Graceful Music Docs (Cheat Sheet)

Sam Minns, Victoria University

Overview

Everything in Graceful Music happens in the context of a Performance. A Performance is an environment made up of two parts, an instrument that is for live playing and the sequencer which is like a backing band!
When you give a Performance a Synth as its instrument that Synth then shows up on your MIDI keyboard as shown below.

Performance with only a playable synthesiser

```
        def s = Synth.name("Howard") wave("sine") chord("")
        def p = Performance.instrument(s) sequencer("")
```

You can now play ‘s’ on your keyboard like a normal synthesiser. In order to get a backing band we need to create some Parts.

A Part is a combination of an instrument and a rhythm. We give parts to a Sequencer which takes care of playing the Parts for us, then we give the Sequencer to the Performance so our live instruments and our band are in the same place.

As shown below, we create two Synths, one we give a chord we want it to play. We give that Synth to a Part along with a rhythm, then that Part we give to the Sequencer. All of that goes into the Performance and we’re off!

Performance with only a playable synthesiser and sequencer

```
        def seqSynth = Synth.name("s") wave("triangle") chord("C-MAJ7")
        def liveSynth = Synth.name("live") wave("square") chord("")

        def seqSynthPart = Part.instrument(sequencedSynth) rhythm("CR")
        def seq = Sequencer.parts(list.with(sequencedSynthPart))

        def p = Performance.instrument(liveSynth) sequencer(seq)
```

Notice when we give Parts to the Sequencer we surround them with the *list.with(x, y, z)* syntax. This allows us to send as many Parts to the Sequencer as we want.

class Synth :: Instrument

Polyphonic Synthesiser, when passed as the instrument argument to a Performance object can be played via a MIDI interface. When passed a chord and passed to a sequencer it can be sequenced using a RhythmUtil constant.

Constructor

```
        def s = Synth.wave(wave‘) chord(chord‘)
```

Constructor Arguments

Name	Argument Type	Required	Options
name	string	yes	
wave	string	yes	sine, square, triangle, saw
chord	string	yes	HarmonyUtil.CONSTANT

Methods

Name	Argument Type	Required	Return	Constraints / Options
setEnvelopeAttack	float	yes	void	+ve
setEnvelopeSustain	float	yes	void	+ve
setEnvelopeRelease	float	yes	void	+ve
insert	string	yes	void	“reverb”
				“distortion”
invertChord	none	n/a	void	chord not null
pan	float	yes	void	-1.0..1.0

class DrumMachine :: Instrument

A three part Drum Machine, hihat, snare and kick are loaded on instantiation. Can currently only be passed with a Part to a Sequencer to be sequenced.

Example: DrumMachine sequenced

```
        def d = DrumMachine.name("rockbeat")
        def dp = Part.instrument(d) rhythm("ROCK_BEAT_ONE")

        def seq = Sequencer.parts(list.with(dp))
        def p = Performance.instrument("") sequencer(seq)
```

Example: DrumMachine sequenced with playable Synth

```
        def s = Synth.name("s") wave("triangle") chord("C-MAJ7")
        s.setEnvelopeRelease(2.0)
        s.insert("reverb")
        s.insert("distortion")

        def d = DrumMachine.name("rockbeat")
        def dp = Part.instrument(d) rhythm("ROCK_BEAT_ONE")

        def seq = Sequencer.parts(list.with(dp))
        def p = Performance.instrument(s) sequencer(seq)
```

class LoopPlayer :: Instrument

Example: LoopPlayer with first sample file

```
        def s = Synth.name("s") wave("triangle") chord("C-MAJ7")
        def lp = LoopPlayer.name("lp" ) filename("sample_2.wav")

        def lpprt = Part.instrument(lp) rhythm("ONE_BAR_LOOP")

        def seq = Sequencer.parts(list.with(lpprt))
        def p = Performance.instrument(s) sequencer(seq)
```

Constructor Arguments

Name	Argument Type	Required	Options
name	string	yes	
filename	string	yes	sample_1.wav
			sample_2.wav
			sample_3.wav
			sample_4.wav
			kick.wav
			snare.wav
			hihat.wav

class Part

Wraps any subclass of Instrument and any RhythmUtil constant referenced as a string.

constructor

```
        def lp = Part.instrument(instrument) rhythm("rhythm")
```

Constructor Arguments

Name	Argument Type	Required	Options
instrument	Instrument	yes	
rhythm	string	yes	RhythmUtil.CONSTANT

class RhythmUtil

Exposes simple rhythmic patterns that may be paired with Instruments to be sequenced.

Class Constants

Name	Desc.	
CR	One note per beat for one bar. (crotchet)	
CR_OFF	One note per off beat for one bar. (crotchet)	
QU	Two notes per beat for one bar. (quaver)	
SQ	Four notes per beat for one bar. (semiquaver)	
ONE_THREE	Notes on the first and third beats for one bar.	
TWO_FOUR	Notes on the second and fourth beats for one bar.	
FOUR_FOUR	Notes on every beat for one bar.	
CLAVE	2/3 Clave Rhythm	
ONE_BAR_LOOP	Note on beat one for one bar	
TWO_BAR_LOOP	Note on beat one for two bars	
FOUR_BAR_LOOP	Note on beat one for four bars	
ROCK_BEAT_ONE	Hihat > QU Snare > TWO_FOUR KICK > ONE_THREE	DrumMachine only
HOUSE_BEAT	Hihat > CR_OFF Snare > TWO_FOUR Kick > FOUR_FOUR	DrumMachine only

class HarmonyUtil

Exposes simple harmonies for use with a sequenced Synth object, passed as a string to the Synth constructor. Must be prefixed with a note name from A-G.

Example:

```
        def s = Synth.name("Harold") wave("sine") chord("C-MAJ7")
```

Class Constants

Name	Desc.
-MAJ	Major triad 1 - 3 - 5
-MAJ7	Major seven 1 - 3 - 5 - 7
-MAJ9	Major ninth 1 - 3 - 5 - 7 - 9
-DOM	Dominant seventh 1 - 3 - 5 - b7
-MIN	Minor 1 - b3 - 5
-MIN7	Minor seventh 1 - b3 - 5 - b7
-MIN9	Minor ninth 1 - b3 - 5 - b7 - 9
-MIN11	Minor eleventh 1 - b3 - 5 - b7 - 9 - 13 (4)
-DIM	Diminished 1 - b3 - b5
-AUG	Augmented 1 - 3 - #5

class Sequencer

Sequences multiple instruments according to the RhythmUtil constant passed in the Part object.

Example with multiple sequenced parts and a playable Synth

```
        def s = Synth.name("s") wave("triangle") chord("")

        def lp = LoopPlayer.name("lp" ) filename("sample_1.wav")
        def lpprt = Part.instrument(lp) rhythm("ONE_BAR_LOOP")

        def d = DrumMachine.name("rockbeat")
        def dp = Part.instrument(d) rhythm("ROCK_BEAT_ONE")

        def seq = Sequencer.parts( list.with(lpprt, dp) )
        def p = Performance.instrument(s) sequencer(seq)
```

constructor

```
        def s = Sequencer.parts( list.with(parts))
```

Constructor Arguments

Name	Argument Type	Required	Options
list.with(partOne, partTwo)	List -> Part	yes	

class Performance

A live performance environment that supports live MIDI playback of a single Synth or LoopPlayer object and sequencing of arbitrarily many Parts.

constructor

```
        def p = Performance.instrument(instrument‘) sequencer(sequencer‘)
```

Constructor Arguments

Name	Argument Type	Required
instrument	Instrument	no
sequencer	Sequencer	no