aybose, katielee, meenasub

6.005 REVISED DESIGN

Summary of Changes from Initial Design

In the Parser, we added some more helper methods because making a note, making a note with an accidental, and making a chord are all repetitive steps. The Parser calls these methods directly when it runs into the appropriate token or inside a tuplet. We chose not to handle a chord of tuplets. We also don't end the piece if a double_bar or end_bar token is found; the Parser only terminates on an error or once it has finished all of the tokens.

We also decided to use the interpreter pattern instead of the visitor pattern for evaluating our voice parts and notes.

Design

Sound Package

<class> HeaderReader

- reads in the .abc file and parses all relevant information from the header
- methods:
    - findKeySignatureMap() → instantiates and returns a Map of the key signatures given the key specified in the header
    - getTempo() → get tempo information
    - getComposer() → get composer information
    - getDefaultNoteLength()→ get default note length information
    - getMeter() → get meter information
    - getVoices() → returns a list of Strings representing each voice
        - We only check for voices in the header
        - If no voice is listed, returns "SingleVoice"

<class> KeySigMap() (mutable)

- the class that basically generates a map given a key signature
- has a HashMap (originalKey) (immutable) that never changes
    - flat = -1, sharp = 1, natural = 0
    - example: key of G would have the map {G: 0, A:0, B:0, C: 0, D:0, E: 0, F:1,
    - not case sensitive (because it should be able to handle different octaves)
- has a HashMap (accidentalKey) (mutable) that stores accidental nature for each note with a corresponding accidental in a bar
- methods
    - addAccidental() → Adds an accidental note to the accidentalKey, override if already in accidentalKey
    - clearAccidental() → Clears accidentalKey hashmap every time a measure bar

token is seen
- getNoteValue(note)
    - checks the accidental key and then the regular key signature to get the accidental value of the note.
    - for a given note, should return the integer value (-2, -1, 0, 1, or 2)

<class Token> (immutable)
- initializes different tokens based on their Types
    - BASENOTE-- any basic letter A-G (not case-sensitive)
    - REST-- z
    - TIME-- time of note, chord, or tuplet
    - ACCIDENTAL-- sharps, flats, natural (^, ^^, _, __, =)
    - OCTAVE-- octave shifts (string of commas and apostrophes)
    - MEASURE_BAR-- |
    - START_REPEAT_BAR-- |:
    - END_REPEAT_BAR-- :|
    - DOUBLE_BAR-- ||
    - END_BAR-- |]
    - START_BAR-- [|
    - NTHREPEAT1-- [1
    - NTHREPEAT2-- [2
    - TUPLET_SPEC-- (2, (3, or (4
    - OPEN_CHORD-- [
    - CLOSE_CHORD--]
    - VOICE-- voice token

<class> Lexer
- reads in the .abc file and creates a list of Tokens
- methods
    - getTicksPerDefaultNote() → returns an int for the number of ticks per default note length
    - getTokenList() → returns a List of Tokens
    - next() → returns the next token in the token list
    - prev() → returns the previous token in the token list
      lcm() → returns the least common multiple of two integers
    - toString() → string representation of all of the tokens together
        - useful for debugging maybe

<class> Parser
- takes in Lexer and HeaderReader as input
- methods:
    - parse() → reads through tokens, instantiating Elements with a calculated note length and adding them to a Voice object's List.
        - returns a List of Voice Objects, which are passed on to the evaluator
    - makeNoteWithAccidental() → called when parser runs into an accidental token, stores accidental information, then calls makeNote()
        - returns a note with an accidental value
    - makeNote() → called when parser runs into a basenote token, looks for octave and time tokens
        - returns a note
    - makeChord() → called when parser runs into a open_chord token, looks for

notes
        - returns a multinote

Abstract Syntax Tree Package

<interface> Element (immutable)

- implemented by the classes Note, Rest, Tuplet, MultiNote
- common methods:
    - getNumTicks() → returns number of ticks that the note, tuplet, or rest is held for
    - eval() → uses the interpreter pattern to return a MIDI note or a set of MIDI notes that the player can play
    - toString() → toString method
- <class> Note:
    - attributes: length, pitch, octave, accidental (these are set when Note is instantiated)
    - methods:
        - evalNote()→ creates a pitch based on the attributes specified in the Note class
- <class> Rest:
    - attributes: length
- <class> Tuplet
    - attributes: length of tuplet in total, each note in the tuplet
    - instantiates however many Elements (Note, Rest, or MultiNote) are in the tuplet
        - ex: TupleElement(triplet) would be instantiate 3 Elements
- <class> MultiNote
    - attributes: length of the chord, each note in the chord
    - instantiates Notes of the same length (however many notes are in the chord)

