Programmation et Conception Orientées Objet
TD n° 2 : Synthèse musicale

Contexte

Nous allons créer un programme qui permet de générer des fichiers sonores à partir de fichiers de partitions de musique de manière très simplifiée : en prenant en entrée un fichier MIDI, notre programme doit générer un fichier WAV, dans lequel la partition est interprétée par des instruments simples.

Les fichiers MIDI

Un fichier MIDI est une "partition musicale" résumée dans un fichier binaire, pour être décodée par un synthétiseur. Son extension est `.mid` ou `.midi`. Ce fichier ne contient aucun son, mais des instructions :

- **`"0.1 seconde : commencer de jouer la note do"`**
- **`"0.2 seconde : arrêter de jouer la note do"`**
- **`"0.25 seconde : commencer de jouer la note ré"`**
- **`"0.35 seconde : arrêter de jouer la note ré"`**
- **`"Etc."`**

Plus précisément, un fichier MIDI possède des messages de différents types :

- `MetaMessage` : Donner des instructions à un synthétiseur. Par exemple : "À partir de maintenant, la musique se joue à tel rythme".
- `ShortMessage` : Commencer/arrêter de jouer une note dans un canal (Un canal sert à transmettre des instructions vers un instrument virtuel, comme un piano, un violon ou un saxophone).

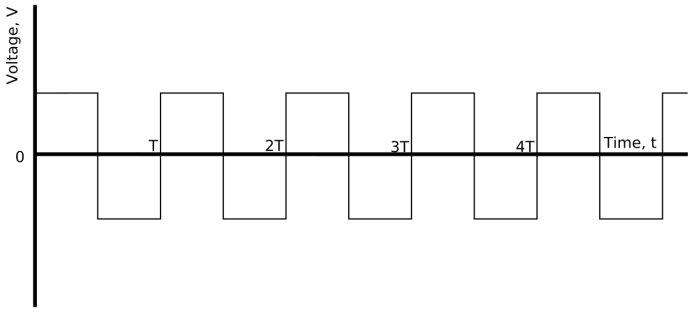
Les fichiers MIDI ont deux types principaux :

- `Type 0` : les messages sont mélangés ensemble, puis redirigés vers différents instruments.
- `Type 1` : les messages sont organisés en pistes, correspondant aux canaux des instruments.

Nous allons uniquement nous concentrer sur des fichiers MIDI de Type 1, les plus courants.

Les fichiers WAV

Un fichier WAV contient une liste d'entiers correspondant au son d'un morceau de musique. Par exemple, un fichier WAV peut contenir une liste d'entier permettant de générer le signal carré suivant :



Le package `midi` de Java

Java permet de lire des traiter des fichiers MIDI grâce au package `javax.sound.midi`. Ce package permet de gérer les `MetaMessage` et les `ShortMessage` de manière très basique ... Cependant, les classes fournies sont très bas-niveau, c'est-à-dire qu'elles sont plutôt fastidieuses à utiliser pour un développeur.

Transcrire un message en note de musique

Par exemple, voici l'algorithme qui permet de récupérer les informations importantes d'un `ShortMessage` :

```
MidiEvent event; // Un événement MIDI qui contient un ShortMessage
ShortMessage sm = (ShortMessage) event.getMessage(); // Le message MIDI

int channel = sm.getChannel(); // Le canal de la note
long tick = event.getTick(); // L'instant auquel la note est jouée
int key = sm.getData1(); // La note en partant du bas du piano
int octave = (key / 12)-1; // L'octave de la note
int note = key % 12; // La note elle-même : 0 pour do, 1 pour do#, 2
   pour ré, etc.
int volume = sm.getData2(); // Le volume de la note, variant entre 0
   (note silencieuse) et 127 (maximum)
int cmd = sm.getCommand(); // cmd == 0x90, signifie que c'est un début
   de note ; cmd == 0x80 signifie que c'est une fin de note
```

Travailler avec le temps

Autre exemple, pour récupérer le tempo d'un morceau MIDI, il faut parcourir les messages jusqu'à tomber sur un `MetaMessage` avec certaines valeurs et effectuer un calcul fastidieux :

```
int tempo = 120; // Un tempo de 120 peut être une valeur par défaut

MidiEvent event; // Un événement MIDI qui contient un MetaMessage

MetaMessage mm = (MetaMessage) event.getMessage(); // Le message MIDI
byte[] msg = mm.getMessage(); // Le contenu du message
if (((msg[1] & 0xFF) == 0x51) && (msg[2] == 0x03)) {
  int mspq = (msg[5] & 0xFF)
    | ((msg[4] & 0xFF) << 8)
    | ((msg[3] & 0xFF) << 16); // Une manipulation fastidieuse
  tempo = Math.round(60000001f / mspq); // Le tempo est calculé
}
```

