3. Javan ääniohjelmointi

1. `java.applet`
2. `javax.sound.sampled`

3.1. `java.applet`

- tarjoaa yleisen, yksinkertaisen ja laitteistoriippumattoman äänipalvelun
- käytettävissä:
  - appleteissa (JDK 1.0)
  - sovelluksissa (JDK 1.2)
- tukee ääniformaatteja AIFF, AU, WAV, MIDI, RMF

`Applet`-luokan äänimetodeja

- `void play(URL url)`
- `void play(URL url, String name)`
- `AudioClip getAudioClip(URL url)`
- `AudioClip getAudioClip(URL url, String name)`
- `static AudioClip newAudioClip(URL url)`

Muita esille tulevia `Applet`-metodeja

- `void init()`
- `void start()`
- `void stop()`
- `URL getCodeBase()`
- `void showStatus(String msg)`
- `String getParameter(String name)`

`AudioClip`-rajapinta

- metodit yksittäistoistoon, silmukointiin ja pysäyttämiseen

```
interface AudioClip {
    public void play();
    public void loop();
    public void stop();
}
```

`URL`-luokka

- sijaitsee pakkauksessa `java.net`
- `getCodeBase`-metodilla saadaan appletin perusosoite, johon lisätään äänitiedoston nimi
- osoite luodaan `URL`-luokan konstruktorilla
  `public URL(String spec) throws MalformedURLException`
- poikkeus on käsiteltävä

SimpleAudioApplet.java 1(3)

```
import java.applet.*;
import java.net.*;

public class SimpleAudioApplet
                  extends Applet {

  private AudioClip sound = null;
```

SimpleAudioApplet.java 2(3)

```
  public void init() {
    try {
      URL sf = new URL(getCodeBase()
                       + "sound.wav");
      sound = getAudioClip(sf);
    } catch (MalformedURLException e) {
      showStatus("Cannot load the "
                 + "audio file.");
    } // try
  } // init()
```

SimpleAudioApplet.java 3(3)

```
  public void start() {
    if (sound != null)
      sound.play();
  } // start()
} // class
```

SimpleAudioApplet.html

```
...
<applet
code="SimpleAudioApplet.class"
width=300 height=300>
Your browser doesn't
support applets.
</applet>
...
```

Appletin elinkaari

- applettiolio luodaan
- appletti alustetaan kutsumalla `init`-metodia
- appletin suoritus aloitetaan kutsumalla `start`-metodia
- jos appletista poistutaan (esim. vaihdetaan sivua, minimoidaan selain), kutsutaan `stop`-metodia
- kun applettiin palataan takaisin, kutsutaan `start`-metodia
- appettioliota poistettaessa kutsutaan `finalize`-metodia

BackgroundMusicApplet.java 1(2)

```
public void init() {
  String name = getParameter("BG_MUSIC");
  try {
    String base = getCodeBase() + "snd/";
    sound = getAudioClip(new
                         URL(base + name));
  } catch (MalformedURLException e) {
    showStatus("Cannot load audio file "
               + name + ".");
  } // try
} // init()
```

BackgroundMusicApplet.java 2(2)

```
public void start() {
  if (sound != null)
    sound.loop();
} // start()

public void stop() {
  if (sound != null)
    sound.stop();
} // stop()
```

BackgroundMusicApplet.html

```
...
<applet
code="BackgroundMusicApplet.class"
width=300 height=300>
<param name="BG_MUSIC"
value="muzak.wav">
Your browser doesn't
support applets.
</applet>
...
```

Säikeistä

- jos äänitiedostot ovat pitkiä, niiden lataaminen kannattaa siirtää taustalle omaan säikeeseen

→ appletti voi aloittaa suorituksensa heti, eikä sen tarvitse odottaa äänitiedostojen latautumista

- säieolio periytyy joko `Thread`-luokasta tai se toteuttaa `Runnable`-rajapinnan