<class> Voice (mutable)

- handles the Elements associated with each part by adding them to a List
- only check for voices listed in the ParseHeader
- attributes:
    - int open: when we have an open repeat symbol
    - int nthRepeatOne: when we have an alternate ending
    - int globalTick: increments the global tick as each element with a certain numTicks is added
- addElement(Element e) → takes in an Element and adds it to the List of Elements, updates globalTick
- getOpenRepeat() → returns the index of an open repeat in the voice part
- setOpenRepeat() → sets the index of an open repeat in the voice part
- getNthRepeatOne() → returns the index of a 1] token
- setNthRepeatOne() → sets the index of a 1] token
- alternateEnding2() → is called when adds the appropriate repeated segment to the voice list (from open repeat to getNthRepeatOne() index, up until the alternateEnding2 index
- repeat() → repeats from the open repeat to closed repeat and adds tokens to voice list
- getElements() → returns the list of Elements for the voice part
- getGlobalTicks() → get the total number of ticks in the voice part
- getFullVoiceString() → returns a String representation of all of the Elements in avoice part

    - toString() → returns the name of the voice part
    - eval(int totalTicks, SequencePlayer player) → evaluates each Element in voiceList (using Element.eval(totalTicks, player)), updating totalTicks as notes are added to the player.

Test Classes

We will test the following classes:

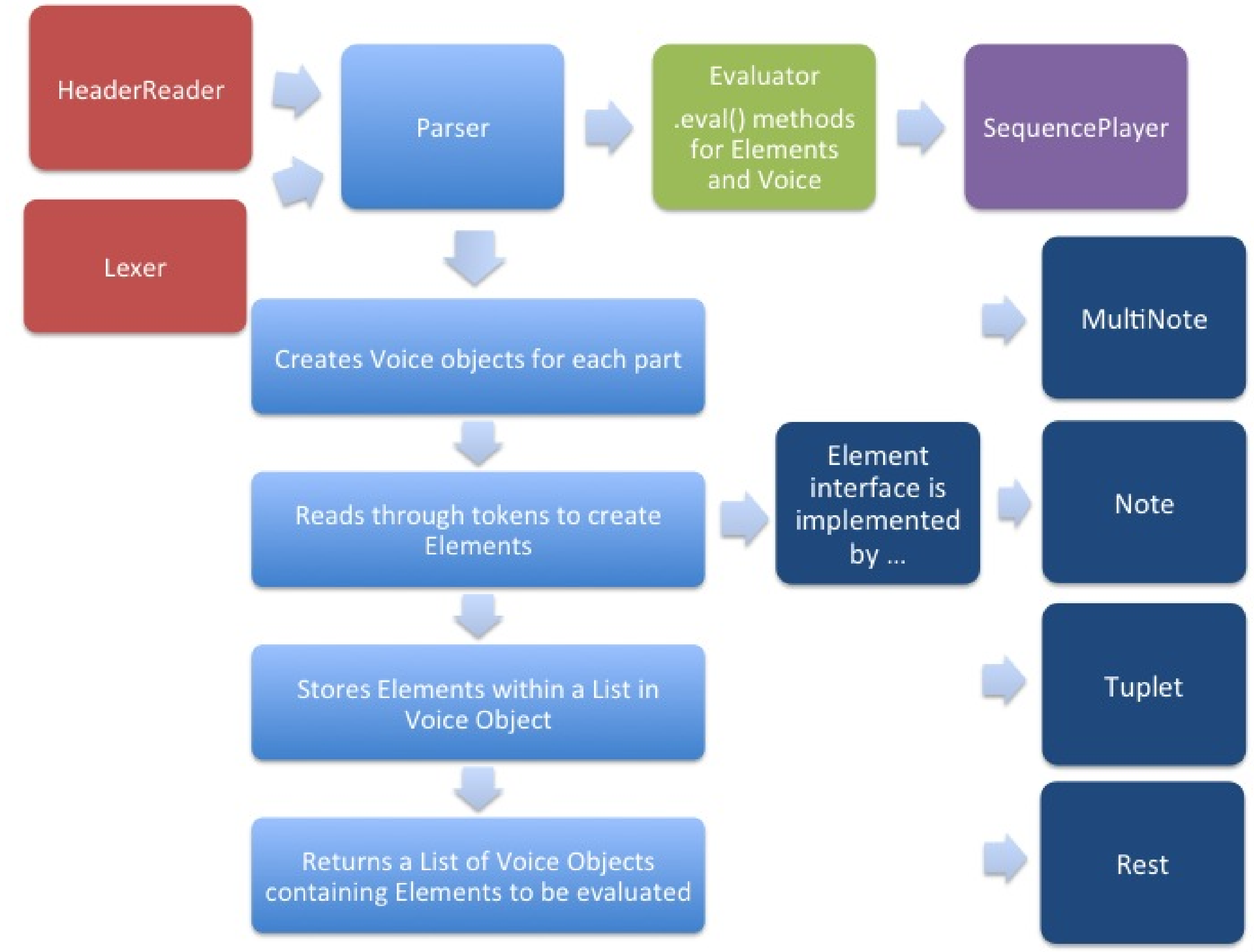
- Token:
    - Each token's attributes
        - type
        - text
- HeaderReader:
    - Structure Checks:
        - Basic X:, T:, K:
        - X:, T:, K:, with other methods, defined
    - Method Checks:
        - Constructor
        - getTempo()
        - getComposer()
        - getMeter()
        - getDefaultNoteLength()
            - With different tempo listings
            - With tuplets
            - With chords
        - getVoices()
            - Single voice
            - Multiple voice
            - Unlisted voice
- KeySigMap:
    - Adding to accidentalKey
    - Getting from accidentalKey
    - Getting from originalKey
    - Octave handling
    - Clearing accidentalKey
    - Invalid key signature handling
- Lexer:
    - File Checks
        - Whitespace handling
        - Linefeeds handling
        - Incorrect token handling
        - Comment handling
    - Method Checks
        - Lexer Constructor
        - getTicksPerDefaultNote() method
        - getTokenList() method
    - Helper Method Checks
        - fractionChange
            - numerator only (single digit, multiple digit)

