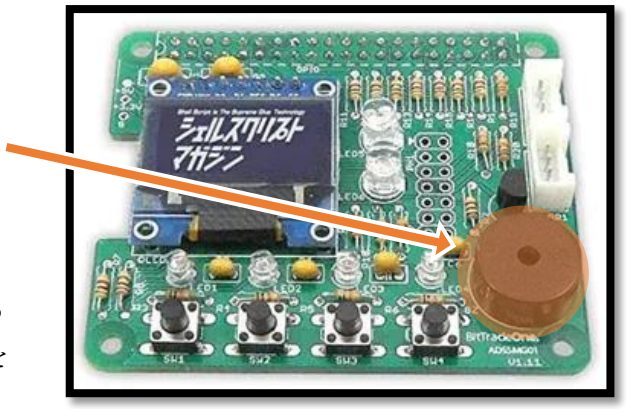
圧電ブザーとは



圧電ブザーとは、圧電振動板を共鳴器に組み込んだ発音部品のこと。
電圧を加える事で接着している圧電セラミックスが伸縮し、金属板が振動することで音波が発生する。
入門ボードには 1 個搭載されており、接続されているピンにハードウェア PWM 出力をすることで、音色を奏でることができる。

ハードウェア PWM 出力とは

圧電ブザーを制御する方法として、今回はハードウェア PWM 出力を使用する。
前回も PWM 出力を使用してフルカラーLED を制御したが、正確にはソフトウェア PWM 出力と言い、プログラムによって高速に電圧の High - Low を切り替えてデューティ比を調節していた。
ソフトウェア PWM は電圧の切り替えがプログラムに依存する為、CPU が別の処理をしていたりプログラムが終了すると正しく出力されない場合があり、出力の精度が低いというデメリットがある。

一方、今回使用するハードウェア PWM は、Raspberry Pi の CPU とは独立したクロックを使用して電圧の切り替えを行う為、CPU やプログラムに依存せず精度の高い出力を行うことができる。
こちらはハードウェアに依存している為、使用できるピンが限られるというデメリットがある。

圧電ブザーの GPIO 番号

今回使用する圧電ブザーの GPIO 番号は以下の通りです。

BUZZER = 1

※Raspberry Pi でハードウェア PWM 出力が可能なピンは 1, 23, 24, 26 番の 4 つのみ

表 5.1.1　Raspberry Pi の拡張コネクタ

備　考	機　能	ピン名	GPIO番号	ピン番号		ピン番号	GPIO番号	ピン名	機　能	備　考
50mAまで（1番ピンと17番ピンの合計）		3.3V		1		2		5V		
1.8kΩプルアップ抵抗付	I²C1（SDA）	GPIO2	8	3		4		5V		
1.8kΩプルアップ抵抗付	I²C1（SCL）	GPIO3	9	5		6		GND		
	GPCLK0	GPIO4	7	7		8	15	GPIO14	UART0（TXD）	起動時にシリアルコンソールとして使用
		GND		9		10	16	GPIO15	UART0（RXD）	起動時にシリアルコンソールとして使用
						12	1	GPIO18	SPI1（CS0）、PWM0	その他の機能：PCM_CLK
		GPIO27	2	13		14		GND		
		GPIO22	3	15		16	4	GPIO23		
50mAまで（1番ピンと17番ピンの合計）		3.3V		17		18	5	GPIO24		
	SPI0（MOSI）	GPIO10	12	19		20		GND		
	SPI0（MISO）	GPIO9	13	21		22	6	GPIO25		
	SPI0（SCLK）	GPIO11	14	23		24	10	GPIO8	SPI0（CS0）	
		GND		25		26	11	GPIO7	SPI0（CS1）	
拡張基板（Hat）に搭載のEEPROM用信号		ID_SD		27		28		ID_SC		拡張基板（Hat）に搭載のEEPROM用信号
		GPIO5	21	29		30		GND		
		GPIO6	22	31		32	26	GPIO12	PWM0	
	PWM1	GPIO13	23	33		34		GND		
その他の機能：PCM_FS	SPI1（MISO）、PWM1	GPIO19	24	35		36	27	GPIO16	SPI1（CS2）	
		GPIO26	25	37		38	28	GPIO20	SPI1（MOSI）	その他の機能：PCM_DIN
		GND		39		40	29	GPIO21	SPI1（SCLK）	その他の機能：PCM_DOUT

