学生証番号：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿　名前：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

5. Java と音

本実験で配布するヘッドセットのケーブルを色に合わせて PC 側にあるスピーカ出力とマイク入力端子に押し込もう。スピーカとマイクの音量は、デスクトップ下段にあるツールバーの右側にあるスピーカのアイコンをダブルクリックすると調整メニューが現れ、そこから調整できる。また、音ファイルの再生には windows media player などを使えばよい。

音は、物体の振動が空気を振動させ、さらにその振動が人間の鼓膜を振動させることにより聴こえるものである。また、物体の振動以外にも、物体の移動や空気の流れで音が発生する場合と、空気の急激な膨張・収縮によって音が発生する場合などがある。即ち音は波であり、物体や空間の一部の振動が隣から隣へと、次々に伝わっていく現象である。音の 3 要素は高さ、強さ、音色である。音の高さは波の周波数と、強さは振幅と、音色は波形と密接な関連がある。

Java によるサウンドアプリケーションの開発に次のページが有効である。

http://docs.oracle.com/javase/jp/1.5.0/guide/sound/programmer_guide/contents.html

5.1 MIDI

Java を用いて一番簡単に音を出す方法の一つが MIDI を使うことである。MIDI データは音、特に音楽を作成するためのレシピと考えることができる。MIDI データは、他のオーディオデータのようにサウンドを直接記述することはない。代わりに、シンセサイザにより作成されるサウンドに影響を与えるイベントを記述する。シンセサイザは、一般的には主に電子工学的手法により楽音等を合成する楽器の総称である。

sample 4.1 は、MIDI を使い簡単な音を発する例題である。midiplay.playNote(60)の 60 は中央の C 音を意味し、中央の D は 62、中央の E は 64 を意味し、61、63 は C、D、E の間の半音である。詳しい内容は、API マニュアルの javax.sound.midi のインタフェース MidiChannel を参考する。

```
/* sample 4.1 */
import javax.sound.midi.*;

public class Midiplay
{

	public Synthesizer synthesizer;
	public MidiChannel channel;

	public Midiplay()
	{
		try
		{
			synthesizer = MidiSystem.getSynthesizer();
			MidiChannel[] channelList = synthesizer.getChannels();
			channel = channelList[0];

			synthesizer.open();
		}
```

```
                    catch(Exception e)
                    {
                            close();
                    }
            }

        /** Note on する */
            private void noteOn(int noteNo)
            {
                    channel.noteOn(noteNo, 127);
            }

        /** Note off する */
            private void noteOff(int noteNo)
            {
//                  channel.noteOff(noteNo);
            }

        /** Synthesizer を閉じ，終了する */
            public void close()
            {
                    if(synthesizer != null)
                    {
                            synthesizer.close();
                    }
            }

            public void playNote(int noteNo) throws Exception
            {
                    noteOn(noteNo);
//                  Thread.sleep(length);
                    noteOff(noteNo);
            }
}


class midisample
{
            public static void main(String[] args)
            {
                    Midiplay midiplay = new Midiplay();

                    try
                    {
                            midiplay.playNote(60);
                            Thread.sleep(500);
                            midiplay.playNote(62);
                            Thread.sleep(500);
                            midiplay.playNote(64);
                            Thread.sleep(500);
                    }
                    catch(Exception e)
                    {
                            System.out.println(e);
                            midiplay.close();
                    }
```

```
                midiplay.close();
                System.exit(0);
        }
}
```

Sample 4.1 は、default 楽器であるピアノの音でドレミを演奏する。noteOn は、指定されたノートの音を指定されたキーダウン速度で出し始める。速度は、ボリュームまたは明るさ、あるいはその両方を制御ものである。

- 演習１０（＊）

channel.noteOn(noteNo, 127)の 127 という値を他の値(0~255)に変更し、その差を確認せよ。

- 演習１１（＊）

また、public void playNote(int noteNo)に加えて public void playNote(int noteNo, int length)のメソッドを追加し、public void playNote(int noteNo)と中身は一緒だけど Thread.sleep(length)のコメントアウト(//)を消して音の長さが指定できるようにせよ。length 値を変えながら実行結果を確認せよ。もちろん、main の中の midiplay.playNote();と Thread.sleep();も midiplay.playNote(,)に変更する必要がある。さらに、private void noteOff(int noteNo)の中の//も消して、実行結果を確認せよ。

