```
@startuml

class Sampler {
    - globalVolume: int
    - presetFileDirectory: String
    - sampleLibraryDirectory: String
    - globalRootPitch: int
    - sampleFile: File
    - audioOutputLine: DataLine
    - settingsChangeObservers: SettingsChangeObserver
[0..*]
    + getInstance(): Sampler
    + getVolume() : int
    + getSample() : File
    + getGlobalRootPitch() : int
    + setVolume() : void
    + setGlobalRootPitch() : void
    + setSampleLibraryDirectory(libraryPath: String
) : void
    + loadSample(sampleFile: File) : void
    + createSettingsMemento() :
SamplerSettingsMemento
    + restoreSettings(previousSettings:
SamplerSettingsMemento) : void
    + notifyObserversOfSettingsChange() : void
}
note left: **Singleton**\nControls global and
administrative settings. Creates and\nrestores state
from SamplerSettingsMemento.\nProvides audio data to
system audio data line.


interface MidiEffectChain {
    process(note: MidiMessage) : MidiMessage
}


class BasicMidiGenerator implements MidiEffectChain {
    + process(note: MidiMessage) : MidiMessage
}

```

```

class MidiDecorator implements MidiEffectChain {
    - wrappedMidiEffect : MidiEffectChain
    + MidiDecorator(midiEffect: MidiEffectComponent) : MidiDecorator
    + process(note: MidiMessage) : MidiMessage
}
note top: **Decorator**\nObjects receive MidiMessages and apply\nprocessing before providing the processed\ndata to the next decorated object.


class RandomizePitch extends MidiDecorator {
    + process(note: MidiMessage) : MidiMessage
}


class RandomizeVolume extends MidiDecorator {
    + process(note: MidiMessage) : MidiMessage
}


class Arpeggiator extends MidiDecorator {
    + process(note: MidiMessage) : MidiMessage
}


class NoteFactory {
    - notes: String
'    - midiEffectChain: MidiEffectChain
    + getNote(note: MidiMessage): Note
}
note bottom: **Flyweight**\nCreates Note objects based on \nuser input to the system


class Note {
    - pitch: int
    - velocity: int
    + getPitch(): int
    + getVolume(): int
}
note right: Encapsulates user input from key\npresses as pitch and volume values.


class NoteManager {
    - inputLine : Scanner
    - noteFactory: NoteFactory
    - noteBuffer: Note[]
    - effectsChain: MidiEffectChain
    + getNoteBuffer : Note[]
    + isValidInput(input : SystemInput) : boolean
}
note bottom: Receives input messages from System.\nGets processed data from MidiEffectChain\nand requests Note objects from NoteFactory.


class Voice implements SettingsChangeObserver {
    - level: int
    - rootPitch: int
    - isIdle: boolean
    - voiceAudioDataBuffer: byte[]
    - voiceDataConsolidator: VoiceDataConsolidator
    + generateAudioData(note: Note) : void
    + writeToBuffer(): void
    + onSettingsChange() : void
}
note left of Voice: Writes audio data to VoiceDataConsolidator\n buffers. Observes Sampler for changes to pitch and volume.


class VoiceManager {
    - voicePool: Voice[1..4]
    - noteManager: NoteManager
    + delegateNoteToAvailableVoice(note: Note): void
    + createVoice(): void
}
note right: Creates Voice objects.\nGets Note objects from NoteManager and \ndistributes them to
```

```
idle Voice objects.


class VoiceDataConsolidator {
    - voiceAudioDataBuffers: byte[1..4][ ]
    - effectsChain: AudioEffect
    + getSummedData() : byte[]
    + sumVoiceAudioDataBuffers() : byte[]
    + sendAudioToAudioEffectChain(effectChain: AudioEffect) : void
}
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.


