情報メディア基盤ユニット用資料（2013 年 7 月 9 日分）

Processing 言語による情報メディア入門

音情報の取り扱い

神奈川工科大学情報メディア学科　佐藤尚

Special Thanks: 黒川先生

はじめに

今までの授業では、Processing を用いて図形を描いたり、画像の表示を行ってきました。Processing では、音を取り扱うことができます。そのためには、外部ライブラリを利用する必要があります。外部ライブラリには、Processing 本体に同梱されているものと、ユーザがインストールする必要があるものがあります。ここでは、Processing に同梱されている Minim（ミニム）を使って、音情報を取り扱います。他にも､Processing で音を扱うライブラリがあります。また、MAX/MSP などと組み合わせて使うことも出来るようです。

Minim は、非常に高度な機能を持っています。今回の授業ではすべてを扱うことは出来ません。興味のある人は、公式のサイトを覗いてみてください。公式のサイトの URL は、
http://code.compartmental.net/tools/minim
です。

また、今回紹介する Minim の機能や説明の際に何気なく使っているいくつかの言葉の意味を理解するためには、オブジェクト指向と呼ばれる考え方を知っている必要があります。

オブジェクト指向の話は、は次回にやる予定です。

Minim は次のような機能を持っています。

- 音声データファイルの再生、wav,aiff,au,snd,mp3 などの形式のファイルに対応しています。
- MP3 ファイルの ID3 タグデータの取得。トラック名､アーティスト名などの情報を取り出すことが出来ます。
- マイクなどからの音声入力の取得
- 基本波形、ノイズの発生
- ローパスフィルタなどの適応
- FFT

などです。

以下の説明で何気なく使っているいくつかの言葉の意味を理解するためには、オブジェクト指向と呼ばれる考え方を知っている必要があります。今のところは、メソッドという言葉は関数のことだと思って下さい。また、クラスという言葉も出てきますが、これはデータ型のことだと思って下さい。

Processing で音を鳴らす

Processing の外部ライブラリである Minim を利用して音声ファイルを再生するためには、次の 3 つの方法があります。

- AudioPlayer：ファイルサイズの大きい mp3 音声ファイルをストリーミング再生します。
- AudioSnippet: ファイルサイズの大きくない音声ファイルの再生等を行います。

- AudioSample: 非常に短い音声ファイルの再生を行います。

はじめに

まずは準備です。このライブラリを使うには、メニューから [Sketch] > [Import Library] > [minim] を選びます。スケッチコードの先頭から

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
```

が挿入されれば準備完了です。この 4 行を自分で入力してもかまいません。

最初は、AudioPlayer を利用した、ストリーミング再生の方法を紹介します。ストリーミング再生のため大きなファイルサイズの音声ファイルを取り扱うことが出来ます。ただし、再生開始が少し遅れることがあります。

音源ファイルを準備する

演奏する音楽ファイルを準備してください。ファイル形式は wav や mp3 など様々な形式が扱えます。ファイルは画像のときと同じようにまずは作成しているプログラムの pde ファイルと同じ場所 (Show Sketch Folder で表示される) に保存してください。 画像ファイルと同じ方法で、音声ファイルも Processing のスケッチ内に取り込むことが出来ます。

一番単純な音声ファイル生成のためのプログラムがサンプル 11-1 です。このサンプルはマウスをクリックすると音声ファイルの再生を行うものです。

サンプルプログラム 11-1

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer player;

void setup(){
  size(100,100);
  minim = new Minim(this);// Mimin オブジェクトの生成
  player = minim.loadFile("schoolsong.mp3");
}
void draw(){
  // やりたいことを書く
}
```

```
void mouseClicked(){
  player.play();
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

このサンプルは次のような手順で音声ファイルの再生を行っています。

1.Minim オブジェクトの生成

Minim ライブラリに含まれている様々な機能（メソッド）を利用するためには、まず Minim クラスのインスタンス（Minim オブジェクト）を生成します。Minim オブジェクトはコード全体で利用するのでグローバル変数として宣言します。つまり、プログラムの先頭に以下の行を追加します。