- 演習１２

sample 4.1 を修正し、本資料の最後のページの曲の何れかを演奏してみよう。（他の曲でも良い）

- 演習１３

演習１２を修正し、和音で曲を演奏するプログラムを完成せよ。（楽譜がない場合は本資料の最後のページ参考）

- 挑戦８

演習１２または１３を修正し、楽譜ファイルを読み込んで曲を演奏するプログラムを完成せよ。ただし、楽譜ファイルのフォーマットは自分で工夫せよ。

- 挑戦９

ホームページの HINT を参考に複数の楽器を用いて曲を演奏するプログラムを作成せよ。

5.2　サンプリング

予め登録された楽器の波形で音を鳴らす MIDI とは異なるサンプリング（sampling）による音の発し方がある。サンプリングとはアナログ信号の強さを一定時間ごとに採取し、デジタル記録が可能な形にすることである。特に、サンプリングの意味として、音声をデジタルデータとして記録するために、一定時間ごとに音の強度を採取する処理を指すことが多い。信号の強度を採取する周期のことを「サンプリング周波数」と呼ぶ。単位は Hz で、1 秒間に何回採取するかを表す。音声の場合、サンプリング周波数が大きいほど高い音の記録が可能で、サンプリング周波数の半分にあたる周波数成分までなら元のアナログ信号に非常に近い信号を復元することができる。サンプリング方式として PCM（Pulse Code Modulation）が一番代表的である。

sample 4.2.1 はサンプリングされたサウンドファイルである wav ファイルを再生する例である。（SoundPlay.class がおいてあるフォルダと同じフォルダにある test.wav を再生するので実行する前

に test.wav が同じフォルダ内にあるか確認しよう。）sample 4.2.2 はマイクを使い音声をサンプリングし、wav ファイルで保存する例であるためマイクの使用が必要である。

```
/* sample 4.2.1 */
import java.io.File;
import javax.sound.sampled.*;

public class SoundPlay
{
	private static final int EXTERNAL_BUFFER_SIZE = 128000;

	public static void main(String[] args)
	{
		try
		{
	// File クラスのインスタンスを生成
			File soundFile = new File("test.wav");

	// オーディオ入力ストリームを取得
			AudioInputStream audioInputStream =
			    AudioSystem.getAudioInputStream(soundFile);
	// オーディオ形式を取得
			AudioFormat audioFormat = audioInputStream.getFormat();
	// データラインの情報オブジェクトを生成
			DataLine.Info info =
				new DataLine.Info(SourceDataLine.class,audioFormat);
	// 指定されたデータライン情報に一致するラインを取得
			SourceDataLine line =
				(SourceDataLine) AudioSystem.getLine(info);
	// 指定されたオーディオ形式でラインを開く
			line.open(audioFormat);
	// ラインでのデータ入出力を可能にする
			line.start();

			int nBytesRead = 0;
			byte[] abData = new byte[EXTERNAL_BUFFER_SIZE];
			while (nBytesRead != -1)
			{
	// オーディオストリームからデータを読み込む
				nBytesRead =
					audioInputStream.read(abData, 0,
abData.length);
				if (nBytesRead >= 0)
				{
	// オーディオデータをミキサーに書き込む
					int nBytesWritten =
						line.write(abData, 0, nBytesRead);
				}
			}
	// ラインからキューに入っているデータを排出する
			line.drain();
	// ラインを閉じる
			line.close();
			System.exit(0);
		}
```

```
                catch (Exception e)
                {
                        System.out.println(e);
                        System.exit(1);
                }
        }
}
```