※Raspberry Pi タイプ B は 26 ピンまでになります。　　拡張コネクタのピンアサイン

練習（TryBuzzer.java）

以下のサンプルプログラムを記述して実行し、SW1 を押している間に LED1 が点灯しブザーからドの音が綺麗に奏でられ、SW1 を離したら LED1 が消灯しブザーが消音されるか確認しなさい。
※スイッチの状態の変化を判定しないと音の周波数が何度も設定され、綺麗に奏でることができない。
　今回のようにスイッチが押された時に一度だけ処理を行いたい場合はスイッチの状態を保存する。

```
/*
 * TryBuzzer.java
 * Date   : 2022/01/01
 * Author : IE1A 99 K.Murakami
 */

// GPIO ピンを利用するために必要なクラスを読み込む
import com.pi4j.wiringpi.Gpio;
import com.pi4j.wiringpi.GpioUtil;

public class TryBuzzer {
    // Thread.sleep メソッドで発生する割り込み例外を throws する
    public static void main (String[] args) throws InterruptedException {

        System.out.println("プログラム開始");

        // デジタル出力信号を定数化
        final int HIGH = 1;
        final int LOW = 0;
        // デジタル入力信号を定数化
        final int ON = 0;
        final int OFF = 1;

        // LED のピン番号を宣言
        final int LED1 = 15;
        // SW のピン番号を宣言
        final int SW1 = 7;

        // BUZZER のピン番号を宣言
        final int BUZZER = 1;
```

```
// 待ち時間の定数を定義（120 テンポ）
final int L1 = 2000;    //全音符
final int L2 = 1000;    //2 分音符
final int L4 =  500;    //4 分音符
final int L8 =  250;    //8 分音符
final int L16 = 125;    //16 分音符

// 使用する音の周波数を定義
final int C5 =  523;    //ド
final int D5 =  587;    //レ
final int E5 =  659;    //ミ
final int F5 =  698;    //ファ
final int G5 =  784;    //ソ
final int A5 =  880;    //ラ
final int NONE = 0;     //消音

// GPIO を初期化
Gpio.wiringPiSetup();
System.out.println("GPIO 初期化完了");

// 各 LED を出力に設定
Gpio.pinMode(LED1, Gpio.OUTPUT);
// 各 SW を入力に設定
Gpio.pinMode(SW1, Gpio.INPUT);
// BUZZER をハードウェア PWM 出力に設定
Gpio.pinMode(BUZZER,Gpio.PWM_OUTPUT);
// BUZZER 初期設定
Gpio.pwmSetMode (Gpio.PWM_MODE_MS); // MarkSpace モードに設定
Gpio.pwmSetRange(100);              //Range を百分率に設定
Gpio.pwmSetClock(NONE);             //消音
Gpio.pwmWrite(BUZZER,  50);         //綺麗に奏でる為デューティ比は 50%
System.out.println("GPIO 入出力設定完了");

// 各 SW の状態を保持する変数
int sw1Data;
// 前回の各 SW の状態を保持する変数（初期値は OFF）
int sw1DataOld = OFF;
```

```
        // プログラムを終了させない為に無限ループする
        while(true){
            // 現在の各 SW の状態を読み取る
            sw1Data = Gpio.digitalRead(SW1);

            // 現在の SW の状態が、前回の SW の状態と違う（変化があった）場合
            if(sw1Data != sw1DataOld){
                if(sw1Data == ON){
                    Gpio.digitalWrite(LED1, HIGH);  //LED 点灯
                    Gpio.pwmSetClock(192000 / C5);  //ドを発音
                }else{
                    Gpio.digitalWrite(LED1, LOW);   //LED 消灯
                    Gpio.pwmSetClock(NONE);         //消音
                }
            }

            // 次回の為に現在の各 SW の状態を保存する
            sw1DataOld = sw1Data;

            // 100 ミリ秒待機する
            Thread.sleep(100);
        }
    }
}
```

