CrestMuse Toolkit (CMX) ver.0.60
説明書

北原 鉄朗
（日本大学 文理学部 情報システム解析学科）
kitahara [at] kthrlab.jp

Lesson 0　概要

CMXで何ができるのか

- 音楽データに対する様々な処理を容易にするJavaライブラリ.
- たとえば, こんなことができる.
  - MusicXML, 標準MIDIファイルなどの入出力
  - MIDIやオーディオデバイスからの入力のリアルタイム処理
  - ベイジアンネットワークを用いた音楽データの自動生成（weka.jarが別途必要）
- プログラミングのためのライブラリであって, 単体で便利に使えるアプリではない.
- Javaの他, JVM上の各種言語（Processing, Groovy, etc.）から利用できる.

インストール方法

- zipファイルを展開し，cmx.jarと，libに入っているいくつかのjarファイルを，所定のディレクトリ（フォルダ）にコピーすればOK
  - Processingから使う場合：
    ~/sketchbook/libraries/cmx/library
  - Groovyから使う場合：~/.groovy/lib/
  - Javaから使う場合：（JDKのインストール先）/jre/lib/ext/
- 任意の場所に置いてCLASSPATHを通すのでもよい.
- install.sh を実行すると，これを自動でやってくれる.
  （ルート権限で実行すること. つまり，$ sudo ./install.sh ）

上記は，UNIX系OSの場合. OSによってファイルの置き場などが異なるので，自身で確認すること.

使用方法 その1 ～コマンドラインから～

- MusicXML, SCCXML, MIDIXML, 標準MIDIファイルなどの相互変換をコマンドラインから行うことができる.
- 基本的な使い方

```
$ cmx （Javaコマンドのオプション）コマンド名（オプション）
```

- 例

```
$ cmx smf2scc myfile.mid
```

「myfile.mid」という名の標準MIDIファイルをSCCXML形式に変換

```
$ cmx smf2scc myfile.mid -o mysccfile.xml
```

上と同様の変換を行って「mysccfile.xml」に保存

- ヘルプ

```
$ cmx help
```

使用方法 その2 ～Processingから～

- CMXには様々な機能があり，それらを実現している膨大なクラスが存在するが，主要な機能は CMXController というクラスから呼び出せるようになっている．

```
import jp.crestmuse.cmx.processing.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  /* ここで1回だけ行う処理を記述する */
}

void draw() {
  /* ここで繰り返し実行する処理を記述する */
}
```

詳細は，CMXControllerのJavaDocを確認すること．

使用方法 その3 ～Groovyから～

- CMXの機能に加えProcessingの機能も使えるように，ProcessingとCMXControllerの両方の機能を備えたCMXAppletというクラスを用意しているので，これを利用する．

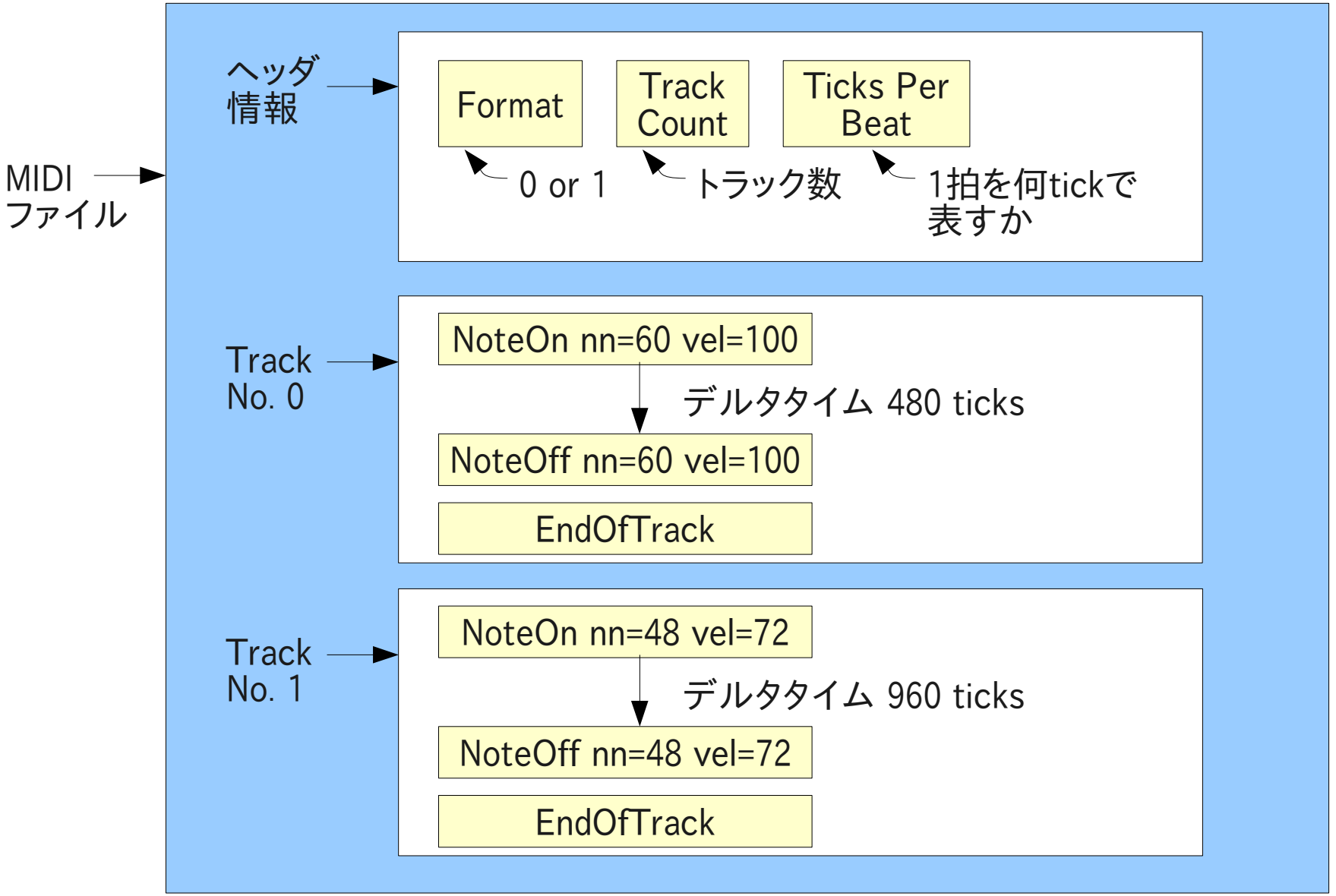
```
import jp.crestmuse.cmx.processing.*

class MyApplet extends CMXApplet {
  void setup() {
    /* ここで1回だけ行う処理を記述する */
  }
  void draw() {
    /* ここで繰り返し実行する処理を記述する */
  }
}

MyApplet.start("MyApplet")
```

詳細は，CMXAppletのJavaDocを参照すること．

Lesson 1　MIDIファイルを読み書きする

標準MIDIファイル（SMF）とは

基本的にはMIDIケーブルに流れるMIDI信号をファイルに書き出したもの



SMFの中身を見てみたい

- SMFはバイナリーファイルなので，中身を簡単に見ることができない
- 既存のMIDIシーケンサ（e.g. Rosegarden）にインポートすれば中身を見ることはできるが，MIDIシーケンサでは音楽データを独自のフォーマットで表す場合が多く，インポート時に独自フォーマットに変換されてしまっていることが考えられる．
- CrestMuse Toolkitでは，SMFをそのままXML形式に書き直した「MIDI XML」という形式に対応している．この形式に変換してみよう．

cmxの使い方の基本

```
$ cmx 処理内容 オプション 入力ファイル名
```

このドル記号は，端末のプロンプト（コマンド入力待ち）を表し，自分で入力するわけではない．

処理内容

- smf2midi: SMF→MIDI XML
- smf2scc: SMF→SCCXML
- midi2smf: MIDI XML→SMF
- scc2midi: SCCXML→MIDI XML
- などなど

オプション

- -o ファイル名:　XMLの場合の出力ファイル指定
- -smf ファイル名: SMFの場合の出力ファイル指定

-o を省略すると画面に表示される

例

```
$ cmx smf2midi -o sample1.xml sample.mid
```

「sample.mid」というSMFをMIDI XMLに変換して「sample1.xml」というファイルに出力

MIDI XMLの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MIDIFile PUBLIC "-//Recordare//DTD MusicXML 1.1 MIDI//EN"
                          "http://www.musicxml.org/dtds/midixml.dtd">
<MIDIFile>
  <Format>1</Format>
  <TrackCount>2</TrackCount>
  <TicksPerBeat>480</TicksPerBeat>
  <TimestampType>Delta</TimestampType>
  <Track Number="1">
    <Event>
      <Delta>1920</Delta>
      <EndOfTrack/>
    </Event>
  </Track>
  <Track Number="2">
    <Event>
      <Delta>0</Delta>
      <NoteOn Channel="1" Note="67" Velocity="100"/>
    </Event>
    <Event>
      <Delta>0</Delta>
      <ControlChange Channel="1" Control="7" Value="100"/>
    </Event>
    <Event>
      <Delta>0</Delta>
      <ProgramChange Channel="1" Number="0"/>
    </Event>
```

続き

