

```

1 @startuml
2
3 class Sampler {
4     - globalVolume: int
5     - presetFileDirectory: String
6     - sampleLibraryDirectory: String
7     - globalRootPitch: int
8     - sampleFile: File
9     - audioOutputLine: DataLine
10    - settingsChangeObservers: SettingsChangeObserver
    [0..*]
11    + getInstance(): Sampler
12    + getVolume() : int
13    + getSample() : File
14    + getGlobalRootPitch() : int
15    + setVolume() : void
16    + setGlobalRootPitch() : void
17    + setSampleLibraryDirectory(libraryPath: String
    ) : void
18    + loadSample(sampleFile: File) : void
19    + createSettingsMemento() :
    SamplerSettingsMemento
20    + restoreSettings(previousSettings:
    SamplerSettingsMemento) : void
21    + notifyObserversOfSettingsChange() : void
22 }
23 note left: **Singleton**\nControls global and
    administrative settings. Creates and\nrestores state
    from SamplerSettingsMemento.\nProvides audio data to
    system audio data line.
24
25
26 interface MidiEffectChain {
27     process(note: MidiMessage) : MidiMessage
28 }
29
30
31 class BasicMidiGenerator implements MidiEffectChain {
32     + process(note: MidiMessage) : MidiMessage
33 }
34

```

```

35
36 class MidiDecorator implements MidiEffectChain {
37     - wrappedMidiEffect : MidiEffectChain
38     + MidiDecorator(midiEffect: MidiEffectComponent
39         ) : MidiDecorator
40     + process(note: MidiMessage) : MidiMessage
41 }
42
43
44 class RandomizePitch extends MidiDecorator {
45     + process(note: MidiMessage) : MidiMessage
46 }
47
48
49 class RandomizeVolume extends MidiDecorator {
50     + process(note: MidiMessage) : MidiMessage
51 }
52
53
54 class Arpeggiator extends MidiDecorator {
55     + process(note: MidiMessage) : MidiMessage
56 }
57
58
59 class NoteFactory {
60     - notes: String
61     - midiEffectChain: MidiEffectChain
62     + getNote(note: MidiMessage): Note
63 }
64
65 note bottom: **Flyweight**\nCreates Note objects
66     based on \nuser input to the system
67
68 class Note {
69     - pitch: int
70     - velocity: int
71     + getPitch(): int
72     + getVolume(): int

```

```

72 }
73 note right: Encapsulates user input from key\
    npresses as pitch and volume values.
74
75
76 class NoteManager {
77     - inputLine : Scanner
78     - noteFactory: NoteFactory
79     - noteBuffer: Note[]
80     - effectsChain: MidiEffectChain
81     + getNoteBuffer : Note[]
82     + isValidInput(input : SystemInput) : boolean
83 }
84 note bottom: Receives input messages from System.\
    nGets processed data from MidiEffectChain\and
    requests Note objects from NoteFactory.
85
86
87 class Voice implements SettingsChangeObserver {
88     - level: int
89     - rootPitch: int
90     - isIdle: boolean
91     - voiceAudioDataBuffer: byte[]
92     - voiceDataConsolidator: VoiceDataConsolidator
93     + generateAudioData(note: Note) : void
94     + writeToBuffer(): void
95     + onSettingsChange() : void
96 }
97 note left of Voice: Writes audio data to
    VoiceDataConsolidator\n buffers. Observes Sampler
    for changes to pitch and volume.
98
99
100 class VoiceManager {
101     - voicePool: Voice[1..4]
102     - noteManager: NoteManager
103     + delegateNoteToAvailableVoice(note: Note): void
104     + createVoice(): void
105 }
106 note right: Creates Voice objects.\nGets Note
    objects from NoteManager and \ndistributes them to

```

```

106 idle Voice objects.
107
108
109 class VoiceDataConsolidator {
110     - voiceAudioDataBuffers: byte[1..4][ ]
111     - effectsChain: AudioEffect
112     + getSummedData() : byte[]
113     + sumVoiceAudioDataBuffers() : byte[]
114     + sendAudioToAudioEffectChain(effectChain:
      AudioEffect) : void
115 }
116 note bottom: Combines audio data generated by Voice
      objects into a \nsingle buffer for every global
      clock time step and sends it\nto the effects chain.
117
118
119 class Sample {
120     - file: File
121     - fileType: String
122     - filePath: String
123     + getFile(): File
124 }
125
126
127 class AudioEffectChain {
128     - effectChain: AudioEffect [0..*]
129     - settingsChangeObservers:
      SettingsChangeObserver [0..*]
130     - effectFactory: AudioEffectFactory
131     - sampler: Sampler
132     + addEffect(effectType: String): void
133     + removeEffect(effectType: String): void
134     + rebuildEffectChain(): void
135     + createSettingsMemento() :
      EffectsChainSettingsMemento
136     + restoreSettings(previousSettings:
      EffectsChainSettingsMemento) : void
137     + notifyObserversOfSettingsChange() : void
138 }
139 note right: Maintains an effects chain.\nCreates and
      restores state from\nEffectsChainSettingsMemento.