スイッチの状態を保持しておき、
前回（前のループ）の状態から
変化があったかを判定する

下記に解説あり

次回（次のループ）のスイッチの
状態の変化を判定する為に、
今回のスイッチの状態を保存する

◆解説

ブザーに発音させる場合、

Gpio.pwmSetClock();　の()内に目的の音の PWM 周波数を設定する必要がある。

PWM 周波数は【 ベースクロック / 音の周波数 / range 】という計算式で求められる。

Raspberry Pi の PWM が持つベースクロックは 19.2MHz であり、Range は百分率に設定している為、

Gpio.pwmSetClock(192000 / 音の周波数);　と記述すれば良い。

その為、サンプルプログラムのように**ド（523Hz）**を発音したい場合は

Gpio.pwmSetClock(192000 / **523**);　となり、更に音の周波数を定数に置き換えて

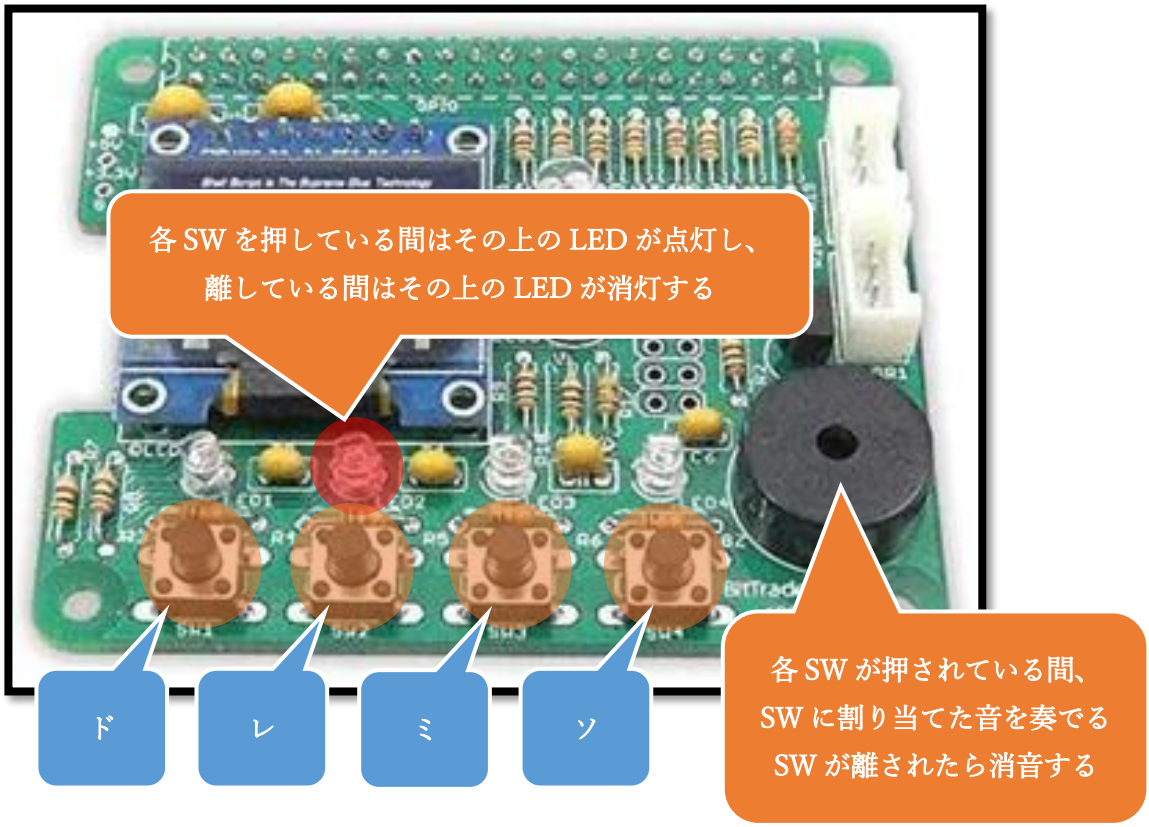
Gpio.pwmSetClock(192000 / **C5**);　と記述している。

※注意：消音する場合は Gpio.pwmSetClock (NONE); で、(192000 / NONE)とは記述しないように。
NONE＝0 なのでゼロ除算例外 (ArithmeticException) が発生してプログラムが強制終了する。

