(c)도경구 version 1.01 (2022/10/26)

8. 객체와 클래스

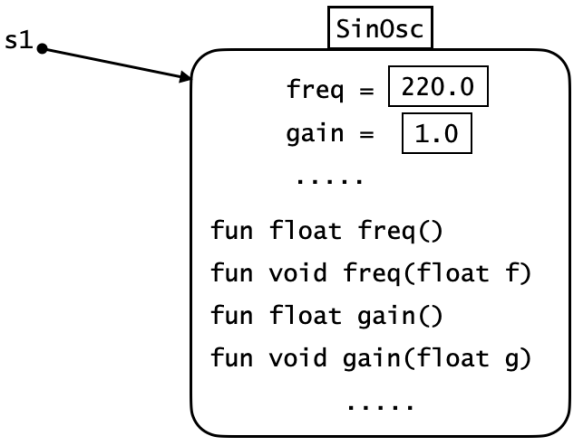
8-1. 객체

ChucK 프로그래밍언어는 객체지향 프로그래밍 언어(object-oriented programming language)이다. 계산의 대상이 되는 데이터를 객체(object)로 보고, 객체를 필요한 대로 실행 중에 만들어 계산을 수행한다. 메모리에 거주하는 객체는 각자 고유의 특징과 상태를 필드변수(field)에 기억하고 있으며, 고유의 기능을 메소드(method)라는 함수로 갖추고 있어서 객체들끼리 메소드 호출 메시지를 주고 받으며 상태를 참고하거나 변경하면서 계산을 수행한다.

우리가 쓰고 있는 진동기도 객체이다. 예를 들어 다음과 같이 `SinOsc` 진동기를 하나 설치하고 이름을 붙이면,

```
SinOsc s1;
```

메모리에 객체가 다음 그림과 같은 모양으로 생긴다고 상상할 수 있다.



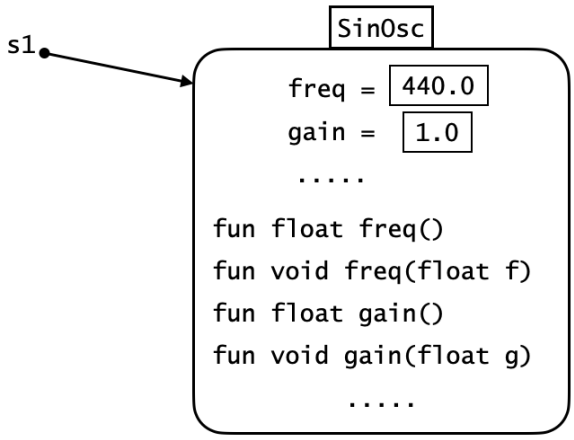
`s1` 이름으로 언제든지 이 `SinOsc` 진동기 객체에 접근할 수 있다.
다음과 같은 형식으로 메소드 호출 메시지를 보내서 주파수 정보를 알아볼 수도 있다.

```
<<< s1.freq() >>>;
```

이 객체의 주파수 변경은 다음과 같은 두 가지 형식으로 가능하다.

```
s1.freq(440.0);
440.0 => s1.freq;
```

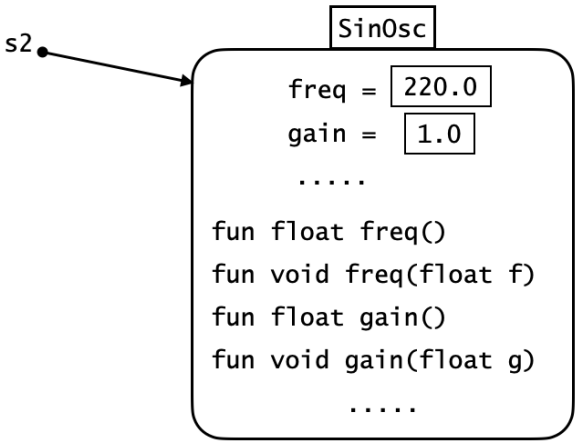
변경 후 `SinOsc` 진동기 객체는 다음과 같다.



`SinOsc` 진동기를 필요한 대로 몇개든 설치할 수 있다.
각 진동기는 이름을 다르게 붙여서 구별한다.

```
SinOsc s2;
```

그러면 메모리에 똑 같은 `SinOsc` 객체가 새로 또 하나 생긴다.