ストリームとは、データのソースまたはデスティネーション(宛先)の抽象的な概念である。ストリームにより、同じテクニックを使用して各種物理デバイスと接続することが可能となる。たとえば、一つの入力ストリームで、キーボード、ファイル、メモリといった異なるデバイスからそのデータを読み取ることができる。また、一つの出力ストリームで、モニター、ファイル、メモリといった異なるデバイスにそのデータを書き込むこともできる。sample 4.2.1 はファイルから読み取ったサウンドデータをスピーカにストリームとして流している。非常に複雑に見えるかも知れないが、本資料の 4 章で紹介した Java Sound プログラマーズガイドのページを参考にしながらそれぞれのメソッド、インスタンスの意味を確認しよう。このサンプルは wav ファイルを再生する場合、必要最小限に近い構造なのでこのパターンをそのまま覚えるのもいいかも知れない。

```
/* sample 4.2.2 */
import java.io.IOException;
import java.io.File;
import javax.sound.sampled.*;

public class SoundRecord extends Thread
{
        private TargetDataLine m_line;
        private AudioFileFormat.Type m_targetType;
        private AudioInputStream m_audioInputStream;
        private File m_outputFile;

        public SoundRecord(TargetDataLine line,AudioFileFormat.Type targetType,File file)
        {
                m_line = line;
                m_audioInputStream = new AudioInputStream(line);
                m_targetType = targetType;
                m_outputFile = file;
        }

        //録音開始
        public void startRecording()
        {
                m_line.start();
                super.start();
        }

        //録音停止
        public void stopRecording()
        {
                m_line.stop();
                m_line.close();
        }
```

```
public void run()
{
	try
	{
		AudioSystem.write(m_audioInputStream,
					m_targetType,m_outputFile);
		System.out.println("録音が終了しました");
	}
	catch (IOException e)
	{
		System.out.println(e);
	}
}

public static void main(String[] args)
{
	try
	{
// File クラスのインスタンスを生成
		File outputFile = new File("myvoice.wav");

// 44.1 kHz, 16 bit, ステレオの設定でオーディオ形式を生成します
		AudioFormat audioFormat =
			new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,
					44100.0F, //サンプリング周波数
					16, //量子化データの大きさ 16bit
					2, //ステレオ
					4,
					44100.0F,
					false);

// データラインの情報オブジェクトを生成します
		DataLine.Info info =
			new DataLine.Info(TargetDataLine.class, audioFormat);
// 指定されたデータライン情報に一致するラインを取得します
		TargetDataLine targetDataLine =
			(TargetDataLine) AudioSystem.getLine(info);
// 指定されたオーディオ形式でラインを開きます
		targetDataLine.open(audioFormat);

// 書き込むオーディオファイルの種類を設定します
		AudioFileFormat.Type targetType = AudioFileFormat.Type.WAVE;
// 録音するインスタンスを生成
		SoundRecord recorder =
		new SoundRecord(targetDataLine,targetType,outputFile);
		System.out.println("ENTER キーを押すと録音を開始します");
		byte[] b = new byte[2];
		System.in.read(b);
// 録音を開始します
		recorder.startRecording();
		System.out.println("録音中...");
		System.out.println("ENTER キーを押すと録音を停止します");
		System.in.read(b); //enter キーを待つ
		recorder.stopRecording();
		System.out.println("録音を停止しました");
		System.exit(0);
```

```
                }
                catch (Exception e)
                {
                        System.out.println(e);
                        System.exit(1);
                }
        }
}
```

sample 4.2.2 はマイクからの音声ストリームデータをファイルに保存している。プログラムはマルチスレッドで動き、AudioSystem.write(m_audioInputStream, m_targetType,m_outputFile) でマイクからの音声ストリームデータをファイルに保存し続ける作業と main()でキー入力を待つ System.in.read(b)が同時に並列で動いている。もしこの状態で enter キーを押すと main スレッドは次に進み recorder.stopRecording()を呼び出す。stopRecording()メソッドでストリームを閉じ、AudioSystem.write はファイルの書き込みを終了し、そのスレッドも終了する。