AudioApplet.java 1(6)

```
base = getCodeBase() + "sounds/";
for (int i = 0; i < sounds.length; i++) {
  String fileName = getParameter("SOUND"
                                  + i);
  if (fileName != null) {
    AudioLoader audioLoader =
          new AudioLoader(fileName, i);
    audioLoader.start();
  } // if
```

AudioApplet.java 2(6)

```
for (int i = 0; i < sounds.length; i++) {
  Button button =
              new Button("Sound " + i);
  button.addActionListener(
                    new ButtonPress(i));
  add(button);
} // for
```

AudioApplet.java 3(6)

```
public void stop() {
  for (int i = 0; i < sounds.length; i++)
    if (sounds[i] != null)
      sounds[i].stop();
} // stop()
```

AudioApplet.java 4(6)

```
private class AudioLoader
                    extends Thread {
  private String fileName;
  private int finger;

  public AudioLoader(String n, int f) {
    setDaemon(true);
    fileName = n;
    finger = f;
  } // constructor
```

AudioApplet.java 5(6)

```
  public void run() {
    try {
      sounds[finger] = getAudioClip(
             new URL(base + fileName));
    } catch (MalformedURLException e) {
      showStatus("Cannot load file "
                 + fileName + ".");
    } // try
  } // run()
} // class
```

AudioApplet.java 6(6)

```
private class ButtonPress
                  implements ActionListener {
  private int soundNumber;

  public ButtonPress(int s) {
    soundNumber = s;
  } // constructor

  public void actionPerformed(ActionEvent e) {
   AudioClip sound = sounds[soundNumber];
   if (sound != null) sound.play();
  } // actionPerformed()
 } // class
```

AudioApplet.html

```
...
<applet code="AudioApplet.class"
width=500 height=100>
<param name="SOUND0" value="music0.wav">
<param name="SOUND1" value="music1.wav">
<param name="SOUND2" value="music2.wav">
...
<param name="SOUND5" value="music5.wav">
Your browser doesn't support applets.
</applet>
...
```

Jar-tiedostot

- useista luokkatiedostoista koostuvat appletit on syytä koota yhdeksi jar-tiedostoksi

→ vältetään useiden pienten tiedostojen lataaminen

- esim.

```
jar cvf AudioApplet.jar *.class
```

JarredAudioApplet.html

```
...
<applet code="AudioApplet.class"
archive="AudioApplet.jar"
width=500 height=100>
<param name="SOUND0" value="music0.wav">
...
<param name="SOUND5" value="music5.wav">
Your browser doesn't support applets.
</applet>
...
```

`Applet`-metodien käyttö sovelluksissa

- staattinen metodi `newAudioClip` palauttaa `AudioClip`-olion
  - huom. parametriksi annetaan `URL`-olio eikä esim. tiedostokahva
  - vinkki: sovelluksen oletushakemiston polun saa metodikutsulla `System.getProperty("user.dir")`
- saatua `AudioClip`-oliota voidaan käyttää normaalisti

`AudioApplication.java` 1(3)

```
public class AudioApplication {
  public static void
                    main(String[] args) {
  AudioClip[] sounds =
              new AudioClip[args.length];
  String base = "file:" +
    System.getProperty("user.dir") + "/";
```

`AudioApplication.java` 2(3)

```
for (int i = 0; i < args.length; i++) {
  try {
    sounds[i] = Applet.newAudioClip(
                new URL(base + args[i]));
  } catch (MalformedURLException e) {
    throw new RuntimeException(
              "Cannot load audio file "
              + args[i] + ".");
  } // try
} // for
```

`AudioApplication.java` 3(3)

```
    for (int i = 0;
         i < sounds.length; i++) {
      sounds[i].loop();
    } // for
  } // main()
} // class
```