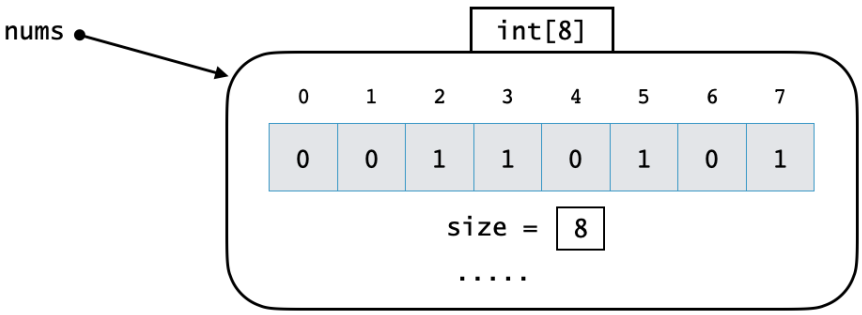


사실 `UGen` 은 모두 객체이다. 심지어는 버철머신 `Machine` 도 객체이다.

배열도 객체이다.
다음과 같이 배열을 만들어 선언하면,

```
[0,0,1,1,0,1,0,1] @=> int nums[];
```

메모리에 배열 객체가 다음 그림과 같은 모양으로 생긴다고 상상할 수 있다.



8-2. 클래스 만들기

클래스(class)는 객체를 만드는 일종의 형틀(template) 프로그램 이다.

8-2-1. 클래스 정의와 객체 생성

사례 1

```
// 클래스 정의
class PianoKey {
    60 => int note;
    1.0 => float gain;

    fun int changeOctave() {
        return note + 12;
    }

    fun int changeOctave(int n) {
        return note + n * 12;
    }
}

// 객체 생성
PianoKey key;

<<< key.note, key.gain, key.changeOctave() >>>;

2 +=> key.note;
0.3 -=> key.gain;
<<< key.note, key.gain, key.changeOctave(-1) >>>;
```

사례 2

```
class ResonantPop {
    Impulse imp => ResonZ filt => dac;
    100.0 => filt.Q => filt.gain;
    1000.0 => filt.freq;

    fun void freq(float freq) {
        freq => filt.freq;
    }

    fun void setQ(float Q) {
        Q => filt.Q;
    }

    fun void setGain(float gain) {
        filt.Q() * gain => imp.gain;
    }

    fun void noteOn(float volume) {
        volume => imp.next;
    }
}

ResonantPop pop;

while (true) {
    Std.rand2f(1100.0,1200.0) => pop.freq;
    1 => pop.noteOn;
    0.1 :: second => now;
}
```

8-2-2. `public` 클래스

정의한 클래스를 다른 파일에서 사용할 수 있도록 하려면, `class` 키워드 앞에 `public` 을 명시하여 공개 의사를 밝혀야 한다. `public` 키워드를 붙이고 실행하면, 선언한 클래스가 버철 머신에 공개 등록되면서 다른 프로그램 파일에서 접근하여 사용할 수 있게 된다. 버철 머신에 공개용으로 일단 등록이 되면, 동일 이름의 `public` 클래스의 재실행은 불가능하다. 클래스의 수정이 필요하다면, `clearVM` 단추를 눌러 버철머신을 청소하여 초기 상태로 되돌려 놓은 다음 재실행하는 수밖에 없다.

	public class	private class
예제	`public class Sample { … }`	`class Sample { … }` `private class Sample { … }`
가시 범위	외부에 공개	파일 내부에만 공개 파일 외부엔 가려짐

사례

아래 `BPM` 클래스는 템포 설정을 편리하게 할 용도로 제작하였다. 이 클래스의 `tempo` 메소드는 기준 박자를 받아서 1박자, 반박자, 1/4박자, 1/8박자를 한꺼번에 설정해준다.

