Digitaalheli

MIDI abil võib kord kokku pandud pillide helisid uuesti esitada, mängida saab vaid helikõrgusega ning kestusega. Olematute pillide hääli aga sünteesida ei õnnestu, samuti tuleb loobuda muudest heliefektidest. Kui palju kõlari membraan mingil hetkel välja venitatud on, seda otsustab meie eest helikaart või vastav süsteemiprogramm ning programmeerijal selle koha pealt kuigi palju kaasa rääkimist ei ole. "Hariliku" muusika loomise puhul ongi nii hea, sest saame rahumeeli noodikõrgustele ja -kestustele mõelda ning ei pruugi tehniliste andmete peale üleliia oma energiat kulutada. Kui aga tahta olemasolevat häält moonutada, kaja tekitada või hoopis uusi kõlasid luua, siis tuleb hakata mõtlema helilainete kuju peale. Edasi tuleb koostatud kuju kvantida, et saaks kusagil arvumassiivis hoida igale ajahetkele vastavat heligraafiku väärtust sarnaselt nagu matemaatikaski võime esitada funktsiooni x-idele vastavate y-ite massiivina, kus x-teljel oleks aeg ning y-il helirõhu kõrvalekalle tasakaaluasendist. Kvantimissagedusest (kandesagedusest) sõltub helikõvera edasiandmise kvaliteet. Mängitav kõver luuakse punkte ühendavatest sirgete järgi ning mida tihedamalt on punkte, seda lähedasemalt õnnestub säilitada esialgse lindistatud või välja arvutatud kõvera kuju.

Lihtne piiks

Järgnevalt on püütud kokku panna võimalikult lihtne heli, mida ka kõrvaga kuulda oleks. Kvantimissageduseks on määratud 1000 mõõtmist sekundis. Helirõhud on kordamööda määratud maksimumile ja miinimumile. Sedasi on väljastatava heli sageduseks pool kvantimissagedusest ehk 500 hertsi. Mõõtmistäpsuseks on üks bait ning heli väljastatakse ühe kanalina (mono, mitte stereo). Nõnda saab mõõtmistulemused panna ilusti baidimassiivi, kus igale kvandile vastab üks bait. AudioFormat'i abil määratakse, millist tüüpi andmed tulemas on. AudioSystem'i abil küsitakse operatsioonisüsteemilt SourceDataLine, kuhu saadetud heliandmed jõuavad lõpuks kasutaja kõrvadeni. Pärast voo avamist võime sinna andmeid saata kirjutades andmeid loodud voogu.

```
import javax.sound.sampled.*;
public class Piiks1{
 public static void main(String[] args) throws Exception {
   int     kandesagedus =1000;
   byte[] andmed=new byte[5*kandesagedus]; //5 sekundit
   for(int i=0; i<andmed.length; i++){
     if(i%2==0)andmed[i]=(byte) 127;
     else      andmed[i]=(byte)-128;
   }
   AudioFormat formaat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
                kandesagedus, 8, 1, 1, kandesagedus, false); //8bitine heli, 1, kanal, 1 bait raami kohta
   SourceDataLine line = (SourceDataLine) AudioSystem.getLine(
       new DataLine.Info(SourceDataLine.class, formaat,  AudioSystem.NOT_SPECIFIED)
   );
   line.open(formaat);
   line.start();
   line.write(andmed, 0, andmed.length);
   System.exit(0);
 }
}
```

Siinusekujuline laine

Soovides koostada keerukama kujuga helilaineid, peab ka kandesagedus suurem olema. Kui soovida helilainele anda siinuse lainjat kuju, peab ühe täisvõnke kohta rohkem mõõtmisi olema kui et ainult üks laine harja ning teine põhja tarvis. All näites ~20 mõõtmist täisvõnke kohta peaks juba silmaga kaugelt vaadates enamjaolt sinusoidi sarnase kuju andma ning ka kõrvaga kuulates ei tohiks karedus kuigivõrd tunda anda. Liiatigi kui nii tehniliste vahendite kui inimkõrva poolsed tasandused juurde arvata. Ka suurema kvantimissageduse kuid üheaegsete mitmete keerulisemate helide korral ei mõõdeta

ühe heli andmeid oluliselt täpsemalt.

Veidi seletusi arvutamise kohta. Kui iga baidi järjekorranumbrist võtta siinus ning selle väärtus panna kvandile, siis saaksime sinusoidi, mille lainepikkus oleks 2*Pi baiti ehk 10000 Hz kandesageduse puhul tuleks helisageduseks 10000/6,28 ehk ~ 1600 Hz. Helitugevus oleks aga imetilluke, sest kasutada olevast 256-ühikulisest (8-bitisest) piirkonnast on tarvitatud vaid vahemikku miinust ühest üheni ehk siinuse väljundväärtust. Vahemikku saab kergesti suurendada, korrutades tulemuse kordajaga (siin juhul 100). Sel juhul kõigub väärtus miinus saja ning +100 vahel ning kasutatakse suurem osa (200/256) väljundpiirkonnast. Soovides helilainet kiiremini võnkuma panna, tuleb sama aja (baitide) jooksul suurendada arvu, millest siinust võetakse, kiiremini. Suurema kandesageduse puhul on aga ühe baidi jaoks eraldatud aeg väiksem. Baidi järjekorranumbri ja kandesageduse suhe aga näitab aega sekundites ning kui soovime, et kandesageduse muutumisel jääks helisagedus samaks, siis peab siinuse arvutamisel kandesagedus olema murrujoone all. Math.sin(2*Math.PI*nr/kandesagedus) puhul tehtaks parajasti üks võnge sekundis, et saaksime kõrgemat (kuuldavat) heli, selleks tuleb siinuse parameetriks olev väärtus soovitud sagedusega läbi korrutada. Nii saamegi lainekujulise heli arvutamiseks valemi andmed[nr]=(byte)(100*Math.sin(2*Math.PI*nr*sagedus/kandesagedus));

```
import javax.sound.sampled.*;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Piiks2Rakend extends Applet implements ActionListener{
 Button nupp=new Button("Piiksu");
 public Piiks2Rakend(){
   setLayout(new BorderLayout());
   add(nupp);
   nupp.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e){
  try{
   int  kandesagedus =10000;
   int  sagedus=440;  //440 hertsi
   int  nr=0;
   byte[] andmed=new byte[5*kandesagedus]; //5 sekundit
   while(nr<andmed.length){
     andmed[nr]=(byte)(100*Math.sin(2*Math.PI*nr*sagedus/kandesagedus));
     nr++;
   }
   AudioFormat formaat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
               kandesagedus, 8, 1, 1, kandesagedus, false); //8bitine heli, 1, kanal, 1 bait raami kohta
   SourceDataLine line = (SourceDataLine) AudioSystem.getLine(
       new DataLine.Info(SourceDataLine.class, formaat,  AudioSystem.NOT_SPECIFIED)
   );
   line.open(formaat);
   line.start();
   line.write(andmed, 0, andmed.length);
   line.close();
  }catch(Exception ex){ex.printStackTrace();}
 }
 public static void main(String[] argumendid){
   Frame f=new Frame("Piiks");
   Piiks2Rakend p=new Piiks2Rakend();
   f.add(p);
   f.setSize(200, 100);
   f.setVisible(true);
   p.actionPerformed(null);
 }
}
```

Tõusev heli

Soovides heli panna ühtlaselt tõusma, peab selle sagedust püsiva ajavahemiku järel (kahe) kordistama. Kaks korda suurem helisagedus annab oktaavi jagu kõrgema heli. Nõnda võib heli arvutamisel siinuse parameetrina aja (andmebaidi järjekorranumbri) kordajana kasutada astmefunktsiooni, mille astendajat pidevalt kasvatada.

```
andmed[nr]=(byte)(100*Math.sin(2*Math.PI*nr*
          Math.pow(2, nr*tous_oktaavites/andmed .length)*sagedus/kandesagedus));
```

Nõnda esineb aega tähistav number valemis kahes kohas. Esmalt annab sin (nr*tegur), kus tegur=2*Pi*sagedus/kandesagedus välja ühtlase sinusoidi, mille järgi võib pidevat samal kõrgusel püsivat tooni kuulata. Tooni pidevaks kergitamiseks tuleb siinuse parameeter läbi korrutada teisegi, nüüd juba ajast sõltuva koefitsiendiga, mille tulemusena saigi kokku eelpool toodud valem.

```
import javax.sound.sampled.*;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Piiks2aRakend extends Applet implements ActionListener{
 Button nupp=new Button("Piiksu");
 public Piiks2aRakend(){
   setLayout(new BorderLayout());
   add(nupp);
   nupp.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e){
  try{
   int  kandesagedus =10000;
   int  sagedus=100;
   int  nr=0;
   double tous_oktaavites=2;
   byte[] andmed=new byte[5*kandesagedus];
   while(nr<andmed.length){
     andmed[nr]=(byte)(100*Math.sin(2*Math.PI*nr*
          Math.pow(2, nr*tous_oktaavites/andmed.length)*sagedus/kandesagedus));
     nr++;
   }
   AudioFormat formaat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
               kandesagedus, 8, 1, 1, kandesagedus, false);
   SourceDataLine line = (SourceDataLine) AudioSystem.getLine(
       new DataLine.Info(SourceDataLine.class, formaat,  AudioSystem.NOT_SPECIFIED)
   );
   line.open(formaat);
   line.start();
   line.write(andmed, 0, andmed.length);
   line.close();
  }catch(Exception ex){ex.printStackTrace();}
 }
 public static void main(String[] argumendid){
   Frame f=new Frame("Tõusev toon");
   Piiks2aRakend p=new Piiks2aRakend();
   f.add(p);
   f.setSize(200, 100);
   f.setVisible(true);
   p.actionPerformed(null);
 }
}
```

Kahebaidine kvant

Kvaliteetsema heli edastamiseks kaheksabitisest kvandist ei piisa. Jutu ning lihtsama heli saab selle abil küllalt hästi edasi anda, kuid linnuhäälte või orkestrimuusika puhul läheb märgatav kõrvale kuuldav osa kaduma. Kui mõõtmistäpsust suurendada, jääb rohkem algsest helist alles. Arvutusvalemite põhimõte jääb ikka samaks, kuid kui enne kaheksabitise heli korral oli kvandi suurimaks võimalikuks väärtuseks 127, siis kuueteistbitise juures annab maksimumi 32767. Seda tuleb valemite juures arvestada, muidu jääb hääl väga vaikseks või pole seda lihtsama helitehnika juures üldse kuulda.