```

```
140
141
142 interface AudioEffect {
143     setNextEffect(nextEffect: AudioEffect) : void
144     processAudio(audioDataBuffer: byte []) : byte []
145 }
146 note right: Interface for AudioEffectProcess chain
    of responsibility.
147
148
149 class AudioEffectProcess implements AudioEffect,
    SettingsChangeObserver {
150     - nextEffect: AudioEffect
151     - audioOutputLine: DataLine
152     + setNextEffect(nextEffect: AudioEffect) : void
153     + processAudio(audioDataBuffer: byte []) : byte
        []
154     +onSettingsChange() : void
155
156 }
157 note left: **Chain of responsibility**\nAll
    AudioEffectProcess objects process audio data and
    give\nprocessed audio data to their nextEffect
    object or give\nit to the output data line for the
    system to turn into sound.
158 note right of AudioEffectProcess: These objects
    apply processing to\nthe audio data they're passed.
159
160
161 interface AudioEffectFactory {
162     + createAudioEffect(effectType: String) :
        AudioEffect
163 }
164 note right: **Factory**\nInterface for creation of
    AudioEffect objects.
165
166
167 class ConcreteAudioEffectFactory implements
    AudioEffectFactory {
168     + createAudioEffect(effectType: String) :
        AudioEffect
```

```
169 }
170 note right: Contains logic for creation of
    AudioEffect objects.
171
172
173 class Reverb extends AudioEffectProcess {
174     + processAudio(audioDataBuffer: byte []) : byte
    []
175 }
176
177
178 class Delay extends AudioEffectProcess {
179     + processAudio(audioDataBuffer: byte []) : byte
    []
180 }
181
182
183 class Chorus extends AudioEffectProcess {
184     + processAudio(audioDataBuffer: byte []) : byte
    []
185 }
186
187
188 interface SettingsChangeObserver {
189     + onSettingsChange() : void
190 }
191 note right: **Observer**
192
193
194 class SamplerSettingsMemento {
195     - sampleFilePath: String
196     - globalVolume: int
197     - globalRootPitch: int
198     - numberOfVoices: int
199 }
200 note left: **Memento**
201
202
203 class EffectsChainSettingsMemento {
204     - audioEffectChain; String[0..*]
205     - midiEffectChain; String[0..*]
```

```

206 }
207 note left: **Memento**
208
209
210 class ConsolidatedSettings {
211     - samplerSettings: SamplerSettingsMemento
212     - effectsChainSettings:
        EffectsChainSettingsMemento
213 }
214
215
216 class SettingsUndoRedoManager {
217     - originatorSampler: Sampler
218     - originatorEffects: AudioEffectChain
219     - stateHistory: ConsolidatedSettings [0..*]
220     - currentStateIndex: int
221     + onSettingsChange() : void
222     + undoState() : void
223     + redoState() : void
224 }
225 note right: Manages list of previous system states
        and\tells Sampler and AudioEffectChain when to\
        nrestore a state.