```
Minim minim;
```

2. 音源ファイルの読み込み

minim オブジェクトの準備ができたら、メンバメソッドである loadFile を用いて音源ファイルを読み込みます。loadFile の引数は音源ファイルのファイル名です。loadFile メソッド（関数）は AudioPlayer 型のデータが戻り値となっています。そこで、その戻り値を AudioPlayer 型の変数に保存しておきます。画像ファイルを読み込んだ際に、PImage 型の変数に保存したのと似ています。この変数はコード全体で利用するのでグローバル変数として宣言します。読み込みは準備的なことですので setup 中で書いています。これで準備は完了です。

正確には、AudioPlayer 型は「AudioPlayer クラスのインスタンス（AudioPlayer オブジェクト）」です。

3. 再生

読み込んだ音声ファイルを再生するためには、loadFile 関数の結果を保存した変数（このサンプルでは player）に対して、メンバメソッド play を呼び出します。つまり、

```
player.play();
```

とすれば演奏が開始されます。戻り値を代入した変数が player でない場合には、その変数に置き換えて下さい。例えば、song 変数に代入した場合には、

```
song.play();
```

となります。サンプル 11-1 では、mouseClicked 関数の中に再生開始の命令 player.play() が入っていますので、マウスをクリックすると再生が開始されます。

4. 後始末

サンプル 11-1 の最後の方に注目して下さい。音声ファイル再生のようにプログラム本体とは別の処理が続くような処理を行っている時には、プログラム実行終了時に明示的に後始末処理を書くことが必要になる場合があります。今回の Minim ライブラリも明示的に終了処理を書く必要があります。stop 関数はプログラムの実行終了時（Stop ボタンを押す、ウインドウの閉じるボタンを押すなど）に呼び出される関数です。Minim クラス、AudioPlayer クラスのインスタンスを生成し利用した場合は、スケッチが終了するときに必ず後始末として、AudioPlayer クラスのメンバメソッド close、Minim クラスのメンバメソッド stop、および、super, stop を呼び出す必要があります。

この処理を行わないと、再度の実行の際に音声ファイルの再生などが正常に行われないなどの不都合が起きる場合があります。

繰り返し再生

サンプル 11-1 では、一度再生が終了してしまうと、再びマウスボタンをクリックしても、再生が行われません。再び再生が行われるようにするためには、どのようにしたら良いでしょうか？一番簡単な解決方法は、繰り返し再生させることです。音声ファイルの繰り返し再生を行うためには、play メソッドの代わりに loop メソッドを使用することです。これを行ったものがサンプル 11-2 です。

サンプルプログラム 11-2

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer player;

void setup(){
   size(100,100);
  minim = new Minim(this);// Mimin オブジェクトの生成
  player = minim.loadFile("schoolsong.mp3");
}

void draw(){
    // やりたいことを書く
}


void mouseClicked(){
  player.loop(); // ここを変更しました。
}
```

```
void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

もう一つの方法は、rewind メソッドと play メソッドを組み合わせて使う方法です。つまり、再生を開始する前に rewind メソッドを呼び出し、その直後に play メソッドを呼び出します。つまり、rewind メソッドで再生開始位置をファイルの先頭に戻してから、play メソッドで再生を開始します。サンプル 11-1 をこのように変更すると、マウスボタンをクリックする毎に、先頭から音声ファイルの再生が開始されます。この変更を加えたものがサンプル 11-3 です。

サンプルプログラム 11-3

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer player;

void setup(){
  size(100,100);
  minim = new Minim(this);// Mimin オブジェクトの生成
  player = minim.loadFile("schoolsong.mp3");
}

void draw(){
  // やりたいことを書く
}

void mouseClicked(){
  player.rewind(); // ここを変更しました。
  player.play(); // ここを変更しました。
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

一時停止

音声ファイルの再生を一時的に停止したいことがあります。この目的のために、pause メソッドが用意されています。マウスのクリッ

クだけでは、沢山の操作を区別することができないので、keyPressed 関数と組み合わせたサンプルを作ってみます。サンプル 11-4 では、マウスをクリックすると再生開始、p ボタンを押すと一時停止、r ボタンを押すと巻き戻しとします。

サンプルプログラム 11-4

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer player;

void setup(){
  size(100,100);
  minim = new Minim(this);
  player = minim.loadFile("schoolsong.mp3");
}

void draw(){
  // Write what you do
}

void mouseClicked(){
  player.play();
}

// ここを追加しました。
void keyPressed(){
  if(key == 'p'){
    player.pause();
  }else if(key == 'r'){
    player.rewind();
  }
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

ここで使用したメソッドをまとめると、以下の表のようになります。

表 11-1 AudioPlayer の再生に関連したメソッド

メソッド（関数）	機能
loadFile(“ ファイル名 ”)	音声ファイルの読み込む
play();	再生の開始

メソッド（関数）	機能
pause();	再生の一時停止
rewind();	再生開始位置を先頭に移動させる
close();	再生を中止し、ファイルストリーミングを閉じる
play(millis);	ファイルの先頭から millis 秒経過した場所から再生を開始する

ファイルサイズの余り大きくない場合

ファイルサイズの大きくないファイルを再生する場合には、AudioPlayer データ型（クラス）ではなく、AudioSnippet データ型（クラス）を利用します。AudioSnippet を利用する際には、音源ファイルを読み込むときに Minim クラスの loadSnippet メソッドを利用します。再生の方法は、AudioPlayer の場合と同じです。AudioSnippet を利用したものがサンプル 11-5 です。機能はサンプル 11-5 と同じです。

サンプルプログラム 11-5