3.1. `javax.sound.sampled`

- mukana JDK-versiosta 1.3 alkaen
- tarjoaa matalan tason liittymän alustan äänilaitteistoon (myös havainnointi)
- pyrkii silti olemaan alustariippumaton ja yleistettävissä oleva rakennelma
- mahdollistaa äänisignaalin
  - vastaanottamisen (esim. äänitys)
  - käsittelyn (esim. vahvistus tai kaiunta)
  - toistamisen
- yhä kehitysvaiheessa? ←outoja bugeja ja puutteita

Pakkaukset

- `javax.sound.sampled`
  - rajapintoja ja luokkia samplatun äänisignaalin tallennukseen, muokkaamiseen ja toistoon
- `javax.sound.midi`
  - rajapintoja ja luokkia MIDI-käyttöön
- `javax.sound.sampled.spi`
- `javax.sound.midi.spi`
  - ulkopuolisille palveluntarjoajille (*service providers*) tarkoitettuja apuluokkia

Piirteitä

- pähkinänkuoressa: äänidataa sisältävien tavujen lukua, kirjoitusta ja operointia
- liittymät syöttö- (esim. mikrofoni tai tiedosto) ja tuloslaitteisiin (esim. kaiutin tai tiedosto)
- äänidatan puskurointi (esim. reaaliaikainen äänivirta)
- äänisignaaleiden yhdistäminen
- käyttäjän komennot: aloita, pysäytä, jatka, lopeta

Äänidatan käsittelytavat

- puskuroitu (*buffered*)
  - virta (*streaming*): reaaliaikaisen äänidatan käsittelyä
  - operoitava (esim. äänitettävä tai käsiteltävä) tavuja likimain samassa tahdissa kuin missä niitä lähetetään
- puskuroimaton (*unbuffered*)
  - äänidata sijaitsee (kokonaisuudessaan) muistissa
  - monipuolisempi toisto: silmukointi, aloituspaikan valinta

Äänidatan formaatit 1(2)

- dataformaatti
  - kertoo kuinka sarja tavuja eli ”raaka” samplattu äänidata pitää tulkita
  - `AudioFormat`-luokka
- tiedostoformaatti
  - määrittelee äänitiedoston rakenteen
  - `AudioFileFormat`-luokka

Äänidatan formaatit 2(2)

- vaikka tarjolla on metodeja
  - erilaisten ääniformaattien muuttamiseen
  - yleisten tiedostoformaattien lukemiseen ja tallentamiseen
- kyse ei silti ole kaikenkattavasta äänityökalusta
  - palveluntarjoajilta tukea ja täydennystä valikoimaan

`AudioFormat`-luokka

- koodaustekniikka (esim. PCM, a-law tai μ-law)
- kanavien määrä (1 = mono, 2 = stereo jne.)
- samplaustaajuus
- kvantisointitaso (so. käytettyjen bittien määrä)
- kehystaajuus (*frame rate*)
  - kehys (*frame*) = kaikki tiettyyn hetkeen kuuluva data; (esim. kanavien nykyiset näytearvot)
- kehyksen koko tavuina
- tavujärjestys: *big-endian* tai *little-endian*

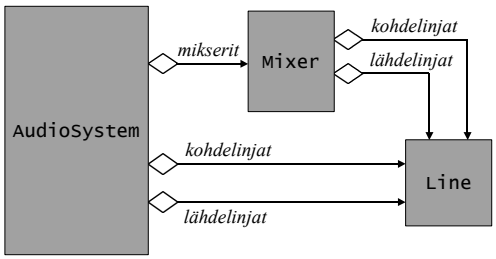
`AudioFileFormat`-luokka

- tiedostotyyppi (esim. WAVE, AIFF)
- tiedoston pituus tavuina
- äänidatan pituus kehyksinä
- `AudioFormat`-olio, joka määrittelee tiedoston sisältämän äänidatan muodon

