muusikavahenditega, jääb ka teateakna avamisel programmi sees miski sisemine lõim töösse ning peaprogrammi lõppemisega ei lülitata virtuaalmasinat välja. Kui aga viimatimainitud käsklus lõppu panna, siis suleb see masina ning programm lõpetab rahulikult oma töö. Number 0 meetodi parameetrina näitab, et kõik lõppes õnnelikult ning mingeid lahendamata probleeme ei jäänud.

```
import javax.swing.*;
public class SwingiTeateaknad{
  public static void main(String[] argumendid){
     JOptionPane.showMessageDialog(new JFrame(), "Tervitus");
     JOptionPane.showMessageDialog(new JFrame(), "Tervitus", "Sõbralik teade",
```

```
        JOptionPane.PLAIN_MESSAGE);
      System.exit(0);
  }
}
```


Kui soovida pakutud teatele kinnitust või ümber lükkamist, siis aitab selleks käsklus showConfirmDialog. Edasi tuleb lihtsalt käituda vastavalt kinni püütud vastusele.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class SwingiDialoog3 extends JApplet implements ActionListener{
  JButton nupp=new JButton("Vajuta");
  JTextField tekst1=new JTextField();
  public SwingiDialoog3(){
    Container c=getContentPane();
    c.setLayout(new GridLayout(2, 1));
    c.add(nupp);
    c.add(tekst1);
    nupp.addActionListener(this);
  }
  public void actionPerformed(ActionEvent e){
     if(JOptionPane.showConfirmDialog(
         this, "Kas uneaeg on käes", "Valik on sinu",
         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE
        )==JOptionPane.OK_OPTION){
       tekst1.setText("Head ööd");
     } else {
       tekst1.setText(" *** ");
     }
  }
  public static void main(String[] argumendid){
    JFrame f=new JFrame("Valikudialoog");
    f.getContentPane().add(new SwingiDialoog3());
    f.setSize(200, 200);
    f.setVisible(true);
  }
}
```


Teate sisestamist lubava akna loomiseks on käsklus JOptionPane.showInputDialog. Programmi töös jäädakse rahumeeli kasutaja sisestust ootama ning pärast nupulevajutust liigutakse taas rahumeeli edasi.

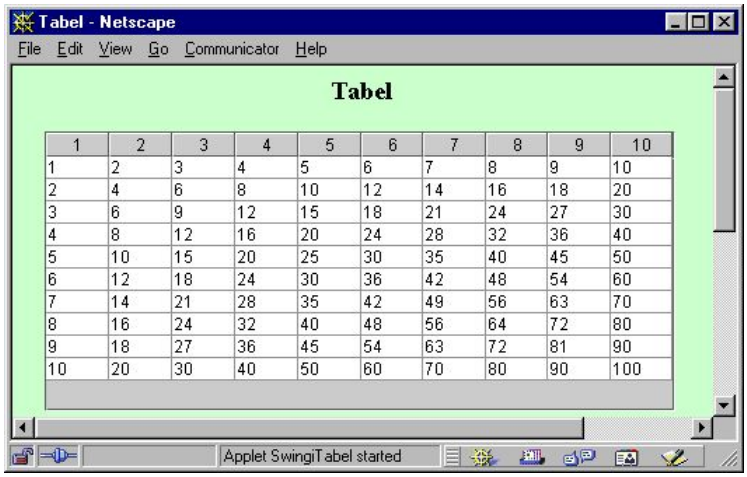
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class SwingiDialoog4 extends JApplet implements ActionListener{
  JButton nupp=new JButton("Vajuta");
  JTextField tekst1=new JTextField();
  public SwingiDialoog4(){
    Container c=getContentPane();
    c.setLayout(new GridLayout(2, 1));
    c.add(nupp);
    c.add(tekst1);
    nupp.addActionListener(this);
```

```
  }
  public void actionPerformed(ActionEvent e){
     tekst1.setText(JOptionPane.showInputDialog("Mis su nimi on?"));
  }
}
```