        - denominator only (single digit, multiple digit)
        - no numerator or denominator
    - LCM
        - larger first number
        - larger second number
        - zero case
- Parser:
    - Valid one-voice piece (ex. scale.abc)
    - Valid two-voice piece (ex. fur_elise.abc)
    - Valid multiple-voice piece (ex. prelude.abc)
    - Basic note (ex. scale.abc)
    - Basic rest (ex. prelude.abc)
    - Note with accidental, octave, and time (ex. little_night_music.abc)
    - Note with accidental and octave (ex. little_night_music.abc)
    - Note with accidental and time (ex. little_night_music.abc)
    - Note with octave and time (ex. little_night_music.abc)
    - Basic chord (ex. little_night_music.abc)
    - 2 Tuplet (ex. valid_onevoice.abc)
    - 3 Tuplet (ex. valid_onevoice.abc)
    - 4 Tuplet (ex. valid_onevoice.abc)
    - Tuplet with cut time (ex. valid_onevoice.abc)
    - Whole piece repeat (ex. valid_onevoice.abc)
    - Start bar piece repeat (ex. fur_elise.abc)
    - Alternate ending test (ex. paddy.abc)
    - Up 1 octave (uppercase, lowercase) (ex. octave_up1_lower.abc, octave_up1_upper.abc)
    - Up 4 octave(uppercase, lowercase) (ex. octave_up4_lower.abc, octave_up4_upper.abc)
    - Down 1 octave(uppercase, lowercase) (ex. octave_down1_lower.abc, octave_down1_upper.abc)
    - Down 4 octave (uppercase, lowercase) (ex. octave_down4_lower.abc, octave_down4_upper.abc)
    - Accidental in default key (ex. valid_onevoice.abc)
    - Accidental in different key (ex. valid_onevoice.abc)
    - Accidental reset per different line (ex. valid_onevoice.abc)
    - Accidental handling for notes in different octaves (ex. valid_onevoice.abc)
    - All different bars (seen across many examples)
    - Different Tempos (seen across many examples)

  Invalid Tests
    - rest+accidental (ex. invalid_restaccidental.abc)
    - rest+octave (ex. invalid_restoctave.abc)
    - Unequal repeats (ex. unequalrepeats.abc)
- VoiceTest
    - tests whether elements are added correctly to each voice part
    - tests whether repeats, nth repeats are handled correctly
- Evaluator:
    - Basically creates Elements and then tries to evaluate them
    - tests individual .eval() methods for Note, Rest, MultiNote, Tuplet, and Voice
- Integrated Test
    - uses Main.play() to play the pieces and test them

