ABC Player Project
Design Document

Sarah Edris, Rachael Naphtal, Ranna Zhou

6.005 Spring 2012

GRAMMARS HANDLED BY LEXER

Header

```
header-field            ::= field-number | field-title | field-composer | field-default-length | field-meter |
                                  field-tempo | field-key
field-voice             ::= "V:" text end-of-line
```

Music

```
mid-tune-field          ::= field-voice
note                    ::= [accidental] basenote [octave] [note-length]
rest                    ::= "z" [note-length]
chord                   ::= "[" (note | rest)+ "]"
tuplet                  ::= "(" DIGIT (note | rest)+
barline                 ::= "|" | "||" | "[|" | "|]" | ":|" | "|:"
nth-repeat              ::= "[1" | "[2"
comment                 ::= "%" text linefeed
```

GRAMMARS HANDLED BY PARSER

Header

```
field-number            ::= "X:"DIGIT+
field-title             ::= "T:" text
field-composer          ::= "C:" text
field-default-length    ::= "L:" DIGIT+"/"DIGIT+
field-meter             ::= "M:" "C"|"C|" | DIGIT+"/"DIGIT+
field-tempo             ::= "Q:" DIGIT+
field-key               ::= "K:"  basenote ["#"|"b"]["m"]
field-voice             ::= "V:" text end-of-line
```

Music

```
field-voice             ::= "V:" text end-of-line
element                 ::= note | rest | chord | tuplet | barline | nth-repeat
accidental              ::= "^" | "^^" | "_" | "__" | "="
basenote                ::= "C" | "D" | "E" | "F" | "G" | "A" | "B" | "c" | "d" | "e" | "f" | "g" | "a" | "b"
note-length             ::= [DIGIT+] ["/" [DIGIT+]]
octave                  ::= ("'"+) | (","+)
barline                 ::= "|" | "||" | "[|" | "|]" | ":|" | "|:"
nth-repeat              ::= "[1" | "[2"
```

DATATYPE DEFINITIONS
ABCPlayer = ABCPlayer (fileName: String)

ABCPlayer.play()
The ABCPlayer is the overarching class which handles playing ABC files. It has the capacity to read in the files, send them to be tokenized then parsed, and then ultimately play them.

ABCLexer = matcher: Matcher + abcString: String
The ABCLexer stores the matcher used to match the file contents to our ABC regexes and the abcString that holds the entire file contents. In the lex method, we use Java's Pattern class and capturing groups in regexes to recognize the various parts of an ABC file and then return a list of Tokens for the parser.

ABCParser = ABCParser (tokens: List<Token>)
The Parser is the class which handles parsing the given Tokens down to their basic parts, and uses these to create MusicElements and build the ABCTune.

ABCTune = header: Header + music: Music
ABCTune is the overall class to store a given piece of music. It contains the header and the music.

Header = Header (number: int, title: String, key: KeySig, defaultLength: Fraction, meter: Fraction, tempo: int, voiceNames: String[])
The Header is the class which contains information on the header written for the piece of music. These fields persist throughout the piece of music and are thus immutable. Some fields are optional and default to listed things (if not given) when constructed.

Fraction = Fraction (numerator: int, denominator: int)
Multiple fields require fractions, however to avoid rounding errors by making them doubles, this class is used to store the fractions and can manipulate them later. We also utilize the least common denominator method in order to compute the number of ticks to assign to a quarter note.

KeySig = KeySig(keyNote: char, keyAccidental: int, modeMinor: boolean)
The key signature keeps track of

Music = ABCMusic (voices: HashMap<String, Voice>)
Music holds the HashMap of voices which links voice names to their Voice object. When the music section of the file is read, it makes the voices and adds them to this class. Ultimately each of the voices is able to be played.

interface MusicElement()
This is the general interface for storing all musical elements. All elements have a toString method and an accept method when given a Visitor (described below). This is useful as more methods can be added easily using the visitor implementation and they can be recursive.

```
public void toString();
public void accept(Visitor v);
public List<Note> getAllNotes();
public void divideNoteDuration (int type);
```

Voice = Voice (measures: List<Measure>)
This is the class which contains the actual measures which make up the music that a Voice plays converts.