Suurema mõõtmistäpsuse puhul tuleb lõivu maksta suurema mahuga. Kui ennist sai läbi aetud iga kvandi juures ühe baidiga, siis nüüd läheb vaja kahte.

```
byte[] andmed=new byte[5*kandesagedus*mitmeBitineHeli/8]; //5 sekundit
```

Kvandile vastavat numbrit arvutatakse sama valemi järgi, vaid valjust on niivõrd suurendatud, et

see jääks parajalt kuulatavatesse piiridesse

```
int tulemus=(int)(valjus*Math.sin(Math.PI*nr*
       Math.pow(2, 0.5*Math.sin(nr*Math.PI/andmed.length))*sagedus/kandesagedus));
```

Baidimassiivis säilitamiseks ning edasiandmiseks tuleb täisarv kahe baidi vahel jagada. Siinses kodeeringus pannakse ettepoole viimane (madalam) bait, taha esimene (kõrgem) bait, kuid sõltuvalt formaadist võib järjekord ka teistpidine olla. Viimase baidi kättesaamiseks jäetakse alles vaid sellele baidile vastavad bitid, muud asendatakse nullidega. Tüübimuunduse abil baidiks muundamise järel jõuabki soovitud väärtus sihtmassiivi kohale. Täisarvust teise baidi kättesaamiseks nihutatakse see kõigepealt kaheksa biti jagu paremale esimese baidi kohale ning tehakse eelpool toodud tehe. Arvus 0xFF ehk 255 on esimese baidi kõik bitid ühed ning & tehtega algsest arvust vaid need ühed alles jättes, millele 255 ühed vastu panna on, tulebki kokku ühebaidiline väärtus. Nüüd piisab see vaid andmemassiivi üle kanda, et seda pärast kuulata annaks.

```
     andmed[nr++]=(byte)(tulemus & 0xFF);       //viimane bait
     andmed[nr++]=(byte)(tulemus >> 8 & 0xFF);  //eelviimane bait
```

Ning näide ise:

```
import javax.sound.sampled.*;
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;


public class Piiks3Rakend extends Applet implements ActionListener, Runnable{
 Button nupp=new Button("Piiksu");
 Checkbox korda=new Checkbox("Korda");
 public Piiks3Rakend(){
   setLayout(new BorderLayout());
   add(nupp);
   add(korda, BorderLayout.SOUTH);
   korda.setState(true);
   nupp.addActionListener(this);
 }
 public void actionPerformed(ActionEvent e){
   new Thread(this).start();
 }
 public void run(){
  try{
   int  kandesagedus =44100;
   int  sagedus=400;
   int  mitmeBitineHeli=16;
   int  kanaliteArv=1;
   int  valjus=7000;  //max 32767
   int  nr=0;
   byte[] andmed=new byte[5*kandesagedus*mitmeBitineHeli/8]; //5 sekundit
   while(nr<andmed.length){
     int tulemus=(int)(valjus*Math.sin(Math.PI*nr*
       Math.pow(2, 0.5*Math.sin(nr*Math.PI/andmed.length))*sagedus/kandesagedus));
     andmed[nr++]=(byte)(tulemus & 0xFF);       //viimane bait
     andmed[nr++]=(byte)(tulemus >> 8 & 0xFF);  //eelviimane bait
        }
   AudioFormat formaat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
               kandesagedus, mitmeBitineHeli, kanaliteArv,
               kanaliteArv*mitmeBitineHeli/8, kandesagedus, false);
   SourceDataLine line = (SourceDataLine) AudioSystem.getLine(
       new DataLine.Info(SourceDataLine.class, formaat,  AudioSystem.NOT_SPECIFIED)
   );
   line.open(formaat);
   line.start();
   do{
     line.write(andmed, 0, andmed.length);
   }while(korda.getState());
   line.close();
  }catch(Exception ex){ex.printStackTrace();}
 }
 public static void main(String[] argumendid){
   Frame f=new Frame("Piiks");
   Piiks3Rakend p=new Piiks3Rakend();
   f.add(p);
   f.setSize(200, 100);
   f.setVisible(true);
```

```
    p.actionPerformed(null);
  }
}
```

Digitaalheli redaktor.

Mõnerealiste koodinäidetega õnnestus läbi proovida märgatav osa kvanditud heliga ümberkäimise tehnilistest käskudest. Ehkki lõpuks taandub suurem osa tegemisi otseste kvantide väärtuste väljaarvutamise ning tulemuste analüüsi peale, õnnestub kõrvale ehitatud kestprogrammi abil matemaatiliselt kuivana tunduvat arvutamist kasutajale märgatavalt mugavamaks muuta. Heliredaktorit on märkimisväärne osa meist kasutanud, paljudel pole aga aimu, kui kerge või keeruline võiks olla selline rakendus kokku panna. Siin on püütud alustada võimalikult lihtsast näitest ning korduste teel jõutud rohkemate võimalustega rakenduseni. Siin ettetulevaga sarnaseid küsimusi peab lahendama enamiku kasutajalt sisestust ootava helidega seotud rakenduste puhul.

Lihtsaim heliredaktor õnnestub kirjutada mõneteistkümne reaga ning tema ainuke oskus on koodi otse sisse kirjutatud. Olgu selleks siis faili salvestamine vaiksemaks, tagurpidi või kahe helifaili sisu kokkuliitmine. Üldiseks malliks ikka, et tuleb algsed failid sisse lugeda, leida juurdepääs üksikute kvantide andmete juurde. Seal soovitud muutused sisse viia või olemasolevate andmete põhjal uus massiiv kokku kirjutada ning lõpuks tulemus kasutajale ette mängida või faili salvestada. Rakendused asuvad keerukamaks minema selle osa juurest, kus kasutajal antakse voli määrata, milliseid muutusi andmetega ette võtta.

Märgistusala

Esimeseks näiteks on redaktor, kus kasutaja saab faili programmi mällu lugeda, seal andmeid kustutada ja kopeerida ning siis tulemuse uude faili talletada. Heli mängimise ega helikõvera vaatamise võimalust ei pakuta. Sobiva lõigu märkimiseks on loodud eraldi klass Margistusala1. Klassi nähtavaks osaks on lõuend, kus kasutaja saab omale soovitava lõigu märkida. Jäetakse lihtsalt meelde protsendid, kust maalt kuhu maale sedakorda kasutaja oma lõigu märkis. Alale eraldatud laiust tähistatakse 100% ehk 1,0-ga. Kui kasutaja juhtus märkima näiteks loo keskmise kolmandiku, siis see talletatakse muutujates väärtused 0,33 ja 0,66 abil vastavalt algus- ja lõppprotsendi tarbeks. Muutuja kaudu kannatab isendile ette anda mälus hoitava andmeploki kogupikkuse ning sellele vastavalt teatavad meetodid algKaader ja loppKaader, mitmendast kaadrist mitmenda kaadrini kasutaja märgistatud ala paikneb. Et testiks on klassil küljes ka main-meetod, saab komponenti ka eraldi katsetada ning veenduda, et kasutajal soovitud lõike ka märgistada õnnestub.

```
import java.awt.*;
import java.awt.event.*;
public class Margistusala1 extends Panel implements MouseListener{
  int kogupikkus=50000;
  double algusprotsent=0.1, loppprotsent=0.4;
  Color margistusvarv=Color.black;
  public Margistusala1(){
    addMouseListener(this);
  }
  public void paint(Graphics g){
    g.setColor(margistusvarv);
    g.fillRect(
      (int)(algusprotsent*getWidth()), 0,
      (int)((loppprotsent-algusprotsent)*getWidth()), getHeight()
    );
  }
  public void mousePressed(MouseEvent e){
    algusprotsent=e.getX()/(double)getWidth();
  }
  public void mouseReleased(MouseEvent e){
    loppprotsent=e.getX()/(double)getWidth();
    if(loppprotsent<algusprotsent){
      double abi=algusprotsent; algusprotsent=loppprotsent; loppprotsent=abi;
    }
    repaint();
  }
  public int algKaader(){
```

```
    return (int)(algusprotsent*kogupikkus);
  }
  public int loppKaader(){
    return (int)(loppprotsent*kogupikkus);
  }
  public void mouseClicked(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
  public static void main(String[] arg){ //ala testimine
    Frame f=new Frame();
    f.add(new Margistusala1());
    f.setSize(300, 200);
    f.setVisible(true);
  }
}
```

Iseenesest ei pea loodud märgistusala sugugi olema seotud helitöötlusprogrammiga. Sarnaselt on sinna külge võimalik haakida mõnda muud osa andmete väljarealdamist vajavat rakendust - olgu selleks või nimede eraldamine telefoniraamatust. Järgnevalt aga näha, kuidas heliredaktor eelnevalt kirjeldatud märgistusala vajab.

Kopeerimine ja kleepimine

Et rakenduses saaks pärast avamist midagi heliga peale hakata, selleks tuleb mõni olemasolev helifail sisse lugeda. Tänuväärseks vahendiks on AudioInputStream, mille abil vähemasti teoreetiliselt on võimalik ühtse lähenemise kaudu kätte saada andmebaite kõigist formaatidest, millega Java hakkama saama peaks. Kaheksabitise heli puhul olen märganud, et AudioInputStreamil on raskusi tuvastamisega, kas andmeid hoitakse märgita või märgiga arvudena ning tõenäoliselt võib leida ebakõlasid ehk muudegi vormingute puhul, kuid üldiselt on klassi võimalused täiesti kasutatavad.

Järgnevalt eeldatakse, et tegemist on ühebaidise ehk kaheksabitise heliga ning helifaili kvandid loetakse ükshaaval failist ja pannakse mälupuhvrisse. Kui andmed otsas (read() väljastab -1), siis muudetakse puhvri sisu baidimassiiviks. Nõnda mäluvoo abil on mugav toimida, kui pole teada saabuvate andmete pikkust.

```
AudioInputStream sisse=AudioSystem.getAudioInputStream(new File(tfLae.getText()));
ByteArrayOutputStream malu=new ByteArrayOutputStream();
int nr=sisse.read();
while(nr!=-1){      //loetakse voo sisu mälupuhvrisse
  malu.write(nr);
  nr=sisse.read();
}
andmed=malu.toByteArray();
ala.kogupikkus=andmed.length;
formaat=sisse.getFormat();
```

Mälust faili kirjutamisel luuakse kõigepealt AudioInputStream mälus olevast andmemassiivist ning saabunud voog suunatakse soovitud formaadis faili kettal.

```
  AudioInputStream ais=new AudioInputStream(
   new ByteArrayInputStream(andmed), formaat, andmed.length
 );
```

```
    AudioSystem.write(ais, AudioFileFormat.Type.WAVE, new File(tfSalvesta.getText()));
```

Kasutaja soovitud redigeerimisoperatsioonides tuleb mälus olevate andmetega lihtsalt soovitud muutused ette võtta. Soovides, et märgitud ploki kohal oleks helis vaikus, tuleb kõikide selle ploki andmete väärtused kirjutada ühesugusteks. Kui tegemist on märgiga täisarvuga kvantidega, siis sobib null vaikimisväärtuseks täiesti. Kui märgiga osa puudub (tüübiks näiteks PCM_UNSIGNED), siis
võiks vaikuse korral olla väärtus pool vastava arvu bittidega tekitatavast maksimumväärtusest.

```
    for(int i=ala.algKaader(); i<=ala.loppKaader(); i++){
      andmed[i]=(byte)0;
    }
```

Kopeerimise puhul luuakse kõigepealt sobiva pikkusega puhver ning siis kopeeritakse ükshaaval andmeplokis märgitud baidid puhvrisse.

```
    puhver=new byte[ala.loppKaader()-ala.algKaader()];
    for(int i=0; i<puhver.length; i++){
      puhver[i]=andmed[ala.algKaader()+i];
    }
```

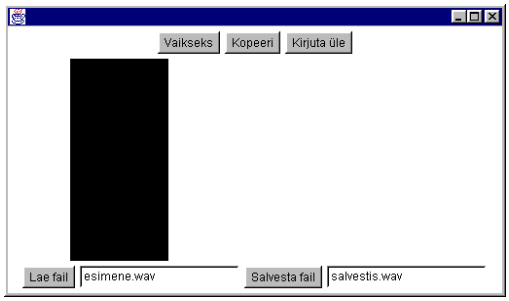
Andmete ülekirjutuse juures kirjutatakse puhvris olevad baidid andmemassiivi baitidele peale alates kursori asukohast. Nii nagu vanematel tekstiredaktoritel on tunduvalt tavalisem ülekirjutus kui vahelekirjutus, nii ka heliredaktori puhul on andmeid eelmiste peale lihtsam panna kui vahele. Nõnda ei pea hakkama ülejäänud andmeid edasi liigutama. Ning et pool muna on parem kui tühi koor, siis piirdume siin pealekirjutusega.

```
    if(puhver==null)return;
    for(int i=0; i<puhver.length; i++){
      andmed[ala.algKaader()+i]=puhver[i];
    }
```

Edasi lihtsakoelise redaktori kood tervikuna.