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioSnippet player;

void setup(){
  size(100,100);
  minim = new Minim(this);
  player = minim.loadSnippet("schoolsong.mp3");
}

void draw(){
  // Write what you do
}

void mouseClicked(){
  player.play();
}

void keyPressed(){
  if(key == 'p'){
    player.pause();
  }else if(key == 'r'){
    player.rewind();
  }
}
```

```
void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

表 11-2 AudiSnippet の再生に関連したメソッド

関数（メソッド）	機能
loadSnippet(“ ファイル名 ”)	音声ファイルの読み込む
play();	再生の開始
pause();	再生の一時停止
rewind();	再生開始位置を先頭に移動させる

効果音を鳴らす

効果音など短い音を鳴らす時には AudioPlayer や AudioSnippet クラスではなく、AudioSample クラスを利用します。このクラスを利用するためには、音源ファイルを読み込むときに Minim クラスの loadSample メソッドを利用します。また、再生には trigger メソッドを使用します。この trigger メソッドは、必ず先頭から再生が始まります。また、ストリーミング再生ではないので、再生開始に遅れが発生することもありません。サンプル 11-6 は AudioSample を使ったものです。

サンプルプログラム 11-6

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioSample player;

void setup(){
  size(100,100);
  minim = new Minim(this);
// 読み込むファイルが変わっています。
  player = minim.loadSample("score.wav");
}

oid draw(){
  // Write what you do
}
void mouseClicked(){
  player.trigger();
}
```

```
void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

表 11-3 AudiSample の再生に関連したメソッド

関数（メソッド）	機能
loadSample(“ ファイル名 ”)	音声ファイルの読み込む
trigger();	再生の開始

音声ファイル情報の取得

mp3 ファイルにはアーティスト情報などのメタデータが記録されていることがあります。Minim には、このメタデータを取り出すための仕組みが用意されています。このメタデータの取り込みを行ったものがサンプル 11-7 です。

音声ファイルのメタデータを取り出すためには、getMetaData メソッドを使用します。メタデータを取り出した音声ファイルのデータが変数 player に代入されているとすると、「player.getMetaData();」を実行します。戻り値は、AudioMetaData 型となりますので、AudioMetaData 型の変数に代入しておきます。例えば、「meta = player.getMetaData();」を実行すると、メタデータが変数 meta に代入されます。そして、「meta.fileName()」とするとファイル名が取り出せます。また、「meta.length()」とすると再生時間（ミリ秒）の情報が取り出せます。これ以外にも、表 11-4 のようなメタデータを取り出すことが出来ます。なお、メタデータの情報が日本語で保存されていると、文字化けしてしまうようです。ちょっと面倒なことをすると、直ると思うのですが。

サンプルプログラム 11-7

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
Minim minim;
AudioPlayer player;
AudioMetaData meta;
PFont font;
void setup(){
  size(400,400);
  minim = new Minim(this);
  player = minim.loadFile("schoolsong.mp3");
  meta = player.getMetaData();// メタデータの読み込み
  font = loadFont("MS-Mincho-36.vlw");
  textFont(font,24);
}
```

```
void draw(){
  background(255);
  fill(0);
  text("File Name:" + meta.fileName(), 5, 50);
  text("Length (in milliseconds):" + meta.length(), 5, 50+60);
  text("Title:" + meta.title(), 5, 50+2*60);
  text("Author:" + meta.author(), 5, 50+3*60);
}

void mouseClicked(){
  player.play();
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

表 11-4 メタデータ取得に関連したメソッド

メソッド	機能
getMetaData()	メタデータの取得
fileName()	FileName
length()	演奏時間 (単位はミリ秒)
title()	タイトル
author()	演奏者
album()	アルバム名
date()	日付
comment()	コメント
track()	Track
genre()	Genre
copyright()	コピーライト
disc()	Disc
composer()	Composer
orchestra()	Orchestra
publisher()	Publisher
encoded()	Encoded

メタデータの各項目のうまい日本語訳があれば、教えて下さい。

再生中の情報取得

AudioPlayer などで再生を行っている場合には、どの場所を再生しているかの情報を取り出すことも出来ます。次の表 11-5 のようなデータを AudioPlayer 型などの変数から取り出すことが出来ます。

表 11-5 AudioPlayer 型などから直接取り出せる情報

メソッド	機能
length()	演奏時間 (単位はミリ秒)
position()	再生経過時間（単位はミリ秒）
isPlaying()	再生中かどうかを boolean 型のデータとして返す

サンプル 11-8 は、length と position を利用したものです。再生時間に応じて、バーが伸びてきます。音声ファイルの演奏時間とウインドウの幅は同じではないので、map 関数を利用して、バーの幅を計算しています。どうも、ちゃんと動いていな気が。length が返す値が少し大きいようです。

音声ファイルによっては、ちゃんと動くのですが。何か情報を持っている人がいたら教えて下さい。

サンプルプログラム 11-8

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer player;

void setup(){
  size(400,100);
  minim = new Minim(this);