```
<Event>
  <Delta>360</Delta>
  <NoteOff Channel="1" Note="67" Velocity="100"/>
</Event>
<Event>
  <Delta>0</Delta>
  <NoteOn Channel="1" Note="69" Velocity="100"/>
</Event>
<Event>
  <Delta>120</Delta>
  <NoteOff Channel="1" Note="69" Velocity="100"/>
</Event>
<Event>
  <Delta>0</Delta>
  <NoteOn Channel="1" Note="67" Velocity="100"/>
</Event>
<Event>
  <Delta>240</Delta>
  <NoteOff Channel="1" Note="67" Velocity="100"/>
</Event>
<Event>
  <Delta>0</Delta>
  <NoteOn Channel="1" Note="65" Velocity="100"/>
</Event>
<Event>
  <Delta>240</Delta>
  <NoteOff Channel="1" Note="65" Velocity="100"/>
</Event>
```

（残り省略）

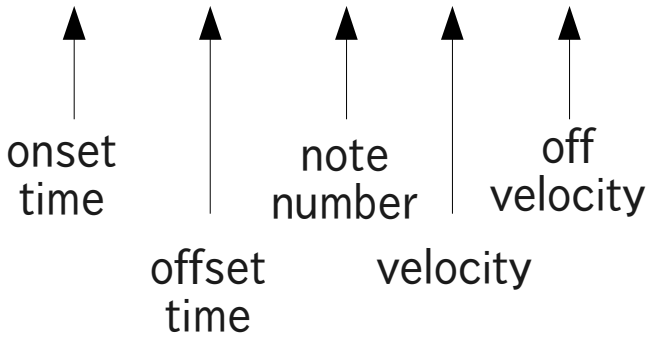
もう1つのXML形式「SCCXML」

- MIDI XMLは、SMFを完全にそのままXML化しているので、どんなSMFでも変換できるのがメリットだが、煩雑である。
- 特に、発音（Note On）と消音（Note Off）が分かれており、たとえば、
  ある音符の長さを調べようと思ったら、Note OnとNote Offの対応関係を調べるなど、ややこしい処理をしないといけない。
- より単純化された形式として、SCCXMLというものを用意している。
- 「sample.mid」というSMFをSCCXMLに変換する：

```
$ cmx smf2scc -o sample2.xml sample.mid
```

SCCXMLの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE scc PUBLIC "-//CrestMuse//DTD CrestMuseXML SCCXML//EN"
                     "http://www.crestmuse.jp/cmx/dtds/sccxml.dtd">
<scc division="480">
  <header/>
  <part ch="1" pn="0" serial="1" vol="100">
    <note>0 360 67 100 100</note>
    <note>360 480 69 100 100</note>
    <note>480 720 67 100 100</note>
    <note>720 960 65 100 100</note>
    <note>960 1200 64 100 100</note>
    <note>1200 1440 65 100 100</note>
    <note>1440 1920 67 100 100</note>
    <note>1920 2160 62 100 100</note>
    <note>2160 2400 64 100 100</note>
    <note>2400 2880 65 100 100</note>
    <note>2880 3120 64 100 100</note>
    <note>3120 3360 65 100 100</note>
    <note>3360 3840 67 100 100</note>
  </part>
</scc>
```

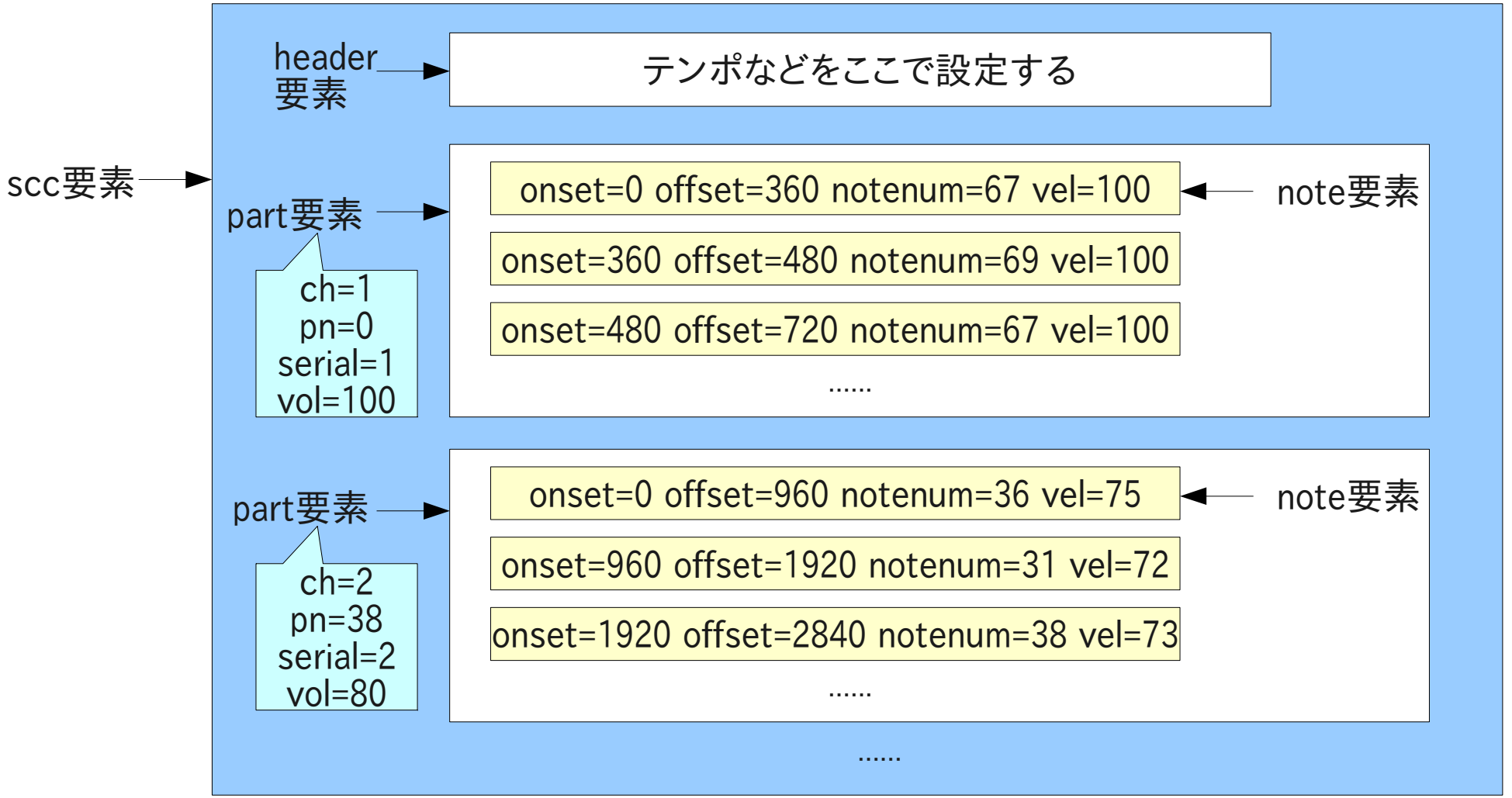


MIDIシーケンサの
「イベントリスト画面」に似せた表記

チャンネル番号(cn)、音色番号(pn)、
ボリューム(vol)をpartの属性として記述

頑張れば手入力もできる程度の複雑さ

SCCXMLの構造



- SCCXMLは、1つのscc要素で構成される
- scc要素は、1つのheader要素と0個以上のpart要素で構成される
- part要素は、0個以上のnote要素で構成される（他にもあるが）

CMXFileWrapperクラス

- CrestMuse Toolkitでは，読み書きできるファイル形式の1個1個にラッパクラスが用意されている．
- ラッパクラスは，共通の基底クラスCMXFileWrapperのサブクラス．
- ラッパクラスの例：

  MusicXML形式 → MusicXMLWrapperクラス

  SCCXML形式 → SCCXMLWrapperクラス

  MIDIXML形式 → MIDIXMLWrapperクラス
- CMXControllerクラスにあるreadまたはreadfileメソッドを使うと，ファイル形式に合ったラッパクラスのオブジェクトが得られるので，ダウンキャストして用いる．

```
CMXController cmx = CMXController.getInstance();
SCCXMLWrapper scc = (SCCXMLWrapper)cmx.readfile("myscc.xml");
```

SCCXMLを読み込んで，各音符の情報を取得する

さきほどの方法を用いてSCCXMLWrapperオブジェクトを得たら，

SCCXMLWrapperクラスが提供する各メソッドを用いればよい

例：SCCXMLファイルを読み込んで，各パート/各音符の情報を標準出力


```
import jp.crestmuse.cmx.filewrappers.*
import jp.crestmuse.cmx.processing.*

def cmx = CMXController.getInstance()
def scc = cmx.readfile("sample.xml")
def partlist = scc.getPartList()
partlist.each { part ->
  println("パート発見！")
  println("ch=${part.channel()}, pn=${part.prognum()}" +
          "serial=${part.serial()}, vol=${part.volume()}")
  notelist = part.getNoteOnlyList()
  notelist.each { note ->
    println("音符発見！")
    println("onset=${note.onset()}, offset=${note.offset()}" +
            "notenum=${note.notenum()}, vel=${note.velocity()}")
  }
}
```

例: SCCXMLファイルを読み込んで, 各パート/各音符の情報を標準出力

```
import jp.crestmuse.cmx.filewrappers.*;
import jp.crestmuse.cmx.processing.*;
import javax.xml.transform.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  try {
    SCCXMLWrapper scc =
       (SCCXMLWrapper)cmx.read(createInput("sample.xml"));
    SCCXMLWrapper.Part[] partlist = scc.getPartList();
    for (SCCXMLWrapper.Part part : partlist) {
      println("パート発見！");
      SCCXMLWrapper.Note[] notelist = part.getNoteOnlyList();
      for (SCCXMLWrapper.Note note : notelist) {
        println("onset="+note.onset()+", offset="+note.offset()+
                ", notenum="+note.notenum()+", vel="+note.velocity());
      }
    }
  } catch (TransformerException e) {
    println("エラー");
  }
}

void draw() {
}
```

Lesson 2　MusicXMLを読み込み，楽譜に付与されている記号に従って演奏を生成する

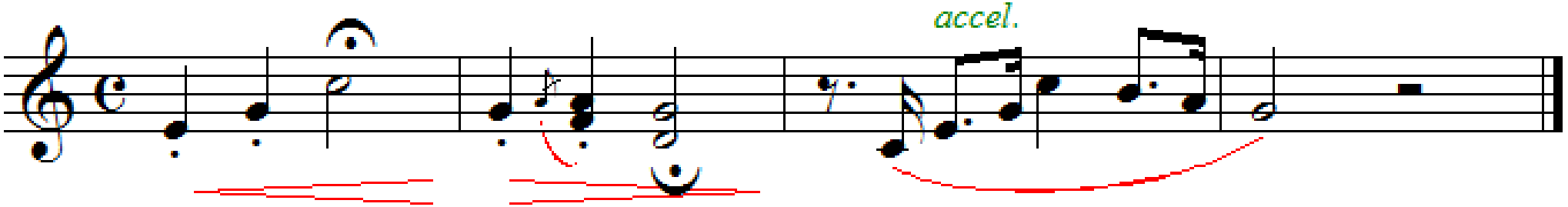
MusicXMLとは

楽譜を記述するためのXMLフォーマット

なんでMIDIファイルじゃダメなの?