BPM.ck

```
public class BPM { // Beats Per Minute
    dur quarter; // 1
    dur one_8th; // 1/2
    dur one_16th; // 1/4
    dur one_32nd; // 1/8

    fun void tempo(float beat) { // beat in BPM
        60.0 / beat => float spb; // seconds per beat
        spb::second => quarter;
        quarter / 2.0 => one_8th;
        quarter / 4.0 => one_16th;
        quarter / 8.0 => one_32nd;
    }
}
```

위의 `BPM` 클래스는 `public` 으로 정의되었으니, 실행하면 버철머신에 공개 등록된다. 따라서 아래와 같이 다른 파일에서 자유로이 `BPM` 객체를 생성하여 활용할 수 있다.

useBPM1.ck

```
SinOsc s => dac;
0.3 => s.gain;
BPM bpm;
bpm.tempo(300);

for (400 => int freq; freq < 900; 50 +=> freq) {
    freq => s.freq;
    bpm.quarter => now;
}
```

useBPM2.ck

```
SinOsc s => dac;
0.3 => s.gain;
BPM bpm;
bpm.tempo(200);

for (900 => int freq; freq > 400; 50 -=> freq) {
    freq => s.freq;
    bpm.quarter => now;
}
```

8-2-3. `static` 변수

필드 변수를 아래와 같이 `static` 으로 선언하면 어떤 차이점이 있나?

- `static` 이 아닌 필드 변수는 객체 소속으로, 설정한 값은 생성한 객체가 살아있는 동안만 유효하고 실행이 끝나면 객체와 함께 사라진다.
- `static` 필드 변수는 클래스 소속으로, 객체의 생사 여부와 상관없이 영구히 존재한다. 따라서 필드 변수 값은 버철 머신 전체에서 공유된다.

BPM.ck

```
public class BPM { // Beats Per Minute
    static dur quarter; // 1
    static dur one_8th; // 1/2
    static dur one_16th; // 1/4
    static dur one_32nd; // 1/8

    fun void tempo(float beat) { // beat in BPM
        60.0 / beat => float spb; // seconds per beat
        spb::second => quarter;
        quarter / 2.0 => one_8th;
        quarter / 4.0 => one_16th;
        quarter / 8.0 => one_32nd;
    }
}
```

아래 `useBPM1.ck` 를 실행하면 `bpm.tempo(300)` 메소드 호출로 `BPM` 클래스 소속 4개의 `static` 필드 변수 값이 각각 설정된다. 이후 이 파일의 실행이 끝난 이후에도 이 필드 변수 값은 그대로 살아있어서, 아래 `useBPM2.ck` 를 실행하면 별도로 다른 템포를 설정하지 않으면 기존에 설정된 값으로 프로그램을 실행한다. 직접 실행하여 확인해보자.

useBPM1.ck

```
SinOsc s => dac;
0.3 => s.gain;
BPM bpm;
bpm.tempo(300);

for (400 => int freq; freq < 900; 50 +=> freq) {
    freq => s.freq;
    bpm.quarter => now;
}
```

useBPM2.ck

```
SinOsc s => dac;
0.3 => s.gain;
BPM bpm;
// bpm.tempo(200);

for (900 => int freq; freq > 400; 50 -=> freq) {
    freq => s.freq;
    bpm.quarter => now;
}
```

8-2-4. 합주 활용 사례 1

위의 3 파일과 아래 3 파일을 같은 폴더에 넣고 `starter.ck` 파일을 실행시키면 다음과 같은 순서로 `score.ck` 의 쉬레줄에 따라 시간에 맞추어 다음과 같은 순서로 쉬레드가 생긴다. 실행하여 버철머신 모니터를 관찰해보자.

- shred 1 - `starter.ck` at 0:00
- shred 2 - `BPM.ck` at 0:00
- shred 3 - `score.ck` at 0:00
- shred 4 - `useBPM1.ck` at 0:00
- shred 5 - `useBPM2.ck` at 2:00
- shred 6 - `useBPM2.ck` at 7:00
- shred 7 - `useBPM3.ck` at 8:00
- shred 8 - `useBPM2.ck` at 10:00
- shred 9 - `useBPM3.ck` at 11:00
- shred 10 - `useBPM2.ck` at 13:00
- shred 11 - `useBPM3.ck` at 14:00
- ...

starter.ck

```
Machine.add(me.dir()+"BPM.ck");
Machine.add(me.dir()+"score.ck");
```