class Sample {
    - file: File
    - fileType: String
    - filePath: String
    + getFile(): File
}


class AudioEffectChain {
    - effectChain: AudioEffect [0..*]
    - settingsChangeObservers: SettingsChangeObserver [0..*]
    - effectFactory: AudioEffectFactory
    - sampler: Sampler
    + addEffect(effectType: String): void
    + removeEffect(effectType: String): void
    + rebuildEffectChain(): void
    + createSettingsMemento() : EffectsChainSettingsMemento
    + restoreSettings(previousSettings: EffectsChainSettingsMemento) : void
    + notifyObserversOfSettingsChange() : void
}
note right: Maintains an effects chain.\nCreates and restores state from\nEffectsChainSettingsMemento.
```

```


interface AudioEffect {
    setNextEffect(nextEffect: AudioEffect) : void
    processAudio(audioDataBuffer: byte []) : byte []
}
note right: Interface for AudioEffectProcess chain of responsibility.


class AudioEffectProcess implements AudioEffect, SettingsChangeObserver {
    - nextEffect: AudioEffect
    - audioOutputLine: DataLine
    + setNextEffect(nextEffect: AudioEffect) : void
    + processAudio(audioDataBuffer: byte []) : byte []
    +onSettingsChange() : void

}
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.
note right of AudioEffectProcess: These objects apply processing to\nthe audio data they're passed.


interface AudioEffectFactory {
    + createAudioEffect(effectType: String) : AudioEffect
}
note right: **Factory**\nInterface for creation of AudioEffect objects.


class ConcreteAudioEffectFactory implements AudioEffectFactory {
    + createAudioEffect(effectType: String) : AudioEffect
```

```
}
note right: Contains logic for creation of AudioEffect objects.


class Reverb extends AudioEffectProcess {
    + processAudio(audioDataBuffer: byte []) : byte []
}


class Delay extends AudioEffectProcess {
    + processAudio(audioDataBuffer: byte []) : byte []
}


class Chorus extends AudioEffectProcess {
    + processAudio(audioDataBuffer: byte []) : byte []
}


interface SettingsChangeObserver {
    + onSettingsChange() : void
}
note right: **Observer**


class SamplerSettingsMemento {
    - sampleFilePath: String
    - globalVolume: int
    - globalRootPitch: int
    - numberOfVoices: int
}
note left: **Memento**


class EffectsChainSettingsMemento {
    - audioEffectChain; String[0..*]
    - midiEffectChain; String[0..*]
```

```
}
note left: **Memento**


class ConsolidatedSettings {
    - samplerSettings: SamplerSettingsMemento
    - effectsChainSettings: EffectsChainSettingsMemento
}


class SettingsUndoRedoManager {
    - originatorSampler: Sampler
    - originatorEffects: AudioEffectChain
    - stateHistory: ConsolidatedSettings [0..*]
    - currentStateIndex: int
    + onSettingsChange() : void
    + undoState() : void
    + redoState() : void
}
note right: Manages list of previous system states and\ntells Sampler and AudioEffectChain when to\nrestore a state.


