Design for ABC Player

ekl, hpang, morganti

1. List of Datatypes
   All of our datatypes will be immutable.

- Header: This will be a class for handling the data in the "header" of a music piece, which contains info such as composer, tempo, etc.
    - Methods:
        - String getComposer(): returns the composer name as a string
        - Will have similar methods for each of the parameters
- MusicElement (interface): This is an interface for individual "music elements" which are part of a measure -- i.e. notes, chords, and rests. Methods required by this interface:
        - isRest(): returns a boolean saying whether the object is a rest (Aids in assigning lyrics to notes)
        - addToPlayer(Player player, int currentCount, int ticksPerBeat): Adds the note/rest to the sequence of MIDI events in the player at the correct time. It is necessary to pass the parameter ticksPerBeat since ticksPerBeat cannot be determined from the player object.
    - Classes implementing MusicElement:
        - Note: Class representing a single note. Has attributes for the "base" pitch (represented by a single capital letter), an accidental (0 if none, positive for sharp, negative for flat), the octave (0 if in the same octave as middle C, positive if higher, negative if lower), and the duration in fractions of a beat, represented as a Duration object
            - Constructor method parameters: char baseNote, int accidental, int octave, Duration duration
        - Rest: Class for a rest. Has attribute for duration in fractions of a beat, represented as a Duration object
            - Constructor method parameters: Duration duration
        - Chord: Class for a chord. Has attribute for the notes (represented as a list of Note objects). (Note: Our Chord class currently also contains a Duration parameter, but we've recently decided that's unnecessary and will be removing it.)
            - Constructor method: parameters: list of Notes
- Duration: Class for duration of a note, rest, or chord -- essentially represents a fraction. Has attributes numerator and denominator (Capable of determining correct numerator and/or denominator even if they are "missing" from the abc notation, e.g. A/ , A/3, A). We typically store duration in fractions of a beat, which will be determined by looking at meter and default note length in the header.
    - Constructor method: There are two different constructors.
        - Duration(String): Takes a string representing the duration as it appears in the ABC file -- the part after the note letter and accidentals/octaves (e.g. /, /4, 1/4) and determines correct numerator and denominator
        - Duration(int numerator, int denominator): Constructs a duration object by directly

passing values for numerator and denominator.

- Measure: Class for measures (groups of MusicElements)
    - boolean beginRepeat()
        - Returns true if a repeat section begins at this measure; false otherwise
    - endRepeat()
        - Returns true if a repeat section ends at this measure; false otherwise
    - addToPlayer(SequencePlayer player, int currentCount, int ticksPerBeat)
        - Adds measures to the sequence player as MIDI events
        - Iterates through the elements of the measure and calls the addToPlayer method of each note/chord/rest
- Voice: Class for grouping measures together into a single voice
    - Constructor: takes in list of Measures
    - We haven't directly needed this datatype for our demo (we were able to separate the voices by grouping the notes into measures, and feeding the measures for each voice into the player), but we anticipate needing it for when we are parsing the abc files.
    - Since we haven't worked as much on our lexer and parser, we might be adding more methods to this class if they are needed.
- PieceOfMusic: Class representing a single piece of music (i.e. the contents of one ABC file)
    - Contains a Header, a list of Voice objects, and a list of Lyric objects
- Lyric: Class for a single lyric syllable
    - Constructor
        - parameters: String syllable, int duration
- LyricsIterator: Class used for properly adding Lyric elements to the player
    - Constructor
        - parameter: List of Lyrics
    - addNextLyric(Player player, int currentCount)
        - Adds the next lyric syllable to the player at the current tick

2. Grammar
    a. We plan to base our grammar off the one provided. There are no significant changes, only changes to add both comprehensive but also logical and readable parser and lexer rules. The current ABCMusic.g4 file is the current grammar design that we are working with. It follows closely the given grammar guideline.

3. Strategy for ANTLR
    a. The lexer creates all the tokens for the grammar. We are following the basic guidelines for the grammar given. For instance, for the lexer, some terminals are: accidentals, basenote, octave, rest, barline, text, meter, whitespace, and more. These were all decided on based on the grammar given and also general knowledge of parsing music into elements.
    b. The parser follows the grammar and closely uses the lexer to be efficient and

comprehensive. For instance, the parser examines 'tune : header piece EOF' which is the topmost level of the AST. Beneath that, the tree steps through 'piece : music lyrics?' 'header: number comment? title others? field_key', which are at the same level. The parser continues on in this way to parse through the tune, mostly following the guidelines given.

c. Errors: We will handle errors by throwing an exception with an appropriate message for the user stating what is wrong with the input.

4. Transformation from AST to ADT to playable format
    a. In the Listener methods we will call the constructors for our ADT elements, which correspond closely to the lexer tokens we are using
    b. Once we've constructed the ADT, we will use the method addToPlayer() for the measures in our ADT, which will transform the instances of MusicElement into a form that can be played using SequencePlayer

5. Testing
    a. Abstract Data Type: We will test the methods within each class of our abstract data type
        i. MusicInterface classes (Note, Chord, Rest)
            1. isRest: test on instances of Rest, Chord, Note
            2. addToPlayer:
                a. Test this on single instances of Rest, Chord, Note.
                b. Additionally, we'll test this on chords/notes (including tuplets) with different durations and octaves in order to make sure it sounds right
                    i. examples (these are in abc notation for simplicity -- the ADTs will actually be what is passed into addToPlayer):
                        1. c2 E,3 c''3/2
                        2. F/2 z z
                        3. (3ABC a b'2
            3. toString:
                a. Test on single instances of Rest, Chord, Note to make sure that toString correctly represents them as a string.
                b. Additionally, we'll test different combinations of chords/notes/rests with different durations and octaves.
                    i. examples (these are in abc notation for simplicity -- the ADTs will actually be what is passed into toString):
                        1. b3 A,4 a''5/2
                        2. D/2 z/4
                        3. (3aBc' b c'2
        ii. Measure
            1. addToPlayer: Test on measures composed of:

a. notes
b. chords
c. mixture of rests, notes, and chords

2. isBeginRepeat and isEndRepeat: test on multiple pieces which have repeats
    a. piece with a single, simple repeat
    b. piece that has multiple repeated sections
    c. piece with nested repeats (e.g., a piece with a chorus with repeated lines)

iii. Voice

1. If we add more methods to Voice, we will probably need to test them.

b. Lexer

i. We will test our lexer in order to make sure it can correctly recognize all of our tokens.

ii. We will have tests for each token.

iii. We will have separate tests for tokens that have multiple variations in the abc notation (e.g. duration)

c. Parser/Listener

i. We will test our parser by parsing short pieces of music into an ADT, then making sure the string representation of those ADTs is correct. Test cases will include

1. a piece consisting of only notes
2. a piece consisting of notes+rests
3. a piece which contains chords
4. A piece with tuplets
5. A piece with lyrics
6. A piece with multiple voices
7. A piece with repeats

d. End-to-end testing

i. We will have tests that make sure the entire program performs correctly. We will use several different pieces of music (with varying degrees of complexity) to test the piece. Types of pieces we will test:

1. A piece with no lyrics
2. A piece with lyrics
3. A piece with repeats
4. A piece with chords
5. A piece with tuplets
6. A piece with multiple voices