## *Tabel*

Andmete tabelina esitamiseks on Swingi paketti eraldi komponent loodud. Lihtsamal juhul tuleb luua näidatavatest elementidest kahemõõtmeline Object tüüpi massiiv, massiivi põhjal JTable tüüpi komponent ning paluda saadud andmed ekraanile näidata. JTable loomise konstruktoris JTable tabel=new JTable(andmed, andmed[0]); soovitakse ette saada kaks parameetrit: kõigepealt kahemõõtmeline massiiv lehel asuvate andmete kohta ning teise parameetrina ühemõõtmeline massiiv tulpade nimedega. Siin näites on antud tulpade nimedeks suure massiivi esimene rida ning nagu jooniselt näha, on sealt saadud numbrid ka ilusti tulpade pealkirjadeks vastu võetud.

```
import java.awt.*;
import javax.swing.*;
public class SwingiTabel extends JApplet {
  static JTable korrutustabel(){
  Object[][] andmed=new Object[10][10];
       for (int i=1;i<=10;i++)
       for (int j=1;j<=10;j++)
        andmed[i-1][j-1]=i*j+"";
       JTable tabel=new JTable(andmed, andmed[0]);
         //andmed ja pealkiri, milleks on ühega korrutamise rida.
       return tabel;
}

public void init(){
  getContentPane().add(new JScrollPane(korrutustabel()));
}

public static void main(String args[]) {
 JFrame f=new JFrame("Swingitabel");
 f.setSize(250,250);
 f.getContentPane().add(new JScrollPane(korrutustabel()));
 f.setVisible(true);
}
}
```