226
227
228 Sample "1" -o "1" Sampler
229 Sample <-- Voice
230
231 'Note and Voice
232 NoteFactory "1" --* "1" NoteManager
233 Note "1..128" --o "1" NoteFactory
234 VoiceManager o-- NoteManager
235 Voice "1..4" ---* "1" VoiceManager
236 Voice ---> Note
237 Voice o--> VoiceDataConsolidator
238
239 'SettingsUndoRedoManager
240 SamplerSettingsMemento <-- Sampler
241 ConsolidatedSettings "0..*" --* "1"
        SettingsUndoRedoManager
242 SamplerSettingsMemento "1" --o "1"

```

```
242 ConsolidatedSettings
243 EffectsChainSettingsMemento "1" --o "1"
    ConsolidatedSettings
244 SettingsUndoRedoManager ..|> SettingsChangeObserver
245 SettingsUndoRedoManager --> AudioEffectChain
246 Sampler <-- SettingsUndoRedoManager
247 Sampler "1" o-- "0..*" SettingsChangeObserver
248 EffectsChainSettingsMemento <-- AudioEffectChain
249 SettingsChangeObserver "0..*" --o "1"
    AudioEffectChain
250
251 'Audio and MIDI effects
252 AudioEffect "0..*" --o "1" AudioEffectChain
253 AudioEffect <---o AudioEffectProcess
254 VoiceDataConsolidator o---> AudioEffect
255 AudioEffectChain o-- AudioEffectFactory
256 ConcreteAudioEffectFactory -> AudioEffect
257 MidiEffectChain --o NoteManager
258
259 @enduml
```



```
1 @startuml
2 |User|
3 start
4
5 :Enters text in search field;
6
7 |Sampler|
8 :Searches library directory for matches;
9 if (Matches found?) then (Yes)
10     :Displays matches in results field;
11
12     |User|
13     if (Selects an effect?) then (Yes)
14
15         |Sampler|
16         if (Is audio or MIDI effect?) then (Audio)
17
18             |AudioEffectChain|
19             :Add effect to audio processing chain;
20
21         |Sampler|
22         else (MIDI)
23
24             |NoteManger|
25             :Add effect to MIDI processing chain;
26         endif
27
28         |SettingsUndoRedoManager|
29         :Tells AudioEffectChain to create memento;
30         :Stores memento in state history;
31         note right: Save state for undo/redo
32
33     |Sampler|
34     else (No)
35     endif
36
37 |Sampler|
38 else (No)
39     :Display "No matches found" message to User;
40 endif
41
```

```
42 |User|  
43 stop  
44  
45 @enduml
```

```

1 @startuml
2
3 |User|
4 start
5 :Enters text in search field;
6
7 |Sampler|
8 :Searches library directory for matches;
9 if (Matches found?) then (Yes)
10     :Displays matches in results field;
11
12     |User|
13     if (Selects a sample?) then (Yes)
14
15         |Sampler|
16         if (Different file is already loaded?) then (
17             Yes)
18             :Display "Load new sample?" message;
19
20             |User|
21             if (Confirms "Yes"?) then (Yes)
22
23                 |Sampler|
24                 :Perform selection validation;
25                 if (Selection is valid?) then (Yes)
26                     :Load file into sampler
27                     application and readies it for playback;
28                     :Displays "Sample loaded" message
29                     ;
30                 else (No)
31                     :Display "Invalid file" message;
32                 endif;
33
34             |User|
35             else (No)
36
37                 |Sampler|
38                 :Close dialogue;
39                 :Displays matches in results field;
40
41             |User|

```

```
39         endif
40         |Sampler|
41     else (No)
42         :Perform selection validation;
43         if (Selection is valid?) then (Yes)
44             :Load file into sampler application
and readies it for playback;
45             :Displays "Sample loaded" message;
46         else (No)
47             :Display "Invalid file" message;
48         endif;
49     endif
50
51 '     |User|
52     else (No)
53     endif
54
55 |Sampler|
56 else (No)
57     :Display "No matches found" message to User;
58 endif
59
60 |User|
61 stop
62
63 @enduml
```

```
1 @startuml
2
3 |User|
4 start
5 :Presses a key on computer keyboard or MIDI device;
6
7 |Sampler|
8 if (Sample is loaded?) then (Yes)
9
10     |NoteManager|
11     :Process MIDI input;
12     :Send Notes to VoiceManager;
13
14     |VoiceManager|
15     :Send note values to an available Voice object;
16
17     |Voice|
18 floating note right: Per voice parallel processing
19     fork
20         :Generate audio data;
21     fork again
22         :Generate audio data;
23     fork again
24         :Generate audio data;
25     fork again
26         :Generate audio data;
27     end fork
28
29     |VoiceDataConsolidator|
30     :Process audio data;
31
32     |AudioEffectsChain|
33     :Process audio data;
34     :Send processed audio data to output line;
35
36     |Sampler|
37
38 else (No)
39 endif
40
41 |User|
```

```
42 stop
43
44
45 @enduml
46
47
```

```
1 @startuml
2
3
4 |User|
5 start
6 :Clicks "Save Preset" button on main UI;
7
8 |PresetManager|
9 :Opens "Save Preset" dialogue box;
10
11 |User|
12 :Enters name for preset;
13 :Clicks "Save";
14
15 |PresetManager|
16 if (Is valid file name?) then (Yes)
17     if (No PresetMemento in preset list?) then (Yes)
18         :Creates PresetMemento object with current
        state;
19     else (no)
20     endif
21     :Save last PresetMemento as preset values into a
        file in preset directory;
22     :Notifies User that preset has been saved;
23 else (no)
24     :Displays "Invalid file name" message;
25 endif
26
27 |User|
28 stop
29
30 @enduml
```

```
1 @startuml
2
3 actor User
4
5 autonumber
6
7 participant HostSystem
8 participant NoteManager
9 participant VoiceManager
10 participant Voice
11 participant VoiceDataConsolidator
12 participant EffectsProcessingChain
13
14 User -> HostSystem: Key press
15 HostSystem -> NoteManager: Input data
16
17 note right NoteManager
18 Applies MIDI processing
19 end note
20
21 activate NoteManager
22 NoteManager -> VoiceManager: Note objects
23 deactivate NoteManager
24 activate VoiceManager
25
26 note right VoiceManager
27 Assigns Note objects to
28 idle Voice objects
29 end note
30
31 VoiceManager -> Voice: Note object
32 deactivate VoiceManager
33
34 par For each active Voice
35     activate Voice
36     note right Voice
37     Write audio data into
38     into buffers
39     end note
40     VoiceDataConsolidator <- Voice : Audio data
41     deactivate Voice
```



```
42     activate VoiceDataConsolidator
43 end
44
45 loop For every global clock time step
46     VoiceDataConsolidator -> EffectsProcessingChain:
        Summed audio data
47     deactivate VoiceDataConsolidator
48     activate EffectsProcessingChain
49 end
50
51 EffectsProcessingChain --> HostSystem: Processed
    audio data
52 deactivate EffectsProcessingChain
53
54 note over HostSystem
55 HostSystem converts audio data
56 into sound.