Sample "1" -o "1" Sampler
Sample <-- Voice

'Note and Voice
NoteFactory "1" --* "1" NoteManager
Note "1..128" --o "1" NoteFactory
VoiceManager o-- NoteManager
Voice "1..4" ---* "1" VoiceManager
Voice ---> Note
Voice o--> VoiceDataConsolidator

'SettingsUndoRedoManager
SamplerSettingsMemento <-- Sampler
ConsolidatedSettings "0..*" --* "1" SettingsUndoRedoManager
SamplerSettingsMemento "1" --o "1"
```

```
ConsolidatedSettings
EffectsChainSettingsMemento "1" --o "1" ConsolidatedSettings
SettingsUndoRedoManager ..|> SettingsChangeObserver
SettingsUndoRedoManager --> AudioEffectChain
Sampler <-- SettingsUndoRedoManager
Sampler "1" o-- "0..*" SettingsChangeObserver
EffectsChainSettingsMemento <-- AudioEffectChain
SettingsChangeObserver "0..*" --o "1" AudioEffectChain

'Audio and MIDI effects
AudioEffect "0..*" --o "1" AudioEffectChain
AudioEffect <---o  AudioEffectProcess
VoiceDataConsolidator o---> AudioEffect
AudioEffectChain o-- AudioEffectFactory
ConcreteAudioEffectFactory -> AudioEffect
MidiEffectChain --o NoteManager

@enduml
```

```
@startuml
|User|
start

:Enters text in search field;

|Sampler|
:Searches library directory for matches;
if (Matches found?) then (Yes)
    :Displays matches in results field;

    |User|
    if (Selects an effect?) then (Yes)

        |Sampler|
        if (Is audio or MIDI effect?) then (Audio)

            |AudioEffectChain|
            :Add effect to audio processing chain;

        |Sampler|
        else (MIDI)

            |NoteManger|
            :Add effect to MIDI processing chain;
        endif

        |SettingsUndoRedoManager|
        :Tells AudioEffectChain to create memento;
        :Stores memento in state history;
        note right: Save state for undo/redo

    |Sampler|
    else (No)
    endif

|Sampler|
else (No)
    :Display "No matches found" message to User;
endif

```

```
|User|
stop

@enduml
```

```
@startuml

|User|
start
:Enters text in search field;

|Sampler|
:Searches library directory for matches;
if (Matches found?) then (Yes)
    :Displays matches in results field;

    |User|
    if (Selects a sample?) then (Yes)

        |Sampler|
        if (Different file is already loaded?) then (
Yes)
            :Display "Load new sample?" message;

            |User|
            if (Confirms "Yes"?) then (Yes)

                |Sampler|
                :Perform selection validation;
                if (Selection is valid?) then (Yes)
                    :Load file into sampler
application and readies it for playback;
                    :Displays "Sample loaded" message
;
                else (No)
                    :Display "Invalid file" message;
                endif;

            |User|
            else (No)

                |Sampler|
                :Close dialogue;
                :Displays matches in results field;

'             |User|
            endif
        |Sampler|
        else (No)
            :Perform selection validation;
            if (Selection is valid?) then (Yes)
                :Load file into sampler application
and readies it for playback;
                :Displays "Sample loaded" message;
            else (No)
                :Display "Invalid file" message;
            endif;
        endif

'    |User|
    else (No)
    endif

|Sampler|
else (No)
    :Display "No matches found" message to User;
endif

|User|
stop

@enduml
```

```
@startuml

|User|
start
:Presses a key on computer keyboard or MIDI device;

|Sampler|
if (Sample is loaded?) then (Yes)

    |NoteManager|
    :Process MIDI input;
    :Send Notes to VoiceManager;

    |VoiceManager|
    :Send note values to an available Voice object;

    |Voice|
floating note right: Per voice parallel processing
    fork
        :Generate audio data;
    fork again
        :Generate audio data;
    fork again
        :Generate audio data;
    fork again
        :Generate audio data;
    end fork

    |VoiceDataConsolidator|
    :Process audio data;

    |AudioEffectsChain|
    :Process audio data;
    :Send processed audio data to output line;

    |Sampler|

else (No)
endif

|User|
stop


@enduml


```

```
@startuml


|User|
start
:Clicks "Save Preset" button on main UI;

|PresetManager|
:Opens "Save Preset" dialogue box;

|User|
:Enters name for preset;
:Clicks "Save";

|PresetManager|
if (Is valid file name?) then (Yes)
    if (No PresetMemento in preset list?) then (Yes)
        :Creates PresetMemento object with current state;
    else (no)
    endif
    :Save last PresetMemento as preset values into a file in preset directory;
    :Notifies User that preset has been saved;
else (no)
    :Displays "Invalid file name" message;
endif

|User|
stop

@enduml
```

```
@startuml

actor User

autonumber

participant HostSystem
participant NoteManager
participant VoiceManager
participant Voice
participant VoiceDataConsolidator
participant EffectsProcessingChain

User -> HostSystem: Key press
HostSystem -> NoteManager: Input data

note right NoteManager
Applies MIDI processing
end note

activate NoteManager
NoteManager -> VoiceManager: Note objects
deactivate NoteManager
activate VoiceManager

note right VoiceManager
Assigns Note objects to
idle Voice objects
end note

VoiceManager -> Voice: Note object
deactivate VoiceManager

par For each active Voice
    activate Voice
    note right Voice
    Write audio data into
    into buffers
    end note
    VoiceDataConsolidator <- Voice : Audio data
    deactivate Voice
    activate VoiceDataConsolidator
end

loop For every global clock time step
    VoiceDataConsolidator -> EffectsProcessingChain: Summed audio data
    deactivate VoiceDataConsolidator
    activate EffectsProcessingChain
end

EffectsProcessingChain --> HostSystem: Processed audio data
deactivate EffectsProcessingChain

note over HostSystem
HostSystem converts audio data
into sound.