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

### *Kujundatava tekstiga paneel*

Harilikul tekstiväljal tuleb kogu tekstile määrata ühesugune kuju ning värv. Aastaid on pidanud veebikaudsete suhtlussüsteemide kirjutajad välja mõtlema imenippe kasutajate eristamiseks ning muul puhul värvide ja kirjatüüpidega mängimiseks. JTextPane aga võimaldab igale lisatavale tekstilõigule määrata omapoolsed atribuudid ning kujundamine muutub märksa paindlikumaks. Kui kord on atribuutide kogum omistatud ühele objektile, siis võib seda kogumit edaspidi mitmes kohas tarvitada – kõikjal, kus on soovi samade parameetritega teksti järele.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.text.*;
public class SwingiTekst extends JApplet{
  SimpleAttributeSet sinine=new SimpleAttributeSet();
  SimpleAttributeSet kursiiv=new SimpleAttributeSet();
  public SwingiTekst(){
   try{
    StyleConstants.setForeground(sinine, Color.blue);
    StyleConstants.setItalic(kursiiv, true);
    JTextPane tekstipaneel=new JTextPane();
    getContentPane().add(tekstipaneel, BorderLayout.CENTER);
    tekstipaneel.getDocument().insertString(0, "Tere, ", null);
    tekstipaneel.getDocument().insertString(tekstipaneel.getDocument().getLength(),
       "armas ", kursiiv);
    tekstipaneel.getDocument().insertString(tekstipaneel.getDocument().getLength(),
       "kool.", sinine);
   } catch (Exception e){
     e.printStackTrace();
   }
  }
  public static void main(String argumendid[]){
    JFrame f=new JFrame("Swingi tekst");
    f.setSize(200, 100);
    f.setLocation(100, 100);
    f.getContentPane().add(new SwingiTekst());
    f.setVisible(true);
  }
}
```

### *Veebiseilur*

JEditorPane peal on võimalik edukalt näidata nii HTML (3.2) kui RTF-vormingus dokumente. Samuti tunneb komponent ära vajutused veebiviidetel ning nendele on võimalik kuular külge panna. Nõnda annab küllalt lihtsa vaevaga kokku panna lihtne veebiseilur, mille abil tavalised leheküljed täiesti vaadatud saab. Loodud paneelile piisab lihtsalt anda käsklus `setPage`, millele antakse ette vastav URL ning muu eest hoolitseb juba komponent ise. Programmi struktuur võib välja näha veidi harjumatu, sest suhteliselt lahkesti on kasutatud sisemisi klasse. Nii HyperlinkListener kui ActionListener on loodud otse meetodi sulgude sees, mis on aga täiesti lubatud tegevus. HyperlinkListeneri sees on lehekülje uuendamise käsud pandud Runnable liidest realiseeriva sisemise klassi run meetodi sisse ning tõmmatakse SwingUtilities.invokeLater käsu abil eraldi lõimes käima alles pärast seda, kui viitevajutuse peale tööle hakanud lõim on jõudnud oma töö lõpuni. Nii püütakse vältida programmi hangumist veebist andmete kohale tirimise ajaks. Eelnevalt nähtav dokument võetakse igaks juhuks hoiule, et kui lehe avamisega peaks probleeme tekkima, siis pannakse vana sisu tagasi ning kasutajale näib, nagu poleks midagi kahtlast juhtunud.

Hüperlinkide puhul reageeritakse vaid sündmusetüübile `HyperlinkEvent.EventType.ACTIVATED`, ehk kui kasutaja on hiirega viitele vajutanud ning kavatseb viimase sisu vaatama hakata. Kui soovida staatusreal hiire alla jäävate viidete aadresse näidata, nagu ametlikes seilurites tavaks, siis tuleks reageerida ka sündmustele `HyperlinkEvent.EventType.ENTERED` ning `HyperlinkEvent.EventType.EXITED`.

Veateate näitamiseks on loodud eraldi meetod. Sellisel juhul on kergem veateate kuju soovi korral muuta. Praegugi kutsutakse vastavat alamprogrammi mitmest kohast välja.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.text.*;
import java.net.*;
import java.io.*;
public class Seilur extends JPanel {
    public Seilur() {
     setLayout (new BorderLayout ());
     final JEditorPane jt = new JEditorPane();
     final JTextField input =
      new JTextField("http://www.tpu.ee");
     jt.setEditable(false);
     // reageeri viidetele
     jt.addHyperlinkListener(new HyperlinkListener () {
       public void hyperlinkUpdate(final HyperlinkEvent e) {
           if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
             SwingUtilities.invokeLater(new Runnable() {
              public void run() {
               Document doc = jt.getDocument();
               try {
                URL url = e.getURL();
                jt.setPage(url);
                input.setText (url.toString());
               } catch (IOException io) {
                 //vahetus ebaõnnestus
                 veateade();
                 jt.setDocument (doc);
               }
              }
             });
           }
         }
     });
     JScrollPane pane = new JScrollPane();
     pane.setBorder (
      BorderFactory.createLoweredBevelBorder());
     pane.getViewport().add(jt);
     add(pane, BorderLayout.CENTER);
```

```
    input.addActionListener (new ActionListener() {
      public void actionPerformed (ActionEvent e) {
       try {
        jt.setPage (input.getText());
       } catch (IOException ex) {
        veateade();
       }
      }
    });
    add (input, BorderLayout.SOUTH);
    try{jt.setPage(input.getText());}
    catch(IOException e){veateade();}
   }

   void veateade(){
    JOptionPane.showMessageDialog (
    Seilur.this, "Vigane URL",
    "Tõenäoliselt vigane URL",
    JOptionPane.ERROR_MESSAGE);
   }

   public static void main(String argumendid[]){
    JFrame f=new JFrame("Veebiseilur");
    f.getContentPane().add(new Seilur());
    f.setSize(300, 300);
    f.setVisible(true);
   }
}
```


## *Kokkuvõte*

Swingi paketis on hulk klasse, mis peaksid aitama kasutajal programmiga meeldivalt suhelda. Samuti on püütud nende loomisel silmas pidada erivahendite kasutajaid (ekraanilt lugejad, lihtsustatud klaviatuurid). Action-liidese abil saab samastada menüüelemendi ning tööriistariba nupu. Tabelisse paigutada aitab JTable ning teksti näidata ja redigeerida saab JEditorPane ning mitme muu klassi abil.

## *Ülesandeid*

**Swingi tabel**

- Loo nulle ja x-e sisaldav 3x3 JTable.
- Luba kasutajal andmeid muuta ning loe kokku, mitu risti on märgitud.
- Lase kasutajal andmed valida valikmenüüst.

