Project 1 Design Document

Team Members: Ryan Liu, John Wang, Kevin White

Grammar

Terminals:

AbstractNote(pitch: Pitch, duration: double, accidentals: Integer)
Rest(duration: Integer)
StartRepeat(measureNumber:Integer)
EndRepeat(measureNumber:Integer)
EndMajorSection(measureNumber:Integer)
FirstEnding(measureNumber:Integer)
SecondEnding(measureNumber:Integer)

Non-Terminals:

Piece::= Voice+
Voice::= (Measure| Repeat|Ending)+
Measure ::= Playable+
Playable ::= Phrase|Rest
Phrase ::= Chord |Tuple | SingleNote
Chord ::= AbstractNote{2,}
Tuple ::= AbstractNote{2,4}
SingleNote ::= AbstractNote
Repeat ::= StartRepeat|EndRepeat|EndMajorSection
Ending ::= FirstEnding|SecondEnding

Definitions:

Abstract Note: An AbstractNote object represents a single note with fields pitch, duration, and accidentals.

SingleNote: SingleNote objects represent a phrase of a single note with a single abstract note as a class.

Chord: Chord objects represent a musical chord and has a field called noteList that is a list of AbstractNotes.

Tuple: Tuple objects represent duplets, triplets, or quadruplets and have a field noteList that is a list of AbstractNotes in the tuple.

Phrase: Phrase is an interface that allows for the implementation of abstract notes within classes. Classes that implement Phrase are SingleNote, Chord, and Tuple.

Rest: Rest objects signify a rest in music and contain a duration field.

Measure: Measure objects are the building blocks of voices: its field contains a list of playable objects.

Playable: Playable is an abstract class that is extended by Phrase and Rest. It signifies a collection of notes, a single note, or a rest.

Repeat: Repeat is an interface that encapsulates all objects related to repeats. StartRepeat, EndRepeat, and EndMajorSection implement the interface Repeat.

StartRepeat: StartRepeat objects in a voice indicate a left repeat sign.

EndRepeat: EndRepeat objects in a voice indicate a right repeat sign.

EndMajorSection: An EndMajorSection object is an indicator that identifies that a major portion of the piece has completed. This can also be used as a StartRepeat object in repeat-handling.

Ending: Ending is an interface encapsulating information regarding first and second endings in music. FirstEnding and SecondEnding implement Ending.

FirstEnding: FirstEnding objects indicate that the next measure will be a first ending.

SecondEnding: SecondEnding objects indicate that the next measure will be a second ending.

Voice: Voice objects signify a single voice in a piece and have fields measureList, endingsList, and repeatList which are a list of Measures, Ending Objects, and Repeat Objects respectively.

Piece: Piece objects signify an entire piece, composed of a field voiceList that is a list of Voices in the piece.