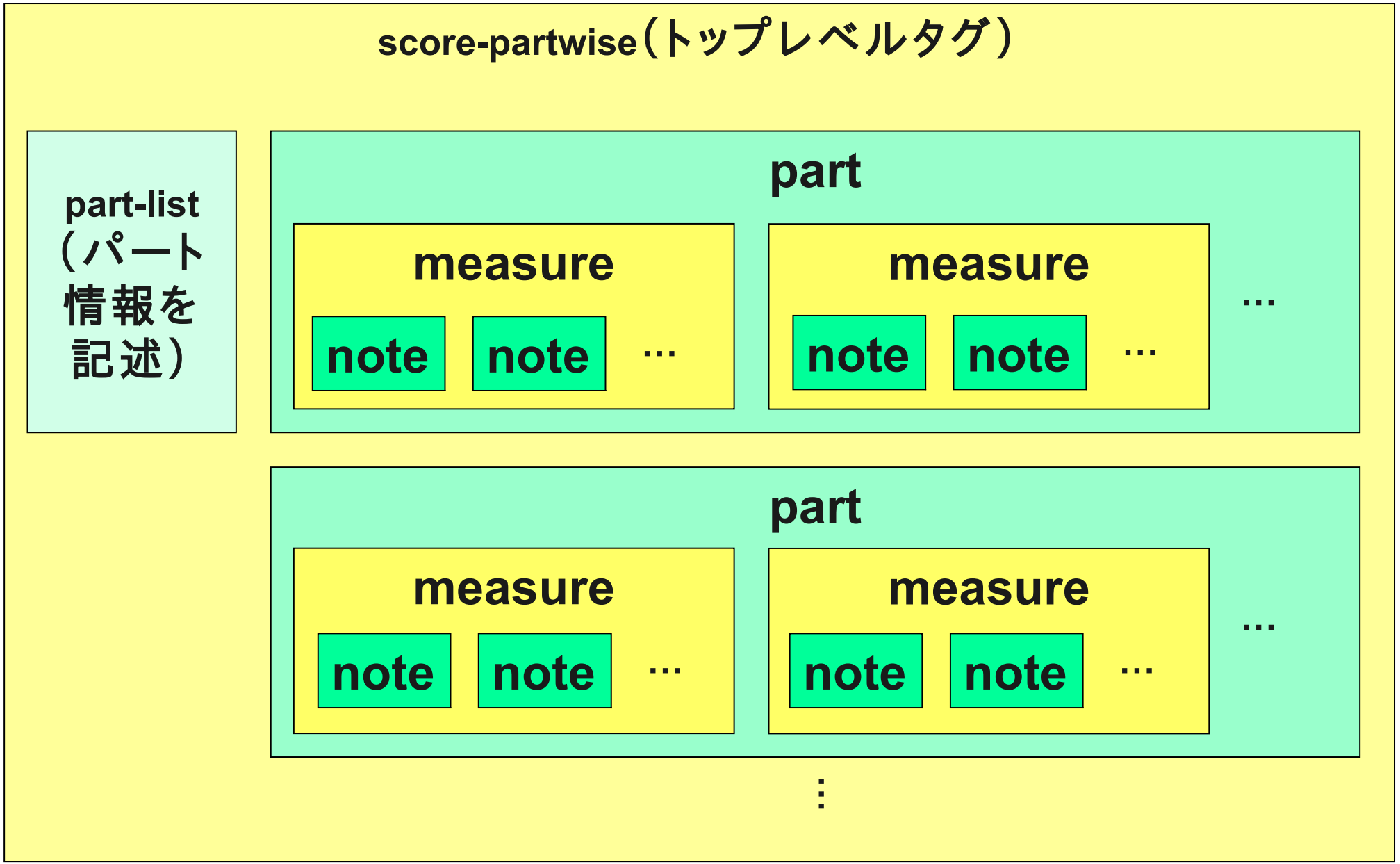
- MIDIファイルは、楽譜ではなく「演奏」を記録するフォーマット
  - 小節線や休符という概念がない。
  - スタッカートやフェルマータなどを記述できない。

MusicXMLの例




```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE score-partwise PUBLIC
"-//Recordare//DTD MusicXML 1.0 Partwise//EN"
"http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
   （中略）
  <part-list>
    <score-part id="P1">
     （中略）
    </score-part>
  </part-list>
  <!--=================================-->
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>8</divisions>
        （中略）
      </attributes>
      <direction placement="below">
        <direction-type>
          <wedge default-y="-72" spread="0"
                      type="crescendo"/>
        </direction-type>
        <offset>3</offset>
      </direction>
      <note>
        <pitch>
          <step>E</step>
          <octave>4</octave>
        </pitch>
        <duration>8</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        <notations>
          <articulations>
            <staccato placement="below"/>
          </articulations>
        </notations>
      </note>
      <note>
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>8</duration>
        <voice>1</voice>
        <type>quarter</type>
        <stem>up</stem>
        <notations>
          <articulations>
            <staccato placement="below"/>
          </articulations>
        </notations>
      </note>
      <note>
        <pitch>
          <step>C</step>
          <octave>5</octave>
        </pitch>
        <duration>16</duration>
        <voice>1</voice>
        <type>half</type>
        <stem>down</stem>
        <notations>
          <fermata type="upright"/>
        </notations>
      </note>
      <direction>
        <direction-type>
          <wedge default-y="-71“
            spread="12" type="stop"/>
        </direction-type>
        <offset>-3</offset>
      </direction>
    </measure>
    <!--=========================-->
    <measure number="2">
      <direction placement="below">
        <direction-type>
以下省略
```

MusicXML(partwise)の基本構造



演奏生成の基本方針

- ここでのタスクは,「楽譜を読んでそれを演奏すること」
- ここでは簡単に,
  ・スタッカートが付いている音符は半分の長さで,
  ・フェルマータが付いている音符は半分のテンポで
  演奏することとする.
- 楽譜はMusicXML, 演奏はMIDI(SCCXML)で表される
- 演奏は, 完全に楽譜通りの演奏から適度に逸脱させることで音楽らしさを作っている. この逸脱の情報の記述のために DeviationInstanceXML が用意されている.
- まず, DeviationDataSetクラスのインスタンスを生成し, DeviationInstanceWrapperオブジェクトに変換し, その後, SCCXMLWrapperオブジェクトに変換する.

MusicXMLから各音符の情報を取得する

Lesson 1と同様に，CMXControllerのreadまたはreadfileメソッドでファイルを読み込む（MusicXMLWrapperオブジェクトが得られる）


```
import jp.crestmuse.cmx.filewrappers.*
import jp.crestmuse.cmx.processing.*

def cmx = CMXController.getInstance()
def musicxml = cmx.readfile("sample.xml")
musicxml.eachnote { note ->
  if (note.hasArticulation("stacatto") {
    /* スタッカートがあったときの処理 */
  } else {
    /* スタッカートがなかったときの処理 */
  def notations = note.getFirstNotations()
  if (notations != null && notations.fermata() != null) {
    /* フェルマータがあったときの処理 */
  } else {
    /* フェルマータがなかったときの処理 */
  }
}
```

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.filewrappers.*;
import jp.crestmuse.cmx.filewrappers.MusicXMLWrapper.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  MusicXMLWrapper musicxml =
              MusicXMLWrapper)cmx.read(createInput("sample.xml"));
  Part[] partlist = musicxml.getPartList();
  for (Part part : partlist) {
    Measure[] measurelist = part.getMeasureList();
    for (Measure measure : measurelist) {
      MusicData[] mdlist = measure.getMusicDataList();
      for (MusicData md : mdlist) {
        if (md instanceof Note) {
          Note note = (Note)md;
          if (note.hasArticulation("staccato")) {
            /* スタッカートがあったときの処理 */
          } else {
            /* スタッカートがなかったときの処理 */
          }
          Notations notations = note.getFirstNotations();
          if (notations != null && notations.fermata() != null) {
            /* フェルマータがあったときの処理 */
          } else {
            /* フェルマータがなかったときの処理 */
          }
        }
      }
    }
  }
}

void draw() {
}
```

演奏データを生成する

ここでは，実際の演奏データの代わりに，
演奏の楽譜からの差分（deviationデータ）を生成する．

どうやって?

deviationの生成には，DeviationDataSetクラスを使う．

- ある音符の長さを短くするには，addNoteDeviationを使う．

  addNoteDeviation(attack, release, dynamics, endDynamics);

  attack：発音タイミング
  release：消音タイミング
  dynamics：発音の強さ
  endDynamics：消音の強さ

  attack, release：楽譜通りの発音・消音時刻を0、四分音符の長さを1.0とした相対値

  dynamics, endDynamics：基準値を1.0とした相対値

- テンポを変えるには，addNonPartwiseControlを使う．

  addNonPartwiseControl(measure, beat, “tempo-deviation”, value);

  measure：小節番号
  beat：拍番号
  value：基準テンポを1.0とした相対値