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.sound.sampled.*;
public class Digiheliredaktor1 extends Panel implements ActionListener{
  byte[] andmed;
  byte[] puhver;
  AudioFormat formaat;
  Button kopeeri=new Button("Kopeeri");
  Button kirjutaYle=new Button("Kirjuta üle");
  Button tyhjenda=new Button("Vaikseks");
  Margistusala1 ala=new Margistusala1();
  Button lae=new Button("Lae fail");
  TextField tfLae=new TextField("esimene.wav", 20);
  Button salvesta=new Button("Salvesta fail");
  TextField tfSalvesta=new TextField("salvestis.wav", 20);
  public Digiheliredaktor1(){
    Panel p=new Panel();
    p.add(tyhjenda);
    p.add(kopeeri);
    p.add(kirjutaYle);
    Panel alapaneel=new Panel();
    alapaneel.add(lae);
    alapaneel.add(tfLae);
    alapaneel.add(salvesta);
    alapaneel.add(tfSalvesta);
    setLayout(new BorderLayout());
    add(p, BorderLayout.NORTH);
    add(ala, BorderLayout.CENTER);
    add(alapaneel, BorderLayout.SOUTH);
    kopeeri.addActionListener(this);
```

```
    kirjutaYle.addActionListener(this);
    tyhjenda.addActionListener(this);
    lae.addActionListener(this);
    salvesta.addActionListener(this);
  }
  public void actionPerformed(ActionEvent e){
    if(e.getSource()==lae){laeFail(); }
    if(e.getSource()==salvesta){salvestaFail(); }
    if(e.getSource()==tyhjenda){tyhjendaLoik(); }
    if(e.getSource()==kopeeri){kopeeriLoik(); }
    if(e.getSource()==kirjutaYle){kirjutaLoikYle(); }
  }
  public void laeFail(){
   try{
    AudioInputStream sisse=AudioSystem.getAudioInputStream(new File(tfLae.getText()));
    ByteArrayOutputStream malu=new ByteArrayOutputStream();
    int nr=sisse.read();
    while(nr!=-1){      //loetakse voo sisu mälupuhvrisse
      malu.write(nr);
      nr=sisse.read();
    }
    andmed=malu.toByteArray();
    ala.kogupikkus=andmed.length;
    formaat=sisse.getFormat();
   }catch(Exception viga){
     viga.printStackTrace();
   }
  }

  public void salvestaFail(){
   try{
      AudioInputStream ais=new AudioInputStream(
       new ByteArrayInputStream(andmed), formaat, andmed.length
     );
    AudioSystem.write(ais, AudioFileFormat.Type.WAVE, new File(tfSalvesta.getText()));
   }catch(Exception viga){
     viga.printStackTrace();
   }
  }

  public void tyhjendaLoik(){
    for(int i=ala.algKaader(); i<=ala.loppKaader(); i++){
      andmed[i]=(byte)0;
    }
  }

  public void kopeeriLoik(){
    puhver=new byte[ala.loppKaader()-ala.algKaader()];
    for(int i=0; i<puhver.length; i++){
      puhver[i]=andmed[ala.algKaader()+i];
    }
  }

  public void kirjutaLoikYle(){
    if(puhver==null)return;
    for(int i=0; i<puhver.length; i++){
      andmed[ala.algKaader()+i]=puhver[i];
    }
  }


  public static void main(String[] argumendid){
    Frame f=new Frame();
    f.add(new Digiheliredaktor1());
    f.setSize(400, 300);
    f.setVisible(true);
  }
}
```



Redigeeritud heli kuulamine eraldi programmis.

Teine ring

Märgistusalale lisati helikõvera joonistusoskus. Nii pole vaja vaid ligikaudselt protsente piiluda. Kui on tegemist muutuva valjusega heliga, siis õnnestub kergesti välise pildi järgi eristada, millal uus rõhuline koht algab. Märgistusvärv joonistatakse alla ning helikõver selle peale, nõnda õnnestub mõlemat üheaegselt vaadata.

Joonistamisel liigutakse tsükliga läbi kogu andmeploki, tõmmates joone eelmisest punktist jooksvasse.
(andmed[i]&0xff)/250.0*getHeight() lahtiseletatuna:

0xff on täisarv, mille bitid kahendsüsteemis on kõik ühed. Kui byte-tüüpi väärtus &-operaatoriga sellise arvuga ühendada, siis tulemuseks on int-väärtus, millel samad bitid kui algsel byte-väärtusel, ent väärtuseks on positiivne arv 0 ja 255 vahel byte -128 ja +127 asemel. 250ga läbijagamine ning ala kõrgusega läbi korrutamine hoolitseb, et helilained oleksid võrdelised näidatava ala suurusega. Kui andmeid pole veel määratud, siis on vastava muutuja väärtuseks null ning helikõverat pole vaja ega võimalik joonistada.

Andmete edastamiseks märgistusalale loodi meetod seaAndmed. Seal antakse ette uus andmemassiiv, mille peale jääb märgistusala muutuja osutama, andmeid ei kopeerita. Lisaks jäetakse meelde massiivi pikkus ning nullitakse muud vajalikud muutujad. repaint() teatab, et pilt tuleb joonistada uuendatud andmete järgi.

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

public class Margistusala2 extends Panel implements MouseListener{
  private int kogupikkus=50000;
  byte[] andmed;  //heli andmed
  double algusprotsent=0.1, loppprotsent=0.1;
  Color margistusvarv=Color.green;
  Color joonistusvarv=Color.black;


  public Margistusala2(){
    addMouseListener(this);
  }
  public void paint(Graphics g){
    g.setColor(margistusvarv);
    g.fillRect(
      (int)(algusprotsent*getWidth()), 0,
      (int)((loppprotsent-algusprotsent)*getWidth()), getHeight()
    );
    g.drawLine(  //kursor
      (int)(algusprotsent*getWidth()), getHeight()/2,
      (int)(algusprotsent*getWidth()), getHeight()
    );
    if(andmed!=null){
      g.setColor(joonistusvarv);
      for(int i=1; i<andmed.length; i++){
        g.drawLine(
          (int)((i-1.0)/andmed.length*getWidth()),
          (int)(getHeight()-(andmed[i-1]&0xff)/250.0*getHeight()),
          (int)(((double)i)/andmed.length*getWidth()),
          (int)(getHeight()-(andmed[i]&0xff)/250.0*getHeight())
        );
      }
    }
  }


  public void seaAndmed(byte[] uuedAndmed){
    andmed=uuedAndmed;
    kogupikkus=andmed.length;
    algusprotsent=0;
    loppprotsent=0;
    repaint();
  }
```

```
  public void mousePressed(MouseEvent e){
    algusprotsent=e.getX()/(double)getWidth();
  }

  public void mouseReleased(MouseEvent e){
    loppprotsent=e.getX()/(double)getWidth();
    if(loppprotsent<algusprotsent){
      double abi=algusprotsent; algusprotsent=loppprotsent; loppprotsent=abi;
    }
    repaint();
  }

  public int algKaader(){
    return (int)(algusprotsent*kogupikkus);
  }

  public int loppKaader(){
    return (int)(loppprotsent*kogupikkus);
  }
  public void mouseClicked(MouseEvent e){}
  public void mouseEntered(MouseEvent e){}
  public void mouseExited(MouseEvent e){}
}
```

Redaktorile tulid juurde nupud vahelekleepimise ning vahelt lõikamise tarbeks. Samuti õnnestub maha mängida nii kogu andmemassiivis olevat heli kui kasutaja märgitud helilõiku.

Nii lõikamise kui kleepimise puhul kasutatakse massiivide kopeerimiseks käsku System.arraycopy, mis pidada suurte andmemahtude korral olema arvutile valutum kui tuhandete üksikute baitide ükshaaval kopeerimine. Nagu allpool koodist näha, koostatakse lõikamise puhul kõigepealt uus massiiv, mis on algsest väljalõigatava osa jagu lühem. Edasi kopeeritakse märgitud osa puhvrisse, et vajadusel õnnestuks see sobivasse kohta kleepida. System.arraycopy soovib enesele viis argumenti. Esimeseks massiiv kust kopeerida, praegusel juhul muutuja nimega andmed. Edasi järjekorranumber, mitmendast elemendist kopeerimist alustada, ehk ala.algKaader(). Siis tuleb massiiv, kuhu kopeeritavad andmed paigutada - nimeks siin juhul puhver. Siis järjekorranumber, kust alates paigutatakse elemente uude massiivi. Et soovime puhvrisse paigutada lõigatu alates algusest siis selleks väärtuseks 0. Ja lõpuks kopeeritavate elementide arv ehk märgistatud lõigu pikkus.

Järgnevate kopeerimistega veel algsest andmeplokist nii märgistuseelne kui märgistusjärgne lõik uude massiivi ning lõikus ongi valmis.

```
  public void loikaLoik(){
    byte[] uus=new byte[andmed.length-(ala.loppKaader()-ala.algKaader())];
    puhver=new byte[ala.loppKaader()-ala.algKaader()];

    System.arraycopy(andmed, ala.algKaader(), puhver, 0, (ala.loppKaader()-ala.algKaader()));
    System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
    System.arraycopy(andmed, ala.loppKaader(), uus,
                     ala.algKaader(), andmed.length-ala.loppKaader());
    andmed=uus;
    uuendaAla();
  }
```

Kleepimine näeb lõikamisega küllalt sarnane välja. Vaheks ainult, et uus massiiv on eelmisest pikem ning puhvri sisu tuleb kursorile eelneva ning kursorile järgneva ploki vahele paigutada.

```
  public void kleebiLoik(){
    if(puhver==null)return;
    byte[] uus=new byte[
       andmed.length+puhver.length-(ala.loppKaader()-ala.algKaader())];
    System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
    System.arraycopy(puhver, 0, uus, ala.algKaader(), puhver.length);
    System.arraycopy(andmed, ala.loppKaader(), uus,
                ala.algKaader()+puhver.length, andmed.length-ala.loppKaader());
```

```
    andmed=uus;
    uuendaAla();
  }
```

Kui jooksvalt võimalik oma töö vilju kuulata, siis sujub heli töötlemine ladusamalt. Hea on järele proovida, et kõik plaanitu ka kõrvade jaoks sobilikult kokku saab ning tulemust võib ka enne faili salvestamist ja selle maha mängimist kontrollida.