Datatypes

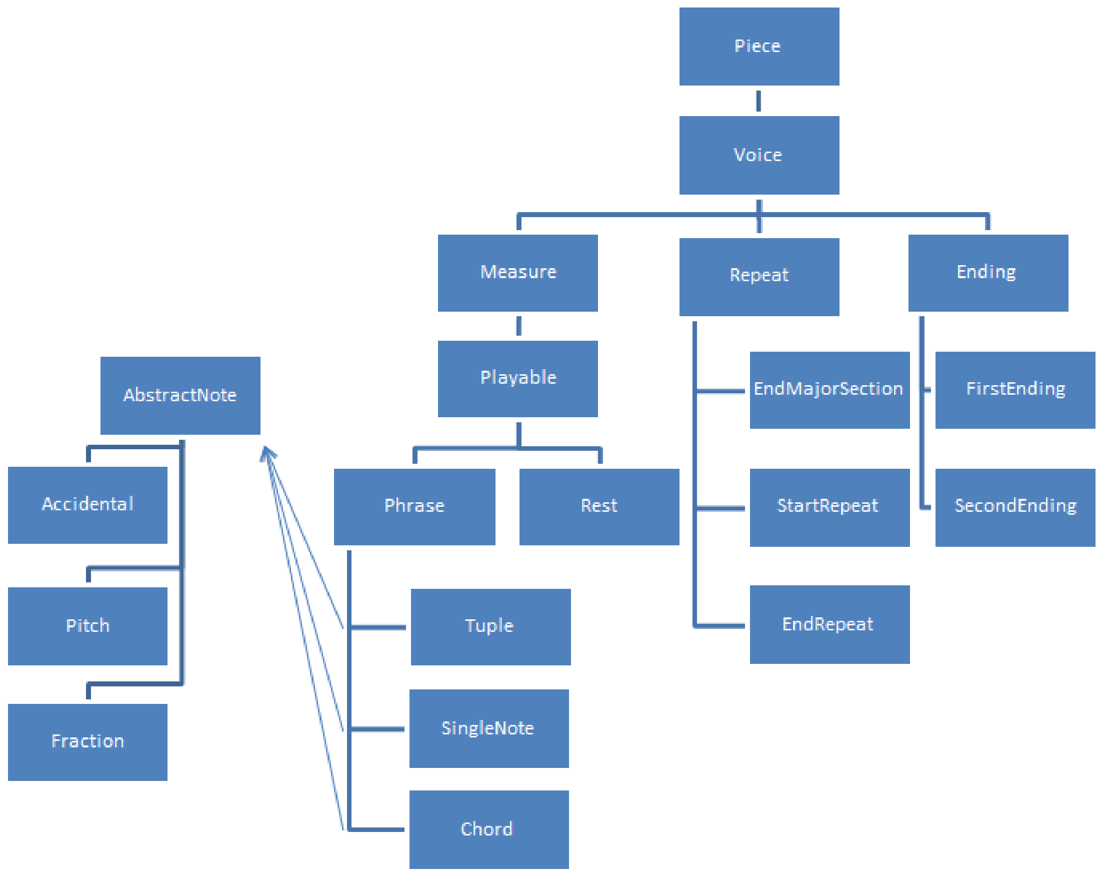


Figure 1. Datatype Hierarchy

The datatypes for the project have been designed in an attempt to model actual music. Each piece consists of multiple voices, which consist of measure, repeat, and ending objects. These objects are designed so that playing the piece will only require a call to a single method, which will then make calls to the requisite music object.

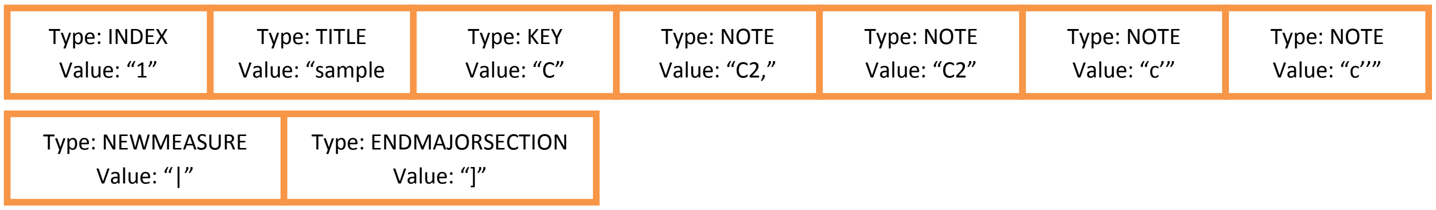
The voice object will contain a list of measures, repeats, and endings. Based on the type of repeat object, the voice object will organize measures into an immutable list. Each measure will contain a series of phrases and rests. These phrases will then be decomposed into note objects, such as Tuple, SingleNote, and Chord which can be concatenated into an ordered list. Once this ordered list of notes is created, they will be passed into the music sequencer, and the piece will be played.

Each of the objects that implements the Phrase interface will contain a list of AbstractNotes. These objects are the basis for a note, and contain a single pitch, the length of the pitch, and any accidentals on that pitch. The entire piece is therefore composed implicitly of a list of AbstractNotes. One can use this to construct a list of PlayedNotes using the getNotes() method in AbstractNotes. Finally, one can use the getMidi() method of the PlayedNotes class to get all the midi notes to pass to the sequencer.