UseBPM3.ck

```
SinOsc s => dac;
0.3 => s.gain;
BPM bpm;
Math.random2f(300.0,1000.0) => bpm.tempo;

for (900 => int freq; freq > 400; 50 -=> freq) {
    freq => s.freq;
    bpm.quarter => now;
}
```

score.ck

```
Machine.add(me.dir()+"useBPM1.ck");
2.0::second => now;
Machine.add(me.dir()+"useBPM2.ck");
3.0::second => now;
// rest
2.0::second => now;
while (true) {
    Machine.add(me.dir()+"useBPM2.ck");
    1.0 :: second => now;
    Machine.add(me.dir()+"useBPM3.ck");
    2.0 :: second => now;
}
```

8-2-5. 합주 활용 사례 2 : 드럼 머신

BPM 클래스로 합주 박자 동기화하기

kick.ck

```
SndBuf kick => dac;
1 => kick.gain;
me.dir() + "audio/kick_01.wav" => kick.read;

BPM bpm;
while (true) {
    // Oxxx|Oxxx|Oxxx|Oxxx
    for (0 => int beat; beat < 4; beat++) {
        0 => kick.pos;
        bpm.quarter => now;
    }
}
```

snare.ck

```
SndBuf snare => dac;
0.5 => snare.gain;
me.dir() + "audio/snare_01.wav" => snare.read;
snare.samples() => snare.pos;

BPM bpm;
while (true) {
    // xxxxOxxxxxxxOOxx
    bpm.quarter => now;
    0 => snare.pos;
    2.0 * bpm.quarter => now;
    0 => snare.pos;
    bpm.quarter / 4.0 => now;
    0 => snare.pos;
    3.0 * bpm.quarter / 4.0 => now;
}
```

cowbell.ck

```
SndBuf cow => dac;
0.3 => cow.gain;
me.dir() + "audio/cowbell_01.wav" => cow.read;

BPM bpm;
while (true) {
    // xxxx|xxxx|xxxx|xxOx
    for (0 => int beat; beat < 8; beat++) {
        if (beat == 7)
            0 => cow.pos;
        bpm.one_8th => now;
    }
}
```

hihat.ck

```
SndBuf hat => dac;
0.3 => hat.gain;
me.dir() + "audio/hihat_02.wav" => hat.read;

BPM bpm;
while (true) {
    // OxOx|OxOx|OxOx|Oxxx
    for (0 => int beat; beat < 8; beat++) {
        if (beat != 7)
            0 => hat.pos;
        bpm.one_8th => now;
    }
}
```

clap.ck

```
SndBuf clap => dac;
0.3 => clap.gain;
me.dir() + "audio/clap_01.wav" => clap.read;

BPM bpm;
while (true) {
    // ????|????|????|???? (3/8 probability random)
    for (0 => int beat; beat < 16; beat++) {
        if (Math.random2(0,7) < 3) {
            0 => clap.pos;
        }
        bpm.one_16th => now;
    }
}
```

score.ck

```
BPM bpm;
bpm.tempo(120.0);

Machine.add(me.dir()+"kick.ck") => int kickID;
8.0 * bpm.quarter => now;
Machine.add(me.dir()+"snare.ck") => int snareID;
8.0 * bpm.quarter => now;
Machine.add(me.dir()+"hihat.ck") => int hatID;
Machine.add(me.dir()+"cowbell.ck") => int cowID;
8.0 * bpm.quarter => now;
Machine.add(me.dir()+"clap.ck") => int clapID;
8.0 * bpm.quarter => now;

<<< "Play with tempo" >>>;
80.0 => float new_tempo;
bpm.tempo(new_tempo);
8.0 * bpm.quarter => now;
2 *=> new_tempo;
bpm.tempo(new_tempo);
8.0 * bpm.quarter => now;

<<< "Gradually decrease tempo" >>>;
while (new_tempo > 60.0) {
    20 -=> new_tempo;
    bpm.tempo(new_tempo);
    <<< "tempo = ", new_tempo >>>;
    4.0 * bpm.quarter => now;
}

// pulls out instruments, one at a time
Machine.remove(kickID);
8.0 * bpm.quarter => now;
Machine.remove(snareID);
Machine.remove(hatID);
8.0 * bpm.quarter => now;
Machine.remove(cowID);
8.0 * bpm.quarter => now;
Machine.remove(clapID);
```

starter.ck

```
Machine.add(me.dir()+"BPM.ck");
Machine.add(me.dir()+"score.ck");
```