Siin näites ei hoita helikanalit pidevalt redigeerimisprogrammi all kinni, vaid küsitakse kanal alles siis, kui kavatsetakse mängima hakata. Tähtsaim kanali küsimise juures on formaat, milles kavatsetakse andmeid teele saatma hakata. Selleks jääb praegusel juhul sama algsest helifailist loetud formaat. Käsuga write saadetakse andmed helikaardi poole teele ning drain ootab, kuni saadetud andmed on jõutud maha mängida. Edasi võib kanali selleks korraks sulgeda, et liialt palju arvuti ressursse programm enese käes ei hoiaks.

```
  public void mangi(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     kanal.write(andmed, 0, andmed.length);
     kanal.drain();
     kanal.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }
```

Märgistatud lõigu mängimine erineb eelmisest vaid selle poolest, et kogu massiivi pikkuse asemel tuleb määrata baidid/kaadrid, kust alates ja kui palju neid helikaardile mängimiseks saata tuleb.

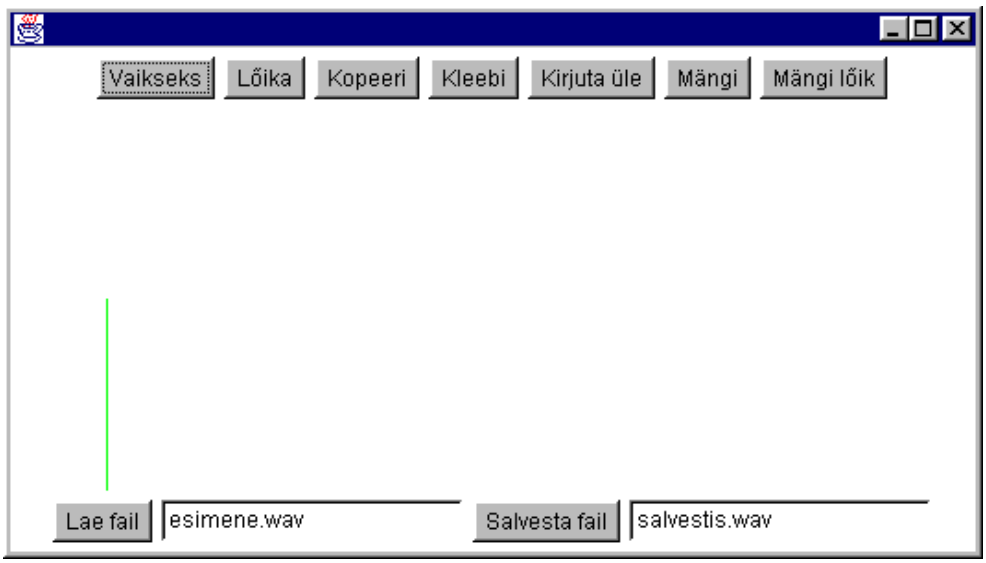
```
     kanal.write(andmed, ala.algKaader(), ala.loppKaader()-ala.algKaader());
```

Ning teise redaktori kood tervikuna.

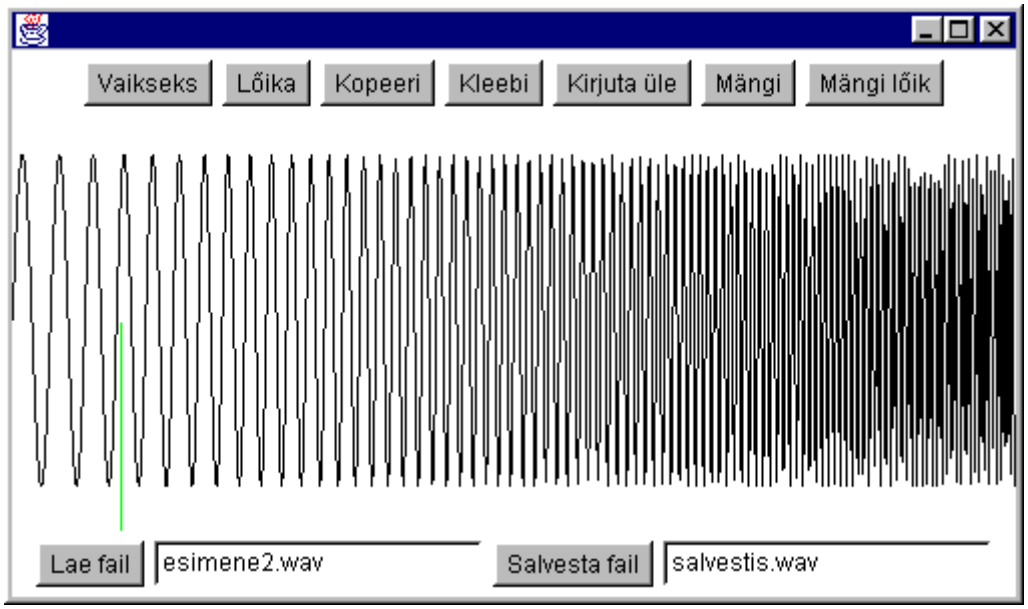
```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.sound.sampled.*;
public class Digiheliredaktor2 extends Panel implements ActionListener{
  byte[] andmed;
  byte[] puhver;
  AudioFormat formaat;
  Button kopeeri=new Button("Kopeeri");
  Button loika=new Button("Lõika");
  Button kleebi=new Button("Kleebi");
  Button kirjutaYle=new Button("Kirjuta üle");
  Button tyhjenda=new Button("Vaikseks");
  Button mangi=new Button("Mängi");
  Button mangiLoik=new Button("Mängi lõik");
  Margistusala2 ala=new Margistusala2();
  Button lae=new Button("Lae fail");
  TextField tfLae=new TextField("esimene.wav", 20);
  Button salvesta=new Button("Salvesta fail");
  TextField tfSalvesta=new TextField("salvestis.wav", 20);
  SourceDataLine kanal;
  public Digiheliredaktor2(){
    Panel p=new Panel();
    p.add(tyhjenda);
    p.add(loika);
    p.add(kopeeri);
    p.add(kleebi);
    p.add(kirjutaYle);
    p.add(mangi);
    p.add(mangiLoik);
    Panel alapaneel=new Panel();
    alapaneel.add(lae);
    alapaneel.add(tfLae);
    alapaneel.add(salvesta);
    alapaneel.add(tfSalvesta);
    setLayout(new BorderLayout());
    add(p, BorderLayout.NORTH);
```

```
  add(ala, BorderLayout.CENTER);
  add(alapaneel, BorderLayout.SOUTH);
  loika.addActionListener(this);
  kopeeri.addActionListener(this);
  kleebi.addActionListener(this);
  kirjutaYle.addActionListener(this);
  tyhjenda.addActionListener(this);
  lae.addActionListener(this);
  salvesta.addActionListener(this);
  mangi.addActionListener(this);
  mangiLoik.addActionListener(this);
}
public void actionPerformed(ActionEvent e){
  if(e.getSource()==lae){laeFail(); }
  if(e.getSource()==salvesta){salvestaFail(); }
  if(e.getSource()==tyhjenda){tyhjendaLoik(); }
  if(e.getSource()==loika){loikaLoik(); }
  if(e.getSource()==kopeeri){kopeeriLoik(); }
  if(e.getSource()==kleebi){kleebiLoik(); }
  if(e.getSource()==kirjutaYle){kirjutaLoikYle(); }
  if(e.getSource()==mangi){mangi(); }
  if(e.getSource()==mangiLoik){mangiLoik(); }
}
public void laeFail(){
 try{
  AudioInputStream sisse=AudioSystem.getAudioInputStream(new File(tfLae.getText()));
  System.out.println(sisse.getFormat());
  ByteArrayOutputStream malu=new ByteArrayOutputStream();
  int nr=sisse.read();
  while(nr!=-1){      //loetakse voo sisu mälupuhvrisse
    malu.write(nr);
    System.out.print(nr+" ");
    nr=sisse.read();
  }
  andmed=malu.toByteArray();
  //ala.kogupikkus=andmed.length;
  ala.seaAndmed(andmed);
  formaat=sisse.getFormat();
  uuendaAla();
 }catch(Exception viga){
   viga.printStackTrace();
 }
}

public void salvestaFail(){
 try{
    AudioInputStream ais=new AudioInputStream(
     new ByteArrayInputStream(andmed), formaat, andmed.length
   );
  AudioSystem.write(ais, AudioFileFormat.Type.WAVE, new File(tfSalvesta.getText()));
 }catch(Exception viga){
   viga.printStackTrace();
 }
}

public void loikaLoik(){
  byte[] uus=new byte[andmed.length-(ala.loppKaader()-ala.algKaader())];
  puhver=new byte[ala.loppKaader()-ala.algKaader()];


  System.arraycopy(andmed, ala.algKaader(), puhver, 0, (ala.loppKaader()-ala.algKaader()));
  System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
  System.arraycopy(andmed, ala.loppKaader(), uus,
                   ala.algKaader(), andmed.length-ala.loppKaader());
  andmed=uus;
  uuendaAla();
}


public void tyhjendaLoik(){
  for(int i=ala.algKaader(); i<=ala.loppKaader(); i++){
    andmed[i]=(byte)0;
  }
  uuendaAla();
}

public void kopeeriLoik(){
  puhver=new byte[ala.loppKaader()-ala.algKaader()];
  for(int i=0; i<puhver.length; i++){
    puhver[i]=andmed[ala.algKaader()+i];
  }
}

public void kleebiLoik(){
  if(puhver==null)return;
  byte[] uus=new byte[
```