```
import jp.crestmuse.cmx.filewrappers.*
import jp.crestmuse.cmx.processing.*

def cmx = CMXController.getInstance()
def musicxml = cmx.readfile("sample.xml")
def deviation = new DeviationDataSet(musicxml)
musicxml.eachnote { note ->
  if (note.hasArticulation("stacatto"))
    deviation.addNoteDeviation(note, 0.0,
                          -note.actualDuration()/2, 1.0, 1.0)
  def notations = note.getFirstNotations()
  if (notations != null && notations.fermata() != null)
    deviation.addNonPartwiseControl(note.measure().number(),
                          note.beat(), "tempo-deviation", 0.5)
}
def devins = deviation.toWrapper()
devins.finalizeDocument()
devins.writefile("deviation.xml")
devins.toSCCXML(480).toMIDIXML().writefileAsSMF("result.mid")
```

（Processingコードは省略）

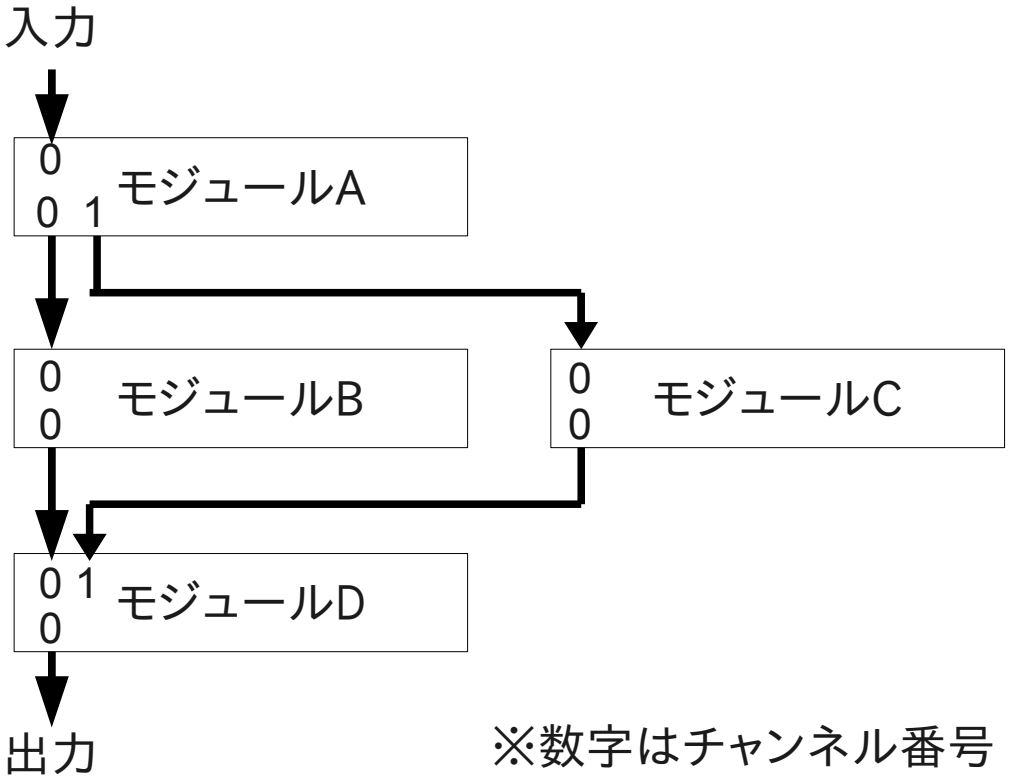
Lesson 3　MIDI入力をリアルタイムで処理する

基本的な考え方

処理全体がいくつかの「モジュール」に分割され、
モジュールの中をデータが流れていくものと考える

||

データフロー型プログラミング



※数字はチャンネル番号

「モジュール」の特徴

- 何らかのデータが入力され、処理された後、出力される。
- 複数種のデータが入力or出力されてもいい。
- データが到着するたびに、処理が自動的に行われる。

基本的な処理の流れ

- 「モジュール」のオブジェクトを生成する
  - モジュールは，SPModuleクラスのサブクラスである
  - 主要なモジュールは，CMXController または CMXAppletクラスから生成することができる（すべてではない）
- 「モジュール」を「登録」する
  - CMXController または CMXApplet の addSPModuleメソッドを用いる
- 「モジュール」の接続方法を定義する
  - CMXController または CMXApplet の connectメソッドを用いる
- 「モジュール」を実行を開始する
  - CMXControllerクラスのstartSPメソッドを用いる
  - CMXAppletクラスの場合は自動的に開始する

Step 1 既存のモジュールを組み合わせて使う

- 下は，仮想鍵盤での演奏をMIDIデバイスに出力するプログラム
  - createVirtualKeyboard ･･･ 仮想鍵盤のモジュール
  - createMidiOut ･･･ MIDIデバイスに出力するモジュール


```
import jp.crestmuse.cmx.processing.*

class MyApplet extends CMXApplet {

  void setup() {
    def vk = createVirtualKeyboard()
    def mo = createMidiOut()
    addSPModule(vk)
    addSPModule(mo)
    connect(vk, 0, mo, 0)
  }

  void draw() {
  }
}

MyApplet.start("MyApplet")
```

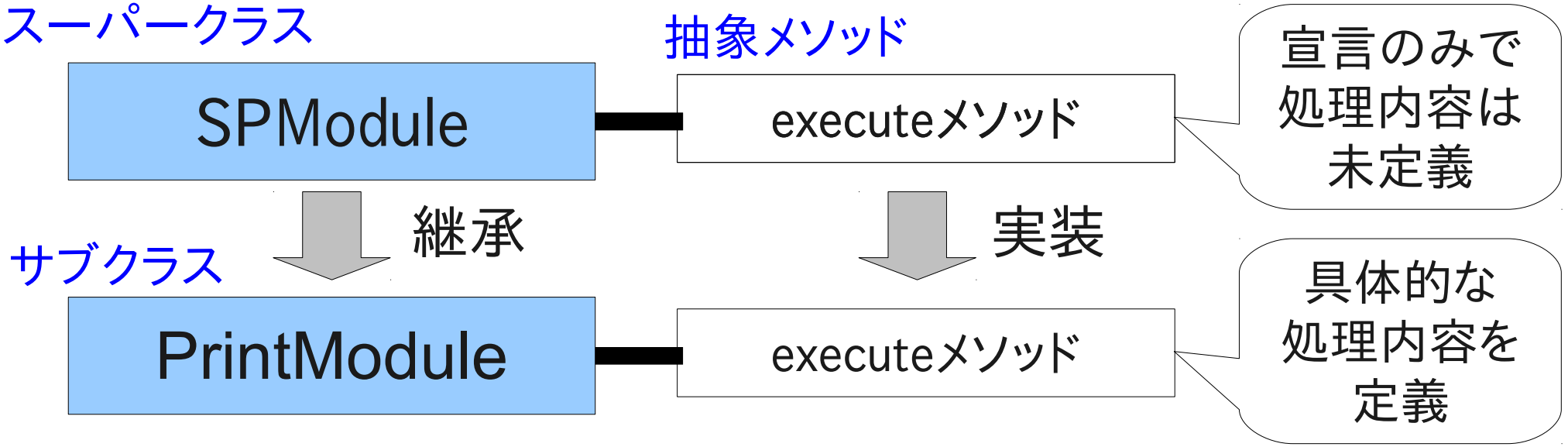
```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.amusaj.sp.*;

CMXController cmx = CMXController.getInstance();

void setup() {
  MidiInputModule vk = cmx.createVirtualKeyboard(this);
  MidiOutputModule mo = cmx.createMidiOut();
  cmx.addSPModule(vk);
  cmx.addSPModule(mo);
  cmx.connect(vk, 0, mo, 0);
  cmx.startSP();
}

void draw() {
}
```

Step 2 独自モジュールを作成する

MIDIメッセージを受け取って画面表示する
モジュール「PrintModule」を作成してみよう

基本的な考え方

SPModuleクラスを継承し、必要なメソッドを実装



SPModuleのサブクラスを作成するには，次の3つのメソッドを実装する

```
class PrintModule extends SPModule {
  void execute(Object[] src, TimeSeriesCompatible[] dest) {
        ここにモジュールがすべき処理内容を記述する
  }

  Class[] getInputClasses() {
        ここに、このモジュールが受け付けるオブジェクトのクラス名を書く
  }

  Class[] getOutputClasses() {
        ここに、このモジュールが出力するオブジェクトのクラス名を書く
  }
}
```

PrintModuleの基本方針

- PrintModuleは，仮想鍵盤の演奏データを受け取り，画面表示し，そのまま出力する
- 演奏データは，MIDIEventWithTicktimeクラスとして扱う

  ⇨ src[0]がMIDIEventWithTicktimeオブジェクトなので，必要なメソッドを使用したのち，dest[0]にそのままadd

```
class PrintModule extends SPModule {
  void execute(Object[] src, TimeSeriesCompatible[] dest) {
  }
```

各入力チャンネルから得られたデータがここにある

dest[チャンネル番号].add(出力データ)とすればデータが出力される

今回は、src[0]が入力されたMIDIデータ

今回は、dest[0].add(MIDIデータ)とすればよい

```
import jp.crestmuse.cmx.processing.*
import jp.crestmuse.cmx.amusaj.sp.*
import jp.crestmuse.cmx.sound.*

class MyApplet extends CMXApplet {
  class PrintModule extends SPModule {
    void execute(Object[] src, TimeSeriesCompatible[] dest) {
      // src[0]からステータスバイトとデータバイトを取得
      def (status, data1, data2) =
                          src[0].getMessageInByteArray()
      // 取得したステータスバイトとデータバイトを画面表示
      println(status + " " + data1 + " " + data2)
      // 入力されたデータをそのまま出力
      dest[0].add(src[0])
    }

    Class[] getInputClasses() {
      [MidiEventWithTicktime.class]
    }

    Class[] getOutputClasses() {
      [MidiEventWithTicktime.class]
    }
  }
```

（続き）