57 end note
58
59 HostSystem --> User
60
61 @enduml
62
63
```

```
1 @startuml
2
3 |User|
4 start
5 fork
6 :Changes Global Volume UI slider;
7 fork again
8
9 |Sampler|
10 :Listens for Global Volume UI changes;
11 :Updates internal global volume attribute;
12 :Notifies observer objects;
13 :Changes the displayed Global Volume value;
14 end fork
15
16 |Voice|
17 :Updates internal volume attribute;
18 :Applies volume value when generating audio data;
19
20 |SettingsUndoRedoManager|
21 :Tells Sampler to create memento;
22 :Stores memento in state history list;
23 note right: Save state for undo/redo
24
25
26 |User|
27 stop
28
29 @enduml
```

```
1 @startuml
2
3 |User|
4 start
5
6 :Clicks "Upload Sample" icon in GUI;
7
8 |Sampler|
9 :Presents file upload interface;
10
11 |User|
12 :Provides file to upload interface;
13
14 |Sampler|
15 :Performs file validation checks on file;
16 if (File is validated?) then (Yes)
17     :Copies file into application's sample library
    directory;
18     :Displays "File uploaded" message;
19
20 else (No)
21 :Displays "File upload error" with relevant
    validation error;
22 endif
23 |User|
24 :Closes upload interface;
25
26 |User|
27 stop
28
29 @enduml
```

```
1 @startuml
2
3 |User|
4 start
5 fork
6 :Presses keyboard;
7
8 |NoteManager|
9 fork again
10     :Listens for input;
11 end fork
12 :Receives input;
13 if (Is valid MIDI?) then (Yes)
14     :Applies MIDI effects processing chain;
15     :Generates Note objects;
16     :Sends Note objects to VoiceManager;
17 else (No)
18     :Display "MIDI input error";
19 endif
20
21 |User|
22 stop
23
24 @enduml
```

```
1 @startuml
2
3 actor User
4 participant HostSystem
5 participant NoteManager
6 participant MidiEffectChain
7 participant NoteFactory
8
9 autonumber
10
11 User -> HostSystem: Key press
12
13 HostSystem -> NoteManager: Input data
14 activate NoteManager
15
16 alt Input data is valid
17     NoteManager -> MidiEffectChain: Valid input data
18     activate MidiEffectChain
19 else Input data is invalid
20     NoteManager -> HostSystem: Error message
21 end
22
23 activate MidiEffectChain
24
25 note right of MidiEffectChain
26 Applies MIDI
27 processing to input data.
28 end note
29
30 MidiEffectChain --> NoteManager: Processed data
31 deactivate MidiEffectChain
32 NoteManager -> NoteFactory: Request for Note object
33 activate NoteFactory
34 NoteFactory --> NoteManager: Note objects
35 deactivate NoteFactory
36
37
38 @enduml
```

```
1 @startuml
2
3 |User|
4 fork
5 start
6 :Changes Global Root Pitch wheel on main UI;
7 fork again
8
9 |Sampler|
10 :Listens for Global Root Pitch UI changes;
11 :Notifies observer objects;
12 :Updates internal global root pitch attribute;
13 :Sends updated Global Root Pitch value to
    VoiceManager;
14 :Changes the displayed Global Root Pitch value;
15 end fork
16
17 |Voice|
18 :Updates internal pitch attribute;
19 :Applies pitch value when generating audio data;
20
21 |SettingsUndoRedoManager|
22 :Tells Sampler to create memento;
23 :Stores memento in state history list;
24 note right: Save state for undo/redo
25
26 |User|
27 stop
28 @enduml
```