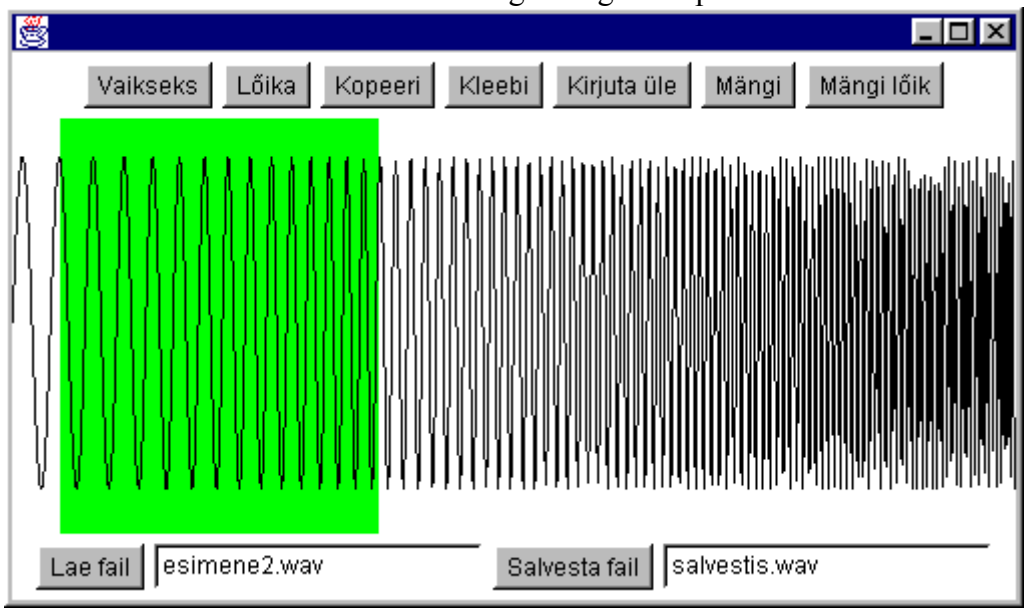
```
      andmed.length+puhver.length-(ala.loppKaader()-ala.algKaader())];
    System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
    System.arraycopy(puhver, 0, uus, ala.algKaader(), puhver.length);
    System.arraycopy(andmed, ala.loppKaader(), uus,
                ala.algKaader()+puhver.length, andmed.length-ala.loppKaader());
    andmed=uus;
    uuendaAla();
  }

  /**
  * Joonistusala uuendamine pärast andmete muutust.
  */
  void uuendaAla(){
    ala.seaAndmed(andmed);
  }

  public void kirjutaLoikYle(){
    if(puhver==null)return;
    for(int i=0; i<puhver.length; i++){
      andmed[ala.algKaader()+i]=puhver[i];
    }
    uuendaAla();
  }

  public void mangi(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     kanal.write(andmed, 0, andmed.length);
     kanal.drain();
     kanal.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }

  public void mangiLoik(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     kanal.write(andmed, ala.algKaader(), ala.loppKaader()-ala.algKaader());
     kanal.drain();
     kanal.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }

  public static void main(String[] argumendid){
    Frame f=new Frame();
    f.add(new Digiheliredaktor2());
    f.setSize(400, 300);
    f.setVisible(true);
  }
}
```



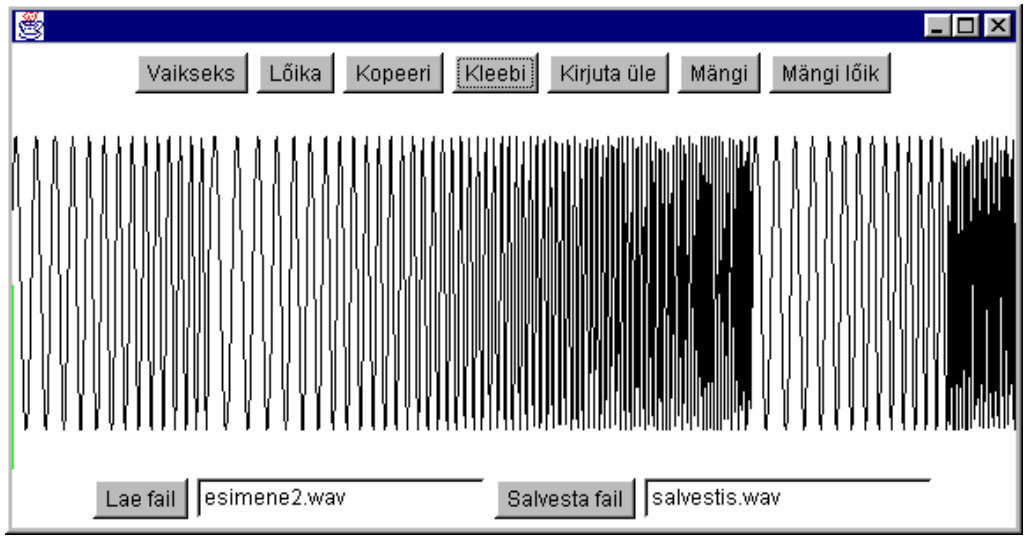
Avatud tühi redaktor



Suureneva helisagedusega faili pilt



Märgistatud lõik



Madalama sagedusega heli kopeerituna kõrgesageduslikuma vahele.

Kolmas ring

Tegemist pole sugugi veel täieliku redaktoriga, kuid siinset rakendust peaks saama juba mitmete "päris" helide juures rakendada. Arvestatud on nii mono- kui stereoheli ning kaheksa- kui kuueteistbitiseid märgiga ning märgita kvante, kusjuures kahebaidise kodeeringu puhul võib kõrgem bait olla nii ees- (big endian) kui tagapool (little endian). Mõnedki kontrollid on jäänud koodi lühiduse huvides peale panemata, kuid eks rakenduse eriomaduste lisamise järel tulevad nagunii kontrolli vajavad kohad paremini esile.

Võrreldes eelnenuga hoitakse märgistusala komponendis lisaks andmetele meeles formaati, sest samad andmed võivad eri formaatide puhul hoopis eri tulemuse anda. Näiteks stereo puhul loetakse samast andmemassiivist kvante kordamööda ning samade andmete monona mängimine ei pruugi viisi sugugi äratuntavaks jätta.

Märgistuse alustus- ja lõpetuskohta ei hoita enam meeles protsendina vaid kaadri järjekorranumbriga. Nii on võimalik täpsemalt määrata, milline koht märgistatud on. Terviküksusena käivad koos kaadrid, mida on mõtet vaid korraga kopeerida, lisada või eemaldada. Kui ühebaidise ja ühe kanaliga heli puhul oli kaadri suuruseks üks bait, siis kahebaidise stereoheli puhul on kaadri suuruseks juba neli baiti ning seda nelikut pole pea kusagil töötluse juures mõistlik lahutada.

Joonistusala on märgistusala komponendi sisse loodud eraldi sisemise lõuendiklassina. Nii on mugavam paluda alal enese sisse helikõver joonistada ning samal ajal saab märgistusala alla serva paigutada kerimisribad asukoha ja koefitsientide määramiseks ilma, et peaks programmis liialt energiat kulutama arvutamisele hoolitsemaks et lainekõver kogemata kerimisribade alla ei satuks.

Suuremaks arvutamist nõudvaks ülesandeks on soovitud kanalilt soovitud järjekorranumbriga helikvandi küsimine. Kvandi väärtus väljastatakse täisarvuna. Tulgu see siis ühe või kahebitine ehk märgiga või märgita.

```
int kysiKanaliltKvant(int kanal, int kaader){
```

Kõigepealt leitakse kaadri algus - koht massiivi algusest kaadri suuruse ja kaadri järjekorranumbri jagu baite edasi.

```
int kaadriAlgus=kaader*formaat.getFrameSize();
```

Edasi kaadri seest kvandi algus. Esimese kanali (kanali number 0 puhul) langevad kaadri ja kvandi algus kokku, muul juhul tuleb kvandi laiuse jagu edasi liikuda.

```
            int kvandiAlgus=kaadriAlgus+kanal*formaat.getSampleSizeInBits()/8;
```

Kaheksabitise heli korral sobibki vastava baidi väärtused arvuks kirjutatuna võtta - juhul kui on tegemist märgita täisarvudega salvestuse puhul.

```
            if(formaat.getSampleSizeInBits()==8){
              return andmed[kvandiAlgus] & 0xFF;
            }
```

Kuueteistbitise heli korral tuleb järgnevast kahest baidist kokku kombineerida kvandi asukohta määrav arv.

```
            int b1=andmed[kvandiAlgus] & 0xFF;
            int b2=andmed[kvandiAlgus+1] & 0xFF;
```

Märgita arvude puhul on arvutus veidi lihtsam.

```
            if(formaat.getEncoding().equals(AudioFormat.Encoding.PCM_UNSIGNED)){
```

Kui kõrgem bait eespool, siis tuleb seda loodavas arvus ühe baidi jagu vasakule nihutada.

```
             if(formaat.isBigEndian()){
              return b1 << 8 | b2;
             } else {
```

Muul juhul tuleb nihutamiseks võtta madalam bait.

```
              return b2 << 8 | b1;
             }
            }
```

Kui kvante esitatakse märgiga arvude abil, siis tuleb hoolitseda, et vasakpoolne märgibitt kaduma ei läheks. Üheks võimaluseks peaks kõigepealt olema omistada väärtus kahebaidisele täisarvule short, kus vasakpoolseim bitt hoolitseb märgi eest. Edasine muundamine int-muutujaks enam väärtust ei muuda.

```
if(formaat.getEncoding().equals(AudioFormat.Encoding.PCM_SIGNED)){
             if(formaat.isBigEndian()){
              return (int)((short)(b1 << 8 | b2));
             } else {
              return (int)((short)(b2 << 8 | b1));
             }
            }
```

Kui tegemist tundmatu kodeeringuga, siis väljastatakse 0.

```
            return 0;
        }
```

Eraldi tähelepanu väärib kerimisribadest väärtuste välja küsimine. Esimeses ribas hoitakse väärtusi nullist sajani ning loetud sisu saab ette kujutada protsendina loo kogupikkusest. Nõnda siis leitakse joonistuseAlgKaader nihkeprotsendi ja kaadrite arvu korrutisena.

Nii x- kui y-suunalise suurenduse puhul on pandud kerimisriba väärtus arvu kaks astendajat muutma. Nõnda suurendab kümne pügala jagu skaalal liikumine laiust kaks korda ning järgmise kümne pügala jagu liikumine jälle kaks korda. Selline lähenemine peaks vaatajale loomulikum tunduma.

```
        public void adjustmentValueChanged(AdjustmentEvent e){
           joonistuseAlgKaader=sb1.getValue()*kaadriteArv/100;
           piksleidKaadriKohta=Math.pow(2, sb2.getValue()/10.0-7);
           suurendusKordaja=Math.pow(2, sb3.getValue()/10.0);
           ala.repaint();
        }
```

Ka kanali joonistamise tarbeks on eraldi alamprogramm loodud. Nõnda õnnestub paint-meetodit mõnevõrra lihtsustada. Meetodile antakse ette graafiline kontekst kuhu joonistada, kanal kust andmeid võtta ning ala suurus ja asukoht helikõvera paigutamiseks.

```
void joonistaKanal(Graphics g, int kanal, int vasak, int yla,
        int laius, int korgus){
```

Et poleks põhjust kogu helifaili jagu kaadreid läbi käia, vaid võiks piirduda ainult nähtavate kaadritega, siis leitakse kohe algul mahtuvate kaadrite arv

```
  int mahtuvateKaadriteArv=(int)(laius/piksleidKaadriKohta);
```

Kogu helikõver moodustatakse üksikutest joontest. Et joone puhul on vaja teada kahte otspunkti, siis on mõistlik punktid ühekaupa välja arvutada ning iga kord eelmise punkti andmed meeles pidada, et oleks võimalik vanast punktist uude joon tõmmata.