Measure = Measure (elements: List<MusicElement>)
This class represents a measure in the tune. After the parser reads the lines of music, it converts them into elements and adds them into a list of elements which make up the measure. Repeats will cause measures to be added twice to the list of measures in voices.

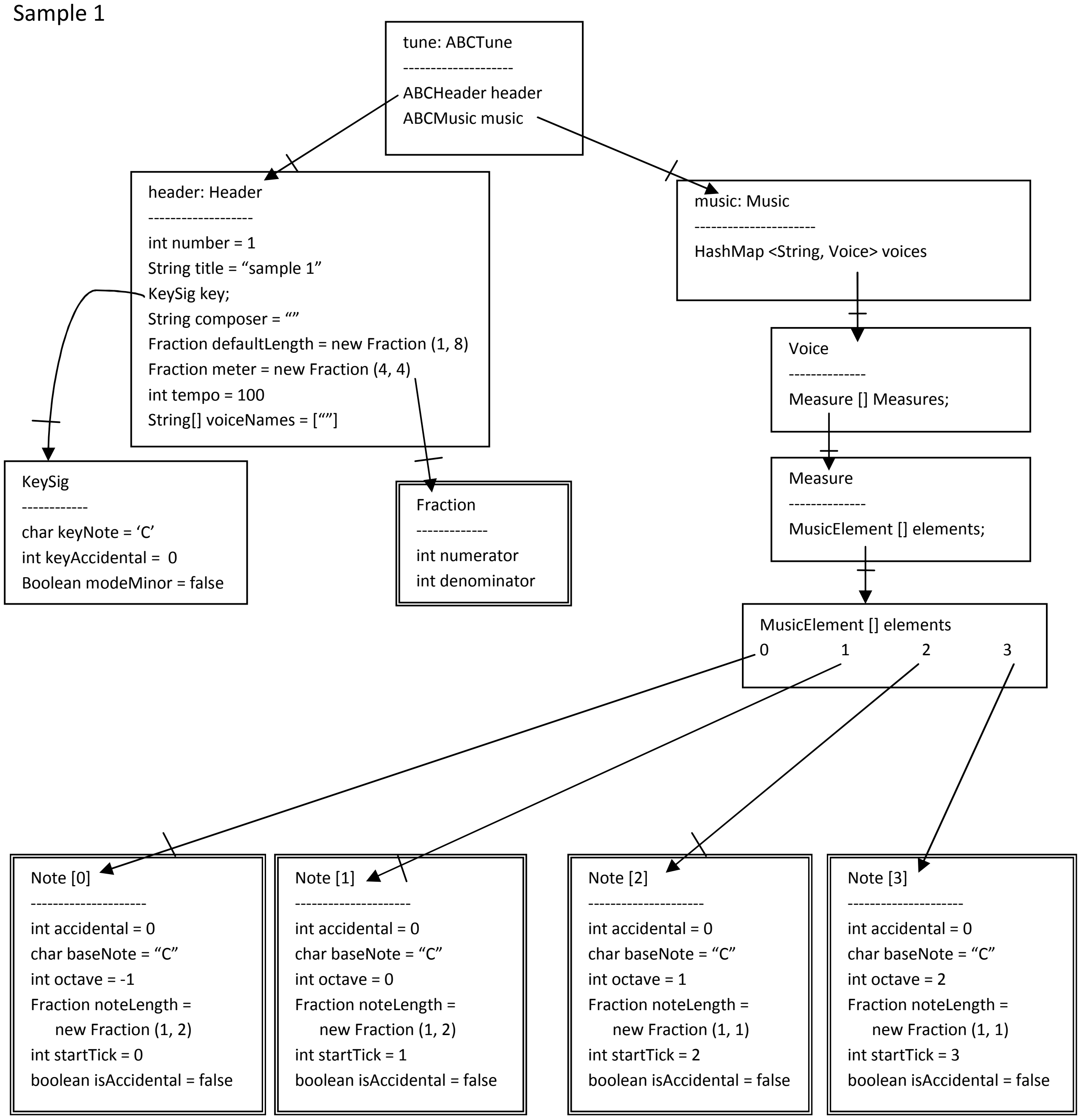
class Note implements MusicElement
class Rest implements MusicElement
class Tuplet implements MusicElement
class Chord implements MusicElement
Chords are made of notes which are stored in a list (as we do not know how many there will be). These will be played simultaneously.