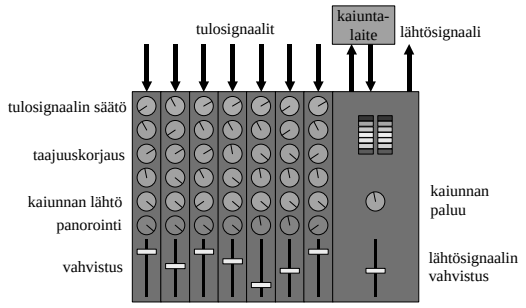
Java Sound -perusosat

- äänijärjestelmä: `AudioSystem`
- mikseri: `Mixer`
- linja: `Line`
- portti: `Port`

Perusrakenne



Analogia: miksauspöytä



Äänijärjestelmä (*audio system*)

- kokoaa yhteen kaikki laitteiston ja ohjelmiston tarjoamat äänipalvelut:
  - mikserit
  - linjat
  - portit
  - äänivirrat
  - tiedostoformaatit
  - ääniformaatit

`AudioSystem`-luokan metodeita

- `static Mixer.Info[] getMixerInfo()`
- `static Mixer getMixer(Mixer.Info info)`
- `static Line getLine(Line.Info info)`
- `static AudioFileFormat getAudioFileFormat(File file)`
- `static AudioInputStream getAudioInputStream(File file)`

Mikseri (*mixer*)

- abstrahoi äänilaitetta (*audio device*), esim. äänikortti
- saa syötteenä yhden tai useamman äänivirran ja antaa tulokseksi yhden tai useamman äänivirran
  - esim. miksaa kaksi ääntä (syöte) yhdeksi ääneksi (tulos)
  - voi tukea äänten synkronointia
- voi edustaa fyysistä laittetta tai sen ominaisuutta
- voi edustaa kokonaan ohjelmistolla toteutettua ominaisuutta

`Mixer`-rajapinnan metodeita

- `Line.Info[] getSourceLines()`
- `Line.Info[] getTargerLines()`
- `Line getLine(Line.Info info)`
- `void synchronize(Line[] lines, boolean maintainSync)`

Linja (*line*)

- johtaa joko sisään äänijärjestelmään (tai mikseriin) tai siitä ulos
- voi sisältää rinnakkaisia kanavia (mono, stereo)
- tila: avoin tai suljettu
- tapahtumat
  - viestien välitys rekisteröityneille kuuntelijoille
- voi sisältää säätöjä, esim. vahvistus, panorointi, kaiunta, toistotaajuus, mykistys
- mikserit ja portit ovat linjoja ← periytyminen

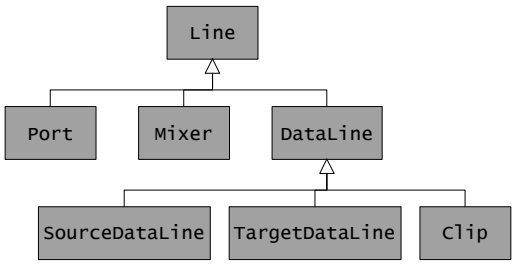
`Line`-rajapinnan metodeja

- `void open()`
- `void close()`
- `void addLineListener(LineListener listener)`
- `Control[] getControls()`
- `Control getControl(Control.Type control)`

Portti (*port*)

- abstrahoi laitteistotason liittymiä äänijärjestelmään, esim. mikrofoni tai kaiutin
- huom. `Port`-rajapinta on kyllä määritelty mutta sillä ei ole toteutuksia?! (bugi #4384401)

Rajapintojen periytymishierarkia



Datalinja

- assosioi linjan tiettyyn ääniformaattiin
- puskuroitu: tavuvektori
- käynnistys ja pysäytys
- nykyinen sijainti (*media position*)
- taso (*level*): tämänhetkisen signaalin amplitudi
- tyhjennys (*flush*): poistaa prosessoimattoman datan puskuri
- valutus (*drain*): odottaa kunnes kaikki prosessoimaton data on saatu käsiteltyä
- aktiivisuus: onko linjassa signaalia