  player = minim.loadFile("schoolsong.mp3");
}

void draw(){
  background(255);
  float x = map(player.position(),0,player.length(),0,width-1);
  stroke(0);
  fill(120);
  rect(0,0,x,height);
}

void mouseClicked(){
  player.play();
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

サンプル 11-9 は、isPlaying を利用して、再生中にマウスをクリックすると再生が中断し（pause）、再度マウスをクリックすると演奏が開始されるものです。なお、再生はプログラムの実行時から loop で行っています。

サンプルプログラム 11-9

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
```

```
Minim minim;
AudioPlayer player;

void setup(){
  size(400,100);
  minim = new Minim(this);
  player = minim.loadFile("schoolsong.mp3");
  player.loop();
}

void draw(){

}

void mouseClicked(){
  if(player.isPlaying()){
    player.pause();     // 再生中なら pause を実行
  }else{
    player.play();      // 再生中でなければ、play を実行
  }
}

void stop(){
  player.close(); // AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

正弦波などを鳴らす

音は空気の振動です。音の高低は波の周波数で決まります。Processing では、音楽ファイルを再生するだけでなく、周波数を指定して、音を発生させることが出来ます。ある一定の周波数の音を正弦波と呼びます。これを行っているのが、サンプル 11-10 です。正弦波で音の鳴らすためには、どのような波形かという情報とそれをどこに音を出すかの 2 つの情報が必要となります。

正弦波の発生 サンプル 11-10

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioOutput out;
SineWave sine;
```

```
void setup(){
  minim = new Minim(this);
  out = minim.getLineOut(Minim.STEREO);
  sine = new SineWave(440, 0.5, out.sampleRate());
  out.addSignal(sine);
}
void draw(){
}
void stop(){
  out.close(); // ライン出力の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

前者の情報を与えるために、SineWave メソッドを利用します。このメソッドは任意の周波数の正弦波を生成することができます。後者の情報を与えるために、getLineOut メソッドを利用しています。getLineOut メソッドは AudioOutput 型の値を返します。このサンプルでは、ステレオで音の出力を行うので、getLineOut の引数に、Minim.STEREO を渡しています。もし、モノラルでの出力を行う場合には、Minim.MONO とします。これに、addSignal メソッドで発生される波形を設定することで、正弦波の音波を出すことができます。周波数で考えると、周波数を倍にすると 1 オクターブ上，半分にすると 1 オクターブ下になります。

通常の音声や楽器の音などは、1 つの周波数の音だけでなく、複数の正弦波が混ざって出来ています。フーリエ級数という数学の理論を利用すると、様々な波形は複数の正弦波を足しあわせたものとして表現することが出来ます。Processing でも複数の正弦波を足しあわせた音を鳴らすことが出来ます。これを行ったものが、サンプル 11-11 です。

正弦波の組み合わせ　サンプル 11-11

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioOutput out;
SineWave sine1,sine2,sine3;
```

```
void setup(){
  minim = new Minim(this);
  out = minim.getLineOut(Minim.STEREO);
  sine1 = new SineWave(440, 0.5, out.sampleRate());
  sine2 = new SineWave(880, 0.2, out.sampleRate());
  sine3 = new SineWave(1760, 0.1, out.sampleRate());
  out.addSignal(sine1);
  out.addSignal(sine2);
  out.addSignal(sine3);
}
void draw(){
}
void stop(){
  out.close(); // ライン出力の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

Minim では、正弦波だけなく、矩形波やノコギリ波の発生も行うことが出来ます。矩形波やノコギリ波で音を鳴らすためには、サンプル 11-10 の正弦波を生成している部分を、矩形波やノコギリ波を発生させるものに変更すれば、大丈夫です。表 11-6 に各種の波形の発生方法をのせておきます。サンプリングレートは、どれ位の間隔で発生させるデータの値を計算するか、どの位の間隔でデータを取り込むかを表す値です。通常は、現在のサンプリングレートを使えば大丈夫です。サンプル 11-10 や 11-11 では、波形情報を出力する先である、AudioOutput 型の out が持っているメソッド sampleRate() を使って、現在のサンプルレートを取得しています。

表 11-6 波形発生のメソッド

波形の種類	メソッド名
正弦波	SineWave(周波数 , 振幅 , サンプリングレート) 振幅は 0 ～ 1 の数値
矩形波	SquareWave(周波数 , 振幅 , サンプリングレート)
ノコギリ波	SawWave(周波数 , 振幅 , サンプリングレート)

そこで、矩形波を発生させるようにしたものが、サンプル 11-12 です。

矩形波の発生 サンプル 11-12

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioOutput out;
SquareWave squ;

void setup(){
  minim = new Minim(this);
  out = minim.getLineOut(Minim.STEREO);
  squ = new SquareWave(440, 0.5, out.sampleRate());
  out.addSignal(squ);
}

void draw(){
}
void stop(){
  out.close(); // ライン出力の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

サンプル 11-13 はノコギリ波を発生させるものです。

ノコギリの発生 サンプル 11-13