- 演習１４

sample 4.2.1 と 4.2.2 を一つのプログラムに統合して録音した後、再生ができるようにせよ。

- 挑戦１０

8bit、モノでサンプリングされた wav ファイルの波形を表示するプログラムを作成せよ。

- 挑戦１１

8bit、モノでサンプリングされた wav ファイルを逆演奏するプログラムを作成せよ。

5.3 効果音

音を動きに合わせて出力ことでよりリアルな感覚をユーザに味あわせることを効果音と呼ぶ。sample 4.3 は動きに合わせて音を発する例である。Wav ファイルを再生する部分は sample 4.2.1 と一緒である。マルチスレッドの使い方に注目し、ソースを読んでみよう。

```
/* sample 4.3 */
import java.lang.*;
import java.io.File;
import javax.sound.sampled.*;

public class Reflection
{
        private static final int FIELD_WIDTH = 61;
        private String puck; // フィールドとパックを表示するための文字列

        // コンストラクタ
        public Reflection()
        {
                puck = new String("");
        }

        public void play() throws Exception
        {
```

```
                int p;
                int position;    // パックの位置
                int tempo = 10; // 反射の速さ
                while(tempo > 0)
                {
        // 一往復
                        for(int i=0; i<(FIELD_WIDTH-1)*2; i++)
                        {
                                puck = "|";
        // 往復の折り返しを考慮してパックの位置を計算
                                position =
                                    FIELD_WIDTH - Math.abs(i - (FIELD_WIDTH -1)) - 1;
                                for(p=0; p<position; p++)
                                {
                                        puck += " ";
                                }
                                puck += "O";
                                for(p+=1; p<FIELD_WIDTH; p++)
                                {
                                        puck += " ";
                                }
                                puck += "|";
                                System.out.println(puck);

                                // パックが壁に衝突したとき出音
                                if(position==0 || position==FIELD_WIDTH-1)
                                {
                                        AudioPlayer audioPlayer = new AudioPlayer();
                                        audioPlayer.start();
                                }

                                Thread.sleep(tempo+2);
                        }
                        tempo--;
                }
        }

        public static void main(String[] args) throws Exception
        {
                Reflection reflection = new Reflection();
                reflection.play();
                System.exit(0);
        }
}

class AudioPlayer extends Thread
{
        private static final int EXTERNAL_BUFFER_SIZE = 128000;

        public void run()
        {
                try
                {
        // File クラスのインスタンスを生成
                        File soundFile = new File("click.wav");
        // オーディオ入力ストリームを取得
```

```
                        AudioInputStream audioInputStream =
                                        AudioSystem.getAudioInputStream(soundFile);
        // オーディオ形式を取得
                        AudioFormat audioFormat = audioInputStream.getFormat();

        // データラインの情報オブジェクトを生成
                        DataLine.Info info =
                                new DataLine.Info(SourceDataLine.class, audioFormat);
        // 指定されたデータライン情報に一致するラインを取得
                        SourceDataLine line =
(SourceDataLine)AudioSystem.getLine(info);
        // 指定されたオーディオ形式でラインをオープン
                        line.open(audioFormat);
        // ラインでのデータ入出力を可能にします
                        line.start();

                        int nBytesRead = 0;
                        byte[] abData = new byte[EXTERNAL_BUFFER_SIZE];
                        while(nBytesRead != -1)
                        {
        // オーディオストリームからデータを読み込む
                                nBytesRead =
                                        audioInputStream.read(abData, 0,
abData.length);
                                if(nBytesRead >= 0)
                                {
        // オーディオデータをミキサーに書き込む
                                        int nBytesWritten =
                                                line.write(abData, 0,
nBytesRead);
                                }
                        }
        // ラインからキューに入っているデータを排出する
                        line.drain();
                        line.close();
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                        System.exit(1);
                }
        }
}
```

- 演習１５（＊）

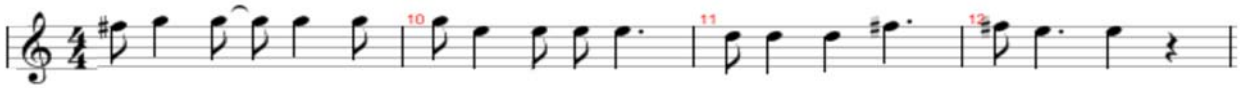
4.2.2 を使い「う」と「さ」の二つの wav ファイルを作り、sample 4.3 のパックが右に当たったときは「う」、左に当たったときは「さ」のサウンドが再生されるようにせよ。(「う」の再生が終わる前に「さ」の再生をしないように「う」と「さ」はなるべく早口で喋って録音する。また、パックの移動速度を十分遅くしてパックと壁の次の衝突までに前の効果音の再生が終わるようにしよう。)

