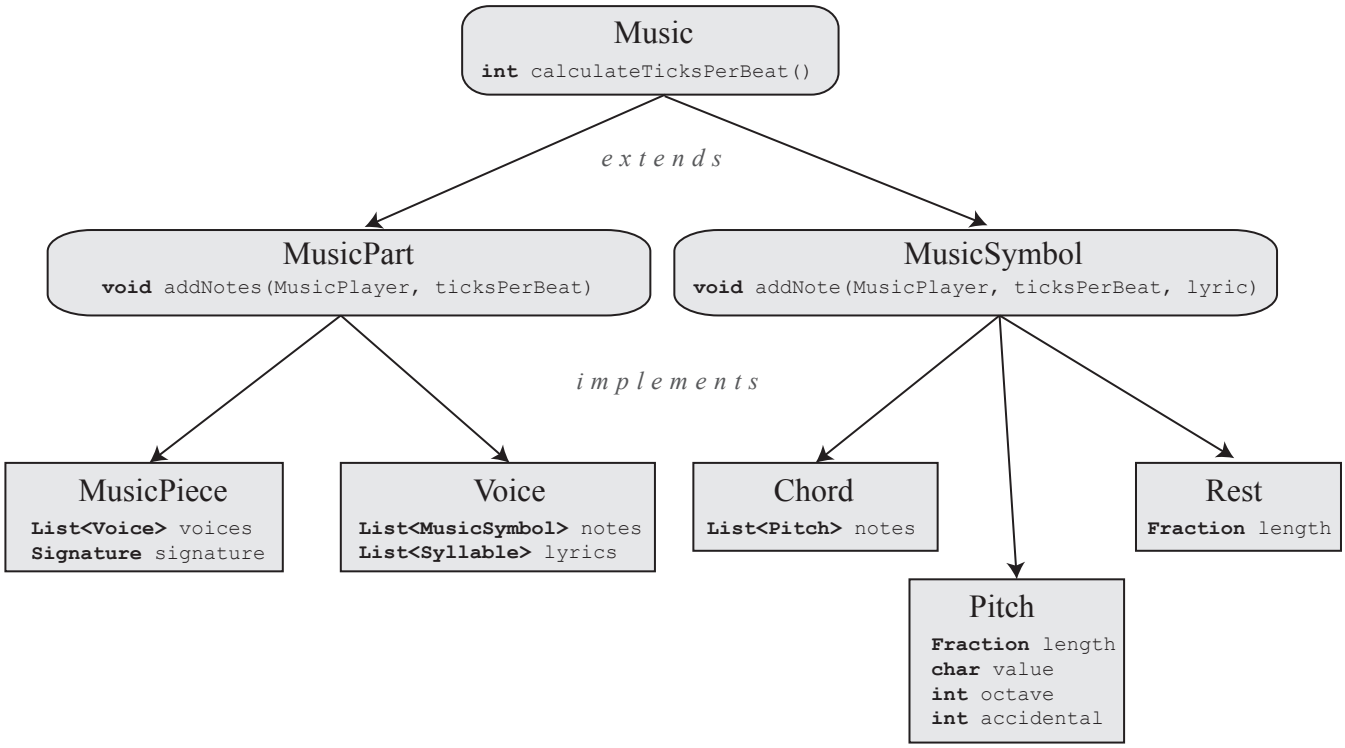
Design Description

We represent ???add some bullshit???



`Music` is an interface that will represent our ADT. It defines the following method

- `calculateTicksPerBeat` is an auxiliar method needed to determine the number of ticks per beat for the player such that any notelength can be represented as an integer number of ticks. The method is called recursively.

`MusicPart` is an interface that extends `Music` and represents either a `MusicPiece` or `Voice`. It defines a method

- `addNotes` which takes an instance of `MusicPlayer` (described later) and adds to it all notes and lyrics found in the current `MusicPart`

and is implemented by the following classes:

- `MusicPiece` represents a final piece of music. It has a `signature` attribute which contains all the header information and `List<Voice> voices` attribute which represents a list of different voices contained in this piece.

- `Voice` represents a single voice in a piece of music. It has a `List<MusicSymbol>notes` attribute which represents the sequence of notes the voice is made of and a `List<Syllable> lyrics` attribute that represents the lyrics that accompain the voice.