```
import ddf.minim.*;
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioOutput out;
SawWave saw;

void setup(){
  minim = new Minim(this);
  out = minim.getLineOut(Minim.STEREO);
  saw = new SawWave(440, 0.5, out.sampleRate());
  out.addSignal(saw);
}
void draw(){
}
void stop(){
  out.close(); // ライン出力の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

また、SineWave 型、SquareWave 型、SawWave 型は、次のようなメソッドを持っています。

表 11-7 波形発生に関するメソッド

メソッド名	昨日
setFreq(周波数);	派生させる音の周波数を変更する
setAmp(振幅)	派生させる音の振幅を変更する (0 ～ 1)
setPan(パン位置)	-1（左チャンネルのみ）～ 0（中央）～ 1（右チャンネルのみ）の範囲の数値で、パン位置を設定する

表 11-7 に載っている setFreq と setAmp を使用したサンプルを示します。サンプル 11-14 では、マウスの X 座標の値を使ってパンの値を決めています。つまり、マウスの X 座標の値が 0 なら（一番左なら）パンの位置を -1 に、マウスの X 座標の値が width-1 なら（一番右なら）パンの位置を 1 に設定しています。また、マウスの Y 座標の値によって周波数を変更しています。マウスが一番上なら（0 なら）周波数を 400Hz、マウスが一番下なら（height-1 なら）1600Hz に設定しています。途中の値は、map 関数を使って計算し、その値を setPan メソッドや setFreq メソッドに渡しています。これらの設定は、マウスが動いたときに行えば良いので、これらの処理は mouseMoved 関数の中に書かれています。

表 11-7 のメソッドを利用 サンプル 11-14

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioOutput out;
SineWave sine;

void setup(){
  size(600,200);
  minim = new Minim(this);
  out = minim.getLineOut(Minim.STEREO);
  sine = new SineWave(440, 0.5, out.sampleRate());
  out.addSignal(sine);
}
void draw(){
  background(255);
  stroke(0);
  fill(200);
  ellipse(mouseX,mouseY,40,40);
}
```