Un `tick` est un instant dans la partition. En fonction de plusieurs paramètres, comme le tempo de la musique et la résolution du fichier (sa "précision"), il peut être converti en secondes, comme ceci :

```
Sequence sequence = MidiSystem.getSequence(new File("mon_fichier.mid"));
long resolution = sequence.getResolution();

float time = (((float) tick) * 60 / (4 * tempo * resolution));
```

Écrire des fichiers WAV

Nous pouvons créer des fichiers WAV grâce aux classes du package `javax.sound.sampled`. Nous utiliserons les conventions suivantes, qui nous permettront de simplifier la création des fichiers :

— Fréquence d'échantillonnage (sample rate) : 44100 Hz
— Taille des messages (sample size) : 8 bits
— Nombre de canaux (channels) : 1 (son mono)
— Entiers signés, en Big Endian (pour faciliter le traitement du signal)

Cela nous permettra de créer un fichier comportant un signal généré à partir des partitions.

Nous utilisons la convention suivante :

— Si la note de la partition MIDI est envoyée vers le canal 9, il s'agit d'une percussion, représentée par un bruit blanc (un signal aléatoire).
— Si la note de la partition MIDI est envoyée vers un autre canal, il s'agit d'un signal carré, joué à une fréquence correspondant à la note et à son octave.

Pour calculer la fréquence d'une note, il suffit de prendre les références suivantes pour l'octave 4 :

Note	Fréquence (Hz)
Si	493.883
La#	466.164
La	440.000
Sol#	415.305
Sol	391.995
Fa#	369.994
Fa	349.228
Mi	329.628
Ré#	311.127
Ré	293.665
Do#	277.183
Do	261.626

Les fréquences de l'octave 3 sont obtenues en divisant celles de l'octave 4 par 2, et ainsi de suite. Les fréquences de l'octave 5 sont obtenues en multipliant celles de l'octave 4 par 2, et ainsi de suite. De manière générale :

```
float frequence = frequence_ref * (float) Math.pow(2, octave - 4);
```

Les valeurs du tableau d'entiers d'un fichier WAV varient entre -128 et 127.

Pour générer un signal carré oscillant entre -128 et 127 à une certaine fréquence pendant une seconde, nous pouvons utiliser la formule suivante :

```
int[] signal = new int[44100]; // stocker 441000 échantillons, soit une
   seconde de son
for (int i = 0; i < 44100; i++) {
  if ((4*i % (44100f/frequence)) > 22050f/frequence) {
    signal[i] = 127;
  } else {
    signal[i] = -128;
  }
}
```

Pour générer un bruit blanc pendant une seconde, nous pouvons utiliser la formule suivante :

```
import java.util.Random; // Au début du fichier

int[] signal = new int[44100]; // stocker 441000 échantillons, soit une
    seconde de son
for (int i = 0; i < 44100; i++) {
  buffer[i] = (new Random()).nextInt(-128, 128); // la borne supérieure
      est exclue
}
```

Il faut ensuite enregistrer ces valeurs dans un tableau d'octets, qui contient les valeurs transformées en 4 octets, les uns à la suite des autres :

```
import java.nio.ByteBuffer;    // Au début du fichier

byte[] bytes = new byte[8]; // On va stocker 2 entiers, soit 8 octets
ByteBuffer bb = ByteBuffer.allocate(4);
bb.putInt(42); // On transforme notre entier 42 grâce à un ByteBuffer
bytes[0] = bb.get(0);
bytes[1] = bb.get(1);
bytes[2] = bb.get(2);
bytes[3] = bb.get(3); // On stocke ces octets dans un tableau d'octets
bb = ByteBuffer.allocate(4);
bb.putInt(-42); // On transforme notre entier -42 grâce à un ByteBuffer
bytes[4] = bb.get(0);
bytes[5] = bb.get(1);
bytes[6] = bb.get(2);
bytes[7] = bb.get(3); // On stocke ces octets dans un tableau d'octets
```

Une fois que nous avons ce tableau d'octets, nous pouvons les stocker dans un fichier WAV comme ceci :

```
import javax.sound.sampled.*; // Au début du fichier
import java.io.*;             // Au début du fichier

AudioFormat format = new AudioFormat(44100, 8, 1, true, true);
AudioInputStream ais = new AudioInputStream(new
   ByteArrayInputStream(bytes), format, bytes.length);
File file = new File("mon_fichier.wav");
AudioSystem.write(ais, AudioFileFormat.Type.WAVE, file);
```

Conception et Implémentation

Concevez en UML, puis implémentez une couche logicielle par-dessus l'API MIDI fournie par Java, pour manipuler vos messages MIDI de manière plus naturelle et proches de la Programmation et Conception Orientées Objet.

À partir de cette couche logicielle, concevez un synthétiseur qui génère le signal résultant de cette partition et le stocke dans un fichier WAV, à partir de l'addition de signaux carrés et de bruits blancs placés aux bons endroits.

Pour réaliser ce logiciel, vous pouvez vous documenter via l'API MIDI de Java. Vous pourrez mettre à jour votre code pour l'optimiser grâce aux notions que nous verrons aux cours 5 et 6.