`MusicSymbol` is another interface that extends `Music` and represents either a `Pitch`, a `Rest` or a `Chord`. It defines a method

- `addNote` which takes an instance of `MusicPlayer` and a lyric corresponding to the current symbol and modifies the player by adding notes and lyric to it. The parameter `ticksPerBeat` is used to convert the length of the note from a fraction to the number of ticks.

and is implemented by classes:

- `Pitch` represents a single pitch. Its attribute `length` is represented by a fraction of the length of the default note. The attribute `value` is a pitch A,B,C,...,G from the middle octave, `octave` represents the offset from the middle octave and `accidental` is 1 for sharp and -2 for flat. Using these conviniences, a `Pitch` from out ADT can be easily converted to the `Pitch` object described in the `sound` package.
- `Rest` represents a single rest and has an attribute `length` represented by a fraction of the length of the default note.
- `Chord` represents a chord of several pitches and stores them in the `List<Pitch>` attribute.

These three basic music symbols let us implement any "musical expression" defined in the 6.005 subset. Any other structures like tuplets, triplets, repeats, etc. are converted to these basic music symbols during the ParseTree walk.

Another important class is a mutable `MusicPlayer` which has two attributes

- `player` that represents an instance of the `SequencePlayer`. It collects the notes and lyrics of the song.
- `currentTick` which represents the current tick inside the player. It is used when the notes and lyrics are added consecutively to the player by the method `addNote` to keep track of the position where the notes need to be inserted. `addNote` method increments it according to the length of the note.

It has the following methods:

- `addNote (int note, int numTicks)` that takes a converted Midi note and inserts it in the `player` at the `currentTick` position for a length of `numTicks`.
- `addLyric (String Lyric)` that takes a syllable and adds it to the `LyricListener` at the `currentTicks` position.
- `addTime (int numTicks)` increases the `currentTick` by `numTicks`.
- `resetTime()` resets the `currentTicks` to 0. This is used when starting a new voice.
- `play()` plays the notes and lyrics added.

Parsing

- **Lexer**: The lexer is designed such that all of the header lines (ie. title, composer, etc) are each lexed as one token, as are all lyric and comment llines. Notes are lexed together with their modifiers (accidentals, octaves, duration) as one token. Tuplet and chord brackets/parenthesis/digits are lexed separately from their notes. Pipes, repeats, and end of measure symbols are lexed as their own tokens. All whitespace except for newlines (/n/r) is skipped.

- **Parser**: The parser has rules for the whole musical piece, which is then broken into the header rules and the actual musical body rules. Each header line has its own rule, with title, number and key being mandatory. The musical body consists of either measures optionally followed by a lyric line or a voice or a comment. Measures are note elements optionally surrounded by repeats or pipes, but must end with either a repeat, an end of measure symbol, or a newline.

  A note element is a note, rest, chord, or tuplet. Tuplets are a parenthesis followed by a digit and a number of notes or chords. A chord is a number of notes enclosed in square brackets. Note, rest, and lyric have their own rules as well, which are just their respective lexer tokens.

- **Errors**: Errors in parsing and lexing result in an exception being thrown. (reportErrorsAsExceptions is invoked on both parser and lexer).

- **Listener**: While parsing the tree, we only care about these events: exit note, exit rest, exit chord, exit tuplet, exit lyric, enter voice, exit header, exit measure, exit music.

  When exiting a header, the information is extracted from the context, and defaults are added if necessary. A default voice is set if none are provided. The current voice is kept track of.

  When entering a voice, the current voice is switched to one with the same name.

  When exiting a note or rest, we extract the needed information from the context and add a Pitch or Rest object to the current voice. When exiting a tuplet, the number of notes inside is determined, then they are popped, and their duration is modified accordingly, and the new notes are inserted back into the current voice.

  When exiting a chord, the number of notes inside is determined, then they are popped, and then added to a Chord object, which is then added to the current voice (this object represents a list of notes, since they all start at the same time).

  When exiting a lyric, the chunk of text will be broken up into syllables (discarding the w:) and then further treated according to whether or not these new syllables have special characters. Then a list of strings representing the syllables will be used to create a Lyric object, which is added to the current voice. Ensuring that there are the right number of syllables will be done at a later time.

  When exiting a measure, repeats are detected (if any), and then the notes inside are doubled.

When exiting music, voices and signature will be added to a MusicPiece object, which is then added to the stack.