Kvandi väärtuse enese saab käsuga kysiKanaliltKvant. Muid tehteid kasutatakse helikõvera parajaks etteantud alasse paigutamiseks. Muutuja keskArv hoiab eneses vastava vormingu helikvantide väärtust vaikuseolekus, ehk väärtust, mis peaks sattuma kõvera keskele vertikaalsihti mööda. Suurima võimaliku väärtuse ehk maxArvuga läbi jagamine ning korgus/2-ga korrutamine venitab graafiku etteantud piiridesse. Kerimisribast sõltuva suurendusKordaja abil saab seda edaspidi täiendavalt skaleerida.

```
  int vanax=0;
  int vanay=(int)(yla+korgus/2-(kysiKanaliltKvant(kanal, joonistuseAlgKaader)-keskArv)
*korgus*suurendusKordaja/2/maxArv);
```

Samm näitab, mitme kaadri võrra igal korral edasi liigutakse. Kui kaadreid peaks paiknema laiuse ekraanipunktil rohkem kui üks, siis suurendatakse sammu niivõrd, et iga järgnev arvutatav kaader satuks uuele ekraanipunktile. Selliselt on võimalik hoolitseda, et ka pikkade helifailide andmete ligikaudne ekraanile joonistamine liialt kaua aega ei võtaks. Kui iga kaadri jaoks pole ekraanil punkti, siis paratamatult jääb kujutis mõnevõrra ebatäpne. Küll aga peaksid näha olema tähtsamad kõikumised valjuses ehk amplituudis. Kui iga kaadri asukohta tähistada punktiga ekraanil, tuleks kujutis mõnevõrra täpsem, kuid samas oleks heliosa ekraanil tihedate joonte tõttu pea ühtlaselt must ning joonistamine nõuaks masinalt enam jõudu. Sammu aga ühest väiksemaks ei lasta minna, sest järgmine alumine täisarvuline väärtus oleks null ning nõnda jääks programm igavesse tsüklisse.

```
  int samm=(int)(1.0/piksleidKaadriKohta);
  if(samm<1){samm=1;}
  for(int k=joonistuseAlgKaader+1; k<kaadriteArv &&
k<joonistuseAlgKaader+mahtuvateKaadriteArv; k+=samm){
```

Edasi juba uus koordinaatide arvutus, sama põhimõttega kui ülalpoolgi. Ning ka x-suunal tuleb kaadri numbri, esimese joonistatava kaadri numbri ning venituskoefitsendi abil ekraanile sobiva punkti asukoht leida.

```
    int uusx=(int)((k-joonistuseAlgKaader)*piksleidKaadriKohta);
    int uusy=(int)(yla+korgus/2-(kysiKanaliltKvant(kanal, k)-
            keskArv)*korgus*suurendusKordaja/2/maxArv);
    g.drawLine(vanax, vanay, uusx, uusy);
    vanax=uusx; vanay=uusy;
  }
 }
```

Ning nagu ikka - märgistusala kood tervikuna. Loodetavasti aitavad siin sees paiknevad kommentaarid mõistmist samuti veidi selgemaks teha.

```
import java.awt.*;
import java.awt.event.*;
import javax.sound.sampled.*;


/**
* Abiklass Digiheliredaktor3 tarbeks. Võimaldab näidata helilaine kuju
* ning märgistada kasutaja soovitud ploki andmetest.
*/
public class Margistusala3 extends Panel implements MouseListener, AdjustmentListener{
        /**
        *  Heliandmete interpreteerimiseks vajalik formaat. Eeldatavalt antakse
        *  ette koos andmetega.
        */
        AudioFormat formaat;
        /**
        * Massiiv näidatavate andmete hoidmiseks.
        */
       byte[] andmed=new byte[0];
       /**
       * Märgistatud ala alustava kaadri järjekorranumber.
       */
       protected int algusKaader;
       /**
       * Märgistatud ala lõpetava kaadri järjekorranumber.
       */
       protected int loppKaader;
       /**
       * Ala helikõvera tegelikuks näitamiseks.
       */
       protected JoonistusAla ala=new JoonistusAla();
       /**
       * Märgistatud piirkonna taustavärv. Samuti kursori värv.
       */
       Color margistusvarv=Color.green;
       /**
       * Helikõvera värv.
       */
       Color joonistusvarv=Color.black;
       /**
       * Koefitsient näitamaks mitu pikslit laiuse suunas on ühe kaadri
       * andmete näitamiseks. Ühest väiksem väärtus tähistab, et
       * sama ekraanipunkti peale peab mahtuma mitu kaadri väärtust.
       */
       double piksleidKaadriKohta=1;
       /**
       * Koefitsient vertikaalsihis suurenduse tarbeks. Kui väärtuseks on 1, siis
       * võtab heli oma maksimumväärtuste puhul piikide tipud kogu kanali andmete
        * joonistamiseks ette nähtud ala ulatuses.
       */
       double suurendusKordaja=1;
       /**
       * Kaadri number, millest alates on helikõver joonistusalas näha.
       */
       int joonistuseAlgKaader=0;
       /**
       * Helikaadrite arv etteantud andmeplokis. Arvutatakse uue ploki etteandmisel.
       */
       int kaadriteArv;
       /**
       * Helikõvera väärtus hariliku vaikuse puhul. Märgiga täisarvuga
       * kvantide puhul väärtuseks 0, märgita kvantide puhul pool täisväljalöögist.
       */
       int keskArv;
       /**
       * Helikvandi suurim võimalik väärtus jooksva vormingu puhul.
       */
       int maxArv;
       /**
       * Kerimisriba määramaks, millisest kohast alates asutakse helikõverat ekraanil näitama.
       * Kerimisriba väärtusi saab kujutada protsentidena kogu lõigu pikkusest.
       */
       Scrollbar sb1=new Scrollbar(Scrollbar.HORIZONTAL, 1,  1, 0, 100);
       /**
       * Kerimisriba määramaks helikõvera horisontaalsuunalist väljavenitatust.
       * Joonistamisel kasutatakse logaritmilist skaalat.
       */
       Scrollbar sb2=new Scrollbar(Scrollbar.HORIZONTAL, 50, 1, 0, 100);
       /**
       *  Kõrgusesuunalise helikõvera väljavenitatuse määramine.
       */
       Scrollbar sb3=new Scrollbar(Scrollbar.HORIZONTAL, 10, 1, 0, 100);
```

```
        /**
        *  Konstruktor paigutuse ja kuularite sättimiseks.
        */
       public Margistusala3(){
               setLayout(new BorderLayout());
               add(ala);
              ala.addMouseListener(this);
              Panel ribaPaneel=new Panel(new GridLayout(1, 6));
              ribaPaneel.add(new Label("Asukoht:"));
              ribaPaneel.add(sb1);
              ribaPaneel.add(new Label("Laius:"));
              ribaPaneel.add(sb2);
              ribaPaneel.add(new Label("Kõrgus:"));
              ribaPaneel.add(sb3);
              add(ribaPaneel, BorderLayout.SOUTH);
              sb1.addAdjustmentListener(this);
              sb2.addAdjustmentListener(this);
              sb3.addAdjustmentListener(this);
       }

       /**
       * Soovitud kanali ja kaadri järgi helikvandi küsimine
       * täisarvuna. Väärtus antakse vastavalt andmetega
       * kaasas olevale vormingule.
       */
       int kysiKanaliltKvant(int kanal, int kaader){
           int kaadriAlgus=kaader*formaat.getFrameSize();
           int kvandiAlgus=kaadriAlgus+kanal*formaat.getSampleSizeInBits()/8;
           if(formaat.getSampleSizeInBits()==8){
             return andmed[kvandiAlgus] & 0xFF;
           }
           int b1=andmed[kvandiAlgus] & 0xFF;
           int b2=andmed[kvandiAlgus+1] & 0xFF;
           if(formaat.getEncoding().equals(AudioFormat.Encoding.PCM_UNSIGNED)){
            if(formaat.isBigEndian()){
             return b1 << 8 | b2;
            } else {
             return b2 << 8 | b1;
            }
           }
           if(formaat.getEncoding().equals(AudioFormat.Encoding.PCM_SIGNED)){
            if(formaat.isBigEndian()){
             return (int)((short)(b1 << 8 | b2));
            } else {
             return (int)((short)(b2 << 8 | b1));
            }
           }
           return 0;
       }



        /**
        * Kanali helikõvera joonistus vastavalt etteantud parameetritele.
        */
        void joonistaKanal(Graphics g, int kanal, int vasak, int yla, int laius, int korgus){
          int mahtuvateKaadriteArv=(int)(laius/piksleidKaadriKohta);
          int vanax=0;
          int vanay=(int)(yla+korgus/2-(kysiKanaliltKvant(kanal, joonistuseAlgKaader)-keskArv)
*korgus*suurendusKordaja/2/maxArv);
          int samm=(int)(1.0/piksleidKaadriKohta);
          if(samm<1){samm=1;}
          for(int k=joonistuseAlgKaader+1; k<kaadriteArv &&
k<joonistuseAlgKaader+mahtuvateKaadriteArv; k+=samm){
            int uusx=(int)((k-joonistuseAlgKaader)*piksleidKaadriKohta);
            int uusy=(int)(yla+korgus/2-(kysiKanaliltKvant(kanal, k)-keskArv)
*korgus*suurendusKordaja/2/maxArv);
            g.drawLine(vanax, vanay, uusx, uusy);
            vanax=uusx; vanay=uusy;
          }
        }



        /**
        * Märgistusalale uute andmete seadmine või andmemuutusest teatamine.
        * Leitakse edaspidiseks joonistamiseks tarvilikud konstandid.
        */
       public void seaAndmed(byte[] uuedAndmed, AudioFormat uusFormaat){
              andmed=uuedAndmed;
              formaat=uusFormaat;
              algusKaader=0;
              loppKaader=0;
              kaadriteArv=andmed.length/formaat.getFrameSize();
              maxArv=255;
```

```
        if(formaat.getSampleSizeInBits()==16){
          maxArv=65535;
        }
         if(formaat.getEncoding()==AudioFormat.Encoding.PCM_SIGNED){
           keskArv=0;
           maxArv=maxArv/2;
         } else {
           keskArv=maxArv/2;
         }
        ala.repaint();
}

 /**
 * Kerimisribade muutuste peale konstantide väljaarvutus.
 */
public void adjustmentValueChanged(AdjustmentEvent e){
   joonistuseAlgKaader=sb1.getValue()*kaadriteArv/100;
   piksleidKaadriKohta=Math.pow(2, sb2.getValue()/10.0-7);
   suurendusKordaja=Math.pow(2, sb3.getValue()/10.0);
   ala.repaint();
}

 /**
 * Alguskaadri leidmine vastavalt hiirevajutuse asukohale.
 */
public void mousePressed(MouseEvent e){
  algusKaader=(int)(e.getX()/piksleidKaadriKohta+joonistuseAlgKaader);
}


 /**
 * Lõppkaader vastavalt hiirevajutuse asukohale. Kui märgistati
 * paremalt vasakule, siis hoolitsetakse, et algus
 * oleks ikka enne lõppu.
 */
public void mouseReleased(MouseEvent e){
        loppKaader=(int)(e.getX()/piksleidKaadriKohta+joonistuseAlgKaader);
        if(loppKaader<algusKaader){
          int abi=algusKaader; algusKaader=loppKaader; loppKaader=abi;
        }
        ala.repaint();
}


 /**
 * Andmebaidi järjekorranumber, millest alates
 * hakkab märgistatud alguskaader. Väärtused langevad
 * kokku vaid ühe kanali ja ühebaidise heli puhul.
 * Tarvilik Digiheliredaktorile teadmaks, millisest
 * baidist alates tuleb andmeid muutma või kopeerima asuda.
 */
 public int algKaader(){
   return algusKaader*formaat.getFrameSize();
 }
 /**
 * Märgistuse lõppkaadri esimese baidi järjekorranumber.
 */
 public int loppKaader(){
   return loppKaader*formaat.getFrameSize();
 }


 /**
 * Tühi meetod hiireliidese realiseerimiseks.
 */
public void mouseClicked(MouseEvent e){}
 /**
 * Tühi meetod hiireliidese realiseerimiseks.
 */
public void mouseEntered(MouseEvent e){}
 /**
 * Tühi meetod hiireliidese realiseerimiseks.
 */
public void mouseExited(MouseEvent e){}
/**
* Lõuend tegelikuks joonistamiseks.
*/
class JoonistusAla extends Canvas{
   /**
   * Kuvatakse nii kursor, märgistatud ala kui
   * kõige peale helikõver ise.
   */
   public void paint(Graphics g){
     if(formaat==null){return;}
     g.setColor(margistusvarv);
     g.fillRect(
```

```
              (int)((algusKaader-joonistuseAlgKaader)*piksleidKaadriKohta), 0,
              (int)((loppKaader-algusKaader)*piksleidKaadriKohta), getHeight()
            );
            g.drawLine(  //kursor
             (int)((algusKaader-joonistuseAlgKaader)*piksleidKaadriKohta),
               getHeight()/2,
             (int)((algusKaader-joonistuseAlgKaader)*piksleidKaadriKohta), getHeight()
            );
            if(andmed!=null){
               g.setColor(joonistusvarv);
               if(formaat.getChannels()==1){
                 joonistaKanal(g, 0, 0, 0, getWidth(), getHeight());
               } else {
                 joonistaKanal(g, 0, 0,
                    0, getWidth(), getHeight()/2);
                 joonistaKanal(g, 1, 0, getHeight()/2, getWidth(), getHeight()/2);
               }
            }
        }
    }
}
```