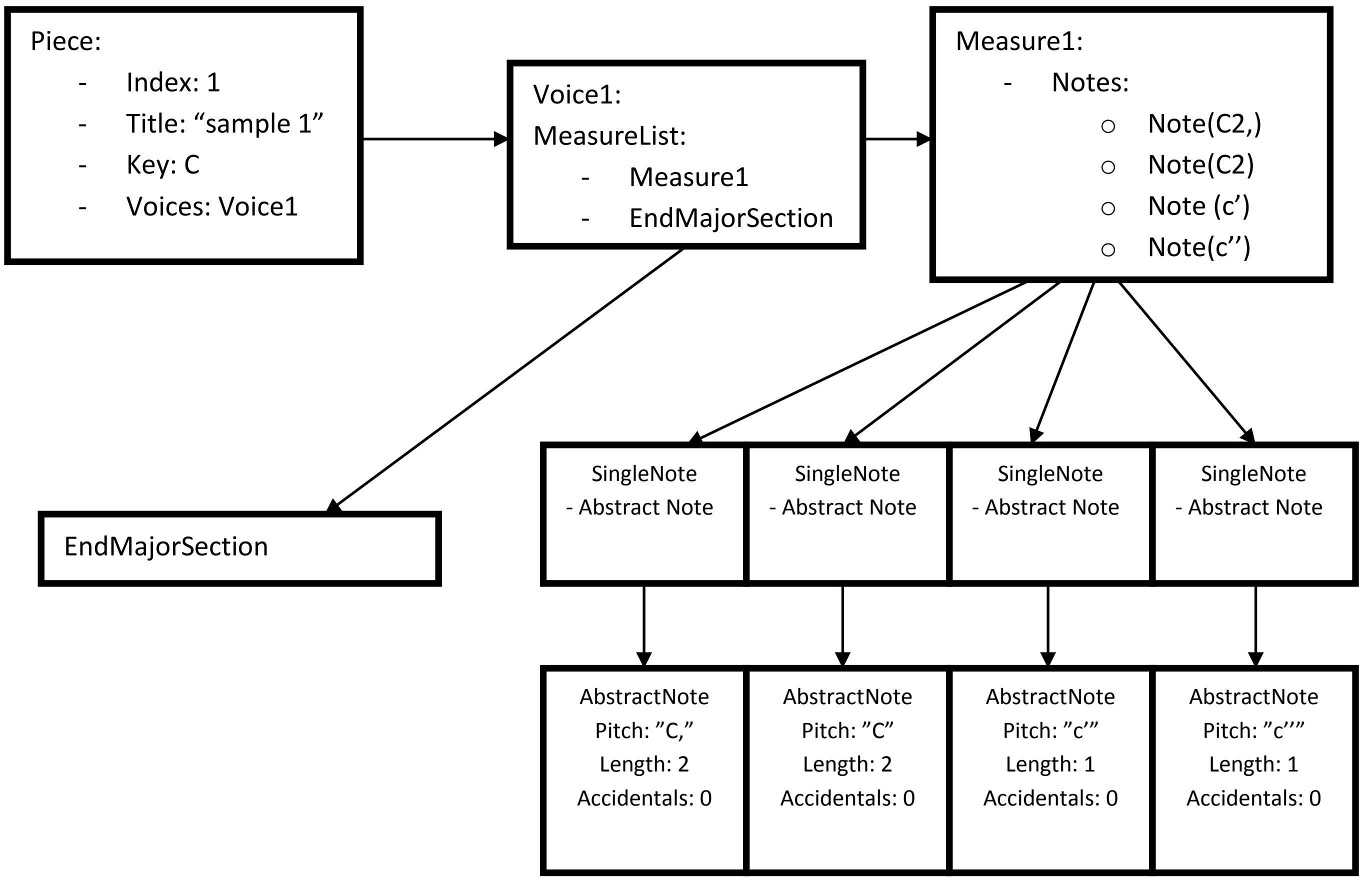
Snapshot Diagrams

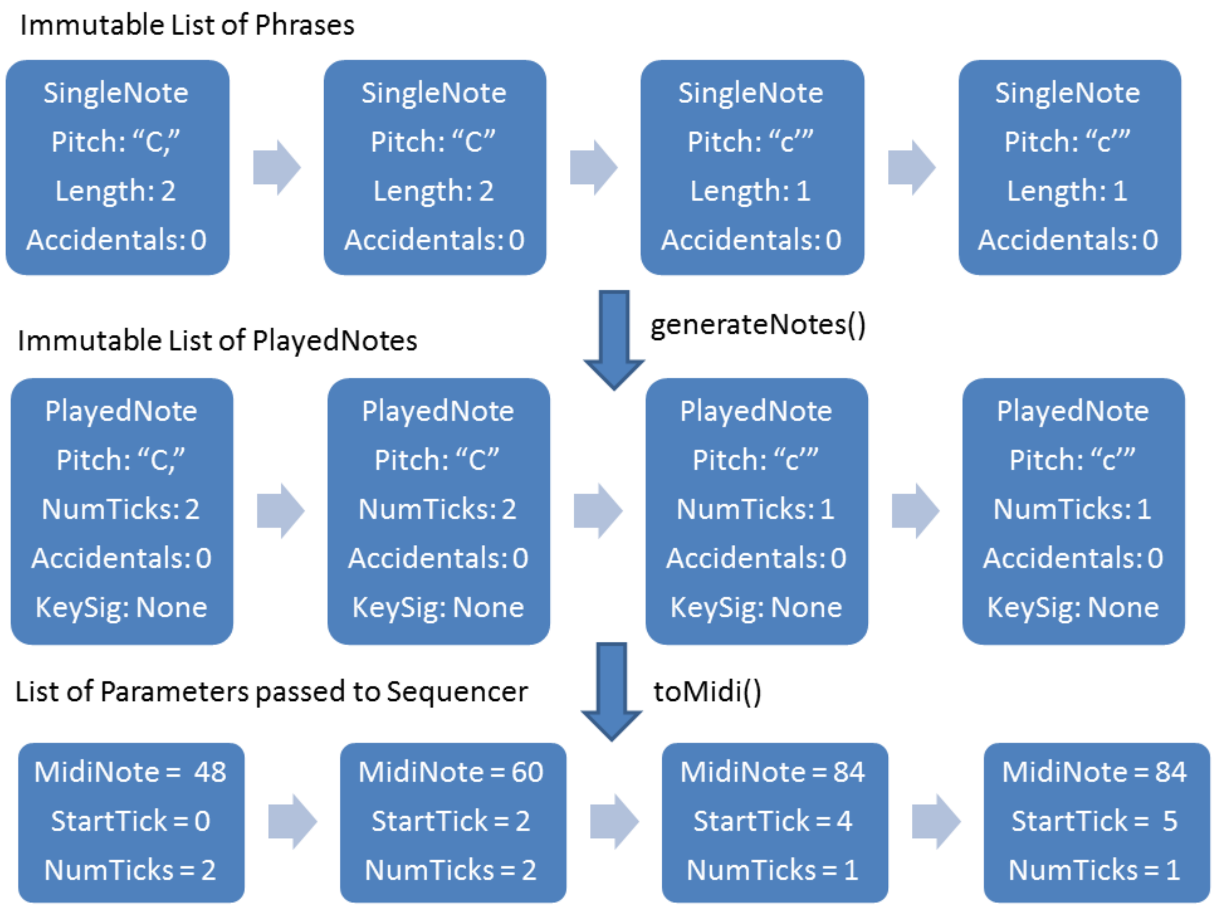
```
Sample1.abc

X:1
T:sample 1
K:C
C2, C2 c' c''|]
```