```
void mouseMoved(){
// 周波数を計算する
  float freq = map(mouseY,0,height-1,400,1600);
// パンの値を計算する
  float pan = map(mouseX,0,width-1,-1,1);
  sine.setFreq(freq);// 周波数を変更する
  sine.setPan(pan);  // パン位置を変更する
}
void stop(){
  out.close(); // ライン出力の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

複数のファイルを扱う

ここでは、複数のファイルを扱うサンプルを紹介します。このサンプルでは、AudioSample を利用して、音声ファイルの再生を行います。1 ～ 4 までの数字キーを押すと対応するファイルの再生が行われます。このサンプルでは、AudioSample 型の配列に loadSample メソッドの実行結果を保存し、keyPressed 関数の中で Processing 変数の key の値を調べ、対応する音声ファイルの再生を行っています。変数 key には押されたキーの情報が入っているので、キー 1 が押されたかは、key=='1' で調べることができます。

複数音声ファイルの扱い例 サンプル 11-15

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioSample[] se; // 音声ファイルの情報をしまう配列
void setup(){
  size(100,100);
  minim = new Minim(this);
  se = new AudioSample[4];// 音声ファイルの情報をしまう配列の確保
// 音声ファイルの読み込み
  se[0] = minim.loadSample("appear01.wav");
  se[1] = minim.loadSample("appear02.wav");
  se[2] = minim.loadSample("appear03.wav");
  se[3] = minim.loadSample("appear04.wav");
}

void draw(){
// 何も書いてなくても、これがないと音が鳴りません。
}
```

```
void keyPressed(){
  if(key == '1'){
    se[0].trigger();
  }else if(key == '2'){
    se[1].trigger();
  }else if(key == '3'){
    se[2].trigger();
  }else if(key == '4'){
    se[3].trigger();
  }
}
void stop(){
// すべての AudioPlayer の機能を終了する必要があります。
  for(int i=0;i<se.length;i++){
    se[i].close();// AudioPlayer の機能を終了する
  }
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

サンプル 11-15 にちょっとした機能を付け加えると、少しゲームのようなプログラムを作ることが出来ます。サンプル 11-16 では、millis 関数を利用して時間を計り、1 秒 (1000 ミリ秒) 経つと、押すべきキーの表示が変わります。押すべきキーの決定には、random 関数を利用しています。

複数音声ファイルの扱い例その 2 サンプル 11-16

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
Minim minim;
AudioSample[] se;
int startTime; //経過時間を計るための変数
int idx; // どのキーを押すべきかを決める変数
PFont font;
void setup() {
  size(300, 100);
  font = createFont("Serif", 48);
  textFont(font, 36);
  textAlign(CENTER);
  minim = new Minim(this);
  se = new AudioSample[4];
  se[0] = minim.loadSample("appear01.wav");
  se[1] = minim.loadSample("appear02.wav");
  se[2] = minim.loadSample("appear03.wav");
  se[3] = minim.loadSample("appear04.wav");
  update();
}
```

```
// 一定時間経過したので、情報を更新する
void update() {
  startTime = millis();
  idx = int(random(4))+1;
}
void draw() {
  background(255);
  fill(0);
  text("Hit "+idx+" key", width/2, height/2);
  if (millis()-startTime >= 1000) {// 1 秒経過したので情報を更新
    update();
  }
}
void keyPressed() {
  if (key == '1') {
    if (idx == 1) { // 押されたキーが指定されたキーかを調べる
      se[0].trigger();
    }
  }else if (key == '2') {
    if (idx == 2) { // 押されたキーが指定されたキーかを調べる
      se[1].trigger();
    }
  }else if (key == '3') {
    if (idx == 3) { // 押されたキーが指定されたキーかを調べる
      se[2].trigger();
    }
  }else if (key == '4') {
    if (idx == 4) { // 押されたキーが指定されたキーかを調べる
      se[3].trigger();
    }
  }
}
void stop(){
// すべての AudioPlayer の機能を終了する必要があります。
  for(int i=0;i<se.length;i++){
    se[i].close();// AudioPlayer の機能を終了する
  }
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

波形の描画

Minim は音声ファイルの再生だけはなく、色々なことができます。そのうちの一つが入力や出力される音声データの取り込みです。これを利用すると波形の描画をすることが出来ます。