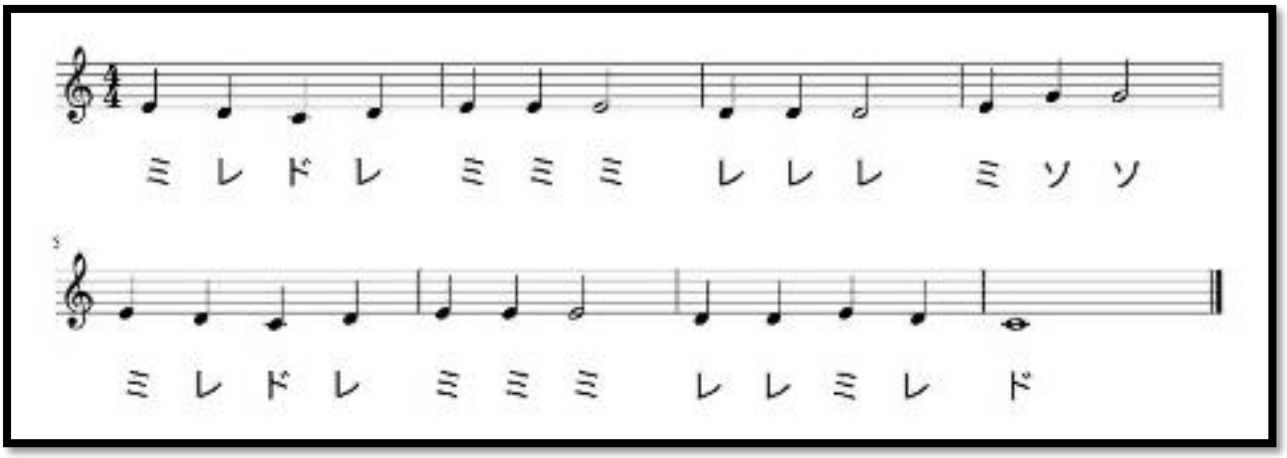
課題 1 （BuzzerSound.java）

前述の練習を別名保存し、以下の条件で音を奏でるプログラムを追記しなさい。

・SW1 → 練習のまま。押している間は LED1 を点灯し、ドを発音する。離すと消灯し、消音する。
・SW2 → 押している間は LED2 を点灯し、レを発音する。離すと消灯し、消音する。
・SW3 → 押している間は LED3 を点灯し、ミを発音する。離すと消灯し、消音する。
・SW4 → 押している間は LED4 を点灯し、ソを発音する。離すと消灯し、消音する。