interface Visitor()
This is the visitor interface which allows classes to be made to handle methods on the interpreter classes. The two classes that we currently have extending it are shown below.

```
public void visit (Voice voice);
public void visit (Pitch pitch);
public void visit (Rest rest);
public void visit (Tuplet tuplet);
public void visit(Chord chord);
public void visit (Measure measure);
```

class AddToSequencerVisitor implements Visitor
This class adds the element to the ABC sequencer for the ABCTune. It handles this for all MusicElements. It keeps track of current time in the piece, current key signature, and accidentals.

Sample 1


tune: ABCTune

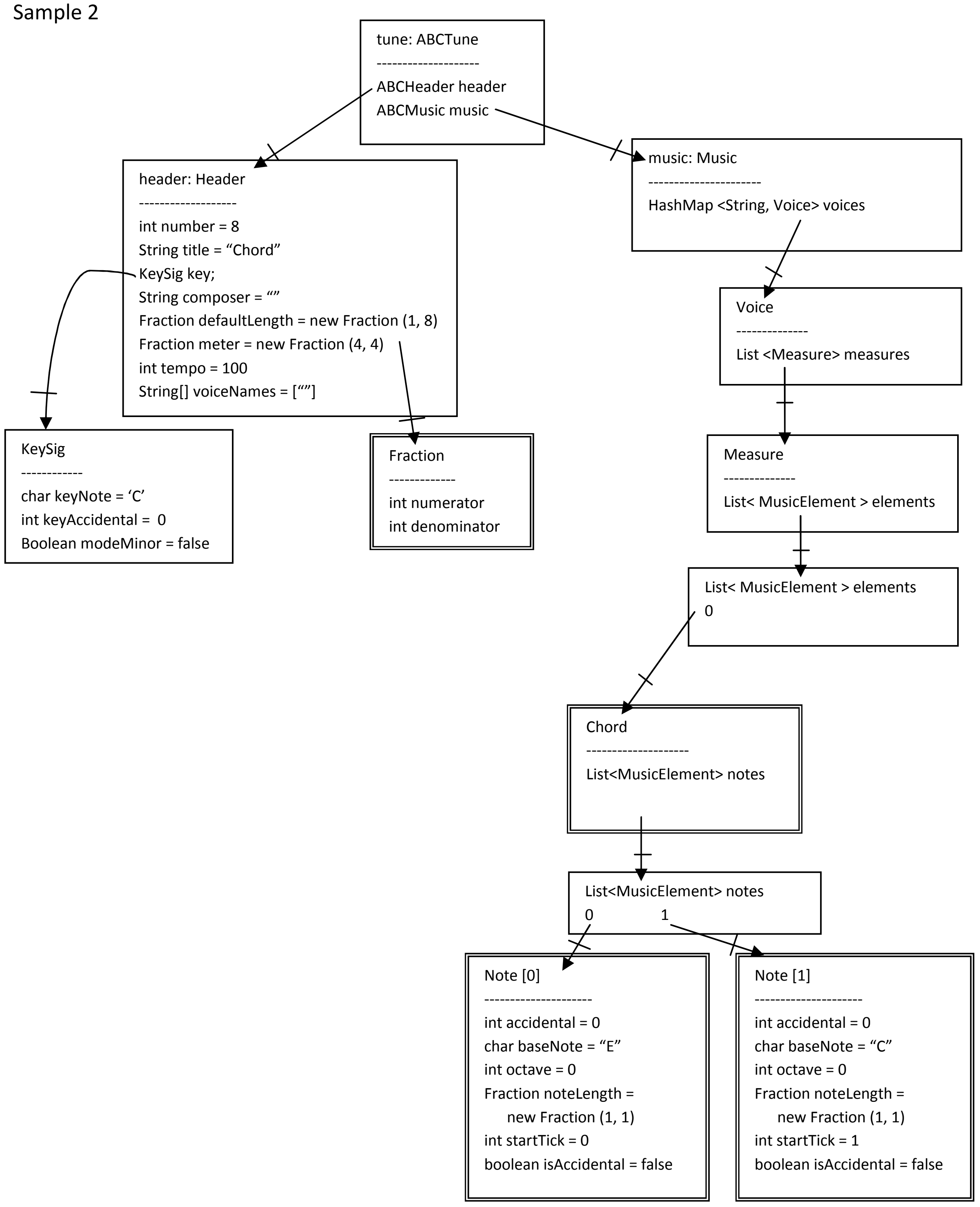
ABCHeader header
ABCMusic music
header: Header
----
int number = 1
String title = “sample 1”
KeySig key;
String composer = “”
Fraction defaultLength = new Fraction (1, 8)
Fraction meter = new Fraction (4, 4)
int tempo = 100
String[] voiceNames = [“”]
music: Music