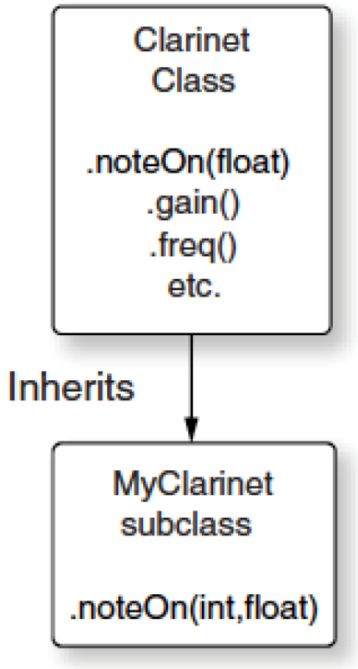
8-3. 상속

기존 클래스의 속성과 기능을 그대로 상속(inheritance)받아 재사용하면서 새로운 클래스를 손쉽게 만들 수 있다.

```
class Child extends Parent
```

위와 같이 `extends` 키워드를 사용하여 상속관계를 언급하면, `Parent` 클래스의 모든 것을 물려받아 `Child` 클래스를 작성한다는 뜻이 된다. 다시 말해, `Parent` 클래스에 작성되어 있는 코드는 `Child` 클래스에서 언급하지 않아도 모두 있는 것으로 간주한다는 말이다. `Child` 클래스에는 새로운 코드를 추가하거나, 중복시키거나(overload), 물려받은 코드를 무효화하고 새로 만들 수 있다(override).

8-3-1. 상속 사례 1 : 악기 개인화



MyClarinet.ck

```
public class MyClarinet extends Clarinet {
    // override
    fun void noteOn(int note, float volume) {
        Std.mtof(note) => this.freq;
        volume => this.noteOn;
    }
}
```

play.ck

```
MyClarinet clarinet => dac;

[60,62,64,65,67,69,71,72] @=> int scale[];

for (0 => int i; i < scale.size(); i++) {
    clarinet.noteOn(scale[i], 0.2);
    0.5::second => now;
    1 => clarinet.noteOff;
}
```

starter.ck

```
Machine.add(me.dir()+"myclarinet.ck");
Machine.add(me.dir()+"play.ck");
```

상속 사례 2 : 다형 (Polymorphism)

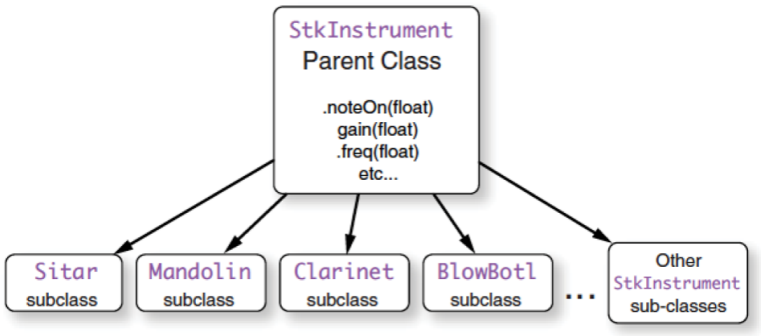
Parent 클래스의 변수에는 Child 객체를 담을 수 있다.
예를 들어 `Osc` 은 `SinOsc` , `TriOsc` , `SqrOsc` , `SawOsc` 의 Parent 클래스이다. 따라서 다음 사례와 같이 `Osc` 타입의 파라미터 변수 `osc` 는 이 네 종류의 진동기를 모두 수용할 수 있다.

```
TriOsc s => dac;
swell(s, 0.0, 1.0, 0.01);

fun void swell(Osc osc, float begin, float end, float step) {

    // swell up volume
    for (begin => float v; v < end; step +=> v) {
        v => osc.gain;
        0.02 :: second => now;
    }
    // swell down volume
    for (end => float v; v >= begin; step -=> v) {
        v => osc.gain;
        0.02:: second => now;
    }
}
```

`StkInstrument` 는 아래 그림에서 볼 수 있듯이 다양한 악기의 Parent 클래스이다. 따라서 아래 코드 사례에서 볼 수 있듯이, `StkInstrument` 배열은 소속의 어떤 악기 객체도 담을 수 있다.




```
StkInstrument inst[4];
Sitar inst0 @=> inst[0] => dac;
Mandolin inst1 @=> inst[1] => dac;
Clarinet inst2 @=> inst[2] => dac;
BlowBotl inst3 @=> inst[3] => dac;

for (0 => int i; i < 4; i++) {
    500.0 - (i*100.0) => inst[i].freq;
    1 => inst[i].noteOn;
    second => now;
    1 => inst[i].noteOff;
}
```

실습

1. 반달

버전 1

지난 실습 시간에 작성한 반달을 연주하는 프로그램을 다음 두 클래스 파일을 활용하는 버전으로 수정하자.