Lexer: List of Tokens



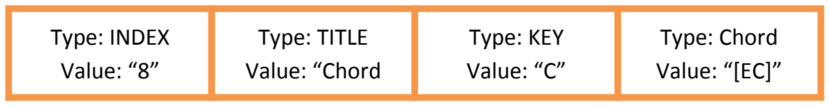
Parser: Each box represents an object



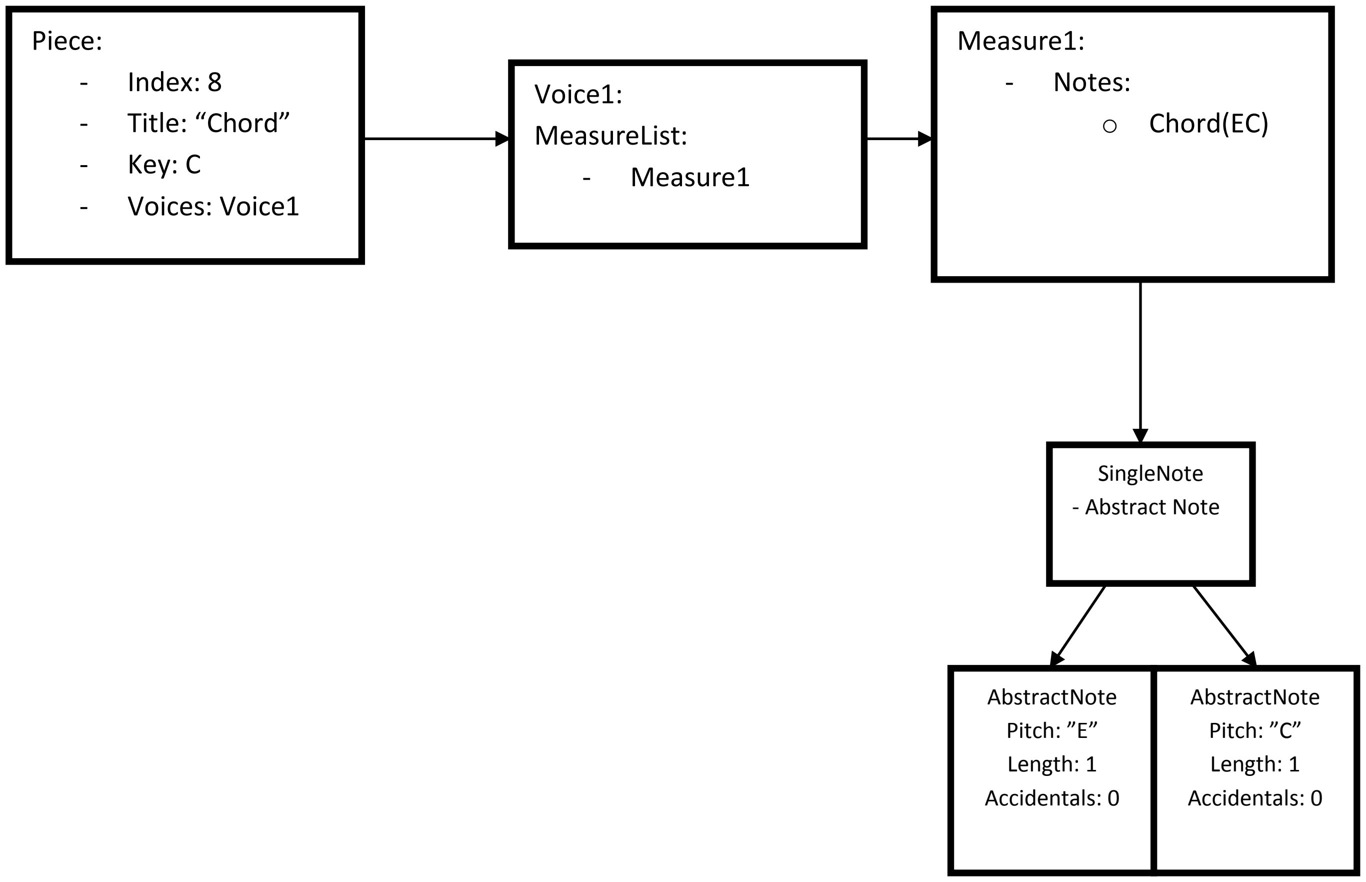

Immutable List of Phrases
SingleNote Pitch: “C,” Length: 2 Accidentals: 0
SingleNote Pitch: “C” Length: 2 Accidentals: 0
SingleNote Pitch: “c’” Length: 1 Accidentals: 0
SingleNote Pitch: “c’” Length: 1 Accidentals: 0
generateNotes()
Immutable List of PlayedNotes
PlayedNote Pitch: “C,” NumTicks: 2 Accidentals: 0 KeySig: None