end note

HostSystem --> User

@enduml


```

```
@startuml

|User|
start
fork
:Changes Global Volume UI slider;
fork again

|Sampler|
:Listens for Global Volume UI changes;
:Updates internal global volume attribute;
:Notifies observer objects;
:Changes the displayed Global Volume value;
end fork

|Voice|
:Updates internal volume attribute;
:Applies volume value when generating audio data;

|SettingsUnoRedoManager|
:Tells Sampler to create memento;
:Stores memento in state history list;
note right: Save state for undo/redo


|User|
stop

@enduml
```

```
@startuml

|User|
start

:Clicks "Upload Sample" icon in GUI;

|Sampler|
:Presents file upload interface;

|User|
:Provides file to upload interface;

|Sampler|
:Performs file validation checks on file;
if (File is validated?) then (Yes)
    :Copies file into application's sample library directory;
    :Displays "File uploaded" message;

else (No)
:Displays "File upload error" with relevant validation error;
endif
|User|
:Closes upload interface;

|User|
stop

@enduml
```

```
@startuml

|User|
start
fork
:Presses keyboard;

|NoteManager|
fork again
    :Listens for input;
end fork
:Receives input;
if (Is valid MIDI?) then (Yes)
    :Applies MIDI effects processing chain;
    :Generates Note objects;
    :Sends Note objects to VoiceManager;
else (No)
    :Display "MIDI input error";
endif

|User|
stop

@enduml
```

```
@startuml

actor User
participant HostSystem
participant NoteManager
participant MidiEffectChain
participant NoteFactory

autonumber

User -> HostSystem: Key press

HostSystem -> NoteManager: Input data
activate NoteManager

alt Input data is valid
    NoteManager -> MidiEffectChain: Valid input data
    activate MidiEffectChain
else Input data is invalid
    NoteManager -> HostSystem: Error message
end

activate MidiEffectChain

note right MidiEffectChain
Applies MIDI
processing to input data.
end note

MidiEffectChain --> NoteManager: Processed data
deactivate MidiEffectChain
NoteManager -> NoteFactory: Request for Note object
activate NoteFactory
NoteFactory --> NoteManager: Note objects
deactivate NoteFactory


@enduml
```

```
@startuml

|User|
fork
start
:Changes Global Root Pitch wheel on main UI;
fork again

|Sampler|
:Listens for Global Root Pitch UI changes;
:Notifies observer objects;
:Updates internal global root pitch attribute;
:Sends updated Global Root Pitch value to
VoiceManager;
:Changes the displayed Global Root Pitch value;
end fork

|Voice|
:Updates internal pitch attribute;
:Applies pitch value when generating audio data;

|SettingsUnoRedoManager|
:Tells Sampler to create memento;
:Stores memento in state history list;
note right: Save state for undo/redo

|User|
stop
@enduml
```