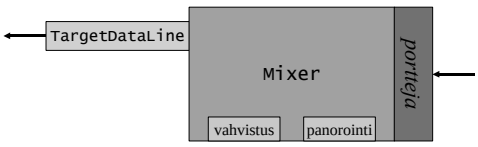
`DataLine`-rajapinnan metodeja

- `AudioFormat getFormat()`
- `int getBufferSize()`
- `void start()`
- `void stop()`
- `int getFramePosition()`
- `float getLevel()`
- `void flush()`
- `void drain()`
- `boolean isActive()`

Kohdedatalinja

- linja josta voidaan lukea dataa
- mikseri voi toimittaa linjaan dataa esim. mikrofonista → äänitys
- huom. linja on kohde (*target*) mikserin näkökulmasta

Esimerkki: äänitys



`TargetDataLine`-rajapinnan metodeja

- `void open(AudioFormat format)`
- `int read(byte[] b, int off, int len)`

Lähdedatalinja

- linja johon voidaan kirjoittaa dataa
- mikseri voi toimittaa kirjoitetun datan esim. kaiuttimiin → toisto
- huom. linja on lähde (*source*) mikserin näkökulmasta

`SourceDataLine`-rajapinnan metodeja

- `void open(AudioFormat format)`
- `int write(byte[] b, int off, int len)`

Pätkä (*clip*)

- linja johon voidaan ladata dataa ennen toistoa
- äänidatan pituus tunnetaan ennen toistoa → aloituspaikka voidaan valita vapaasti
- toistoa voidaan silmukoida (muttei annetulle välille ← bugi #4386052)

Esimerkki: toisto

`Clip`-rajapinnan metodeja

- `void open(AudioInputStream stream)`
- `int getFrameLength()`
- `long getMicrosecondLength()`
- `void setFramePosition(int frames)`
- `void setMicrosecondPosition(long milliseconds)`
- `void loop(int count)`
- `void setLoopPoints(int start, int end)`

Mikserin haku

- `Mixer.Info`-olio sisältää mikserin kuvauksen
- pyydetään äänijärjestelmältä lista mikserikuvauksia `getMixerInfo`-metodilla
- valitaan listasta sopiva ja pyydetään sitä `getMixer`-metodilla

Linjan haku

- `Line.Info`-olio sisältää linjan kuvauksen
- pyydetään äänijärjestelmältä tai mikserilta annettua kuvausta vastaava linja `getLine`-metodilla
- käsiteltävä poikkeus `LineUnavailableException`
- porttia tai datalinjaa pyydetään vastaavalla tavalla `Port.Info`- ja `DataLine.Info`-olioilla

`AudioSystemTest.java` 1(2)

```
import javax.sound.sampled.*;

public class AudioSystemTest {

  public static void main(String[] args) {
    Mixer.Info[] mi = AudioSystem.getMixerInfo();

    for (int i = 0; i < mi.length; i++) {
      System.out.println(mi[i]);
      Mixer m = AudioSystem.getMixer(mi[i]);
```

AudioSystemTest.java 2(2)

```
        Line.Info[] sli = m.getSourceLineInfo();
        for (int j = 0; j < sli.length; j++)
          System.out.println("source: " + sli[j]);

        Line.Info[] tli = m.getTargetLineInfo();
        for (int j = 0; j < tli.length; j++)
          System.out.println("target: " + tli[j]);

        System.out.println();
    }  }  }
```

Äänivirran haku

- pyydetään äänijärjestelmältä `AudioInputStream`-olio kutsumalla `getAudioInputStream`-metodia
- parametri voi olla `File`-, `URL`- tai `InputStream`-olio
- käsiteltävä poikkeukset `UnsupportedAudioFileException` ja `IOException`

SimplePlayer.java 1(5)

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

public class SimplePlayer {

  public static void main(String[] args) {
    if (args.length == 0) System.exit(0);
    File file = new File(args[0]);
    int loopCount = 0;
```

SimplePlayer.java 2(5)