PlayedNote Pitch: “C” NumTicks: 2 Accidentals: 0 KeySig: None
PlayedNote Pitch: “c’” NumTicks: 1 Accidentals: 0 KeySig: None
PlayedNote Pitch: “c’” NumTicks: 1 Accidentals: 0 KeySig: None
toMidi()
List of Parameters passed to Sequencer
MidiNote = 48 StartTick = 0 NumTicks = 2
MidiNote = 60 StartTick = 2 NumTicks = 2
MidiNote = 84 StartTick = 4 NumTicks = 1
MidiNote = 84 StartTick = 5 NumTicks = 1

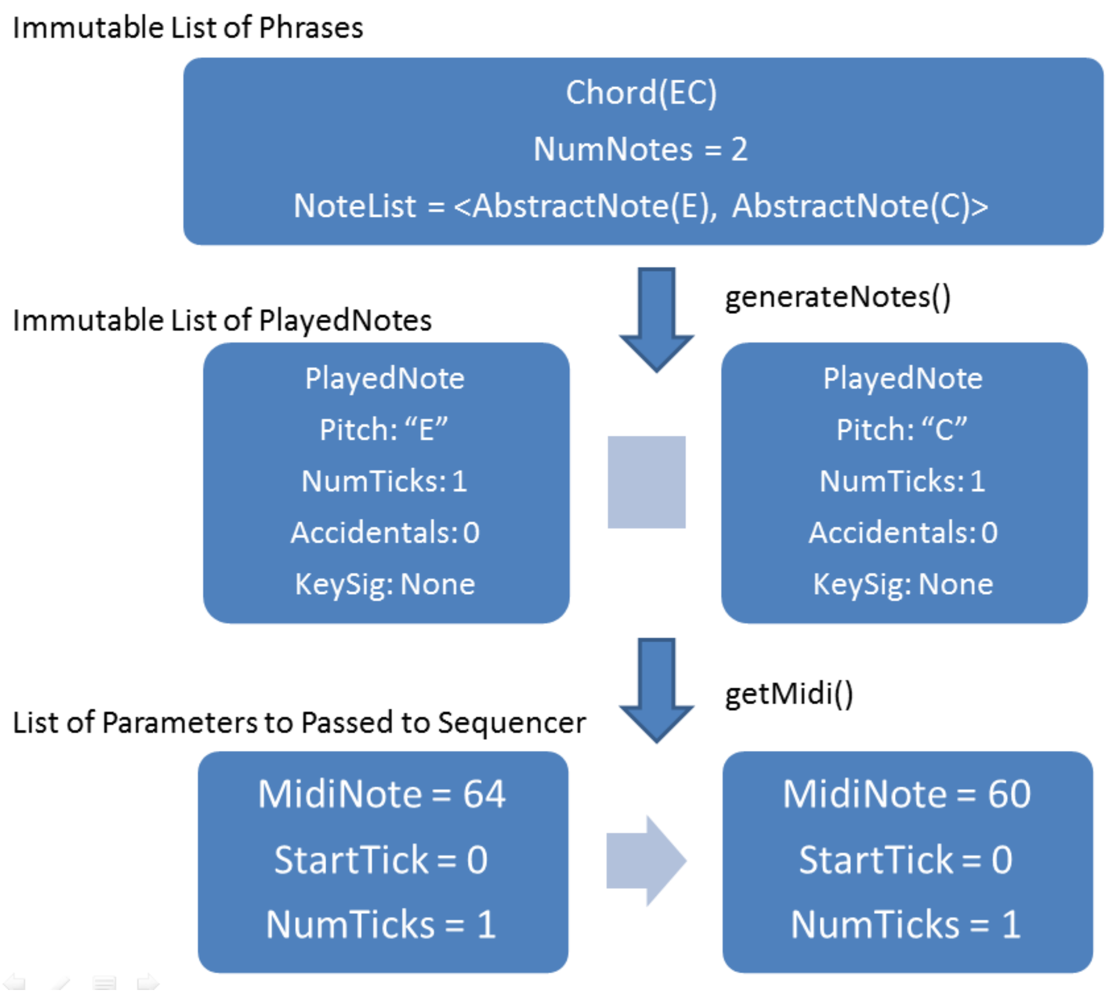
```
Sample2.abc

X:8
T:Chord
K:C
[EC]
```