```
  void setup() {
    def vk = createVirtualKeyboard()
    def pm = new PrintModule()
    def mo = createMidiOut()
    addSPModule(vk)
    addSPModule(pm)
    addSPModule(mo)
    connect(vk, 0, pm, 0)
    connect(pm, 0, mo, 0)
  }

  void draw() {

  }
}

MyApplet.start("MyApplet")
```

```
import jp.crestmuse.cmx.processing.*;
import jp.crestmuse.cmx.amusaj.sp.*;
import jp.crestmuse.cmx.sound.*;

class PrintModule extends SPModule {
    void execute(Object[] src, TimeSeriesCompatible[] dest)
      throws InterruptedException {
      // src[0]はMidiEventWithTicktimeオブジェクトなのでキャスト
      MidiEventWithTicktime midievt = (MidiEventWithTicktime)src[0];
      // src[0]からステータスバイトとデータバイトを取得
      byte[] data = midievt.getMessageInByteArray();
      // 取得したステータスバイトとデータバイトを画面表示
      println(data[0] + " " + data[1] + " " + data[2]);
      // 入力されたデータをそのまま出力
      dest[0].add(midievt);
    }

    Class[] getInputClasses() {
      return new Class[]{MidiEventWithTicktime.class};
    }

    Class[] getOutputClasses() {
      return new Class[]{MidiEventWithTicktime.class};
    }
}
```

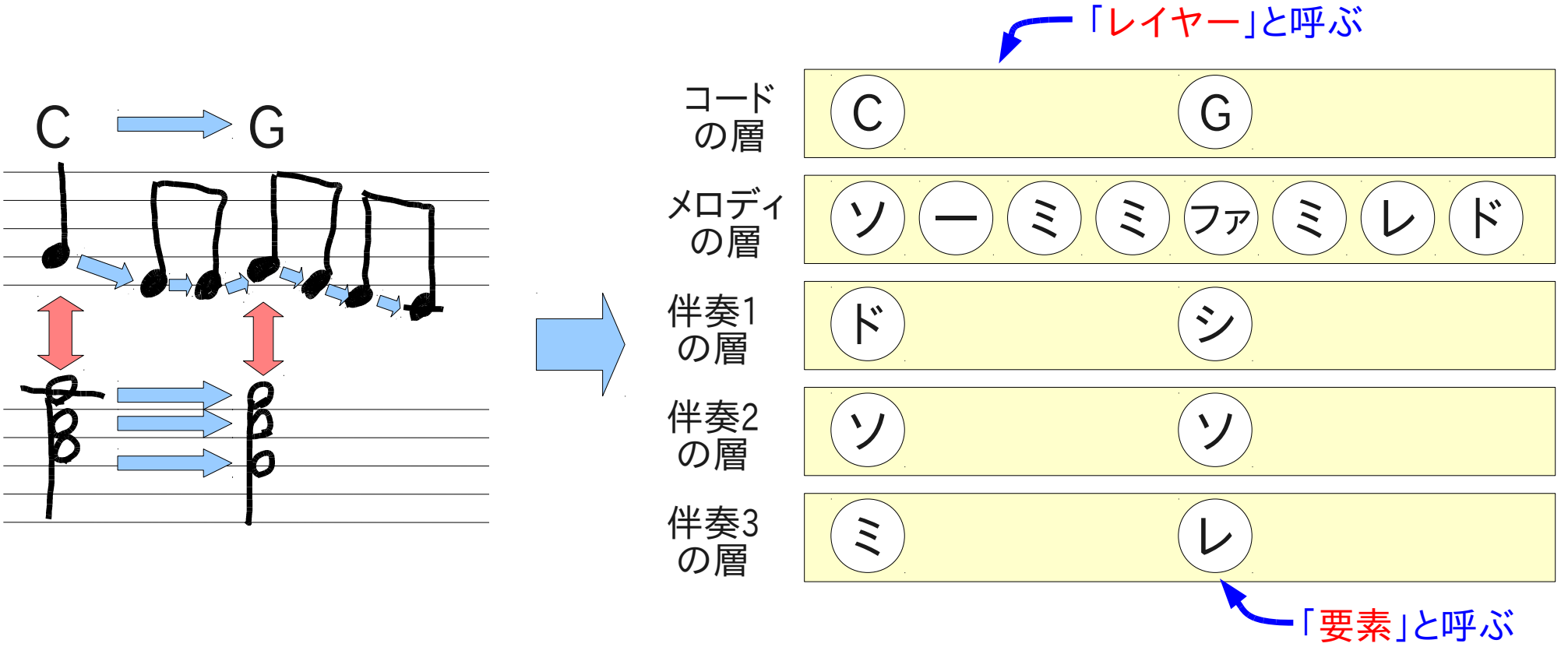
（続き）

```
CMXController cmx = CMXController.getInstance();

void setup() {
    MidiInputModule vk = cmx.createVirtualKeyboard(this);
    PrintModule pm = new PrintModule();
    MidiOutputModule mo = cmx.createMidiOut();
    cmx.addSPModule(vk);
    cmx.addSPModule(pm);
    cmx.addSPModule(mo);
    cmx.connect(vk, 0, pm, 0);
    cmx.connect(pm, 0, mo, 0);
    cmx.startSP();
}

void draw() {

}
```

Lesson 4　音楽データの確率推論のためのデータ構造「MusicRepresentation」を使う

基本的な考え方

音楽は，時間軸に沿って一列に並んだデータ列が，何層に渡って出来ていると考えるとわかりやすい．



- 音楽は，1つ以上のレイヤーからなり，各レイヤーは1つ以上の要素からなる．
- 各要素には，音名やコード名などのラベルをセットできる．
- 要素にセットできるラベルは，レイヤーごとに決まっている．
  （コードレイヤーならコード名，メロディレイヤーなら音名）
- 1小節に何個の要素があるのかはレイヤーによって異なる．
  （メロディレイヤーなら8個，コードレイヤーなら2個のように）

この条件を満たすようにデータ構造を構築して音楽データを表す．
さらに確率推論ができるように，次の条件を追加する．

- 各要素は，各ラベルをとりうる確率を保持する．
  （例：p(〃C〃)=0.2, p(〃C#〃)=0.05, p(〃D〃)=0.1, p(〃D#〃)=0.05, ...）
- 各要素に，あるラベルの取りうる確率を1とし，それ以外を
  取りうる確率を0にすることを，エビデンスをセットするという．

MusicRepresentationインタフェース

前述のデータ構造を実現するものとして，MusicRepresentationというインタフェースが用意されているので，これを利用する．

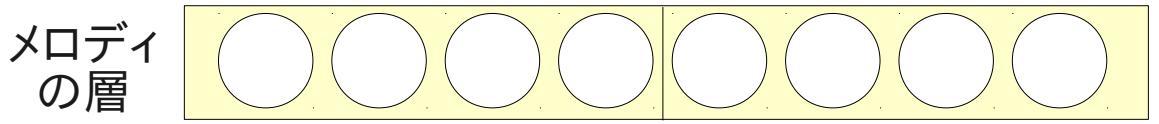
では，さっそくMusicRepresentationのインスタンスを用意しよう．

```
import jp.crestmuse.cmx.processing.*
import jp.crestmuse.cmx.inference.*

def cmx = CMXController.getInstance()
def mr = cmx.createMusicRepresentation(2, 4)
```

小節数

1小節に何個要素を作るか

この段階では，レイヤーが1つもない空の状態なので，メロディレイヤーを追加しよう．




```
def notenames = ["C", "C#", "D", "D#", "E", "F",
                 "F#", "G", "G#", "A", "A#", "B"]
mr.addMusicLayer("melody", notenames)
```

レイヤーの名前を文字列で指定

そのレイヤーの各要素がとりうる値の一覧を文字列の配列で指定

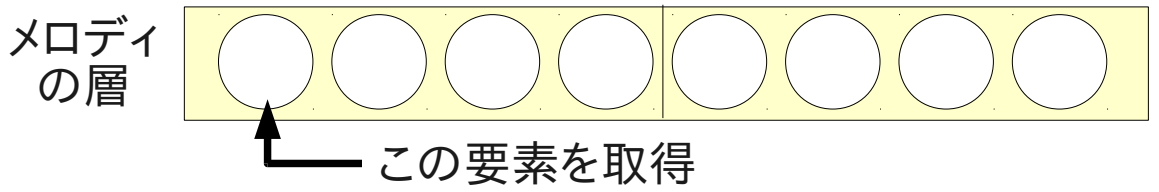
MusicElementインタフェース

次に，今追加してメロディレイヤーの各要素をいろいろいじってみよう．

各要素はMusicElementインタフェースのオブジェクトとして扱われる．

```
def e0 = mr.getMusicElement("melody", 0, 0)
```

melodyレイヤー，0小節め，
0番めの要素を取得



この要素に「ソ」をエビデンスとしてセットしてみよう．

エビデンスをセットするには，setEvidenceメソッドを使う．

```
e0.setEvidence("G")
```

この要素が各値を取る確率を出力してみよう．〝G〟にエビデンスをセットしたのだから，p(〝G〟)=1，p(それ以外)=0 になるはずだ．

```
notenames.each { x ->
  println("p(" + x + ")=" + e0.getProb(x))
}
```

今度は，その次の要素の各ラベルに確率をセットしてみよう．

```
def e1 = mr.getMusicElement("melody", 0, 1)
e1.setProb("A", 0.6)
e1.setProb("F", 0.2)
e1.setProb("G", 0.1)
```

確率が最も高いラベルは，getMostLikelyメソッドで取得できる．