```
if (args.length > 1 && args[1].equals("loop")) {
  if (args.length > 2) {
    try {
     loopCount = Integer.parseInt(args[2]) - 1;
    } catch (NumberFormatException e) {
      System.err.println("Ei kokonaisluku: "
                         + args[2]);
      System.exit(1);
    }
  } else loopCount = Clip.LOOP_CONTINUOUSLY;
} // if
```

SimplePlayer.java 3(5)

```
try {
  AudioInputStream source =
           AudioSystem.getAudioInputStream(file);
  AudioFormat format = source.getFormat();

  DataLine.Info info =
           new DataLine.Info(Clip.class, format);

  if (!AudioSystem.isLineSupported(info)) {
    System.err.println("Ei sopivaa linjaa.");
    System.exit(1);
  }
```

SimplePlayer.java 4(5)

```
try {
  Clip clip = (Clip)AudioSystem.getLine(info);
  clip.open(source);

  if (loopCount == 0) clip.start();
  else clip.loop(loopCount);
} catch (LineUnavailableException e) {
  System.err.println("Linjaa ei voi käyttää.");
}
```

SimplePlayer.java 5(5)

```
      } catch (IOException e) {
        System.err.println(
                    "Virhe tiedoston luvussa.");
      } catch (UnsupportedAudioFileException e) {
        System.err.println(
                "Tuntematon tiedostoformaatti.");
      } // try
    } // main()
  } // class
```

Linjan kuuntelija

- kuuntelija toteuttaa `LineListener`-rajapinnan
- kuuntelija liitetään linjaan `addLineListener`-metodilla
- rajapinnan `update`-metodi saa parametrina `LineEvent`-olion, jolta voi tiedustella tapahtuman tyyppiä
- `LineEvent.Type`-tapahtumatyyppejä: OPEN, CLOSE, START, STOP

RandomSequencePlayer.java 1(5)

```
public class RandomSequencePlayer {
  private Random random = new Random();
  private Clip[] clips;

  public RandomSequencePlayer(File[] files) {
    try {
      AudioInputStream[] sources =
             new AudioInputStream[files.length];
      AudioFormat[] formats =
             new AudioFormat[files.length];
      DataLine.Info[] infos =
             new DataLine.Info[files.length];
```

RandomSequencePlayer.java 2(5)

```
for (int i = 0; i < sources.length; i++) {
  sources[i] =
    AudioSystem.getAudioInputStream(files[i]);
  formats[i] = sources[i].getFormat();
  infos[i] = new DataLine.Info(Clip.class,
                               formats[i]);
  if (!AudioSystem.isLineSupported(infos[i])) {
   System.err.println("Ei sopivaa linjaa.");
   System.exit(1);
  } // if
} // for
```

RandomSequencePlayer.java 3(5)

```
try {
  clips = new Clip[sources.length];
  for (int i = 0; i < clips.length; i++) {
   clips[i] =
            (Clip)AudioSystem.getLine(infos[i]);
   clips[i].addLineListener(new Changer());
   clips[i].open(sources[i]);
   }
} catch (LineUnavailableException e) {
  System.err.println("Linjaa ei voi käyttää.");
}
```

RandomSequencePlayer.java 4(5)

```
public void startRandomClip() {
  int finger = random.nextInt(clips.length);
  clips[finger].setFramePosition(0);
  clips[finger].start();
}

public static void main(String[] args) {
  File[] files = new File[args.length];
  for (int i = 0; i < files.length; i++)
    files[i] = new File(args[i]);
  RandomSequencePlayer rsp =
                new RandomSequencePlayer(files);
  rsp.startRandomClip();
}
```

RandomSequencePlayer.java 5(5)