- 挑戦１２

現時刻を声で知らせるプログラムを作成せよ。

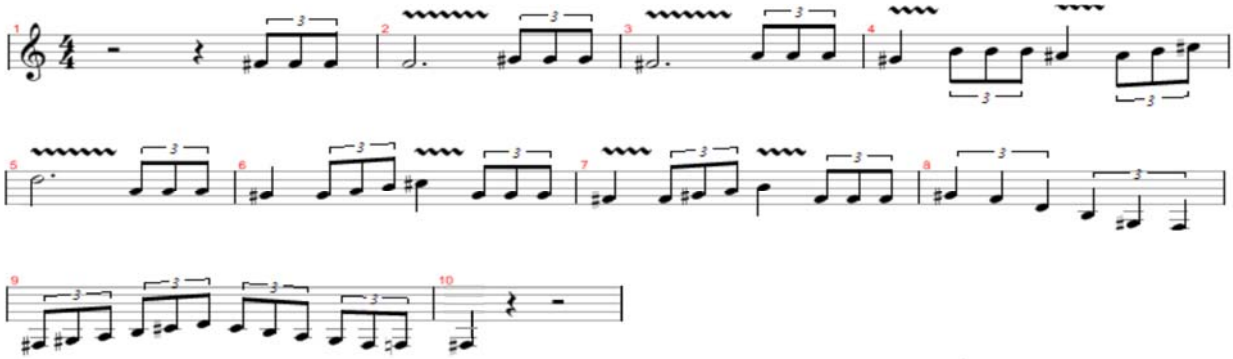
【楽譜例】

＜短音＞



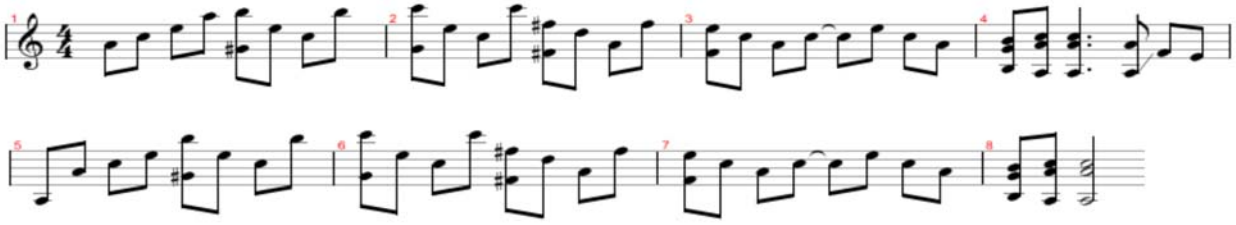
（Paranoid より）

（Crazy Train より）

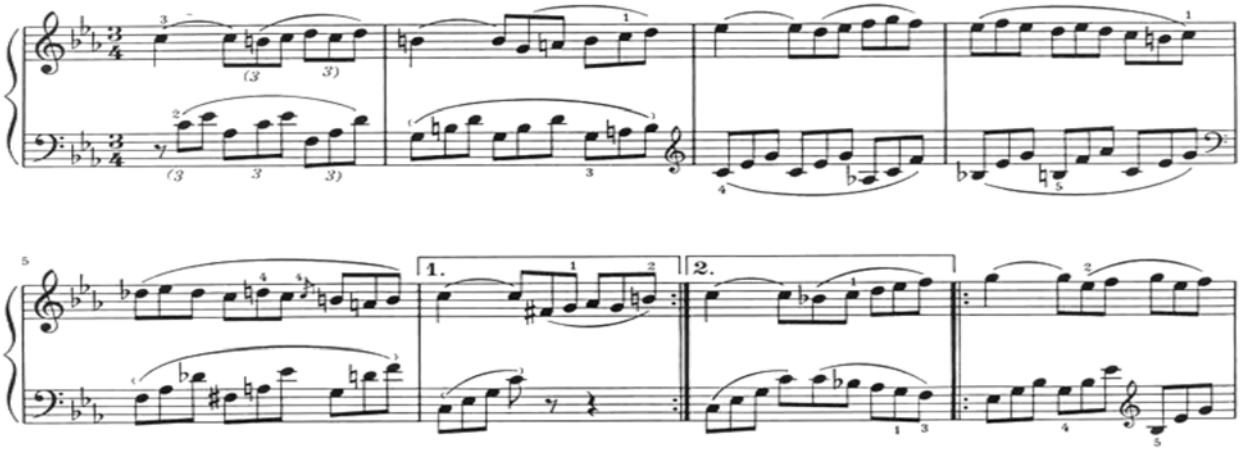


（Far beyond the sun より）

＜和音＞



（Stairway to heaven より）



（Variations WoO78 より）