`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[] getTargetLines()`
- `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

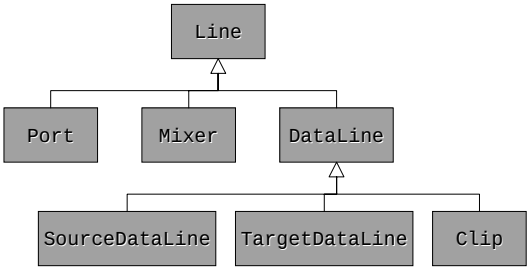
`Line`-rajapinnan metodeja

- `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

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

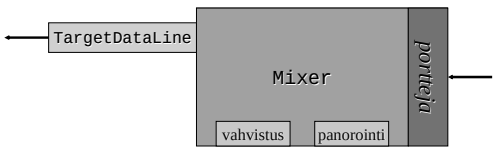
`DataLine`-rajapinnan metodeja

- `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

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();
```