読み込まれた音声データは、バッファ (buffer) と呼ばれる場所に少しずつコピーをされながら、再生されていきます。このバッファの中に保存されている値を取り出すのが get メソッドです。ステレオの場合には、左右があるので、サンプル 11-17 の赤色の行のように、左

右を指定して取り出します。取り出される情報は -1 ～ 1 までの数値データです。このバッファの大きさ(いくつのデータが入っているか)を取り出すのが bufferSize メソッドです。これを利用して波形データを描いたものが、サンプル 11-17 です。

波形データの表示その 1 サンプル 11-17

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer song;

void setup(){
  size(400,200);
  smooth();
  minim = new Minim(this);
  song = minim.loadFile("schoolsong.mp3");
  song.loop();
}
void draw(){
  background(255);
  stroke(0);
  beginShape();
  for(int i = 0; i < song.bufferSize() ; i++){
    float x=map(i,0,song.bufferSize(),0,width-1);
    vertex(x, 60 + song.left.get(i)*50);
  }
  endShape();
  beginShape();
  for(int i = 0; i < song.bufferSize() ; i++){
    float x=map(i,0,song.bufferSize(),0,width-1);
    vertex(x, 170 + song.right.get(i)*50);
  }
  endShape();
}
void stop(){
  song.close();// AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```

Minim では、音声ファイルからだけはなく、パソコンに付いているマイクからの音声情報を取り出すことが出来ます。基本的にはサンプル 11-17 と同じですが、マイクからの入力になるので、AudioPlayer の代わりに AudioInput 型の変数に minim.getLineIn(Minim.STEREO) の戻り値を代入します。この変数から音声情報を取り出すことが出来ます。これを利用したものがサンプル 11-18 です。サンプル 11-17

と異なっているのは、赤字の部分です。

波形データの表示その 2 サンプル 11-18

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;
Minim minim;
AudioInput in;
int bufferSize = 1024;
float [] buffer = new float[bufferSize];
void setup(){
  size(400,230);
  smooth();
  minim = new Minim(this);
  in = minim.getLineIn(Minim.STEREO);
}
void draw(){
  background(255);
  stroke(0);
  beginShape();
  for(int i = 0; i < in.bufferSize() ; i++){
    float x=map(i,0,in.bufferSize(),0,width-1);
    vertex(x, 60 + in.left.get(i)*50);
  }
  endShape();
  beginShape();
  for(int i = 0; i < in.bufferSize() ; i++){
    float x=map(i,0,in.bufferSize(),0,width-1);
    vertex(x, 170 + in.right.get(i)*50);
  }
  endShape();
}
void stop(){
 in.close();// AudioPlayer の機能を終了する
 minim.stop(); // Minim の機能を停止する
 super.stop(); // 停止の際のおまじない
}
```

FFT

音声情報などを分析する際に FFT と呼ばれる方法を利用することがあります。Minim では、この FFT の機能を持っています。FFT の説明をするには、下手をすると一学期かかってしまいます。このあたりの話は、サウンド解析やサウンド情報処理で扱われます。その授業を受ける際にでも、Processing で FFT が出来たことを思い出して下さい。FFT の機能を使ったものがサンプル 11-19 です。興味のある人は、マニュアルなどを頼りに、動作を調べて見て下さい。

FFT サンプル 11-19

```
import ddf.minim.*;
import ddf.minim.signals.*;
import ddf.minim.analysis.*;
import ddf.minim.effects.*;

Minim minim;
AudioPlayer song;
FFT fft;

void setup(){
  size(512, 200);
// always start Minim first!
  minim = new Minim(this);
  song = minim.loadFile("schoolsong.mp3", 512);
  song.loop();
  fft = new FFT(song.bufferSize(), song.sampleRate());
}
void draw(){
  background(0);

  fft.forward(song.mix);
  stroke(255, 0, 0, 128);
  for(int i = 0; i < fft.specSize(); i++){
    line(i, height, i, height - fft.getBand(i)*4);
  }
  stroke(255);
  for(int i = 0; i < song.left.size() - 1; i++){
    line(i, 50 + song.left.get(i)*50, i+1, 50 + song.left.
get(i+1)*50);
    line(i, 150 + song.right.get(i)*50, i+1, 150 + song.right.
get(i+1)*50);
  }
}

void stop(){
  song.close();// AudioPlayer の機能を終了する
  minim.stop(); // Minim の機能を停止する
  super.stop(); // 停止の際のおまじない
}
```