Lexer:



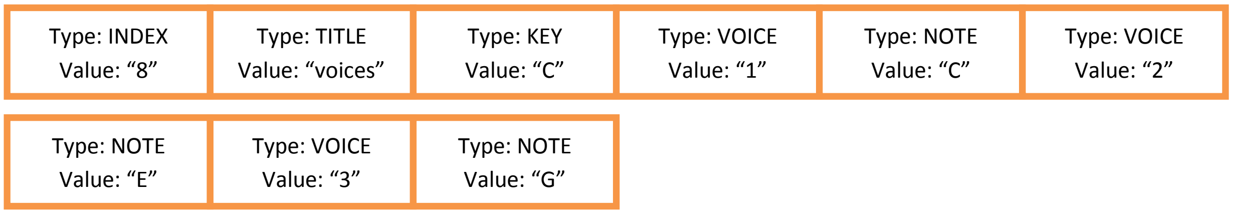
Parser:



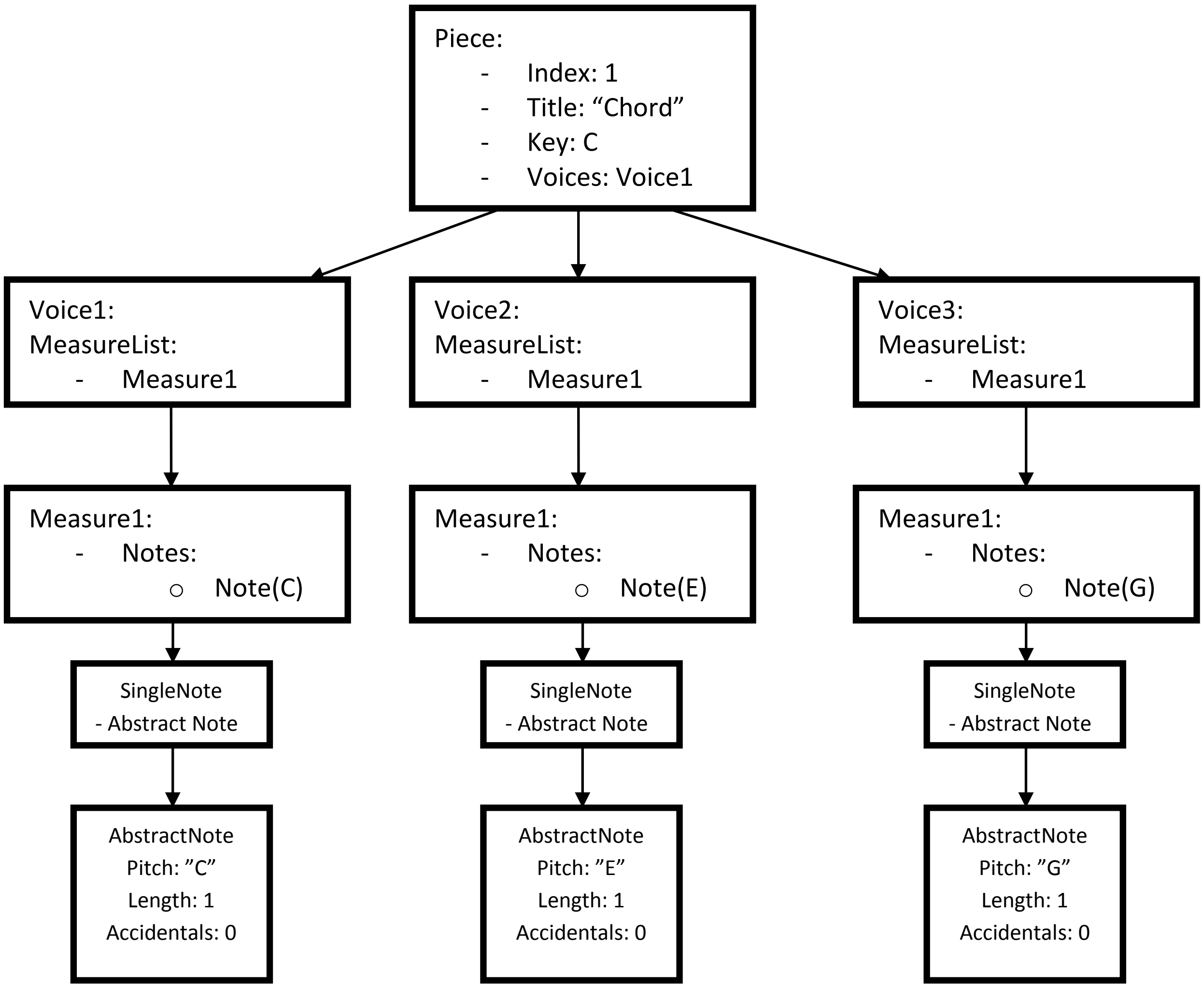

Immutable List of Phrases
Chord(EC)
NumNotes = 2
NoteList = <AbstractNote(E), AbstractNote(C)>
generateNotes()
Immutable List of PlayedNotes
PlayedNote
Pitch: “E”
NumTicks: 1
Accidentals: 0
KeySig: None