- no_didit

Process Flow Diagram


HeaderReader
Lexer
Parser
Evaluator
.eval() methods for Elements and Voice
SequencePlayer
Creates Voice objects for each part
Reads through tokens to create Elements
Stores Elements within a List in Voice Object
Returns a List of Voice Objects containing Elements to be evaluated
Element interface is implemented by ...
MultiNote
Note
Tuplet
Rest

Modified Grammar

We changed :

element ::= note-element | tuplet-element | barline | nth-repeat | space
note-element ::= (note | multi-note)

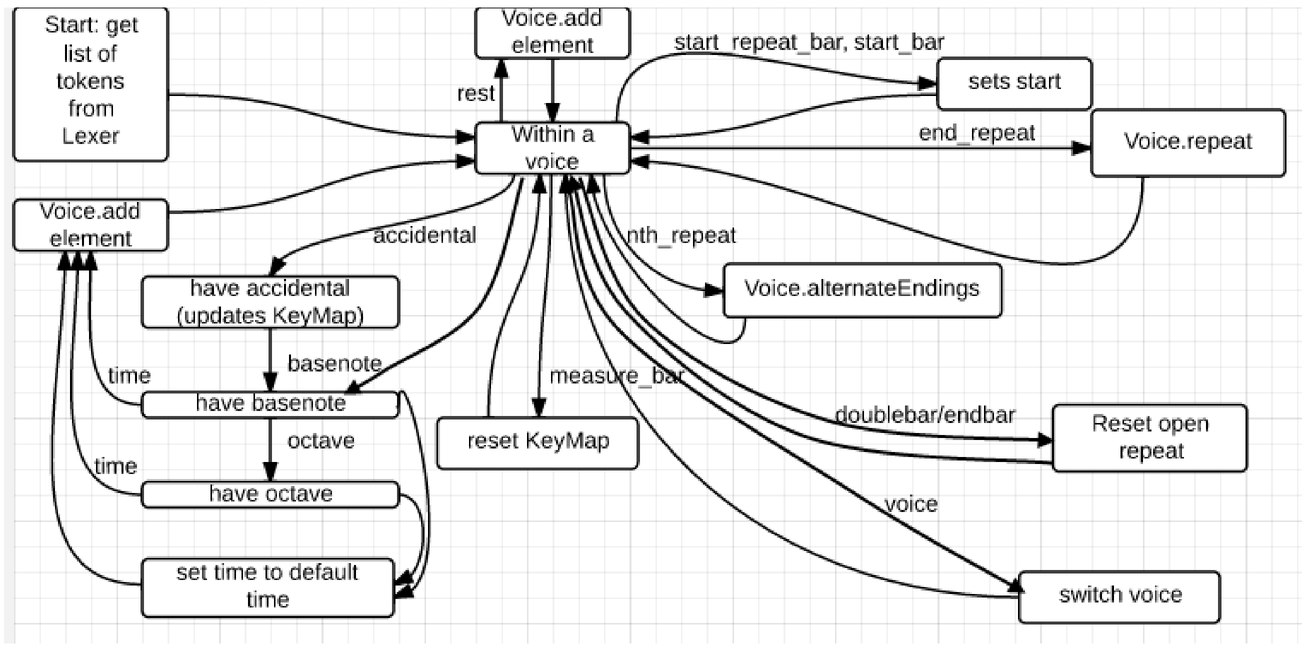
to:

element ::= note | multi-note | tuplet-element | barline | nth-repeat | space

Parser State-Machine Diagram

With tuples and chords, the left side of the diagram will loop for x number of times for a tuple or chord. We omitted this from the diagram to keep it simple. Any unexpected tokens (in other words, no arrow from that state exists with that token) will throw an error.

This diagram is a large simplification of the Parser.


Start: get list of tokens from Lexer
Voice.add element
rest
Within a voice
start_repeat_bar, start_bar
sets start
end_repeat
Voice.repeat
Voice.add element
accidental
nth_repeat
Voice.alternateEndings
have accidental (updates KeyMap)
basenote
time
have basenote
measure_bar
reset KeyMap
doublebar/endbar
Reset open repeat
octave
time
have octave
voice
set time to default time
switch voice