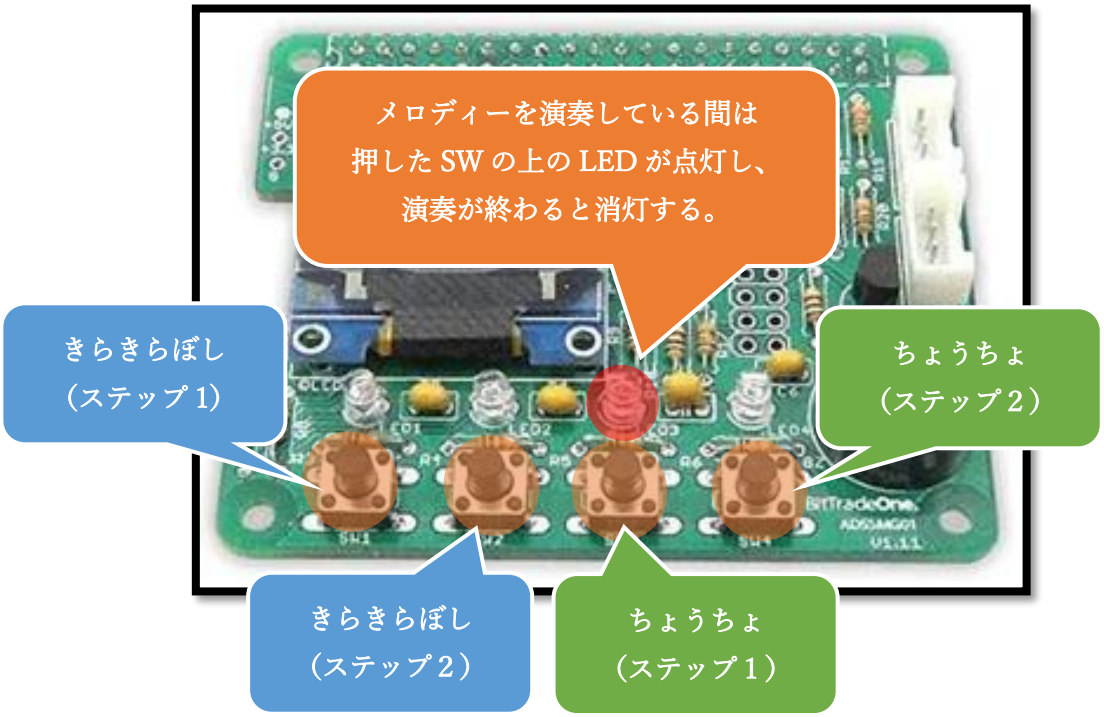


プログラムを実行したら↓の楽譜のとおりに音を奏でてみましょう。何の曲でしょうか？



課題 2 （BuzzerMelody.java）

SW を押した時に、割り当てたメロディーを楽譜のとおりに自動で演奏するプログラムを作成しなさい。
メロディーの音調と間隔は**定数配列で管理し、for 文を使用して**メロディーを奏でること。
なお、音を奏でた後は全て 16 分休符を挟むこと。

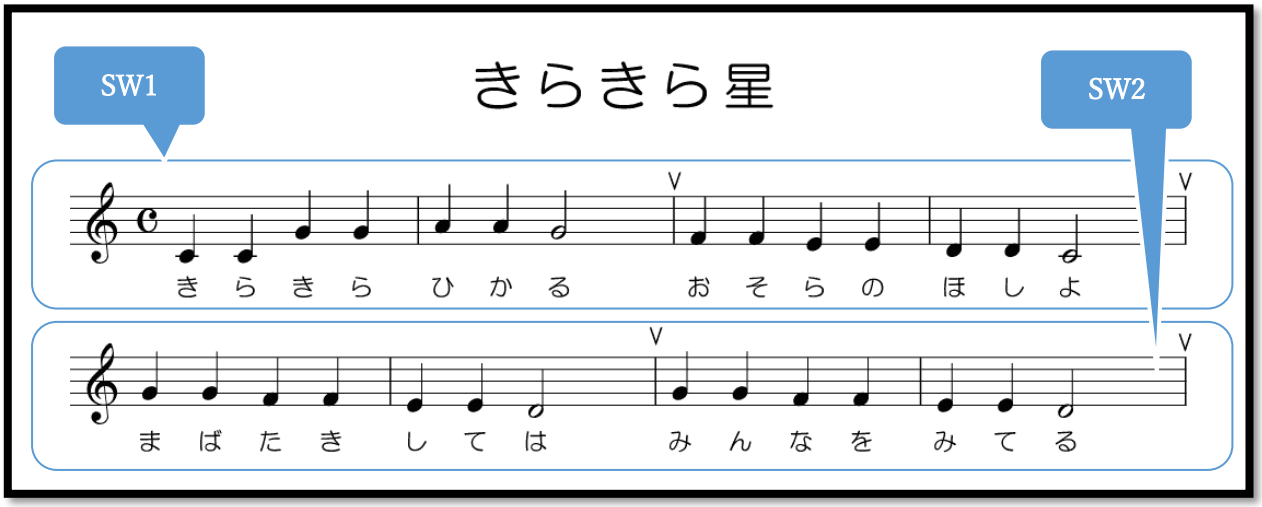


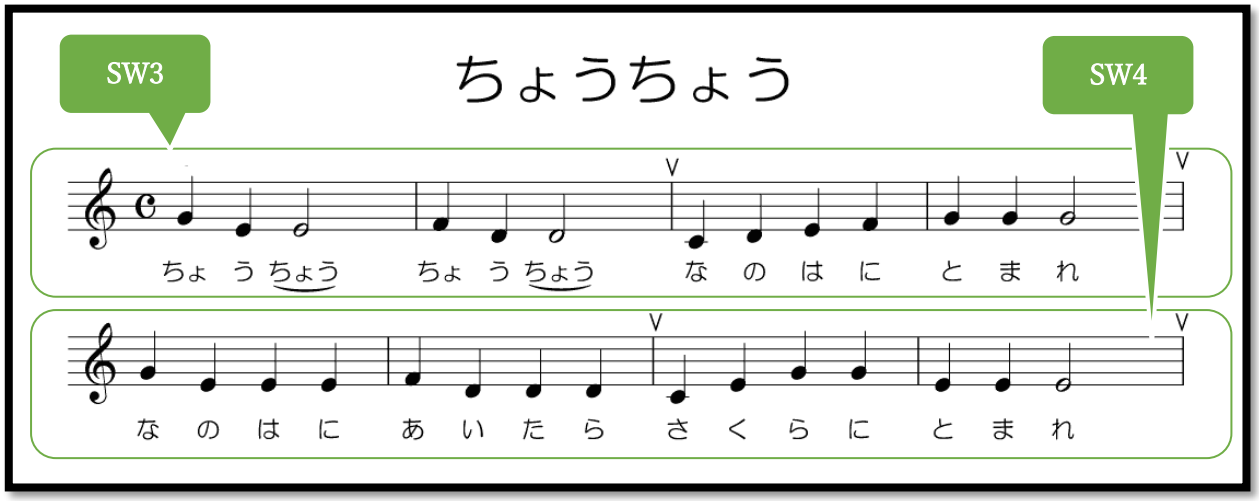
◆ヒント
演奏する為に発音の命令を一々記述するのは効率が悪い為、予め音調と間隔を定数配列に格納しておく。
課題 1 の楽譜を例に、ステップごとの音調と間隔を定数配列に記述すると以下のようになる。