BPM.ck

```
public class BPM { // Beats Per Minute

    static dur n1, n2, n3, n5;

    fun void tempo(float beat) { // beat in BPM
        60.0 / beat => float spb; // seconds per beat
        spb::second => n1; // 1/6
        n1 * 2 => n2; // 2/6
        n1 * 3 => n3; // 3/6
        n1 * 5 => n5; // 5/6
    }
}
```

Tool.ck

```
public class Tool {

    fun void play(StkInstrument instrument, int note[], dur length[]) {
        for (0 => int i; i < note.size(); i++)
            playnote(instrument, note[i], length[i]);
    }

    fun void playnote(StkInstrument instrument, int note, dur length) {
        if (note != -1) {
            Std.mtof(note) => instrument.freq;
            1 => instrument.noteOn;
        }
        length => now;
        1 => instrument.noteOff;
    }

}
```

반달을 연주하는 프로그램은 `play.ck` 파일에 작성하고, 다음 `starter.ck` 파일을 사용하여 실행한다.

`starter.ck`

```
Machine.add(me.dir()+"BPM.ck");
Machine.add(me.dir()+"Tool.ck");
Machine.add(me.dir()+"play.ck");
```

버전 2

이번에는 멜로디를 연주하는 파일 `melody.ck` 과 화음을 연주하는 `harmony.ck` 파일을 따로 만들어 합주해보자.

2. 돌림노래 Row-Row-Row-Your-Boat

지난 실습 시간에 작성한 반달을 연주하는 다음 프로그램을 `BPM` 과 `Tool` 클래스 파일을 활용하는 버전으로 수정하자.

`BPM` 클래스는 이 노래의 박자에 맞게 수정해야 할 것이다.

```
// tempo
0.2::second => dur beat;
beat => dur n1; // 1/6
beat * 2 => dur n2; // 2/6
beat * 3 => dur n3; // 3/6
beat * 6 => dur n6; // 6/6

[ // melody
60,60,             60,62,64,          64,62,64,65, 67,
72,72,72,67,67,67, 64,64,64,60,60,60, 67,65,64,62, 60
] @=> int melody[];

[ // time
n3,n3,             n2,n1,n3,          n2,n1,n2,n1, n6,
n1,n1,n1,n1,n1,n1, n1,n1,n1,n1,n1,n1, n2,n1,n2,n1, n6
] @=> dur length[];

Rhodey piano[4];
piano[0] => dac;
piano[1] => dac;
piano[2] => dac;
piano[3] => dac;
spork ~ play(piano[0], melody, length);
n6 * 2 => now;
spork ~ play(piano[1], melody, length);
n6 * 2 => now;
spork ~ play(piano[2], melody, length);
n6 * 2 => now;
spork ~ play(piano[3], melody, length);
n6 * 8 => now;

fun void play(StkInstrument instrument, int note[], dur length[]) {
    for (0 => int i; i < note.size(); i++) {
        if (note[i] != -1) {
            Std.mtof(note[i]) => instrument.freq;
            1 => instrument.noteOn;
        }
        length[i] => now;
        1 => instrument.noteOff;
    }
}
```

3. Bach의 Crab Canon

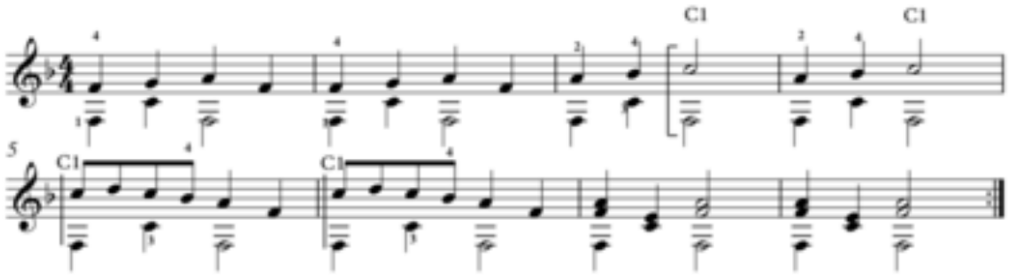
지난 실습 시간에 작성한 다음 프로그램을 `BPM` 과 `Tool` 클래스 파일을 활용하는 버전으로 수정하자.
이번에는 `BPM` 과 `Tool` 클래스를 이 노래에 맞게 수정해야 할 것이다.