```
println(e1.getMostLikely())
```

generateメソッドを使うと，セットされている確率分布に従って，
ランダムにラベルを出力する．

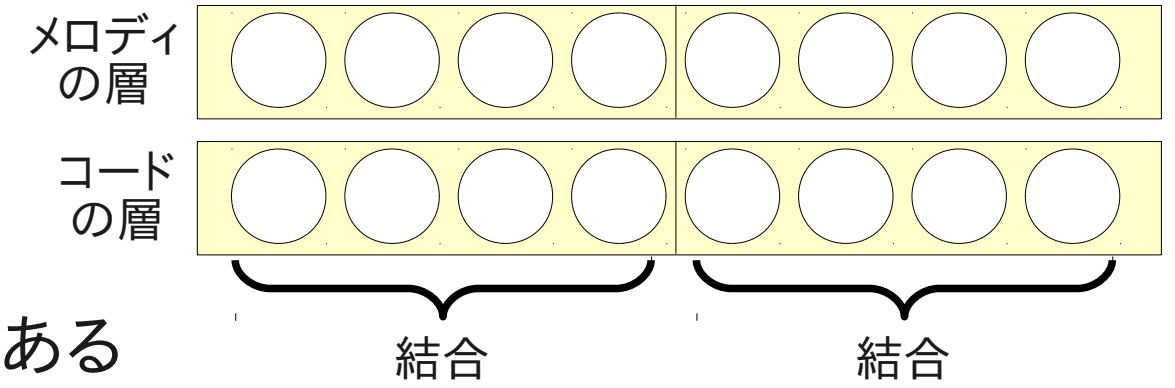
```
20.times {
  println(e1.generate())
}
```

レイヤーを増やす

では次に，コード進行を表すレイヤーを追加しよう．

```
String[] chordnames = ["C", "Dm", "Em", "F", "G", "Am"]
mr.addMusicLayer("chord", chordnames)
```

ただし，これではコードレイヤーも2×4要素できてしまう．
コードは1小節に1個にするなら，要素を結合する必要がある
（Excelの「セル結合」みたいなもの）



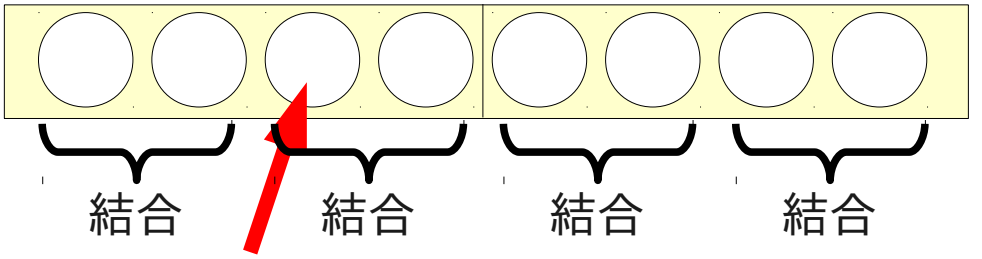

```
（上のmr.addMusicLayer(…)の代わりに）
mr.addMusicLayer("chord", chordnames, 4)
```

何個ずつ結合するかを指定

注意

要素を結合した場合でも，
「何番めの要素か」は
結合前の番号で表す

例：2個ずつ結合した場合



mr.getMusicElement(″chord″, 2)

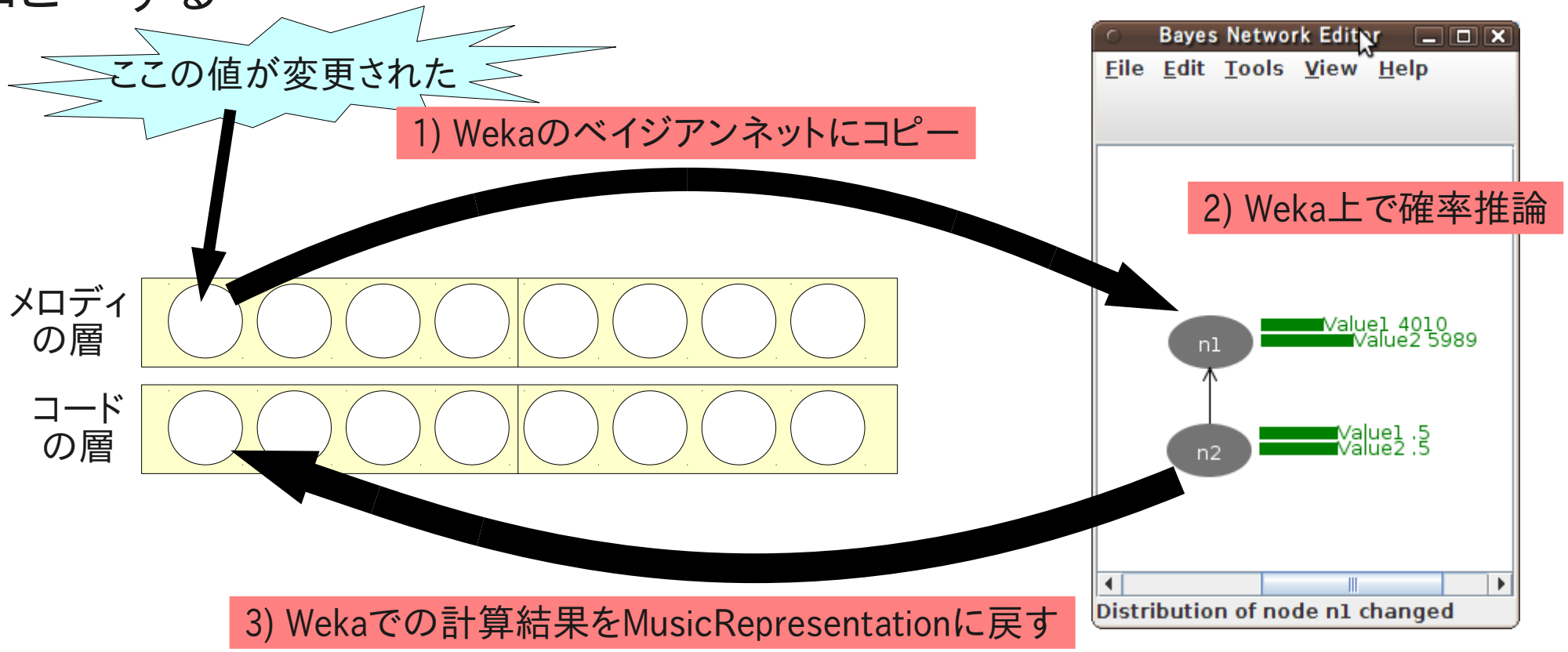
コードレイヤーについても，setEvidence, setProb, getMostLikely, generateを試してみよう.

```
def e2 = mr.getMusicElement("chord", 0, 0)
e2.setEvidence("C")
chordnames.each { x ->
  println("p(" + x + ")=" + e2.getProb(x))
}

def e3 = mr.getMusicElement("chord", 1, 0)
e3.setProb("C", 0.2)
e3.setProb("F", 0.4)
e3.setProb("G", 0.3)
e3.setProb("Am", 0.1)
20.times {
  println(e3.generate())
}
```

Lesson 5　データマイニングツール「Weka」で構築したベイジアンネットワークを使う

基本的な考え方

MusicRepresentation上の状態をWekaのベイジアンネットにコピーして，Weka上で確率推論を行って，その結果を再びMusicRepresentationにコピーする



上の例では，メロディに関する情報をWeka上では「n1」が，
コードに関する情報を「n2」が表していることを前提としている．

基本的な手順

1. Weka上でベイジアンネットを作成して，ファイルに保存
   - 作成する際には，個々のノードが MusicRepresentation のどのレイヤーに対応するのかを考えること
   - 各ノードのとりうる値の個数と，MusicRepresentation 上で対応するレイヤーでのとりうる値の個数が一致すること
2. Wekaへのコピーなどを自動的に行ってくれる「BayesianCalculator」オブジェクトを作成
3. WekaとMusicRepresentationの対応を示す「BayesianMapping」オブジェクトをBayesianCalculatorに登録
4. BayesianCalcultorをMusicRepresentationに登録
5. あとは，MusicRepresentationからMusicElementを取得してsetEvidenceなどすれば，自動的に推論してくれる

0. MusicRepresentationオブジェクトを作成

Lesson 4を参考に，MusicRepresentationオブジェクトを作成しよう．