HashMap <String, Voice> voices
Voice

Measure [] Measures;
Measure

MusicElement [] elements;
KeySig
----
char keyNote = ‘C’
int keyAccidental = 0
Boolean modeMinor = false
Fraction
----
int numerator
int denominator
MusicElement [] elements
0
1
2
3
Note [0]
----
int accidental = 0
char baseNote = “C”
int octave = -1
Fraction noteLength =
new Fraction (1, 2)
int startTick = 0
boolean isAccidental = false
Note [1]
----
int accidental = 0
char baseNote = “C”
int octave = 0
Fraction noteLength =
new Fraction (1, 2)
int startTick = 1
boolean isAccidental = false
Note [2]
----
int accidental = 0
char baseNote = “C”
int octave = 1
Fraction noteLength =
new Fraction (1, 1)
int startTick = 2
boolean isAccidental = false
Note [3]
----
int accidental = 0
char baseNote = “C”
int octave = 2
Fraction noteLength =
new Fraction (1, 1)
int startTick = 3
boolean isAccidental = false

Sample 2



tune: ABCTune

ABCHeader header
ABCMusic music
header: Header
----
int number = 8
String title = “Chord”
KeySig key;
String composer = “”
Fraction defaultLength = new Fraction (1, 8)
Fraction meter = new Fraction (4, 4)
int tempo = 100
String[] voiceNames = [“”]
music: Music