4. J.S. Bach, Canon a 2 perpetuus (BWV 1075)

지난 실습 시간에 작성한 다음 프로그램을 `BPM` 과 `Tool` 클래스 파일을 활용하는 버전으로 수정하자.
이번에는 `BPM` 과 `Tool` 클래스를 이 노래에 맞게 수정해야 할 것이다.

숙제 (마감 11월 2일)

지난 실습 시간에 만든 프로그램을 이번엔 멜로디 2중창을 파일 하나에, 베이스를 다른 파일에 따로 두고 합주하는 방식으로 개선해보자. 'BPM' 클래스는 만들어 활용하고, `Tool` 클래스는 아래의 클래스를 가져다 활용하도록 한다.




```
[
"F4","G4","A4","F4",            "F4","G4","A4","F4",
"A4","Bb4","C5",                "A4","Bb4","C5",
"C5","D5","C5","Bb4","A4","F4", "C5","D5","C5","Bb4","A4","F4",
"F4","C4","F4",                 "F4","C4","F4"
] @=> string melody[];

[
"F4","G4","A4","F4",            "F4","G4","A4","F4",
"A4","Bb4","C5",                "A4","Bb4","C5",
"C5","D5","C5","Bb4","A4","F4", "C5","D5","C5","Bb4","A4","F4",
"A4","E4","A4",                 "A4","E4","A4"
] @=> string melody_high[];

[
 qn,  qn,  qn,  qn,             qn,  qn,  qn,  qn,
 qn,  qn,  hn,                  qn,  qn,  hn,
 en,  en,  en,  en,   qn,  qn,  en,  en,  en,  en,   qn,  qn,
 qn,  qn,  hn,                  qn,  qn,  hn
] @=> dur durs[];

[
"F3","C4","F3", "F3","C4","F3", "F3","C4","F3", "F3","C4","F3",
"F3","C4","F3", "F3","C4","F3", "F3","C4","F3", "F3","C4","F3"
] @=> string bass[];

[
 qn,  qn,  hn,   qn,  qn,  hn,   qn,  qn,  hn,   qn,  qn,  hn,
 qn,  qn,  hn,   qn,  qn,  hn,   qn,  qn,  hn,   qn,  qn,  hn
] @=> dur durs_bass[];
```

```
public class Tool {

    fun void play(StkInstrument instrument, string notes[], dur durs[]) {
        for (0 => int i; i < notes.size(); i++)
            playnote(instrument, notes[i], durs[i]);
    }

    fun void playnote(StkInstrument instrument, string note, dur duration) {
        Std.mtof(ntom(note)) => instrument.freq;
        if (note != "REST")
            1 => instrument.noteOn;
        duration => now;
        1 => instrument.noteOff;
    }

    fun int ntom(string name) { // note name to MIDI number
        [21,23,12,14,16,17,19] @=> int notes[]; // A0,B0,C0,D0,E0,F0,G0
        name.charAt(0) - 65 => int base; // A=0,B=1,C=2,D=3,E=4,F=5,G=7
        notes[base] => int note;
        if (0 <= base && base <= 6) {
            if (name.charAt(1) == '#' || name.charAt(1) == 's') // sharp
                notes[base] + 1 => note;
            if (name.charAt(1) == 'b' || name.charAt(1) == 'f') // flat
                notes[base] - 1 => note;
        }
        else
            <<< "Illegal Note Name!" >>>;
        name.charAt(name.length()-1) - 48 => int oct; // 0, 1, 2, ..., 9
        if (0 <= oct && oct <= 9) {
            12 * oct +=> note;
            return note;
        }
        else
            <<< "Illegal Octave!" >>>;
    }

}
```