PlayedNote
Pitch: “C”
NumTicks: 1
Accidentals: 0
KeySig: None
getMidi()
List of Parameters to Passed to Sequencer
MidiNote = 64
StartTick = 0
NumTicks = 1
MidiNote = 60
StartTick = 0
NumTicks = 1

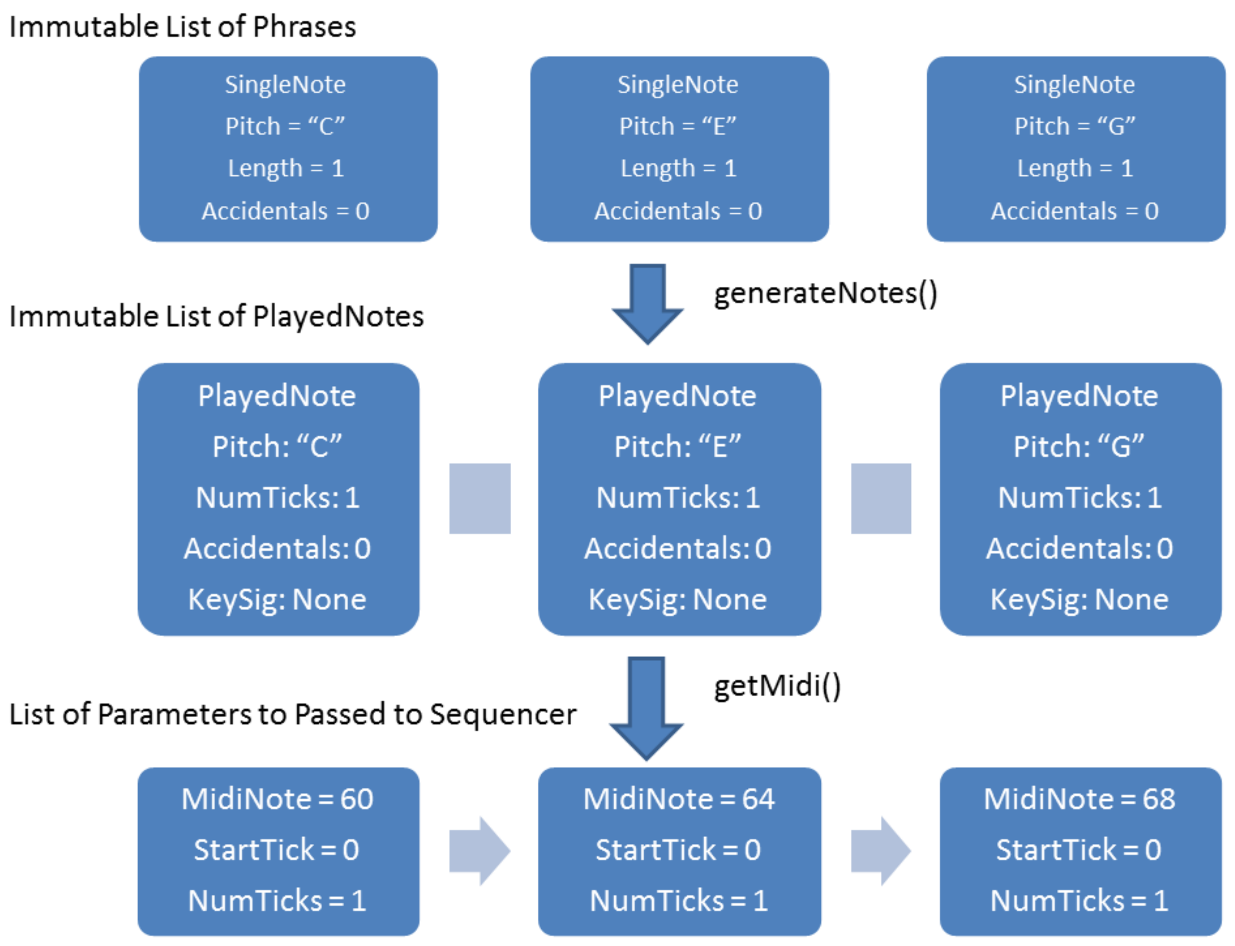
```
Sample3.abc

X:1
T:voices
K:Cm

V: 1
C
V: 2
E
V: 3
G
```