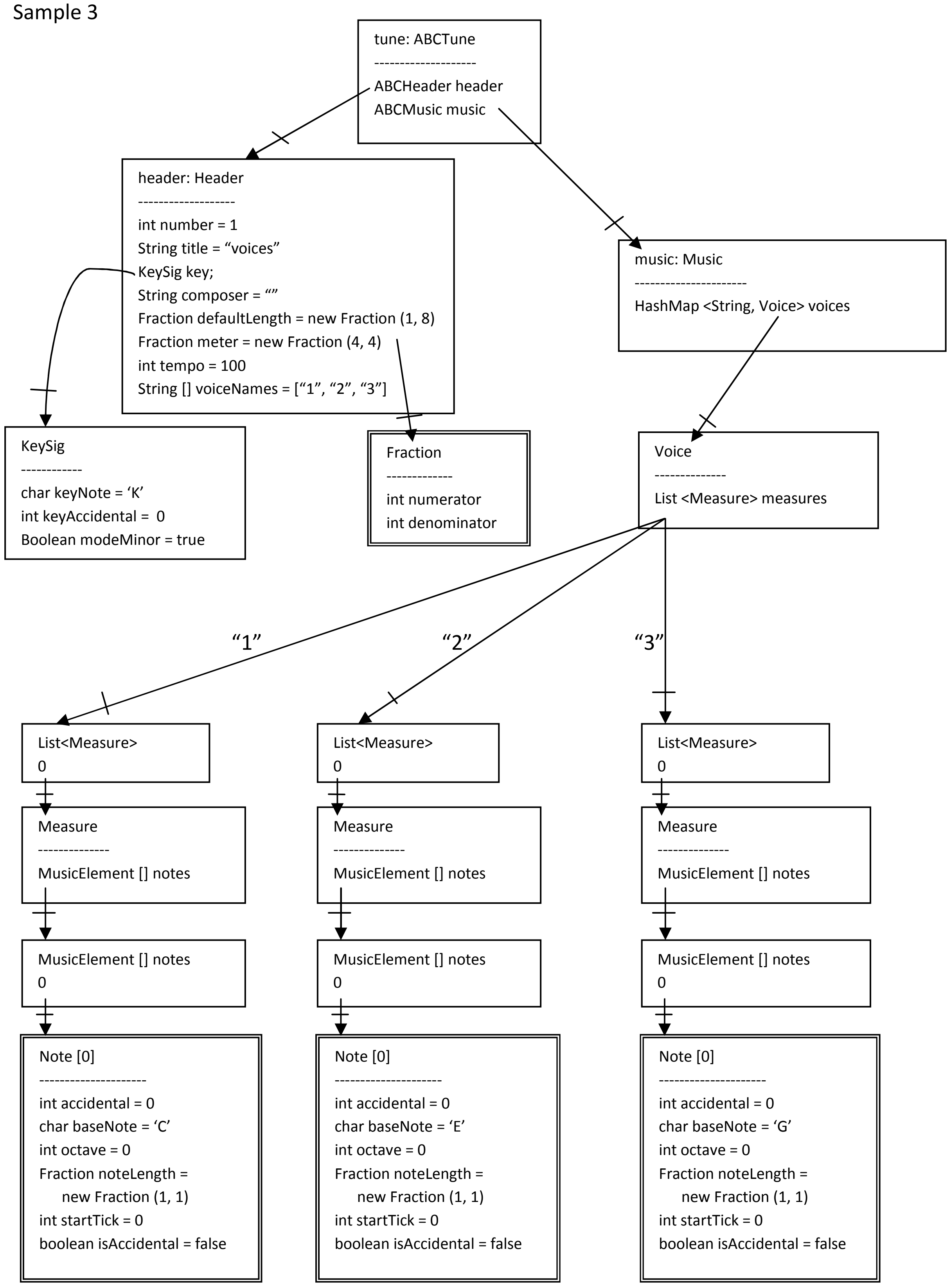
HashMap <String, Voice> voices
Voice

List <Measure> measures
KeySig
----
char keyNote = ‘C’
int keyAccidental = 0
Boolean modeMinor = false
Fraction
----
int numerator
int denominator
Measure

List< MusicElement > elements
List< MusicElement > elements
0
Chord

List<MusicElement> notes
List<MusicElement> notes
0 1
Note [0]
----
int accidental = 0
char baseNote = “E”
int octave = 0
Fraction noteLength =
new Fraction (1, 1)
int startTick = 0
boolean isAccidental = false
Note [1]
----
int accidental = 0
char baseNote = “C”
int octave = 0
Fraction noteLength =
new Fraction (1, 1)
int startTick = 1
boolean isAccidental = false

Sample 3


tune: ABCTune

ABCHeader header
ABCMusic music
header: Header
----
int number = 1
String title = “voices”
KeySig key;
String composer = “”
Fraction defaultLength = new Fraction (1, 8)
Fraction meter = new Fraction (4, 4)
int tempo = 100
String [] voiceNames = [“1”, “2”, “3”]
music: Music

HashMap <String, Voice> voices
KeySig
----
char keyNote = ‘K’
int keyAccidental = 0
Boolean modeMinor = true
Fraction
----
int numerator
int denominator
Voice

List <Measure> measures
“1”
“2”
“3”
List<Measure>
0
Measure

MusicElement [] notes
MusicElement [] notes
0
Note [0]
----
int accidental = 0
char baseNote = ‘C’
int octave = 0
Fraction noteLength =
new Fraction (1, 1)
int startTick = 0
boolean isAccidental = false
List<Measure>
0
Measure

MusicElement [] notes
MusicElement [] notes
0
Note [0]
----
int accidental = 0
char baseNote = ‘E’
int octave = 0
Fraction noteLength =
new Fraction (1, 1)
int startTick = 0
boolean isAccidental = false
List<Measure>
0
Measure

MusicElement [] notes
MusicElement [] notes
0
Note [0]
----
int accidental = 0
char baseNote = ‘G’
int octave = 0
Fraction noteLength =
new Fraction (1, 1)
int startTick = 0
boolean isAccidental = false