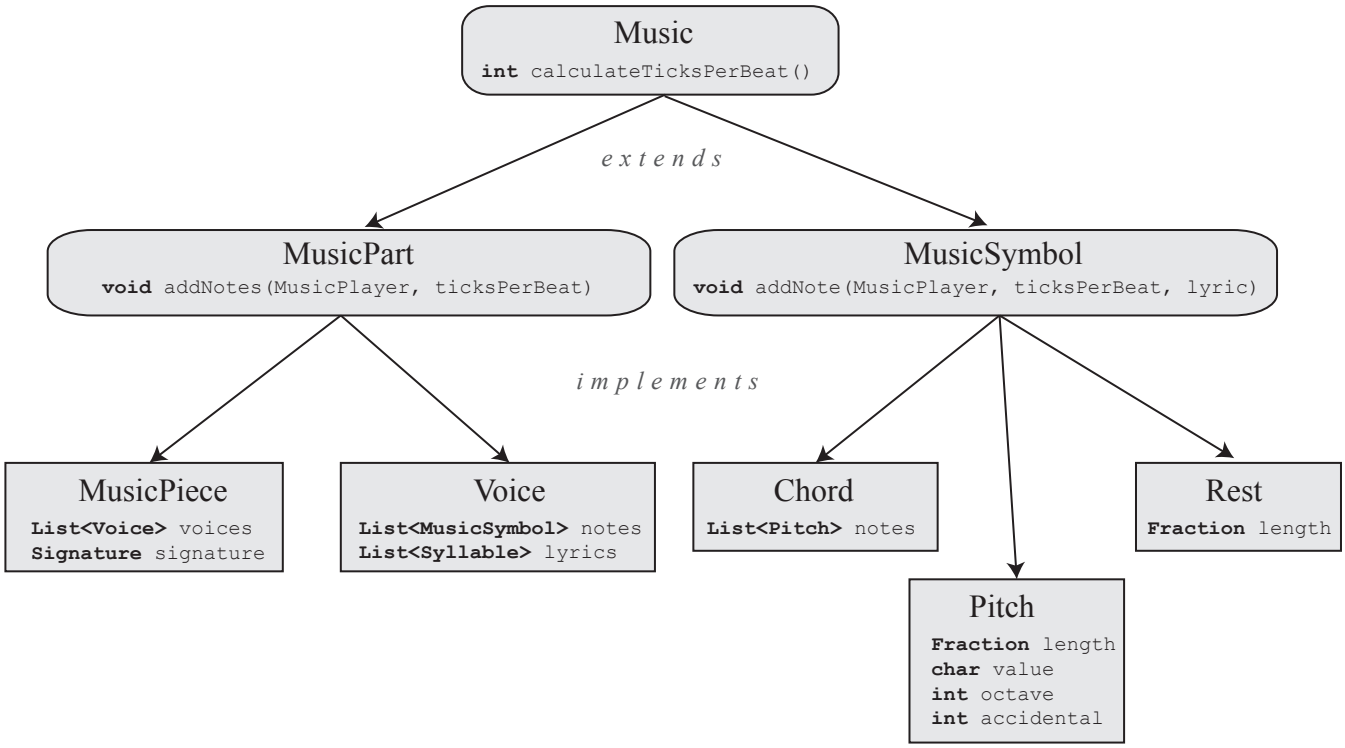
Design Description

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.