Redaktoriklassil enesel juurde tulnud mitmebaidiliste helide sisselugemisvõimalus, samuti lõigu mängimine korduvalt. Lisaks leiduvad nüüd koodi juures Javadoci-stiilis kommentaarid, mis teevad mahu küll pikemaks, kuid peaksid aitama meelde tuletada, milleks miski koodilõik mõeldud on.

Andmete failist sisselugemise puhul on tähtsaimaks teada, mitme baidi jagu tuleb mälus ruumi varuda. Vormingu käest on võimalik küsida kaadri suurus ning helisisendvoo käest kaadrite arv. Nende korrutis annabki heli hoidmiseks tarvilike baitide arvu. Edasi pole muud kui andmed ühe read-käskluse abil sisse lugeda.

```
    andmed=new byte[formaat.getFrameSize()*(int)sisse.getFrameLength()];
    sisse.read(andmed);
    ala.seaAndmed(andmed, formaat);
```

Kui märgistatud lõik juhtub olema lühemapoolne, siis ühekordne kuulamine tekitab vaid tunde, et "oli midagi". Täpsemaks mõistmiseks aga tuleb lõiku vähemasti paar-kolm korda kuulata. Eriti hea aga, kui õnnestub lõik pidevalt mängima panna ning siis muutusi tehes pidevalt jälgida, mis operatsioonide tulemusena muutus. Kasutaja saab kordust määrata märkeruudu kaudu.

```
  Checkbox loiguKordus=new Checkbox("Lõigu kordus");
```

Et omasoodu ja pidevalt korduv lõik muud programmi kinni ei jooksutaks ning oleks võimalik mängimise ajal maha võtta näiteks kordust tähistava märkeruudu märgistust, selleks peab korduv mängimine paiknema omaette lõimes. Et redaktori juures praegu muid omaette töötamist nõudvaid lõike pole, siis võis lõigu paigutada rakenduse enese run-meetodisse ning eraldi lõimel paluda seda jooksutada.

```
    if(e.getSource()==mangiLoik){new Thread(this).start(); }
```

Meetod run ise sarnaneb terve ploki mängimise näitega. Vaid juures on määratlus, kust alates ning kuhu maani anmeid kanalisse saatma peab. Ning ümber tsükkel, mis hoolitseb, et märgitud ruudu puhul lõiku muudkui korrataks ja korrataks.

```
  public void run(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     do{
```

```
        kanal.write(andmed, ala.algKaader(), ala.loppKaader()-ala.algKaader());
      }while(loiguKordus.getState());
      kanal.drain();
      kanal.close();
     }catch(Exception ex){
       ex.printStackTrace();
     }
   }
```

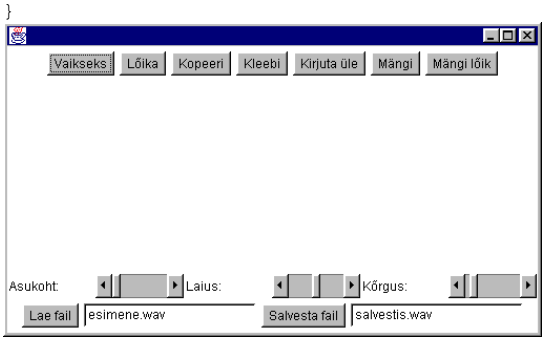
Ning kommenteeritud redaktori kolmanda versiooni kood tervikuna.

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.sound.sampled.*;


/**
* Redaktori põhiklass. Suudab näidata, redigeerida ja
* salvestada PCM-kodeeringus (nt. wav) 8 ja 16 bitise heli
* ning 1 või kahe kanaliga helifaile.
*/
public class Digiheliredaktor3 extends Panel implements ActionListener, Runnable{
  /**
  *  Mängimis- ja töötlemiskõlbulikud andmed.
  *  Sisu tarvitatakse vastavalt sisseloetud
  *  formaadile.
  */
  byte[] andmed;
  /**
  * Abipuhver kopeerimise ja lõikamise tarbeks.
  */
  byte[] puhver;
  /**
  * Sisseloetud faili formaat. Kirjeldus näiteks
  * PCM_SIGNED, 22050.0 Hz, 16 bit, stereo, little-endian, audio data
  */
  AudioFormat formaat;
  /**
  * Kopeerimisnupp. Märgitud andmed kopeeritakse puhvrisse.
  */
  Button kopeeri=new Button("Kopeeri");
  /**
  * Lõikamisnupp.
  */
  Button loika=new Button("Lõika");
  /**
  * Kleepimisnupp.
  */
  Button kleebi=new Button("Kleebi");
  /**
  * Ülekirjutusnupp. Puhvris olevad andmed kirjutatakse
  * andmeploki peale kursorist alates. Heli pikkus ei muutu.
  */
  Button kirjutaYle=new Button("Kirjuta üle");
  /**
  * Märgitud piirkonnas asendatakse kvantide väärtused nullidega.
  */
  Button tyhjenda=new Button("Vaikseks");
  /**
  * Kogu mälus oleva andmeploki sisu mängitakse inimesele korra ette.
  */
  Button mangi=new Button("Mängi");
  /**
  * Märgistatud ploki sisu mängitakse kuulajale.
  */
  Button mangiLoik=new Button("Mängi lõik");
  /**
  * Märkeruudu seesolek jätab märgitud mängitava lõigu kordama.
  * Kordamine lõppeb ruudu vabastamisel.
  */
  Checkbox loiguKordus=new Checkbox("Lõigu kordus");
  /**
  * Ala sisseloetud helifaili andmete näitamiseks, märgistamiseks
  * ning muutuste näitamiseks.
  */
  Margistusala3 ala=new Margistusala3();
  /**
  * Faili laadimisnupp
  */
  Button lae=new Button("Lae fail");
```

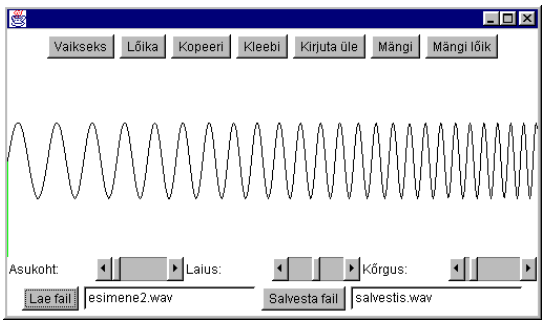
```
/**
* Laetava faili nimi.
*/
TextField tfLae=new TextField("esimene.wav", 20);
/**
* Faili salvestusnupp. Kodeering jääb samaks kui laadimisel.
*/
Button salvesta=new Button("Salvesta fail");
/**
* Salvestatava faili nime sisestus.
*/
TextField tfSalvesta=new TextField("salvestis.wav", 20);
/**
* Kanal kasutaja kuulatava heli teele saatmiseks.
*/
SourceDataLine kanal;
/**
* Paigutuse ja kuularite paika sättimine.
*/
public Digiheliredaktor3(){
  Panel p=new Panel();
  p.add(tyhjenda);
  p.add(loika);
  p.add(kopeeri);
  p.add(kleebi);
  p.add(kirjutaYle);
  p.add(mangi);
  p.add(mangiLoik);
  p.add(loiguKordus);
  Panel alapaneel=new Panel();
  alapaneel.add(lae);
  alapaneel.add(tfLae);
  alapaneel.add(salvesta);
  alapaneel.add(tfSalvesta);
  setLayout(new BorderLayout());
  add(p, BorderLayout.NORTH);
  add(ala, BorderLayout.CENTER);
  add(alapaneel, BorderLayout.SOUTH);
  loika.addActionListener(this);
  kopeeri.addActionListener(this);
  kleebi.addActionListener(this);
  kirjutaYle.addActionListener(this);
  tyhjenda.addActionListener(this);
  lae.addActionListener(this);
  salvesta.addActionListener(this);
  mangi.addActionListener(this);
  mangiLoik.addActionListener(this);
}

/**
* Sündmuse valik vastavalt nupuvajutusele.
*/
public void actionPerformed(ActionEvent e){
  if(e.getSource()==lae){laeFail(); }
  if(e.getSource()==salvesta){salvestaFail(); }
  if(e.getSource()==tyhjenda){tyhjendaLoik(); }
  if(e.getSource()==loika){loikaLoik(); }
  if(e.getSource()==kopeeri){kopeeriLoik(); }
  if(e.getSource()==kleebi){kleebiLoik(); }
  if(e.getSource()==kirjutaYle){kirjutaLoikYle(); }
  if(e.getSource()==mangi){mangi(); }
  if(e.getSource()==mangiLoik){new Thread(this).start(); }
}

/**
*  Faili mällu lugemine. Andmete pikkus aimatakse kaadri suuruse ning
*  kaadrite arvu järgi. Formaat jäetakse meelde.
*/
public void laeFail(){
 try{
  AudioInputStream sisse=AudioSystem.getAudioInputStream(new File(tfLae.getText()));
  formaat=sisse.getFormat();
  andmed=new byte[formaat.getFrameSize()*(int)sisse.getFrameLength()];
  sisse.read(andmed);
  ala.seaAndmed(andmed, formaat);
  System.out.println(formaat);
 }catch(Exception viga){
   viga.printStackTrace();
 }
}

/**
* Helifaili talletamine kettale. Vorming ja parameetrid
* jäävad endiseks, kasutaja sai sisu lisada, muuta või lühendada.
*/
```