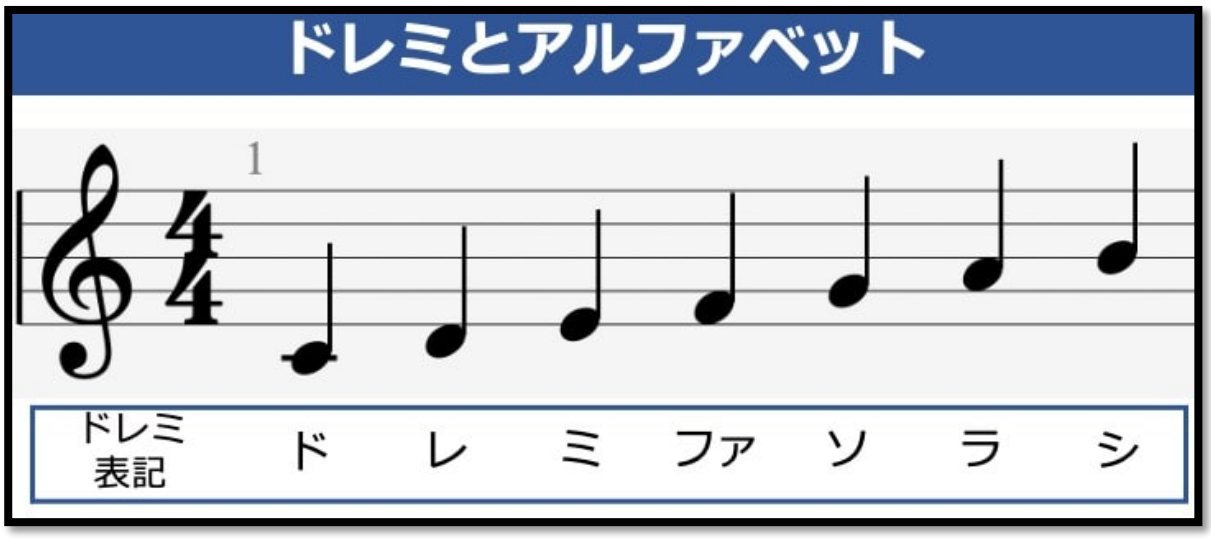
```
// メリーさんのひつじ（ステップ１）
final int[] MARY1_TONE = {E5, D5, C5, D5, E5, E5, E5, D5, D5, D5, E5, G5, G5};
final int[] MARY1_LENG = {L4, L4, L4, L4, L4, L4, L2, L4, L4, L2, L4, L4, L2};
// メリーさんのひつじ（ステップ２）
final int[] MARY2_TONE = {E5, D5, C5, D5, E5, E5, E5, D5, D5, E5, D5, C5};
final int[] MARY2_LENG = {L4, L4, L4, L4, L4, L4, L2, L4, L4, L2, L4, L1};
```

上記の配列を使用して以下のように記述するとコード量が少なくなる。

```
// メリーさんのひつじステップ１演奏
for(int i = 0; i < MARY1_TONE.length; i++){
    Gpio.pwmSetClock(～ 省略 ～);    //音調を設定
    Thread.sleep(～ 省略 ～);        //間隔分待機
    Gpio.pwmSetClock(～ 省略 ～);    //消音
    Thread.sleep(～ 省略 ～);        //休符分待機
}
```







休符の間隔は一律
16 分とする