```
private class Changer implements LineListener {

  public void update(LineEvent e) {
    if (e.getType().equals(LineEvent.Type.START))
      System.out.println("New clip started.");
    if (e.getType().equals(LineEvent.Type.STOP))
      startRandomClip();
  }
}
```

Lähdedatalinjan käyttö

- varataan puskuriksi `byte`-taulukko
- puskurin koko
  - lyhyt: nopeampi vaste, katkosten riski
  - pitkä: hitaampi vaste, sietää katkoksia
- `write`-metodin kutsu aloittaa toiston (mm. lähettää kuuntelijalle aloitusviestin)
- `drain`-metodi odottaa, kunnes kaikki kirjoitettu data on toistettu
- `flush`-metodi poistaa kirjoitetun datan

SynthPlayer.java 1(5)

```
import java.io.*;
import java.util.*;
import javax.sound.sampled.*;

public class SynthPlayer {
  public static void main(String[] args) {
    float sampleFreq = 44100.0f;
    int bitsInQuantization = 8;
    int channels = 1;
    boolean signed = true;
    boolean bigEndian = true;
    AudioFormat format = new AudioFormat(
                sampleFreq, bitsInQuantization,
                channels, signed, bigEndian);
```

SynthPlayer.java 2(5)

```
DataLine.Info info = new DataLine.Info(
                  SourceDataLine.class, format);

if (AudioSystem.isLineSupported(info)) {
  try {
    SourceDataLine line =
       (SourceDataLine)AudioSystem.getLine(info);

    int bufferSize = 6 * (int)sampleFrequency;
    byte[] buffer = new byte[bufferSize];

    int twoSecMarker = 2 * (int)sampleFrequency;
    int fourSecMarker = 4 * (int)sampleFrequency;
```

SynthPlayer.java 3(5)

```
Random random = new Random();
for (int i = 0; i < twoSecMarker; i++)
  buffer[i] = (byte)random.nextInt();

int waveLength = (int)sampleFrequency / 440;
for (int i = twoSecMarker;
     i <= (fourSecMarker - waveLength);
     i += waveLength) {
  for (int j = i; j < i + waveLength / 2; j++)
    buffer[j] = Byte.MAX_VALUE;
  for (int j = i + waveLength / 2;
       j < i + waveLength; j++)
    buffer[j] = Byte.MIN_VALUE;
} // for
```

SynthPlayer.java 4(5)

```
for (int i = fourSecMarker;
     i <= (bufferSize - waveLength);
     i += waveLength) {
  for (int j = i; j < i + waveLength; j++)
    buffer[j] = (byte)(127.0 * Math.sin(
         (double)j / waveLength * 2 * Math.PI));
} // for
```

SynthPlayer.java 5(5)

```
// Avataan linja ja aloitetaan toisto.
line.open(format);
line.start();
// Kirjoitetaan puskuri linjalle.
line.write(buffer, 0, bufferSize);
// Odotetaan linjan tyhjentymistä
// ennen kuin lopetetaan.
line.drain();
line.stop();
line.close();
```

Kohdedatalinjan käyttö

- varataan puskuriksi `byte`-taulukko
- puskurin koon vaikutus kuten lähdedatalinjassa
- `read`-metodin kutsu lukee puskuriin dataa ja palauttaa sen määrän tavuina
- `drain`-metodi odottaa, kunnes mikserissä oleva data tulee luetuksi
- `flush`-metodi poistaa lukemista odottavan datan (muuten se jää odottamaan mikseriin)

Karaoke.java 1(2)

```
DataLine.Info infoTarget = new
     DataLine.Info(TargetDataLine.class, format);
DataLine.Info infoSource = new
     DataLine.Info(SourceDataLine.class, format);
if (AudioSystem.isLineSupported(infoTarget)
  & AudioSystem.isLineSupported(infoSource) {
  try {
    TargetDataLine lineTarget = (TargetDataLine)
                AudioSystem.getLine(infoTarget);
    SourceDataLine lineSource = (SourceDataLine)
                AudioSystem.getLine(infoSource);
int bufferSize =  (int)(bufferLength *
  format.getFrameSize() * format.getFrameRate());
byte[] buffer = new byte[bufferSize];
```

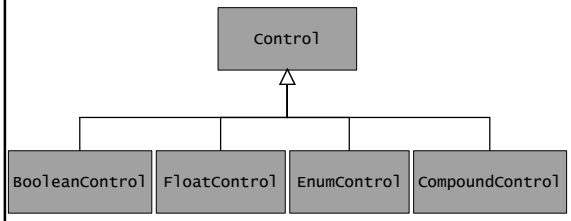
Karaoke.java 2(2)