```
import jp.crestmuse.cmx.processing.*
import jp.crestmuse.cmx.inference.*

def cmx = CMXController.getInstance()
def mr = cmx.createMusicRepresentation(1, 4)
```

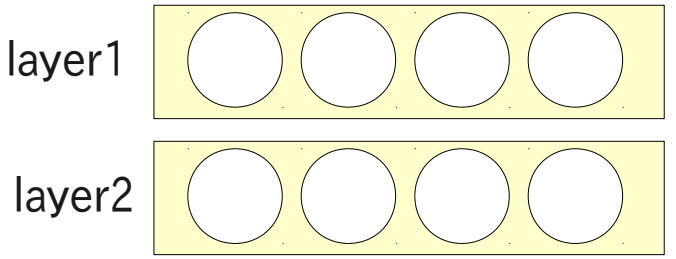
次に，レイヤーを2つ作ってみよう．

ここでは，名前を「layer1」，「layer2」とする（名前は何でもよい）．

また，layer1は〝A〟，〝B〟の2つの値を，layer2は〝X〟，〝Y〟の2つの値をとりうるものとする．

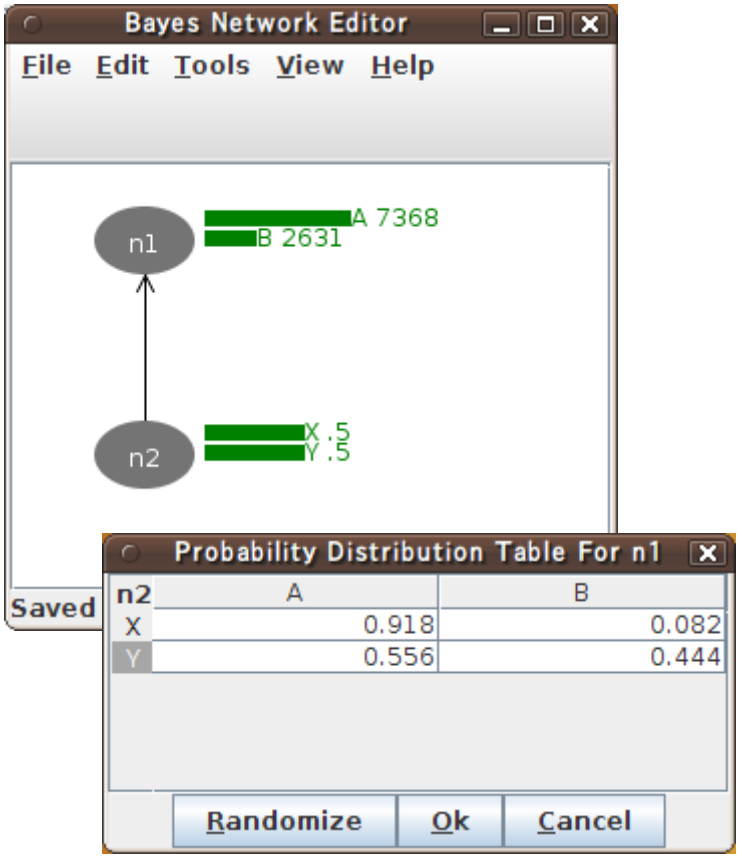
```
def values1 = ["A", "B"]
def values2 = ["X", "Y"]
mr.addMusicLayer("layer1", values1)
mr.addMusicLayer("layer2", values2)
```

ここまではLesson 4の内容



1. Wekaでベイジアンネットを作る

Wekaで適当なベイジアンネットを作って，BIFXML形式で保存しよう.

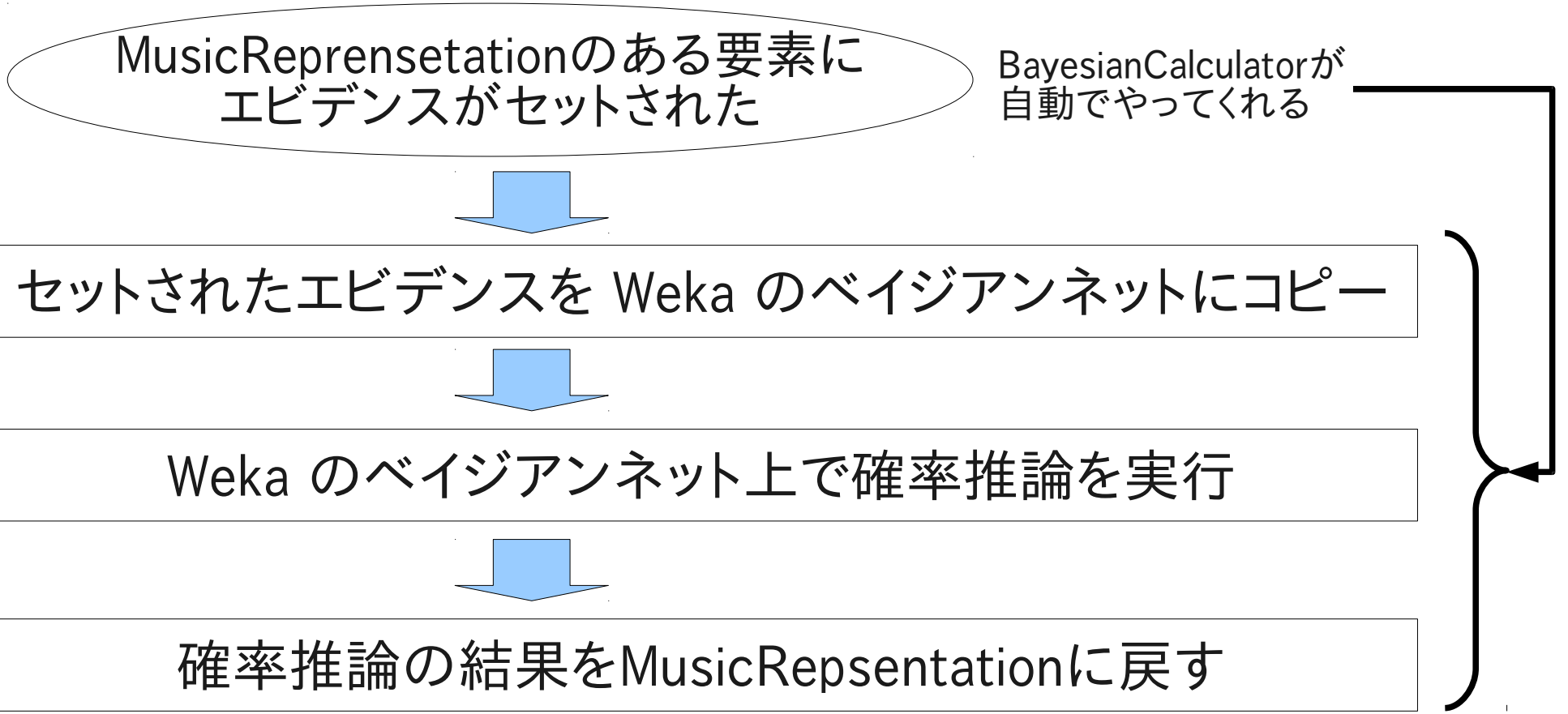


ここでは，「layer1」の値を「n1」にコピーして
Wekaの確率推論機能で求めた「n2」の値を
再び「layer2」にコピーする，というのを想定

注意点

- 先ほど作ったMusicRepresentationどのレイヤーが，どのノードに対応するかを考えながら作ること.
  - ここでは，「layer1」が「n1」に，「layer2」が「n2」に対応している.
- 各ノードがとりうる値は，MusicRepresentation側の対応するレイヤーのとりうる値に合わせること
  - 「layer1」と「n1」のとりうる値が〃A〃，〃B〃
  - 「layer2」と「n2」のとりうる値が〃X〃，〃Y〃

2. BayesianCalculatorオブジェクトを作成




```
/* Wekaで保存したベイジアンネットのファイルを読み込む部分 */
def bn = new BayesNetWrapper("mybn.xml")

/* BayesianCalculatorオブジェクトの作成(読み込んだベイジアンネットを指定する) */
def bc = new BayesianCalculator(bn)
```

jp.crestmuse.cmx.filewrappers.* をインポートする必要あり

3. BayesianMappingオブジェクトを作成

MusicRepresentation と Weka のベイジアンネットの対応関係を定義

BayesianMappingオブジェクトの作り方：

new BayesianMapping(レイヤー名, 0, 0, ノード名, bn)

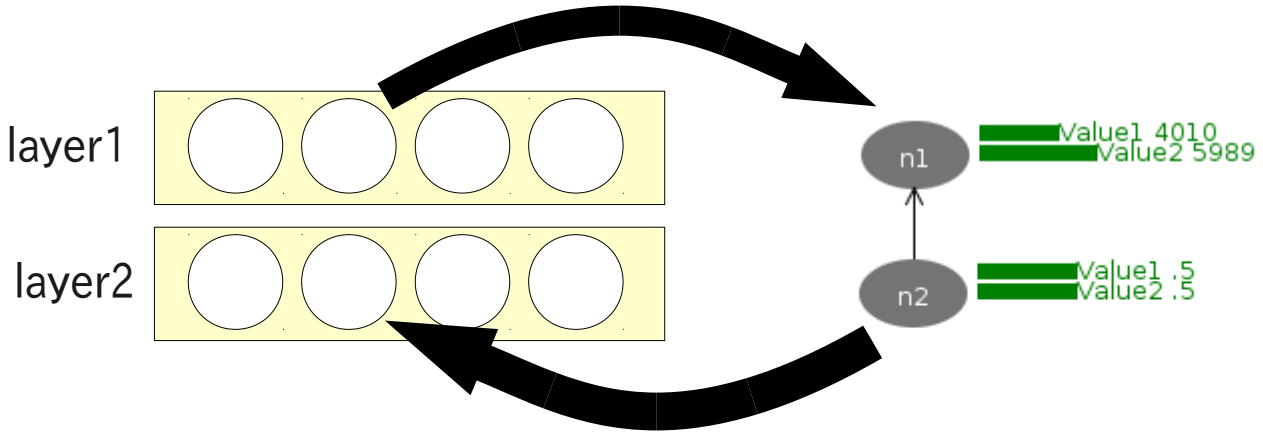
MusicRepresentation側　　Weka側　　さきほど読み込んだベイジアンネット

例　layer1 の i 番めの要素にエビデンスがセットされた場合

```
new BayesianMapping("layer1", 0, 0, "n1", bn)
```

layer1 の i 番めの要素が 右側の n1 に対応

addReadMappingメソッドで BayesianCalculatorに登録



addWriteMappingメソッドで BayesianCalculatorに登録

layer2 の i 番めの要素が 右側の n2 に対応

```
new BayesianMapping("layer2", 0, 0, "n2", bn)
```

```
bc.addReadMapping(new BayesianMapping("layer1", 0, 0, "n1", bn))
bc.addWriteMapping(new BayesianMapping("layer2", 0, 0, "n2", bn))
```

4. BayesianCalculatorオブジェクトを登録

後は，BayesianCalculatorをMusicRepresentationに登録すれば準備完了

```
mr.addMusicCalculator("layer1", bc)
```

5. MusicRepresentationにエビデンスをセット

- レイヤー「layer1」の最初の要素に〝A〟をエビデンスとしてセット
  - レイヤー「layer2」の最初の要素が自動的に更新されればOK

```
def e1 = mr.getMusicElement("layer1", 0, 0)
e1.setEvidence("A")

def e2 = mr.getMusicElement("layer2", 0, 0)
println(e2.getProb("X"))
println(e2.getProb("Y"))
```

- レイヤー「layer1」の2番めの要素に〝B〟をエビデンスとしてセット
  - レイヤー「layer2」の2番めの要素が自動的に更新されればOK