Lexer:



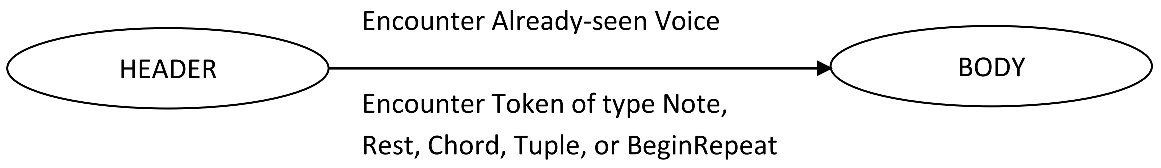
Parser:




Immutable List of Phrases
SingleNote
Pitch = “C”
Length = 1
Accidentals = 0
SingleNote
Pitch = “E”
Length = 1
Accidentals = 0
SingleNote
Pitch = “G”
Length = 1
Accidentals = 0
generateNotes()
Immutable List of PlayedNotes
PlayedNote
Pitch: “C”
NumTicks: 1
Accidentals: 0
KeySig: None
PlayedNote
Pitch: “E”
NumTicks: 1
Accidentals: 0
KeySig: None
PlayedNote
Pitch: “G”
NumTicks: 1
Accidentals: 0
KeySig: None
getMidi()
List of Parameters to Passed to Sequencer
MidiNote = 60
StartTick = 0
NumTicks = 1
MidiNote = 64
StartTick = 0
NumTicks = 1
MidiNote = 68
StartTick = 0
NumTicks = 1

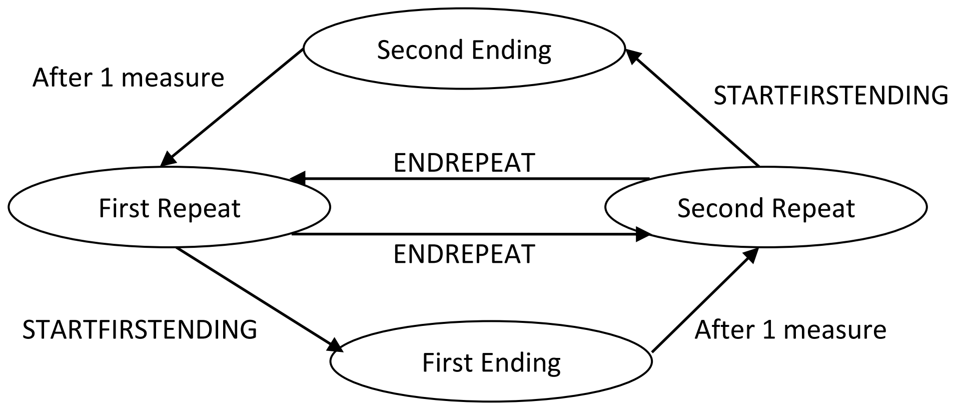
State Machines

Parser


Encounter Already-seen Voice
HEADER
BODY
Encounter Token of type Note,
Rest, Chord, Tuple, or BeginRepeat

Above is a simple state machine representing our parser. Initially, our parser will start in the “HEADER” state, where it will treat the tokens as if they were headers (composer, title, tempo, etc.). Then, after encountering an already-seen voice (which will happen if all voices are accounted for in the header) or seeing any token with type Note, Rest, Chord, Tuple, or BeginRepeat, it will transition to the state “BODY,” where it will treat tokens as if they were in the body of the music.

Voice.getNotes()



The above diagram shows a state machine for how we will implement repeats. Our initial state will be “First Repeat,” which means that the piece is playing this section for the first time. If we see a repeat sign, then we will switch to the second repeat sign, since the piece will be playing this section for the 2nd time. Another way to transition to the second repeat is by encountering a STARTFIRSTENDING object, and playing through that measure.

If the state machine is in the “Second Repeat” state, it can get back to the “First Repeat” state by either encountering another ENDREPEAT object, or by via the “Second Ending.” This temporary state “Second Ending” is reached when the machine encounters a STARTFIRSTENDING object, since when that object is encountered, the state machine will automatically skip to the second ending.