```
lineTarget.open(format, bufferSize);
lineSource.open(format);
lineTarget.start();
lineSource.start();

while (true) {
  int dataSize =
          lineTarget.read(buffer, 0, bufferSize);

  // Tässä kohtaa puskurissa olevalle äänelle
  // voisi tehdä jotain jäynää...

  lineSource.write(buffer, 0, dataSize);
}
```

Linjan säätimet (*controls*)

- `Control`-luokasta periytyy neljä säädintyyppiä:
  - `BooleanControl`: katkaisin
    - esim. mykistys (*mute*)
  - `FloatControl`: säätökytkin
    - esim. vahvistus, panorointi
  - `EnumControl`: valintakytkin
    - esim. kaiunnan esivalinnat
  - `CompoundControl`: säädinkokoelma
    - esim. taajuuskorjain voi olla kokoelma `FloatControl`-tyyppisiä säätökytkimiä

Säädinten luokkahierarkia



Linjan säädinten haku

- `getControls`-metodi palauttaa taulukon linjan tarjoamista säätimistä
- `isControlSupported`-metodi palauttaa onko halutun tyyppistä säädintä tarjolla
- `getControl`-metodi palauttaa pyydetyn tyyppisen säätimen

Säädintyyppejä

- `BooleanControl.Type.MUTE`
- `EnumControl.Type.REVERB`
- `FloatControl.Type.MASTER_GAIN`
- `FloatControl.Type.PAN`
- `FloatControl.Type.SAMPLE_RATE`

`FloatControl`-luokan metodeja

- `float getValue()`
- `void setValue(float newValue)`
- `float getMaximum()`
- `float getMinimum()`
- `float getPrecision()`
- `String getMaxLabel()`
- `String getMidLabel()`
- `String getMinLabel()`
- `String getUnits()`

`ControlPlayer.java` 1(3)

```
boolean gain = false, pan = false,
        rate = false, mute = false;
if (args.length >= 1) {
  gain = args[1].equals("gain");
  pan = args[1].equals("pan");
  rate = args[1].equals("rate");
  mute = args[1].equals("mute");
}
float parameter = 0.0f;
if (args.length == 3) {
  try {
    parameter = Float.parseFloat(args[2]);
  } catch (NumberFormatException e) { ... }
}
```

`ControlPlayer.java` 2(3)

```
clip.open(source);
clip.start();

if (gain && clip.isControlSupported(
       FloatControl.Type.MASTER_GAIN)) {
  FloatControl gainCtrl =
     (FloatControl)clip.getControl(
         FloatControl.Type.MASTER_GAIN);
  gainCtrl.setValue(parameter);
}
```

`ControlPlayer.java` 3(3)

```
if (mute && clip.isControlSupported(
                   BooleanControl.Type.MUTE)) {
  BooleanControl muteCtrl =
       (BooleanControl)clip.getControl(
                    BooleanControl.Type.MUTE);
  muteCtrl.setValue(true);
} // if

Control[] ctrl = clip.getControls();
for (int i = 0; i < ctrl.length; i++)
  System.out.println(ctrl[i]);
```

Ääniominaisuuksien käyttöoikeudet

- määritelty `AudioPermission`-luokassa:
  - toisto
  - äänitys
- appletti: saa toistaa muttei äänittää
- sovellus: saa toistaa ja äänittää
- ohjelmien oikeuksia voidaan muuttaa Policy Tool -ohjelmalla