```
def e3 = mr.getMusicElement("layer1", 0, 1)
e3.setEvidence("B")

def e4 = mr.getMusicElement("layer2", 0, 1)
println(e4.getProb("X"))
println(e4.getProb("Y"))
```

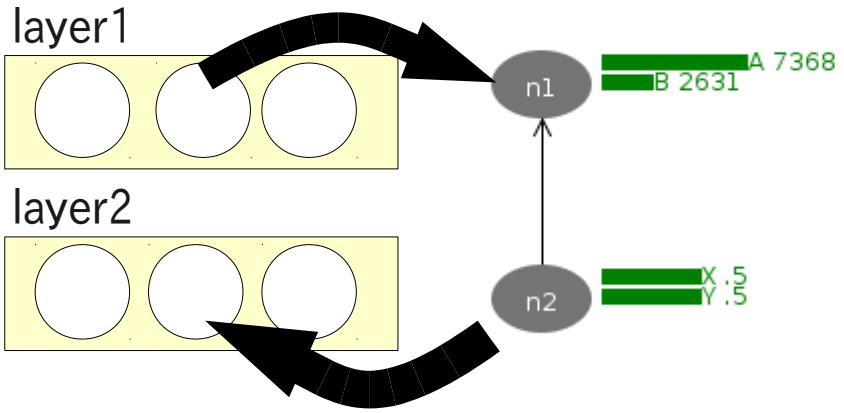
プログラムリスト 完全版

```
import jp.crestmuse.cmx.processing.*
import jp.crestmuse.cmx.inference.*
import jp.crestmuse.cmx.filewrappers.*

def cmx = CMXController.getInstance()
def mr = cmx.createMusicRepresentation(1, 4)
def values1 = ["A", "B"]
def values2 = ["X", "Y"]
mr.addMusicLayer("layer1", values1)
mr.addMusicLayer("layer2", values2)

def bn = new BayesNetWrapper("mybn.xml")
def bc = new BayesianCalculator(bn)
bc.addReadMapping(new BayesianMapping("layer1", 0, 0, "n1", bn))
bc.addWriteMapping(new BayesianMapping("layer2", 0, 0, "n2", bn))
mr.addMusicCalculator("layer1", bc)

def e1 = mr.getMusicElement("layer1", 0, 0)
e1.setEvidence("A")

def e2 = mr.getMusicElement("layer2", 0, 0)
println(e2.getProb("X"))
println(e2.getProb("Y"))

def e3 = mr.getMusicElement("layer1", 0, 1)
e3.setEvidence("B")

def e4 = mr.getMusicElement("layer2", 0, 1)
println(e4.getProb("X"))
println(e4.getProb("Y"))
```

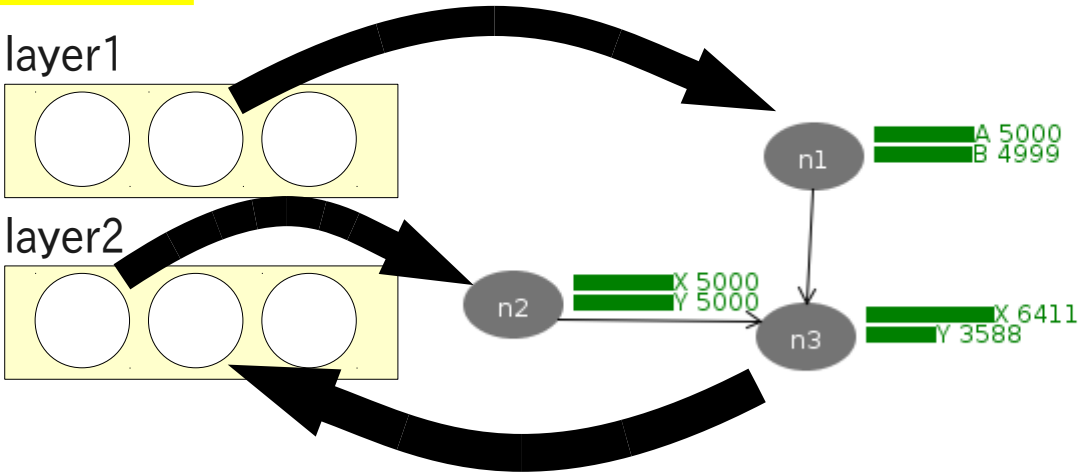
次のステップ

- さきほどは，レイヤーが2つあるMusicRepresentationを対象に，片方のレイヤーの要素をWeka側にコピーして，確率推論後に，推論結果をもう片方のレイヤーの同時刻の要素にコピーした．
- 今度は，1つ前の時刻の要素も利用してみよう．
- たとえば，n番めのコードを決めるときに，その時刻のメロディに依存するけど，直前（n-1番め）のコードにも依存するはず．

さっきまで



今度は



改造のポイントはBayesianMapping

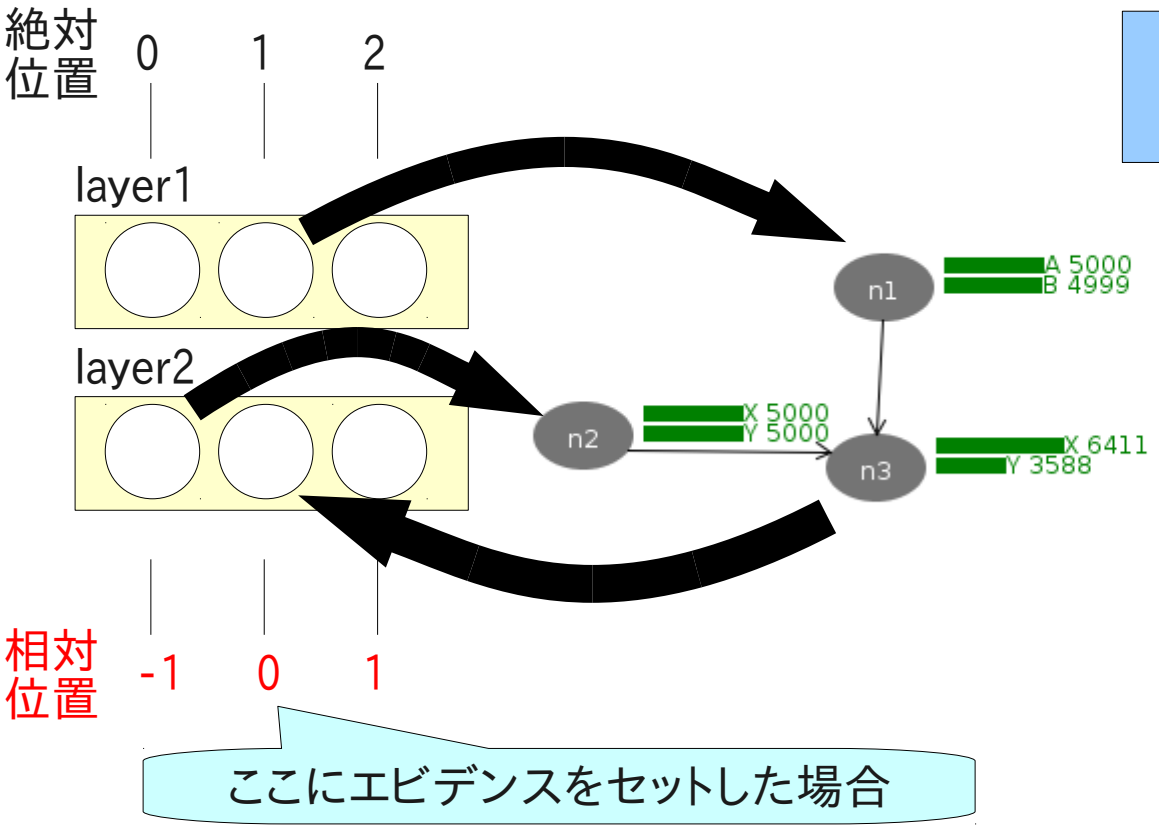
基本的には，さきほどのプログラムを改造すればよい．

改造のポイントは，BayesianMapping．

new BayesianMapping(レイヤー名, 相対位置, 0, ノード名, bn)

さっきは常に0だった

「相対位置」というのは・・・



相対位置：エビデンスをセットした
指定した位置が基準(0)となる

だから・・・

```
bc.addReadMapping(
  new BayesianMapping(
  "layer1", 0, 0, "n1", bn))
bc.addReadMapping(
  new BayesianMapping(
  "layer2", -1, 0, "n2", bn))
bc.addWriteMapping(
  new BayesianMapping(
  "layer2", 0, 0, "n3", bn))
```

おわりに

この説明書で扱わなかったこと

- MusicXML, DeviationInstanceXML, SCCXML以外のXML
  - 旋律の階層構造を記述するMusicApexXMLなどがある
- 外部のMIDIデバイスの利用
  - 外部のMIDI音源やMIDIキーボードを簡単に選ぶことができる
- MIDIファイルの再生とリアルタイムMIDI処理の連携
  - MIDIキーボードが打鍵される度に，MIDIファイルの再生位置を取得して処理を切り替えたり，様々なことができる
- 音響信号処理
  - WAVファイルやマイクからの入力音声に対してフーリエ変換などをすることができます
- 行列計算
  - Groovyの演算子オーバーロードと組み合わせると便利

お願い

- 現時点では十分なドキュメント作成ができていません．
  ご不明な点は，ぜひ遠慮せずに聞いてください．
- また，動作検証も十分にできていません．バグらしき現象がありましたら，ご報告いただければ幸いです．
- 共同開発に興味のある方は，ぜひ連絡ください．コーディングでなくても，テスト，ドキュメント作成などでも大歓迎です．

お問い合わせ先

- Web: http://cmx.sourceforge.jp/
  http://sourceforge.jp/projects/cmx/
- メール： kitahara [at] kthrlab.jp