```
public void salvestaFail(){
 try{
    AudioInputStream ais=new AudioInputStream(
     new ByteArrayInputStream(andmed), formaat, andmed.length
   );
  AudioSystem.write(ais, AudioFileFormat.Type.WAVE, new File(tfSalvesta.getText()));
 }catch(Exception viga){
   viga.printStackTrace();
 }
}


/**
* Märgitud lõigu kopeerimine mällu ning eemaldamine mängitavast jadast.
* Kuna märgistusalas hoolitsetakse, et märgistus algab ja
* lõpeb alati kaadri esimese baidiga, siis õnnestub siin
* vastavate algKaadri ja lõppKaadri vaheliste baitide töötlemine
* sõltumata kanalite ning kvandis asuvate baitide arvust.
*/
public void loikaLoik(){
  byte[] uus=new byte[andmed.length-(ala.loppKaader()-ala.algKaader())];
  puhver=new byte[ala.loppKaader()-ala.algKaader()];


  System.arraycopy(andmed, ala.algKaader(), puhver, 0, (ala.loppKaader()-ala.algKaader()));
  System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
  System.arraycopy(andmed, ala.loppKaader(), uus,
                   ala.algKaader(), andmed.length-ala.loppKaader());
  andmed=uus;
  uuendaAla();
}


/**
* Lõigu tühjendus, kõikide seal asuvate baitide väärtuseks
* antakse 0.
*/
public void tyhjendaLoik(){
  for(int i=ala.algKaader(); i<=ala.loppKaader(); i++){
    andmed[i]=(byte)0;
  }
  uuendaAla();
}


/**
* Märgitud lõigu andmed kopeeritakse puhvrisse.
* Mängitava lõigu sisu ei muutu.
*/
public void kopeeriLoik(){
  puhver=new byte[ala.loppKaader()-ala.algKaader()];
  for(int i=0; i<puhver.length; i++){
    puhver[i]=andmed[ala.algKaader()+i];
  }
}

/**
* Puhvrist lisatakse kursori kohale vahele lõik. Andmete
* hoidmiseks luuakse uus massiiv ning sinna paigutatakse
* tulemus sobival kujul.
*/
public void kleebiLoik(){
  if(puhver==null)return;
  byte[] uus=new byte[
     andmed.length+puhver.length-(ala.loppKaader()-ala.algKaader())];
  System.arraycopy(andmed, 0, uus, 0, ala.algKaader());
  System.arraycopy(puhver, 0, uus, ala.algKaader(), puhver.length);
  System.arraycopy(andmed, ala.loppKaader(), uus,
            ala.algKaader()+puhver.length, andmed.length-ala.loppKaader());
  andmed=uus;
  uuendaAla();
}

/**
* Joonistusala uuendamine pärast andmete muutust.
* Ala hoolitseb seeläbi juba ise enese pildi uuendamise eest.
*/
void uuendaAla(){
  ala.seaAndmed(andmed, formaat);
}


/**
* Puhvris olev lõik kirjutatakse kursorist alates
```

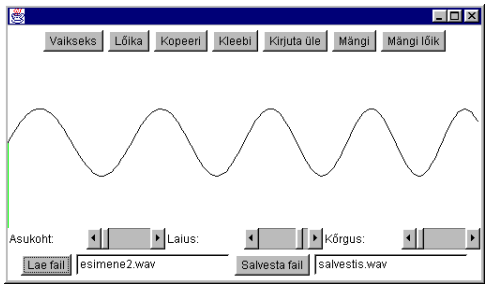
```
  * algse heli andmete asemele.
  */
  public void kirjutaLoikYle(){
    if(puhver==null)return;
    for(int i=0; i<puhver.length; i++){
      andmed[ala.algKaader()+i]=puhver[i];
    }
    uuendaAla();
  }


  /**
  * Mängitakse mälus olevad andmed kogu pikkuses.
  */
  public void mangi(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     kanal.write(andmed, 0, andmed.length);
     kanal.drain();
     kanal.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }


  /**
  * Eraldi lõimes lükatakse mängima märgitud lõik.
  * Kui märkeruut sees, siis jäädakse kordama, muul
  * juhul kõlab lõigu sisu ühe korra.
  */
  public void run(){
    try{  //Kontrollib, et opsüsteemilt on võimalik kanal küsida.
     kanal=(SourceDataLine)AudioSystem.getLine(
      new DataLine.Info(SourceDataLine.class, formaat, AudioSystem.NOT_SPECIFIED)
     );
     kanal.open();
     kanal.start();
     do{
       kanal.write(andmed, ala.algKaader(), ala.loppKaader()-ala.algKaader());
     }while(loiguKordus.getState());
     kanal.drain();
     kanal.close();
    }catch(Exception ex){
      ex.printStackTrace();
    }
  }

  /**
  * Redaktori käivitus käsurealt.
  */
  public static void main(String[] argumendid){
    Frame f=new Frame();
    f.add(new Digiheliredaktor3());
    f.setSize(400, 300);
    f.setVisible(true);
  }
}
```



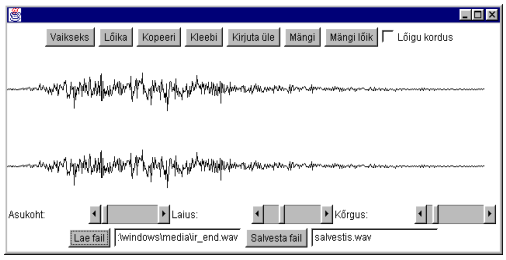
Enne faili avamist



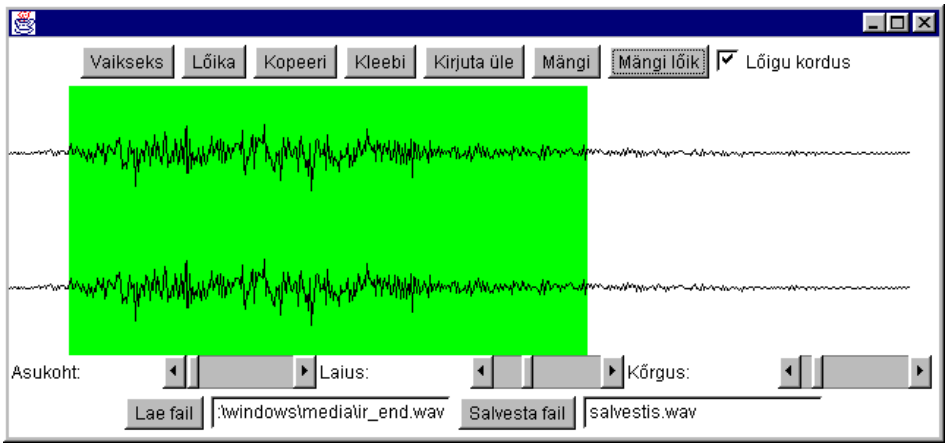
Kõrgenev ehk tiheneva sagedusega heli



Venitatud laiusega lõik



Heli kahel kanalil



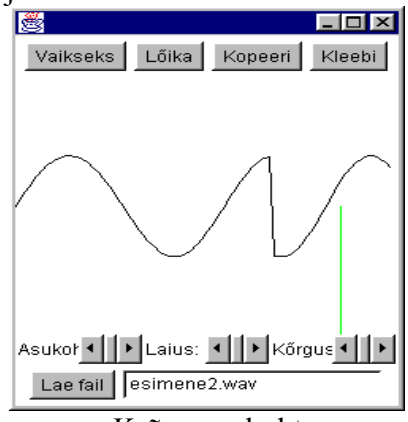
Korduse mängimine

```
D:\arhiiv\konspektid\gm\naited>java Digiheliredaktor3
PCM_SIGNED, 22050.0 Hz, 16 bit, stereo, little-endian, audio data
```

Heli väljatrükitud formaat.

Kommenteeritud rakendus pole veel kaugeltki täiuslik, kuid ega eesmärgiks polegi ühe järjekordse helitöötlusprogrammi loomine, milliseid juba nii vaba kui kommertstarkvara hulgas mitmeid kätte saada on. Pigem tuleb ise kirjutada selliseid lõike, mis olemasolevat lindistust soovitud suunas analüüsivad või siis kasutaja toimetuste saateks sobilikke helisid kokku panevad.

Nagu ehk katsetuste juures kuulda võisite, kippusid lõikamise ja kleepimise juures krõpsud sisse tulema. Põhjuseks, et nõnda lõikamisel ei tea baite kopeeriv koodilõik helikõvera kujust midagi ning sujuva joone sisse tekivad järsud murdekohad.



Krõpsuga koht

Üheks krõpsude vähendamise võimaluseks on kokkuliidetavad pooled veidi üksteise peale paigutada ning eelnev heli mõne võnke jooksul summutada ning järgnev heli sama aja sees vaikusest paari võnkega üles võimendada.

Kui kokku liidetavad helid on ligikaudugi sama amplituudi ja võnkesagedusega, siis võib õnnestuda võnkumised kokku ühendada samas faasis ning nii silmade kui kõrvade jaoks tundub, et helid oleksid nagu algusest peale kokku kuulunud.

Kokkuvõte

Java muusikavõimalused on tasapisi laienenud ning alates versioonist 1.3 õnnestub nii MIDI vahenditega orkestrilugusid kokku panna kui ise digitaalhelisid luua ja töödelda. Nagu mujal, nii ka siin pole programmeerimiskeel imevahend, selle kaudu lihtsalt õnnestub arvuti vastavatele seadmetele ligi pääseda ning soovitud helisid salvestada või kuuldavale tuua. Muusikalised ja matemaatilised tagamaad tuleb ikka enesel läbi mõelda ning pärast esmavahenditega tutvumist tulebki rakenduse loomisel enam sellele pühenduda. Lugudele saate koostamisel tuleb mõelda harmoonia ning taustalõikude peale neid sobivasse helistikku paigutades. Olenevalt muusikastiilist saab sageli kolme põhiduuriga enamikuga soovitust toime ning alles ilustuste ja kaunistuste juures tuleb muud oskused ja vahendid meelde tuletada ning arvutile selgeks teha. Olemasolevat noodifaili muutes tuleb arvestada, millistel radadel asuvate häältega me mida ette võtta soovime.

Digitaalhelis tuleb helilaine kuju ise välja arvutada. Nõnda kulub küll juba lihtsate häälte tekitamiseks paras kogus matemaatikat ning vähegi ilusamate helide loomiseks tuleb hulga arvutada ning tarkadest raamatutestki tarkust juurde ammutada, kuid põhimõtteliselt on võimalik luua või olemasolevatest kokku panna pea iga heli, mis vähegi ettekujutatav võiks olla.

Ülesandeid

Heli kõrgus

- Mängi signaali sagedusega 440 Hz.
- Pane signaali kõrgus sõltuma hiire asukohast ekraanil.
- Hoolitse, et kõrguse muutumine oleks sujuv.
- Korraga kõlab kaks heli. Ühe kõrgus sõltub hiire x- ning teine y-koordinaadist.
- Salvesta loodud heli faili.

Helifaili muundamine

- Salvesta helifail kaks korda üksteise järele.
- Salvesta helifail algsest poole vaiksemalt.
- Salvesta helifail iseenesega kajanihkega.