**Swingi puu**

- Loo puu, mille juure nimeks on "1" ning lehtedeks "2" ja "3".
- Loo puu, mille lehtedeks on arvud ühest tuhandeni. Oksteks algarvud nii, et mööda okstest koosnevat teed pidi korrutades jõutaks lehe väärtuseni. Mitme teguri korral panna väiksem juure poole.
- Lisaks eelmisele peab saama määrata sisestatavate arvude vahemikku. Arvu küsimise peale avatakse puu tee soovitud leheni.

**Progressiriba**

- Liiguta progressiriba väärtust algusest lõpuni.
- Ring liigub ühest servast teise. Ühes sellega kasvab progressiriba väärtus.
- Ring liigub ekraanil kümme korda ühest servast teise ja tagasi. Ühe progressiriba väärtus kasvab kogu ulatuses paremale liikumise ajal, teise väärtus vasakule liikumise ajal ning kolmas kasvab tasapisi, jõudes maksimumi liikumise lõpuks.

**Slaider**

- Loo ekraanile Swingi slaider.
- Lisa slaiderile suured kriipsud 5 ning väikesed 3 ühiku järel. Lisa tekstiväli. Slaideri väärtust näidatakse tekstiväljas. Slaideri all on skaala.
- Lisaks eelmisele muudetakse tekstiväljas oleva numbri muutmisel slaideri väärtust. Skaala väärtuse osa on tumedam (ClientProperty JSlider.isFilled), skaalaks on sõnad "külm", "leige", "soe", "tuline".

**Valikupaneel**

- Loo valikupaneel Eesti suuremate linnade nimedega.
- Lisa igale linnale ikoon ning vihjeks (ToolTip) ligikaudne rahvaarv. Iga paneeli sisuks kirjuta suure kirjaga seda linna läbiva jõe nimi. Pane linnad automaatselt iga viie sekundi tagant vahetuma.
- Iga linna puhul saab kasutaja anda viiepallisüsteemis hinnangu vaatamisväärsuste, toitlustuse ning teedevõrgu kohta. Kasutaja saab soovi korral linnu lisada. Valikute vastused salvestatakse faili.

**Tekstiredaktor valikupaneelil**

- Loo kahe valikuga valikupaneel, kus kummalgi paneelil paikneb tekstiala.
- Lisa kummalegi paneelile nupud failist lugemise ning sinna salvestamise kohta. Esimene paneel on seotud failiga katse1.txt ning teine failiga katse2.txt.
- Kasutaja saab avada ja salvestada soovitud tekstifaile. Kõik avatud failid on näha valikupaneeli paneelidena.

**Menüüd**

- Loo raam menüüga "Tervitused" elementidega "Sünnipäev" ning "Kooli lõpp"
- Lisa alammenüü, valitav menüüelement ((J)CheckboxMenuItem) ning toimiv menüükäsk programmi töö lõpetamiseks.
- Lisa menüü "numbrid" alammenüüdega "0".."9", igaühes asuvad vastava kümne numbrid. Numbrile vajutamisel kirjutatakse vastav arv tekstivälja. Loo hüpikmenüü (PopupMenu)

kolmega jaguvate menüüelementide lubamiseks/keelamiseks ning algarvuliste menüüelementide peitmiseks/näitamiseks.

**Sisemised raamid.**

- Loo aken kolme sisemise raamiga.
- Kasuta sisemisi raame pildifailide näitamiseks. Kasutaja valitud pildifail avatakse uues raamis. Raami suuruse muutudes muutub ka pilt. Sulgemisnupule vajutades raam kaob.
- Lisaks eelmisele saab avatud pilti redigeerida ning jpeg-failina salvestada.

**Swingi komponendid**

- Loo JFrame.
- Paiguta selle ülaserva pildiga nupp.
- Nupule vajutades valitakse JColorChooseri abil nupu värv.
- Raami vasakusse serva paigutatakse puu (JTree). Puu elementideks on arvud ühest kümneni
- Igal esimese taseme elemendil on samuti alamelemendid ühest kümneni.
- Valitud elemendi väärtus kirjutatakse lehe allservas olevasse tekstivälja.
- Paremal servas olevale nupule vajutades avatakse JFileChooser, kust valitud pildifail joonistatakse raami keskele paigutatud paneelile.